Mirai DDoS Botnet: Source Code & Binary Analysis

simonroses.com/2016/10/mirai-ddos-botnet-source-code-binary-analysis/

By Simon Roses October 27, 2016

Mirai is a DDoS botnet that has gained a lot of media attraction lately due to high impact attacks such as on journalist Brian Krebs and also for one of the biggest DDoS attacks on Internet against ISP Dyn, cutting off a major chunk of Internet, that took place last weekend (Friday 21 October 2016).

Besides the media coverage, Mirai is very interesting because we have both binary samples captured in the wild, but also because the source code was released recently – for sure we can expect many variants of Mirai code soon. Having both binary and source code allows us to study it in more detail.

It is quite amazing that we are in 2016 and still talking about worms, default/weak passwords and DDoS attacks: hello Morris Worm (1988) and Project Rivolta (2000) to mention a few.

Source Code Analysis

We have compiled Mirai source code using our Tintorera, a VULNEX static analysis tool that generates intelligence while building C/C++ source code. This gives us the big picture fast.

From Tintorera we get an application detail summary counting compiled files, lines of code, comments, blanks and additional metrics; Tintorera also calculates the time needed to review the code. Mirai is a small project and not too complicated to review. (Figure 1)

Figure 1

Mirai is using several functions from the Linux API, mostly related to network operations. (Figure 2)

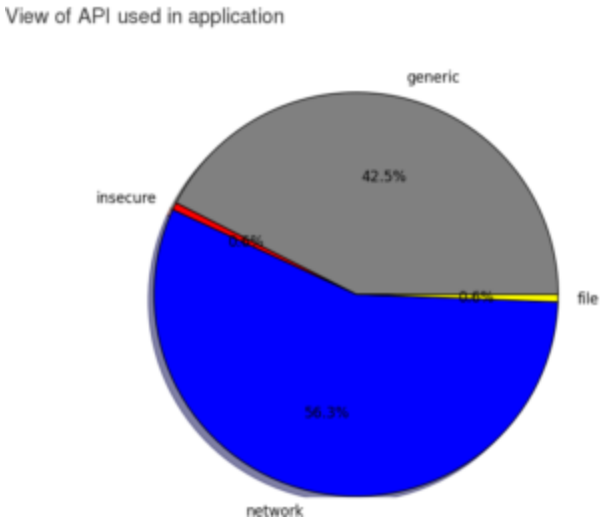

Figure 2

In the Tintorera intelligence report we have a list of files, functions names, basic blocks, cyclomatic complexity, API calls and inline assembly used by Mirai. By examining this list we can get an idea of the code. (Figure 3)

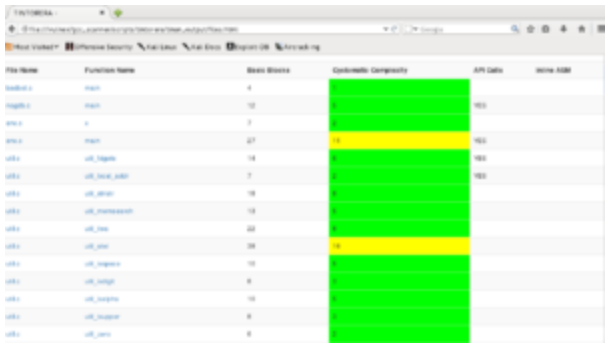
Figure 3

In file killer.c there is a function named killer_init that kills several services: telnet (port 23), ssh (port 22) and http (port 80) to prevent access to the compromised system by others. (Figure 4)

Figure 4

In same file, killer.c, another function named memory_scan_match search memory for other Linux malwares. (Figure 5)

Function Code

```
static BOOL memory_scan_match(char *path)
{
    int fd, ret;
    char rdbuf[4096];
    char *m_qbot_report, *m_qbot_http, *m_qbot_dup, *m_upx_str, *m_zollard;
    int m_qbot_len, m_qbot2_len, m_qbot3_len, m_upx_len, m_zollard_len;
    BOOL found = FALSE;

    if ((fd = open(path, O_RDONLY)) == -1)
        return FALSE;

    table_unlock_val(TABLE_MEM_QBOT);
    table_unlock_val(TABLE_MEM_QBOT2);
    table_unlock_val(TABLE_MEM_QBOT3);
    table_unlock_val(TABLE_MEM_UPX);
    table_unlock_val(TABLE_MEM_ZOLLARD);

    m_qbot_report = table_retrieve_val(TABLE_MEM_QBOT, &m_qbot_len);
    m_qbot_http = table_retrieve_val(TABLE_MEM_QBOT2, &m_qbot2_len);
    m_qbot_dup = table_retrieve_val(TABLE_MEM_QBOT3, &m_qbot3_len);
    m_upx_str = table_retrieve_val(TABLE_MEM_UPX, &m_upx_len);
    m_zollard = table_retrieve_val(TABLE_MEM_ZOLLARD, &m_zollard_len);

    while ((ret = read(fd, rdbuf, sizeof (rdbuf))) > 0)
```

Figure 5

In file scanner.c function named get_random_ip generates random IPs to attack while avoiding a white list addresses from General Electric, Hewlett-Packard, US Postal Service and US Department of Defense. (Figure 6)

Function Code

```
static ipv4_t get_random_ip(void)
{
    uint32_t tmp;
    uint8_t o1, o2, o3, o4;

    do
    {
        tmp = rand_next();

        o1 = tmp & 0xff;
        o2 = (tmp >> 8) & 0xff;
        o3 = (tmp >> 16) & 0xff;
        o4 = (tmp >> 24) & 0xff;
    }
    while (o1 == 127 ||                             // 127.0.0.0/8      - Loopback
           (o1 == 0) ||                             // 0.0.0.0/8        - Invalid address space
           (o1 == 3) ||                             // 3.0.0.0/8        - General Electric Company
           (o1 == 15 || o1 == 16) ||                // 15.0.0.0/7       - Hewlett-Packard Company
           (o1 == 56) ||                            // 56.0.0.0/8       - US Postal Service
           (o1 == 10) ||                            // 10.0.0.0/8       - Internal network
           (o1 == 192 && o2 == 168) ||              // 192.168.0.0/16   - Internal network
           (o1 == 172 && o2 >= 16 && o2 < 32) ||    // 172.16.0.0/14    - Internal network
           (o1 == 100 && o2 >= 64 && o2 < 127) ||   // 100.64.0.0/10    - IANA NAT reserved
           (o1 == 169 && o2 > 254) ||               // 169.254.0.0/16   - IANA NAT reserved
           (o1 == 198 && o2 >= 18 && o2 < 20) ||    // 198.18.0.0/15    - IANA Special use
           (o1 >= 224) ||                           // 224.*.*.*+       - Multicast
           (o1 == 6 || o1 == 7 || o1 == 11 || o1 == 21 || o1 == 22 || o1 == 26 || o1 == 28 || o1 == 29
    );
```

Figure 6

Mirai comes with a list of 62 default/weak passwords to perform brute force attacks on IoT devices. This list is setup in function scanner_init of file scanner.c. (Figure 7)

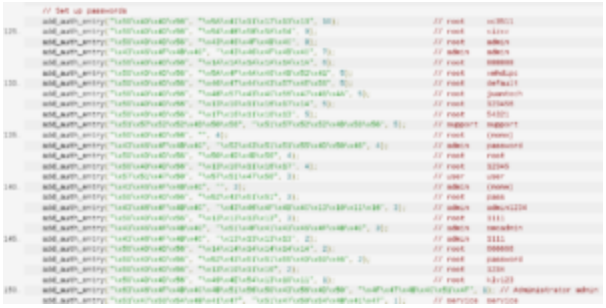

Figure 7

In main.c file we can find the main function that prevents compromised devices to reboot by killing watchdog and starts the scanner to attack other IoT devices. In Figure 8 we see a callgraph of file main.c

Figure 8

Mirai offers offensive capabilities to launch DDoS attacks using UDP, TCP or HTTP protocols.

Binary Analysis

Now let's move to binary analysis. So far we have been able to study 19 different samples obtained in the wild for the following architectures: x86, ARM, MIPS, SPARC, Motorola 68020 and Renesas SH (SuperH).

For the binary analysis we have used VULNEX BinSecSweeper platform that allows analyzing binaries among other things/files in depth combining SAST and Big Data.

In Figure 9 we see a chart showing all the files magic to give us an idea of the file types/ architectures. All samples are 32 bits.

Figure 9

By using BinSecSweeper we obtained a lot of information for each sample, similarities between them and different vulnerabilities. Currently not many Antivirus identify all the samples, so beware what Antivirus you use! In Figure 10 we have a visualization of file sizes in bytes.

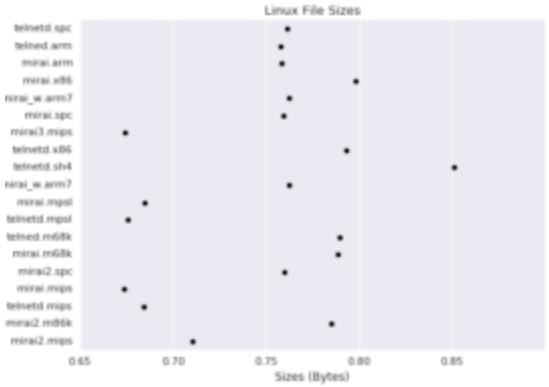

Figure 10

We analyzed all section names in the samples and Figure 11 is the result.

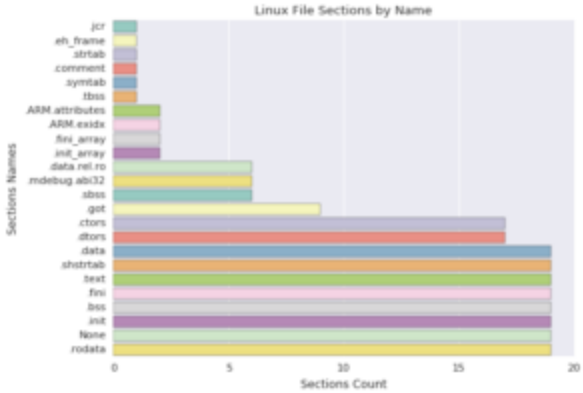

Figure 11

As mentioned before the samples are for different architectures so in this post we are not showing you the code analysis results.

We have updated BinSecSweeper analysis engine to identify Mirai malware samples. A full binary analysis report is available from VULNEX Cyber Intelligence Services to our customers, please visit our website or contact us.

Conclusions

Despite being a fairly simple code, Mirai has some interesting offensive and defensive capabilities and for sure it has made a name for itself. Now that the source code has been released, it is just a matter of time we start seeing variants of Mirai.

Mirai Botnet is a wakeup call to IoT vendors to secure their devices. Unfortunately millions of devices have been already deployed on Internet and there are insecure by default, so embrace yourself for more IoT attacks in the near future.

What do you think about IoT security?

— Simon Roses Femerling / Twitter @simonroses