CSE 2431 Final Project Midterm Report

Group Name: Unnamed

Mengyuan Li

Xuewei Wang

1 Introduction

In the final project, we design a malicious software based on the idea of remote shell. In our malicious software, the attacker functions as a server and the victim acts as a client. Besides some basic network components including attack listening to a given port and waiting the victim to automatically connect to the attacker, our malicious software includes some certain malicious activities(e.g., send certain commands to remote victim, get the results from the victim, steal browser's password, automatic startup and hide its presence) after the connection is established.

The main function in our malicious software includes:

- Execute the command line requested by the remote attacker and return the results (use tree function to help attacker to have a better look of victim's files).
- Read and transfer files to the attacker.
- Auto startup every time the system reboot.
- Continuous execution without victim's awareness and hiding our software in system directory.
- Steal victim's stored password and website preference in Firefox.

The rest of the report is organized as follows. In section 2, we will discuss how we implement different function in detail. In section 3, we will show a case study of our program. In section 4, we will discuss some alternative implementation choices of our malicious software with responsibility of each team member and conclusion in section 5 and 6.

2 Description of each function

(1) Execute the command line requested by the remote attacker and return the results.

Socket is first used in order to connect server and client. Since the server and the client are running on two hosts on the Internet, we chose Internet domain as the address domain. In addition, we chose stream sockets and TCP protocol over datagram sockets and UDP protocol. The server first creates a new socket, binds together, and wait for connection, at the same time the client creates a socket and established a connection to the server using its IP address and port number.

After the connection, read() and writer() functions can transfer messages between the server and the client. The commands entered by users will be automatically sent to the client, and then the client will run the message as a command and send the results back to the server with function popen(). The server can then receive the results from the client and display them on the terminal.

(2) Read and transfer files to the attacker.

Besides command line execution, the program can read the file and transfer it from the client to the server. The functionality is also performed by read() and write() calls. Computer are required to be in the reading mode and the command must be typed as "$ read filePath". Once the server program identifies it as a file reading instruction, the command string is split into 2 parts: "read" and the file path. The file path is then sent to the client and the client is designed to read the specified file and transfer the contents back to the server. Then the server would store contents locally and open it with vim.

(3) Auto startup every time the system reboot.

In order achieve auto startup every time the system reboot, our malicious software will check whether this is the first time it being executed. If the answer is no, it will skip the modification part and run the remaining part. And If the answer is yes, it will modify a certain system file "/etc/rc.local". After the modification, every time the system restart, it will run our malicious software.

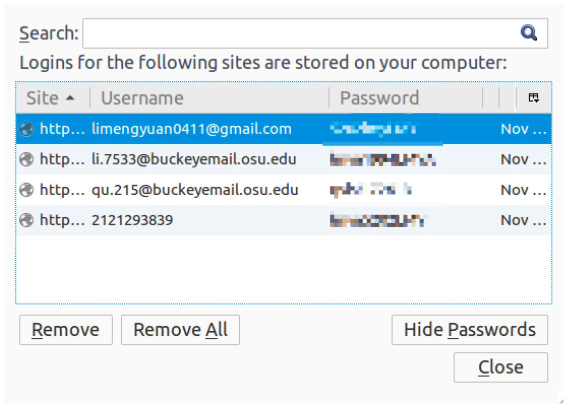
(4) Hiding our software in system directory and Continuous execution without victim's awareness.

During the first execution, our malicious software will backup itself in a system directory. So even if the original software is deleted, it can still work quietly. Moreover, we use a POSIX command "nohup" to ignore the HUP signal. Every time victim restart its device, he will not be able to notice our malicious software is continuous executed. Also, we will hide our log data in a unnoticeable system directory.

(5) Steal victim's stored password and website preference in Firefox.

Compared with transferring many normal data, able to get some private data such as victim's passwords sounds more exciting. Our malicious software is able to steal some browsers' stored passwords via file's transmission. Firefox is a pre-installed browser in Ubuntu and we use Firefox as an example. Attacker need to force victim's device to send 3 files to server, which is "key3.db", "logins.json" and "prefs.js" in

"/home/'username'/.mozilla/firefox/'xxx.default/'". The simplest way to decrypt those files are copying those file into your own Firefox directory and there are some other decryption approaches using different functions. The results can be shown in the figure below.


Search:

Logins for the following sites are stored on your computer:

Site	Username	Password	
http…	limengyuan0411@gmail.com	[redacted]	Nov …
http…	li.7533@buckeyemail.osu.edu	[redacted]	Nov …
http…	qu.215@buckeyemail.osu.edu	[redacted]	Nov …
http…	2121293839	[redacted]	Nov …

3 Evaluation

In this section, we will use a case study of our malicious software to help explain how our malicious software works. First, we will introduce the workflow of our malicious software.

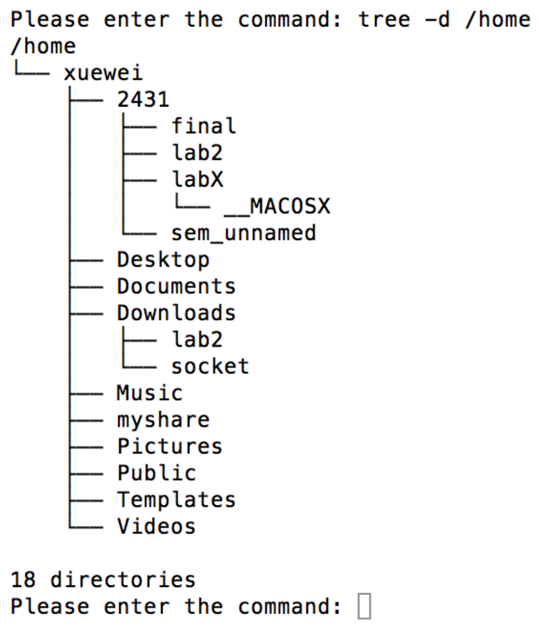
Step 1: In the victim's device, open the directory "client", makefile and run client with root permission. (Note that before we run the makefile, victim should modify attacker's IP address and port in "client.c", which should be modified in 3 points of "client.c" . This is to make sure victim can connect to a certain attacker's device).

Step 2: In the attacker's device, open the directory "server", makefile and run server using command. /server <port>. (Note that one should make sure the server is on when try to execute client. For a restart client, attacker need to make sure server is executed before victim login the system. Then the victim's device will automatically connect to attacker's device)

Step 3: After a successful connection is established, the attacker will receive a message and are able to use different command.

Step 4: If the attacker chooses to use the command line, he/she can enter "1" to enter the command line mode and begin to enter commands. Here is an example of command

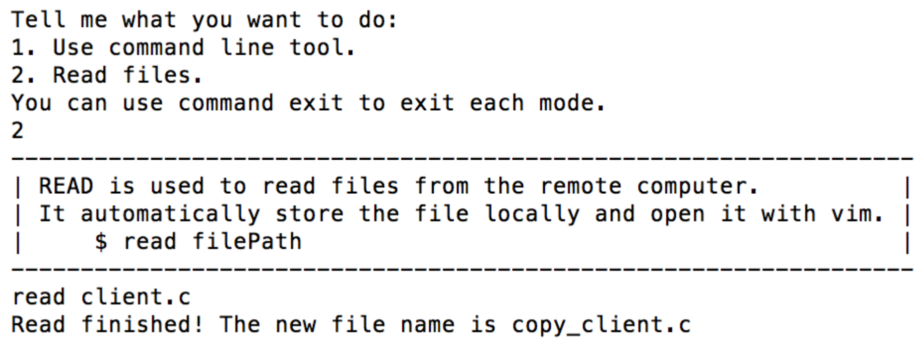
```
$ tree -d /home
```



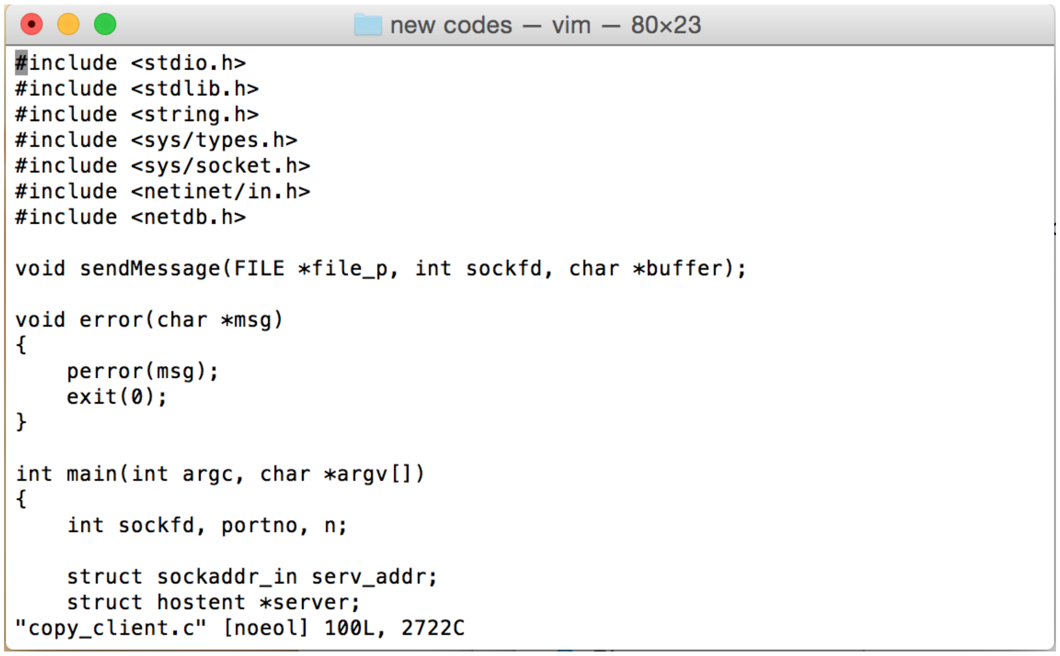
```
Please enter the command: tree -d /home
/home
└── xuewei
    ├── 2431
    │   ├── final
    │   ├── lab2
    │   ├── labX
    │   │   └── __MACOSX
    │   └── sem_unnamed
    ├── Desktop
    ├── Documents
    ├── Downloads
    │   ├── lab2
    │   └── socket
    ├── Music
    ├── myshare
    ├── Pictures
    ├── Public
    ├── Templates
    └── Videos

18 directories
Please enter the command: 
```

Step 5: If the attacker chooses to read files, he/she can enter “2” to enter the read mode and begin to enter reading commands. Here is an example of command

$ read client.c



```
Tell me what you want to do:
1. Use command line tool.
2. Read files.
You can use command exit to exit each mode.
2
-----------------------------------------------------------------
| READ is used to read files from the remote computer.          |
| It automatically store the file locally and open it with vim. |
|     $ read filePath                                           |
-----------------------------------------------------------------
read client.c
Read finished! The new file name is copy_client.c
```



```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <netdb.h>

void sendMessage(FILE *file_p, int sockfd, char *buffer);

void error(char *msg)
{
    perror(msg);
    exit(0);
}

int main(int argc, char *argv[])
{
    int sockfd, portno, n;

    struct sockaddr_in serv_addr;
    struct hostent *server;
"copy_client.c" [noeol] 100L, 2722C
```

4 Discussion on alternative implementation choices

The issue of the program is that the command line operations and the file operations have to be separated. If the attacker wants to read or write files in the command line mode, there is no response. The problem happens because the popen() function will only return a pointer to the open stream. An alternative choice to this problem is checking the command first before sending it to the client. If the command requires file operations, like reading and writing, the server will first send a transfer request to the client and store it locally. Then the server program will run system() call to run the command using the received file.

The way we use to hide malicious software to avoid detection is easily applying POSIX command "nohup". Combined with automatic startup and hidden files, victim can hard detect our malicious software. However, if the victim use command "ps" and carefully look through each ongoing process, he will be able to notice our latent malicious software. The ways to hide our software more completely includes using a proper framework, modify top/ps/… binaries, modify libc and modify the system calls in the kernel. These work can be done in our future work.

5 Responsibility

Mengyuan Li: Ways to hide malicious software to be detected. Some optional functions such as passwords inference or encrypt victim's files.

Xuewei Wang: Network communication including file transmission. Ways to read and transfer files remotely.

6 Conclusion

In the final project, we design a malicious software based on the idea of remote shell. The function of our work includes Execute the command line requested by the remote attacker and return the results (use tree function to help attacker to have a better look of victim's files), read and transfer files to the attacker, auto startup every time the system reboot, continuous execution without victim's awareness and hiding our software in system directory and steal victim's stored password and website preference in Firefox.

Some further work can be done based on some following plans: a) Automatically finish the software install step after victim click a malicious website; b) Write a program with less permission which is able to add and hide some short segments during victim editing a normal .c file. So if the victim compile and run the normal .c file, the install step of our malicious will be executed.