Reverse Shell Exploit

IT17049832 - H. K. N. Dabare

Contents

[The Concept of a reverse shell 2](#_Toc39991223)

[The Exploit 2](#_Toc39991224)

[The tools 3](#_Toc39991225)

[The script 4](#_Toc39991226)

[The Libraries 4](#_Toc39991227)

[The code 5](#_Toc39991228)

[Assigning the variables 5](#_Toc39991229)

[The functionality 5](#_Toc39991230)

[The payload delivery mechanism 6](#_Toc39991231)

[Alternative Versions 10](#_Toc39991232)

[Credits 10](#_Toc39991233)

[References 10](#_Toc39991234)

# The Concept of a reverse shell

Remote shells are usually used in client-server architectures for the ease of work. In such an instance, the client is usually the one who initiates the connection while the server listens for an input on certain ports, allowing the client to set up a shell. A reverse shell is the opposite. In this scenario, it is the server (in our case, the target) that will initiate the connection. The client (AKA attacker) will listen on a certain port and open a server shell when the connection is set. This means, that the client will open a server shell on the client.

The reason this is used among attackers is because of the way firewalls are usually configured. Servers usually block incoming connections, and only a few ports are usually open. These open ports might already be occupied by other processes as well. In such a scenario, it becomes difficult for an attacker to open a shell on a server. But the server firewall is less likely to terminate outgoing traffic. This makes it easier for an attacker to expose a reverse shell. An attacker just has to set up the client to listen on a specified port for the connection.

What opening a reverse shell would do is open up a server shell on the client, allowing the client (attacker) to run commands on the server. This could vary from simple directory traversing to downloading and executing malware.

# The Exploit

A screenshot of a computer screen

Description automatically generated

The exploit I’ve planned is opening a reverse shell. The attacker will be a Kali Linux Operating system. The target will be a Linux Ubuntu 16.04 Operating system. Both Operating systems chosen were Linux distributions to prevent any dependency breakdowns, but this should work on Windows as well. Both operating systems will be hosted on Oracle VirtualBox.

The exploit will be written in C language and the compiled executable will be used to expose the reverse shell.

# The tools

* Oracle VirtualBox

Obviously, VMware works as well, but I have used VirtualBox. There are a few settings which have been changed. The Network preference was changed to NAT network. A new NAT network was also created. When creating the two virtual machines (Kali and Ubuntu), the network host was set to NAT network, and the created network was assigned to it [1]. The reason for this is so that the two VMs will be on the same network. When creating the VMs, you could give lesser resources to the Ubuntu VM.

A separate set of tools were installed in the Kali VM for the purpose of creating and testing the exploit. They are as follows.

* GNU code compiler (gcc)

This is used to compile code written in C language, and as we will be using C, this should be installed in the Kali VM. It might be already installed ( try **gcc –help** to check if its already there) If not, type in **sudo apt install gcc** to the terminal.

* Nano text editor

Any text editor can be used for this exploit, but I’ve chosen the built-in nano text editor.

* Netcat

Netcat (nc) will be necessary to set a port to listen for any incoming connections. You could use **ifconfig** or **ip a** to identify the ip addresses on Ubuntu and Kali respectively (I am using the latest version of Kali, and as such, the net-tools containing **ifconfig** command isn’t there anymore. The alternative is **ip a** [2])

* Bzip2

Bzip2 is a compression tool which is usually delivered in Linux. This should be installed in both attack and target machines.

# The script

A screenshot of a computer screen

Description automatically generated

## The Libraries

* Sys/socket.h

This library is which assists us to configure a socket for communication purposes [1]. This is unique to the UNIX library, and as such, will not work on Windows. If the exploit is written on windows, you can use **winsock** (or its upgraded counterpart **winsock2**).

* Sys/types.h, stdlib.h, unistd.h & stdio.h

These are common libraries used in C. Even on this simple program, these libraries can help in debugging and logging as well. They were added for various functionality necessary in the program [2].

* Netinet/in.h & arpa/inet.h

These libraries contain the mechanism to communicate and configure network objects, which will be necessary to set up the connection between the target and attacker [2].

## The code

### Assigning the variables

**Sockt** is the variable for the socket we will be creating. It is this configured socket which will be used in the attack. This value is assigned to a socket I have created. The **socket** function takes in 3 arguments. The first is **AF\_INET** which is how we specify which type of IP address will be used. **AF\_INET** symbolizes IPv4 addresses while **AF\_INET6** can be used.

The next variable defines the type of connection, and for this purpose **SOCK\_STREAM** has been used. This is because SOCK\_STREAM initiates a seamless bidirectional connection which is necessary for the reverse shell to interact with the target. The final argument defines the protocol. The documentation states if 0 is used, a suitable default protocol for the selected type will be used. Thus, 0 was passed into the argument.

**Port** variable is an integer which holds the value of the port which we will be listening to. It has been assigned ‘4444’ but any open port number can be used here.

The **sockaddr**\_in is a structure defined in the netinet/in.h library. This will hold he socket address. For the purpose of this exploit, I will be using IPv4 addresses. If we are to use IpV6 addresses the structure which must be used is **sockaddr\_in6**. The variable we have defined of this structure is **revsockaddr**.

The revsockaddr structure contains three variables to be assigned. Thefirst is the **sin\_Family**. This value defines the type of IP address. **AF\_INET** will be assigned as IPv4 will be used. The **sin\_port** value takes the value of the port. The value defined earlier will be assigned to this, but at as this value must be assigned as net bytes, the value is passed through **htons()** function. The final value to be set is the **sin\_addr** which uses another struct defined in the netinet/in.h libray named **sin\_addr**. The value set is the IP address of the attacker. This is because, when the program is run on the target, it must identify the client to send the request to it. The IP address is passed into **inet\_addr** as the value must be sent as network bytes.

### The functionality

The main functionality of this code is to establish a connection with the target machine and open a reverse shell. The connection is established via the **connect()** function. The three arguments passed into this include the socket we created (**sockt**), the revsockaddr and the length of the revsockaddr. Since the struct **sockaddr\_in** cannot be used directly, it must be parsed as a **sockaddr**. I have used the **sizeof()** function to get the size of the revsockaddr variable.

The next section is to expose a terminal. **Execve** is the function used to execute a function and is found in the **unistd.h**. It takes in 3 arguments. The first is the pathname, which will be specified as **/bin/bash** (This is what actually runs when a terminal is open.) The next argument takes the **argv[]** array. This lets the function know what command line arguments are used with the script of the given path name. The documentation defines that the **argv[0**] should contain the filename (**/bin/bash** again) and the **argv[argc**] should be NULL. **Argc** is the number of arguments. In this instance, **argc** is 1. Therefore, the index 1 is set to NULL. The final argument is the **envp[]**. This holds a list of environment variables. The documentation states that the only requirement is that the array must e terminated with a NULL pointer (end with NULL). As I will not be sending any environment variables, the value passed will be **NULL** (It will act as an array with one item; a NULL) [1].

The **dup2()** functions are used to duplicate the file descriptors. File descriptor is an abstract indicator used to access a file or any other resource. This applies to sockets as well. Therefore, the file descriptors must be modified as well. The three **dup2()** functions use **sockt** as the old file descriptor and duplicate them with the values 0, 1 and 2. These values are the integer constants for the values **STDIN\_FILENO**, **STDOUT\_FILENO** and **STDERR\_FILENO** respectively. This allows the new socket to communicate with the relevant files, even on a device other than the host.

Since the **main** function was declared to return an integer, I will put return 0 at the end.

# The payload delivery mechanism

The payload will be the executable after the code written above is compiled. I will use a simple mechanism such as mailing the executable to the target, with the intention of getting the user of the target machine to download and run the file.

A screenshot of a computer

Description automatically generated

Even in such a simple mechanism, there are a few barriers.

1. Mail applications do not allow attaching executable files, or to send executable files. This hurdle can be overstepped by compressing the executable. **Bzip2** was used to compress the executable. This was selected as it is already built into Linux distributions including Kalin and Ubuntu.
2. Permissions are changed when a file is downloaded by another user. This means that the executable cannot be directly run even after decompressing it. Of course, in an event of a good social engineering attack or hiding it as a trojan horse will force the target to change the permissions and run the executable. Since the focus of this report is the exploit itself, we will assume that the target user willingly gives executable permissions and run the script.

Once the code has been successfully sent over, the attacker must now listen for any incoming traffic on port 4444.

A screenshot of a computer screen

Description automatically generated

The user of the target host will decompress the file, change permissions and run the file.

A screenshot of a computer

Description automatically generated

This will trigger the code and open the reverse shell on the client host.

A screenshot of a computer screen

Description automatically generated

# Alternative Versions

This exploit is specific to a certain scenario. But, there could many variants of this code which may cater to different scenarios. Though a few have been discussed in this report, I will list a few more to give an idea of other exploits which could be developed.

* IPv6 Compatible

As iterated before, the above code works when dealing with IPv4 addresses. To port it to work with IPv6 addresses, the following changes must be done

1. Change the value **AF\_INET** to **AF\_INET6.**
2. Change **sockaddr\_in** to **sockaddr\_in6**.
3. Redefine variables for **sockaddr\_in6**. This struct has 5 variables (including the 3 from **sockaddr\_in** except with a 6 in front of the sin- prefix; **sin family** is **sin6\_family**). The extra variables are **sin6\_flowinfo** and **sin6\_scope\_id**.

# Credits

Credits go to HackerSploit and his great tutorial on Exploit development.

<https://www.youtube.com/watch?v=YOtugkJbhy0&list=PLBf0hzazHTGPoP8BjixXswrM01ttNkfnp&index=11>

# References

|  |  |
| --- | --- |
| [1] | [Online]. Available: https://www.techrepublic.com/article/how-to-create-multiple-nat-networks-in-virtualbox/. |
| [2] | [Online]. Available: https://forums.kali.org/showthread.php?36392-bash-ifconfig-command-not-found!!!. |
| [3] | [Online]. Available: https://pubs.opengroup.org/onlinepubs/009695399/functions/socket.html. |
| [4] | [Online]. Available: https://pubs.opengroup.org/onlinepubs/009695399/basedefs/sys/types.h.html. |
| [5] | [Online]. Available: https://pubs.opengroup.org/onlinepubs/009695399/basedefs/netinet/in.h.html. |
| [6] | [Online]. Available: http://man7.org/linux/man-pages/man2/execve.2.html. |