

Remote SSH Tunneling Tool

Devansh Gupta
Computer Science & Engineering
Meerut Institute of Engineering and
Technology
Meerut, India
devansh.gupta.cs.2018@miet.ac.in

Gauri Sharma
Computer Science & Engineering
Meerut Institute of Engineering and
Technology
Meerut, India
gauri.shrikant.cs.2018@miet.ac.in

Kanika Chaudhary
Computer Science & Engineering
Meerut Institute of Engineering and
Technology
Meerut, India
kanika.chaudhary.cs.2018@miet.ac.in

Ashish Kumar
Computer Science & Engineering
Meerut Institute of Engineering and
Technology
Meerut, India
info2ashish.cs@gmail.com

Ajay Kumar Singh
Computer Science & Engineering
KIET Group of Institutions, Delhi-
NCR, Ghaziabad
ajay41274@gmail.com

Abstract— As technology is evolving day by day, the world is getting more connected and more volatile. In the modern world of technology, data is the most valuable asset one can have. People cannot afford to put their data on risk. The communication on the internet is not secure and is exposed to attacks. The login details, transactional details can be taken out and misused. The chief purpose of this paper is to show that data theft can be minimized by “SSH Tunneling”. In this work remote port forwarding and local port forwarding will be used. The idea of SSH tunneling is a good safeguarding mechanism in opposition to data breaches and can stifle communication on wireless networks. Also, how SSH tunneling provides encryption has been explained.

Keywords—SSH tunneling, Local port forwarding, Remote port forwarding, Packet sniffing, Wireshark

I. INTRODUCTION

Almost every organization uses wireless communication or the internet for their day to day tasks. Usage of the internet also adds some notable threats. Subtle data of users can be taken out, changed or reiterated for unethical purposes. There is easy availability of tools which are used for data stealing. So it has become easier for hackers to compromise the data of users on the internet. The purpose of this paper is to show SSH tunneling as a solution for providing a secure connection between the client machine and the server machine to keep away from data attacks. Port forwarding is a Secure Shell (SSH) feature that allows non-safe TCP/IP traffic to be routed across both public and private networks via a secure and encrypted connection.[1] In this work both local and remote port forwarding using socket programming is used in order to achieve the tunneling between sites.

Socket programming is a process which allows two nodes to be connected to each other for communication over a network. In this process, while the other node reaches out to build a connection, it listens on a certain port at an IP address. The server generates the listener socket while the client is connected to the server.

Local port forwarding is a function that allows a port on a local machine (also known as an SSH client) to be forwarded to a port on a distant machine (also known as an SSH server), which is then forwarded to a port on the destination machine. The SSH client waits for connections on a certain port and forwards them to a remote SSH server, which subsequently connects to a port on the target system. The destination can be a remote SSH server or any other system.

Distant port forwarding is a function that allows you to forward a port on a remote system (often referred to as an SSH server) to a port on a local machine (sometimes referred to as an SSH client), which is then forwarded to a port on the destination machine. The SSH server listens on a specific port and forwards any connections to that port to the local SSH client, which then connects to a port on the target system.[2] It makes no difference whether the destination machine is local or remote.

This research paper is arranged as follows: In section II, we will analyze packet sniffing attacks and describe the working of the Wireshark tool. In section III, we go over the basics of SSH tunneling, including how it's set up and how it operates. Concluding thoughts are offered in section IV, after that comes a list of references.

II. LITERATURE REVIEW

Packet sniffing is the technique of collecting and monitoring active data which moves over a network to gain an abstract of what is going on. It is also known as packet or protocol analysis.[3] A packet sniffer is a tool that recognizes atomic data traveling across a network and packet analyzing or sniffing is performed on that data. Packet analysis can assist us in defining network features, determining who has been working on the network, determining frequency and operation, recognizing highest network operational time, detecting different probable attacks or unfavorable activities, and identifying less secure and unusual applications. Packet sniffing programs come in a variety of shapes and sizes.[4] Every program is created with specific goals and objectives in mind. TCP dump (a command-line program), OmniPeek, and Wireshark are examples of popular packet analysis programs.

A packet sniffer interrupts and logs data packets present in the network so that they can later be decrypted in accordance with a set of rules. Wireless packet sniffers are frequently used because of their ability in network management and to monitor network activities at the MAC layer as well as layers above.[5] Packet sniffers, in particular, are commonly used to identify network problems and trace a specific task, but they can also be used to block sensitive or personal information like usernames and passwords.

Administrators can troubleshoot wireless networks using Wireshark's comprehensive wireless protocol analysis support. With the correct driver support, Wireshark can capture traffic "from the air" and decode it into a format that administrators can use to track down elements that are causing poor performance, irregular or weak connectivity, and a few other typical issues. Wireshark makes use of application programming interfaces (APIs) for recording network traffic, also called p cap, to capture packets. A variation in interfaces, including IEEE 802.11, Ethernet, and loopback interfaces, can be used to record live packet data. Recorded data can be presented along with detailed protocol information in the Wireshark GUI, and recorded data can be refined along with the generation of input/output statistical graphs.[6] Furthermore, the GUI makes it simple to track packets. Refining different features such as unique IP address, name of host or address of Ethernet host, TCP and UDP port numbers is possible with data packet filtering, which uses primitive expressions combined with and/or primitives.[7]

Packet sniffing is the process of recognizing, decrypting, auditing, and transcribing the data contained in a network packet on a TCP/IP network. The primary goal is to sniff and take away data, such as personal passwords, different port numbers, unique credit card numbers, user identities, network information, and so on. Sniffing is a type of "passive" attack in which the attackers are present but remain silent or hidden on the network.[7] As a result, it's complicated to spot and detect, making it a potentially deadly attack. TCP/IP packets

contain critical information that allows interfaces of two different networks to communicate with each other.

Both the source and the destination fields include IP addresses, ports, sequence numbers, and protocol type. TCP/IP ensures that a packet is formed, placed on an Ethernet packet frame, and securely sent from the sender to the recipient over networks by virtue of its architecture. However, there are no measures in place to secure data by default.

As a result, the responsibility for ensuring that data in packets is not tampered with falls to the top network layers.

III. PROPOSED METHODOLOGY

In this section, we will discuss the methodology that we will use to implement SSH Tunneling. SSH Tunneling is useful for sending network data from services that require an unsecured protocol, such as VNC or FTP, as well as accessing geographically-restricted content and getting through intermediate firewalls. Basically, any TCP port can be forwarded and traffic tunneled through a secure SSH connection.[8] It even provides us with an easy solution to expose HTTP/HTTPS endpoints over the Internet/Intranet. It can also be considered as an easy alternative over legacy SSH Port Forwarding Tools like Linux SSH Client, Putty etc.

Virtual and secure tunnels over a wireless network are provided by the mechanism of SSH tunneling, in which client machine and server machine act as terminals of the tunnel. The virtual tunnel can be considered as a secured way to transport the data from the client node to the server node. All the data that passes through this virtual tunnel is by default secured. A virtual tunnel is required if a client wants to communicate or access the data of the server. The client can acquire this tunnel with the help of a mechanism called Port Forwarding. Over the internet, port forwarding allows computers or services in private networks to communicate with public or private computers or services.

Our SSH Tool will have the following components –

- Backend Server- It is used for authentication of Users. Successful authentication allows the users to create SSH Tunnels and further access the tool.
- SSH Certificate Repository- It acts like an SSH certificate authority that provides SSH certificates to users so that they can use or access our tool to create SSH Tunnels. It is also used to hold the SSH Certificates for the users since we will use certificate based authentication for authenticating our users. An SSH certificate is a way for one SSH key to sign the signature of another SSH key. SSH Certificate based authentication uses a signed certificate attached to each key for verification of identities.[9]
- Command Line Interface- The Command Line Interface (CLI) is a text-based application that allows users to reply to

visual prompts by entering single commands and receiving an immediate answer. In our tool, CLI will be used to access the features of the tool through simple commands.

- **Graphical User Interface-** A graphical user interface (GUI) is an interface that facilitates interaction with various electronic devices by using icons and other visual indicators. As the Command Line Interface may become quite complicated and challenging to learn, a Graphical User Interface will be developed. The GUI will provide us with an easy user friendly interface to access the tool. It is dependent on the Command Line Interface.

- **SSH Server-** An SSH server is a piece of software that accepts connections from distant machines via the secure shell protocol. It can be established to manage SSH transfers. It will be used for the creating and accessing encrypted Tunnels.

The SSH Tool will have the following features -

Authentication - It involves user signup, login and logout.

Configuration Setup - It is used to set up local as well as environment variables. A local variable (also known as a Shell Variable) is used to configure the shell, store data, or output commands.[10] By default, it is local to a single shell and is not inherited by any child shells. On the other side, environment variables are used to configure other commands or applications. They are inherited by child shells. This configuration setup is dependent on the CLI. It includes -

- Setup Configuration
- View Configuration

Local Tunnel Management - A Local SSH Tunnel is used to establish a connection between a local machine and a remote machine using TCP ports. In our SSH Tool, this feature will enable us to create a new as well as duplicate an existing local port. For example - copying the data of port X on port Y. It is also dependent on the CLI. It includes-

- **Connecting Local Tunnels-** This service requires the source port (the port to duplicate) as well as the destination port (the newly available Port)

- **Disconnecting Local Tunnels-** This service requires the destination port (the newly available Port) only.

Remote Tunnel Management- We can redirect a port on the distant machine to a port on the local machine using a Remote SSH Tunnel. We can create as well as disconnect remote tunnels through this feature. It is a very handy and easy to use feature through which we can have access to remote tunnels. It is dependent on CLI, Backend server,SSH Certificate Repository as well as SSH Server. It includes-

- **Connecting Remote Tunnels-** This requires the source as well as destination port.

- **Disconnecting Remote Tunnels-** This requires only the destination port.

List Local/Remote Tunnels - Through this feature, the users will be able to see the list of created/available tunnels. It consists of simple commands, with filters to show only Local/Remote Commands. It is reliant on both the CLI and the Backend Server.

The basic flow of working of our Tool (as shown in Fig 1) will be as follows-

1. The GUI of our Tool will ask CLI to create SSH Tunnels.
2. CLI would further ask the Backend Server for Authentication.
3. The Backend Server is supposed to return an SSH Certificate for the user which it does by requesting the SSH Certificate Repository to give an SSH Certificate for the user.
4. The Backend Server then returns this SSH Certificate to the CLI.
5. The CLI then requests for Tunnel creation to the SSH Remote Server.
6. SSH Remote Server updates DNS records if (custom) subdomain is requested.

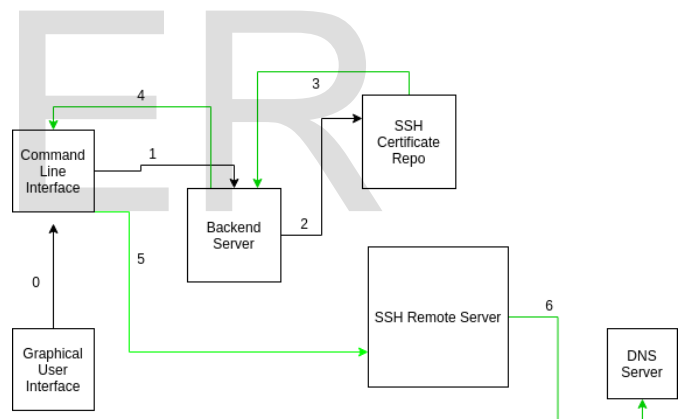


Figure 1. Methodology

IV. RESULT DISCUSSION

A. CONFIGURATIONS

The configuration of SSH tunneling tool is described as follows-

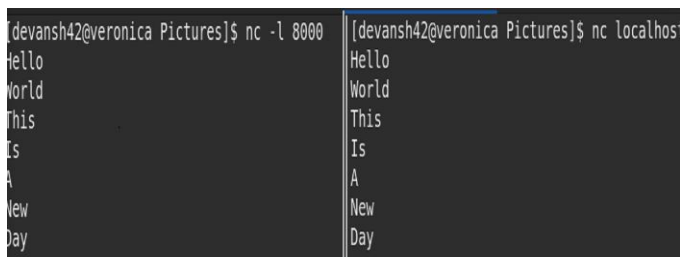
- To sign in to use the tool-
sign- command for login/logout
- To know the current user-
whoami, who- reveals the current user
- To make Local Tunnels-
connect- makes a local tunnel from source port to destination port [11]

- To disconnect local tunnels-
disconnect- turn off the local tunnel
- To set Command Line properties-
set- sets CLI properties like remote server address
- To get specific properties-
get- gets any specific property e.g. get -- backend
- To make remote tunnels-
expose- makes a remote tunnel from source (on remote machine) to destination (on local machine)
- To disconnect remote tunnel-
unexpose- turn off the remote tunnel
- To get the list of active tunnels-
list- lists local/remote tunnels
- To get any help-
help, h- shows a list of commands or help for one command
- To exit from the tool-
exit, quit- exit gracefully

B. Result of Features

- Creating Local Tunnels-

Local Tunnel makes it simple to expose a web service on your local development workstation without having to fiddle with DNS or firewall settings. Local Tunnel will provide you with a unique publicly available URL that will be used to redirect all requests to your local web server. This functionality in our SSH tool will allow users to create a new local port as well as replicate an existing one. The result of this feature allows users to create and disconnect local tunnels with extreme ease. As shown in fig 2, a local tunnel has been created at port number 8080 which is used to access the source port i.e port number 8000.



```
[devansh42@veronica Pictures]$ nc -l 8000 [devansh42@veronica Pictures]$ nc localhost:8080
Hello Hello
World World
This This
Is Is
A A
New New
Day Day
```

Figure 2. Local Tunnel

- Creating Remote Tunnels

A remote tunnel enables remote control of items that are not connected to your local network. If you are simply and remotely controlling equipment on your company's internal network, then you don't need to set up a tunnel. As shown in fig 3, a remote tunnel has been created at port number 8080

which is used to access the destination port i.e port number 8000.

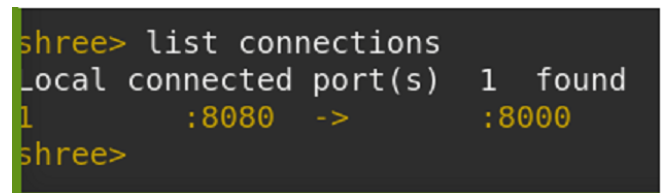


```
shree> connect --src 8080 --dest 8000
Successfully Port Forwarding Established
[::]:8080 -> :8000
shree>
```

Figure 3. Remote Tunnel

- Listing of Tunnels-

Listing of tunnel features results in a list of all the active local and remote tunnels. This feature can be accessed using both Command Line Interface and Graphical User Interface. Users can filter out the remote and local tunnels using GUI as per their convenience. A list of tunnels with their source and destination ports can be seen in fig 4.



```
shree> list connections
Local connected port(s) 1 found
1 :8080 -> :8000
shree>
```

Figure 4. List of Tunnels

V. CONCLUSION

SSH tunneling is a simple and efficient method of delivering secure content over the internet. This method will protect internet usage from packet sniffing efficiently. When compared to other security measures for links, networks, and applications, as well as their setup and configuration, Secure Shell is a relatively safe, dependable, rapid, and simple programme. Companies can construct a comprehensive security system by implementing Secure Shell, a tunneling platform that can be utilized for a variety of purposes which deploy a wide range of security procedures to ensure data security, numerous people's privacy, legitimacy, authorisation, and integrity various applications This research paper explains the fundamentals and implementation of SSH Tunneling.

VI. REFERENCES

1. D. J. Barrett and R. Silverman, "SSH, The Secure Shell - The Definitive Guide," *Book*. p. 438, 2001
2. P. Range, "Port Forwarding," *Current*. pp. 2008–2008, 2008.
3. Brian Wippich, "Detecting and Preventing Unauthorized Outbound Traffic", GCIH Gold Certification, October 15th, 2007. Copyright: SANS Institute.

4. Chris Sanders and Chris Sanders, "Practical Packet Analysis", May 17, 2007 ISBN-13: 978-1-59327-149-7. Communications Security Corp, January, 2006, Copyright: The Internet Society
5. ANSI/IEEE, "802.11: Wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) Specifications," 2000.
6. Sung Jun Ban, Hyeonwoo Cho, Chang Woo Lee, and Sang Woo Kim, "Implementation of IEEE 802.15.4 Packet Analyzer", International Journal of Electrical and Electronics Engineering, May, 2008.
7. M. Heusse, F. Rousseau, G. Berger-Sabbatel, and A. Duda, "Performance Anomaly of 802.11b", Proc. IEEE INFOCOM, April, 2003, pp. 836-843.
8. Md. Kamrul Hasan, A.H.M. Amimul Ahsan, and M. Mostafizur Rahman, "IEEE 802.11b Packet Analysis to Improve Network Performance" JU Journal of Information Technology (JIT), Vol. 1, June, 2012.
9. C. Lonvick, 'Ed Cisco Systems Inc.', "RFC 4251 - The Secure Shell (SSH) Protocol Architecture", SSH
- 10.H. Dwivedi, "Implementing SSH : strategies for optimizing the Secure Shell." p. 376, 2004.
- 11.P. Kirkbride, "Using SSH," *Basic Linux Terminal Tips and Tricks*. pp. 89–106, 2020, doi: 10.1007/978-1-4842-6035-7_5.

IJSER