Part 1

The non-trivial passwords we chose for our virtual Linux firewall host and for our virtual Windows host are:

- **Linux** : JlJtR | hIwTw
- **Windows** : Wwg_12,uTe0a1

We used Schneier scheme to make our passwords non-trivial and hard to guess using the prevalent password cracking mechanisms. As Schneier suggests, we took sentences that all of us had a personal context for and could easily remember and turned it into a password. This process provides protection against dictionary attacks as carefully chosen passwords in this manner will not be in anyone's dictionary.

JlJtR | hIwTw is derived from *"Jeff like John the Ripper | he Is weird That way"*
Wwg_12,uTe0a1 is derived from "*We were group_12, until The end O(substituted the number '0' for the letter 'O')f assignment 1."*

Information regarding the Schneier scheme was obtained from here.

Part 2

Keeping in line with the Schneier scheme, we combined a memorable sentence with some personally memorable tricks to modify that sentence into a strong password. The passwords we chose for the two regular Linux user accounts and the two regular Windows user accounts are:

Linux:

user1 : tiLU1_
user2 : TIlu2_

Windows:

user1 : Mi@Dim7T
user2 : teS6_nat7

For the web server, it authenticates by first making sure the username entered is the one that is required by that page. For user1's directory, this is user1. For user2's directory, this is user2. For the password, it'll hash the password and then check it against the password hash stored in the password file. If these match, the user is allowed access. If they don't, they need to try again. It allows 3 attempts, and after that, it sends the user to an error page.

As for the FTP server, it authenticates the user by first getting the username. If the user is allowed to connect, then they'll be allowed to enter their password. If they're not allowed, then they'll get access denied. When an allowed user enters their password, it'll hash it using MD5 and check it against the user's password stored in /etc/passwd. If they match, the user is logged in. Else, they're denied access.

For the TFTP server, since it's all anonymous and doesn't require the user to enter a username, there is no authentication process. It just makes sure they're connected to the right network.

The web server and the FTP server are similar in manner that both will hash the password in some way and check it in a file that has the stored passwords for the selected users. However, they are different in the manner that for the FTP server, you aren't given a second attempt to enter a username/password, it just goes straight to "access denied". They also have separate username and passwords for the login attempt. For the HTTP server, you must provide both the username and password at the same time, and you're allowed 3 attempts before it tells you that authentication failed. For TFTP, obviously since it's anonymous in our case, it doesn't require login information. The FTP and HTTP hashes are also different when setup properly, as FTP uses MD5 and HTTP uses LM/NTLM.

Part 3

In this part of the assignment, we create a firewall to manage the incoming and outgoing traffic for the two unique clients (Windows and Linux). We have put all the rules for the firewall inside a shell script called firewallrules.sh.

<u>Running the script:</u>
NOTE: These firewall rules should be running on our LINUX firewall VM when you log into it. You will be able to view these rules using the iptables –L command.

If the firewall rules aren’t active, you can run the script using the command ./firewallrules.sh

Following are the rules we have put inside our firewall script, comments explaining the rules will be denoted by **bolded sentence** starting with a ”**#**”:

# Logging
# We used this chain setup to implement logging any violations of the listed rules since the only violations of the rules should be when a host that is supposed to be dropped tries to connect to the server it's not allowed on. So they'll be logged then dropped.

```
iptables -N LOG_AND_DROP
iptables -A LOG_AND_DROP -j LOG --log-prefix "Source host denied "
iptables -A LOG_AND_DROP -j DROP
```

# HTTP Server Access
# We used these rules to help forward connections or drop them from accepted IPs to the Windows HTTP server because they aren't connecting with a HTTP server within the Linux VM itself, but the Windows VM that's connected through Linux. So they need to be forwarded and need NAT to allow them to connect to our HTTP server. As well, since they're allowed to connect to either port 80 or port 8080, we need to make sure that both ports end up being allowed, but go to the same place. It makes sure Group 11 (10.229.11.* and 10.229.100.11) as well as 10.229.97.* and 10.229.100.97 can’t access, but all other hosts on network 10.229.*.* can.

```
iptables -A FORWARD -p tcp --dport 80 -s 10.229.97.0/24 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 80 -s 10.229.100.97 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 80 -s 10.229.11.0/24 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 80 -s 10.229.100.11 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 80 -s 10.229.0.0/16 -j ACCEPT

iptables -A FORWARD -p tcp --dport 8080 -s 10.229.97.0/24 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 8080 -s 10.229.100.97 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 8080 -s 10.229.11.0/24 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 8080 -s 10.229.100.11 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --dport 8080 -s 10.229.0.0/16 -j ACCEPT

iptables -t nat -A PREROUTING -p tcp --dport 80 -j DNAT --to 10.229.12.2:8080
iptables -t nat -A PREROUTING -p tcp --dport 8080 -j DNAT --to 10.229.12.2:8080
```

# FTP Server Access
# We used these rules for the the FTP server because the FTP server is on port 21. However, it also uses port 20 for its active mode, and ports for source and destination including and after 1024 for passive mode. Since the ports 8080 (HTTP), 2221 and 2222 (both SSH) are used, those don't get included, but all others do. It makes sure that Group 13 (10.229.13.* and 10.229.100.13) as well as 10.229.96.* and 10.229.100.96 can’t access but the other hosts on network 10.229.*.* can.

```
# Since TCP is bidirectional, that makes it necessary to make sure that we have the input rules and matching output rules for all ports.
iptables -A INPUT -p tcp --dport 21 -s 10.229.96.0/24 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 21 -s 10.229.100.96 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 21 -s 10.229.13.0/24 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 21 -s 10.229.100.13 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 21 -s 10.229.0.0/16 -j ACCEPT

iptables -A OUTPUT -p tcp --sport 21 -d 10.229.96.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 21 -d 10.229.100.96 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 21 -d 10.229.13.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 21 -d 10.229.100.13 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 21 -d 10.229.0.0/16 -j ACCEPT

iptables -A INPUT -p tcp --dport 20 -s 10.229.96.0/24 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 20 -s 10.229.100.96 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 20 -s 10.229.13.0/24 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 20 -s 10.229.100.13 -j LOG_AND_DROP
iptables -A INPUT -p tcp --dport 20 -s 10.229.0.0/16 -j ACCEPT

iptables -A OUTPUT -p tcp --sport 20 -d 10.229.96.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 20 -d 10.229.100.96 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 20 -d 10.229.13.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 20 -d 10.229.100.13 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 20 -d 10.229.0.0/16 -j ACCEPT

# Source port and destination port are both used with the passive mode rules since the packets get transported between those two ports.
iptables -A INPUT -p tcp --sport 1024: --dport 1024: -s 10.229.96.0/24 -j LOG_AND_DROP
iptables -A INPUT -p tcp --sport 1024: --dport 1024: -s 10.229.100.96 -j LOG_AND_DROP
iptables -A INPUT -p tcp --sport 1024: --dport 1024: -s 10.229.13.0/24 -j LOG_AND_DROP
iptables -A INPUT -p tcp --sport 1024: --dport 1024: -s 10.229.100.13 -j LOG_AND_DROP
iptables -A INPUT -p tcp --sport 1024: --dport 1024: -s 10.229.0.0/16 -j ACCEPT

iptables -A OUTPUT -p tcp --sport 1024: --dport 1024: -d 10.229.96.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 1024: --dport 1024: -d 10.229.100.96 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 1024: --dport 1024: -d 10.229.13.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 1024: --dport 1024: -d 10.229.100.13 -j LOG_AND_DROP
iptables -A OUTPUT -p tcp --sport 1024: --dport 1024: -d 10.229.0.0/16 -j ACCEPT

# TFTP Server Access
# We used these rules for TFTP because it's on port 69 and must deny access from Group 1 (10.229.1.* and 10.229.100.1) as well as 10.229.96.* and 10.229.100.96, while allowing access from all other 10.229.*.* network hosts. As well, since it uses UDP, that is required instead of TCP.
# Since UCP has to match the rules used for TCP, that means it’s treated as if it’s bidirectional, which makes it necessary to make sure that we have the input rules and matching output rules for all ports.
# We decided to use UDP services for TFTP because TFTP supports the use of UDP. This is because of UDP's simplicity. In UDP, unlike TCP, it doesn't make sure all packets are sent. It just sends them in a steady stream where they may or may not make it. Because of this, it has reduced overhead and much faster speed than TCP. TFTP in itself doesn't require the amount of work done by FTP, HTTP and SSH in order to actually send over a file. It just sends the packets of the file over when the user requests that file. In order to make sure the recipient actually got the file, since packets can drop, TFTP will require the recipient to acknowledge they got the file.
iptables -A INPUT -p udp --dport 69 -s 10.229.96.0/24 -j LOG_AND_DROP
iptables -A INPUT -p udp --dport 69 -s 10.229.100.96 -j LOG_AND_DROP
iptables -A INPUT -p udp --dport 69 -s 10.229.1.0/24 -j LOG_AND_DROP
iptables -A INPUT -p udp --dport 69 -s 10.229.100.1 -j LOG_AND_DROP
iptables -A INPUT -p udp --dport 69 -s 10.229.0.0/16 -j ACCEPT

iptables -A OUTPUT -p udp --sport 69 -d 10.229.96.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p udp --sport 69 -d 10.229.100.96 -j LOG_AND_DROP
iptables -A OUTPUT -p udp --sport 69 -d 10.229.1.0/24 -j LOG_AND_DROP
iptables -A OUTPUT -p udp --sport 69 -d 10.229.100.1 -j LOG_AND_DROP
iptables -A OUTPUT -p udp --sport 69 -d 10.229.0.0/16 -j ACCEPT

# SSH
# We used these rules for SSH because it's on port 22, but the Linux VM is on 2221 and the Windows VM is on 2222, so those need to be included as well, and all networks are allowed to connect to them.
iptables -A INPUT -p tcp --dport 22 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 22 -j ACCEPT
iptables -A INPUT -p tcp --dport 2221 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 2221 -j ACCEPT
iptables -A INPUT -p tcp --dport 2222 -j ACCEPT
iptables -A OUTPUT -p tcp --sport 2222 -j ACCEPT
```

# ICMP
# We used these rules for ICMP because echo messages (or replying) are of type 0, and responses (pings) are of type 8. Since all networks are allowed, no restrictions are placed on the IP address. As well, since ICMP is bidirectional, and we need to allow our hosts to request and respond, output with those types are accepted as well.

```
iptables -A INPUT -p icmp --icmp-type 8 -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type 8 -j ACCEPT
iptables -A INPUT -p icmp --icmp-type 0 -j ACCEPT
iptables -A OUTPUT -p icmp --icmp-type 0 -j ACCEPT
```

# Windows VM Access
# Since we are restricting the Windows VM, we are restricting the forwarding of packets from port 8080. So we make sure first that NAT has the proper rule for the source port 8080. Next, since we're only restricting the Windows VM from accessing Group 13 (10.229.13.0/24 and 10.229.100.13) as well as 10.229.96.0/24 and 10.229.100.96, we make sure those are dropped if they are encountered, and allow it to go anywhere else if it passes those rules. In the nat rule, 255.255.255.0 is used as the to destination so that way it can go to any network.

```
iptables -A FORWARD -p tcp --sport 8080 -d 10.229.96.0/24 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --sport 8080 -d 10.229.100.96 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --sport 8080 -d 10.229.13.0/24 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --sport 8080 -d 10.229.100.13 -j LOG_AND_DROP
iptables -A FORWARD -p tcp --sport 8080 -j ACCEPT

iptables -t nat -A PREROUTING -p tcp --sport 8080 -j DNAT --to 255.255.255.0
```

# Default Policy
# The default policy is to deny incoming and outgoing access to everything that is NOT specified in some manner. Anything that fails to go through the chain at some point is dropped

```
iptables -P INPUT DROP
iptables -P FORWARD DROP
iptables -P OUTPUT DROP
```

Workload Split

Group 12 followed an ”equal contribution” model for the sliding part of assignment 1 and the non-sliding part of assignment 2. While Jeff primarily worked on cracking the passwords for the sliding part of assignment 1, Deborsi and Tiegan followed the pair-programming model to work on the non-slidings parts of assignment 2. All of the group members are happy and satisfied with each other’s contribution to the completion of the assignment (s).
