Linux Academy

Kali Linux Deep Dive

Study Guide

Ermin Kreponic

ermin@linuxacademy.com

January 18, 2019

Contents

Social Engineering

Objectives

The purpose of this section is to introduce you to social engineering and help you understand how hackers think. This module will give you insight into the many ways hackers use situations and human psychology to their advantage as well as provide tips on how to protect yourself and avoid these attacks. We'll also discuss several real-life examples of how hackers have used social engineering to accomplish their goals.

Introduction to Social Engineering

The Hacker State of Mind

Whenever you hear the term "hacker" or "hacking," you might picture a person sitting at a computer performing some malicious activity. In reality, the hacking process doesn't start at a computer; it starts in the hacker's mind. Hackers think differently than the average person. Hackers are constantly observing their surroundings, thinking about what information would be beneficial to them and how they can get it. They are persistent, adaptive, and imaginative and see every situation as an opportunity they can exploit.

There are several things a hacker might observe before entering a building. First, they would note the general appearance and layout of the building and what the entrance looks like. They would determine whether there are any credential checks at the entrance and if so, what kind (e.g., a key card scanner or security guard). The hacker would think about whether or not authentication occurs at multiple points within the building or only at the main entrance. They might also determine whether the building is shared by multiple companies or only used by one. Before entering the building, it is important for the hacker to look and act like the people around them to avoid suspicion.

For example, let's say that Joe is a hacker, and he decides to enter a hotel. If he looks decent and doesn't attract attention, he most likely won't be bothered by anyone. If the hotel has a public computer, there are several things Joe can do without any interruption. He can copy all of the browser profiles from the computer to a USB drive and open them up on his own computer later to see if someone forgot to log out or if someone saved their login information. Next, he can check to see if he is able to install and run programs on the PC and potentially install a keylogger or rat, or convert the computer into a zombie.

Another thing Joe can do is stand in line at the reception desk to hear the name, surname, and room number of the person in front of him. One of the most common ways of authenticating to Wi-Fi in a hotel is by using these exact credentials. Now Joe has Wi-Fi access to the entire hotel. He can then observe how rooms are accessed—by card or key. If it's by card, he can note the manufacturer of the card and so on.

As you can see, there is an abundance of seemingly insignificant information that a hacker can gather, knowing its potential benefit. All of this is part of the hacker's mindset.

Ways a Hacker Can Exploit You

- Many people who use ATM machines immediately throw away their receipts in the trash. If you have a large amount of money in your account and the wrong person notices, you are automatically a worthy target for theft.
- Hackers can place a hidden card reader and keypad over an ATM, allowing them to collect private credit card information.

- Since wireless credit cards have been put into use, hackers have come up with new ways of stealing information. By carrying a hidden card signal reader and walking through a crowded area, it is possible to scan NFC devices to steal credit card information. A hacker can scan a million credit cards like this and then extract a small amount of money from each account.
- Using obvious methods such as eavesdropping or spying, it is possible to learn someone's password in a public place.
- In the hands of a hacker, a USB drive is a powerful device that can do a lot of harm to a victim just by being plugged into their computer.
- Hackers can infiltrate computers by uploading malicious files (files that appear normal but have been modified with malicious code) to the internet for victims to download.

How to Avoid Hacker Exploitation

- If you own wireless credit cards, carrying an RFID-blocking wallet can help you block unwanted credit card scanning. However, the safest option is not to have wireless credit cards in the first place.
- Checking your accounts frequently and knowing your payment history can help you spot suspicious activity quickly.
- Paying attention to your surroundings, not logging in to accounts in public, and not saying your credentials out loud are basic (but important) precautions you can take.
- Another security measure is to not allow others to handle your devices. Sometimes, even people you know can pose a threat and unknowingly put your device at risk.
- Lastly, it is important to be cautious about what you download from the internet.

Social Engineering Attacks

Social Engineering

Social engineering is a term used to describe a collection of psychological activities that a hacker does to manipulate their target into either completing a specific task or revealing certain information. This is beneficial for the manipulator and, in most cases, harmful for the target.

Social engineering is a powerful technique that can be used to coordinate large attacks. For example, a hacker recently managed to leak the social security numbers, home addresses, and first and last names of 50,000 people who worked for the US Department of Justice (DOJ). He accomplished this using primarily social engineering. The attacker managed to obtain an intern's email address and used the names of other DOJ employees to contact the intern and request information. Since these people were all in the victim's circle of trust, the hacker's requests for information were successful. He continued making requests until he had access to virtually every employee's personal information.

Hardware Checks

When using any device, it is good practice to know and check the hardware beforehand. Many cases of fraud, credit card theft, and information breaches have happened as a result of either neglected hardware checks or a general lack of device security.

Here are a few examples of how hackers have taken advantage of lax device security to exploit their targets:

- Hackers conducted a massive ATM and bank heist by modifying card readers before they were even installed at their designated locations. The attackers figured out that these devices were being manufactured somewhere in Asia. The devices were then shipped to a storage facility in Dubai before they were eventually shipped to Europe. While the devices were in Dubai, the hackers broke into the storage facility, modified the devices, and closed them back up. Once the devices were installed in Europe, the attackers let the devices collect data, including card numbers, PIN numbers, and everything else they needed to create card replicas. Using this data, the attackers were able to max out a lot of cards and steal a lot of money.

- In another case, a company put their private information at risk despite having invested a lot of money in security. The company paid a penetration testing company to test their system and evaluate their overall security. To bypass the company's security, the penetration testing company inserted keyloggers into brand-new keyboards and sent them to the employees as a gift with a note that said, "From management, for a job well done." The employees took the bait and started using the new keyboards without checking the hardware. In a few days, the penetration company was in possession of every employee's credentials.

- Devices like air conditioners and vending machines might not seem like a big security problem on the surface, but in reality, even the simplest of devices can be used to wreak havoc. Since these devices provide an electrical source, other hardware devices like Raspberry Pis can be installed on top of them and become a part of a building's internal network. These devices can then be configured to act as a wireless access point that people can connect to, at which point attackers can steal their data.

- Hotel lobby computers are usually linked to the same network as the hotel's payment system, where accounting and customer information is stored. This makes it easy to breach the system.

As you can see, hardware checks are extremely important. Many devices contain hidden seals that you can check to see if the device has been tampered with. In addition, whenever you order something online, you should always order from a verified seller.

Key Terms

- **Zombie Computer**: A zombie is an internet-connected computer that has been breached by a remote hacker or affected by a virus or trojan and can now be used by the hacker.

- **Keylogger**: A keylogger is a piece of hardware or software that logs all of the keys typed on a keyboard.

- **Black Hat Hacker**: Black hats are hackers who use their knowledge and skills to discover and exploit security vulnerabilities for financial gain or other malicious reasons. Their activities can cause major damage to their targets and their target's systems.

- **Carding**: Carding is the illegal use of credit cards by unauthorized people. It involves a range of activities related to credit card fraud, trafficking, and money laundering.

Anonymity Online

Virtual Private Networks

A virtual private network (VPN) is a network used to establish a secure connection to a private network through the internet. VPNs enable secure connections, access to remote network resources such as files and databases, and the transfer of information between different networks. VPNs also allow users to hide their IP address location and online activity. Companies use VPNs to provide access to their services worldwide.

Key Terms

- **Point-to-Point Tunneling Protocol (PPTP)**: An older protocol with good speed. However, it is not secure, as it does not provide data encryption.
- **IPSec/Layer 2 Tunneling Protocol (L2TP)**: Normally used in mobile devices. It is more secure than PPTP, but it is not always as fast.
- **IPSec/Internet Key Exchange version 2 (IKEv2)**: A proprietary protocol developed by Microsoft. This protocol is very fast and secure and works well with different operating systems and mobile devices.
- **OpenVPN**: An open-source protocol based on SSL/TSL. It uses keys to secure the connection between two parties and is considered the best and most secure protocol.
- **SSH**: Secure Shell (SSH) is a cryptographic network protocol that allows secure operating of network services over unsecured networks.
- **SSH Keys**: SSH keys are public-private key pairs that provide an easy and highly secure way of accessing servers.
- **Vim**: Vim (Vi Improved) is an efficient text editor with both a command line interface and GUI. It is based on its predecessor, vi.
- **Certificate Authority**: A certificate authority is an entity that supplies public key certificates.
- **Diffie-Hellman Key Exchange**: A method for exchanging cryptographic keys securely through a public channel.
- **HMAC**: HMAC (Hash-Based Authentication Code) is a type of message authentication code that is extracted by running a cryptographic hash function over the data and a secretly shared cryptographic key.
- **Client Certificate**: A client certificate is a digital certificate that is used to make verified requests to a remote server.
- **`tee` Command**: The `tee` command reads from standard input and writes to standard output and files.
- **Transport Layer Security**: TLS (Transport Layer Security) is a cryptographic protocol that provides communications security over a network.
- **UFW**: UFW (Uncomplicated Firewall) is designed to be easy to use through a command line interface. It is used for managing Netfilter firewalls and uses `iptables` for configuration.
- **IP Masquerading**: IP masquerading is a networking ability in Linux that allows computers (Linux or non-Linux) with private IP addresses on your network to access the internet through a Linux computer with a public IP address providing the masquerading.

Set Up OpenVPN

1. Install OpenVPN on your server, along with the `vim` and `easy-rsa` packages using `apt` (advanced package tool).

   ```
   sudo apt-get update
   sudo apt-get install openvpn vim easy-rsa
   ```

2. Set up a certificate authority (CA) directory. First, copy the `easy-rsa` template directory into our home directory with `make-cadir`, then change to the new working directory with `cd`.

   ```
   make-cadir ~/openvpn-ca
   cd ~/openvpn-ca
   ```

3. Configure the CA variables. In order to set up the values the CA will use, we need to open and edit the `vars` file in the `vim` editor we installed.

   ```
   vim vars
   ```

   We can edit these values to our liking:

   ```
   export KEY_COUNTRY="US"
   export KEY_PROVINCE="NY"
   export KEY_CITY="NewYork"
   export KEY_ORG="SomeOrg"
   export KEY_EMAIL="myemail@email.com"
   export KEY_OU="SomeOu"
   ```

4. Build the certificate authority.

   - `source vars`
     - After typing this command, it should write: `- NOTE: If you run ./clean-all, I will be doing rm -rf on /home/jane/openvpn-ca/keys`
   - `./clean-all` — Establishes a clean environment
   - `./build-ca` — Builds the root CA

5. Create the server certificate, key, and encryption files.

   - `./build-key-server server` — Generates an OpenVPN server certificate key pair
   - Confirm with `yes`:

     ```
     Sign the certificate? [y/n]: y
     1 out of 1 certificate requests certified, commit? [y/n]: y
     ```

   - `./build-dh` — Generates Diffie-Hellman keys
   - `openvpn --genkey --secret keys/ta.key` — Generates HMAC signature to improve integrity and verification capabilities of traffic between you and the server

6. Generate a client certificate and key pair.

- `source vars` — Re-source the vars file
- `./build-key client1` — Create a client key and certificate for one user
- Optionally change the contents of the following variables: `countryName`, `stateOrProvinceName`, `localityName`, `organizationName`, `organizationalUnitName`, `commonName`, `name`, `emailAddress`
- Confirm with `yes`:

  ```
  Sign the certificate? [Y/n] y
  ```

7. Configure the OpenVPN service.

- `cd keys` — Switch to the `keys` folder
- `sudo cp ca.crt server.crt server.key ta.key dh2048 pem /etc/openvpn/` — Copies the listed files created in `~/openvpn-ca` into `/etc/openvpn`
- `gunzip -c /usr/share/doc/openvpn/examples/sample-config-files/server.conf.gz | sudo tee /etc/openvpn/server.conf` — Copies and unzips a sample OpenVPN configuration file into the configuration directory so that it can be used as a basis for setup. This is done so we don't have to write everything from the start.
- Now we can modify the configuration file:
  - `vim /etc/openvpn/server.conf` — Opens where we can configure files
  - `/tls-auth` — Searches for `tls-auth` in the file
  - The semicolon in this line needs to be removed in order to uncomment:

    ```
    ;tls-auth ta.key 0 # This file is secret
    ```
  - `key-direction 0` — Type after previous line
  - Specification of ciphers is needed:
    - The AES-128-CBC cipher offers a good level of encryption and is well supported.

      ```
      ;cipher BF-CBC
      ;cipher AES-128-CBC - Remove semicolon
      ;cipher DES-EDE3-CBC
      ```
    - `auth SHA256` — Choose secure hash algorithm 256
    - Find user and group settings:
      - `/nobody`

        ```
        ;user nobody – Remove semicolon
        ;group nogroup – Remove semicolon
        ```
    - Add an extra layer and force all connections to use the VPN route (tunnel) and then push the DNS settings to the client computers as well:
      - `/push "redirect"` — Redirect gateway

        ```
        ;push "redirect-gateway def1 bypass-dcp" - Remove semicolon
        ;push "dhcp-option DNS 208.67.222.222" - Remove semicolon
        ;push " dhcp-option DNS 208.67.220.220" - Remove semicolon
        ```
    - Protocol and port can be configured as well if needed
      - `port 1194` — Default port

        ```
        ;proto tcp
        ;proto udp – Slightly faster than tcp
        ```

8. Server networking configuration:

- First, allow IP forwarding:
  - `sudo vim /etc/sysctl.conf`
    - `#net.ipv4.ip_forward=1` — Uncomment this
  - `sudo sysctl -p` — To read the file and adjust the values for the current session
  - Adjusting the UFW rules to masquerade client connections:
    - `ip route | grep default` — Find the public network interface of our machine.
    - `sudo vim /etc/ufw/before.rules` — Opens `before.rules` and allows us to add configuration
      - This file handles configuration that should be set up before UFW rules are loaded. Here we add:

        ```
        #STARTopenvpn
        *nat
        :POSTROUTING ACCEPT [0:0]
        -A POSTROUTING -s 10.8.0.0/8 -o eth0 -j MASQUERADE
        COMMIT
        #END openvpn
        ```
      - This will set a default policy for the post-routing chain in the NAT table and masquerade any traffic coming from the VPN
      - Firewall rules should be added with caution to avoid accidentally disconnecting yourself from the firewall permanently
    - We need to tell UFW to allow forwarded packets by default. We will open the `/etc/default/ufw` file:
      - `sudo vim /etc/default/ufw`

        ```
        DEFAULT_FORWARD_POLICY="DROP" — Change "DROP" to "ACCEPT"
        ```
- Next, we have to adjust the firewall in order to enable traffic to OpenVPN.
  - `sudo ufw allow 1194/udp` — Allow the ports for the set VPN (port 1194 and protocol UDP)
  - `sudo ufw allow OpenSSH` — Allows our own traffic via SSH (otherwise we won't have access to the server)
    - Disable and reenable UFW to load changes from all modified files
      - `sudo ufw disable` — Firewall stopped and disabled on system startup
      - `sudo ufw enable` — Firewall is now active and enabled on system startup

9. Start and enable the OpenVPN service.

- `sudo systemctl start openvpn@server` — Start and enable the OpenVPN service
- `sudo systemctl status openvpn@server` — Perform a status check
  - If the following messages are displayed, then everything is in order:

    ```
    Loaded: loaded ( „specified path")
    Active: active (running)
    Initialization Sequence Completed
    ```
  - The server-side part of the VPN setup is complete.

10. Client-side configuration:

- `mkdir -p /home/jane/client-configs/files/` — One of the first things we need to do for the configuration of the client side files is create a directory
- `vim /home/jane/client-configs/base.conf` — Creates and opens up new base configuration file
  - The IP address of the server needs to be added:
    - `/1194`

      ```
      #The hostname/IP and port of the server.
      #You can have multiple remote entries
      #To load balance between the servers.
      remote ip_address 1194 — IP address and port number
      ```
    - `/nobody`

      ```
      #Downgrade privileges after initialization (non-Windows only)
      ;user nobody — Remove semicolon
      ;group nogroup — Remove semicolon
      ```
    - `/ca ca.crt`
      - Comment out these directives since we will be adding the certs and keys within the file itself:

        ```
        #ca ca.crt
        #cert client.crt
        #key client.key
        ```
    - `/cipher`
      - Mirror the cipher and auth settings that we set in the `/etc/openvpn/server.conf` file:

        ```
        cipher AES-128-CBC
        auth SHA256
        ```
      - Add the key-direction directive somewhere in the file and set it to 1:

        ```
        key-direction 1
        ```
      - Comment these out as well:

        ```
        #script-security 2
        #up /et/openvn/update-resolve-conf
        #down /etc/openvpn/update-resolv-conf
        ```
- Create a configuration generation script.
  - Create and open a file called `make_config.sh` within the `~/client-configs` directory:
    - `vim ~/client-configs/make_config`
      - These 3 lines are variables that contain paths to various items (for keys, files, and `base.conf`)

        ```
        #!/bin/bash — Bash interpreter

        KEY_DIR=~/openvpn-ca/keys
        OUTPUT_DIR=~/client-configs/files
        BASE_CONFIG=~/client-configs/base.conf
        ```
- This part does parsing. You are creating a file with the path `OUTPUT_DIR`.

```
cat ${BASE_CONFIG}
<(echo -e '<ca>')
```

```
${KEY_DIR}/ca.crt
<(echo -e '<ca>\n<cert>')
${KEY_DIR}/${1}.crt
<(echo -e '</cert>\n<key>')
${KEY_DIR}/${1}.key
<(echo -e '</key>\n<tls-auth>')
${KEY_DIR}/ta.key
<(echo -e '</tls-auth>')
>${OUTPUT_DIR}/${1}.ovpn
```

- Mark the file as executable by typing:
    - `chmod 700 make_config.sh`

11. Generate client configurations.

- `./make_config-sh client1`
- If everything went well, we should have a `client1.ovpn` file in our `~/client-configs/files` directory.
    - `ll ~/client-configs/files/` — Confirm
- `scp -r root@ip_address:/home/jane/client-configs/files/client1.ovpn`

12. Install the client configuration.

- Check to see if we have updated the resolve script.
    - `ls /etc/openvpn/`
- `vim client1.ovpn` — Edit client configuration.
    - Go into Insert mode:

```
#script-security 2
#up /etc/openvpn/update-resolv-conf
#down /etc/openvpn/update.resolve-conf
```

- `openvpn --config client1-ovpn` — Connect
    - If it says 'Initialization sequence completed', everything is in order.
- `sudo tail -f /var/log/syslog` — Take a look at the log files to confirm the connection.

At this point we can run any application from the system and it will get routed through the VPN. Make sure you run it with the user with elevated privileges.

Useful Commands

- `ssh` — A program for logging in to and executing commands on a remote machine. It provides encrypted communications between two hosts across an insecure network.
- `mkdir` — Creates a directory or directories if they do not already exist.
- `usermod` — Modifies the system account files to be in correspondence with the changes that are specified on the command line.
- `su` — Allows commands to be run with a substitute user and group ID. When called without arguments, `su` defaults to running an interactive shell as root.

- `sudo` — Allows a permitted user to execute a command as the superuser or another user.
- `make` — Determines which pieces of a program have to be recompiled and sends out the commands to recompile them.
- `cd` — Changes the working directory of the current shell execution environment.
- `vim` — Text editor that can be used to edit different kinds of plain text. Useful for editing programs.
- `pwd` — Prints the name of the working directory.
- `openvpn` — Secure IP tunnel daemon.
- `gunzip` — Compresses or expands files.
- `ip` — Shows or manipulates routing, network devices, interfaces, and tunnels.
- `chmod` — Changes the file mode bits of each given file according to mode, which can be either a symbolic representation of changes to make, or an octal number representing the bit pattern for the new mode bits.
- `ls` — Lists information about the files (in the current directory by default).
- `rm` — Removes files or directories.
- `scp` — Copies files between hosts on a network. It uses SSH for data transfer and uses the same authentication and provides the same security as SSH.
- `apt` — APT (Advanced Package Tool) is a management system for software packages.

Tor

Key Terms

- **Tor**: A network that provides anonymity and protects user privacy by routing traffic via a series of servers.
- **Tor Browser**: A web browser that implements the Tor network.
- **.onion**: A top-level domain (TLD) like .com and .org. A .onion domain consists of a hash of a public key and a .onion domain name. They are usually accessed only through a direct link.
- **Dynamic Chain**: In a dynamic chain, each connection is achieved through chained proxies, and they are chained in the order they are listed. At least one has to be online.
- **Strict Chain**: In a strict chain, each connection is achieved through chained proxies, and they are chained in the order they are listed. Unlike dynamic chains, all proxies must be online.
- **Bridge**: Also called bridge relays, bridges are Tor relays that are not listed in the main Tor relay directory, and are thus an alternate way of accessing the Tor network. They also make it harder for your ISP to know whether or not you are using Tor. Using bridges is not recommended unless you are in a country that censors Tor.
- **Hidden Wiki**: A website that contains direct links to hidden Tor-hosted sites. To get to Hidden Wiki, you have to Google search for the latest available URL, since the link changes often and you need a direct link.

How Tor Works

Tor passes data packets through a series of nodes. That path consists of three types of nodes: an entry node, a middle node, and an exit node. First, the traffic is encrypted and sent to the entry node. Next, layers of the data packet are stripped off each time it passes through one of the middle nodes. Finally, the exit node uses an unencrypted link to communicate with the target server outside the Tor network.

Each node can only access the node before it and the one after it. The more nodes, the greater the anonymity and the less likely a DDoS attack will happen. If Tor is censored in your country, there is a way to bypass the censorship. Go to **Configurations** after the Tor Browser is launched. Check the box next to **Tor is censored in my country**, pick a bridge, and click **Connect**.

Common reasons for using Tor include hiding your IP address and online activities and bypassing government censorship and internet restrictions in your location. To make the network faster, safer, more stable, and more resistant to attacks, you can run a Tor relay. Another safety measure is to create a new user before using the Tor Browser. Running the browser as `root` (a superuser) is not secure, and in the event of an attack, any harm that comes to the system will be worse.

Useful Commands

- `apt-cache search proxychains` – Searches for available proxychains.
- `systemctl status tor` – `systemctl` is used to control the systemd system and service manager. `status tor` starts the Tor network service.
- `systemctl stop tor` – `systemctl` is used to control the systemd system and service manager. `stop tor` stops the Tor network service.
- `vim /etc/proxychains.conf` – Opens the ProxyChains configuration file.
- `:q!` – Force quits without saving.
- `apt-get install tor` – Installs the Tor network service on Linux.
- `apt-get install torbrowser-launcher`– Installs the Tor Browser launcher on Linux.
- `adduser <USERNAME>` – Adds a user with the specified username.

Proxies

Key Terms

- **Proxy server**: A server placed between a client and a remote server such that all client requests go through the proxy to the remote server. There is no direct communication between the client and the remote server. All communication goes through the proxy server.
- **Forward Proxy**: A forward proxy forwards client requests to the remote server in order to establish the connection between the client and the server.
- **Open Proxy**: A type of forwarding proxy used for hiding the user's IP address. There are three different types of open proxies: anonymous, distorting, and elite. Each of these hides the user's IP address differently.
- **Anonymous Open Proxy**: Hides the IP address but reveals that a proxy is being used.
- **Distorting Open Proxy**: Provides a fake IP address but reveals that a proxy is being used.

- **Elite Open Proxy**: Hides both the IP address and the fact that a proxy is being used. Elite open proxies provide the highest level of anonymity.
- **Reverse Proxy**: A reverse proxy looks just like a regular server to the client. The client, thinking it is connected to the server, sends a request for resources. The reverse proxy then forwards the request to the target server and sends the response back to the client.
- **SSH**: SSH (Secure Shell) is a protocol used to make a safe connection to a remote server.
- **SSH Keys**: Keys used as access credentials in the SSH secure protocol, usually with 2048-bit RSA encryption.

How Proxy Servers Work

Like VPNs, with proxy servers, the client and server do not communicate with each other directly. Instead, they only connect to the proxy server. The client sends requests for resources that are located on a remote server, and the proxy server responds to requests by retrieving those resources from the remote server and sending them to the client.

Set Up a Proxy

This tutorial demonstrates how to set up a proxy on a cloud service, but these steps are not limited to a cloud setup. You can also set up a proxy on any device or machine running Linux. As we discussed in the video lesson, one example would be to use a device like an Arduino as a proxy and physically connect it to a network in a building. Then, all traffic that is sent through the network would also pass through the proxy. This is a useful technique for hackers targeting low-security locations in an attempt to gather information on a well-protected target.

To set up an SSH proxy tunnel, you need to generate an RSA-encrypted SSH key. Generate the key in the Linux console by typing `ssh-keygen` and then copy the key to the clipboard with the `cat` command. Once the key is generated, you can connect it to an address by typing `ssh`, followed by the address you want it linked to. When choosing which port to use with the `ssh -N -D <PORT NUMBER>` command, make sure the port number is not being used by anything else.

Manual Proxy Configuration

To manually set up a proxy on your browser, go to *Preferences* > *Advanced* > *Network* > *Connection Settings*. Then, select **Manual proxy configuration**, clear the *HTTP*, *SSL*, and *FTP Proxy* fields, and at the *SOCKS Host* fields, type in the address and the number of the port you chose to use when you created the SSH key.

Security Measures

To ensure your browser does not betray your identity, you can change your browser's Accept-Language information. To edit these settings, type `about:config` in the browser URL search bar. Search for the `accept-languages` reference name, and change the language to one of the countries your proxy IP shows.

When creating a proxy, make sure to use a new, created user, not `root`. Also, make sure you do not leave any other traces of your personal data.

Useful Commands

- `ssh-keygen` – Generates a new SSH key.
- `cd /root/.ssh` – Changes the directory to `/root/.ssh`.
- `rm` – Used to remove files or directories.

- `rm -rf known_host` – Removes all content from the known host file where host keys are stored.
- `ssh root@<IP>` – Makes a connection to a server of the declared IP address as the provided username (in this case, `root`).
- `yum install vim` – Linux command to install vim (advanced version of vi, a text editor that can be used to edit different kinds of plain text).
- `vim /etc/ssh/sshd_config` – A system-wide configuration for OpenSSH that enables the modification of the SSH daemon.
- `systemctl restart sshd` – Restarts the SSH daemon (`sshd` is used to provide a secure, encrypted communication channel over an insecure network).
- `ssh -N -D <PORT NUMBER> <USER>` – Forwards the secure connection to the declared port number. `-N` disables any commands coming in, `-D` forwards which port the key should connect to.
- `ip addr show | grep inet` – Displays the IP address.

DoS and DDoS

Objectives

The purpose of this section is to explain what DoS and DDoS attacks are and describe the techniques used to carry them out. We will go over several different types of DoS and DDoS attacks: UDP flood attacks, SYN flood attacks, Slowloris attacks, and distributed reflection DoS (DRDoS) attacks. We will also talk about three real-life examples of DoS/DDoS attacks.

Key Terms

DoS (Denial of Service)

In a DoS attack, a target system is overloaded with a huge number of false requests, rendering the system useless and preventing its users from accessing the resources they need.

DDoS (Distributed Denial of Service)

In a DDoS attack, a botnet carries out a DoS attack on the target system. To perform a DDoS attack, the attacker first compromises as many systems as possible and then uses those systems (the botnet) to launch a DoS attack on the target.

Attack Techniques

UDP Flood Attack

In a UDP flood attack, the target server's random ports are flooded with a huge number of forged UDP packets, causing the server to continually check for applications on the ports. Because there are no applications associated with the recovered packets, the server responds with an ICMP Destination Unreachable packet. Having to reply to such a huge number of requests eventually renders the system unable to respond to requests coming from legitimate applications.

SYN Flood Attack

A SYN flood attack exploits a flaw in the TCP three-way handshake by sending a huge number of SYN requests with fake source IPs. The target responds with a SYN ACK packet and waits for the sender to send back an ACK packet. However, because the source IP is invalid, the ACK packet never arrives, and the connection remains incomplete. With so many incomplete connections, the server is unable to reply to valid requests.

Slowloris Attack

A Slowloris attack floods the target server with a large number of incomplete HTTP requests. For each received request, the server opens a connection and waits for the request to complete; however, it never does. With so many open connections, the server is eventually unable to receive new legitimate connections.

DRDoS Attack

A DRDoS attack uses two sets of computers to perform an attack on a target. The first set of computers consists of compromised systems, while the second set are uncompromised. The intermediary machines send a large number of packets with spoofed IP addresses to the secondary machines. The secondary machines reply to the IP address that is specified as the source, which is the target's IP address. With so many secondary machines sending so many packets, the target eventually becomes overwhelmed and is unable to function properly.

Famous DoS and DDoS Attacks

The following are three of the biggest and most famous DoS and DDoS attacks ever carried out:

GitHub Attack

In 2018, GitHub was hit with an amplification attack that targeted servers running a distributed memory object caching system called Memcached. This attack did not use botnets. Instead, it targeted public-facing Memcached servers. Memcached server replies are extremely large in size (50x larger than the requests they receive). In this attack, the attackers sent small requests with spoofed GitHub IP addresses to the Memcached servers. The Memcached servers replied with enormous packets, overwhelming GitHub's servers with traffic. This attack is the largest DDoS attack on record, recording traffic of 1.35 Tbps.

Dyn Attack

In 2016, DNS provider Dyn was the target of a major DDoS attack. The attackers used the malware Mirai to create a botnet that directed traffic at Dyn. The attackers first infected a large number of devices and then used the bandwidth from the infected devices' networks to transmit as many packets as possible to the targeted server. Because of this attack, many sites experienced a disruption of service, including Amazon, PayPal, Visa, and Netflix. At the time, it was the biggest DDoS attack on record. However, after the 2018 attack on GitHub, it now holds second place, with recorded traffic of 1.2 Tbps.

Mirai Botnet Attack

In 2016, shortly before the attacks on Dyn, the Mirai malware came to attention. Mirai is a self-propagating virus that compromises unsecured Internet of Things (IoT) devices. After infecting such devices, Mirai turns them into remote-controlled bots and uses them to coordinate large-scale DDoS attacks. The first major Mirai botnet attack was on the French telecommunications company OVH. The second major attack happened a few days later when the security journalism website “Krebs on Security” was attacked. During these attacks, recorded traffic was around 620 Gbps, which was almost twice the size of the previous record-holding attack.

Cracking Wireless

Objectives

The purpose of this module is to introduce you to the different attack methods used for password cracking, as well as to guide you through the process of cracking a wireless access point (WAP) password using a practical example. We will start by installing all of the necessary software and familiarizing ourselves with the relevant terminology before moving on to the process of cracking a wireless password.

Password Basics

A **password** is a sequence of characters used to protect information.

Password cracking is the process of attempting to ascertain the correct sequence of characters in order to gain access to whatever the password is protecting.

The strength of a password depends on its:

- **Length**: The number of characters in the sequence
- **Complexity**: The combination of letters, digits, and special characters used in the sequence
- **Predictability**: How easily an attacker could guess the password

The formula for determining the number of possible character combinations is: **BaseLength**

Password	Base	Length	Combinations
password	a-z (26)	8	208,827,064,576
Password	a-z, A-Z (52)	8	53,459,728,531,456
Passw0rd	a-z, A-Z, 0-9 (62)	8	218,340,105,584,896
P@ssw0rd	a-z, A-Z, 0-9, @ (94)	8	6,095,689,385,410,816
P@s5	a-z, A-Z, 0-9, @ (94)	4	78,074,896

Password Cracking Attacks

Brute Force Attacks

In a brute force attack, the attacker runs every possible combination of characters until the password is cracked. This is a *very* time-consuming process.

Dictionary Attacks

Dictionary attacks involve running a list of commonly used passwords against user accounts to check for matches. The attacker loads a dictionary file of common passwords into a password cracking program, which then checks the password list against user accounts.

Rainbow Table Attacks

A rainbow table is a list of plaintext passwords and their corresponding hashes. In a rainbow table attack, the attacker compares a hashed password with the hashes in the rainbow table. When there is a match, the attacker is able to determine the original password.

Rule-Based Attacks

A rule-based attack is used when the attacker has some information about the password, such as the length, whether or not the password contains digits, etc. In this attack, the attacker combines multiple techniques — such as brute-force, dictionary, and rainbow table attacks — to crack the password. This is the most effective approach.

Capturing a Handshake and DoS-ing Individual Clients

1. Install Aircrack-ng.

2. Enable monitor mode and establish the name of the interface:

   ```
   airmon-ng start <INTERFACE NAME>
   ```

3. It's a good idea to kill certain processes that might interfere with the cracking process. Start by killing the network manager so that it can't reactivate the processes:

   ```
   kill <PID NUMBER>
   ```

4. Scan and list the available wireless access points and associated clients:

   ```
   airodump-ng <INTERFACE NAME>
   ```

The output of this command will list the following wireless access point features:

- BSSID: Used to identify a specific BSS (basic service set) within an area.
- PWR: Reported signal strength. In a sense, it represents the relative distance between you and the wireless access point. The lower the number, the better. Note that you cannot always judge the distance of a point based on reported signal strength.
- CH: Tells you which channel all the wireless devices are running on.
- ENC: Tells you if the access points require authentication and if so, what kind.
- ESSID: Tells you the name of the access points.

5. Now we can attempt to capture a handshake between an access point and a client using the following command:

   ```
   airodump-ng -c <CHANNEL> --bssid <ACCESS POINT MAC ADDRESS> \
   -w capture <INTERFACE>
   ```

6. Next, we deauthenticate the client using the following command:

```
airplay-ng --deauth <NUMBER OF DEAUTH REQUESTS (`0` FOR INFINITE)> \
-a <WIRELESS ACCESS POINT> \
-c <ASSOCIATED CLIENT MAC ADDRESS> <INTERFACE>
```

7. At this point, the handshake is established. Now we just have to wait for the client to reconnect to the network in order to capture the handshake.

Useful Commands

- `airmon-ng` – Part of the `aircrack-ng` package that is used to enable and disable monitor mode on wireless network interfaces.
- `airmon-ng start` – Enables monitor mode.
- `airmon-ng stop` – Disables monitor mode.
- `airodump-ng` – Goes through available wireless channels and conducts scans on available wireless channels.
- `aireplay-ng` – Part of the `aircrack-ng` package that is used to insert wireless frames. It includes several attacks that can deauthenticate wireless clients.

Cracking Wireless with Hashcat and Masks

Download and Set Up Hashcat

1. Download the hashcat files from the hashcat website:

```
wget http://hashcat.net/files/hashcat-5.0.0.7z
```

2. Install the 7-Zip utilities.

```
sudo apt-get install p7zip
```

3. Unpack the zip file.

```
7zr x hashcat-5.0.0.7z
```

4. Enter the extracted folder.

```
cd hashcat-5.0.0
```

5. List and view the folder contents.

```
ls
```

6. What we need is the `hashcat64.bin`. Locate this file and use the `help` command to list useful information:

```
./hashcat64.bin --help
```

7. At the bottom, we'll see a list of sample attacks and their commands:

Attack Mode	Hash Type	Example Command
Wordlist	SPS	`hashcat -a 0 -m 400 example400.has example.dict`
Wordlist + Rules	MD5	`hashcat -a 0 -m 0 example.hash example.dict -r rules/best64.rule`
Brute Force	MD5	`hashcat -a 3 -m 0 example0.hash ?a?a?a?a?a?a`
Combinator	MD5	`hashcat -a 1 -m 0 example0.hash example.dict example.dict`

Creating a Mask

There are two ways we can create masks: * Directly in the command line * Within a file

1. Use the following command to create a mask in the command line:

   ```
   hashcat64.bin -m 2500 -a 3 ../capture.hccapx ?u?l?l?l
   ```

2. Use the`vim myMasks` command to open and create a file which we can populate with password masks. Then enter the following command:

   ```
   hashcat64.bin -m 2500 -a 3 ../capture.hccapx myMasks
   ```

`2500` is the number of the hash we are cracking.

`3` defines the attack mode as brute force.

Here is what the placeholder symbols (character sets) in the password mask mean:

?	Charset
l	abcdefghijklmnopqrstuvwxyz
u	ABCDEFGHIJKLMNOPQRSTUVWXYZ
d	0123456789
h	0123456789abcdef
H	0123456789ABCDEF
s	!"#$%&'()*+,-./:;<=>?@[]^_'{
a	?l?u?d?s
b	0x00 - 0xff

Hashcat preserves the results of its cracks, and we can view them in this file:

```
vim hashcat.potfile
```

Key Terms

- **Monitor Mode**: Enables devices that have wireless network interface controllers to monitor network traffic.
- **Aircrack-ng**: Aircrack-ng is a set of tools used to evaluate Wi-Fi network security. It focuses on several areas of Wi-Fi security, including monitoring, attacking, testing, and cracking.

- **Handshake**: A handshake is an exchange of predetermined signals between a computer and a peripheral device or another computer. This exchange is made when a connection is initially established or at intervals during data transmission in order to assure proper synchronization.
- **Password Guessing**: Password guessing involves manually attempting to log in to the target's machine by guessing the password using information known about the target.
- **Hashcat**: Hashcat is a password recovery tool.
- **Masks**: A mask is a string that contains a specific number of placeholder characters, where each character represents a set of characters that can exist in each placeholder. It is used to narrow down the number of possible combinations when cracking a password.

Reverse Engineering

Objectives

The Reverse Engineering module contains several sections dedicated to explaining the basics of reverse engineering, both theoretical and practical. It starts by familiarizing you with what reverse engineering is, why it is needed, what kinds of tools are needed to perform it, and the required steps for reverse engineering a program.

Introduction to Reverse Engineering

Reverse engineering is the process of taking a program apart and discovering how it works in order to replicate it or identify and exploit its vulnerabilities. It involves recreating the program's source code from its binary code. Reverse engineering serves several purposes. It is used for fixing bugs, improving performance, recreating the original source code if it is lost or inaccessible, understanding how a specific portion of a code works, and analyzing malware.

Steps for Reverse Engineering a Program

1. Define the objective.
    - First, we have to figure out *why* we are reverse engineering something. Let's say we have a web server and we don't know the source code of the server. We are trying to reverse engineer a portion of it where there is, for example, a socket through which network communication flows. We would like to understand how it works. Once we get that information, we can use it to cause the behavior we want (e.g., a buffer overflow). It's important to define our objective so we know what to look for and do.
2. Observe the process.
    - In this step, we observe what is happening and try to get an overview of the situation.
3. Disassemble the program.
    - The next step is to run the program through a disassembler.
4. Analyze the disassembled parts.
    - From the disassembled parts, we can extract all of the information we need to perform our objective.

Compiling, Running, and Configuring a Keylogger

In this part of the module, we will go through the process of compiling, running, and configuring a keylogger, which we'll treat as surveillance-oriented malware. We'll go through this whole process so that we can learn the reverse procedure later. Our goal is to edit the keylogger signature, set up the keylogger, and learn how to read the output the decrypt file gives us. We will also go through the process of locating a keylogger on our own machine, isolating it, and finding information about the attacker.

Getting Started

It is important to alter the keylogger signature because it decreases the likelihood of antivirus software matching a known type of malware. The more times you change it, the better. If you change it enough, the hashes shouldn't match. If you only make a few small changes, it might be matched, due to multiple variations of signatures that already exist.

Decrypt functions need to be the exact inverse of an encrypt function. Whatever was done in an encrypt function must be undone in a decrypt function.

After you have a working decrypt file on your computer, you can easily perform a decryption in a command line shell:

- For Windows Powershell, it would look something like this:

  ```
  cd .\Desktop\decrypt\bin\Debug\
  .\decrypt.exe .\log
  ```

What to look for when trying to detect a suspicious task in your task manager:

- The malicious file will probably be named like another file that is stored on your computer, disguised as a Runtime Broker process or some other program. It is good to check your running processes, find some you are not sure about, and try to determine why that process would be running on your computer.
- You can check the details and see the Username, PID, Status, Memory, and Description of the process to see if anything seems suspicious.
- You can check the properties to see general information like the location of the file and when it was created. In the details, you'll find a more detailed description that tells you the type, product name, product version, copyright, size, and date modified. You can compare the data of two processes that have the same name and see if one looks more reliable than the other.
- We can also check for digital signatures in the file properties.
- You can open up the file location and see if the location is suspicious. For example, why would something called `Notepad` run from the `System32` folder?
- Compare the file hashes of two processes. Sometimes the difference between the two will be because of the version, but sometimes it might signify that something is wrong.

Decompiling a Keylogger

Process Overview

To perform reverse engineering, we need a few tools. IDA is the best tool for reverse engineering. It uses the Hex-Rays decompiler, and while the free version is restricted, it is great for beginners.

There is another decompiler called Snowman that we can also use. It is always a good idea to use more than one decompiler so you can compare the results of both and get a better idea of what is going on. The fastest way to decompile a program is to select and run individual segments of the program in the decompiler and then analyze them.

We need IDA 32-bit because the keylogger is 32-bit. The keylogger is 32-bit because we want it to work on as many systems as possible.

Once we insert the file into IDA, we will see the decompiled program written in assembly language. Some assembly instructions for the x86 Intel instructions we might see include: `lea`, `and`, `or`, `push`, `mov`, `call`, and `sub`.

Tip: Look up the x86 Intel instruction set on the Intel official page. They have very detailed documentation of every instruction and what it does!

Apart from viewing it in assembly language (which is correct, legitimate code), we can view the code in a more readable, yet less trustworthy auto-generated pseudocode format. This view is where we will try to figure out the code and make more sense of it.

The code will seem confusing at first. There will be a lot of compiler-added code because of the way the compiler processes the code. Functions are also tricky, since the first value that seems to be the first parameter of a function is actually the return value.

When decrypting, the code is read from bottom to top, since it is the opposite process of encrypting!

If we recognize some general purpose algorithms like `Base64`, or any other algorithms that are already known, you don't have to reverse engineer them, but rather just write them out at any point it is needed.

Most of the process now is trying to figure out the code. We start by finding variable declarations and renaming the variables to names that are more logical and clear. Next, we move on to the functionalities, which are harder to detect.

While figuring out the program's pseudocode, it is important to write your own pseudocode and try to form the structure of the function.

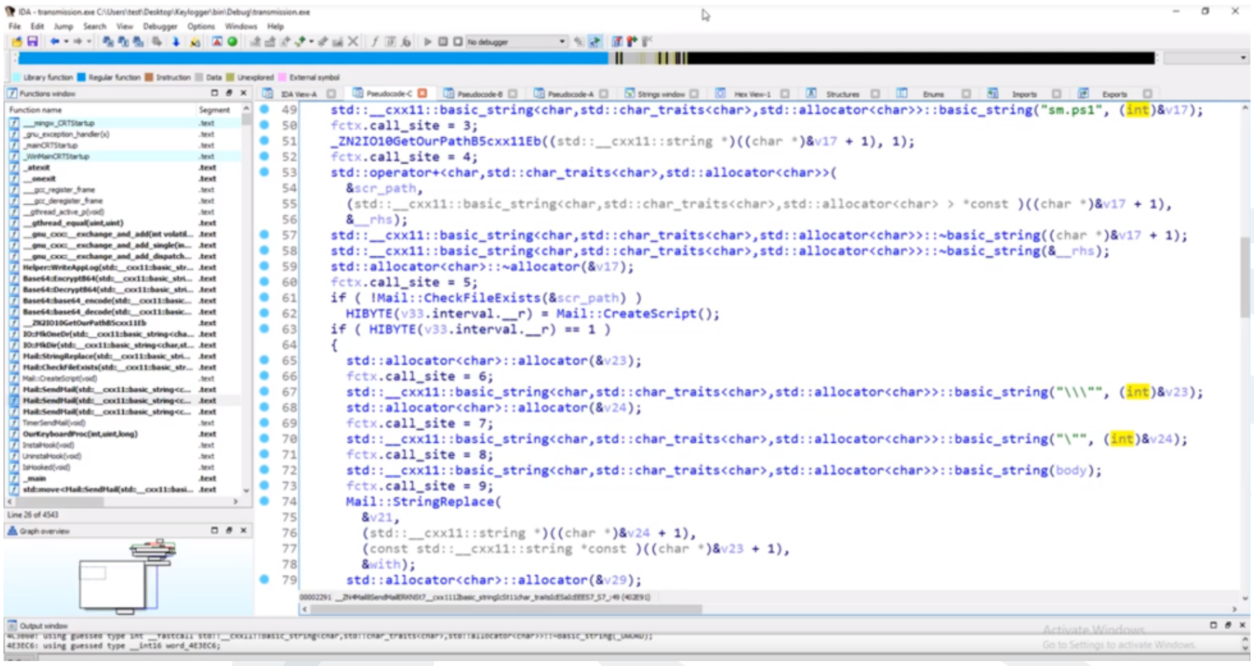
Reference points in the code can help you with orientation and chunking up the program into smaller pieces.

Note that your reversed code will probably never be exactly the same as the pre-compiled (original) code, but it will be close enough to perform all of the necessary functionalities.

After completing the pseudocode, the next step is to write the decryption function in a high-level language and check to see if it works properly.

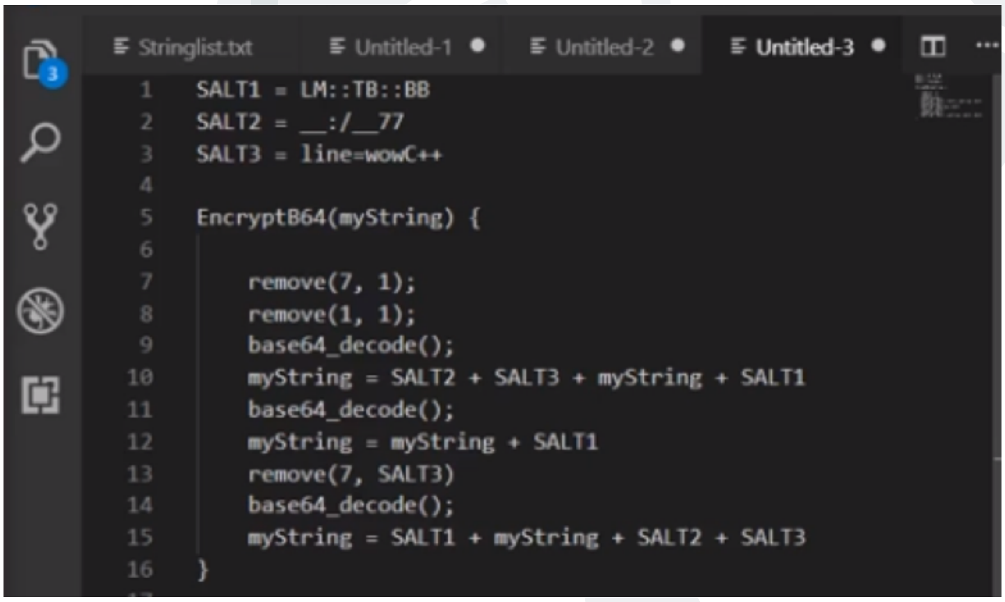
Lastly, the files can be decrypted through a command line Powershell.

We go from computer-generated code:



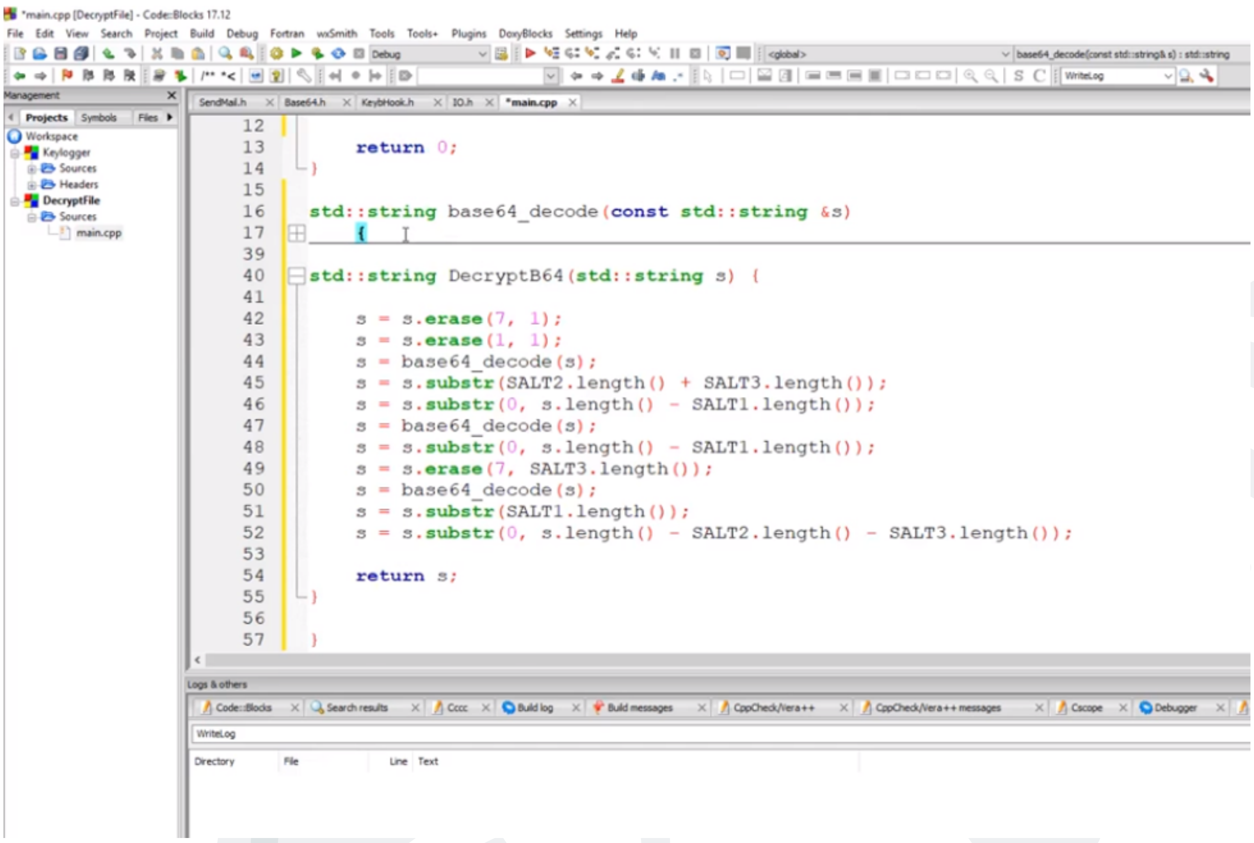
```
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string("sm.ps1", (int)&v17);
fctx.call_site = 3;
_ZN2IO10GetOurPathB5cxx11Eb((std::__cxx11::string *)((char *)&v17 + 1), 1);
fctx.call_site = 4;
std::operator+<char,std::char_traits<char>,std::allocator<char>>(
  &scr_path,
  (std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char> > *const )((char *)&v17 + 1),
  &__rhs);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string((char *)&v17 + 1);
std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::~basic_string(&__rhs);
std::allocator<char>::~allocator(&v17);
fctx.call_site = 5;
if ( !Mail::CheckFileExists(&scr_path) )
  HIBYTE(v33.interval.__r) = Mail::CreateScript();
if ( HIBYTE(v33.interval.__r) == 1 )
{
  std::allocator<char>::allocator(&v23);
  fctx.call_site = 6;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string("\\\"", (int)&v23);
  std::allocator<char>::allocator(&v24);
  fctx.call_site = 7;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string("\"", (int)&v24);
  fctx.call_site = 8;
  std::__cxx11::basic_string<char,std::char_traits<char>,std::allocator<char>>::basic_string(body);
  fctx.call_site = 9;
  Mail::StringReplace(
    &v21,
    (std::__cxx11::string *)((char *)&v24 + 1),
    (const std::__cxx11::string *const )((char *)&v23 + 1),
    &with);
  std::allocator<char>::allocator(&v29);
```

To creating our own pseudocode:



```
SALT1 = LM::TB::BB
SALT2 = __:/__77
SALT3 = line=wowC++

EncryptB64(myString) {

    remove(7, 1);
    remove(1, 1);
    base64_decode();
    myString = SALT2 + SALT3 + myString + SALT1
    base64_decode();
    myString = myString + SALT1
    remove(7, SALT3)
    base64_decode();
    myString = SALT1 + myString + SALT2 + SALT3
}
```

Then, finally an implemented version in CodeBlocks:


*main.cpp [DecryptFile] - Code::Blocks 17.12
File Edit View Search Project Build Debug Fortran wxSmith Tools Tools+ Plugins DoxyBlocks Settings Help
Workspace
Keylogger
Sources
Headers
DecryptFile
Sources
main.cpp
SendMail.h Base64.h KeybHook.h IO.h *main.cpp
Logs & others
Code::Blocks Search results Cccc Build log Build messages CppCheck/Vera++ CppCheck/Vera++ messages Cscope Debugger
WriteLog
Directory File Line Text

```
    return 0;
}

std::string base64_decode(const std::string &s)
{

std::string DecryptB64(std::string s) {

    s = s.erase(7, 1);
    s = s.erase(1, 1);
    s = base64_decode(s);
    s = s.substr(SALT2.length() + SALT3.length());
    s = s.substr(0, s.length() - SALT1.length());
    s = base64_decode(s);
    s = s.substr(0, s.length() - SALT1.length());
    s = s.erase(7, SALT3.length());
    s = base64_decode(s);
    s = s.substr(SALT1.length());
    s = s.substr(0, s.length() - SALT2.length() - SALT3.length());

    return s;
}

}
```

Buffer Overflow

Buffer overflow occurs when more data than the allocated buffer can store is written to the buffer, causing the extra bytes to overflow. Attackers can exploit this vulnerability to write and execute malicious instructions to a system's memory.

How Buffer Overflow Works

A program that requires user input has to reserve a certain number of bytes in memory in order to process that data. If the input is not validated and exceeds the allocated buffer size, the program will simply continue to write the extra bytes to memory, even if it means overwriting data stored in adjacent memory locations.

How Attackers Exploit Buffer Overflow

Attackers can exploit a buffer overflow vulnerability to inject malicious code into a program and its memory. If a program is vulnerable to buffer overflow, attackers can attempt to inject malicious code, jump to previously inserted malicious code, or write malicious code (shellcode).

Injecting Malicious Code

Attackers can exploit a buffer overflow vulnerability by injecting binary code into the memory space reserved for the buffer. The malicious instructions are read and executed, compromising the program and system.

Jumping to Previously Inserted Malicious Code

If there is already a piece of malicious code in memory, the attacker can inject a jump instruction to its address, executing the malicious code and overflowing the buffer.

Writing Malicious Code

Shellcode is a type of malicious code written to memory. Shellcode typically launches a command shell from which the attacker can run any command they want.

Famous Buffer Overflow Attacks

SQL Slammer (2003)

SQL Slammer is a computer worm that caused a DoS attack on multiple internet hosts, slowing down the internet traffic by a considerable amount. A huge number of UDP packets arrived rapidly at UDP ports. The worm exploited a buffer overflow weakness to achieve this.

Conficker (2008)

Conficker is a computer worm that targets the Windows operating system. It uses a combination of advanced malware techniques, making it harder to stop. Most versions of this worm rely on the exploitation of a vulnerable computer that wasn't patched for the MS 08-067 vulnerability. Conficker sends a remote procedure call request to force a buffer overflow and execute shell code on the victim's machine. Once that shell code runs, it joins a larger botnet and is ready to receive its payload from a daily list of 250 domain names. These payloads update the worm from a new variant and have the ability to install additional malware like keyloggers, rootkits, and more.

Key Terms

- **Reverse Engineering**: The process of taking a program apart and discovering how it works in order to replicate it or identify and exploit its vulnerabilities.
- **Hex Dumper**: Software that allows you to view the contents of a file in hexadecimal format rather than in binary.
- **Disassembler**: Software that converts machine language into assembly language.
- **Debugger**: Software that searches for and corrects any errors (bugs) in other programs.
- **IDE**: An IDE (Integrated Development Environment) is an application used to write and compile code.
- **Keylogging**: Keystroke logging (keylogging) is a software surveillance technique that records every keystroke made on the target system.
- **Payload**: The segment of a computer virus that executes malicious activities, or any malicious code that is deployed on the target device.
- **Hash**: A number that has been derived from a text string. Once the string has been encoded, it is much harder to decrypt than regular encryption.
- **Signature**: In computing, a signature is a hash or algorithm that is used to uniquely identify a specific virus or malware. Also called a DAT file or Definition file.
- **Encryption**: The process of altering sensitive information to make it unreadable to everyone except the intended recipient.
- **Decryption**: The process of decoding encrypted data and returning it to a legible form. The decryption process is identical to the data encryption process, but executed in reverse order.
- **Buffer**: A buffer is an area of memory where data is temporarily stored. When programs require user input, that input is stored temporarily in memory before it is processed. To store the input, a portion of memory of fixed size—the buffer for that input—is allocated to the program.

- **Worm**: A worm is a type of malware that replicates itself in programs and documents on a victim's machine by exploiting vulnerabilities on the machine and spreading to other computers as the infected files are transferred.
- **Cheat Engine**: An open-source memory scanner, hex editor, and debugger. It can be used to accomplish many tasks, including browsing through a program's RAM space.

Metasploit

Objectives

The purpose of this module is to teach you the basics of the Metasploit Framework. We'll start by learning what Metasploit is and what it is used for before going over the basic commands you should know before using the tool. Next, we'll demonstrate how to use the framework to exploit a target and learn what to do post-exploitation. Finally, we will learn how to write a Metasploit module.

The Metasploit Framework

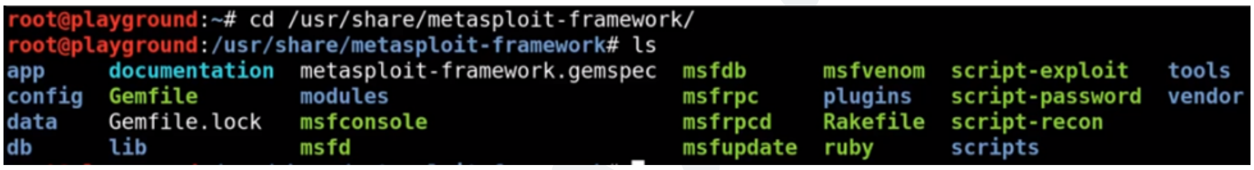
Metasploit is an open-source penetration testing framework that can be used for many kinds of cybersecurity work. It provides an abundance of information, options, and built-in functionalities that automate processes.

The Metasploit Framework is accessed through the msfconsole interface. You can start the console by entering `msfconsole` in your command line prompt.

Before we can use Metasploit, we first need to get familiar with its structure. The framework has a huge number of directories that we can access and view. Use the `help` command any time you need help navigating through the framework.

The top-most directory of the framework can be accessed by typing:

```
cd /usr/share/metasploit-framework/
```



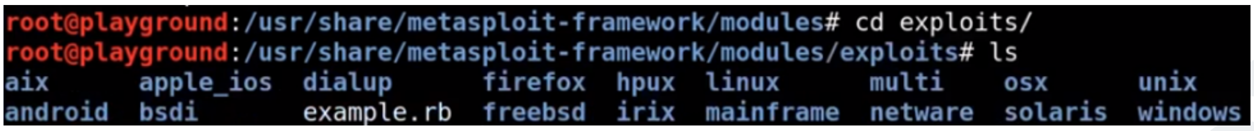
```
root@playground:~# cd /usr/share/metasploit-framework/
root@playground:/usr/share/metasploit-framework# ls
app     documentation  metasploit-framework.gemspec  msfdb      msfvenom  script-exploit   tools
config  Gemfile        modules                       msfrpc     plugins   script-password  vendor
data    Gemfile.lock   msfconsole                    msfrpcd    Rakefile  script-recon
db      lib            msfd                          msfupdate  ruby      scripts
```

Let's familiarize ourselves with a few of these directories.

- The `data` directory contains executable binary files or paths to binary files that are needed by certain exploits.
- The `documentation` directory provides all of the available documentation on the framework.
- The `plugins` directory contains a list of plugins you can add to your project.
- The `lib` directory contains the codebase of the entire framework.
- The `modules` directory contains a variety of built-in modules you can use.

Metasploit modules are sorted into six categories: auxiliary, encoders, exploits, nops, payloads, and ports.

If we enter the `exploit` directory, we'll see several directories containing exploits for different systems and technologies:



```
root@playground:/usr/share/metasploit-framework/modules# cd exploits/
root@playground:/usr/share/metasploit-framework/modules/exploits# ls
aix      apple_ios  dialup      firefox  hpux  linux      multi    osx      unix
android  bsdi       example.rb  freebsd  irix  mainframe  netware  solaris  windows
```

Take some time to browse through all these folders and familiarize yourself with the framework structure.

Database Setup

Follow these steps to set up your project database.

1. Start the database.

   ```
   systemlctl start postgresql
   ```

2. Initialize the database.

   ```
   sudo msfdb init
   ```

3. Start the Metasploit Framework console.

   ```
   msfconsole
   ```

4. Check the status of the database.

   ```
   db_status
   ```

The output of this command should print: `postgresql connected to msf`.

Basic Metasploit Commands

- `banner` – Displays a random Metasploit picture banner
- `?` or `help` – Opens the Help menu
- `cd` – Changes the current directory
- `color` – Toggles whether or not the console output displays color
- `connect` – Allows you to connect to a remote host
- `exit` – Exits the console
- `get` – Gets the value of a context-specific variable

- `getg` — Gets the value of a global variable
- `grep` — Finds a matching string from the input and prints the line containing that string
- `history` — Displays the command history
- `irb` — Switches you to `irb` scripting mode
- `load` — Loads a framework plugin from the plugin directory
- `quit` — Exits the console
- `route` — Routes traffic through a session
- `save` — Saves active datastores
- `session` — Displays information about sessions and dumps session listings
- `set` — Sets the value of a content-specific variable
- `setg` — Sets the value of a global variable
- `sleep` — Tells the console to do nothing for a number of input seconds
- `spool` — Writes the console output to both the screen and a file
- `threads` — Used to manipulate and view background threads
- `unload` — Unloads a framework plugin
- `unset` — Unsets one or more content-specific plugins
- `unsetg` — Unsets one or more global variables
- `version` — Shows the library version numbers of the console and framework

Key Terms

- **Metasploit**: Metasploit is an open-source penetration testing framework that can be used for many kinds of cybersecurity work. It provides an abundance of information and options that automates processes.
- **Exploit**: An exploit is an attack that takes advantage of a vulnerability to gain access to and cause unpredictable behavior on the target computer system.
- **Backdoor**: A backdoor is a secret way of accessing a system that the system owner or user is unaware of. Attackers can use a backdoor to gain unauthorized access to a system.
- **Shellcode**: Shellcode is malicious code that carries a payload, which is run after a system vulnerability is exploited.
- **Workspace**: "Workspace" is the term used to refer to a project in the Metasploit Framework.

Metasploit Workspaces

In Metasploit, the term “workspace” is equivalent to the term “project.” You can have more than one workspace (for example, if you’re using multiple desktops). To work with workspaces in Metasploit, use the following commands:

- `workspace` — Displays the workspaces you currently have selected
- `workspace -a <WORKSPACE NAME>` — Adds a new workspace
- `workspace -d <WORKSPACE NAME>` — Deletes a workspace
- `workspace default` — Takes you to the default workspace
- `workspace -h` — Opens the Help menu

The Exploit Database

The **Exploit Database** is an online database where you can find a huge number of exploits all in one place.

When you click on an exploit, it tells you important information about the exploit, including the ExploitDB ID number, CVE number, author, exploit type, the platform for which it is available, and the date of publication. You can search the database by all of these features.

CVE stands for Common Vulnerabilities and Exposures. It is a structured list of publicly known information security vulnerabilities and exposures. Each entry contains an ID number, description, and at least one public reference. Knowing an exploit’s CVE number is very useful for penetration testing.

Exploiting a System

In order to exploit a target, we have to maneuver through the framework and obtain all of the information about the target we need.

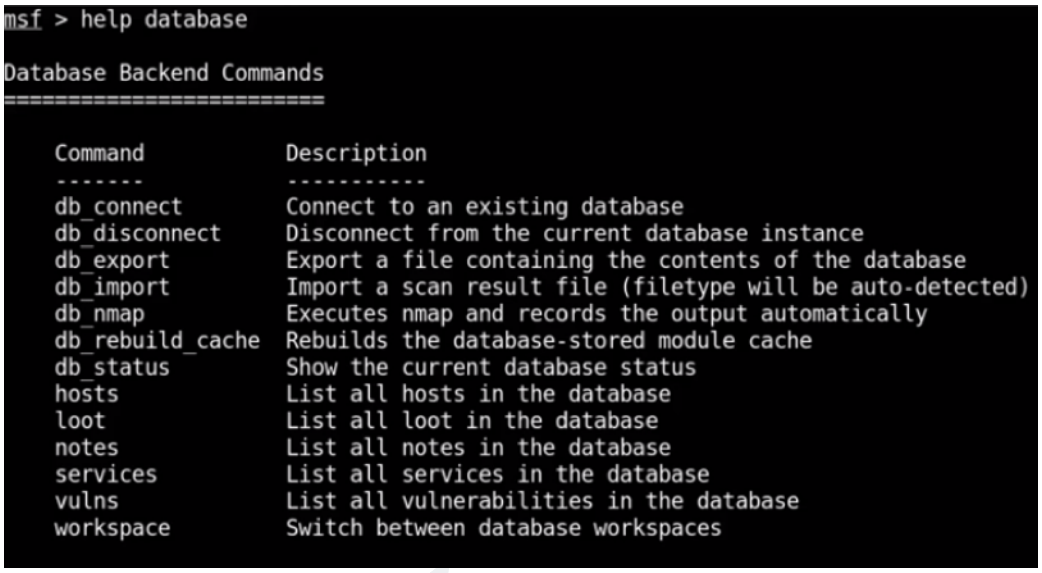
1. First, sweep the network:

   ```
   db_map -sn <NETWORK ADDRESS>
   ```

2. Next, we will perform a detailed scan of a single host on a specific port with the verbosity set to max.

   ```
   db_map -A 192.168.122.13 -p l-65535 -vvv
   ```

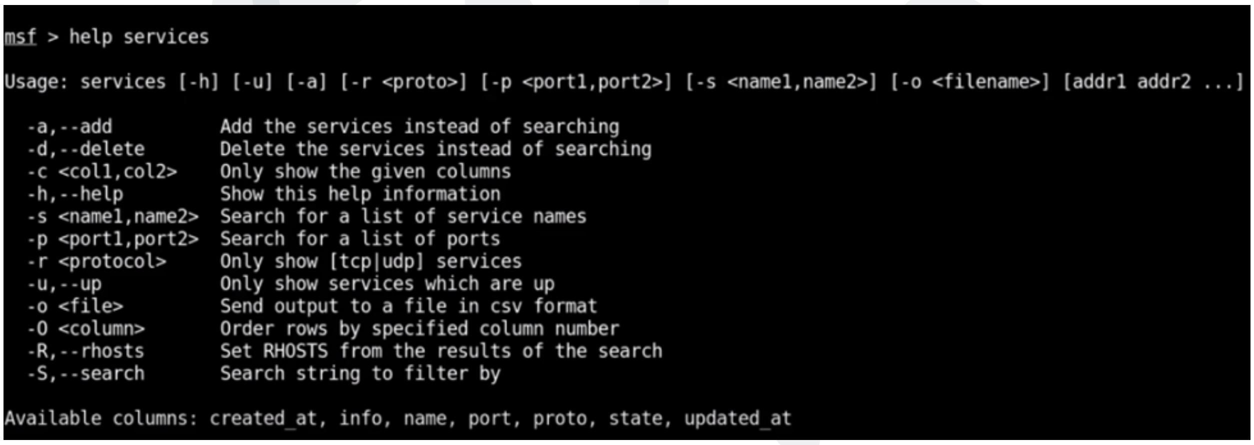
3. Now that the scan is done, type `help database` to list the database backend commands:



```
msf > help database

Database Backend Commands
=========================

    Command           Description
    -------           -----------
    db_connect        Connect to an existing database
    db_disconnect     Disconnect from the current database instance
    db_export         Export a file containing the contents of the database
    db_import         Import a scan result file (filetype will be auto-detected)
    db_nmap           Executes nmap and records the output automatically
    db_rebuild_cache  Rebuilds the database-stored module cache
    db_status         Show the current database status
    hosts             List all hosts in the database
    loot              List all loot in the database
    notes             List all notes in the database
    services          List all services in the database
    vulns             List all vulnerabilities in the database
    workspace         Switch between database workspaces
```

4. By browsing through the database, we can find out which services contain vulnerabilities.
5. There are two commands we can use to display Help options: `help services` and `help vulns`.

`help services`

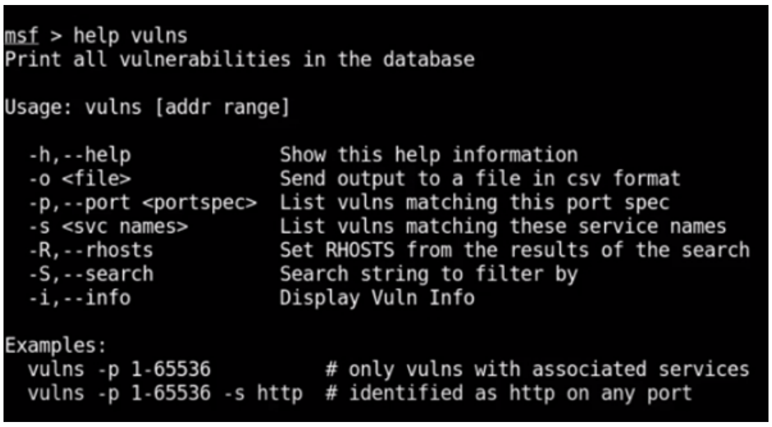


```
msf > help services

Usage: services [-h] [-u] [-a] [-r <proto>] [-p <port1,port2>] [-s <name1,name2>] [-o <filename>] [addr1 addr2 ...]

  -a,--add          Add the services instead of searching
  -d,--delete       Delete the services instead of searching
  -c <col1,col2>    Only show the given columns
  -h,--help         Show this help information
  -s <name1,name2>  Search for a list of service names
  -p <port1,port2>  Search for a list of ports
  -r <protocol>     Only show [tcp|udp] services
  -u,--up           Only show services which are up
  -o <file>         Send output to a file in csv format
  -O <column>       Order rows by specified column number
  -R,--rhosts       Set RHOSTS from the results of the search
  -S,--search       Search string to filter by

Available columns: created_at, info, name, port, proto, state, updated_at
```

We can use the following commands to get more information in the `help services` menu:

- `services` — Lists all services in the database
- `services -c name port` — Lists the columns containing service names and port numbers
- `-c <COLUMN NAMES>`— Shows the requested columns

`help vulns`



```
msf > help vulns
Print all vulnerabilities in the database

Usage: vulns [addr range]

  -h,--help             Show this help information
  -o <file>             Send output to a file in csv format
  -p,--port <portspec>  List vulns matching this port spec
  -s <svc names>        List vulns matching these service names
  -R,--rhosts           Set RHOSTS from the results of the search
  -S,--search           Search string to filter by
  -i,--info             Display Vuln Info

Examples:
  vulns -p 1-65536          # only vulns with associated services
  vulns -p 1-65536 -s http  # identified as http on any port
```

We can use the following commands to get more information in the `help vulns` menu:

- `vulns` — Prints all vulnerabilities in the database
- `vulns -p 1-54536 -i` — Lists vulnerabilities matching the given port spec, lists the vulnerability info, and tells us where we can get the corresponding exploit

6. After we've found the exploit we want to use, enter the `use`command and paste the path to the exploit:

   ```
   use <PATH_OF_CHOSEN_EXPLOIT>
   ```

7. We can also list the module options with the `show options` command.
8. Set the remote host and remote port.

   ```
   set RPORT 21
   set RHOST <IP ADDRESS>
   ```

9. Now we're ready to run the exploit:

   ```
   run
   ```

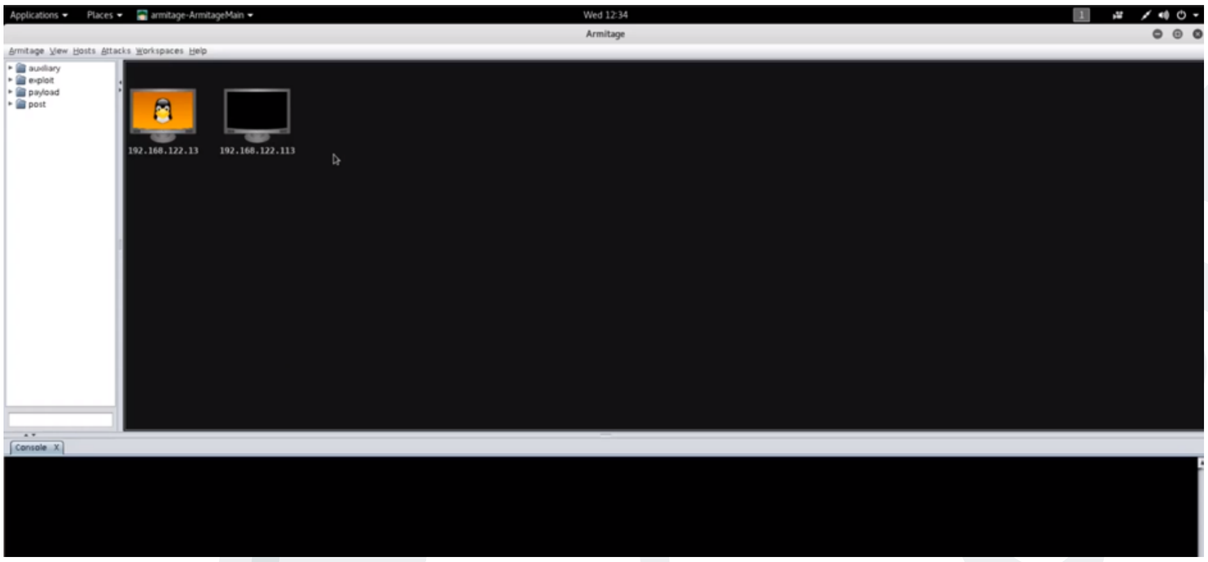
In this example, we run the exploit using the `run` command in order to spawn a shell on the remote target.

10. All that's left to do is establish persistent access; otherwise, we won't be able to access and exploit the target system again if it reboots. Depending on the situation, there are different ways of establishing persistent access. If we are exploiting a server, we know that it is most likely going to have SSH, and it will most likely be configured to run at boot time. So one solution is to add one more SSH key, which will probably go unnoticed.

The Armitage GUI

Armitage is Metasploit's graphical user interface (GUI) tool. Armitage includes all of the options available in the msfconsole interface, plus it recommends exploits and visualizes targets. It also has many post-exploitation options that can help you establish persistent access.

Here's what the Armitage GUI looks like:



An interesting feature of Armitage is the Hail Mary attack. The Hail Mary attack throws every available exploit against a target to see if any of them work. In other words, it launches a flood of exploits at hosts in the current workspace.