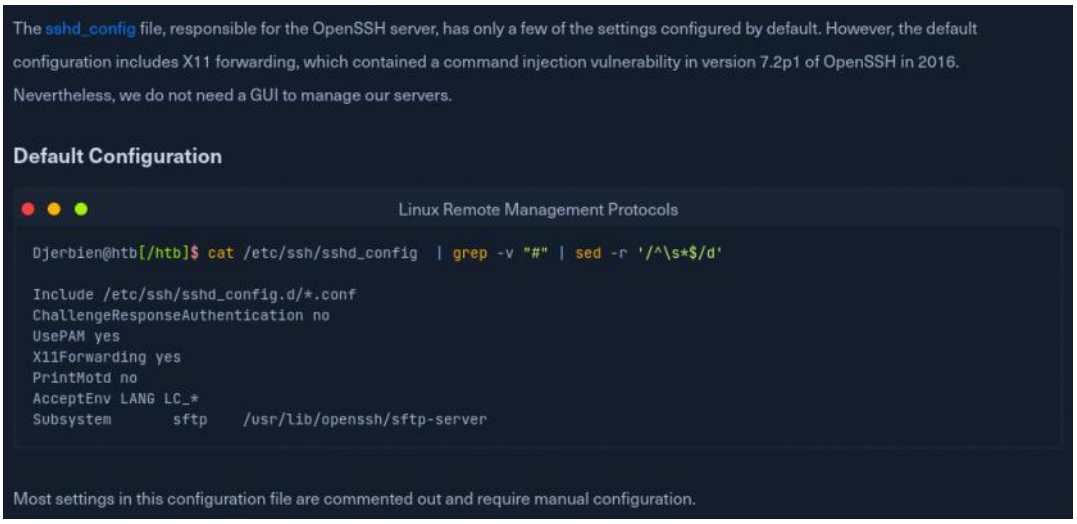
SSH

Configuration d'accès à SSH (connection)

```
1. Password authentication
2. Public-key authentication
3. Host-based authentication
4. Keyboard authentication
5. Challenge-response authentication
6. GSSAPI authentication
```

https://www.golinuxcloud.com/pssh-public-key-authentication-passwordless/
https://www.golinuxcloud.com/openssh-authentication-methods-sshd-config/

Default Configuration



```
# ssh_config(5) man page.

Include /etc/ssh/ssh_config.d/*.conf

Host *
#   ForwardAgent no
#   ForwardX11 no
#   ForwardX11Trusted yes
#   PasswordAuthentication yes
#   HostbasedAuthentication no
#   GSSAPIAuthentication no
#   GSSAPIDelegateCredentials no
#   GSSAPIKeyExchange no
#   GSSAPITrustDNS no
#   BatchMode no
#   CheckHostIP no
#   AddressFamily any
#   ConnectTimeout 0
#   StrictHostKeyChecking ask
#   IdentityFile ~/.ssh/id_rsa
#   IdentityFile ~/.ssh/id_dsa
#   IdentityFile ~/.ssh/id_ecdsa
#   IdentityFile ~/.ssh/id_ed25519
#   Port 22
#   Ciphers aes128-ctr,aes192-ctr,aes256-ctr,aes128-cbc,3des-cbc
#   MACs hmac-md5,hmac-sha1,umac-64@openssh.com
#   EscapeChar ~
#   Tunnel no
#   TunnelDevice any:any
#   PermitLocalCommand no
#   VisualHostKey no
#   ProxyCommand ssh -q -W %h:%p gateway.example.com
#   RekeyLimit 1G 1h
#   UserKnownHostsFile ~/.ssh/known_hosts.d/%k
    SendEnv LANG LC_*
    HashKnownHosts yes
    GSSAPIAuthentication yes

┌──(jerbi㉿Anonymous)-[~/.ssh]
└─$
```

Dangerous Settings :

Despite the SSH protocol being one of the most secure protocols available today, some misconfigurations can still make the SSH server vulnerable to easy-to-execute attacks. Let us take a look at the following settings:

Setting	Description
`PasswordAuthentication yes`	Allows password-based authentication.
`PermitEmptyPasswords yes`	Allows the use of empty passwords.
`PermitRootLogin yes`	Allows to log in as the root user.
`Protocol 1`	Uses an outdated version of encryption.
`X11Forwarding yes`	Allows X11 forwarding for GUI applications.
`AllowTcpForwarding yes`	Allows forwarding of TCP ports.
`PermitTunnel`	Allows tunneling.
`DebianBanner yes`	Displays a specific banner when logging in.

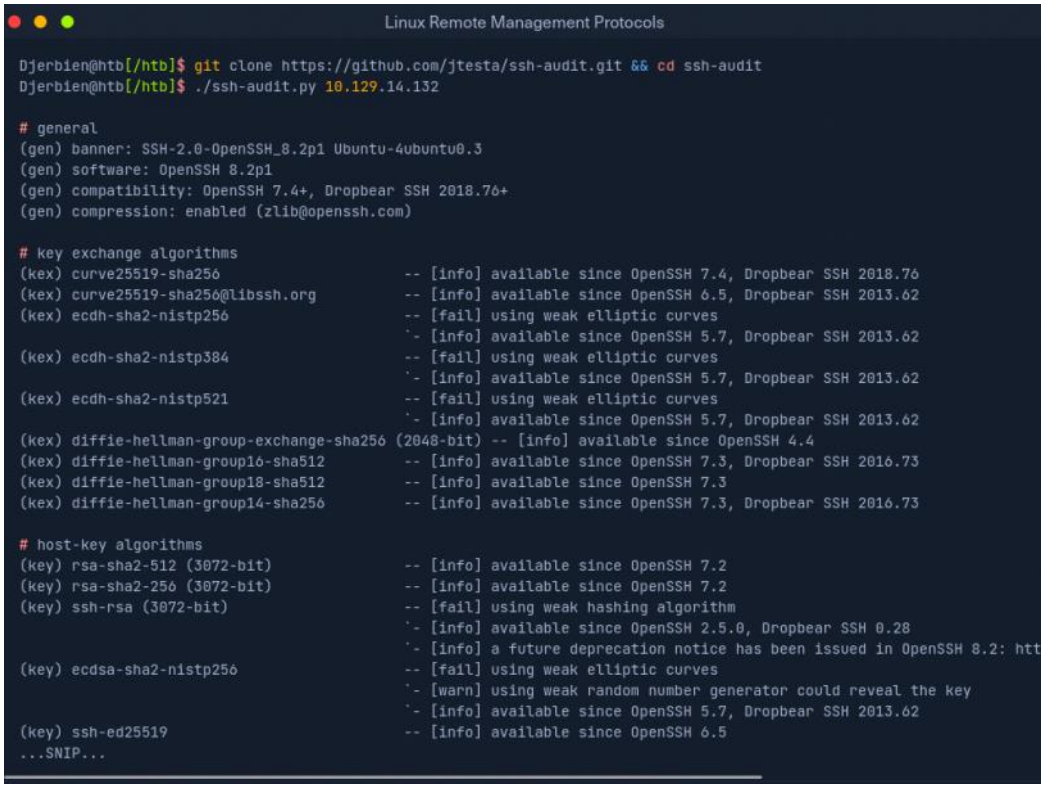
Allowing password authentication allows us to brute-force a known username for possible passwords. Many different methods can be used to guess the passwords of users. For this purpose, specific `patterns` are usually used to mutate the most commonly used passwords and, frighteningly, correct them. This is because we humans are lazy and do not want to remember complex and complicated passwords. Therefore, we create passwords that we can easily remember, and this leads to the fact that, for example, numbers or characters are added only at the end of the password. Believing that the password is secure, the mentioned patterns are used to guess precisely such "adjustments" of these passwords. However, some instructions and hardening guides can be used to harden our SSH servers.

SSH best practice to make it hard to MITM

https://www.ssh-audit.com/hardening_guides.html

Footprinting the Service

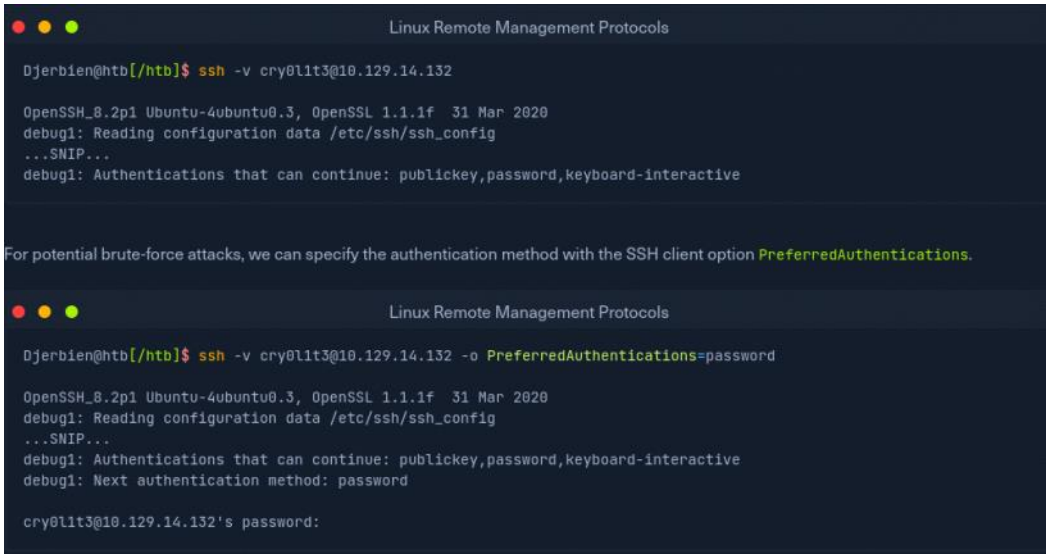
One of the tools we can use to fingerprint the SSH server is ssh-audit. It checks the client-side and server-side configuration and shows some general information and which encryption algorithms are still used by the client and server. Of course, this could be ex ploited by attacking the server or client at the cryptic level later.



The first thing we can see in the first few lines of the output is the banner that reveals the version of the OpenSSH server. The previous versions had some vulnerabilities, such as **CVE-2020-14145**, which allowed the attacker the capability to Man-In-The-Middle and attack the initial connection attempt. The detailed output of the connection setup with the OpenSSH server can also often provide im portant information, such as which authentication methods the server can use.

Change Authentication Method

Djerbien@htb[/htb]$ ssh -v cry0l1t3@10.129.14.132
Djerbien@htb[/htb]$ ssh -v cry0l1t3@10.129.14.132 -o PreferredAuthentications=password



Even with this obvious and secure service, we recommend setting up our own OpenSSH server on our VM, experimenting with it, and familiarizing ourselves with the different settings and options.

We may encounter various banners for the SSH server during our penetration tests. By default, the banners start with the version of the protocol that can be applied and then the version of the server itself. For example, with `SSH-1.99-OpenSSH_3.9p1`, we know that we can use both protocol versions SSH-1 and SSH-2, and we are dealing with OpenSSH server version 3.9p1. On the other hand, for a banner with `SSH-2.0-OpenSSH_8.2p1`, we are dealing with an OpenSSH version 8.2p1 which only accepts the SSH-2 protocol version.

Rsync

Rsync is a fast and efficient tool **for locally and remotely copying files.**
By default, it uses port **873** and can be configured to use SSH for secure file transfers by piggybacking on top of an established SSH server connection.

https://book.hacktricks.xyz/network-services-pentesting/873-pentesting-rsync

Rsync is a fast and efficient tool for locally and remotely copying files. It can be used to copy files locally on a given machine and to/from remote hosts. It is highly versatile and well-known for its delta-transfer algorithm. This algorithm reduces the amount of data transmitted over the network when a version of the file already exists on the destination host. It does this by sending only the differences between the source files and the older version of the files that reside on the destination server. It is often used for backups and mirroring. It finds files that need to be transferred by looking at files that have changed in size or the last modified time. By default, it uses port 873 and can be configured to use SSH for secure file transfers by piggybacking on top of an established SSH server connection.

This guide covers some of the ways Rsync can be abused, most notably by listing the contents of a shared folder on a target server and retrieving files. This can sometimes be done without authentication. Other times we will need credentials. If you find credentials during a pentest and run into Rsync on an internal (or external) host, it is always worth checking for password re-use as you may be able to pull down some sensitive files that could be used to gain remote access to the target.

Let's do a bit of quick footprinting. We can see that Rsync is in use using protocol 31.

Scanning for Rsync

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ sudo nmap -sV -p 873 127.0.0.1

Starting Nmap 7.92 ( https://nmap.org ) at 2022-09-19 09:31 EDT
Nmap scan report for localhost (127.0.0.1)
Host is up (0.0058s latency).

PORT    STATE SERVICE VERSION
873/tcp open  rsync   (protocol version 31)

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 1.13 seconds
```

Probing for Accessible Shares

We can next probe the service a bit to see what we can gain access to.

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ nc -nv 127.0.0.1 873

(UNKNOWN) [127.0.0.1] 873 (rsync) open
@RSYNCD: 31.0
@RSYNCD: 31.0
#list
dev             Dev Tools
@RSYNCD: EXIT
```

Enumerating an Open Share

Here we can see a share called `dev`, and we can enumerate it further.

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ rsync -av --list-only rsync://127.0.0.1/dev

receiving incremental file list
drwxr-xr-x             48 2022/09/19 09:43:10 .
-rw-r--r--              0 2022/09/19 09:34:50 build.sh
-rw-r--r--              0 2022/09/19 09:36:02 secrets.yaml
drwx------             54 2022/09/19 09:43:10 .ssh

sent 25 bytes  received 221 bytes  492.00 bytes/sec
total size is 0  speedup is 0.00
```

From the above output, we can see a few interesting files that may be worth pulling down to investigate further. We can also see that a directory likely containing SSH keys is accessible. From here, we could sync all files to our attack host with the command `rsync -av rsync://127.0.0.1/dev`. If Rsync is configured to use SSH to transfer files, we could modify our commands to include the `-e ssh` flag, or `-e "ssh -p2222"` if a non-standard port is in use for SSH. This guide is helpful for understanding the syntax for using Rsync over SSH.

R-Services

R-Services are a suite of services hosted to enable remote access or issue commands between Unix hosts over TCP/IP.

Ports	R-services span across the ports **512**, **513**, and **514** and are only accessible through a suite of programs known as r-commands.

R-Services are a suite of services hosted to enable remote access or issue commands between Unix hosts over TCP/IP. Initially developed by the Computer Systems Research Group (`CSRG`) at the University of California, Berkeley, `r-services` were the de facto standard for remote access between Unix operating systems until they were replaced by the Secure Shell (`SSH`) protocols and commands due to inherent security flaws built into them. Much like `telnet`, r-services transmit information from client to server(and vice versa.) over the network in an unencrypted format, making it possible for attackers to intercept network traffic (passwords, login information, etc.) by performing man-in-the-middle (`MITM`) attacks.

`R-services` span across the ports `512`, `513`, and `514` and are only accessible through a suite of programs known as `r-commands`. They are most commonly used by commercial operating systems such as Solaris, HP-UX, and AIX. While less common nowadays, we do run into them from time to time during our internal penetration tests so it is worth understanding how to approach them.

The R-commands suite consists of the following programs:

- `rcp (remote copy)`
- `rexec (remote execution)`
- `rlogin (remote login)`
- `rsh (remote shell)`
- `rstat`
- `ruptime`
- `rwho (remote who)`

Each command has its intended functionality; however, we will only cover the most commonly abused `r-commands`. The table below will provide a quick overview of the most frequently abused commands, including the service daemon they interact with, over what port and transport method to which they can be accessed, and a brief description of each.

Command	Service Daemon	Port	Transport Protocol	Description
rcp	rshd	514	TCP	Copy a file or directory bidirectionally from the local system to the remote system (or vice versa) or from one remote system to another. It works like the cp command on Linux but provides no warning to the user for overwriting existing files on a system.
rsh	rshd	514	TCP	Opens a shell on a remote machine without a login procedure. Relies upon the trusted entries in the /etc/hosts.equiv and .rhosts files for validation.
rexec	rexecd	512	TCP	Enables a user to run shell commands on a remote machine. Requires authentication through the use of a username and password through an unencrypted network socket. Authentication is overridden by the trusted entries in the /etc/hosts.equiv and .rhosts files.
rlogin	rlogind	513	TCP	Enables a user to log in to a remote host over the network. It works similarly to telnet but can only connect to Unix-like hosts. Authentication is overridden by the trusted entries in the /etc/hosts.equiv and .rhosts files.

The /etc/hosts.equiv file contains a list of trusted hosts and is used to grant access to other systems on the network. When users on one of these hosts attempt to access the system, they are automatically granted access without further authentication.

/etc/hosts.equiv

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ cat /etc/hosts.equiv

# <hostname> <local username>
pwnbox cry0l1t3
```

Now that we have a basic understanding of r-commands, let's do some quick footprinting using Nmap to determine if all necessary ports are open.

Scanning for R-Services

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ sudo nmap -sV -p 512,513,514 10.0.17.2

Starting Nmap 7.80 ( https://nmap.org ) at 2022-12-02 15:02 EST
Nmap scan report for 10.0.17.2
Host is up (0.11s latency).

PORT    STATE SERVICE    VERSION
512/tcp open  exec?
513/tcp open  login?
514/tcp open  tcpwrapped

Service detection performed. Please report any incorrect results at https://nmap.org/submit/ .
Nmap done: 1 IP address (1 host up) scanned in 145.54 seconds
```

Access Control & Trusted Relationships

The primary concern for r-services, and one of the primary reasons SSH was introduced to replace it, is the inherent issues regarding access control for these protocols. R-services rely on trusted information sent from the remote client to the host machine they are attempting to authenticate to. By default, these services utilize Pluggable Authentication Modules (PAM) for user authentication onto a remote system; however, they also bypass this authentication through the use of the /etc/hosts.equiv and .rhosts files on the system. The hosts.equiv and .rhosts files contain a list of hosts (IPs or Hostnames) and users that are trusted by the local host when a connection attempt is made using r-commands. Entries in either file can appear like the following:

> **Note:** The hosts.equiv file is recognized as the global configuration regarding all users on a system, whereas .rhosts provides a per-user configuration.

Sample .rhosts File

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ cat .rhosts

htb-student     10.0.17.5
+               10.0.17.10
+               +
```

As we can see from this example, both files follow the specific syntax of <username> <ip address> or <username> <hostname> pairs. Additionally, the + modifier can be used within these files as a wildcard to specify anything. In this example, the + modifier allows any external user to access r-commands from the htb-student user account via the host with the IP address 10.0.17.10.

https://debathena.mit.edu/trac/wiki/PAM

As we can see from this example, both files follow the specific syntax of `<username> <ip address>` or `<username> <hostname>` pairs. Additionally, the `+` modifier can be used within these files as a wildcard to specify anything. In this example, the `+` modifier allows any external user to access r-commands from the `htb-student` user account via the host with the IP address `10.0.17.10`.

Misconfigurations in either of these files can allow an attacker to authenticate as another user without credentials, with the potential for gaining code execution. Now that we understand how we can potentially abuse misconfigurations in these files let's attempt to try logging into a target host using `rlogin`.

Logging in Using Rlogin

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ rlogin 10.0.17.2 -l htb-student

Last login: Fri Dec  2 16:11:21 from localhost

[htb-student@localhost ~]$
```

We have successfully logged in under the `htb-student` account on the remote host due to the misconfigurations in the `.rhosts` file. Once successfully logged in, we can also abuse the `rwho` command to list all interactive sessions on the local network by sending requests to the UDP port 513.

Listing Authenticated Users Using Rwho

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ rwho

root     web01:pts/0 Dec  2 21:34
htb-student     workstn01:tty1  Dec  2 19:57  2:25
```

From this information, we can see that the `htb-student` user is currently authenticated to the `workstn01` host, whereas the `root` user is authenticated to the `web01` host. We can use this to our advantage when scoping out potential usernames to use during further attacks on hosts over the network. However, the `rwho` daemon periodically broadcasts information about logged-on users, so it might be beneficial to watch the network traffic.

Listing Authenticated Users Using Rusers

To provide additional information in conjunction with `rwho`, we can issue the `rusers` command. This will give us a more detailed account of all logged-in users over the network, including information such as the username, hostname of the accessed machine, TTY that the user is logged in to, the date and time the user logged in, the amount of time since the user typed on the keyboard, and the remote host they logged in from (if applicable).

Linux Remote Management Protocols

```
Djerbien@htb[/htb]$ rusers -al 10.0.17.5

htb-student     10.0.17.5:console          Dec 2 19:57     2:25
```

Listing Authenticated Users Using Rwho

Linux Remote Management Protocols

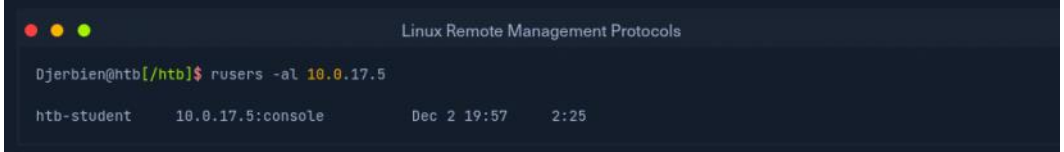
```
Djerbien@htb[/htb]$ rwho

root     web01:pts/0 Dec  2 21:34
htb-student     workstn01:tty1  Dec  2 19:57  2:25
```

From this information, we can see that the `htb-student` user is currently authenticated to the `workstn01` host, whereas the `root` user is authenticated to the `web01` host. We can use this to our advantage when scoping out potential usernames to use during further attacks on hosts over the network. However, the `rwho` daemon periodically broadcasts information about logged-on users, so it might be beneficial to watch the network traffic.

Listing Authenticated Users Using Rusers

To provide additional information in conjunction with `rwho`, we can issue the `rusers` command. This will give us a more detailed account of all logged-in users over the network, including information such as the username, hostname of the accessed machine, TTY that the user is logged in to, the date and time the user logged in, the amount of time since the user typed on the keyboard, and the remote host they logged in from (if applicable).



As we can see, R-services are less frequently used nowadays due to their inherent security flaws and the availability of more secure protocols such as SSH. To be a well-rounded information security professional, we must have a broad and deep understanding of many systems, applications, protocols, etc. So, file away this knowledge about R-services because you never know when you may encounter them.

Final Thoughts

Remote management services can provide us with a treasure trove of data and often be abused for unauthorized access through either weak/default credentials or password re-use. We should always probe these services for as much information as we can gather and leave no stone unturned, especially when we have compiled a list of credentials from elsewhere in the target network.

.