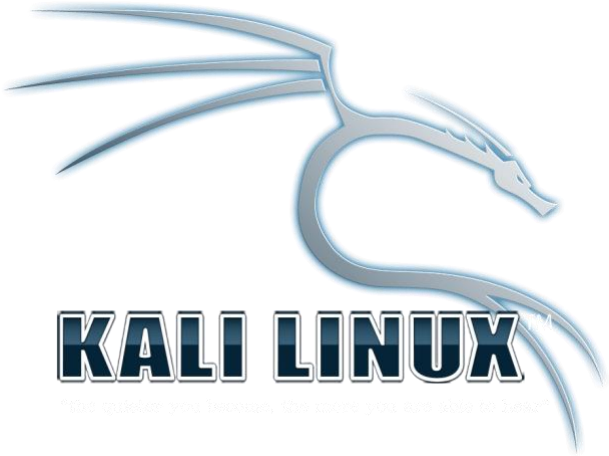
COMP3052.SEC GETTING STARTED WITH KALI

(**Thank you to Mike Pound**)

LAB DESCRIPTION

In this lab you'll be familiarizing yourself with the Kali Linux distribution, this will include a quick grounding in Linux and the way it operates, as well as more Kali-specific tools. This is an exploration lab, it is not assessed, but if you're at all unsure about using Linux, this is a good opportunity to refresh your skills before the later labs. We'll also look at the system and authentication logs, which are vital for those who administrate Linux servers.



INTRODUCTION

Linux is a Unix-like and POSIX-compliant computer operating system developed under the model of free and open source software development. The Linux kernel itself is very low level, providing core operating system functions like file access. Traditionally this is used within "distributions" which add packages, user interfaces etc. to make it easier to use and more functional. In this lab we are using the Kali distribution (www.kali.org), which has been developed to specifically aid white-hat ethical hacking. If there's a useful piece of software for security and penetration testing, it's likely you'll find it in Kali.

BOOTING UP KALI

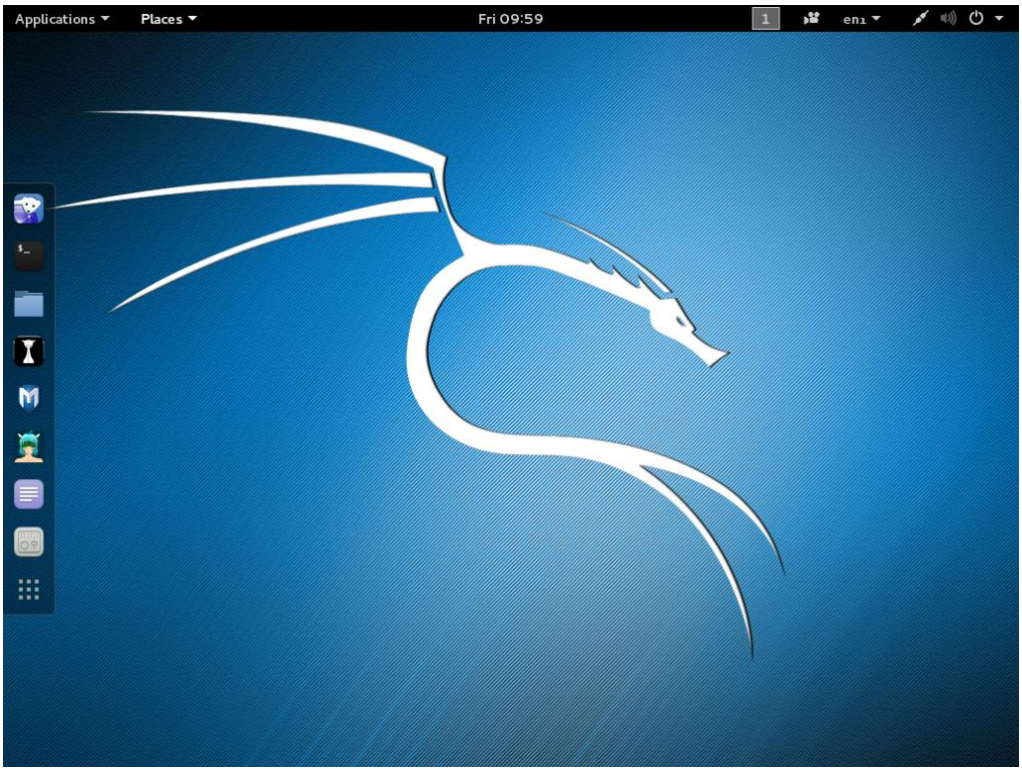
Kali can be booted from USB sticks, hard drives, networks, virtual machines, anywhere really. In this and subsequent labs you'll be running a pre-configured virtual machine. We're using Oracle VM VirtualBox, a VM manager for Windows and other operating systems. VirtualBox provides a link between your own OS and hardware, and the guest OS, Kali. As far as Kali is concerned, it's just running on a standard machine.

Virtualbox has been pre-installed on the lab machines. If you're using your own device, you can install it from www.virtualbox.org. Instructions on getting the virtual machines set up are provided in the "Getting Set Up" documents (*LABSETUP.01a.Getting.SetUp.VDI and *LABSETUP.01b.Getting.SetUp).

* I will frequently omit the "COMP3052.SEC." from filenames like "LABSETUP.01b.Getting.SetUp" ("COMP3052.SEC.LABSETUP.01b.Getting.SetUp")

LOGIN INFORMATION

It's customary to have a standard user for normal use, then elevate your privileges as required using *sudo*. There is a root account, but it's good practice to avoid using it.



The Kali Desktop

Normal User
Username: sec
Password: security

Root
Username: root
Password: toor

Login, and then you'll be presented with the Kali derivation of the Gnome desktop environment.

THE TERMINAL

Kali uses a bash terminal, which is extremely common. You can access it using the menu bar on the left of the screen, click the terminal icon and wait a moment. Once in the terminal, you'll see a prompt that tells you who you are, and what directory you're in. Usually the terminal will start in your home directory, which is shortened to `~` for convenience. This actually corresponds to /home/sec for the sec user, and /root for the root user. Use the `ls` command to list all of the files in your home directory:

```
sec@kali:~$ ls –l
drwxr-xr-x 2 sec sec 4096 Dec 17 08:39 Desktop
drwxr-xr-x 4 sec sec 4096 Dec 17 09:15 Downloads
drwxr-xr-x 5 sec sec 4096 Dec 18 07:22 lab2
…
```

Notice that the `–l` flag provides extra information, including the file mode bits (permissions). The `d` signifies a directory, then the subsequent groups of three characters `rxw` represent read, write and execute permissions for the owner, group and others, respectively. A dash indicates that person does not have that access.

The home directory includes folders for each lab session, `lab1`, `lab2`, etc. This is where you'll be spending most of your time during these labs. Remember that you can return to the home directory from anywhere by simply using `cd ~`.

USEFUL KALI TOOLS

Kali has been designed for ethical hacking. If you click the "Applications" menu in the top corner of the desktop, you'll see sub-categories with various types of tool, as well as some general Linux ones. There are a lot of security focused tools in Kali, only a few of which we will look at. Some of them, like "backdoor-factory" are obviously off limits during these labs!

Beyond security tools which we'll introduce specifically for each lab, Kali includes common editors and other software that will make your life easier:

Firefox: Web browser, browsing to websites will come up in various labs.

Gedit: Probably the most useful editor currently installed. Access from the side-bar, or via the terminal using `gedit filename &`.

Leafpad: An alternative editor if you'd prefer, accessed via `leafpad` or the Show Applications button.

Look around the system. At its core is Debian, which is also what Ubuntu is based on, anyone familiar with either of these systems will be at home here. We're going to be using Kali a lot during the labs, so the more familiar you are with it, the better.

SUDO

Unless you're logged in as root, much of the core file system is off limits. Kali ships as the root user by default, which also means a lot of the shared files are owned by root. If you hit an access problem, it'll be obvious. Something you expect should work, won't, or you'll get a very clear access denied message. In these cases you can temporarily elevate your privileges using the `sudo` command. Simply prepend `sudo` to your own command and permission will be granted.

```
sec@kali:~$ chown sec:sec txtfile
chown: changing ownership of ‘txtfile’: Operation not permitted
sec@kali:~$ sudo chown sec:sec txtfile
sec@kali:~$
```

Of course you'll need the password belonging to the sec account. Try to get used to operating most of the time as a non-root user. It's a pain sometimes, because you'll forget sudo and see a lot of warnings, but it's a lot harder to accidentally delete or alter crucial files. Many modern Linux distributions (and OSX) actually disable the root user completely, only sudo will work. Windows also indirectly does this, with its UAC, but often doesn't require a password.

PIPING

In a Linux shell, piping allows you to pass the output of one command into the input of another. This is a very powerful tool, it saves a huge amount of time and lets you avoid a lot of small intermediate files. Let's look at a simple example, suppose we're trying to debug an internet connection problem, many Ethernet-related messages are stored in the Linux syslog. Let's have a look:

```
sec@kali:~$ sudo cat /var/log/syslog
```

Turns out there's a lot of stuff in there! Let's be more specific using a pipe, we can pipe the output into grep, a utility which will search for a keyword.

```
sec@kali:~$ sudo cat /var/log/syslog | grep eth0
```

`eth0` is the standard name for the primary Ethernet adaptor. This reduces the output from well over 3000 lines to about 130 (in my case). You can actually count the lines by further piping into the `wc` command:

```
sec@kali:~$ sudo cat /var/log/syslog | grep eth0 | wc -l
137
```

There are a lot of small utility functions in linux that can be chained together to perform more complex tasks.

REDIRECTING OUTPUT

In a similar way to the pipe function, we can redirect standard console output to a file, rather than having it appear on the screen. Let's say we want to move the output of the previous command into a file for analysis later. Begin by moving into the lab1 directory, then repeat the last command with the redirection `>`:

```
sec@kali:~$ cd lab1
sec@kali:/lab1$ sudo cat /var/log/syslog | grep eth0 > eth0log
```

This arbitrary example may not seem very useful, but we'll be using pipes and redirection below when looking at the authentication log, and they come up a lot.

AUTHENTICATION LOGS

Any machine with an open SSH port gets bombarded constantly by botnets. They try random common username and password combinations in an effort to gain root access. If they do, they'll install a rootkit and add you into the botnet. Getting attacked like this is a bit surprising the first time you see it, but it's extremely common and something to (usually) ignore.

All access logs for Kali are stored in /var/log/auth.log. Let's have a look:

```
sec@kali:~$ sudo cat /var/log/auth.log
```

A lot more text! Have a scroll through, most of it is fairly intuitive. Any line that begins with `kali sudo` is likely to be someone using `sudo` to elevate privileges. Read the rest of the line and you can see who did this (sec, usually) and what command they used. Linux logs are a great way for administrators to see what's going on. If you attempt to use `sudo` on a school machine, it'll get stored in the logs! (or passed to Santa).

Look for CRON opening a session for user root. This isn't a hacker, Cron is the linux scheduler, and is waking up to do regularly scheduled tasks.

On this Kali machine, SSH isn't enabled, and the machine's only just been powered on anyway. This means the auth.log is pretty clean. Let's look at some access logs from a machine that's been "in the open" for a while, Mike's. We've copied a couple of weeks' worth of December logs from one of the machines on (UK) campus to the lab1 folder. Find it, and use `cat` to read its contents.

Oh dear! Any log entry by sshd is related to someone authenticating with the sshd server on the machine. You'll notice that the vast majority are failed attempts, often with some pretty comical usernames. If you scroll about you'll see a lot of root login attempts, usually until the server gets annoyed and boots them off, at which point they reconnect and try again. Also notice the last few lines, which is Mike using scp to extract these authentication logs. Nothing goes unlogged in Linux!

There's much too much information here to decipher, let's use some commands and scripting to make sense of it. First let's only focus on sshd output, and we'll use an intermediate file to prevent having to use the same commands repeatedly for now. We'll copy the relevant lines of the log into another log:

```
sec@kali:~/lab1$ cat auth.log | grep sshd > sshd.log
```

If you look at this file, there's still way too much info, in fact if you use `ls -l` you'll see that the sizes are nearly the same, that's how much of this log is break-in attempts. Let's keep

only the lines related to failed passwords, they hold plenty of information. Grep can be used with regular expressions. A simple one will let us find lines containing “sshd” and “Failed password”

```
sec@kali:~/lab1$ cat sshd.log | grep -E 'sshd.*Failed password' > failed.log
```

This is getting better, but we’re seeing a lot of the same information on each line. We can use some regex inside a script to start cleaning everything up.

SCRIPTING

Saving lists of commands (and other programming constructs) into shell scripts lets you perform more complex tasks than individual commands and pipes. These can also be scheduled to run at certain intervals. In this case, the bash terminal we’re using contains some neat regular expression parsing we want for this log file. Begin by opening up gedit and a new script:

```
sec@kali:~/lab1$ gedit authscript.sh &
```

Inside gedit, we can edit a script then go back into the terminal to run it. Let’s do a simple one first, add echo “Hello World” in gedit, save, and return temporarily to the terminal.

We need to add execute permissions to this file, then run it:

```
sec@kali:~/lab1$ chmod +x authscript.sh
sec@kali:~/lab1$ ./authscript.sh
Hello World
```

Back to the editor. Ideally we’d like a script that runs through every line in the log, then extracts just the information we want. In this case, we want the username they used, and the IP address they connected from. There will be duplicates, which we’ll worry about later. We’ll also re-implement the commands from above in the script, because we might as well save ourselves from typing those out every time.

Let’s begin by looping through every line in our log, and using grep to quickly filter those we want. We can do this using a while loop that receives the input piped from our grep call:

```
cat auth.log | grep -E 'sshd.*Failed password' | while read -r line;
do
    echo "$line"
done
```

We’ve taken our command from before, and piped it into a while loop. The output of grep is passed into the line variable. Notice that when we assign the line variable, we simply use `line` but when we reference it, we append $, as in `$line`. Now we have each line that we

want in this loop, we simply need to handle each $line as it comes in. You can also delete `sshd.log` and `failed.log` if you like.

Instead of simply echoing the line, we want to use regular expressions to extract the correct parts of our messages. Regex is a language for finding strings of a given structure, it is a lot more powerful than standard string find and replace methods. Update your script to the one below, then we'll look at how it works.

```
regex="(\S+) from (([0-9]{1,3}\.){3}[0-9]{1,3})"
cat auth.log | grep -E 'sshd.*Failed password' | while read -r line ;
do
    if [[ $line =~ $regex ]] ;
    then
        echo "${BASH_REMATCH[1]}:${BASH_REMATCH[2]}"
    fi
done
```

In order to understand what's going on here, let's look at an example string we might encounter:

```
Dec 18 18:21:27 psbss01 sshd[11494]: Failed password for root from
188.126.90.74 port 51905 ssh2
```

We want to ignore as much of the start as possible, and look to match things that sit between "Failed password for" and " port". In fact, regex is cleverer than that, as long as we're very specific about what we're going to see, we can match the exact string "root from 188.126.90.74" and its component parts. Let's break the regular expression down into its component parts. The parser will walk through the line attempting to find a string exactly as we describe it, if it finds one, it will return true, and the matches as strings.

```
"(\S+) from (([0-9]{1,3}\.){3}[0-9]{1,3})"
```

- `(\S+)` : Matches the user name
  - `( )` : A capturing group, one we can reference as a variable at the end
  - `\S` : A non-white space character
  - `+` : one or more occurrences of the previous item
- `from` : The literal string " from ", note the spaces either side
- `(([0-9]{1,3}\.){3}[0-9]{1,3})` : Matches an IP address
  - `( )` : A capturing group
  - `([0-9]{1,3}\.){3}` : Matches three parts of an IP address
  - `[0-9]` : Any digit 0-9
  - `{1-3}` : 1 to 3 occurrences of the previous item
  - `\.` : Matches the . character
  - {3} : Exactly 3 occurrences of the previous item

As you can see, we begin by matching one or more non-whitespace characters, then a " from " string, then the IP address. The IP address consists of three occurrences of 1, 2 or 3 digits followed by a `.`, then the final 1, 2 or 3 digits.

Only if every single item in the entire structure is matched, will this return a true result. At which point, the bash script fills the BASH_REMATCH variable with any captured groups. These come in the order in which they are encountered in the regular expression, in this case $BASH_REMATCH[1] is the user, and [2] is the IP address. For testing purposes, we echo these to the screen.

If you run the script, you'll see a lot of lines of output, but it's now all the information we want. It'll take about 10 seconds to finish, you can always press Ctrl+C to cancel if you have seen enough.

The last thing we want to do is remove the duplicate entries. Some IP addresses try multiple user / password combinations, and the same user is listed multiple times. We can use the linux `uniq` command to preserve only the distinct entries, but this only works on a stream, not a complete list. This means we need the unique entries to be next to each other, otherwise it won't spot them. This is easily done using the `sort` program, so we pipe into both at the end of the loop. Here is the final script:

```
regex="(\S+) from (([0-9]{1,3}\.){3}[0-9]{1,3})"
cat auth.log | grep -E 'sshd.*Failed password' | while read -r line ;
do
    if [[ $line =~ $regex ]] ;
    then
        echo "${BASH_REMATCH[1]}:${BASH_REMATCH[2]}"
    fi
done | sort | uniq
```

Output this to a file:

```
sec@kali:~/lab1$ ./authscript.sh > baddies
```

There we go, a file containing a list of attempted usernames, and the IP addresses of the machines that attempted to log in! Of course we're not going to actually do anything with these IP addresses this time, (least of all counterattack!), but what we've implemented here is the start of a system like denyhosts, or fail2ban — you should check out what they are.

OPTIONAL

If you're interested, we could take this a step further. We have the IP addresses of machines that have been attempting to hack us. We can use an online whois lookup to find information on their ISP. Theoretically we could even send automated emails reporting the abuse. Here's a script that takes an IP address as an argument, looks it up, and attempts to extract the abuse email.

```
line=`whois $1 | grep -iE 'abuse.*@' | head -1`
regex="([A-Za-z0-9]+@[A-Za-z0-9.-]+\.[A-Za-z]{2,}).*"
if [[ $line =~ $regex ]] ;
then
    echo "${BASH_REMATCH[1]}"
fi
```

Can you find an IP address for which this doesn't work? If so, can you debug the code and work out why?

CONCLUSION

In this lab session you've familiarized yourself with the basics of the Linux operating system, and some of its more complex and useful aspects, such as scripting. We've been over what makes Kali a useful operating system for penetration testing, we'll be using a number of its included tools throughout the semester!