

Yookiterm – <http://exploit.courses>

## exploit.courses

### What?

This website provides an interactive exploit development learning area. Every user has access to his own, personal Linux container. The container can be x32 and x64, with and without ASLR - and even 32 bit and 64 bit ARM.

```
# Exploit Writup
## Step by step
## Debugging
## Exploit
```

```
root@container:~# id
uid=0(root) gid=0(root)

root@container:~# ls
challenges

root@container:~# gdb
(gdb)
```

```
+ ---+
|         |
|         |
|         |
+ ---+

Bof Slides
* Offset
* Buf
* Ret
* Exploit
```

You dont need to have anything else then a modern browser. [Login](#), select [Challenges](#), and start hacking!

Just want to play around? Start your container without a challenge in the [Containers](#) tab.

### How?

In short: KVM + LXD.

Together with websockets and xterm.js.

Glued together with AngularJS and GO.

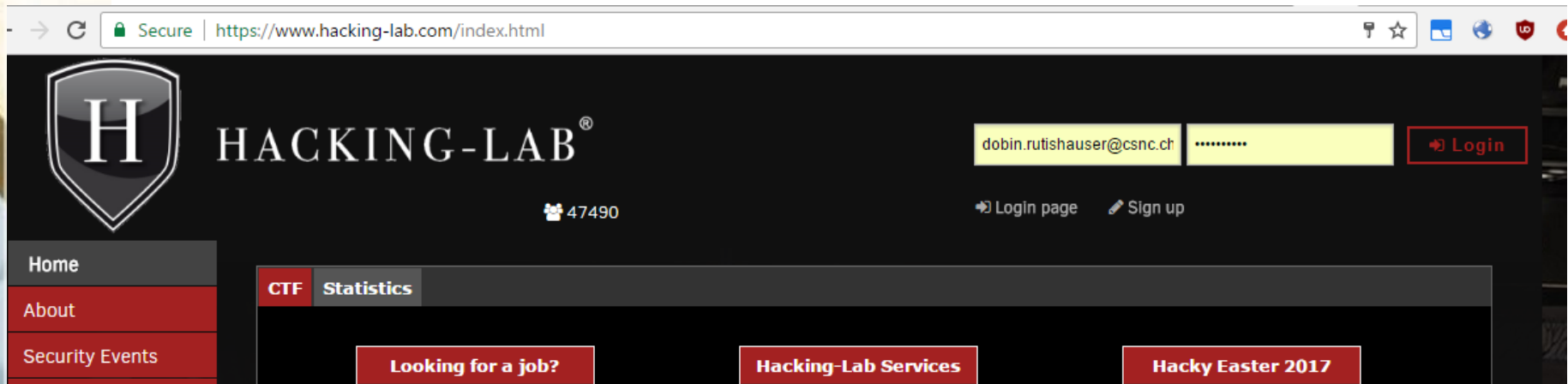
exploit.courses [ [Index](#) | [Challenges](#) | [Slides](#) | [Login](#) ]

## Login Hacking Lab

Login via Hacking-Lab

## Login Manual

Login



← → ↻ Secure | https://exploit.courses/#/challenges ☆

exploit.courses [ Index | Challenges | Slides | Container | dobin ]

ID	Title	Description	Arch	Bits	ASLR
0	<a href="#">Introduction to memory layout - basic</a>	Static and dynamic analysis of an ELF binary with Linux command line tools	intel	32	✓
1	<a href="#">Introduction to memory layout - advanced</a>	Research on where the different types of variables are stored in an ELF file	intel	32	✓
3	<a href="#">Introduction to shellcode development</a>	Create a basic running shellcode (print to console)	intel	32	✗
7	<a href="#">Function Call Convention in x86 (32bit)</a>	Analysis of function calling (gdb static and dynamic analysis)	intel	32	✗
8	<a href="#">C buffer analysis</a>	Analysis of out-of-bound read with gdb	intel	32	✗
10	<a href="#">Simple Buffer overflow</a>	Overwrite local variables on the stack to bypass authentication	intel	32	✗
11	<a href="#">Development of a buffer overflow exploit - 32 bit</a>	How to create a simple buffer overflow exploit	intel	32	✗
12	<a href="#">Development of a buffer overflow exploit - 64 bit</a>	How to create a simple buffer overflow exploit	intel	64	✗
13	<a href="#">Development of a remote buffer overflow exploit - 64 bit</a>	How to create a buffer overflow exploit for a networked server	intel	64	✗
14	<a href="#">Stack canary brute force</a>	How to brute force the stack canary in a remote server	intel	64	✗
15	<a href="#">Development of a remote buffer overflow exploit - 64 bit with ASLR</a>	How to create a contemporary remote buffer overflow exploit	intel	64	✓
20	<a href="#">Introduction to ARM</a>	Basics of the ARM architecture	arm	32	✓
21	<a href="#">Introduction to develop an ARM buffer overflow</a>	Create a simple buffer overflow on ARM	arm	32	✓
31	<a href="#">Heap use-after-free analysis</a>	Analyse a simple use-after-free bug in noteheap	intel	64	✓
50	<a href="#">Introduction to GDB</a>	A reference and playground for GDB	intel	32	✗

← → ↻

Secure | https://exploit.courses/#/challenge/0

☆

🔍

🔒

🔧

exploit.courses [ Index | Challenges | Slides | Container | dobin ]

+ Add Terminal

Container Info

## Introduction to memory layout - basic

### Introduction

In this challenge, we will look at some basic Linux tools which display information about the used memory regions in a program (static analysis) or a process (dynamic analysis).

Tools used:

- file
- readelf
- objdump
- gdb

### Goal

- Learn about memory layout and ELF
- Get a feeling of useful Linux tools

### Source

File: `~/challenges/challenge00/challenge0.c`

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
    if (argc == 1) {
        printf("Call: %s <name>\n", argv[0]);
        exit(0);
    }

    printf("Hello %s\n", argv[1]);
}
```

You can compile it by calling `make` in the folder `~/challenges/challenge00`

← → ↺

Secure | <https://exploit.courses/#/challenge/0>

☆

exploit.courses [ [Index](#) | [Challenges](#) | [Slides](#) | [Container](#) | [dobin](#) ]

+ Add Terminal

Container Info

```
root@hlUbuntu32:~# cd challenges/
root@hlUbuntu32:~/challenges# ls
LICENSE      challenge03  challenge10  challenge14
README.md    challenge07  challenge11  challenge15
challenge00  challenge08  challenge12  challenge21
challenge01  challenge1   challenge13  challenge31
root@hlUbuntu32:~/challenges# cd challenge01/
root@hlUbuntu32:~/challenges/challenge01# ls
Makefile  challenge1.c
root@hlUbuntu32:~/challenges/challenge01# make
gcc challenge1.c -o challenge1
root@hlUbuntu32:~/challenges/challenge01#
```

## Introduction to memory layout - basic

### Introduction

In this challenge, we will look at some basic Linux tools which display information about the used memory regions in a program (static analysis) or a process (dynamic analysis).

Tools used:

- file
- readelf
- objdump
- gdb

### Goal

- Learn about memory layout and ELF
- Get a feeling of useful Linux tools

### Source

File: `~/challenges/challenge00/challenge0.c`

```
#include <stdio.h>
#include <stdlib.h>

int main(int argc, char **argv) {
```

There are three different container you can access:

- ✦ ASLR (32, 64 bit)
- ✦ NO ASLR (32, 64 bit)
- ✦ ARM (32, 64 bit)

32 and 64 bit hosts are usually shared, including data

Container lifetime: 6 days

Container max lifetime: 12 days

Copy your files with SCP



The challenge files (vulnerable programs, exploits):

- ✦ <https://github.com/dobin/yookiterm-challenges-files>

The challenge writeups:

- ✦ <https://github.com/dobin/yookiterm-challenges>