Bacheloroppgave

IBE600 IT og digitalisering

Improving on a CTF Event for IBE321 Information Security and Risk Management

Erik Brendehaug

Totalt antall sider inkludert forsiden: 65
Totalt antall ord: ca. 10100

Molde, 30.05.2023

Obligatorisk egenerklæring/gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

Du/dere fyller ut erklæringen ved å klikke i ruten til høyre for den enkelte del 1-6:		
1.	**Jeg/vi erklærer herved at min/vår besvarelse er mitt/vårt eget arbeid, og at jeg/vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.**	☒
2.	**Jeg/vi erklærer videre at denne besvarelsen:** • ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • ikke refererer til andres arbeid uten at det er oppgitt. • ikke refererer til eget tidligere arbeid uten at det er oppgitt. • har alle referansene oppgitt i litteraturlisten. • ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	☒
3.	Jeg/vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§16 og 36.	☒
4.	Jeg/vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert, jf. høgskolens regler og konsekvenser for fusk og plagiat	☒
5.	Jeg/vi er kjent med at høgskolen vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk	☒
6.	**Jeg/vi har satt oss inn i regler og retningslinjer i bruk av** kilder og referanser på biblioteket sine nettsider	☒

Personvern

Personopplysningsloven

Forskningsprosjekt som innebærer behandling av personopplysninger iht. Personopplysningsloven skal meldes til Sikt for vurdering.

Har oppgaven vært vurdert av Sikt? ☐**ja** ☒**nei**

- Hvis ja:

Referansenummer:

- Hvis nei:

Jeg/vi erklærer at oppgaven ikke omfattes av Personopplysningsloven: ☒

Helseforskningsloven

Dersom prosjektet faller inn under Helseforskningsloven, skal det også søkes om forhåndsgodkjenning fra Regionale komiteer for medisinsk og helsefaglig forskningsetikk, REK, i din region.

Har oppgaven vært til behandling hos REK? ☐**ja** ☒**nei**

- Hvis ja:

Referansenummer:

Publiseringsavtale

Studiepoeng: 15

Veileder: Anolan Yamile Milanes Barrientos

ABSTRACT

In this project the goal was to improve upon the original Capture The Flag (CTF) event that I with some assistance from Simon Aarnes, created for IBE321 Information Security and Risk Management for the autumn 2022 semester.

This report therefore covers the process of taking the original CTF event and improving upon it. Among these improvements are new challenges, improvements for existing challenges, and a rewritten bootcamp document. There are also improvements to the backend code, as well as better documentation for the code to make it easier for future maintainers to understand the codebase and setup of the event. Also covered in the report are the different frameworks and software used to create the event, the challenges that were created and/or improved, and the issues faced during this implementation.

The report concludes with a discussion of the results of the project and gives a couple of notes for any future maintainers, along with a few suggestions for future work. Following the discussion is a summary of the report.

SAMMENDRAG

Hensikten med dette prosjektet var å forbedre det originale Capture The Flag-oppsettet som jeg, med litt assistanse fra Simon Aarnes, lagde for IBE321 Information Security and Risk Management for høstsemesteret 2022.

Denne rapporten går derfor gjennom prosessen av å ta det originale CTF-oppsettet for å forbedre det. Iblant disse forbedringene er: nye oppgaver, forbedringer av originale oppgaver, og et omskrevet bootcamp-dokument. Det var også gjort forbedringer blant det som brukerne/studentene ikke ser. Det vil si koden til nettsiden, samt bedre dokumentasjon av denne koden, slik at fremtidige administratorer for CTF-oppsettet får det lettere. Rapporten dekker også de forskjellige rammeverkene og programvarene som ble benyttet for å skape denne CTFen, samt oppgavene som ble laget/forbedret, samt eventuelle problemer som dukket opp under denne prosessen.

Rapporten konkluderer med en diskusjon om resultatene til prosjektet, og gir noen merknader for fremtidige administratorer, samt noen forslag for fremtidig arbeid. Til slutt er det en kort oppsummering av prosjektet.

PREFACE

I would like to thank my family for supporting me throughout my studies, and for showing me that I could actually do something like this project/report.

I would also like to thank my supervisor, Anolan Yamile Milanes Barrientos, for her guidance and feedback during the project and subsequent report.

Finally, I would like to thank Simon Aarnes, a fellow TA of IBE321, for helping me during brainstorming sessions to come up with the original challenges, his help with writing the original bootcamp document, and his invaluable help with setting up the server hardware, as I could not be on campus to do so myself.

CONTENTS

LIST OF FIGURES

LIST OF CODE

ACRONYMS

AI	Artificial Intelligence.
CSS	Cascading Style Sheets.
CTF	Capture The Flag.
DRY	Don’t Repeat Yourself.
Himolde	Molde University College.
HTB	Hack The Box.
LLM	Large Language Model.
OTW	OverTheWire.
SQL	Structured Query Language.
SSH	Secure Shell.
TA	Teaching Assistant.
VM	Virtual Machine.
VPN	Virtual Private Network.

GLOSSARY

Capture The Flag A limited time event where teams/users complete challenges to gain "flags".

cd A command line tool used to change the current directory in a terminal.

CTFd A platform for user and team based CTFs with built-in scoring tabulation and graphs.

Hack The Box A website that provides multiple challenges via "labs" as well as semi-custom CTFs.

IBE321 A course at the Himolde University College that teaches basics of cybersecurity concepts and principles.

OverTheWire A website that provides several groups of challenges (the first is Bandit) that are completed over a secure shell.

Regular Expressions A method of matching strings of text.

Repl Replit's version of a repo, but with the ability to run code directly from the browser, as well as google docs-like collaborative work.

SQL Structured Query Language, a language used to interact with databases.

SQL injection A method of exploiting a web application by injecting SQL code into a web form.

1 INTRODUCTION

1.1 What is a Capture the Flag Event?

A Capture The Flag event is a limited time event where users/players can either individually, or in teams, compete against each other by completing challenges [Upskilld 2021]. Depending on the setup of the competition players/teams may even be set up against each other, needing not only to gain points by completing competitions, but also by stealing other teams' flags.

1.2 Project Description

This project was originally started in the spring/summer of 2022, when Anolan Milanes, the teacher of IBE321, reached out asking if any previous students would be interested in working on a Capture The Flag event for the IBE321 class of the fall of 2022. I was very interested in this and contacted her asking for details. From there on, we started the project and with the help of one of her teaching assistants, the CTF event was created.

The improvement project aims to build upon this original CTF event, and to improve upon it in various ways. These improvements and their requirements will be covered in more detail in a later section of this report (see chapter 3)

1.3 Initial Motivation and Goals

The initial goal or motivation for creating the CTF was to give the students a fun and engaging way of practicing the theoretical knowledge and techniques that they have learned and discussed as a part of the course and its assignments.

Initially, the CTF was intended to be a single event consisting of several challenges of various topics and hacking techniques to make the students apply their knowledge, as well as to showcase the dangers of some of those vulnerabilities they would be exploiting. Later, however, the single CTF event was split into two events; one smaller event that would serve as part of an assignment, followed by a larger secondary event prior to the course's final exam that would serve as a refresher on topics relevant to the exam.

1.4 Motivation and Goals for the Improved Version

The list of goals for the improved version of the CTF is as follows:

- Create a single CTF event rather than two separate events.
- Create more challenges of varying difficulty.
- Create a proper SQL injection challenge.
- Improve the user interface and user experience (UI & UX) of the challenge server.

To take a deeper dive into the list, there were some shortcomings with the original version. While the overall project was a success, I was personally not completely satisfied with the result. Therefore, for this project of improving the original CTF the goals are to create a single CTF event with more subject-relevant challenges, with at least a few levels of difficulty, and a proper Structured Query Language (SQL) injection vulnerability challenge. Provided that these goals are reached, there is also the goal of improving the overall user interface and user experience (UI & UX) of the challenge server, as the original version was created using what would best be described as a "fairly beginner" level understanding of website design and development.

There were a few challenges or issues faced during the creation of the original CTF. Since this is not a report on the original, only the major ones will be mentioned here.

The first was that due to me being a remote student, I would have to use a VPN to access the school network. The downside to this was that SSH over a VPN is blocked by the school's firewall, and therefore I was unable to access the challenge server from my home computer. This meant that I had to use a third-party hosting service to host the challenge server if I also wanted the ability to easily update the server in the event of a bug in the code being discovered. This was solved by using Replit.com which provides free hosting for projects of many major programming languages [Replit 2023b (note: source mentions static sites, but testing confirms it also works for dynamic websites)]. The downside to this was that the server would be shut down after a period of inactivity, and though it would restart once a user accessed it, it would take a few seconds for the server to start up again. This was not a huge issue, but it did mean that the website would be unresponsive in the beginning for certain users.

The second major issue was with the implementation of the SQL injection challenge. In the original implementation of the challenge server the SQL injection performed by the users was checked against a regular expression (regex) to see if it was valid. This was done to prevent users from using a SQL injection to access the database and view the answers to the challenges. The downside to this, was that the regex was not perfect, and would wrongly flag some valid SQL injections as invalid.

These issues I faced among other more minor ones (navigation, layout, etc.), are what motivated me to improve the original CTF. Though there are more updates and added requirements for the improved version that will be covered in a later chapter, these issues inspired me to rewrite the app in such a way where it could be hosted anywhere rather than just Replit, and where the SQL injection challenge would use actual SQL queries to check if the injection was valid.

2 CTF PLATFORMS

2.1 Introduction to CTFs and Wargames

Before we get into the details of the different CTFs/wargames I took inspiration from when designing and building my own, we quickly need to go over how a wargame differs from a CTF event.

While CTFs are (as mentioned in the introduction) limited time events, wargames have no time limit and can be completed whenever the user wants to do so [Upskilld 2021]. Otherwise, the challenges and the overall setup of the two are very similar.

2.2 The Platforms

The following platforms were among the ones that were recommended when searching for ways to create a CTF event [Cybertalents 2023].

- Hack The Box [`https://www.hackthebox.com/`] - Wargames
- OverTheWire [`https://overthewire.org/wargames/`] - Wargames
- CTFd [`https://ctfd.io/`] - CTF Platform

They all have their own strengths and weaknesses, and I will go over the architecture and setup of each of them in the following sections, as well as why I ended up choosing to use CTFd for my CTF event.

2.3 HackTheBox

2.3.1 About the Platform

Hack The Box is a "massive hacking playground" [Hackthebox 2023] that delivers multiple platforms to train and compete in cybersecurity-based challenges. Of the available platforms, I chose to go over the architectures of Hack The Box (HTB) Labs as well as HTB CTF, but HTB also offers other platforms (i.e., HTB Academy, HTB Business).

2.3.2 Architecture & Setup

The architecture of the two platforms provided by HTB are fairly different. HTB Labs is set up as a training app, where the user can choose the categories in which they would like to attempt various levels of challenges. It also provides a path for the user, starting with the basics of cybersecurity and working upwards through three tiers of "machines", which are virtual machines with various vulnerabilities that the user will explore and exploit throughout the various challenges. HTB CTF, meanwhile, is a platform for CTF events, where the user can sign up for various CTF events and compete against other users/teams.

2.4 OverTheWire Bandit

2.4.1 About the Platform

OverTheWire is a website with several wargames of increasing difficulty and each with multiple levels of their own [Overthewire 2023c]. The one I chose to focus on is the first of said wargames, Bandit, due to it being recommended by OverTheWire (OTW) as a good starting point for beginners [Overthewire 2023a].

2.4.2 Architecture & Setup

Bandit has a simple and linear structure: Starting at level 0 users are asked to connect to a server via Secure Shell (SSH) [Overthewire 2023b]. From there on, the user is tasked with completing various challenges on the server, be it locating files or passwords to gain further access using the "flag" of the current challenge as a password to access the next one.

2.5 CTFd

2.5.1 About the Platform

CTFd is a platform for user and team based CTFs with built-in scoring tabulation and graphs [Cybertalents 2023]. It, unlike HTB Labs and OTW Bandit, does not provide any premade challenges, but rather serves as a platform for hosting fully custom CTFs. This means that the challenges must be written by the CTF host themselves, which can be a daunting task for those who are not familiar with the process. It does, however, mean that the challenges can be tailored to the specific needs of the CTF host, and that the challenges can be as simple or as complex as the host desires.

2.5.2 Architecture & Setup

CTFd's architecture is quite simple. It is a web application that can be hosted on a private server or CTFds own servers [CTFd 2023b]. The server is then accessed by the users via a web browser. They can then register for the CTF, and depending on the setup, join a team. Once the CTF has started, they can then start solving the challenges.

2.6 The Inspirations & the Choice

Or why neither HTB nor OTW were quite the perfect fit.

While HTB Labs and OTW Bandit are both great platforms for learning and practicing cybersecurity, they both only offer premade challenges. The CTF event I was tasked with designing and building, however, needed to be fully custom. This meant that I needed a platform that would allow me to create my own challenges, and that would allow me to host them on a private server.

Choosing CTFd came with a major challenge of its own, I would now have to manually create all the challenges for the CTF. This leads us into the next section, where I will go over the requirements for the CTF event, as well as the design and implementation of the challenges and the scoring system.

3 THE SYSTEM

3.1 Requirements

Being a project focusing on improving an already completed event, we will be going over the original requirements for the event, as well as the requirements for the improvement.

3.1.1 Original Requirements

The original requirements were simple and left a lot of room for exploring and experimenting with different ideas. The requirements were as follows:

- The event must cover topics/challenges relevant to IBE321.
- The event will be split into two parts; part 1 being integrated into an assignment, and part 2 being a separate event.
- The first event will focus on user authentication, providing 2 challenges for the students to complete.
- The second event will be a broader CTF event with a variety of challenges, with the goal of serving as a refresher for the students before the final exam.
- It should be hosted on a local server using hardware from the IT department, for cost reasons since remote hosting can be expensive.
- A bootcamp should be held before the event, to teach the students how to use the platform and how to solve the challenges.

From this, Simon Aarnes (TA) and I extrapolated the following more specific requirements:

- In the first event, challenge one should be about leaked credentials, and challenge two should be about SQL injection.
- The second event then becomes a superset of the first event, meaning it includes the first event, with the addition of challenges about web application security, ciphers, cryptography/hashing, network security, and steganography.
- The challenge total of the second event should be around 7-10 challenges.

3.1.2 Requirements for the Improvement

For the improvement, the focus is on the second event and turning it into the main (and only) event. The goal being to add more challenges, and to make the event more interesting and engaging. The requirements are split into two categories: functional and non-functional requirements. Functional requirements are the requirements that deal with the functionality of the system, while non-functional requirements deal with qualities of the system, such as performance, security, and usability [Martin 2023].

Functional Requirements

- The SQL injection challenge should verify the result of the SQL query being injected rather than the SQL input into the login field.
- There must be challenges based on Linux Directory traversal. Therefore, a Linux VM image must be created and handed out to students/users.
- The User Interface / User Experience must be altered. It currently is created using basic knowledge in CSS. Find a toolset to improve the UI.
- The CTF must be capable of handling over a hundred students at peak load.
- The user input on the website must be verified automatically and correctly. Any incorrect input must be properly rejected, and the user must be notified of the error.

Non-Functional Requirements

- The CTF setup/architecture should be OS independent.
- The current challenge count is 7. This should be increased to 10-14, ideally 14. They should be as relevant to IBE321 as possible.
- The bootcamp document should be rewritten, and either in its entirety or in a summed-up version should be added to actual CTF website/server for ease of access.
- The code should be commented and refactored to make it easier to understand and modify for future maintainers.

The goal of these requirements [1] was not only to make more challenges, but also to make some existing challenges better while also making it so that any future CTF administrator would have an easier time setting up and modifying the CTF.

3.2 Design

Before I could start implementing and programming the original CTF I needed a map of the CTF functionality to better give me an idea of any software and/or frameworks needed to create the CTF event. Using the web application Lucidchart, I sat down and created a simple diagram of how the CTF would be structured with some example tasks included for reference. Said diagram is shown in Figure 3.2.1.

[1]The requirements in the above lists were originally written down in the preliminary report from before this project was started.

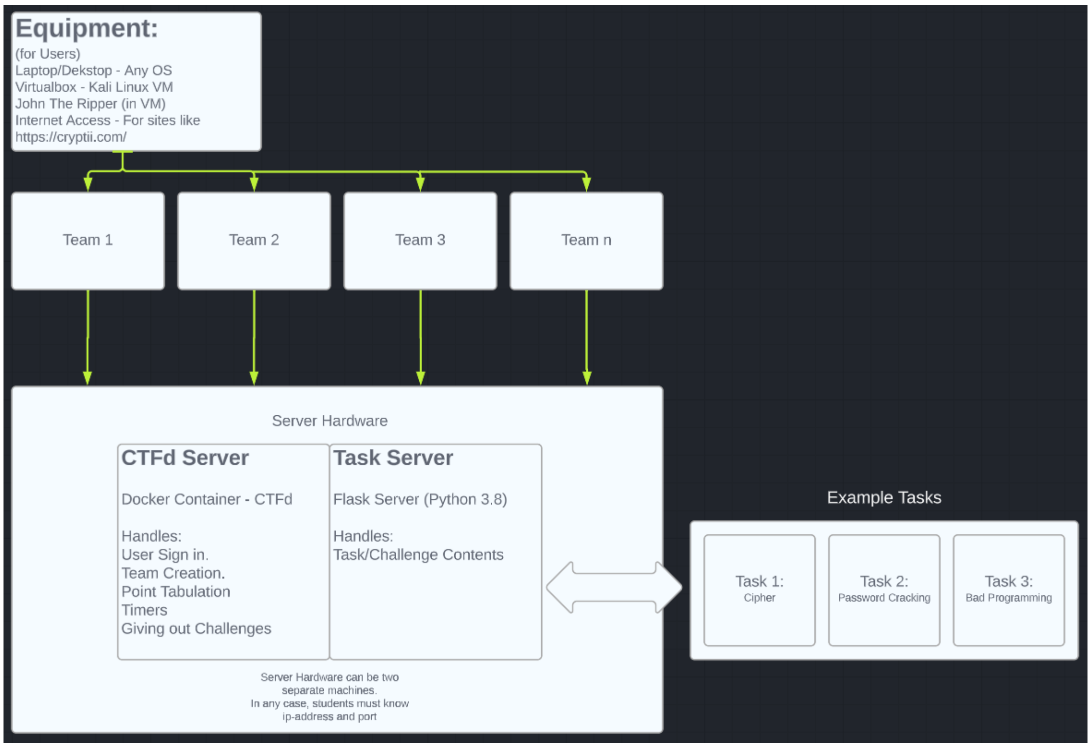


Figure 3.2.1: A diagram showing the proposed layout of the CTF event.

After creating a basic layout, I had to choose which type of CTF I wanted to create. According to CTFd 2022, there are two main types of CTFs: Jeopardy, Attack/Defense.

Jeopardy style CTFs are set up such that the challenges are split into various categories, where within each category there are challenges of increasing difficulty. Each challenge, once solved, will give the solver a flag, teams/or individuals are then given points for each challenge they solve by entering the flags for confirmation. Additionally, the more difficult a challenge is the more points it is worth. The team with the most points at the end of the CTF is the winner.

Attack/Defense CTFs are rarer than Jeopardy style CTFs due to them having more moving parts, and overall being more complex. Because of this, Attack/Defense CTFs are rarely open to the general public. In such a CTF, the teams are given the same set of vulnerabilities in the given software. The teams are given time before the competition to audit the software to discover said vulnerabilities. Once the competition starts, the teams join an isolated network and are tasked with exploiting the vulnerabilities in each other's servers/software, as well as patching vulnerabilities in their own servers. The teams are given points for finding flags in their opponents' servers, and for successfully defending their own flags. Points are also given for maintaining server operations at a nominal level.

Due to the complexity and technical skills needed of Attack/Defense CTFs, and the CTF being centered around the various topics of IBE321, I decided to go with the simpler Jeopardy style CTF.

Ultimately, the newer version of the CTF being created for this project will be much the same as the older version. The major difference in the design is in how the challenges are set up and unlocked.

In the original CTF all the challenges were open from the start of the CTF, this was something I wanted to change now that I had more experience with CTFd's tools. I wanted to make it so that the challenges would be unlocked as the users/students progressed through the CTF, with challenges of increasing difficulty being revealed as they complete the challenges. For example, to view a more difficult hashing challenge, one would have to first solve a simpler hashing challenge. This would make it so that students could not just skip to the more difficult challenges and would have to work their way up to them.

This setup does have a potential downside. If a student is stuck on a challenge, they will not be able to move on to the next challenge until they solve the current challenge. This could lead to students getting frustrated and giving up on the category or the CTF in its entirety. To combat this a hint system has been implemented. In this system, students/users can pay a certain number of points to get a hint for the current challenge. The hints that are given range from the form of short sentences whose goal is simply to guide the students, to any commands they might need for the challenge in question. Students/users can also only get a certain number of hints per challenge, usually 2.

If this downside is considered too great, the challenges can be made available from the start of the CTF using the admin account in CTFd.

3.2.1 Software/Frameworks

This project relies on multiple programming/markup languages, software, and frameworks to function. The following sections will cover the programming languages and frameworks used in this project.

3.2.1.1 Programming/Markup Languages

First, I will go over the programming and markup languages used in this project. These are the languages that were used to create and program the challenge server, its website, and its various components.

The programming languages used in this project are:

- Python
- HTML / Jinja2
- CSS
- JavaScript
- SQLite3

Python

Python is an object-oriented programming language that is used in this project to create the challenge/task server. It is a high-level language that is designed to be easy to learn and use. It is also designed to be very versatile, featuring a large standard library as well as allowing the importing of modules written in compiled languages like C++ [Python.org 2022]. Python is also a very popular programming language, being the fourth most popular language among programming, scripting and markup languages according to a Stack Overflow survey in 2022 [Stackoverflow 2022]. As such, it has a large community of developers and a large number of libraries and frameworks available for use [Coursera 2023].

Because of the information written above, and the fact that I had prior experience with Python from IB151 Practical Programming, I decided to use python to create the challenge server which would need to be able to parse user input to determine if challenges like the SQL injection ones were solved correctly. Python also allowed for the use of real SQL integration rather than just an HTML file with complex regex to determine if the user input was correct. Another reason for using Python was that it allowed for the use of the Flask framework which will be covered in a later section.

The version of python used in this project was initially python 3.8 due to Replit.com's template for Flask servers using that version at the time (Fall 2022). However, during a migration to a local server for development purposes, it was changed to python version 3.10. This led to the replit server being updated to match the local server via the use of Github integration.

HTML, CSS & JS

The challenge server (as any web app) is made up of HTML, CSS, and JavaScript files. HTML is used to create the structure of the webpages, CSS is used to style the webpages, and JavaScript is used to add interactivity to said HTML [Mozilla 2023b]. Using these three languages, one can create a fully functional and interactive web app.

HTML, short for HyperText Markup Language, is a markup language that is used to define the structure of webpages. It is made up of tags or "markup" that are used to define the structure of the webpage. For example, the tag `<p>` is used to define a paragraph, and the tag `<h1>` is used to define a heading. HTML is not a programming language, and as such does not have any variables or functions. It is purely used to define the structure of the webpage. [Mozilla 2023b].

CSS, or Cascading Style Sheets, is a stylesheet language. Its purpose is to style or "decorate" the HTML elements on a webpage. It is used to define the colors, fonts, sizes, and other visual aspects of the webpage. [Mozilla 2023a].

JS or JavaScript is a programming or scripting language that is used to add interactivity to webpages. Examples of this are more complex animations, clickable buttons, pop up menus, and more. JS can be client-side and/or server-side. Client-side JS is run by the user's browser, and server-side JS is run by a server/runtime like Node.js [Mozilla 2023c]. While client-side JS affects the browser and the HTML document, server-side JS allows an application to do things like for example connecting to a database, or write to a file, and can work as a backend for a web application.

In this project, JavaScript is client-side, with the server side (backend) being handled by Python and the Flask framework instead.

Jinja2

Jinja2, or just Jinja, is an HTML templating engine. It lets programmers create a rich frontend without using a JavaScript framework/library like React or Vue [Acsany 2022]. Jinja allows the programmer to use variables and loops inside the HTML files. Or rather, the variables and loops are in the template file, which can be called anything be it .html or .j2 [Docs 2023]. Jinja also lets programmers avoid repeating a lot of "boilerplate" or repeating HTML as one can create a main file and then use it in other jinja files. [Docs 2023] These template files are then rendered by the server (in this case Flask), and the variables and loops are replaced with the actual values and data before sending the rendered HTML file to the end user. This allows for the creation of dynamic webpages that can be easily changed based on user activity without having to write values or data "by hand" into the HTML files.

SQLite3

SQLite is a relational database management system (RDBMS) that is used by this project to provide a target for the SQL injection challenge. The main advantage of SQLite over other RDBMSs is that it is a lightweight self-contained database engine with no configuration required. This means that it does not require a separate server to run. This makes it very easy to set up and use [SQLite 2023].

To use SQLite with the Flask/Python server the module SQLite3 is used. SQLite3 is a Python module that allows for the use of SQLite databases in Python. It lets Python create, read, update, and delete data from SQLite databases [Python.org 2023a]. This lets the challenge server create a database with the correct tables and columns, and then populate it with the correct data. This data then becomes the target of the SQL injection challenge.

3.2.1.2 Frameworks & Software

Now that I have covered the programming and markup languages used in this project, I will go over the frameworks and software used in this project. These not only include the frameworks used to create the challenge server, but also the software used to host the challenge server and the CTFd server, as well as the VM solution used for the Linux CLI challenges.

The frameworks and software used in this project are:

- Flask
- Replit
- Poetry
- CTFd: Covered in section 2.5 & 2.6.
- Docker & Docker Compose
- VirtualBox

Flask

Flask is a web framework for Python. It actually calls itself a microframework, meaning it does not come with a lot of features one would expect from a full framework, like form validation, upload handling, etc. This makes it so that it is very lightweight and easy to use out of the box [Brmwebdev 2015]. While it does not come with a lot of more advanced features pre-installed, developers can easily install extensions to add these features as they need them. Because of this, Flask is not only easy to use, but is also very extendible. This allows programmers to scale it up as they want for more complex web applications [Pythonbasics 2021].

Since I needed to write code that was purposely unsecure for the SQL challenge, Flask, being a microframework, was therefore a perfect fit for the challenge server. Flask was also chosen because it has a lot of documentation and tutorials online, which let me learn how to use it quickly and easily.

This project uses Flask 2.2.x (third number is for bugfixes), which is the latest version of Flask at the time of writing this report (Spring 2023).

Replit

Replit is an integrated development environment (IDE) that runs in your web browser and supports over 50 programming languages [Replit 2023c]. It also comes with many community created templates which are ready-made environments for various languages and frameworks, like for example Python/Flask. Replit also allows for the hosting of the sites and web apps their users create and even provides a full app deployment service should the user need the extra performance [Replit 2023b & Replit 2023d].

Replit was chosen due to Himolde's VPN solution blocking the use of SSH on the school's network for security reasons. This meant that while I could give one of the TAs in IBE321 access to the challenge server GitHub repository and have them install it on a local server connected to the school network, I could not update the server quickly upon discovering a bug. Replit allowed me to host the challenge server on an external machine, completely bypassing the SSH issue, which allowed me to update the server as soon as a bug was fixed.

Poetry

Poetry is an alternative to the built-in pip package installer in Python. Its purpose is to install packages and manage those packages' dependencies [Python-poetry 2023a]. One of its core features is that it isolates each project into virtual environments [Python-poetry 2023b]. What this achieves is that each project has its own dependencies and that those dependencies are not shared with other projects. This means that if one project requires a specific version of a package and another requires a different version of the same package, they can both be installed without any issues. Being installed in separate virtual environments means that they do not interfere with each other. This is very useful when working on multiple projects simultaneously. It allows programmers to use different versions of the same package in different projects without worrying about them interfering with each other [Python.org 2023b].

Poetry also has many other features, like a lock file that ensures that identical versions of the dependencies are installed on all project machines and a dependency resolver that ensures that all dependencies are installed correctly [Python-poetry 2023a].

I did not choose to use poetry as it came with the Replit flask template. The template uses poetry to manage the dependencies and virtual environment for the project. I had no issues with poetry, so I decided to stick with it for the rest of the project.

Docker & Docker Compose

Docker is an open-source platform that lets developers utilize containers. Containers are a way to package software into standardized units that can be run anywhere. This means that developers can package up an application with all of its dependencies and run it on any machine that has Docker installed. This makes it so that the application will run the same on any machine, regardless of the machine's configuration [IBM 2023].

Docker Compose is a tool that lets users define and run applications that utilize multiple containers. It does this by using a YAML file to configure the application's services. This means that the user can define the services that make up the application and how they interact with each other. This makes it so that the user can run the entire application with a single command [Docker 2023a]. That command is "docker-compose up" which will start all the services defined in the YAML file. This makes it so that the user does not have to start each service individually, which can be a hassle if the application has many services [Docker 2023b].

Docker and Docker compose were chosen as it was one of the quickest ways to set up a CTFd server. It also made it easy to update the server when a new version of CTFd was released. All you have to do is delete the containers and images installed by docker, then run "docker-compose up -d" again after using "cd" to enter the CTFd directory. This downloads the latest version of CTFd and installs it without any loss of data from the server. Data loss is avoided by the docker-compose script setting up docker volumes, which is a way to store data outside the container. What this means is that the data is not deleted when the container is deleted [Docker 2023c].

VirtualBox
VirtualBox is a free and (partially) open-source virtualization software that lets users run virtual machines on their computers. This means that users can run multiple operating systems on their computer at the same time [Oracle 2023].

VirtualBox was chosen for this project (specifically for the Linux CLI challenges mentioned in 3.3.2) as it was already used by IBE321 where the students were to use a Kali Linux VM for a lab/canvas module. This meant that the students were already familiar with VirtualBox and how to use it, saving time as they would not have to learn how to use a new virtualization software.

3.2.2 Server Setup

Having covered the software technologies used in the project, this section will cover the physical setup of the CTFd server and the challenge server.

During the original run of the CTF two servers were requisitioned from the IT department. One of those servers was used to host CTFd and the other was used to host the nmap target (one of the challenges). As the servers are currently in storage and not accessible, this section will cover what the proposed setup would be like if the CTF was to be run again.

The specifications of the first server are:

- Intel Xeon
- 32 GB of Ram
- 120 GB of storage
- Ubuntu Server 22.04 LTS

The specifications of the second server are:

- Intel i5
- 16 GB of Ram
- 120 GB of storage
- Ubuntu Server 22.04 LTS

3.2.2.1 CTFd Server

The CTFd server will be hosted on the Intel Xeon based server due to the fact that it has more RAM and a better CPU. This will allow the server to handle the traffic of dozens of students trying to sign up, login, and submit flags at the same time. The server will be connected to the school's network, which means that the students will have to be connected to the school's network to access the server. This is not an issue as the CTF is only meant to be played by students of Himolde. If any students are not on campus, they can use the school's VPN to access the server.

The server will be running Ubuntu Server 22.04 LTS as it is the latest version of Ubuntu Server at the time of writing. It will also be running docker and docker compose to host the CTFd server.

3.2.2.2 Nmap Target Server

Utilized by the Nmap challenge, which will be covered in section 3.3. Though this server could most likely be run on a Raspberry Pi, it will be run on the Intel i5 based server. This server will be running a simplified setup of CTFd whose only function is to display a web page with the flag upon students figuring out the solution to the challenge.

While setting up this server, some care should be taken to make sure to change the docker-compose.yml file, with the one provided in Appendix A.2. This yml or yaml file has been altered slightly to make the server use non-standard ports (i.e., not port 80).

3.2.2.3 Challenge Server

The challenge server will be run externally to the school's network, with the server being hosted on replit, as mentioned earlier in the report.

3.2.2.4 Alternative Setup

An alternative setup to the one described above would be to run the nmap target server being run on a third much cheaper/weaker machine, since as mentioned earlier, it is not necessary for the server to be very powerful. This would allow the challenge server to be run on the i5 based server, provided that the CTF admin would have SSH access, so they could update the server in the event of a bug being found in a challenge.

3.3 Challenges/Tasks

There are 17 challenges (1 challenge is for starting the CTF). Because of this only the challenges that are unique will be covered in this section. The challenges that are not covered are just variations of challenges that will be covered in this section.

Challenges in CTFd can also be categorized into different categories. Therefore, please keep in mind that some sections in this chapter will be named after the category of the challenges when covering multiple challenges at once.

3.3.1 Cryptography Basics

The challenges of this category are meant to teach the students the basics of cryptography. These challenges are fairly simple and involve giving the students a ciphertext or a hash and either asking them to decrypt it or to figure out the hashing algorithm which was used to create the hash. This category also contains the steganography challenge which involves the students having to find a hidden message in an image, with the image being the large Himolde logo displayed in the challenge server.

Another challenge that is not in this category but could be considered adjacent is the Base64 challenge in which the students have to decode a base64 encoded string.

While the category “Cryptography Basics” existed during the original run of the CTF, there were no challenges on hashing algorithms in it.

3.3.2 Linux CLI

The challenges of this category are a series of tasks that involve giving the students a Linux VM and having them navigate the VM using the command line interface (CLI). These challenges are meant to teach the students the basics of the Linux command line.

This category did not exist during the original version of the CTF.

3.3.3 Pcap Forensics

In this challenge, from the original CTF, the students are given a pcap file which is a file that contains a recording of network traffic captured by the application Wireshark. The students are then asked to search the pcap file for the flag. This challenge aims to teach the students how to use Wireshark to analyze network traffic.

3.3.4 Mapping Things Out

“Mapping Things Out” is structured around the students using the tool nmap to scan the provided IP address for any open ports. The students must then enter the IP address with the open port to find the flag. This challenge’s goal is to teach students that even though a port is not advertised to users, it does not mean it is not open to everyone.

This challenge is from the original CTF, but it has been planned to be improved by changing the ports of the server to non-standard ports. Changing the ports can only be done once access to the servers is granted to the CTF admin.

3.3.5 User Authentication

In this challenge (from the original event), the students are given a file with several hashed passwords and are asked to crack them. The hashes are weak passwords that have been hashed without salting, which the students are asked to crack. Though the students are not required to use a specific tool, the tool, which is shown in the bootcamp, is John the Ripper. This challenge is meant to teach students the dangers of using weak passwords.

Note: Salting is a technique that makes it harder to crack passwords. It involves adding a random string of characters to the password before hashing it. This makes it so that even if two users have the same password, their hashes will differ [Omnicybersecurity 2023]. Salting would make this challenge infeasible to solve.

3.3.6 Query

The category “Query” is a set of two challenges that involve the students using SQL injection to break through a login form and retrieve the flag. The first challenge asks the student to retrieve the flag by gaining access to a specific admin user. The second challenge asks the students to retrieve the flag by retrieving all users from the database.

Through this challenge, students learn why it is important to sanitize user input and how easy it is to exploit a website that does not do so.

This was originally just one challenge in the first run of the CTF. Additionally, the challenge was not working properly as the implementation was flawed. This was a major reason why this project to improve the CTF was started.

3.3.7 Server/Path Traversal

This challenge aims to simulate the dangers of allowing user input to dictate which files are accessed by the server. The challenge involves the students having to find a file that is not supposed to be accessible to the user: the shadow file of Linux systems. If the student inputs the correct file path using the technique taught in the bootcamp, they will be given the flag.

3.3.8 Various Challenges

There are also a couple of challenges that concern topics related to website security. These challenges are not part of any major categories but are still important to mention. The first of these is a challenge that showcases that if a URL is not properly protected by user authentication, an attacker can simply change the URL to access the page.

The second challenge is aimed at teaching students the powers of developer tools in modern browsers. The challenge involves the students having to find a password in an HTML comment which gives them access to the flag.

3.3.9 The Challenge Improvements

The original run of the CTF, while functional, had some issues that needed to be addressed. These issues were mostly related to things like the SQL challenge not working properly or there being too few challenges. Therefore, as you will see in the following sections on the implementation, the challenges have been improved upon and the challenge count has been doubled.

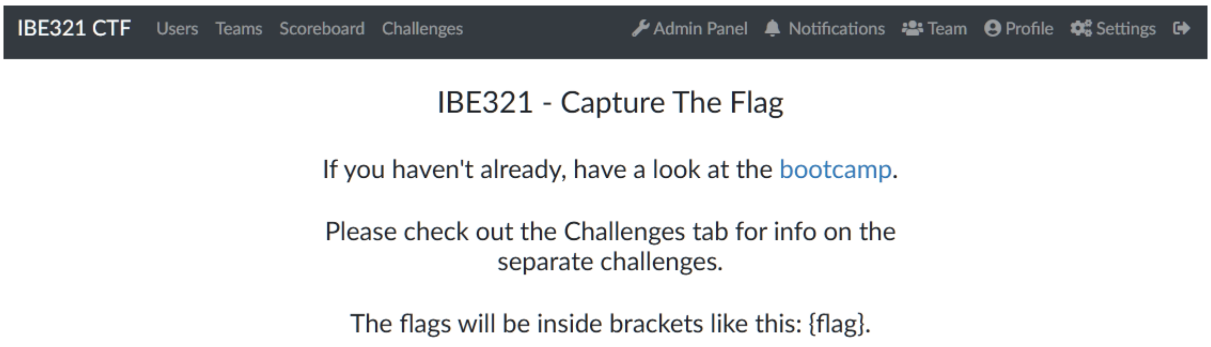
3.4 Implementation

3.4.1 Application Structure

As mentioned in section 3.3, the challenges have to be custom-made to better fit the topics taught during IBE321 Information Security and Risk Management.

Because of this, the CTF setup is split in two. The first part is the CTFd server, which handles user login, point tracking, team effort, flag confirmation, etc. The second part is the custom challenge server which handles things like giving out cipher texts and hashes, to displaying sign in forms that the students somehow have to break through.

CTFd



IBE321 - Capture The Flag

If you haven't already, have a look at the bootcamp.

Please check out the Challenges tab for info on the separate challenges.

The flags will be inside brackets like this: {flag}.

Figure 3.4.1: A screen capture of the CTFd front page, with the admin account signed in.

As seen in figure 3.4.1, the CTFd front page is fairly simple and straight to the point. It points the students towards the bootcamp document in case they haven't already read it or need a refresher. It also tells the students which tab to click to view the challenges and what they can expect flags to look like.

Keeping the front page simple in the improved version was a conscious decision since if the front page is too cluttered with information, the students might not read it.

In figure 3.4.2 we see the CTFd registration page which is one of the reasons why the CTF application was split in two. Since CTFd comes with a built-in registration/login system, a new custom way to register students would not have to be created. This way, the admin can focus on creating the challenges instead of having to worry about creating a secure registration system.

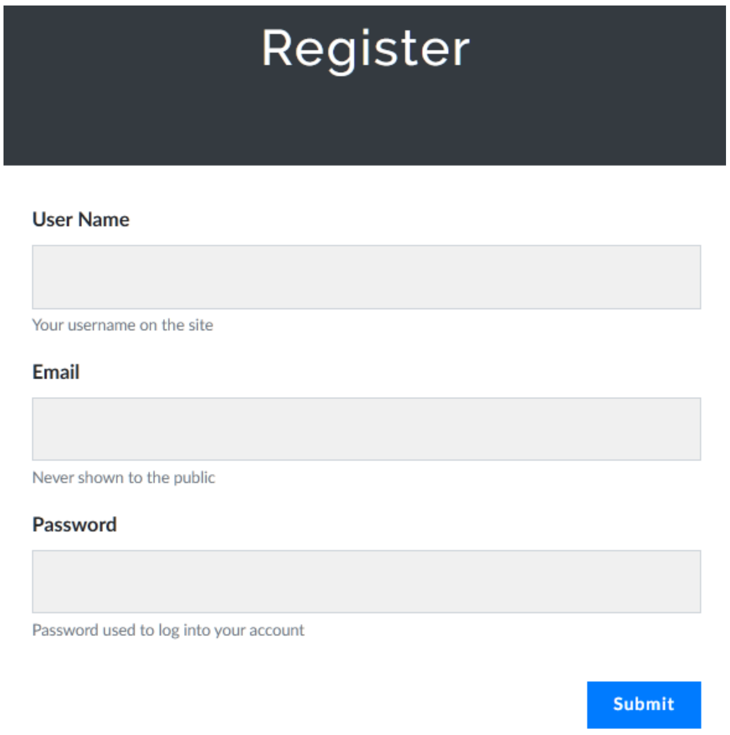
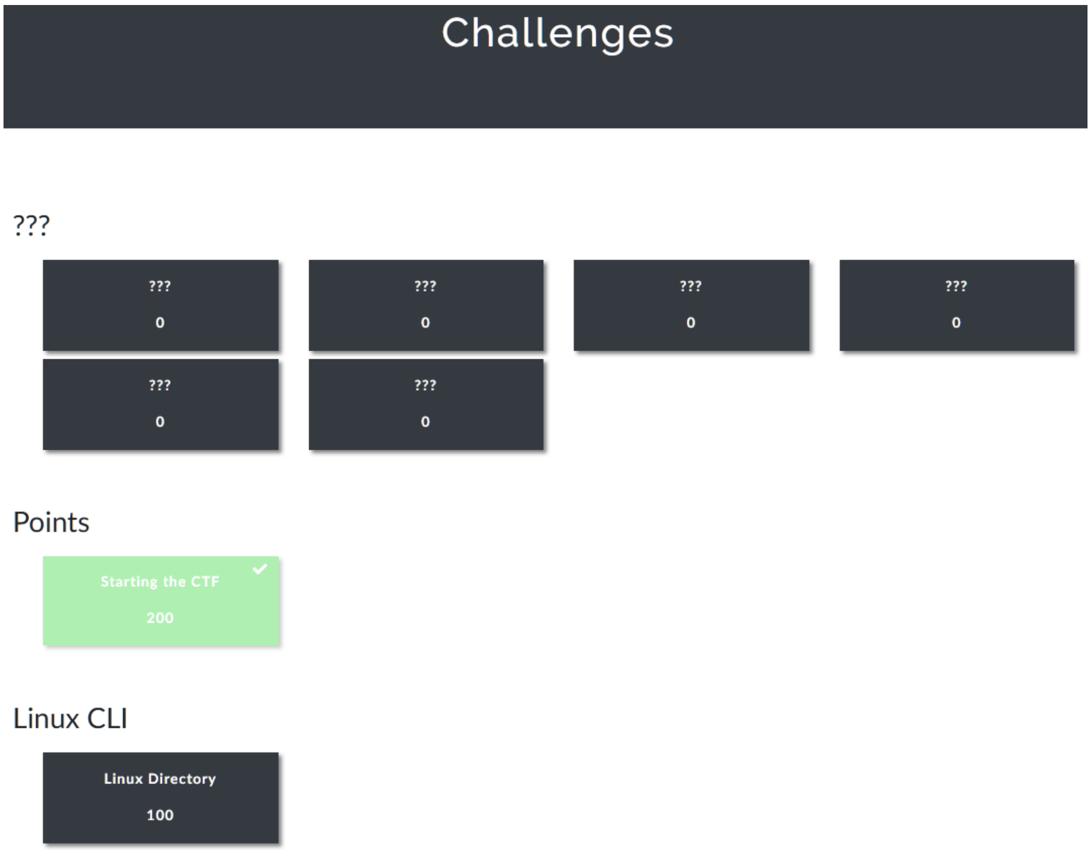

Register
User Name
Your username on the site
Email
Never shown to the public
Password
Password used to log into your account
Submit

Figure 3.4.2: A screen capture of the CTFd registration page.


Challenges

???

??? 0	??? 0	??? 0	??? 0
??? 0	??? 0		

Points

Starting the CTF ✓
200

Linux CLI

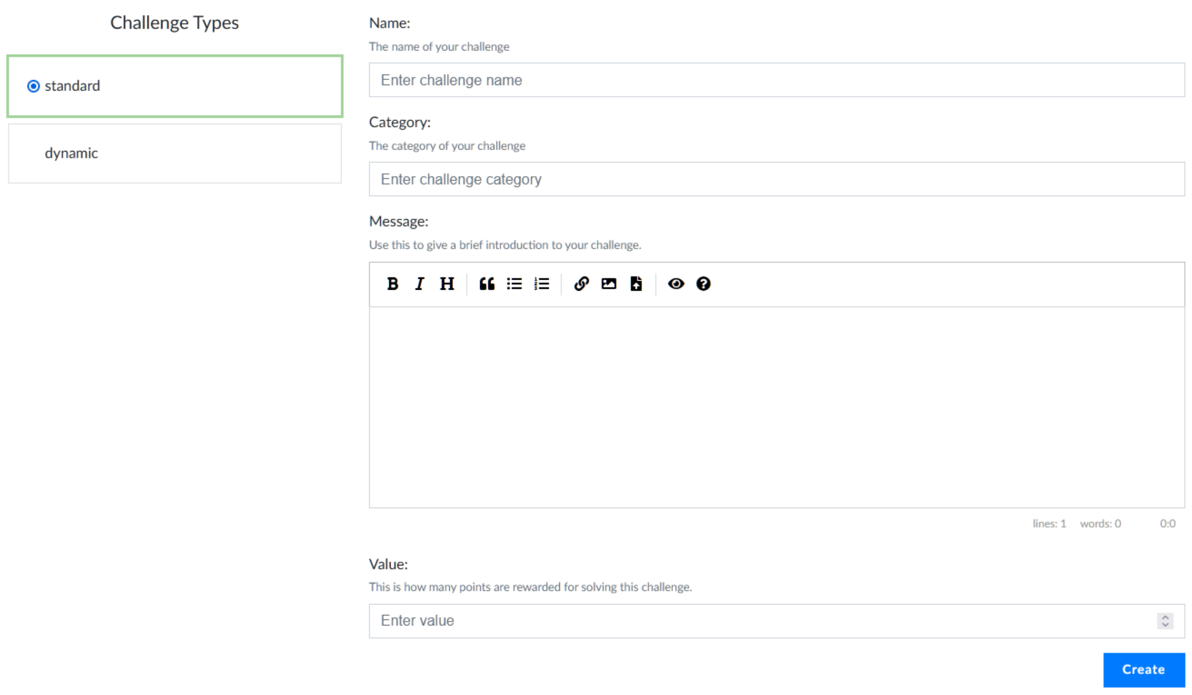
Linux Directory
100

Figure 3.4.3: A screen capture of the CTFd challenge page after the CTF has been unlocked by clicking the "Starting the CTF" challenge.

Figure 3.4.3 showcases that the challenges are locked until the "Starting the CTF" challenge has been solved. This is to ensure that the students have points to spend on unlocking hints (which costs points) before they start solving the actual challenges (more on this in section 3.4.4).

Also, as can be seen in 3.4.3, some challenges are still anonymized as they are not yet unlocked. This is to prevent students from solving challenges that are of a higher level than they are currently at. As students solve challenges they will gradually unlock more of the anonymized challenges.

Note: CTFd refers to it as "anonymized" in the admin challenge creator when a challenge is listed but with the text replaced with question marks. This is why the term is used here. If the admin chooses the "hidden" option, the challenge will not be listed at all until unlocked.


Challenge Types

[✓ standard]

dynamic

Name:

The name of your challenge

Enter challenge name

Category:

The category of your challenge

Enter challenge category

Message:

Use this to give a brief introduction to your challenge.

lines: 1 words: 0 0:0

Value:

This is how many points are rewarded for solving this challenge.

Enter value

Create

Figure 3.4.4: A screen capture of the CTFd challenge creation screen

Figure 3.4.4 shows the challenge creation screen. Here CTF admins can add more challenges should they wish to, and they can also edit the properties of existing ones. The fields in the figure are not the only ones that can be changed. Once the challenges are created, the admin gains access to several more fields that can be changed. These include things like requirements for previous challenges to be completed, hints, tags, etc.

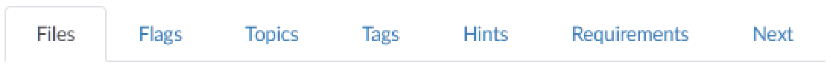
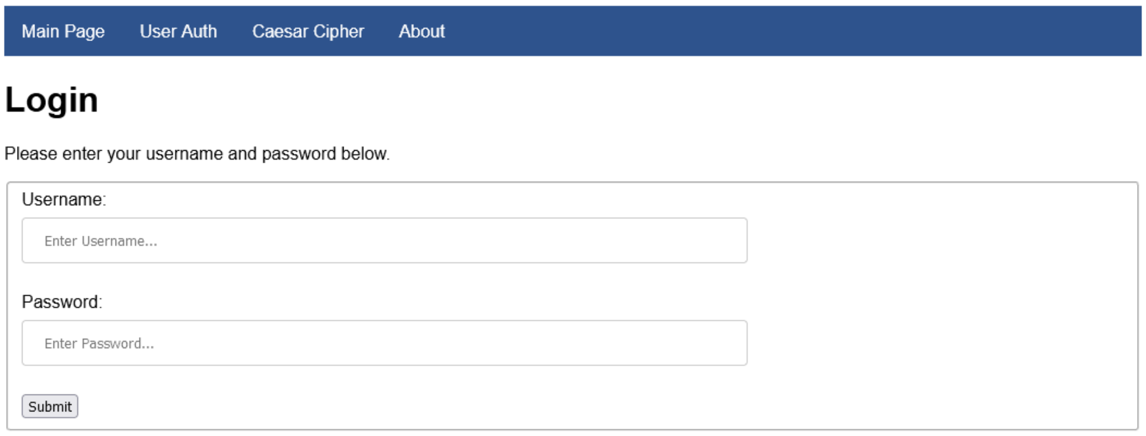


Figure 3.4.5: A screen capture of the CTFd challenge edit tabs.

The next tab allows the administrator to recommend a challenge to the students if they have solved the current challenge (which is being edited). This feature has not been used in the improved CTF. In the future, provided more challenges have been added, this feature could be used to guide students for an improved experience of the difficulty progression.

Challenge Server



Login

Please enter your username and password below.

Username:

Enter Username...

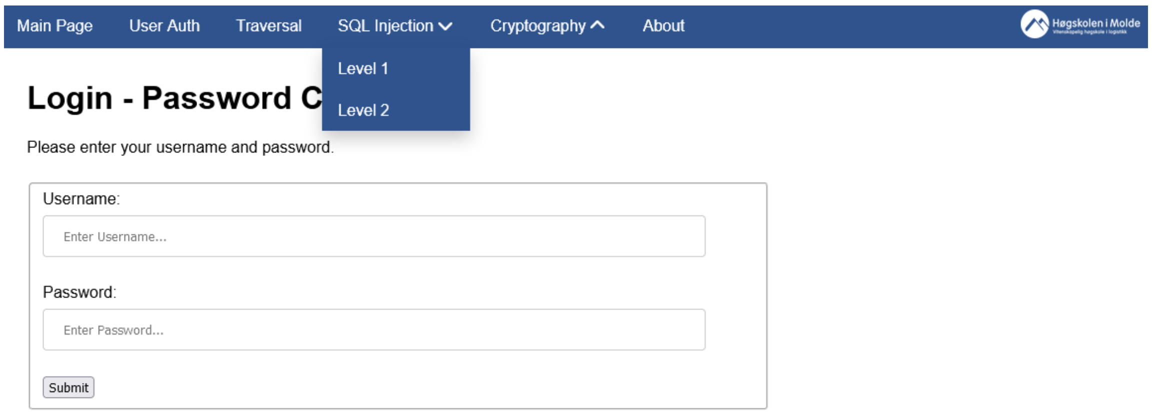
Password:

Enter Password...

Figure 3.4.6: The old version of the challenge server. This view is of the joint password cracking and SQL injection login form.

Figure 3.4.6 shows the old version of the challenge server. This figure is mostly included for comparison purposes. The new version of the challenge server is shown in figure 3.4.7.

One of the points to go over about the old version though, is the tab “User Auth”. This tab was perhaps in a poor decision, made to handle both the SQL injection and the password cracking challenges. Originally, this was done as the teacher of IBE321 wanted some realism or a storyline to the challenges. However, as development progressed, these storylines mostly fell away and the challenges became more about the technical aspects of the challenges and were to serve as refreshers that explained why said challenges where being done (for ex. see fig. B.1.1 in appendix B.1). Some of these storylines are still present in the form of some “flavor text” in some challenges.



Login - Password C

Please enter your username and password.

Username:

Enter Username...

Password:

Enter Password...

Submit

Figure 3.4.7: The new and improved version of the challenge server. In this version the challenges mention in figure 3.4.6 have been split into their own separate login forms. Now with levels of SQL injection.

As seen in figure 3.4.7. The overall theme or look of the challenge server has remained largely unchanged in the new version of the server. The main difference is that the challenges have been split into their own separate login forms. This was done to make the challenges more modular and easier to separate in the backend code. It also allowed for the addition of more levels of SQL injection challenges.

For a side-by-side comparison of the old and new versions of the challenge server, see appendix B.2.

Frontend/Backend

For clarity's sake in case the reader is not familiar with the terms: the frontend refers to the part of the website that the user interacts with, while the backend refers to the code that runs on the server and handles the requests from the frontend [Simmons 2023]. In this case, the frontend is the visual aspects of the challenge server (i.e., the login forms) which is written in HTML, CSS, and JavaScript. The backend is the Python/Flask code that handles the requests from the frontend and interacts with the database.

3.4.2 Code Examples

File Structure

To avoid the main python file becoming too large, the code for the challenges is split loosely categorically into their own files. An overview of the file structure is included in a file called "README_FOR_DEV.md" which can be found in appendix A.1. But for the files directly related to the challenges, the file structure is as follows:

- **main.py:** The main file that runs the Flask application.
- **base.py** For challenges/pages in no specific category.
- **helper.py:** Contains helper functions that are used by multiple challenges.
- **dbconnection.py:** Sets up the custom database class used by code that requires database access.
- **pass_crack.py:** Contains the code for the password cracking task.
- **sql_injection.py:** Contains the code for the SQL injection challenges.
- **crypto.py:** Contains the code for the cryptography category.
- **traversal.py** Contains the code for the server/path traversal challenge.

There are also a couple of files that are not directly related to the challenges and relate more to maintenance of the server database. They can be found in the GitHub repository in appendix A.1.

Database Code

```python
import sqlite3

class Database(object):
        """Custom database class to handle database connections."""
        # Automatically open the database connection
    def __enter__(self):
        self.conn = sqlite3.connect("user_db.db")
        return self

        # Automatically close the database connection
    def __exit__(self, exc_type, exc_val, exc_tb):
        self.conn.close()

    # empty placeholders by default to allow for unsafe queries (sql injection)
    def __call__(self, query, placeholders=""):
        cursor = self.conn.cursor()
        try:
            result = cursor.execute(query, placeholders)
            self.conn.commit()
        except Exception as ex:
            result = ex
        return result
```

Code 1: Custom Database Class used by all challenges that require database access. Taken from dbconnection.py.

Repeatedly calling for a database connection and cursor is tedious and can lead to errors. To avoid this, a custom class (see Code 1) was created that handles the database connection and cursor for the application. This class is used by all challenge code that requires a database connection.

The class is used as a context manager, which means that it can be used with the "with" keyword [Ramos 2023]. The context manager allows for the database connection to be closed automatically once the code inside the "with" block has finished executing. This is done by the "__exit__" method, using .close() to close the database connection. The "__call__" method is used to execute the SQL queries. This method also allows for the use of placeholders to prevent SQL injection attacks, but by default this parameter is empty to allow for true SQL injection attacks in the applicable challenges.

User Authentication Code

```
1
2  with Database() as db:
3          # * Verify input / Give user their result
4          # Secure against SQL injection
5          result = db(
6                  "SELECT * FROM users WHERE name=? AND password=?", (inpUser, inpPass)
7          )
8          result = result.fetchall()
9          print(result)
10
11         if result is None:
12                 return render_template("fail.j2")
13
14         # * password cracking challenge
15         elif result[0][1] == "Jonathan":
16                 return render_template("pass_crack/hintUser.j2", username=hintUser)
17         elif result[0][1] == "Roger":
18                 return render_template("pass_crack/success.j2", username=storedUser)
19         # * check for bad programming challenge
20         elif result[0][1] == "secUsr":
21                 return render_template("pass_crack/super_secret.j2", username=secUser)
22
23         # * fail condition
24         else:
25                 return render_template("fail.j2")
```

Code 2: Part of the User Authentication challenge code. Taken from pass_crack.py.

Code 2 shows part of the code responsible for checking the user input against the database to confirm that the students correctly cracked the passwords in the file given to them by the CTFd server.

As seen in line 5 to 8 the code takes the user input (assignment to inpUser, and inpPass not shown), and properly passes it to the Database class (seen here as db). Since this challenge is not about SQL injection, the code is secure against SQL injection attacks. From there the result is fetched from the database and stored in the variable “result”. The code then checks if “result” contains the correct output from the database. If the output is correct, the applicable template is rendered by the server, and the flag is displayed to the student. If the result is incorrect, the fail template is rendered instead.

As can be seen in one of the comments in the code (marked by a #), the part of the code with the last elif also checks for the correct input for another one of the challenges.

SQL Injection Code

```python

def uauthProtect(inp):
    """Checks for sql statements in user input. Returns touple of
    True/False and the statement if True."""
    if drop := dropProtection(inp):
        return True, drop
    elif insert := insertProtection(inp):
        return True, insert
    elif update := updateProtection(inp):
        return True, update
    elif or_ := orProtection(inp):
        return True, or_
    else:
        return False, None
```

Code 3: Part of the SQL Injection challenge code for blocking certain SQL commands. If one of the variables (drop, or_, etc.) are not empty/None, it returns True, and is used to block unwanted SQL queries. From helper.py.

```python

from main import re          # main is main.py

def dropProtection(inp):
    """Checks for DROP statement in user input."""
    if re.search("drop", inp, re.IGNORECASE) is not None:
        print("'DROP' DETECTED")
        return True
```

Code 4: One of the functions called by Code 3. Uses regex to search for the word “drop” regardless of capitalization. From helper.py.

The code in Code 3 shows the function located in helper.py (a file containing reused functions), that blocks certain SQL commands from being executed. This code is used in the SQL injection challenge to prevent the students from using certain SQL commands to damage the database. As you may have noticed it also calls for its own set of functions defined elsewhere in the helper.py file. This function is called just prior to the code in Code 5.

This function is not the only such piece of code in the helper.py file and is an example of the code being written to follow the Don’t Repeat Yourself (DRY) principle [Fitzgibbons 2018]. The principle states that code should be written in such a way that it can be reused in places where it otherwise would be repeated. This could be done via the use of callable functions or classes, which makes the code easier to maintain and update, as well as easier to read. One important effect of DRY is that once a piece of code is updated, it is updated everywhere it is called, which means that the code is less likely to contain overlooked bugs.

```python
1  with Database() as db:
2              # * Verify input / Give user their result
3              result = db(
4                  f"SELECT * FROM users WHERE name='{inpUser}' AND password='{inpPass}'"
5              )
6              result = result.fetchall()
7              print(result)
8  
9              if result is None:
10                 return render_template("fail.j2")
11 
12             # * password cracking challenge blocker
13             if ans := antiPassCrack(result[0][1]):
14                 return ans
15 
16             # * root'-- or admin'-- sql injection challenge
17             elif result[0][1] == "root" or result[0][1] == "admin":
18                 if result[0][1] == "root":
19                     return render_template("sql_injection/root.j2", username="root")
20                 elif result[0][1] == "admin":
21                     return render_template("sql_injection/root.j2", username="admin")
22                 else:
23                     return render_template("fail.j2")
24 
25             # * fail condition
26             else:
27                 return render_template("fail.j2")
```

Code 5: Part of the SQL Injection challenge code. From sql_injection.py.

Code 5 shows part of the code responsible for giving the user input to the pre-written SQL statement. This code is intentionally vulnerable to SQL injection attacks and is the code that the students are supposed to exploit in order to solve the challenge. What it does wrong, on purpose, is to directly insert user input into the string that is used to query the database (see line 4). This means that if the user input contains a SQL statement, that statement will be executed by the database. This is an easy mistake to make, explaining in part why it is a common vulnerability.

As can be seen on line 13, the code checks for if the input SQL injection targeted a username from the password cracking challenge. If it did, the code returns a short message showing that the students are on the right track (the SQL is therefore correct), but that they have targeted the wrong username. The target for the challenge is an admin user, not a regular user.

When it comes to the SQL vulnerability, what should be, and was done in the User Authentication challenge, is to use parametrized queries [Vishal 2021] as shown in Code 6. Note that SQLite uses question marks (?) as a placeholder for the parameters, while other databases may use other symbols like (%s).

```
result = db(
                "SELECT * FROM users WHERE name=? AND password=?", (inpUser, inpPass)
            )
```

Code 6: Example of a parametrized query, used in the User Authentication challenge code.

Path Traversal

```

@app.route("/traversal/confirm", methods=["get"])
def traversal_confirm():
        inp_path = request.args["file"]

        # count number of ../
        count_dots = inp_path.count("../")
        # count number of /etc/shadow
        count_path = inp_path.count("/etc/shadow")
        #checks if last two directories are etc and shadow
        check_last = inp_path.split("/")[-2:] == ["etc", "shadow"]
        # checks if path contains any other directories than ../, etc and shadow
        check_random = not any(
                [x for x in inp_path.split("/") if x not in ["..", "etc", "shadow"]]
        )

        if count_dots >= 3 and count_path == 1 and check_last and check_random:
                return render_template("traversal/directory.j2", usrInp=inp_path)
        else:
                return render_template("traversal/dir_fail.j2", usrInp=inp_path)
```

Code 7: Part of the Server Traversal challenge code. Taken from traversal.py.

In Code 7 we see the code responsible for checking the user input against the correct path. The code checks for the correct number of “../” and that the path contains “/etc/shadow”. It also checks that the path ends with “etc/shadow” and that the path does not contain any other directories than “../”, “etc” and “shadow”. This should adequately simulate a real-world path traversal vulnerability, with some limitations.

The aforementioned limitations, and reasons for doing these checks rather than actually implementing an insecure file viewer, will be discussed in section 3.4.4.

3.4.3 Database & Admin Tools

Due to the inherent risks of implementing an SQL injection challenge, the database needed to be easily recreated in the event that a student managed to alter or damage it in some unforeseen manner. This was done by creating a simple script that would recreate the database and populate it with the necessary data. This script is to be manually run by the CTF admin upon discovering any issues with the database. The script is called “dbreset.py” and is located in the root folder of the project.

To complement this script, another simple script is included in the project. This script “dbcheck.py” prints out the contents of the database, allowing the CTF admin to quickly check database integrity.

3.4.4 Challenges Faced during the Implementation

CTFd: Cost of Hints

A minor issue with the original and new version, is the way the cost of hints is handled. The way CTFd handles hints (with costs) is that the user must “pay” with their already acquired points to unlock the hint. This means that if the user has not solved any challenges, they will not be able to unlock any hints as a user cannot have negative points [CTFd 2023a]. This is an issue as the students will not be able to unlock hints for the first challenge that they are trying to solve.

To solve this, the suggestion by the CTFd documentation (a “gimmie” challenge) was followed. This resulted in the creation of the challenge called “Starting the CTF”, which exists solely to give the students a starting set of points to spend on hints.

This solution was used in the original run and was kept in the new version with an improvement to usability. The improvement is to make “Starting the CTF” the first and only challenge that the students will see when they open the challenges tab for the first time, unlocking the other challenges once the students have entered its flag (written in the challenge description).

Flask: SQL Challenge

The SQL challenge(s) was an issue during the original run, when poorly implemented with regex, and continued to be an issue during the new version. The issues faced during the improved implementation was that when using proper SQL, the students would be able to break the database by using SQL commands like “DROP” or “INSERT”. This was solved by implementing a function that checks the user input for these commands and blocks them from being executed. This function is shown in Code 3.

Flask: Path Traversal

This issue remains unsolved. The issue is that the path traversal challenge is not a true path traversal vulnerability. Instead, it merely simulates a path traversal. It does this by checking the user input against the correct path, as shown in Code 7. This is due to the fact that a true path traversal vulnerability would be too dangerous to implement as it would allow the students to access the actual shadow file of the server.

Nmap Target Server

The original implementation of the challenge was actually a last-minute addition to the CTF. The challenge was added to the CTF as the teacher of IBE321 wanted to include a challenge that involved the students using nmap to scan a server for open ports. As this was done in between the two original runs, the first CTFd server was reconfigured to host the nmap target server.

While this worked, it wasn’t particularly elegant as it still used the default port 8000, the very same port used by the main CTFd server that the students would connect to.

The solution implemented in the new version is to change the port of the nmap target server to port 8200 (for testing, but any port should work). This was done by changing the ports specified in the docker-compose.yml file. A modified version of this file is included in appendix A.2.

4 DISCUSSION

4.1 Why not use React or Vue?

While the overall navigability of the challenge server has been improved as part of the project, the overall look of the website has remained more or less the same. Why not use a framework like React or Vue to improve the look of the website?

The simple answer is that the priority of this project was to add more challenges and make the challenge server (and CTFd server) easier to use and navigate. While a framework like React or Vue might have made the website look better and provide smoother animations/transitions, it would have taken a lot of time to learn and implement and would have taken time away from the main goal of the project. It would also only add to the look of the website, and not the functionality, which was the main focus of the project.

Thus, while it might be something for future maintainers to consider, it was something that was pursued for the project.

4.2 Notes for Future Maintainers

4.2.1 How to update CTFd

Updating the CTFd docker installation is fairly straightforward. First you need to delete the old images and containers, then you simply need to run the docker-compose up -d command again after cd'ing into the directory where the docker-compose.yml of the git cloned CTFd repo is located. Docker-compose should then pull the latest version of the CTFd docker image and run it.

The physical servers, which have been reserved for IBE321, should already have docker, docker-compose, as well as CTFd installed. Therefore, the only thing that needs to be done is to update the CTFd docker image to the latest version once the servers are brought back online.

4.2.2 Important files for CTF administrators

There are several things that need to be done in order to set up the improved CTF event. In Appendix A.2 there is a Google Drive folder containing several files that are important for the CTF administrator. These files are listed below:

- Arch Linux VM image.
- CTF Bootcamp document.
- CTFd nmap Target server configuration file.
- docker-compose.yml file (for nmap target).
- Main CTFd configuration file.
- Readme with further information on the files above.

For the challenge server, the complete code is included in Appendix A.1, as well as Appendix A.3. If one would like to try out the challenge server locally, they would only need to "git clone" the repository, and then run "poetry install" to set up the virtual environment with its dependencies. Then they would need to run "poetry run python main.py" to start the server. The challenge server/website should then be accessible at `http://localhost:81`.

4.3 Results

Discussing the results of the project without running the CTF event means that some results are not known in a proper objective manner. With this in mind, we can still discuss the results of the project in a more subjective manner. Meaning what this section will discuss is what the author believes to be the results of the project, and whether the project met the goals & requirements listed in sections 1.4 & 3.1.2.

4.3.1 Goals Achieved

The project's stated first goal was to create a CTF event with more challenges than the original implementation. Also, the challenges were to have a few levels of difficulty. This goal was met almost 100%. The part that keeps this goal from being perfectly met, is the part about multiple levels. As it stands, not all categories of challenges have multiple levels. That said, the categories with more complicated topics do have multiple levels. See section 3.3 for more details on which categories have multiple levels.

The second goal was to create a proper SQL injection challenge. This goal was met. The challenge was rewritten to use actual SQL queries to check if the user's input was valid, rather than a regex. This means that the challenge is now much more accurate and will not flag valid SQL injections as invalid.

Upon those goals being reached there was also the goal of improving the overall user interface and user experience (UI & UX). This goal was partially met. The user experience should now be improved, with the challenges being more segregated and easier to navigate. There is also now a warning that shows up should the display or browser window be too small. This is to prevent users from using the website on a mobile device, as the website is not optimized for mobile use. Users are also unlikely to have access to necessary tools on said mobile devices.

The user interface, however, has not been improved. The website theme is still more or less the same as the original version. Behind the scenes the CSS has been rewritten to be more organized and easier to read, but the overall look of the website has not been changed.

4.3.2 Requirements Met

The requirements were divided into two categories: functional and non-functional. In short, the functional requirements are what the project **must** do, while the non-functional requirements are what the project **should** do.

Functional Requirements

Of the five listed functional requirements, 3 were met and 2 were partially met. The first partially met requirement was the requirement of the CTF set up being capable of handling

100+ users. The CTFd server should already be capable of this, but the challenge server might not be. This is because the challenge server is hosted on a free Replit account, which means that it does not have access to the same resources as a paid account [Replit 2023a].

The alternatives for this are to either pay Replit for a paid account, pay Replit for a dedicated VM ($6.40 as of 05/2023), or to host the challenge server locally as with the CTFd server. The latter option does come with caveat that the server would not be as easy to update if the CTF admin is not onsite at Himolde.

The second partially met requirement was the requirement of the user input handled by the website (challenge server) being automatically and correctly verified. In this case the SQL injection challenge is now both automatically and perhaps more importantly correctly verified due to the total rewrite of the challenge backend. However, the challenge based on the path traversal vulnerability, while automatically verified, is not correctly verified. This is due to the security concerns mentioned in section 3.4.4.

Also, regarding the second partially met requirement: In the event of incorrect user input the website/challenge server has always returned an error message, indicating to the user that their input was incorrect. This is still the case, but with some of the more difficult challenges (i.e., the SQL injection challenges) the error message is now more descriptive and should help the user figure out what they did wrong.

Non-Functional Requirements

There were four non-functional requirements listed for the project. Of these, all four were met. The CTF should now be OS-agnostic for its servers. The challenge count is increased. The bootcamp document has been updated/rewritten, and it is accessible from the CTFd server. And lastly, the code comments and documentation have been improved. That said though, that one is more subjective, and it is certainly possible to improve it further.

4.4 Future Work

Path Traversal

Further work on this topic might not be a wise pursuit, due to the previously mentioned security concerns (see 3.4.4). That said, if one were to pursue this topic, it would be wise to investigate running the challenge server on a dedicated VM or on a separate, limited access, user on a physical (local) server.

SSH Access

Another point of possible future work is to figure out if it is possible to give a future CTF admin SSH access to the physical servers. This would allow the admin to update the challenge server without having to go to the physical server hardware at Himolde.

More Challenges

The CTF could always use more topic appropriate challenges. Adding more challenges to CTFd is straightforward and should not be too difficult for future CTF admins. However, adding more to the challenge server, does require some knowledge of Python, JavaScript, CSS and HTML/jinja2 but a student TA that has been through the courses of Himolde's IT & Digitalization program (*Bachelor i IT og digitalisering*) should be able to do it, with some help from online tutorials if needed.

AI & Cybersecurity

One subject that has been increasingly relevant to cybersecurity is Artificial Intelligence (AI), or rather the Large Language Model (LLM)s like ChatGPT 3.5 and 4.0. These LLMs are capable of not only generating text that is nigh indistinguishable from human written text, but they are also capable of generating code and even fully functional programs [Hughes 2023]. This means that it is possible that in the future LLMs like ChatGPT may be used by malicious actors to exploit vulnerabilities or outright generate malware.

The usage of such systems in cybersecurity is still in the early days as of the writing of this report (Spring 2023), but it is something that should be kept in mind for future CTFs as the technology matures and people figure out ways to use it for malicious purposes.

Frontend Framework

As mentioned in section 4.1, the challenge server could be improved by using a frontend framework like React or Vue. Implementing this would require an extensive rewrite of the code, but it could potentially improve the performance of the challenge server by offloading some of the work to the users' browsers.

One of the ways it could improve performance is by putting all challenges that only display simple text into a single page application that loads all the challenge text at once and then displays it as the user navigates the website. This would reduce the number of requests the server has to handle [Mozilla 2023d]. This would also allow the server to focus on handling the more complicated challenges, like the SQL injection challenges.

Further Optimization of SQL Setup

The SQL database in this setup is as mentioned in section 3.2.1.1 an SQLite database. This means that it might struggle with handling many concurrent users. This could be solved by switching to a more robust database like MySQL or PostgreSQL. Due to the modular nature of the database code, this should not be too difficult to implement.

The second optimization would be to look into the possibility of running the user input-based query inside a transaction without committing it to the database. This would allow the database to undo any potential damage done by the unsafe query (DROP Table, etc.). Successfully implementing this would make the database reset script redundant, as the database's state would not be changed by the query.

5 CONCLUSIONS

The overall goal of this project was to improve the original CTF event for the IBE321 course at Himolde, which was held originally in the fall/winter of 2022. The original event was successful, but it was clearly the first attempt at organizing such an event. The goal of this project was to improve the event in several ways.

This improvement was done not only by adding more challenges to the roster, but by improving some of the previous challenges, as well as improving the overall user experience.

The improvement of the challenges included multiple changes like:

- Recreating the SQL injection challenge to use actual vulnerable code, rather than a regex.
- Increasing the clarity of where challenges were meant to be solved on the challenge server/website, and in what order they were meant to be solved in CTFd.
- Adding more challenges of various difficulties and categories.

The user experience improvement included things like adding a warning to the website, should the user's browser window be too small, and by more clearly labeling the challenges on the website.

The bootcamp document was also altered to not only improve clarity and add further reading and information for the existing topics, but also to add topics for the new challenges as well.

In addition to the above, the backend code for the challenge server was also improved. The code was rewritten to be more organized and easier to read. The code was also rewritten to be more modular, which should make it easier to add new challenges in the future. The challenge server is now also able to be hosted both locally and online, as it was previously only tested to be hosted online via replit.com.

This concludes my report, and the project. I hope that any future CTF maintainer or administrator will find this report useful in their work on any future CTF event for IBE321 or Himolde in general.

REFERENCES

Acsany, Philipp. (2022). “Primer on Jinja Templating”. URL: https://realpython.com/primer-on-jinja-templating/ (Accessed 20/04/2023).

Brmwebdev. (2015). “Flask”. URL: https://www.brmwebdev.com/technologies/frameworks-and-cms/flask (Accessed 24/04/2023).

Coursera. (2023). “What Is Python Used For? A Beginner’s Guide”. URL: https://www.coursera.org/articles/what-is-python-used-for-a-beginners-guide-to-using-python (Accessed 16/04/2023).

CTFd. (2023a). “Hints”. URL: https://docs.ctfd.io/docs/challenges/hints (Accessed 06/05/2023).

CTFd. (2023b). “Pricing”. URL: https://ctfd.io/pricing/ (Accessed 24/03/2023).

CTFd. (2022). “What is Capture The Flag?” URL: https://ctfd.io/whats-a-ctf/ (Accessed 02/04/2023).

Cybertalents. (2023). “Top 6 Platforms to Run your CTF On”. URL: https://cybertalents.com/blog/top-platforms-to-run-your-ctf (Accessed 23/03/2023).

Docker. (2023a). “Docker Compose Overview”. URL: https://docs.docker.com/compose/ (Accessed 25/04/2023).

Docker. (2023b). “Key features and use cases”. URL: https://docs.docker.com/compose/features-uses/ (Accessed 25/04/2023).

Docker. (2023c). “Volumes”. URL: `https://docs.docker.com/storage/volumes/` (Accessed 25/04/2023).

Docs, Jinja2. (2023). “Template Designer Documentation”. URL: `https://jinja.palletsprojects.com/en/3.1.x/templates/` (Accessed 20/04/2023).

Fitzgibbons, Laura. (2018). “DRY principle”. URL: `https://www.techtarget.com/whatis/definition/DRY-principle` (Accessed 30/04/2023).

Hackthebox. (2023). “About Us”. URL: `https://www.hackthebox.com/about-us` (Accessed 04/03/2023).

Hughes, Alex. (2023). “ChatGPT: Everything you need to know about OpenAI’s GPT-4 tool”. URL: `https://www.sciencefocus.com/future-technology/gpt-3/` (Accessed 10/05/2023).

IBM. (2023). “What is Docker?” URL: `https://www.ibm.com/topics/docker` (Accessed 25/04/2023).

Martin, Matthew. (2023). “Functional vs Non Functional Requirements”. URL: `https://www.guru99.com/functional-vs-non-functional-requirements.html` (Accessed 28/03/2023).

Mozilla. (2023a). “CSS: Cascading Style Sheets”. URL: `https://developer.mozilla.org/en-US/docs/Web/CSS` (Accessed 16/04/2023).

Mozilla. (2023b). “HTML: HyperText Markup Language”. URL: `https://developer.mozilla.org/en-US/docs/Web/HTML` (Accessed 16/04/2023).

Mozilla. (2023c). “Introduction: What is JavaScript”. URL: `https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Introduction#what_is_javascript` (Accessed 20/04/2023).

Mozilla. (2023d). “SPA (Single-page application)”. URL: https://developer.mozilla.org/en-US/docs/Glossary/SPA (Accessed 11/05/2023).

Omnicybersecurity. (2023). “What does salting the hash mean (and is it effective?)” URL: https://www.omnicybersecurity.com/what-does-salting-the-hash-mean-is-it-effective/ (Accessed 27/04/2023).

Oracle. (2023). “VirtualBox’s user manual Ch. 1”. URL: https://www.virtualbox.org/manual/ch01.html (Accessed 30/04/2023).

Overthewire. (2023a). “Bandit”. URL: https://overthewire.org/wargames/bandit/ (Accessed 04/03/2023).

Overthewire. (2023b). “Bandit”. URL: https://overthewire.org/wargames/bandit/bandit0.html (Accessed 04/03/2023).

Overthewire. (2023c). “Wargames”. URL: https://overthewire.org/wargames/ (Accessed 04/03/2023).

Python-poetry. (2023a). “Introduction”. URL: https://python-poetry.org/docs/ (Accessed 25/04/2023).

Python-poetry. (2023b). “Managing environments”. URL: https://python-poetry.org/docs/managing-environments/ (Accessed 25/04/2023).

Python.org. (2022). “Beginners Guide Overview”. URL: https://wiki.python.org/moin/BeginnersGuide/Overview (Accessed 16/04/2023).

Python.org. (2023a). “sqlite3 — DB-API 2.0 interface for SQLite databases”. URL: https://docs.python.org/3/library/sqlite3.html (Accessed 20/04/2023).

Python.org. (2023b). “venv — Creation of virtual environments”. URL: https://docs.python.org/3/library/venv.html (Accessed 25/04/2023).

Pythonbasics. (2021). “What is Flask Python”. URL: https://pythonbasics.org/what-is-flask-python/ (Accessed 23/04/2023).

Ramos, Leodanis Pozo. (2023). “Context Managers and Python’s with Statement”. URL: https://realpython.com/python-with-statement/ (Accessed 30/04/2023).

Replit. (2023a). “Build & host any project, all in one place”. URL: https://replit.com/pricing (Accessed 08/05/2023).

Replit. (2023b). “Hosting Static Web Pages”. URL: https://docs.replit.com/hosting/hosting-web-pages (Accessed 17/04/2023).

Replit. (2023c). “Instant IDE: Code from your browser”. URL: https://replit.com/site/ide (Accessed 24/04/2023).

Replit. (2023d). “The quickest way to go from idea → production”. URL: https://replit.com/site/deployments (Accessed 24/04/2023).

Simmons, Liz. (2023). “Front-End vs. Back-End: What’s the Difference?” URL: https://www.computerscience.org/bootcamps/resources/frontend-vs-backend/ (Accessed 30/04/2023).

SQLite. (2023). “About SQLite”. URL: https://sqlite.org/about.html (Accessed 20/04/2023).

Stackoverflow. (2022). “2022 Developer Survey”. URL: https://survey.stackoverflow.co/2022/#technology-most-popular-technologies (Accessed 16/04/2023).

Upskilld. (2021). “Free Cybersecurity Labs and Wargames for Beginners”. URL: https://upskilld.com/article/free-cybersecurity-labs-and-wargames-for-beginners/ (Accessed 08/03/2023).

Vishal. (2021). “Python MySQL Execute Parameterized Query using Prepared Statement”. URL: https://pynative.com/python-mysql-execute-parameterized-query-using-prepared-statement/ (Accessed 01/05/2023).

APPENDICES

A LINKS

A.1 GitHub repository

- `https://github.com/eribre/IBE321_CTF_for_IBE600`

A.2 Google Drive folder

- `https://drive.google.com/drive/folders/1i9CFCyxSUKo38LhyzC4sxgFBd8G0fr3Y?usp=sharing`

A.3 Replit

Link to the repl containing the challenge server code

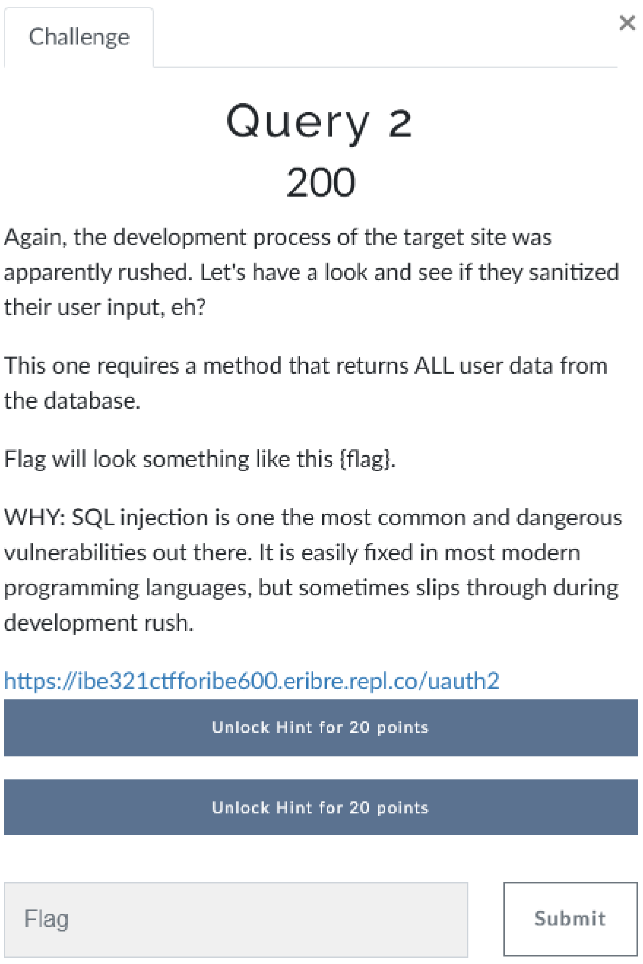
- `https://replit.com/@eribre/IBE321CTFforIBE600`

Link to the website hosted via the repl listed above

- `https://ibe321ctfforibe600.eribre.repl.co`

B MORE SCREENSHOTS

B.1 CTFd Screenshots



Challenge

Query 2

200

Again, the development process of the target site was apparently rushed. Let's have a look and see if they sanitized their user input, eh?

This one requires a method that returns ALL user data from the database.

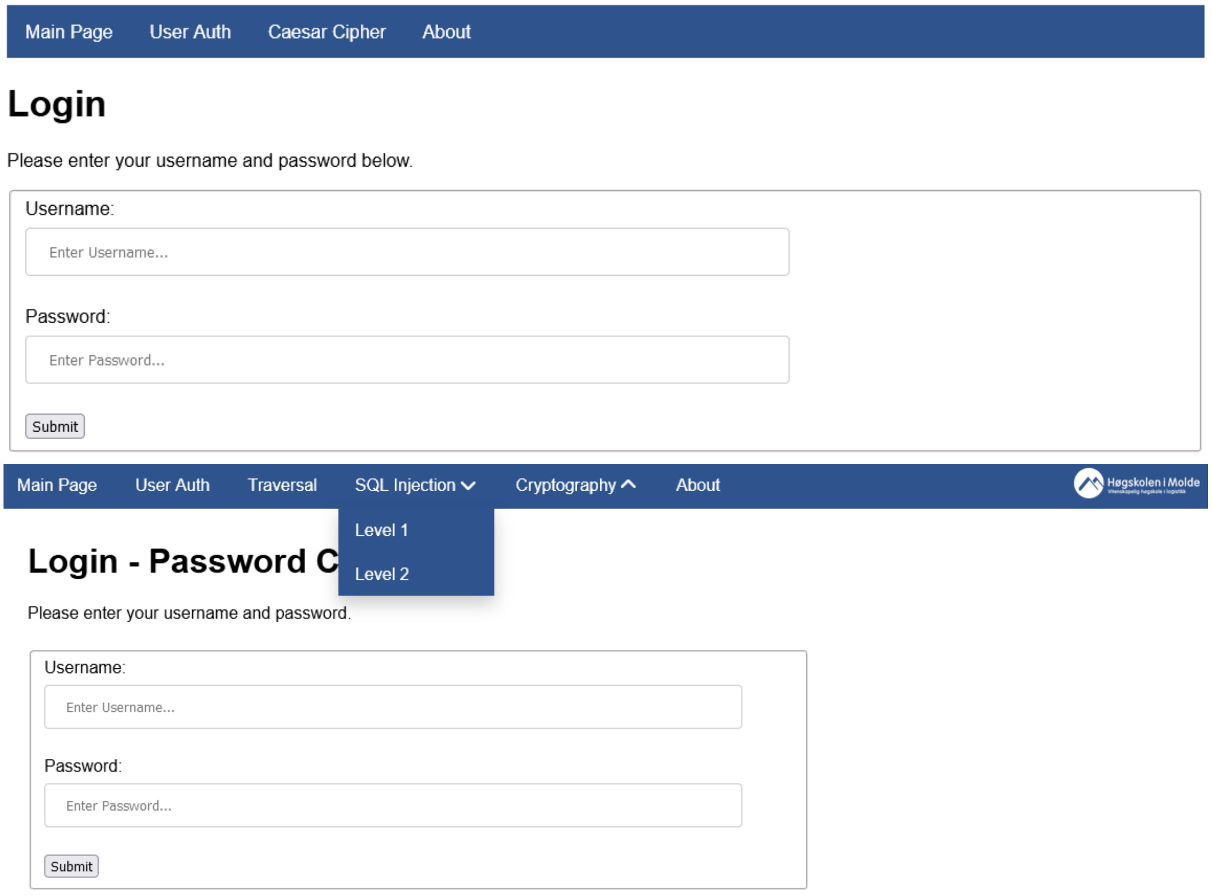
Flag will look something like this {flag}.

WHY: SQL injection is one the most common and dangerous vulnerabilities out there. It is easily fixed in most modern programming languages, but sometimes slips through during development rush.

https://ibe321ctfforibe600.eribre.repl.co/uauth2

Figure B.1.1: This screenshot shows an example of what a user/student sees when opening a challeng.

B.2 Flask/Challenge Server Comparison



Login

Please enter your username and password below.

Username:

Enter Username...

Password:

Enter Password...

Login - Password C

Please enter your username and password.

Username:

Enter Username...

Password:

Enter Password...

Figure B.2.1: A comparison of the old and new versions of the Flask/Challenge Server code. The old version is on top, and the new version is on the bottom.