Zurich University
of Applied Sciences

School of
Engineering
InIT Institute of Applied
Information Technology

Bachelor thesis Computer Science

Smart Contract Scam Checker for Decentralized Finance (SCSC)

Author	Pascal Ackermann
	Stefano Tassone
Main supervisor	Gürkan Gür

Date	June 10, 2022

DECLARATION OF ORIGINALITY

Bachelor's Thesis at the School of Engineering

By submitting this Bachelor's thesis, the undersigned student confirms that this thesis is his/her own work and was written without the help of a third party. (Group works: the performance of the other group members are not considered as third party).

The student declares that all sources in the text (including Internet pages) and appendices have been correctly disclosed. This means that there has been no plagiarism, i.e. no sections of the Bachelor thesis have been partially or wholly taken from other texts and represented as the student's own work or included without being correctly referenced.

Any misconduct will be dealt with according to paragraphs 39 and 40 of the General Academic Regulations for Bachelor's and Master's Degree courses at the Zurich University of Applied Sciences (Rahmenprüfungsordnung ZHAW (RPO)) and subject to the provisions for disciplinary action stipulated in the University regulations.

City, Date:	**Name Student:**
Schaffhausen, June 10, 2022	Pascal Ackermann 
Uttwil, June 10, 2022	Stefano Tassone 

Smart Contract Scam Checker for Decentralized Finance (SCSC)

Author:
Ackermann Pascal, Tassone Stefano

Date:
June 10, 2022

Supervisor, Institute:
Gürkan Gür, InIT Institute of Applied Information Technology

Abstract

The number of scams in decentralized finance (DeFi) is significantly high. Essentially, it is more likely that a newly published contract is a scam than it is legitimate. Therefore, the goal of this project was to find ways of identifying such scams early, using various indicators and live data collected from DeFi-related services for Ethereum. To this end, we analysed common scams and implemented seven indicators. These indicators collect data from external APIs and process them using logic, best practices, and statistical observations extracted from the dataset, which was gathered throughout the creation of this project.

We have implemented three main approaches: a weighted average of the indicator scores, a Support Vector Machine based decision and an XGBoost model trained on the collected dataset. The experimental results show that our proposed solution can successfully differentiate between obvious scams and obvious benign contracts. On the test dataset, the weighted average classifier shows an accuracy of 71%, the SVM classifier accuracy of 83% and the XGBoost classifier accuracy of 86%. However, the results of the test dataset show that it is difficult to mark newly created smart contracts as benign.

As future work and a potential extension to our current bachelor thesis work, a significant improvement would be to address the issue of missing crucial information for some of our indicators due to early analysis of newly created smart contracts. Moreover, a more extensive and more balanced test dataset is crucial to investigate the weak identification performance for benign contracts with our proposed approaches. A periodic rescan or a longer delay between pair creation and automated analysis would be beneficial to mitigate this weakness.

Zusammenfassung

Die Anzahl der Betrugsfälle im Decentralized Finance (DeFi) ist beträchtlich hoch. Es ist wahrscheinlicher, dass ein neu veröffentlichter Smart Contract ein Betrug ist, als dass er legitim ist. Ziel dieses Projekts war es daher, mit Hilfe verschiedener Indikatoren und Live-Daten, die von DeFi-Diensten für Ethereum gesammelt wurden, Wege zur frühzeitigen Erkennung solcher Betrügereien zu finden. Zu diesem Zweck haben wir gängige Betrugsfälle analysiert und sieben Indikatoren implementiert. Diese Indikatoren sammeln Daten von externen APIs und verarbeiten sie mit Hilfe von Logik, Best Practices sowie statistischen Beobachtungen aus dem Datensatz, der während der Erstellung dieses Projekts gesammelt wurde.

Es wurden drei Ansätze für die Bewertungen der Daten implementiert. Als erstes wurde der gewichtete Durchschnitt der Indikatorwerte für die Klassifizierung verwendet, zweitens wurde über eine Support Vector Machine basierende Entscheidung bewertet und drittens wurde ein XGBoost-Modell genutzt, welches auf dem gesammelten Datensatz trainiert wurde. Die experimentellen Ergebnisse zeigen auf, dass unsere vorgeschlagene Lösung erfolgreich zwischen offensichtlichem Betrügen und eindeutig gutartigen Smart Contracts unterscheiden kann. Im Testdatensatz zeigt der gewichtete Durchschnitts-Klassifikator eine Genauigkeit von 71%, der Support Vector Machine-Klassifikator eine Genauigkeit von 83% und der XGBoost-Klassifikator eine Genauigkeit von 86%. Die Ergebnisse des Testdatensatzes zeigen jedoch, dass es schwierig ist, neu erstellte Smart Contracts als legitim zu kennzeichnen.

Als zukünftige Arbeit und als mögliche Erweiterung unserer aktuellen Bachelorarbeit wäre es wichtig, das Problem der fehlenden Informationen aufgrund der frühen Analyse neu erstellter Smart Contracts anzugehen. Diese Informationen sind für einige unserer Indikatoren entscheidend. Darüber hinaus ist ein grösserer und ausgewogenerer Testdatensatz von entscheidender Bedeutung, um die schwache Identifizierungsleistung für legitime Smart Contracts mit unseren vorgeschlagenen Ansätzen zu untersuchen. Um diese Schwäche zu reduzieren, wäre ein regelmässiger Rescan oder eine längere Zeitspanne zwischen Erhalt und der automatischen Analyse der Smart Contracts von Vorteil.

Foreword

Pascal Ackermann came to cybersecurity through his studies. This is where he found his passion. He would rather take a system apart than write it.

During his studies, Stefano Tassone was always committed to doing a project on a blockchain topic, whether written or programmed. Among them, a Uniswap trading bot was programmed, integrating functionalities such as stop-loss and take-profit. The work was also written with Pascal Ackermann.

We started studying together full-time and switched to part-time after one year. Since then, we have worked in the same group whenever possible. That is why the choice of partner for the bachelor's thesis was clear. Now we just had to find a topic. It was clear that it had to be something about which both of us were passionate. That is why we agreed on the area of blockchain and cybersecurity. They have always been interested in hacks. That is why we quickly came up with the idea of writing a vulnerability scanner, which was done in the project work. Then, we looked for a project based on the vulnerability scanner and wanted to give investors a way to distinguish between scams and legitimate projects. Thus, the idea of the bachelor thesis "Smart Contract Scam Checker for Decentralized Finance: SCSC" was born.

Without the support of our supervisors, we would not have come this far! Therefore, we extend many thanks to Peter Berlich for his support in our project work and to Gürkan Gür for his support with the bachelor thesis.

Table of contents

1 Introduction

During the research of our previous work on “Vulnerability Scanning Smart Contracts” [1], we saw a significant number of malicious tokens constantly stealing a sizeable amount of funds from investors. This prompted us to extend our research to include details on the exploitation of investors and resulted in our second work on the blockchain topic in the form of this Bachelor thesis.

1.1 Existing Situation

Uniswap is a decentralized exchange, or DEX for short. It is deployed to the Ethereum Blockchain free from regulation, authority, and dependence on a third party. A study from November 2021 [2] revealed that around 50% of the listed tokens on Uniswap v2 are scams. On April 4, 2022, the trading volume of the Uniswap v3 was $1.7 * 10^9$ USD within 24 hours [3]. These large sums make it highly attractive for scammers, hackers, and other threat agents.

Scammers, in particular, lure investors to invest in their cryptocurrency by stating it will be "the next Bitcoin" or any other successful cryptocurrency. Scammers write targeted smart contracts that rob investors of their capital.

The techniques used by scammers to extract the currency are very diverse, ranging from removing the liquidity pool or placing obfuscated backdoors inside the source code to precisely placing bugs in key places. The more advanced techniques allow the malicious developer to mask their contract by locking or burning the liquidity pool, renouncing the ownership of the contract, verifying their source code, and still having a way to steal the investors’ money.

This poses an interesting challenge in determining what methods scammers use to mask their intent and what information is required to identify the fraud before the scammer has a chance to execute his attack.

This work tries to identify such scams. The aim is to provide an assessment of smart contracts published on Uniswap with the help of indicators identified in this research.

1.2 Objectives

The objective of the paper is to evaluate smart contracts for scam potential. First, the theoretical foundations around the topic of Ethereum, Decentralized Exchange and thread landscape will be explained. A common scam analysis will demonstrate this in Chapter 3.2. Based on this analysis, it should be possible to identify indicators suitable for early recognition of such scams.

Based on those identified indicators, a web platform will be built that monitors and evaluates smart contracts. The system will monitor the decentralized exchange Uniswap for newly

published smart contracts. The analysis will then be stored persistently, allowing for the determination of accuracy and trust in the used indicators.

A vulnerability analysis using Slither is performed for smart contracts that publish the source code. This tool was evaluated in a previous work [1]. All these factors are displayed to the user on the front end. With those results and suggestions, a user of the application can get an overview of the risk involved with the target contract.

There are no suitable datasets for the use case of this study. The goal of this tool is to detect scam tokens before the attack is executed, meaning before the value inside the contract is removed by the attacker. For that, the tool must run on live-captured contracts before it is possible to label the targeted contract as a definite scam. An additional objective of this work is thus to create and label a dataset based on the tokens we scanned and provide it together with the implemented system.

1.3 Related Work

The following works have provided insight into the topic and have been incorporated into the work and the procedure:

- "Trade or Trick" Xia et al. [2]
  In this work, smart contracts that can be traded on Uniswap v2 were analysed using machine learning. For this purpose, 20,000 smart contracts were manually analysed and evaluated.
- "Do Not Rug on Me: Leveraging Machine Learning Techniques for Automated Scam Detection" Mazorra et al. [4]
  This work builds on top of previous research to improve the reliability of detecting rug pulls on Uniswap V2 before they are executed, using additional features for machine learning based algorithms.

2 Theoretical Background

This subchapter deals with the most important terms and concepts related to our work and presents theoretical background.

In previous work, a vulnerability scanner for ERC-20 tokens was developed [1]. There is a sizeable amount of overlap in the theoretical foundations. For this reason, parts of the theoretical background have been adapted and the original sources are still cited, namely Chapters 2.1.1 Blockchain to 2.1.2.3 Smart Contracts except for Chapters 2.1.2.9 Layer-2 Solutions and 2.1.2.10 EVM Compatible smart contracts platforms

2.1 Distributed Ledger Technology

Distributed ledger systems are systems that distribute computing power and information among different participants. The system is assumed to have participants with malicious intentions who should not be able to write, but only read, through confirmation by the other participants.

Depending on the type of implementation, the criteria for security, speed, efficiency, and degree of decentralisation vary.

As can be seen in Figure 1, the following distributed ledger systems exist according to the conference paper of El Ionini et al. [5]:

- Blockchain

With blockchain, information is combined into blocks. The information is hashed, which is considered an identifier. The blocks are arranged in a linked list and continued. More details will be explained in the following subchapter.

- Tangle

Transactions are mutually verified, creating an acyclic graph. The longer a transaction exists, the more the transactions based on it are verified. This results in a weighted graph.

- Hash graphs

Transactions are written over time since the participants know each other's location via the individual nodes.

- Sidechains

These private blockchains store information in their own blockchain and communicate with external blockchains.

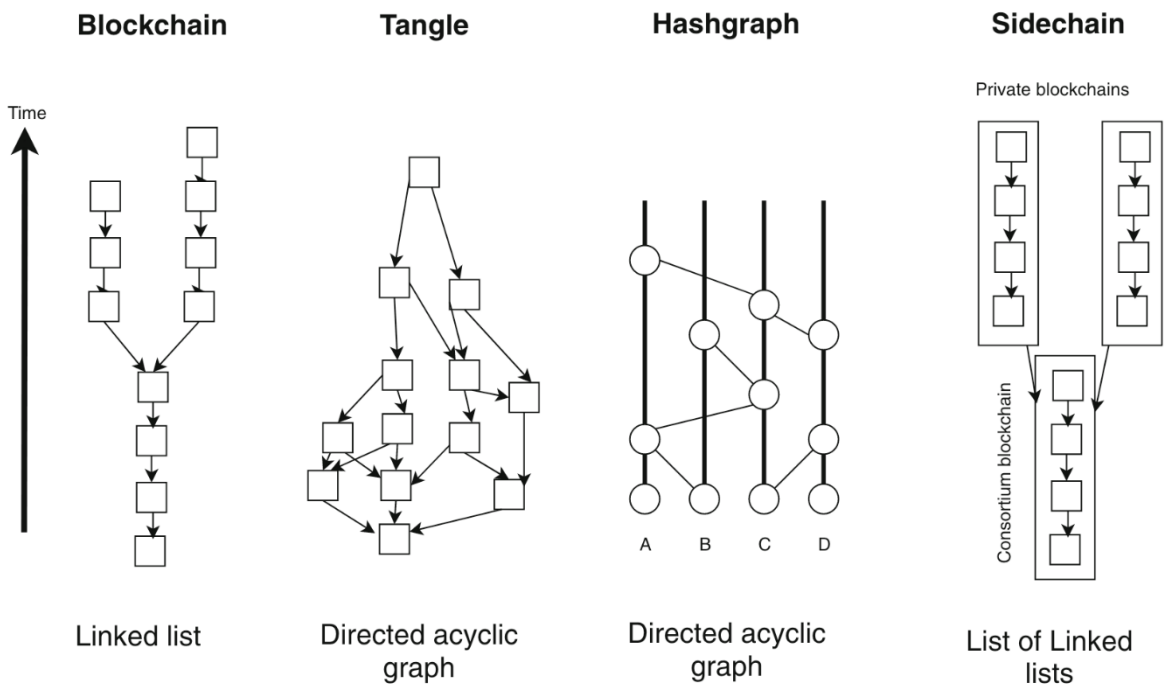


Figure 1: Distributed ledger technology overview [5]

2.1.1 Blockchain

Blockchain technology was made famous in 2008 by Satoshi Nakamoto with Bitcoin [6]. Satoshi Nakamoto is considered an alias; no one knows who this is or if it is just a fictional person or group of developers. The basic idea of Bitcoin is to have a decentralised currency that is not managed by an institution (such as a bank) and therefore provides anonymity. Regulations, central banks, or states should not be able to influence cryptocurrency.

Even though blockchain technology has gained popularity through Bitcoin, cryptocurrencies are not the only application. The technology itself can be used to solve various other problems, i.e. identity management or contract management.

The price is driven by supply and demand for cryptocurrencies. Every transaction in the blockchain is openly displayed in the network. The cryptocurrencies themselves are stored in a wallet, i.e., a digital wallet. More information on wallets will follow in Chapter 2.1.4 Wallet Types. The private key is located in the wallet. The transactions are therefore signed using public-key cryptography. In the process, transactions are combined into a block. A block has a hash value and a reference in the form of a hash of the previous block. This results in a chain of blocks, hence the name blockchain. How the blockchains are divided into blocks and transactions can be seen in Figure 2.

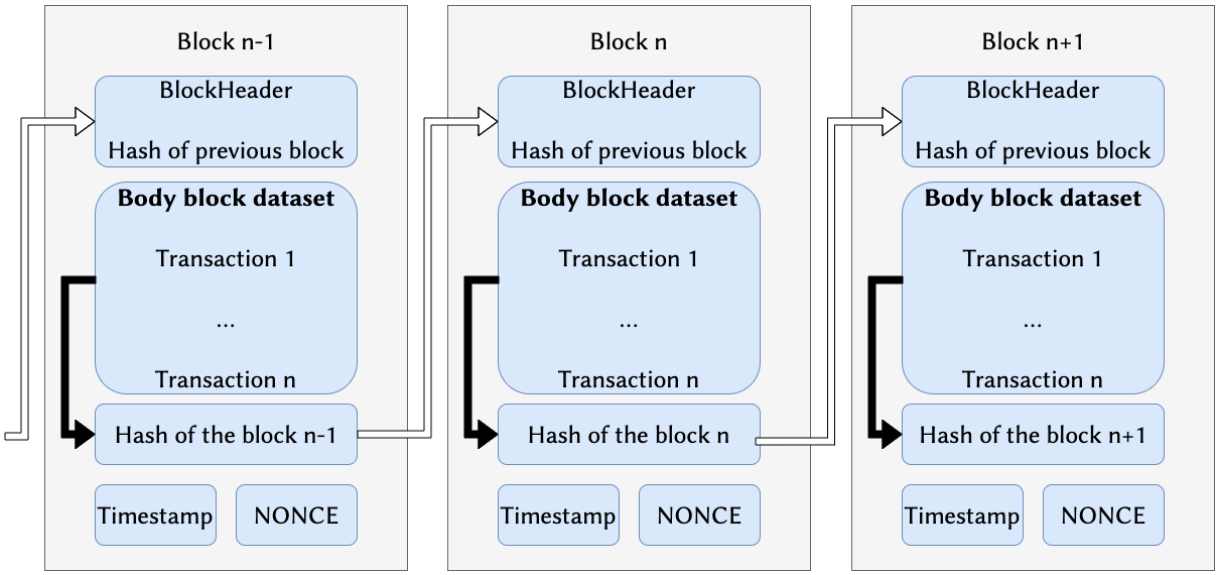


Figure 2: Blockchain structure based on [7]

2.1.2 Ethereum

Ethereum is the largest and first platform for smart contracts since its release in 2014 [8]. As of 2017, Ethereum is second in market capitalisation, with only a few interruptions from Ripple [9]. This is right after Bitcoin, which has been number one since the beginning of its release in 2009. Furthermore, Ethereum is also very popular with miners. It reached the hash rate of 1 Petahash per second, which is 10^{15} Hashes calculated every second, for the first time ever in April 2022 [10]. For these reasons, this bachelor's thesis is primarily based on Ethereum.

2.1.2.1 Verification through Proof of Work

Ethereum is mined using a nonce to be found. The algorithm behind this is the hash function ETHash. This is based on an earlier SHA-3 variant of Keccak, in which the padding was adjusted [11].

A miner must try out different nonces until the resulting hash value of the block corresponds to the current Difficulty Target. As an example, zeros are used at the beginning of the hash value. This illustrates the computing power required to mine a block.

Once a miner has found the solution, the other miners verify this solution. Only when 50% or more of the miners confirm this solution is the corresponding miner paid with Ether. This process is called Proof of Work.

The probability of mining a block is the same for each attempt and is, therefore, a Laplace experiment. The only difference between miners is the amount of computing power they have, which allows the miner to make more attempts in the same amount of time. The more computational power someone has, the greater the chance of being able to calculate the next block, thus reducing transaction costs and obtaining the block reward.

However, this approach is very inefficient. The entire network searches for a solution. If this solution is found and verified, the computing power of all the other computers is wasted.

A security problem exists if 50% or more verify a block that has been forged. Thus, Ether could be created that does not exist and written to a wallet. This corrupted block would then be accepted by the network. This attack scenario is called a 51% attack [12].

2.1.2.2 Verification through Proof of Stake

To address these issues, Ethereum will switch from Proof of Work to Proof of Stake in the future. Proof of Stake no longer requires miners, but nodes. These nodes will compute and verify transactions that are easy to process. In order to do this securely, each participant must stake, i.e. deposit, Ether.

The staked Ether is taken as collateral. If a staker has malicious intentions, the Ether deposited in the network can be removed. In return, a percentage of Ether per year is credited to the account. The transaction fees are burnt, i.e., destroyed, and are lost.

The advantage of the change is that transaction fees can be minimised, as nodes do not require as much computing power and thus consume less energy. This results in an energy saving of 99.95% of the current power consumption of the Ethereum miners [13]. Also, it is much harder to get half of the Ethereum in circulation than half of the hashing power [1].

2.1.2.3 Smart Contracts

A smart contract is a code that enforces specific actions through programmes. In addition, it can store its state on the blockchain. This goes further than the classical blockchain, which can only send currencies without a third party. Smart contracts can set additional conditions.

A smart contract can create its currency stored on a platform such as Ethereum. These currencies are called tokens. This can be illustrated as follows:

> *If a party buys a house, various third parties such as the land registry and banks are involved to ensure that all parties abide by the rules. A house purchase could work like this:*
>
> *A smart contract checks whether the buyers have enough funds in their wallets. If this is the case, it checks whether the financial resources deposited in the last few years are sufficient to pay off the house in a reasonable period. If the financial means are verified, a digital key could be sent to the buyers. For execution, a digital keylock would have to be attached to the house. If the buyers are no longer able to pay for any reason, the smart contract could be automated and lock the buyers out of the house.*

2.1.2.4 Terms: Ethereum, Ether, ETH

Ether is the name given to the currency used to make payments on the Ethereum blockchain, with the term Ethereum referring to the platform itself. This enables a series of decentralised applications to be created. These are called smart contracts.

The accuracy of Ether is limited to 18 decimal places. The smallest possible unit is, therefore, 10^{-18} Ether and is called Wei [13]. Gwei, i.e., Giga wei, is also often used as a unit, which is 10^{9} Wei.

2.1.2.5 Stablecoins

A few tokens have a special use case. For example, the Token USDT is pegged to USD [14]. Every circulating USDT is supposed to be backed by a USD. Those kinds of Tokens are called stablecoins. There are also other fiat currencies like the xCHF, which is backed by Swiss Francs.

For Decentralized Exchanges, stablecoins have a special meaning. Since there is no possibility to withdraw your funds directly to the bank account, stable coins are used on decentralized exchanges instead to hold the fiat value.

2.1.2.6 Gas Fees

The Ethereum network needs energy and capacity. To compensate for the used resources, a fee is charged, called gas fee. Furthermore, the gas fees should not only cover the resources but also be lucrative so that the network can be supported by the miners who maintain the network.

If a smart contract is deployed or a transaction is processed by the blockchain, the gas fee must be paid for each interaction. However, this is not fixed, but is determined by the supply and demand of interactions. Each participant can set the price in Gwei itself. Priority is given according to the number of Gwei paid. Those who pay more get their turn faster. Figure 3 provides an overview of the gas fees.

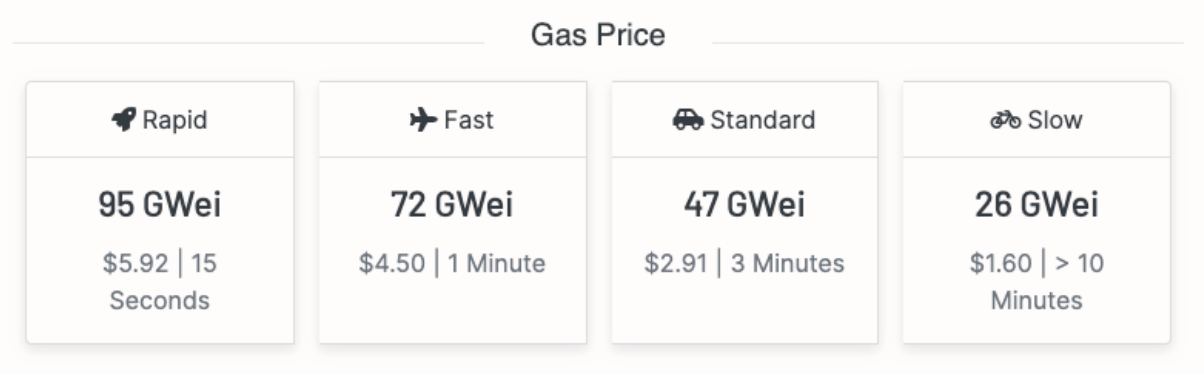


Figure 3: Gas Fees on 23.04.2022 at 00:05 from etherchain.org [15]

It should be noted that gas fees are very volatile. Between April 16, 2022 and April 23, 2022, the average gas fees were between 15.5 Gwei and 244.5 Gwei, as can be seen in Figure 4.

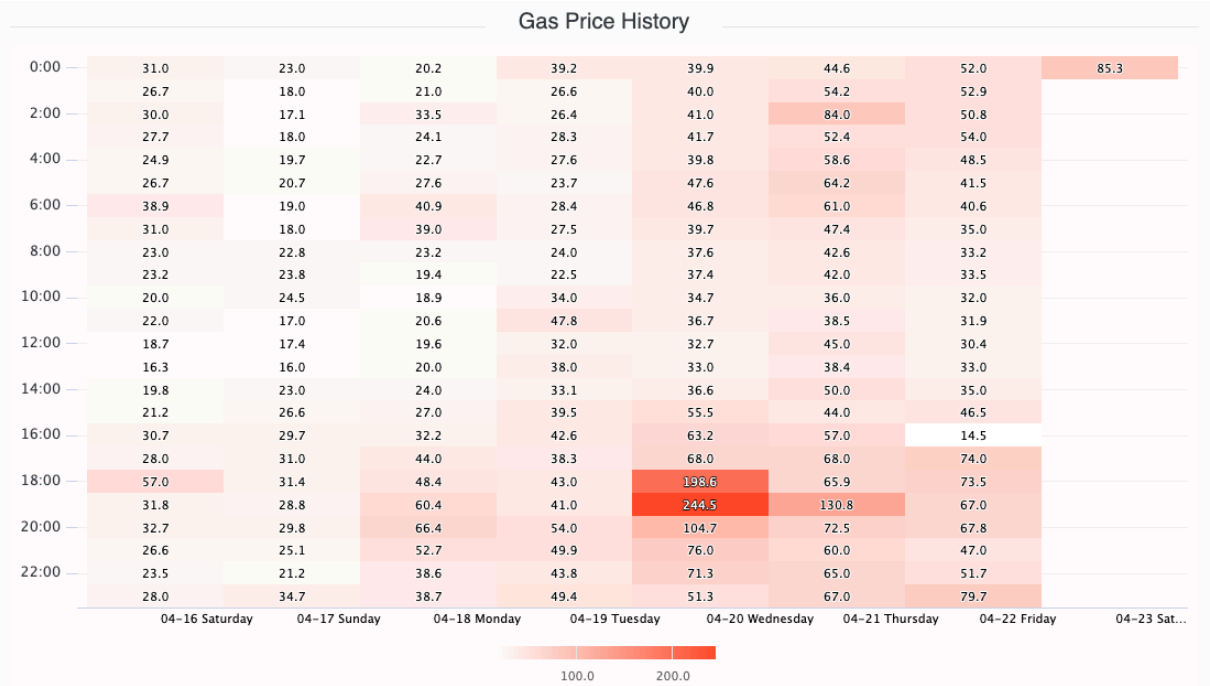


Figure 4: Gas price history from 16.04.2022 to 23.04.2022 on etherchain.org [15]

2.1.2.7 EVM

The Ethereum Virtual Machine, EVM for short, is the environment in which the smart contracts are interpreted and executed. The structure is not like a cloud, but similar to the JavaVM. The EVM has only one instance, which is executed by many computers in the network and is OS-independent. There is only one canonical state. The EVM itself is a Turing-complete state machine. The underlying Ethereum protocol has the task of making the operations of this state machine immutable. Furthermore, the protocol should ensure that operations can run uninterrupted [16].

Ethereum has a data structure that extends classic blockchains as seen in Figure 5. Besides the account information, the EVM also stores the current state in permanent storage, which is zero-initialised. This state only changes from block to block through the EVM bytecode, which is stored immutably in ROM. Non-permanent calculations are stored on the volatile memory, which is also zero-initialised [16].

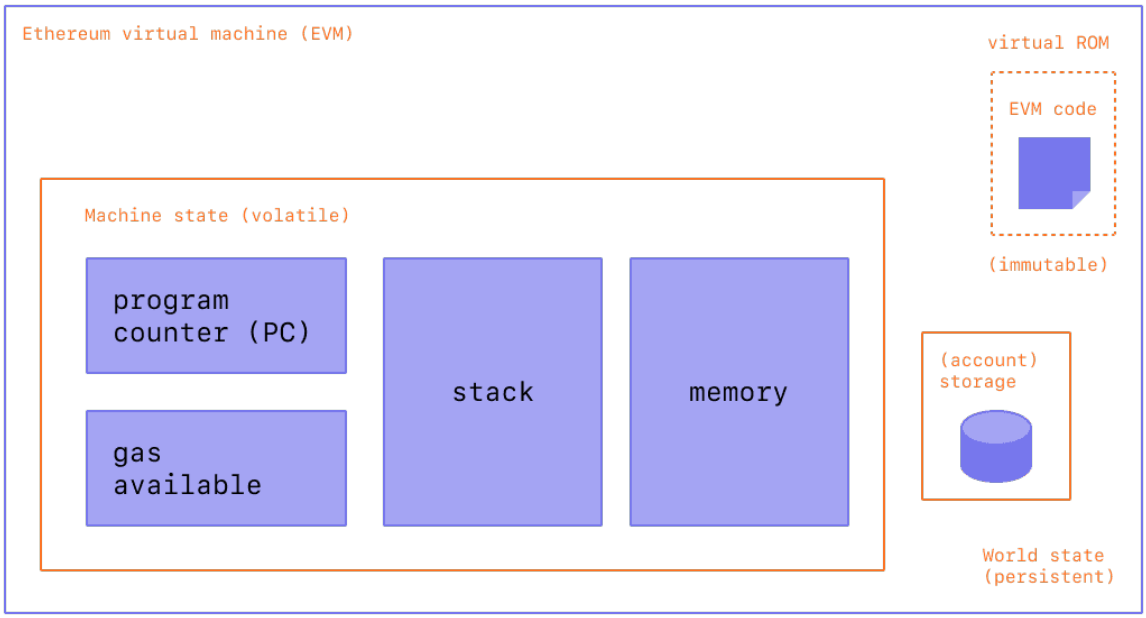


Figure 5: EVM structure [17]

The EVM determines which conditions must be met to write a new valid state into a block. There are two types of transactions: Contract Creation or Message Call. Contract Creation creates a new smart contract and stores it in bytecode. The bytecode can be interacted with via Message Calls [17].

2.1.2.8 Ethereum Improvement Proposal (EIP) and Ethereum Request for Comments (ERC)

Ethereum's development is being pushed forward by the Ethereum Improvement Proposals or EIPs for short. In summary, these are proposals from the community on which direction Ethereum should move. The EIPs are reviewed by the core developers and put to a vote. Anyone who owns Ethereum can vote. To do this, a small amount of Ethereum is sent to a voting wallet. Now it is checked how much Ethereum is on this wallet and receives a proportional voting weight [18].

2.1.2.9 Layer-2 Solutions

These Layer-2 solutions are side-chains of Ethereum to provide scalability and fewer fees. Polygon, for example freezes the ERC-20 on the Ethereum Blockchain and creates them in the Polygon chain over a so-called bridge. The basic idea is that a different validation method is used than in the main blockchain. Often efficiency is increased by making the validation less stringent, and many hackers state that they can get through without compromising security; however, many hacks of smart contracts exploit the weaknesses of the Layer-2 solutions to get at the assets. Examples of this are shown in Chapter 2.3.

2.1.2.10 EVM Compatible smart contracts platforms

Ethereum is so popular that companies took the code, forked Ethereum, and created their own Blockchain. The most popular fork is Binance Smart Chain or BNB for short. Binance is one of the leading crypto exchanges as of 2022. They took Ethereum and made an exchange token out of it. Other EVM compatible blockchains are Avalanche, Fantom, and Polygon, among many others [19].

2.1.3 Solidity

Solidity is a high-level programming language used for the development of smart contracts. It is statically typed and supports inheritance, libraries, and complex user-defined types as well as other functions. The current version of Solidity is version 0.8.13 until 26.04.2022 [22].

Solidity is compiled down to EVM bytecode by the solc Solidity compiler and then pushed onto the blockchain. As soon as the bytecode is published on the blockchain, other contracts can interact with it in various exported functions.

2.1.3.1 Example Smart Contract

The following Code 1 shows what a valid implementation in Solidity can look like. The smart contract named Example is implemented. It accepts incoming and outgoing amounts. The withdrawal must be less than 10^{17} Wei, which is 0.1 Ether.

```
01: // Version for the Solidity Compiler that was written
02: pragma solidity ^0.8.12;
03:
04: // Contract Name
05: contract Example {
06:
07:   // Send Ether to all who request it
08:   function withdraw(uint withdraw_amount) public {
09:
10:       // Limit the payout (Wei)
11:       require(withdraw_amount <= 100000000000000000);
12:
13:       // Send the amount requested to the address that requested it
14:       msg.sender.transfer(withdraw_amount);
15:     }
16:
17:   // Accept the incoming amount
18:   function () external payable {}
19:
20: }
```

Code 1: Example Smart Contract in Solidity [16]

2.1.3.2 Token standards: Ethereum Request for Comments (ERC)

The implementation of the EIPs is specified in the Ethereum Request for Comments or ERC for short. The proposal from the EIP is defined as a standard in the corresponding ERC. Thus,

a standard was proposed in EIP-20 [20] to implement tokens with predefined methods and events. This was then defined accordingly in the ERC-20. The most important standards are ERC-20 [20] and ERC-721 [21]. ERC-20 tokens are fungible tokens. Each token has the same value, which makes all tokens exchangeable, comparable to a traditional currency.

To create a token in the ERC-20 standard, the following methods of Code 2 must be implemented:

```
1: function name() public view returns (string)
2: function symbol() public view returns (string)
3: function decimals() public view returns (uint8)
4: function totalSupply() public view returns (uint256)
5: function balanceOf(address _owner) public view returns (uint256 balance)
6: function transfer(address _to, uint256 _value) public returns (bool suc-
cess)
7: function transferFrom(address _from, address _to, uint256 _value) public
returns (bool success)
8: function approve(address _spender, uint256 _value) public returns (bool
success)
9: function allowance(address _owner, address _spender) public view returns
(uint256 remaining)
```

Code 2: ERC-20 standard functions [22]

In addition to classic functions, there are events, as shown in Code 3, that need to be implemented to store a transaction log on the blockchain.

```
1: event Transfer(address indexed _from, address indexed _to, uint256 _value)
2: event Approval(address indexed _owner, address indexed _spender, uint256
_value)
```

Code 3: ERC-20 standard events [22]

A different token standard is ERC-721, which are non-interchangeable tokens. For this reason, they are called NFTs, which stand for non-fungible tokens. This means that each token has its own value, even though it comes from the same series. Media can be attached to it and sold online for cryptocurrencies. For example, tokens created in the form of a series of paintings can have different prices [22, 23].

An example of this is the ERC-721 token CryptoPunk. This series of NFTs contains 10'000 paintings. The CryptoPunk number 9'998 was sold on October 28, 2021 for about 124'457 Ether. This demonstrates the ERC-721 concept, as there are CryptoPunk tokens that have not yet been purchased and could be bought for a fraction of Ether [24].

2.1.4 Wallet Types

The terms hot wallet and cold wallets refer to the connection that they have to the internet. Hot Wallets have wallet software that includes private keys and are connected to the internet. Whereas the cold wallets have the private key stored offline [25]. Even if it looks like the

crypto coins are present on the physical wallet, there are only the private keys that are saved in a different way. Any funds are always stored on the blockchain [26].

There are different kind of hot wallets and cold wallets:

- **Hot Software Wallets**

Typically, hot wallets are software wallets that are installed on a computer and run in the background such as MetaMask [27] or Coinbase wallet [28].

- **Cold Software Wallets**

The software wallets mentioned above can also be used as cold wallets. These would then be installed on computers that are only switched on for transactions and then access the internet. This is one of the methods used at the beginning of the crypto scene.

Another way that was used early on is simply to write a recovery seed phrase on a piece of paper and enter it into the software when you need it. Once the process is complete, the software is uninstalled and is therefore essentially offline.

- **Cold Hardware Wallets**

The more common option today is to have a hardware wallet like a ledger. This can be protected with a 6-digit pin. If the pin is entered incorrectly 3 times, the private key is deleted. As a safeguard, the seed phrase is entered during the configuration of the hardware wallet.

- **Cold Exchange Wallets**

The more common option today is to have a hardware wallet such as a Ledger [29] or a Trezor [30] which store the private key on a separate device. The Ledger, for example, is protected by a 6-digit pin. If this is entered incorrectly three times, the private key is deleted. As a safeguard, the seed phrase is written down during the configuration of the hardware wallet and requested as verification again before using the device.

One advantage of these devices is that they support different wallets from different blockchains via apps. Another is increased security by physically unplugging the device from the computer.

These devices can also authenticate software wallets via the FIDO U2F standard and increase the security of the software wallets [31].

- **Mixed Exchange Wallets**

Most exchanges have a mix of both. Incoming transactions are sent to a cold wallet and outgoing transactions are used via the hot wallet. It should therefore not be possible to steal the entire capital in the event of a hack.

- **Multisig Wallet**

To secure a wallet owned to the same parts by multiple founders, it can be cryptographically protected using a Multisig Wallet. Each of the participants gets their own personal private key.

The Multisig Wallet can now be set up so that all users must enter the private key to make a transaction. However, it is also possible to set up the wallet in such a way that, for example, only two out of three users must enter their private key. This could serve as a safeguard in case one of the users is no longer able to enter the private key, for various reasons.

2.2 Decentralized Finance DeFi

The basic idea of Decentralised Finance (DeFi) is that everyone has full control over their finances. Anyone with access to Ethereum should be able to participate. Decentralised assets can be managed via decentralised apps. An example of this is that currencies can be lent without the participants knowing each other. In this way, third parties such as banks can be replaced by smart contracts [32].

Another application for this is decentralised exchanges, which will be explained in the following chapters. However, to explain this in more detail, centralised exchanges will be explained first.

2.2.1 Centralized Exchanges CEX

Centralized Exchanges manage transactions and features at a centralised infrastructure, usually backed by an institution or company. In the last few years, more exchanges had to drop the anonymity of their customers due to regulatory reasons. This means the exchanges know their customers, called KYC (know your customers). This measure should prevent the malicious use of cryptocurrencies and scams and promote anti-money laundering.

Most Centralized Exchanges have an order book, which means buyers and sellers set orders that are matched if the price drives in a certain direction. This enables the possibility to set a stop-loss and sometimes a take-profit, where a threshold can be set so that an automatic sell-order should be executed to take profit or the loss. For the matching, customers pay a percentage of the volume for fees.

Figure 6 shows the market shares of centralised exchanges. These are divided into exchanges that offer fiat, i.e., traditional currencies such as USD or CHF, and crypto-only exchanges.

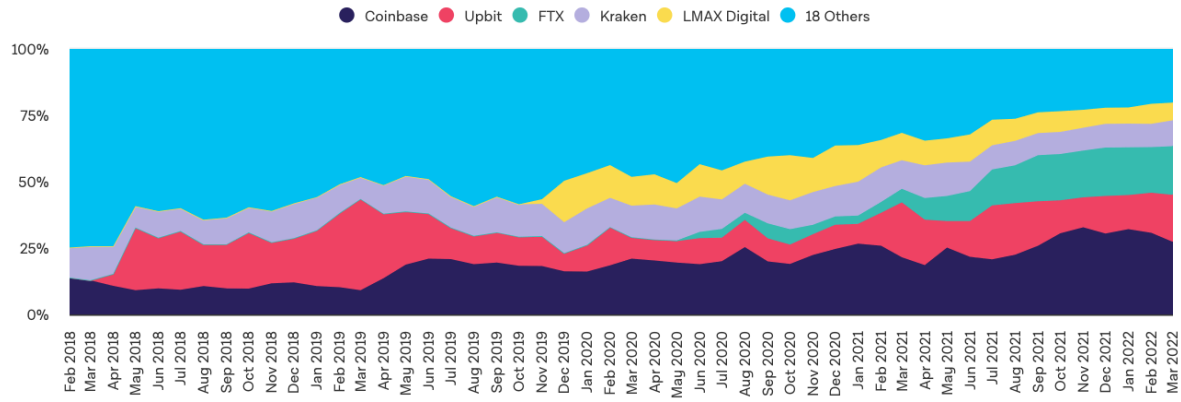


Figure 6: Fiat exchange market share from February 2018 until March 2022 [33]

Figure 7 compares the exchanges that offer crypto-only trading. It stands out that Binance has had over 50% market share since 2021. In a sample from April 4, 2022, this was also shown in the market volume of the last 24 hours. With $23 * 10^9$ USD, it has four times higher market volume than the number two OKX with $5.6 * 10^9$ [33].

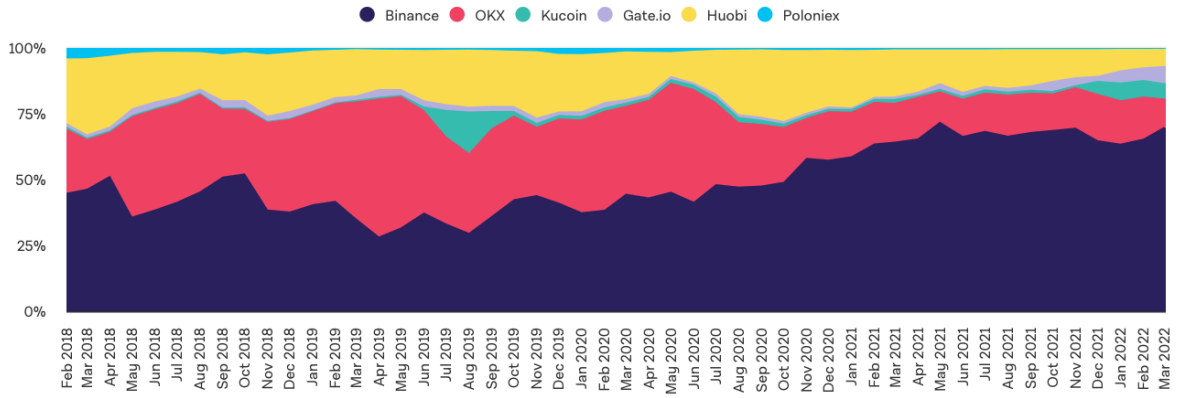


Figure 7: Crypto-only exchange market share from February 2018 until March 2022 [33]

Exchanges present advantages, for example, if a scammer tries to trick people; in such a case, the exchange can ban the address from the platform, not allowing its users to send funds to certain addresses. Another example is that if you lose your password to the account. It is possible to recover your funds through support. This is possible since they own the private key to your accounts.

However, this opportunity leads directly to the disadvantage of private keys being owned by the exchanges. They own all private keys. In crypto, there is a saying: “Not your key, not your coins!” With a private key, one essentially has full access to a wallet. The worst-case scenario is that the exchange key is stolen, and the funds are withdrawn. Since there are security measures this happens rarely, but if it happens, the impact is huge. In particular, this happened to Mt. Gox in 2014 the exchange handled more than 70% of the Bitcoin Transaction at

the time. The hackers were able to 850'000 BTC of which 750'000 BTC belonged to their customers, which were worth 450 million USD at the time and billions right now [34].

2.2.2 Decentralized Exchanges DEX

Decentralized Exchanges address the weaknesses of a centralized exchange by putting the exchange on the blockchain. Therefore, everything is handled peer-to-peer without an intermediary.

In this kind of exchange, there are no KYC requirements and do not have any kind of customer support. This means that on one hand that no company that can interfere with the transactions and there is no way of getting a hacked or forgotten private keys back. On the other hand, this provides maximal anonymity and flexibility. Security can be maximised by carefully selecting the appropriate wallet.

There are also disadvantages in trading. A trade is dependent on the gas fees and are paid statically per transaction. These can be up to 400 USD per trade, as confirmed in a forum [35] and have already been seen by the authors. Another feature that most exchanges provide, which decentralised exchanges do not, is a stop-loss or a take-profit. These are there to track targets when trading. These are especially necessary for volatile assets like cryptocurrencies, especially because they are traded around the clock.

2.2.2.1 Top Decentralized Exchanges

Since June 2021, Uniswap has been hovering at around 50% of the market share by volume. It is the most used decentralized exchange today. It supports Ethereum and the following Layer-2 networks: Polygon, Optimism, and Arbitrum [36]. Uniswap has different versions that operate independently. Since the contracts have renounced ownership as described in Chapter 3.3.6, they cannot be updated, thus all of those versions are still running separately on the main Ethereum blockchain.

In Figure 8, the shares of DEX volume are shown as a percentage. Besides Ethereum, these DEX run on other platforms. A fork of Uniswap for the Binance Smart Chain is PancakeSwap [37], which is based on Uniswap v2. Serum DEX [38] is running on the Solana Platform. The last major DEX is Dodo [39], which offers transactions on and between multiple platforms such as Ethereum, Binance Chain, and Layer-2 networks. The main message of this picture is that Uniswap, with versions v2 and v3, which each have a very large market share of the DEX Volume.

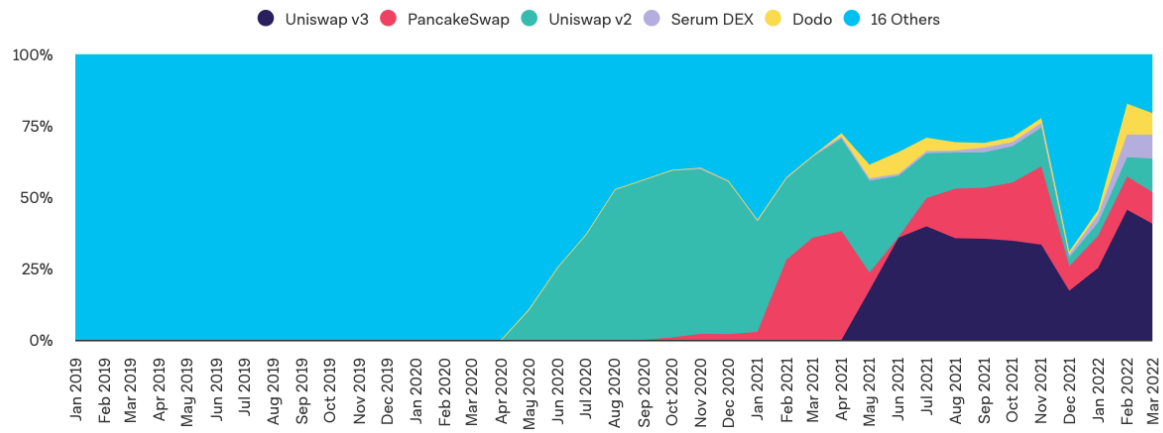


Figure 8: Share of DEX Volume from January 2020 until March 2022 [40]

2.2.2.2 Liquidity Pools & Automated Market Maker

Centralized exchanges use order books to match buyers and sellers. Some older decentralized exchanges like EtherDelta [41] also use the order book model. The problem is that liquidity for smaller ERC-20 tokens is poor, which makes the currencies difficult to trade. Since a transaction is necessary for a buy or sell order, gas fees must be paid. This has major disadvantages in the volatile crypto market. Orders cannot be placed as quickly, as the transactions take place in a short time, and the price could already change drastically at this time.

2.2.2.3 Liquidity Provider

For example, if it is assumed that the USD and USDC are worth the same. The ETH rate is assumed to be 2'500 USD for 1 ETH. For a liquidity provider to be able to add 1 ETH to the liquidity pool, he must simultaneously add 2'500 USDC. At this point, the Liquidity Provider agrees to buy and sell ETH or USDC at any time. For this service, he receives Liquidity Provider Tokens or LP-Tokens for short. This entitles him to receive trading fees from others. The fees, which in Uniswap v2 would be 0.3%, are paid out in proportion to the number of LP-tokens that are circulating. If a Liquidity Provider wants to sell the LP-tokens to get whatever balance of the traded capital back he needs to burn the tokens, this will lead to a reduction of circulating LP-Tokens. Since there are fewer LP-Tokens circulating, the other Liquidity Providers will have a higher stake in percentage [42].

In a formula, it is defined as follows:

$$x * y = k \tag{1}$$

Where x and y represent each a quantity of tokens, whereas k is the constant. The liquidity is distributed uniformly, which provides the liquidity all over the price range from zero to infinity $(0, \infty)$. As mentioned, the liquidity pool provider needs to provide the same worth of tokens x and y, so the constant k remains in balance.

2.2.2.4 Trades on a Liquidity Pool

To show how a trade with a liquidity pool works, the following example is taken. Please note that to maintain legibility, the numbers in the examples are rounded to two decimal places.

We assume we must tokens x_{pool} and y_{pool}. Both have the following amounts to fill the current liquidity pool, which leads to the following constant k:

$$x_{pool} = 25'000$$

$$y_{pool} = 25'000$$

$$k = x_{pool} * y_{pool} = 25'000 * 25'000 = 625'000 \quad (2)$$

If a buyer wants to trade 2000 units of $token_x$, the liquidity pool calculates how many $token_x$ it would have after the trade by just adding those numbers together as follows:

$$x_{trade} = 2'000,$$

$$x_{new\ pool} = x_{pool} + x_{trade} = 25'000 + 2'000 = 27'000 \quad (3)$$

By transforming Equation 1, the liquidity pool calculates how many tokens y should be left in the pool after the trade as follows:

$$y_{new\ pool} = \frac{k}{x_{new\ pool}} = \frac{625'000'000}{27'000} = 23'148.14 \quad (4)$$

Now the difference between the new and old pool amount is the amount of the token that the trader will get Equation 5:

$$y_{trade} = y_{pool} - y_{new\ pool} = y_{pool} - \frac{k}{x_{new\ pool}} = 25'000 - \frac{625'000'000}{27'000} = 1'851.85 \quad (5)$$

Note that in this example the trading fee is neglected. After this calculation, the trading fee would still be calculated by deducting the fees from y_{trade}.

To check whether Equation 6 still applies, the new number of tokens in the pool can be checked with the constant k as follows:

$$k = 625'000'000 = x_{new} * y_{new} = 27'000 * 23'148.14 \quad (6)$$

2.2.2.5 Price Calculations of Tokens

The price of the tokens is based on the amount of ETH inside the liquidity pool. On the Ethereum blockchain, ETH is seen as the base of the value.

This is because the individual tokens cost the exact share that is in the liquidity pool. It is assumed that the previously explained liquidity pool has a value of 500 ETH. In this case, the prices of the tokens before the trade were:

$$x_{worth} = \frac{k_{worth}}{x_{amount}} = \frac{500\ ETH}{25'000} = 0.02\ ETH \tag{7}$$

That brings the price per Token to $0.2\ ETH$. After the trade with x_{new}, the worth of the tokens x declines since the supply increases as follows:

$$x_{worth\ new} = \frac{k_{worth}}{x_{new\ amount}} = \frac{500\ ETH}{27'000} = 0.0185\ ETH \tag{8}$$

In this example, trading 2'000 tokens has a high impact on the price. If the liquidity pool had more than 500 ETH in it, the price impact would be less and the price would not fluctuate so much.

This is one of the problems of the AMM, since the price could spike up for trades with high volume. The solution referred to here is arbitrage. If people can buy tokens on the DEX for less than on a CEX, they can buy it up on the DEX and sell it on the DEX so the price of the DEX would align more to that of the CEX.

2.2.3 Uniswap

Uniswap itself is a protocol for decentralized and permissionless ERC-20-token exchange on the Ethereum Blockchain, created by Uniswap Labs. The protocol is implemented in multiple smart contracts in a persistent and non-upgradable way. Together it makes the AMM, which means Automated Market Maker. Together with the Uniswap Interface, it becomes widely available across the globe [43].

2.2.3.1 Permissionless System

As mentioned briefly, Uniswap works in a permissionless system. It can be used without an account; the only thing needed is a compatible wallet. That means that Uniswap has no restrictions on age, geography, or wealth status [43].

A major advantage of Uniswap and similar decentralized exchanges is the independence of geopolitical restrictions. The US for example, has stronger regulations than Europe. This is why the exchange Binance announced 2019 in a blog post [44] that they will open Binance US next to the regular Binance exchange. The selection of the trading pairs on the US Exchange is not as sizeable as the main one since certain coins are not allowed to be traded by the US citizens.

Another factor is that Uniswap does not need to know who you are. An anonymous account at Binance allows you to withdraw 0.06 BTC per day without verification [45]. The verification is very strict. Next to an ID, they need a face verification and, if you want higher limits,

a confirmation of residence. The situation on Uniswap is completely different. Since you change your tokens over the blockchain, you can exchange as few or as many tokens as you want.

2.2.3.2 Versions and Update

Uniswap was built on purpose in a non-upgradable way. That means once a version is deployed to the Blockchain it cannot be modified. Furthermore, it means that the versions coexist. To this day, Uniswap Labs has deployed three versions of the Decentralized Exchange.

- **Uniswap v1**

  Uniswap v1, released on November 2, 2018, was the first version of the protocol on the Ethereum mainnet [46]. The contract is written completely in the programming language Vyper, which is an alternative to Solidity [47]. The main feature was the implementation of the AAM. There were only Liquidity Pools bound to ETH. This means that trades from ERC-20 to ERC-20 tokens, e.g., Link to DAI, need to be converted from Link to ETH first and then to the second ERC-20 token, which would be DAI in this example. This results in higher gas fees and longer execution time since there are more instructions on the EVM.

- **Uniswap v2**

  The second iteration Uniswap v2 was deployed May 4, 2020, on Ethereum [48]. In summary, the following changes in Uniswap v2 are particularly significant:

  1. As mentioned, trades between ERC-20 tokens were particularly expensive and lengthy. To address this problem, v2 implemented the ability to create ERC-20 to ERC-20 Liquidity Pools [49].
  2. Furthermore, it is possible to make Flashswaps, which are equivalent to a marked order, i.e., an immediate purchase [50].
  3. On the technical side, the smart contract was written in Solidity instead of Vyper [51].

- **Uniswap v3**

  The latest version of Uniswap v3 was uploaded on the May 5, 2021, on the Ethereum Blockchain [52]. The biggest change to Uniswap v3 is the ability to put liquidity into custom ranges instead of the whole range $(0, \infty)$ [42].

  The main reason for this is that the range $(0, \infty)$ is too large and that most of the liquidity was never touched. For example, when looking at stable coins, they should always be in the range around the USD. Here, liquidity between 0 and ∞ is very inefficient. Liquidity providers can set their liquidity, for example, between 0.9 and 1.1, which

increases their efficiency enormously. Figure 9 shows a graph illustrating how the efficiency of such liquidity can be used [42].

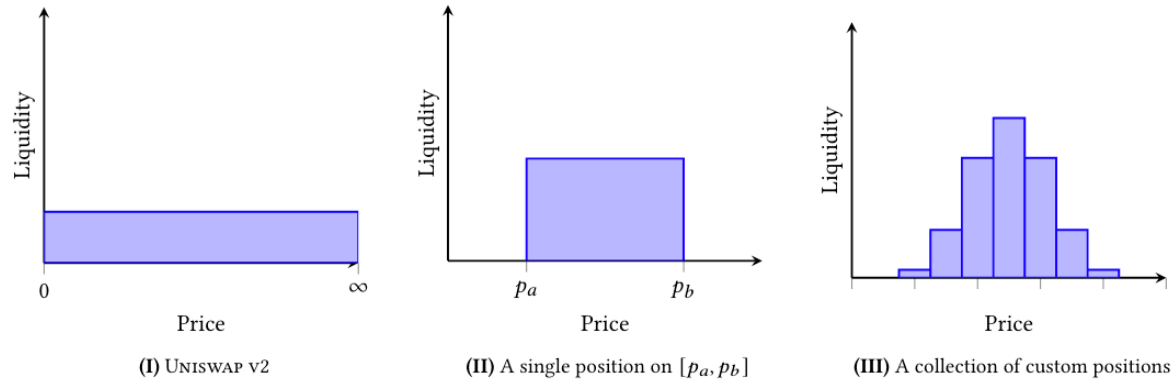


Figure 9: Example of the liquidity distribution on Uniswap v2 to custom ranges [42]

Another change is that the fees can be adjusted per poll between 0.05%, 0.3%, and 1%, and adjusted fees can be enabled with the UNI Token. The last major change is the support of NFTs. There are other changes are in Uniswap v3, but they are not discussed in this work [42].

2.3 Threat Landscape for Decentralized Finance

There is a variety of threats to smart contracts in the decentralized finance space. Some of the described threats also apply to the traditional stock market while others are specific to smart contracts. The threats can come from the developers or also from external sources like hackers.

What they all have in common is that they are driven by monetary factors. As can be read in Figure 10, the largest known hack on smart contracts to date is worth $6 * 10^8$USD.

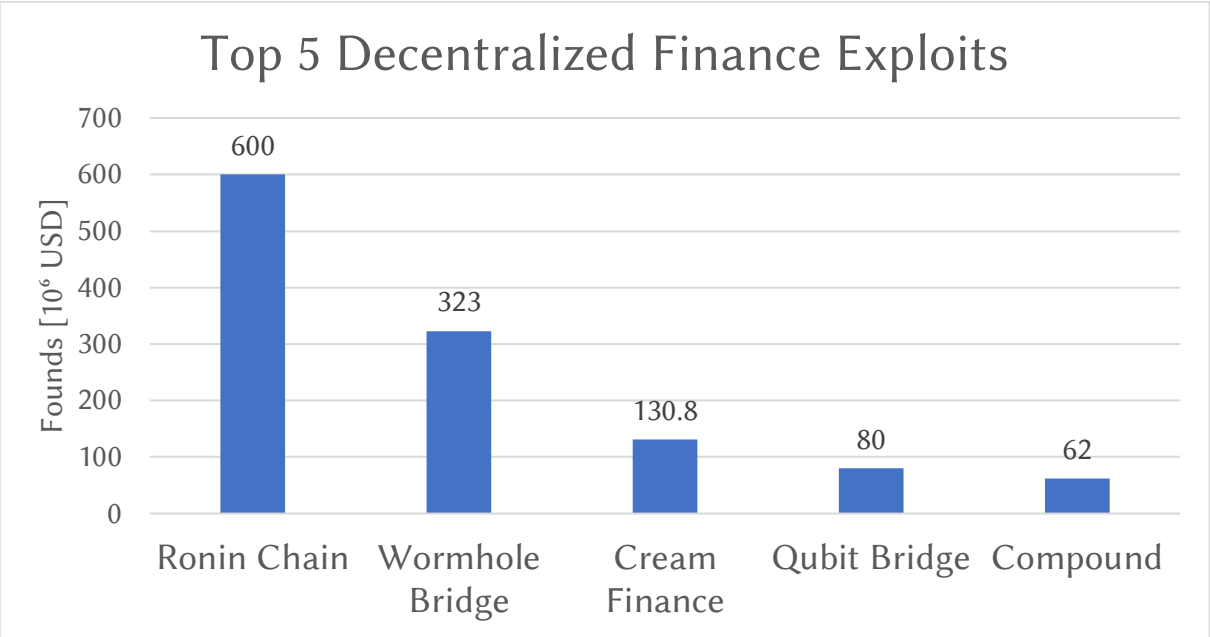


Figure 10: Top Decentralized Finance Exploits [53]

There are other threats that keep profiting from the unregulated markets. *Pump and Dumps* for example are illegal in the traditional stock market, but considered a grey area in the crypto space. The following subchapter describes some of the previously mentioned dangers in more detail.

2.3.1 Admin Keys

A smart contract is deployed from an account with a wallet. This wallet is therefore the owner of the smart contract. The owner can manage and update the smart contract if he does not take any further measures.

As already discussed in Chapter 2.1.4 Wallet Types, wallets are bound to their respective key-pair. The keys that are bound to the smart contracts are often called admin keys or god mode [54].

These can be used to add new features or patches in updates. This is especially the case with smart contracts that are still in the initial phase. An example of this would be AAVE, which keeps the admin keys for security reasons [54].

The problem that arises is that these keys can be hacked if the creators do not store them securely. This is a single point of failure if no Multisig wallet is configured.

Another problem is that a developer can push malicious code or a bug intentionally to make the smart contract vulnerable. This risk can be mitigated by having a good development team with a Multisig wallet. However, if the whole team is corrupt, that does not help either.

2.3.2 Vulnerabilities

As with most software, it is possible that the smart contract related code has been written incorrectly and contains bugs. Millions of USD worth of cryptocurrencies can pass through smart contracts. This makes it particularly attractive to attackers.

As mentioned in Chapter 2.1.3 Solidity, most EVM compatible Smart Contracts are written in Solidity. A well-known list of identified weaknesses is the Smart Contract Weakness Registry or SWC-Registry [55] for short. It currently consists of 36 vulnerabilities of smart contracts written in Solidity. To this end, Slither is a tool used for this work to perform static analysis for open-source smart contracts. More information is provided in Chapter 3.3.2 Vulnerability Scanner: Slither.

2.3.3 Rug Pull

The Rug Pull is an exit scam that removes liquidity from the liquidity-pool and thus makes the asset worthless and untradable. A Rug Pull is one of the most popular scams in the crypto scene, especially on DEX. The reason for this is that smart contracts can be easily uploaded at no additional cost for listing, while on CEX it can cost 250'000 USD according to a news article [56]. Another reason is that DEX does not require any security audits before listing, the only requirement is to have an incentive to attract potential investors.

To help the popularity of this scam, prices are often manipulated and hype is created around the project to lure investors into the trap. This is done by creating a token in which only the developers are the liquidity providers. Investors buy the token and give mostly ETH as currency. This increases the price of the scam token. After a while, the liquidity pool has much more ETH and less of the scam token. In this case, the developers, as the only liquidity provider, can withdraw all the liquidity from the liquidity pool. This has two effects. First, the price of the token falls to zero. Secondly, since there is no liquidity provider and the liquidity pool is empty, no trades can be made and the investors are stuck with the worthless tokens [57].

2.3.4 Sandwich Attack and Front Running

A sandwich attack is initially based on the concept of front running. Additionally, the attacker makes sure to force his trailing transaction as close to the targeted buy as possible.

Front running is comparable to insider trading. In the traditional stock market, this is prohibited. An example of this would be that a bank employee learns from a major investor that he is going to buy a certain share. The employee tells a third person who buys the share before the big investor buys the share. Once the big investor buys the share, the third person has unrealized profit which could be realized into quick profits.

In the cryptocurrency environment, this is even easier. All pending transactions are stored in the so-called *Mempool*, which is located in the memory of the EVM [58]. The pending

transactions are visible to everyone. Therefore, it is possible to search in the pending transactions for large transactions of tokens if this transaction has average gas fees that take a few minutes to process. This gives third parties which is usually a bot monitoring the blockchain the possibility to attack this transaction.

A hypothetical attack could look like this:

> *Alice makes a trade via Uniswap and exchanges 10 ETH for x tokens. On the Uniswap front end, she is given an indication of how many tokens she will get for the current price.*
>
> *As soon as she has sent the transaction, it ends up in the Mempool, where anyone can view it. Eve sees this large transaction with her bot. Eve analyses Alice's gas fees and sees that they are at 30 Gwei. As these are the default fees, Alice will have to wait about 3 minutes for the execution. Eve takes advantage of this and sends two transactions at once. The first one is the buy order with 31 Gwei, this one comes before Alice's buy order. The second is a sell order with the same gas fees as Alice's buy order. The Mempool at this point will look like the one shown in Figure 11.*

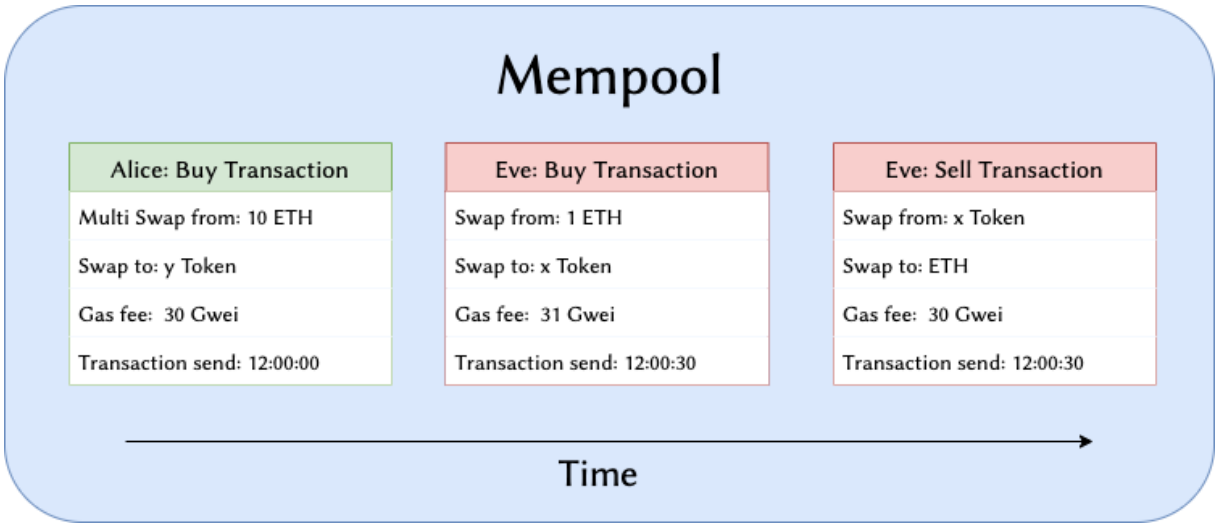


Figure 11: Mempool transaction sandwich attack

Now the miners decide to accept the transactions as follows. As the higher gas fees are always taken first, Eve's buy order comes in, as she paid the most with 31 Gwei. Then come both orders, with 30 Gwei. Since Alice's buy order is received first, it is executed first, followed by Eve's sell order. Alice's order was sandwiched in the middle of Eve's buy and sell, hence the name. However, Alice's price has now changed because Eve has beaten her to it. When she executes it, she will notice that the price has changed. However, by then it is already too late, and the transaction is complete. The executed transactions are shown in Figure 12.

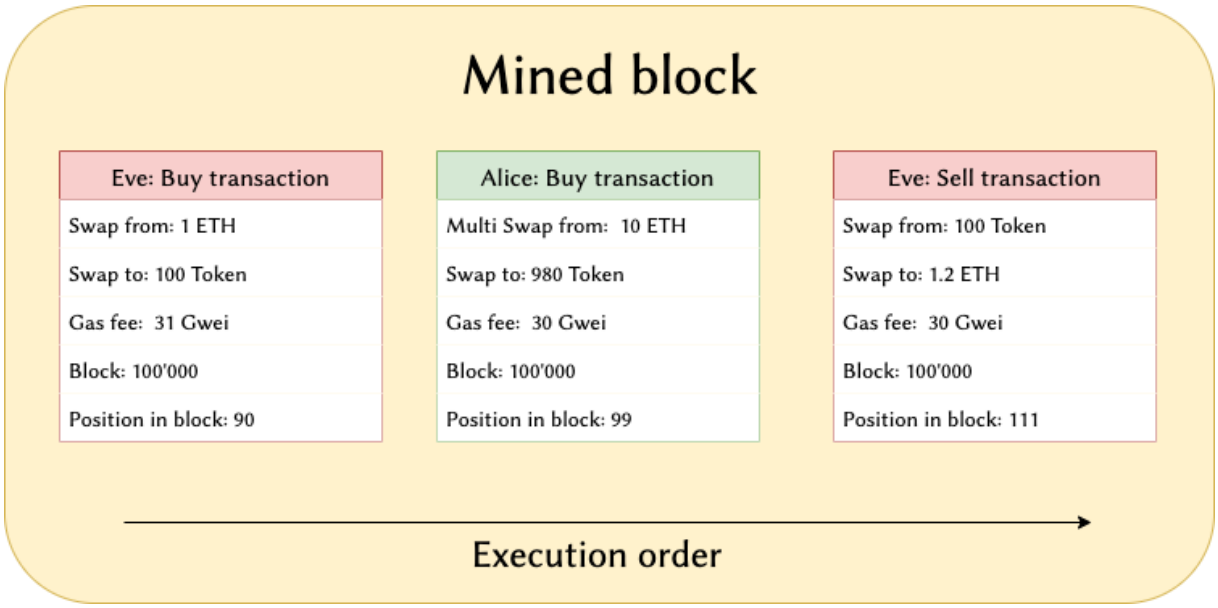


Figure 12: Mined block transactions sandwich attack

In this example, Eve made 0.2 ETH profit. Alice could have avoided this by changing the slippage tolerance of Uniswap in the settings as can be seen in Figure 13. If Eve had forced her transaction before Alice's using high gas fees, Alice's transaction would have failed and the attack would no longer be possible. Therefore, investors should be aware of how high the price impact of their orders will be.

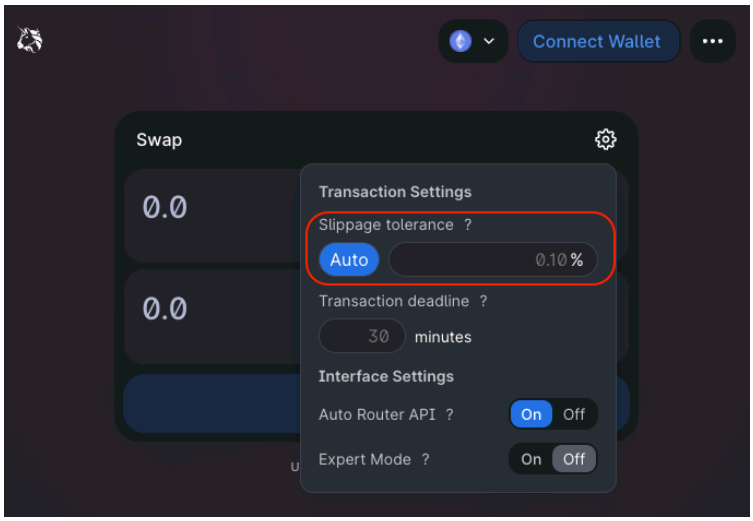


Figure 13: Prevent sandwich attack with slippage tolerance on Uniswap [59]

2.3.5 Whales

Whales are investors, who own a large part of the circulating token supply. They have so much that they can strongly influence the price of cryptocurrencies. This is especially the case for cryptocurrencies with a low market cap.

They are not a classic danger. Because cryptocurrencies are hardly regulated, they can collude and psychologically influence investors to make more profits for themselves.

2.3.6 Pump and Dump

Pump and Dump is a form of market manipulation where the price is artificially moved. The organiser buys in advance a cryptocurrency that has a low market volume. These are usually in a downtrend or going sideways. This is usually done over a longer period so as not to change the price much.

Once this is done, the organiser will notify people via social media such as Telegram or Discord. *Pump and Dump* will take place on a specific date on a specific exchange. The organiser lures with high winnings and fast money. The participants get ready to buy a token on the exchange as quickly as possible. As soon as the cryptocurrency becomes known via the corresponding platform, the price skyrockets and the organiser sell his cryptocurrency at a high price, while the participants often make a high loss.

An example from a study [60] can be seen in Figure 14. The price of the cryptocurrency is relatively stable before the announcement of the coin and the market volume is low. As soon as the name of the cryptocurrency is known, it quickly jumps up. Afterwards, the price fluctuates strongly and falls more slowly.

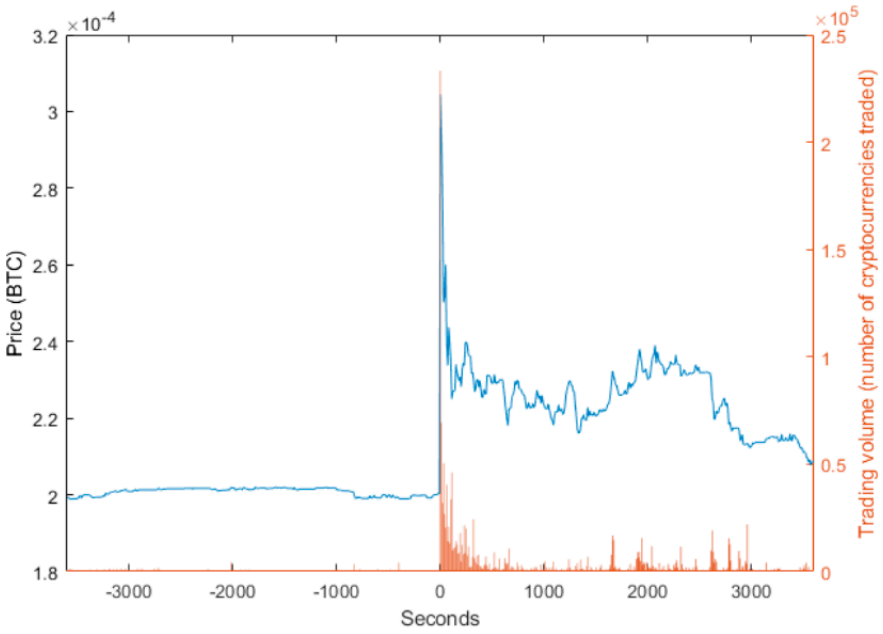


Figure 14: Nexus Price and volume during a Pump and Dump (t=0) [60]

This kind of trading is illegal in stock exchange trading. In the USA, this is being investigated by the U.S. Securities and Exchange Commission (SEC). But there are hardly any regulations on cryptocurrencies [60].

2.3.7 Honeypot

Honeypots in smart contracts are essentially traps for investors. The smart contract is designed so that only selected addresses can sell. This leads to no sales of the token. Due to the lack of sales, the only possibility is that the price rises substantially. The developer lets the

worthless tokens pump by the investors and cash out the profits. The investors' tokens remain worthless since they cannot sell their tokens.

For investors, it looks like the price is skyrocketing and everyone is just buying. This leads to the so-called FOMO or "fear of missing out" and investors enter a trade before the price goes even higher, not knowing that they will lose the money.

Sometimes the scammers try to make it look like people can sell the tokens by allowing multiple of their own wallets to sell tokens. As a result, it looks like it is a successful launch with a few "natural" selloffs.

2.3.8 The Social Dimension of Smart Contracts Scams

It is a tightrope walk to distinguish between meme tokens and scams. Dogecoin is currently one of the top 20 cryptocurrencies in existence. It started as a fork of Litecoin, but has had only slight adjustments. The cryptocurrency should be fun and more attractive than Bitcoin because of its light-hearted image [61].

The next cryptocurrency took the hype around the Dog that is used on Dogecoin as a mascot and a new Token. The cryptocurrency called Shiba Inu or the SHIB Token took the same formula one step further putting the token on Ethereum and creating their own DEX named Shibaswap. Moreover, to get even more attention, they declared their currency as a "Dogecoin Killer". This has also led to this cryptocurrency's creation to date (April 4, 2022) on CoinGecko as currency number 15, just behind DOGE at number 12 [62].

The dangerous part is when the scams masquerade as memes or hype cryptocurrencies. People are trying to find the next DOGE or SHIB to make big money. The scammers take advantage of this to perform rug pulls or honey pots in such projects. In Chapter 3.2.2, a common scam analysis of how a current hype can be exploited is explained.

3 Methodology

The approach of this work is to address some of the previously described threats of the threat landscape in separate case studies. The theory will be applied to the results of the common scam analysis to identify requirements for successful attacks. The comparison of the theory to actual scams consolidates the technical understanding of what a malicious developer needs to implement for a successful scam. Based on those observations, theoretical indicators will be designed and their strengths and weaknesses estimated. If the identified indicators appear effective enough, they will be implemented into the SCSC system and tested during the next chapters.

Additionally, this chapter will show an overview of the SCSC software structure and architecture as well as defining the workflows of the application.

3.1 Software structure

To develop the SCSC, Python is used in the backend and React in the frontend. The same frameworks and programming languages were used as in the project work [1]. Python was used because Slither was one of the indicators written on Python. FastAPI [63] is used to provide an interface to Python. In contrast to the project work, the frontend was not written from scratch, but a template [64] was implemented to provide a user interface.

It was decided to use MongoDB. The expected persistent data fits well into a document-based database and it allows for some flexibility during the development process. Additionally, it is easier to store nested, dynamic data structures in MongoDB.

To give an insight into the project, screenshots are shown in Figure 15. It shows an overview of the latest tokens and their ratings. Figure 16 shows the upper part of the detail page, where the most important information is displayed.

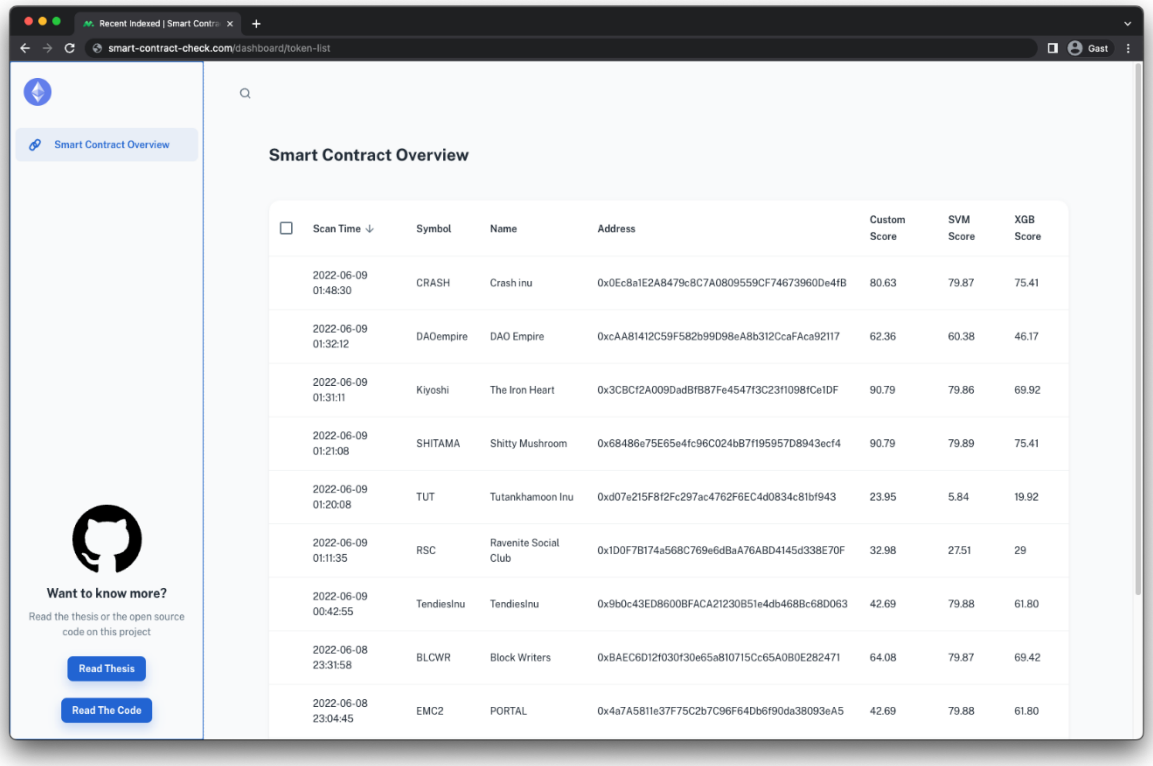


Figure 15: SCSC Overview

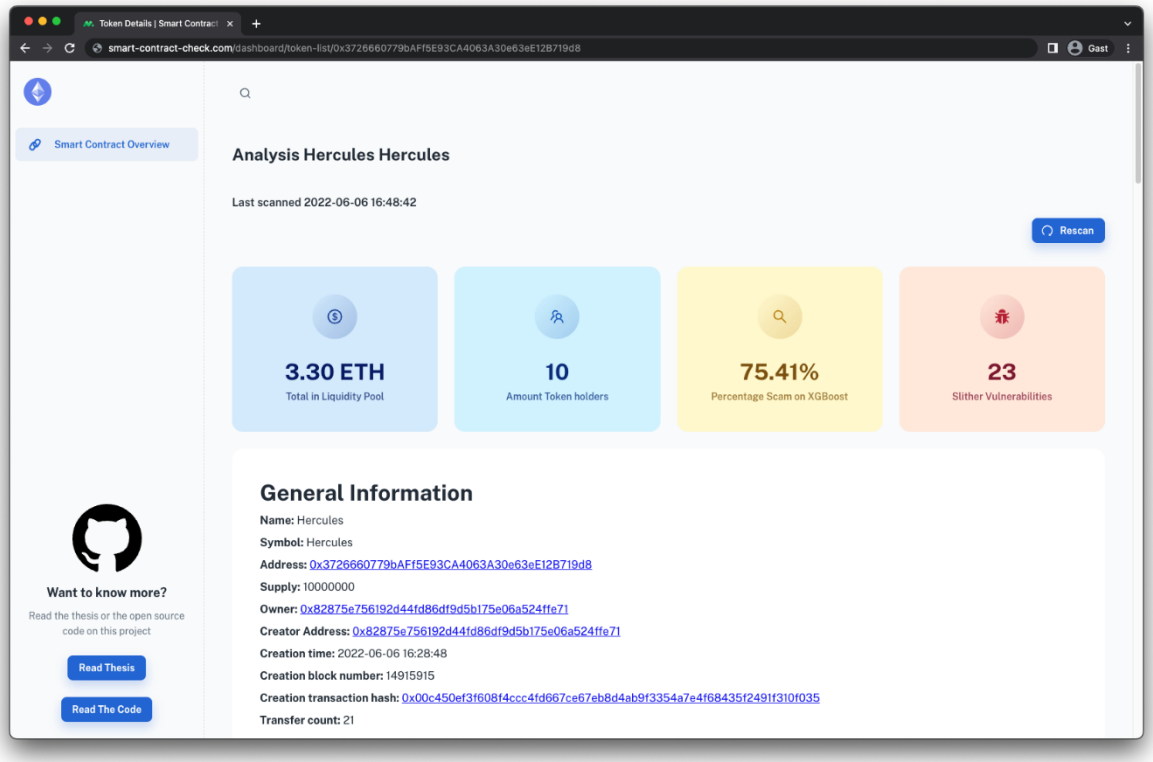


Figure 16: SCSC Detail Page

3.1.1 Architecture

The architecture can be seen in Figure 17. The user interacts with the frontend on which the most recently analysed smart contracts are displayed. The React frontend communicates with the backend via FastAPI. The controller takes care of queries for the frontend, if the user wants to look at a specific smart contract, it queries MongoDB directly and returns it to the user via the FastAPI. If the user reads in a new smart contract, the controller directly addresses the collector and runs the indicators through it. The controller then saves the data in MongoDB. The user can now view the data on the detail page of the smart contract.

The heart of the SCSC is the monitoring function. The monitor polls the Infura [65] API, which offers endpoints to register for on-chain events, for newly created Uniswap V2 pairs. The monitor places the addresses of the new smart contracts in the worker queue, which are processed by the worker. The worker passes the data on to the collector, which also passes the data on to the indicators. The worker then receives the data and stores it inside the MongoDB database. If an error occurs somewhere, logs are written to the file storage. To simplify matters, the file storage was drawn to the FastAPI, since the python logging root is initialized on launch of the FastAPI application.

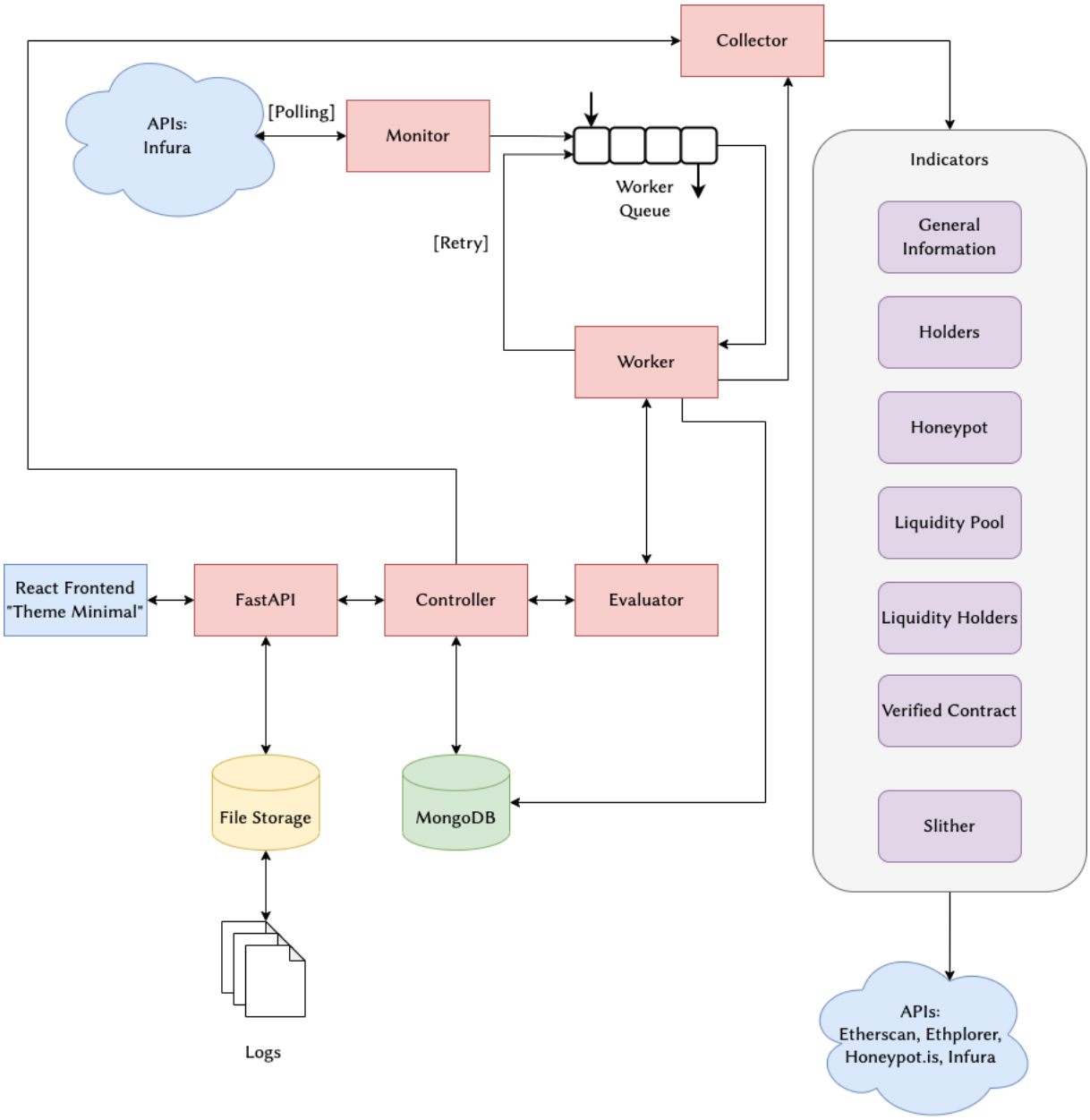


Figure 17: SCSC Architecture

3.1.2 Activity Diagram

Activity diagrams were created to improve the description of the SCSC system. The activity diagram in Figure 18 shows exactly how the monitor operates. To access the data of the smart contracts, the Infura monitor asks for the latest *PairCreated* events. If no new pairs are available, it waits T_p seconds until the API is queried again, in other words, the API is polled. If new pairs have been found, the system waits asynchronously in a loop every T_{poll} seconds until the next call is made and continues with the respective analysis. T_{poll} was selected as 30 seconds. To ensure that the API's, which are used to acquire data for further analysis, have a chance to update their database, the monitor waits for T_{wait} seconds, which is set to 300 seconds, until it puts the targeted token into the API worker queue.

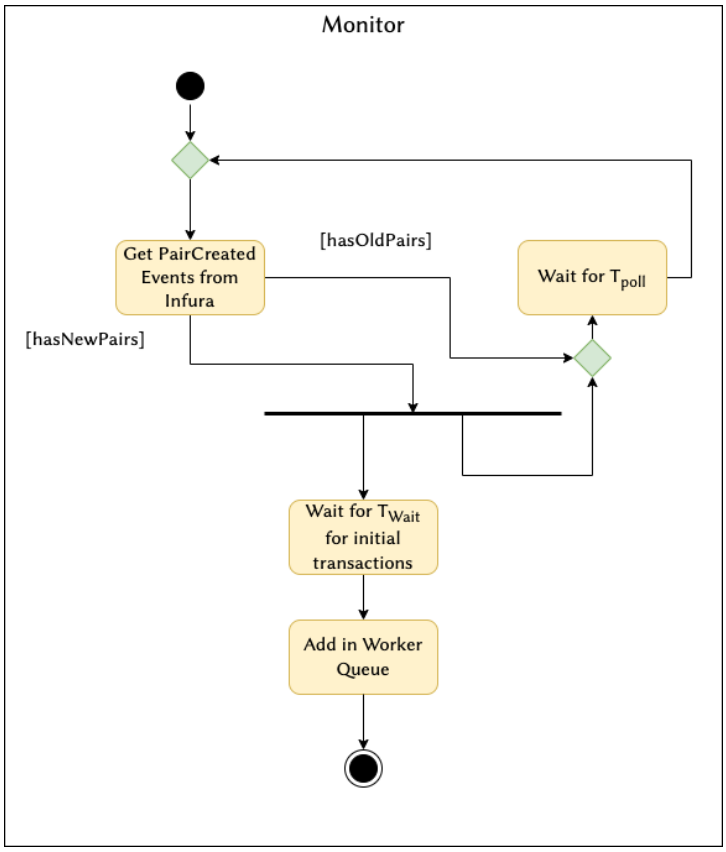


Figure 18: Activity diagram monitoring workflow

Figure 19 shows the two scenarios of how new smart contracts can be analysed by the system. In the first variant, the worker constantly takes addresses from the queue and passes them on for analysis. As soon as the analysis of the data is finished, the data is stored in the monitoring DB.

If the analysis finishes without problems, the collected data is stored inside the monitoring DB. However, if one of the external API's fails to provide essential data, the worker will store the address and information regarding the API error inside the failed tokens database and wait $2^{n}_{retries}$ seconds to reinsert the contract address at the end of the worker queue. If the system is unable to collect information about a smart contract more than $n_{retries}$ times, where $n_{retries}$ is defined as 5, the smart contract addresses as well as information about the API error is stored inside the dropped token database and no further retries are initiated.

For the second variant, a user manually submits a smart contract to the system for analysis. This can also be a previously created smart contract, which is then filled with new data. If the data comes from the manual submission, the data is saved in a separate manual submission database.

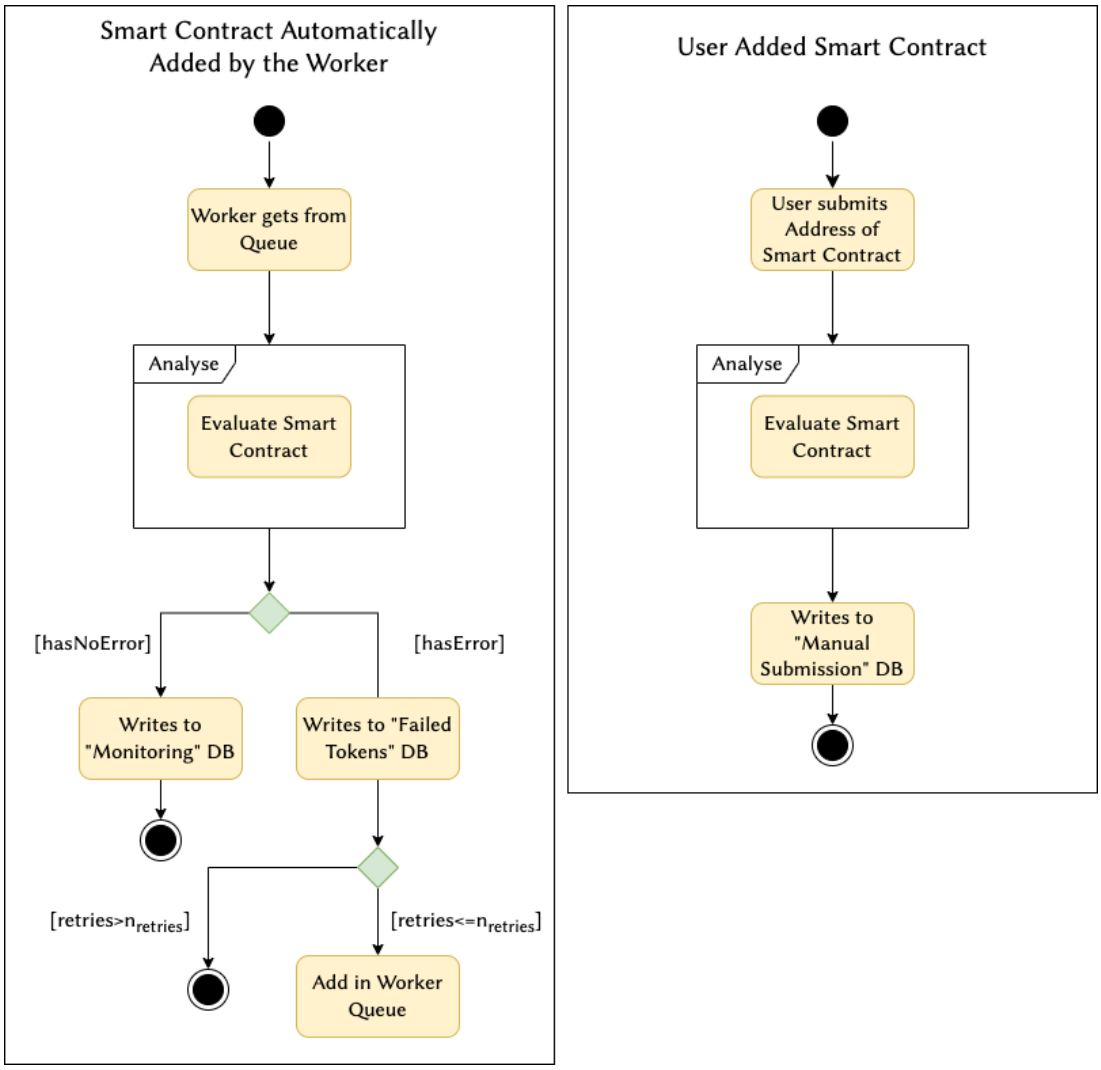


Figure 19: Activity diagram contract processing workflow

Figure 20 shows how the analysis is carried out. The data is collected in the collector, which triggers the individual indicators. Each individual indicator is self-contained. This means that modules can easily be added or removed. By inserting a module, x and y can be changed. Each module accesses the used API itself. This can lead to duplications, but these are negligible. The individual indicators are processed sequentially. Slither can only be executed if the smart contract is verified and is therefore called in the Verified Contract indicator. Once the indicators have returned the data, it is evaluated by the evaluator. Once the evaluation is done, the data is returned and written into the corresponding database.

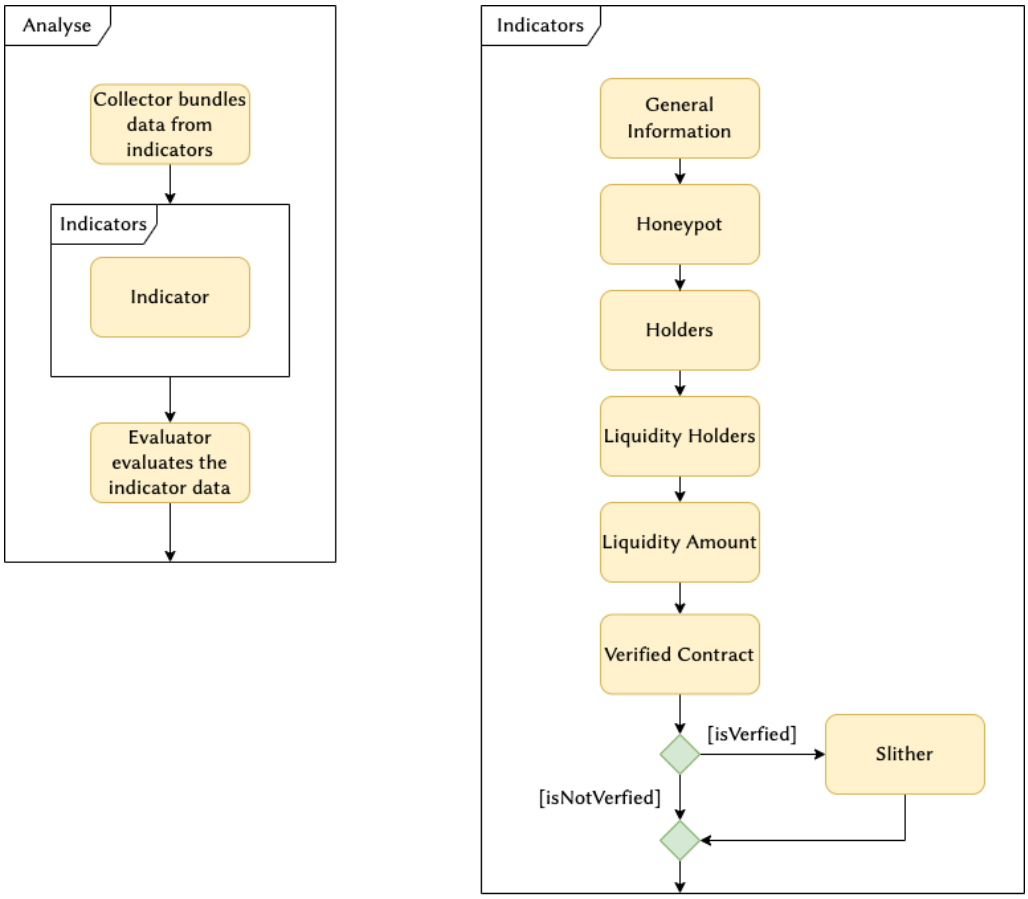
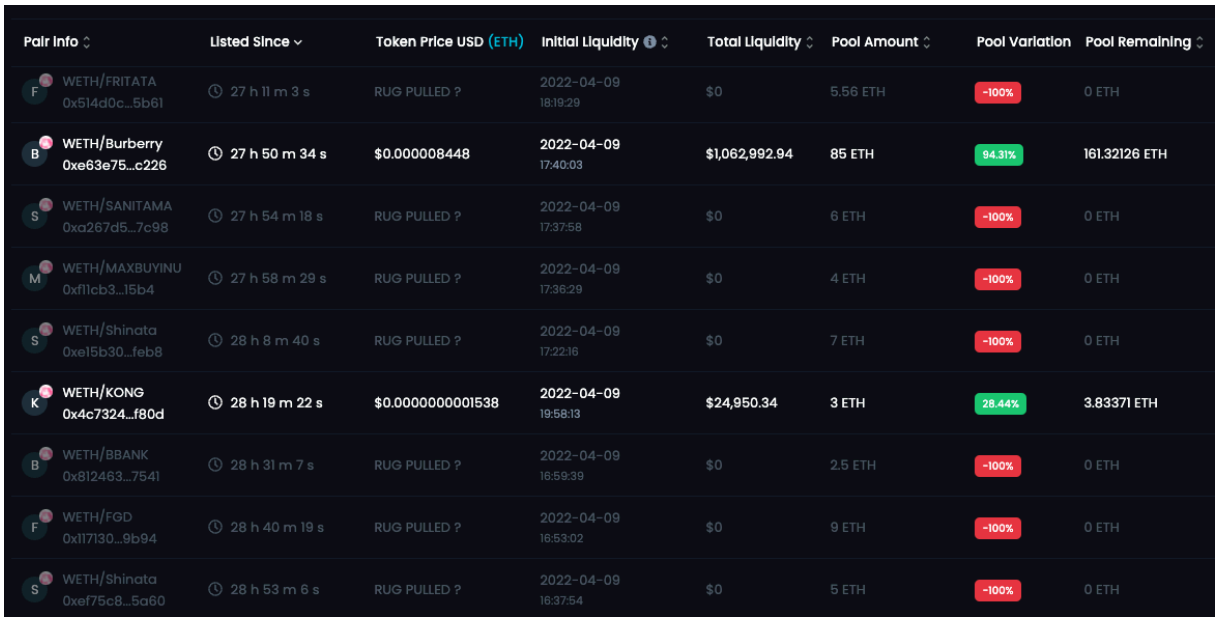


Figure 20: Activity diagram data collection workflow

3.2 Common Scam Analysis

To conduct the common scam analysis, DEXTools [66] was used to efficiently identify smart contracts from the relevant criteria. The platform offers practical information on smart contracts deployed on decentralized exchanges that can be used for security analysis. Especially for finding new smart contracts, the "Live new pairs" view is very helpful [67].

During the first analysis of the "Live new pairs", it was noticed that DEXTools reports a large number of possible rug pulls. This can be seen in Figure 21. On April 9, 2022 alone, 48 were identified as rug pulls out of 95 deployed smart contracts in 24 hours. This sample was verified on April 4, 2022, at 21:00. It is possible that some smart contracts will perform a rug pull even later.



Pair Info	Listed Since	Token Price USD (ETH)	Initial Liquidity	Total Liquidity	Pool Amount	Pool Variation	Pool Remaining
WETH/FRITATA 0x514d0c...5b61	27 h 11 m 3 s	RUG PULLED ?	2022-04-09 18:19:29	$0	5.56 ETH	-100%	0 ETH
WETH/Burberry 0xe63e75...c226	27 h 50 m 34 s	$0.000008448	2022-04-09 17:40:03	$1,062,992.94	85 ETH	94.31%	161.32126 ETH
WETH/SANITAMA 0xa267d5...7c98	27 h 54 m 18 s	RUG PULLED ?	2022-04-09 17:37:58	$0	6 ETH	-100%	0 ETH
WETH/MAXBUYINU 0xf11cb3...15b4	27 h 58 m 29 s	RUG PULLED ?	2022-04-09 17:36:29	$0	4 ETH	-100%	0 ETH
WETH/Shinata 0xe15b30...feb8	28 h 8 m 40 s	RUG PULLED ?	2022-04-09 17:22:16	$0	7 ETH	-100%	0 ETH
WETH/KONG 0x4c7324...f80d	28 h 19 m 22 s	$0.0000000001538	2022-04-09 19:58:13	$24,950.34	3 ETH	28.44%	3.83371 ETH
WETH/BBANK 0x812463...7541	28 h 31 m 7 s	RUG PULLED ?	2022-04-09 16:59:39	$0	2.5 ETH	-100%	0 ETH
WETH/FGD 0x117130...9b94	28 h 40 m 19 s	RUG PULLED ?	2022-04-09 16:53:02	$0	9 ETH	-100%	0 ETH
WETH/Shinata 0xef75c8...5a60	28 h 53 m 6 s	RUG PULLED ?	2022-04-09 16:37:54	$0	5 ETH	-100%	0 ETH

Figure 21: DEXTools "New Live Pairs" around 17:00 on 09.04.22 Screenshot taken on 10.04.22 around 28 hours after listing [67]

3.2.1 Honeypot and Rug Pull (ZILLA)

The following smart contract was used to show an example of a rug pull:

Zilla with the same symbol name was deployed at the token address:

0x8F1e8F0942f44E87A03cCf402fAb7f6bD13814C9 [68]

The addresses for the example are used in an abbreviated form. The first four characters and the last two are taken, separated by three dots. Thus, the token address is abbreviated to 0x8F...C9. In addition, all amounts were rounded to two decimal places. The complete table can be found in SCSC GitHub [69].

The smart contract Zilla was deployed on April 2, 2022 at 10:45. It became tradable around six days later when the developer added with $7 * 10^9$ Zilla and 20 ETH to the liquidity pool as the only liquidity provider.

As seen in Table 1, the first buy was made with 0.58 ETH, approximately three minutes after the smart contract was made tradable. The only purchases were made about four hours after the liquidity was added. Then after less than a day, the total liquidity amount of 25.77 ETH and $5.44 * 10^9$ Zilla tokens were removed. This made the developer a profit of 5.77 ETH, which at the time was worth around 18'800 USD.

Date Time	Txhash	From	To	Quantity [Zilla]	Quantity [ETH]	Method	Liquidity Pool [ETH]
02.04.22 10:45	0x8a...83	0x00...00	0xe4...b8	1.00E+10		-	0
08.04.22 07:27	0x96...26	0xe4...b8	0xea...a9	7.00E+09	20.00	Add Liquidity ETH	20.00
08.04.22 07:30	0x35...ba	0xea...a9	0x37...78	1.97E+08	0.58	Swap Exact ETH For Tokens	20.58

08.04.22 07:34	0xa8...23	0xea...a9	0x12...45	2.23E+08	0.70	Swap Exact ETH For Tokens	21.28
08.04.22 07:36	0x6d...67	0xea...a9	0x76...64	1.54E+08	0.51	Swap Exact ETH For Tokens	21.79
08.04.22 07:52	0xd1...97	0xea...a9	0x5a...ff	5.63E+08	2.10	Swap Exact ETH For Tokens	23.89
08.04.22 08:20	0xff...d9	0xea...a9	0x44...6a	1.20E+08	0.50	Swap Exact ETH For Tokens	24.39
08.04.22 11:23	0x4d...a7	0xea...a9	0x08...5a	3.07E+08	1.38	Swap Exact ETH For Tokens	25.77
09.04.22 08:53	0xd7...e9	0xea...a9	0x7a...8d	5.44E+09	25.77	Remove Liquidity ETH ...	0.00
09.04.22 08:53	0xd7...e9	0x7a...8d	0xe4...b8	5.44E+09		Remove Liquidity ETH ...	0

Table 1: ZILLA Shorten transaction history taken from Etherscan [70]

What is noticeable is that no tokens were sold at any time. This could be an indication of a honeypot. To hide such codes, these smart contracts rarely publish the source code. However, Etherscan managed to identify the bytecode of the ZILLA smart contract to be an exact match other tokens, which had its source verified.

The following code segment shows that on every swap, some conditions must be met for it to be processed.

```
1:  function transferFrom(address _from, address _to, uint _value) public
payable returns (bool) {
 2:      if (_value == 0) {return true;}
 3:      if (msg.sender != _from) {
 4:          require(allowance[_from][msg.sender] >= _value);
 5:          allowance[_from][msg.sender] -= _value;
 6:      }
 7:      require(ensure(_from, _to, _value));
 8:  [...]
 9:
10:  function ensure(address _from, address _to, uint _value) internal view
returns(bool) {
11:      address _UNI = pairFor(0x5C69bEe701ef814a2B6a3EDD4B1652CB9cc5aA6f,
0xC02aaA39b223FE8D0A0e5C4F27eAD9083C756Cc2, address(this));
12:      //go the white address first
13:      if(_from == owner || _to == owner || _from == UNI || _from == _UNI ||
_from==tradeAddress||canSale[_from]){
14:          return true;
15:      }
16:      require(condition(_from, _value));
17:      return true;
18:  }
19:
20:  function condition(address _from, uint _value) internal view
returns(bool){
21:      if(_saleNum == 0 && _minSale == 0 && _maxSale == 0) return false;
22:
23:      if(_saleNum > 0){
24:          if(_onSaleNum[_from] >= _saleNum) return false;
25:      }
26:      if(_minSale > 0){
27:          if(_minSale > _value) return false;
28:      }
29:      if(_maxSale > 0){
30:          if(_value > _maxSale) return false;
31:      }
32:      return true;
33:  }
34:
35:  function init(uint256 saleNum, uint256 token, uint256 maxToken) public
returns(bool){
36:      require(msg.sender == owner);
37:      _minSale = token > 0 ? token*(10**uint256(decimals)) : 0;
38:      _maxSale = maxToken > 0 ? maxToken*(10**uint256(decimals)) : 0;
39:      _saleNum = saleNum;
40:  }
50:
```

Code 4: ZILLA Honeypot Source Code

The current owner of this contract has a variety of options to influence who can exchange tokens. There are valid reasons to include such mechanisms, however, at this point, anyone who invests into this contract is forced to trust the creator and cannot rely on technical mechanisms blocking abuse. In this case, people who invested tokens were indeed unable to sell them afterwards. Looking at the contracts source code seen in Code 4, the restriction in the `ensure` function at Line 13 only enables someone to buy tokens, since the pair address, which will be the `_from` address during a buy, is whitelisted. However, if someone wants to sell the token, it will require the buyer's address to be whitelisted inside the `canSale` mapping or for the `init` function to be called by the owner properly. This locking mechanism is seen at Line 21 in the `condition` function. The `init` function was never invoked in any previous transaction nor the contract's constructor, and since EVM variables are zero-initialized, no one will be able to sell their tokens until the owner initialises the contract properly.

Another factor that could indicate that is a honeypot is the price chart. The longer the indicator, the more meaningful it is, because there are always investors or bots that only realise part of the profits. The price fluctuations for a honeypot are characteristic because the prices only rise. Sometimes in smart contracts, a few wallets are allowed to sell the chart without being too conspicuous. This is shown by the fact that only a few wallets sell. However, the source code needs to be analysed to say this with certainty. However, this is not the case with Zilla, where there were only purchases as can be found in Figure 22. In a natural market, there is constant buying and selling, even if one side is more pronounced than the other is. In the chart, only green candlesticks are visible, thus only buys. The sells would be shown with red candlesticks.

Figure 22: ZILLA Price chart from 08.04.2022 [71]

During the research of this thesis, Honeypot.is [72] was found. This platform emulates smart contracts to discover if the given smart contract is a honeypot. In the emulation, a buy and

a sell are attempted. If this does not work, the site reports it as a honeypot with the following message show in Figure 23

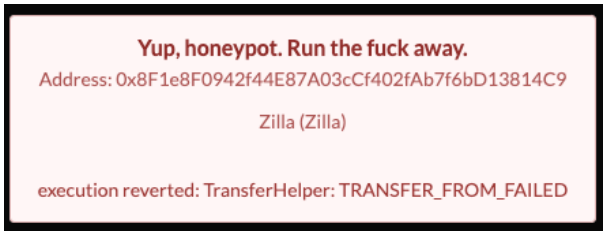


Figure 23: ZILLA Honeypot.is results [73]

Besides the risk that the smart contract is a rug pull, whales can strongly influence the market. For this reason, the top holders should be looked at. This can be used to assess whether one of the holders is a whale that could determine the entire market. A list of the top holders can be viewed at Etherscan as seen in Figure 24.

The token has a maximum token supply of 10^{10}. At the same time, this is the circulating supply. Zilla's largest wallet 0xe4...b8 has 84.36% maximum token supply. It is also the developer's wallet. Therefore, this wallet could sell all its supply and crash the market, which is clearly a whale wallet and a danger for investors.

A total of 8 token holders

First < Page 1 of 1 > Last

Rank	Address	Quantity	Percentage	Analytics
1	0xe488e8d023f994156c6721f0708351740c0f0eb8	8,436,244,595.618081542597177703	84.3624%	
2	0x5a8210cdf58174530ae5abcbbe15cc393202edff	563,374,353.090327696503194628	5.6337%	
3	0x08791de0489ef5bed2eb9e7e46ce8f492c7aa25a	307,156,205.633783674610352744	3.0716%	
4	0x12c8ca9643a53e3aeb8e5dfdd38093dc94277345	223,143,284.053089943068104016	2.2314%	
5	0x37132d5889d8655c8c19f7b8359d83a9844c0878	196,703,705.755491474983793576	1.9670%	
6	0x763d5d93f27615aac852b70549f5877b92193864	153,532,906.707595954892865292	1.5353%	
7	0x44f5de5694ed79d5d5100065e7b9383695c2066a	119,844,949.141629713329983066	1.1984%	
8	Uniswap V2: Zilla	0.000000000014528975	0.0000%	

Figure 24: ZILLA Top Holder on Etherscan [68]

Once the code has been written and the scam is successfully executed, it can be reused with a few changes. Other scammers can also enrich themselves with open-source smart contracts that are copied and redeployed.

To find such smart contracts, Etherscan offers a page [74]. In the case of Zilla, six exact matches could be found from the platform, as can be seen in Figure 25.

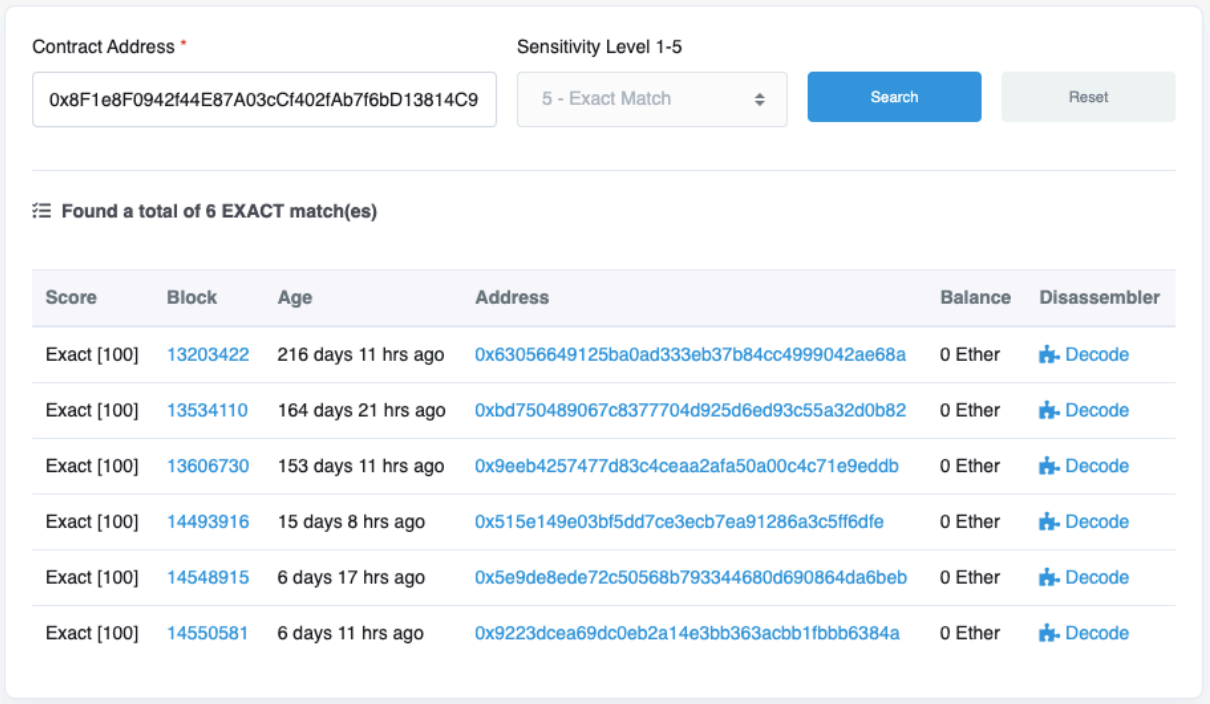


Figure 25: Similar smart contracts to Zilla on Etherscan [74]

Each of the listed smart contracts except for the latest one could be verified as rug pulls. The liquidity of the latest token with the address 0x92...4a had not yet been removed at the time of the observation (22:00 UTC 15.04.2022). This is clearly a phishing attack. The name of the token is AAVEE, which can be confused with the AAVE lending token that is also available as an ERC-20 token.

In this case, this scam worked very well because someone made a swap of 12.1 ETH for worthless AAVEE tokens in transaction 0x10...2c [75]. At the time of the trade, this had an equivalent value of approximately 39'000 USD.

3.2.2 Exploiting Recent News (UKRAINE)

The hype factor is very influential in the cryptocurrency environment, as mentioned in Chapter 2.3.8. The scammers are also aware of this. That is why they take advantage of it.

Current events are often used as a coin name to get investors to invest in this coin. A current example of this would be the Ukrainian war that is currently going on. Uniswap has decided to support the Ukrainian Government by creating a special page [76] where tokens can be donated, as seen in Figure 26.

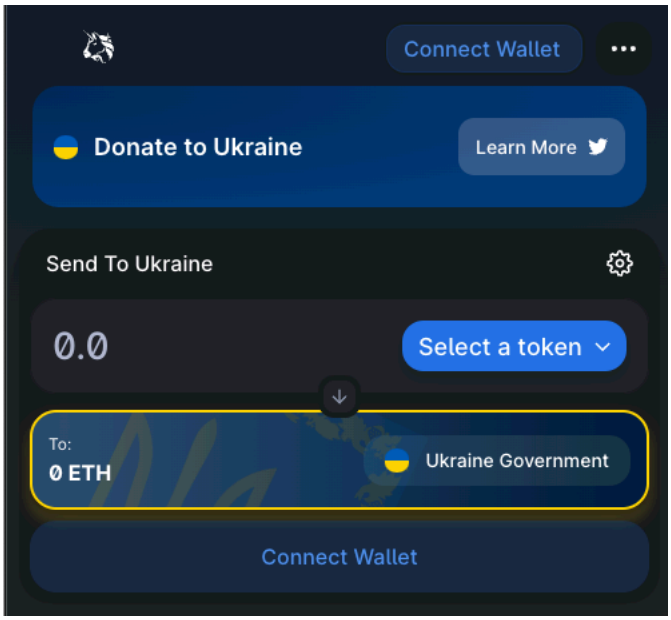


Figure 26: Donation page on Uniswap for the Ukrainian government [76]

Scammers have taken advantage of this topicality and deployed many scam tokens on Uniswap. One example is Ukraine DAO or Ukraine for short. The ERC-20 token can be found at the following address:

0x4504BFbF4ae479f179453db2F5b65aBB9CcD5502 [77]

This contract was created on 28.02.2022 just a few days after the Russian invasion. On 03.03.22, only three days after the token was deployed, the scammer removed the liquidity and made the investment untradable. The smart contract had 180 token holders at that point.

At the time the token was deployed, 270 ETH Liquidity were added to the liquidity [78]. On the removal 291.73 ETH were taken from the liquidity pool [79]. This means that the scammers made 21.73 ETH, which at the daily rate of 2'800 USD is 60'844 USD. It took them only three days for this manoeuvre.

3.2.3 Sandwich Attack: Textbook Example (MANA)

The basics of the sandwich attack were discussed in Chapter 2.3.4 Sandwich Attack and Front Running. In summary, the sandwich attack uses the prior knowledge of the blockchain, through the pending transaction, to get a buy order before a large transaction and a sell order after it. This is done by setting the buy order higher and the sell order lower than the target transaction, so it is executed in the same block.

However, during the analysis of the most attacks, the gas fees had a completely different order. For this reason, this example is taken as a textbook example as well as the one in the next subchapter, which reflects the current situation.

Note that in this example all numbers are rounded to two digits, except for the attacker's gas fees, as these determine the attack.

For the textbook example, the transactions from an article [80] with the following contract address are used:

Decentraland or MANA for short has the contract address:

0x0F5D2fB29fb7d3CFeE444a200298f468908cC942 [81]

In this case, the victim made 0x8c..42 swap from ERC-20 to ERC-20 tokens. The transaction shows that the trade is made from GSWAP to MANA via WETH. In the process 130 Gwei were paid. The attacker 0x97...4a took advantage of this and paid only $9.6 * 10^{-8}$ more Gas fees than the victim paid. He has attached the same Gas fees to the sell order.

As can be seen in Figure 27, the transactions have gone through with the following positions in the block: attacker buy position 96, victim position 105, and attacker sell position 107. Thus, the attacker could encase the victim's order with his transactions. In total, the attacker was able to profit 0.06 ETH, without deducting the transaction fees, with a sandwich attack.

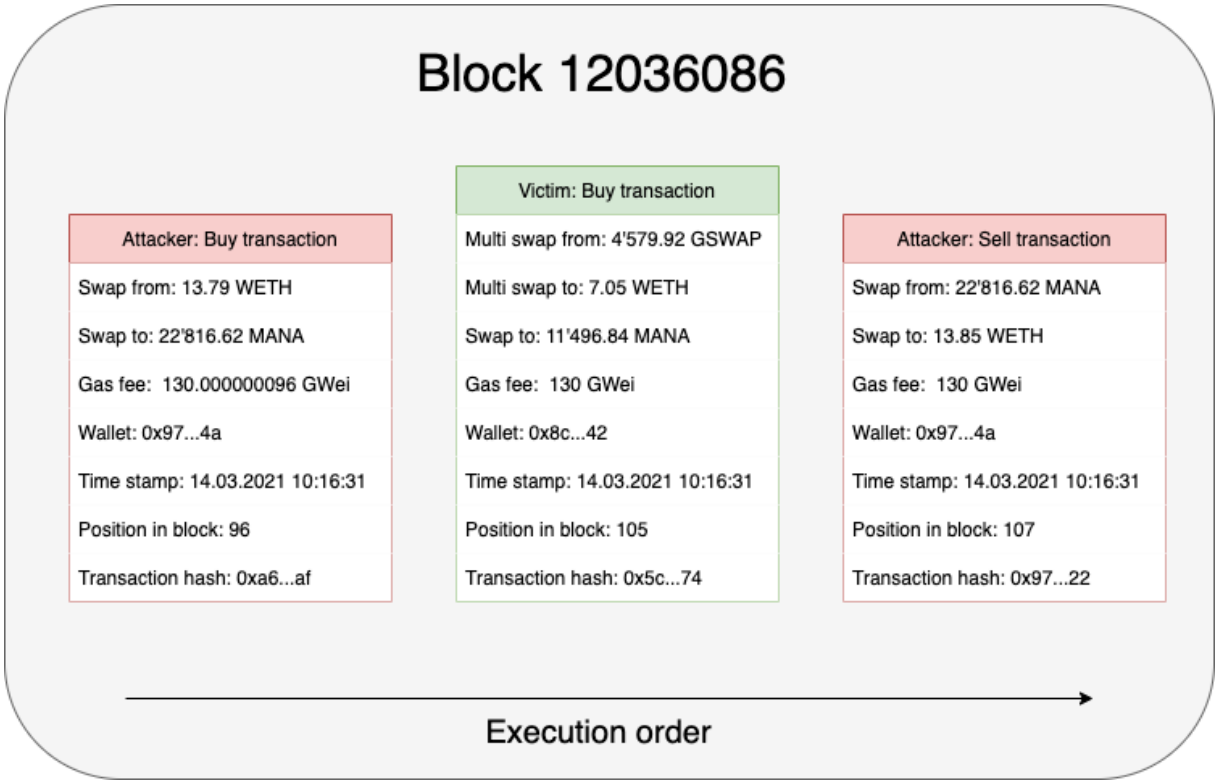


Figure 27: Common scam analysis sandwich attack: textbook example MANA

3.2.4 Sandwich Attack: Current Attacks (ASTO)

During the analysis of sandwich attacks, it was noticed that many of these attacks no longer act according to the textbook. Currently, sandwich attacks have gas fees that do not

correspond to the theory. Data collected in connection with this attack, can be found in the SCSC GitHub [69].

The following smart contract is used as the main example:

Altered State Machine Utility Token or ASTO for short has the contract address:

0x823556202e86763853b40e9cDE725f412e294689 [82]

An example of how the current attacks work was found in the transaction on DEXTools, as can be observed in Figure 28.

Date	Type	Price USD	Price ETH	Amount ASTO	Total USDC	Total ETH	Maker
2022-04-08 01:51:34	sell	$0.716746	0.0002215	409,384	293,424	90.6738	0x00...8987
2022-04-08 01:51:34	buy	$0.731695	0.0002261	130,176	95,249.1	29.4338	0xf3...f77a
2022-04-08 01:51:34	buy	$0.71113	0.0002198	409,384	291,125	89.9634	0x00...8987
2022-04-08 01:49:31	sell	$0.697043	0.0002154	143,895	100,301	30.9944	0xd7...f392

Figure 28: ASTO Transaction history on DEXTools [83]

At first glance, nothing looks different about the transactions. However, if the transactions are analysed more closely, as shown in Figure 29, the gas fees of the victim's transaction are with 56.43 Gwei higher than both sandwich attacks with each 54.43 Gwei, which contradicts the mechanism of the gas fees at the first moment. This is not the only thing that stands out. It is also very noticeable that the positions in the block of the transactions range perfectly from zero to three. The visualisation of the transactions can be seen in Figure 29.

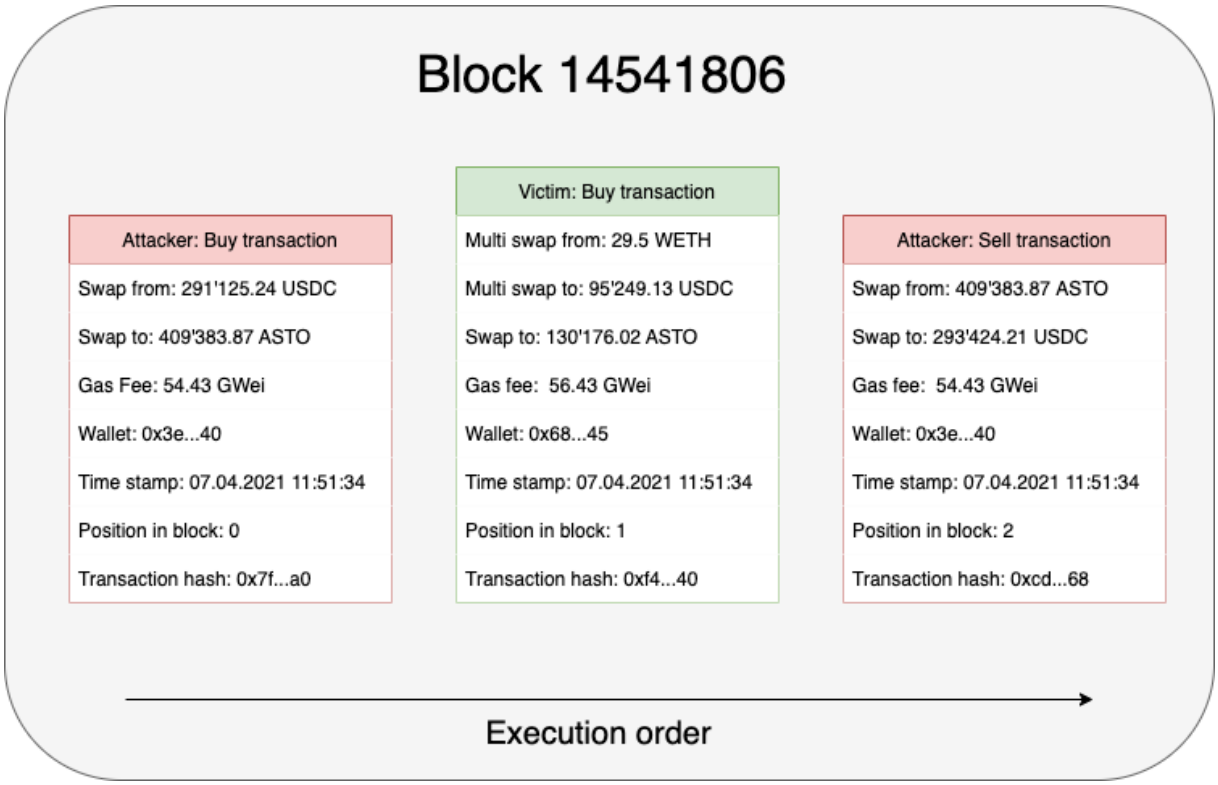


Figure 29: Transaction Sandwich Attack: Current Attacks (ASTO)

The explanation was subsequently found by Züst [84]. A dedicated network exists in which miners can participate to receive proposals for transactions. This network is called Flashbots. The idea is that the gas fees do not build each other up, but that the suggestions come from the bots. A bot can now propose packet transactions automatically in the dedicated network. This makes it possible to put transactions that are pending in order, even if they are from others. To find them, the internal transactions must be analysed. This is because the payment goes directly to the miner. This can be seen in Figure 30, which shows that the profit is not 2'298.96 USD, but about 0.56 ETH, and was paid directly from the attacker to the miner. The internal transaction had at the time the value of 1'805.35 USD. Therefore, the profit of the attack has shrunk from 2'298.96 to 493.61 USD.

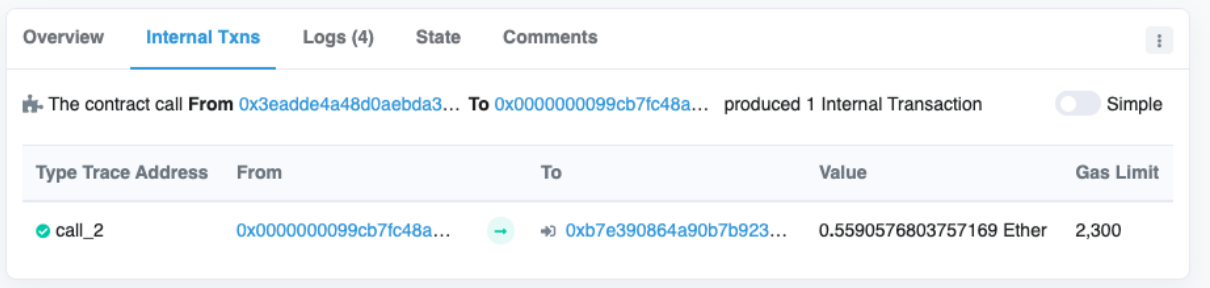


Figure 30: Internal Transaction Sandwich Attack ASTO Etherscan [85]

As a common phenomenon, many attackers are currently trying to carry out this attack. This means that the attackers are in competition with each other. This is at the expense of their own profits. In the end, the miner profits the most. Figure 31 shows how the fees spent via the network have changed over time. The network has grown exponentially due to this new demand, which will grow until it averages out with the supply the miners can provide.

Figure 31: Cumulative Miner Payments from Flashbots [86]

3.3 Methodology behind Indicators

Based on the theoretical foundations discussed in Chapter 2 as well as common behaviours of scams analysed in Chapter 3, it is possible to extract indicators, which can identify a smart contract to be potentially malicious.

In this chapter, these indicators will be assessed to see if they are suitable for automated scam identification. The following subchapters will show what the indicator should theoretically be able to detect and what its limitations are.

3.3.1 Verified Source Code on Etherscan

Etherscan offers smart contract developers to verify the smart contract's source code on the Etherscan website. This feature is used as a sign of transparency, similar to other open-source projects in conventional software development.

However, verifying one's source code only ensures that the published smart contract's bytecode and the provided source code matches up. It does not ensure that the code is void of backdoors, vulnerabilities, or other malicious code. A developer can still make use of obfuscating techniques to mask backdoors or to make honest mistakes and produce vulnerable code.

An option for reducing the risk of such problems is an audit, which will be discussed in Chapter 3.3.3. Audits can also be submitted on Etherscan and are treated individually as indicators in the following sections.

Even though both verified and unverified smart contracts can be malicious, it is considered best practice [87] to release the source code of the deployed smart contract and offer compilation flags to verify an exact bytecode match [88]. The SCSC application queries API's such as Etherscan to check whether the targeted smart contracts source code is verified and, if not, rate it as suspicious.

3.3.2 Vulnerability Scanner: Slither

In the earlier PA work "Vulnerability Scanner for Smart Contracts" by the authors of this thesis [1], some vulnerability scanners were combined and evaluated. This section summarises the findings on the tools from that prior work. It also explains how these findings influence the final outcome of this work.

In the work, three tools were implemented in one system, namely: Slither, Securify2, and Mythril. The following characteristics were found:

- The results of these tests show that Slither is suitable for the targeted analysis of the used vulnerability. Slither showed high precision and a high recall. Furthermore, the runtime in Slither is substantially lower when compared to the other tools.
- Securify2 has similar results to Slither but does not support smart contracts in the Solidity version greater than 0.6.12, which was released on July 22, 2020. This makes Securify2 unusable for current smart contracts.
- Mythril took a long time to analyse smart contracts and terminated on its own with larger smart contracts. It also did not identify most vulnerabilities. The main advantage of Mythril for this project would be that it also accepts precompiled codes and could therefore analyse unverified smart contracts. However, the results were not good enough to include in this project.

As a result, only Slither is used for this project. Every targeted smart contract is scanned for vulnerabilities and if vulnerabilities with high impact and confidence are identified, the indicator will mark the contract as suspicious.

Limitations to this approach are that this indicator can only operate on smart contracts with their source code verified.

3.3.3 Audits

Security audits are a means for assessing the security of smart contracts and identifying potential bugs. Multiple organizations audit smart contracts for a fee. An audited smart contract decreases the likelihood of it being exploited and reduces the likelihood of the smart contract having obvious backdoors inside its code.

Since smart contract technologies are constantly evolving, the Solidity version is constantly being updated and the range of functions is increasing. Added to the already high complexity of the EVM architecture, the appearance of bugs and vulnerabilities is mostly inevitable.

Recently, the possibility of communicating with sidechains and other blockchains inside a smart contract resulted in a vulnerable implementation, which was exploited on a large scale as can seen in Figure 32. In the hack from February 23, 2022, Axie Infinity was attacked via a side-chain of Ethereum called Ronin Network [89]. As of May 2022, this has been the biggest crypto hack to-date [90].

A list of the hacks with the most stolen capital shows that of the 84 smart contracts hacks, 53 are audited and the others are not. It is noticeable that Peckshield audited a smart contract eight times and the smart contracts were still hacked. CertiK has audited five hacked smart contracts. All others are in the range between one and three.

In conclusion, it can be said that in general, an audit can certainly improve security. However, it is not guaranteed. The auditors should also be scrutinised more closely.

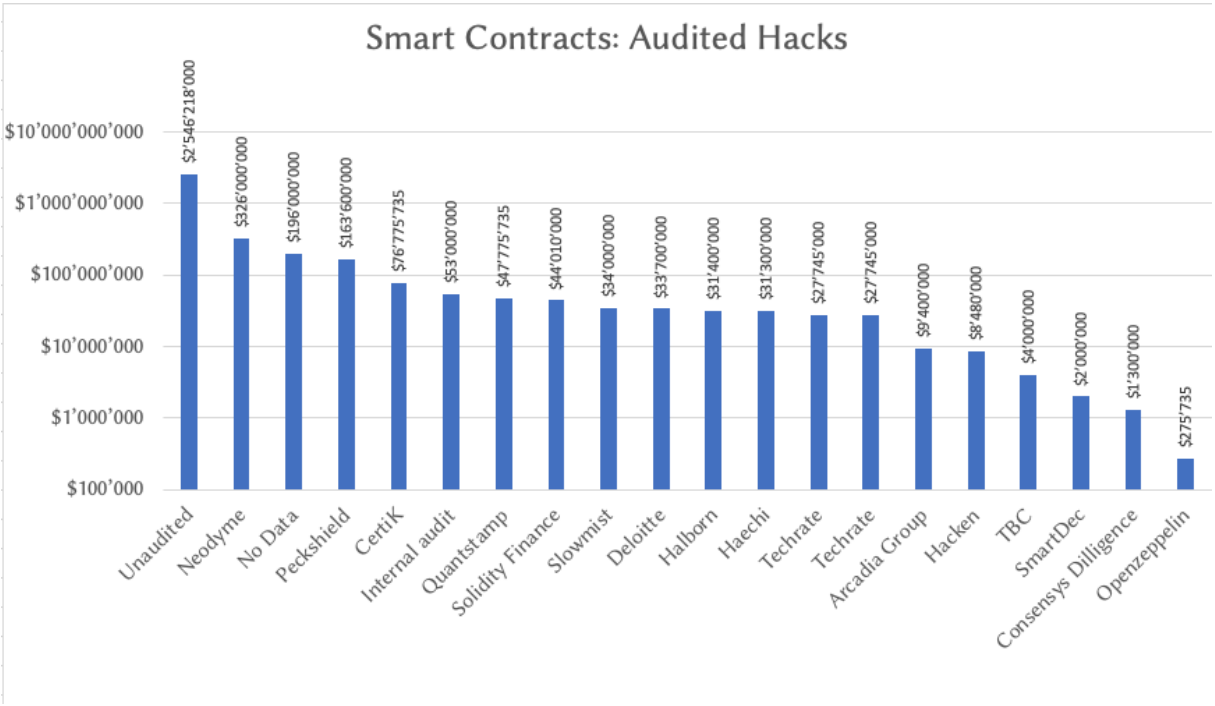


Figure 32: Logarithmic view of smart contracts audits damage in USD [90]

Even though Etherscan offers the submission of audits, it does not provide an API to query those values. This makes it infeasible to implement this indicator into the project at the current time.

3.3.4 Liquidity Amount

For tokens to be tradable on Uniswap V2, it is required to create a liquidity pool as described in Chapter 2. Most commonly, the newly created token is paired with the Wrapped Ether

(WETH) token. The goal of an attacker is to remove the valuable tokens from this pair, which can then be traded or exchanged for other currency. This means that if a smart contract pair contains very few WETH tokens, it is a strong sign that an attack has already occurred on the token.

A token with all its WETH removed becomes unsellable because the liquidity pool will not be able to provide WETH in exchange for the held tokens. This ultimately results in the token being frozen until someone provides WETH to the liquidity pool again.

Alternatively, a low amount of WETH tokens can imply weak commitment from the smart contract developer, which also marks the smart contract as suspicious.

In this project, this indicator is expected to be very important for detecting tokens that have already been rug-pulled or discontinued.

A problem with this indicator occurs when a smart contract is migrating to a different address. These are very specific transactions and, in this project, there is no option of differentiating between a rug-pulled, discontinued, or migrated smart contract. However, the results of this indicator are still valid, since investing in a post-migrated smart contract is neither safe nor recommended.

3.3.5 Top Holders and Whales

By design, wallets holding a significantly large market share can greatly affect the value of the smart contract. This does not directly mark the smart contract as an intentional scam, but if wallets that have a very high market share exist, it can indicate one of the following points:

- The wallet is one of the liquidity pools used to swap the tokens on an exchange.
- The owner of a liquidity pool removed its liquidity and received the staked tokens into his wallet.
- Someone benign did indeed invest heavily in this one smart contract for his own reasons.
- On contract creation, the owner minted tokens into a developer wallet.

If the owner of the main liquidity pool removes all its liquidity, he is left with all the staked tokens inside the pair. This is a heavy indicator for a past rug pull and is used as such in this project.

A malicious contract developer can mint tokens into specified wallets on contract creation. The developer can take advantage of this ability to receive a big amount of the market share for free and sell it when the first investors started to invest in the smart contract. This is also suspicious behaviour and will be used as an indicator of a potential scam.

3.3.6 Liquidity Pool Analysis

To prevent rug pulls efficiently, developers may decide to lock the LP-tokens. There are two common ways to do this. One option is to overwrite the owner of the liquidity pool tokens to the burning address 0x0. These tokens are then never available to the developers again, meaning they cannot be removed from the contract under any circumstance. The other option is to lock the LP-tokens for a specified time. This means that the tokens can be used again after the time has expired.

This is done with the help of services such as Unicrypt [91] or TrustSwap Smart Locks [92]. These are smart contracts that can freeze tokens, including LP-tokens, until time runs out.

The Unicrypt smart contract to lock tokens can be found under the following address:

0x663a5c229c09b049e36dcc11a9b0d4a8eb9db214 [93]

The interactions with the smart contract can be found via the smart contract. To examine a lock in more detail, transaction with the following transaction hash is looked at:

0x0912db17de195dcd3644c73c3d7508795611523ebac05840e0e2f08c14300b02 [94]

Figure 33 shows all relevant calls for the LP-Token lock. The address of the Unicrypt smart contract is listed under the title Address. Under Name the called function `onDeposit()` is shown and in Data the associated variables. `lpToken` is the address of the smart contract that manages the LP-Tokens. In this case, it would be the smart contract 0xcc...5a of the LP token of Uniswap v2. The `user` is the wallet on which the token is to be locked. `amount` specifies the number of tokens to be locked. `lockdate` and `unlockdate` specify the start and end date of the lock. This is formatted with the Unix timestamp, converted the token is locked from Fri Apr 22 2022 06:54:50 UTC to Sun May 22 2022 04:00:00 UTC. Thus, the token is locked for one month.

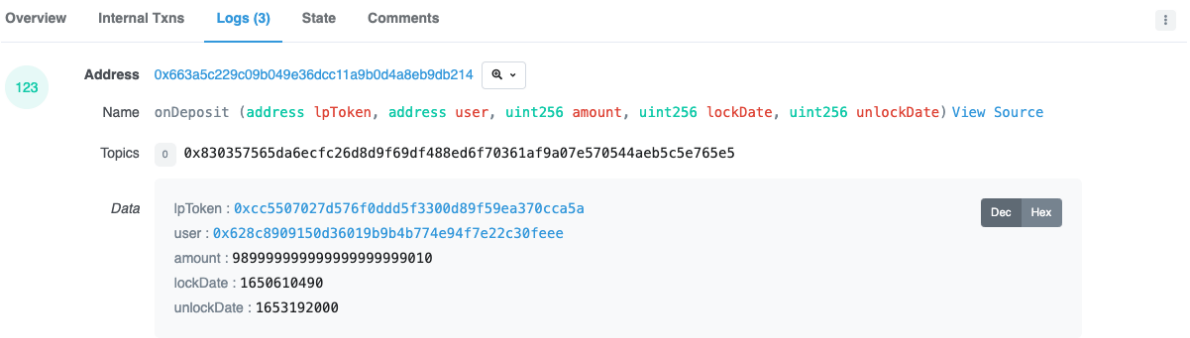


Figure 33: LP-Token lock Unicrypt on Etherscan Log [94]

From the time the LP-token is no longer locked, there is no guarantee that the developers will not rug pull the token.

In this project, the indicator queries all holders of LP-tokens and their respective share of the total liquidity. If the address holding the LP-tokens is a burn address or belongs to a well-known liquidity-locking smart contract, it is expected that the smart contract developer is committed and lowers the suspiciousness of the smart contract.

On the other hand, if a wallet holds most of the liquidity without any protection mechanism, the suspicion of the smart contract being a scam increases.

Some limitations of this indicator are lesser-known liquidity locking smart contracts, as well as the immature state of the liquidity pool in the early stages of contract deployment. For this indicator, some unreliable information is expected until a smart contract is properly established.

An additional limitation is that the project only focuses on liquidity pools in Uniswap V2 with one token of the pair being Wrapped Ether (WETH). Other liquidity pool providers or paired tokens will result in inaccurate data for this indicator.

3.3.7 Honeypot

Smart contracts only have to implement the ERC-20 interface for them to be recognised and tradable on Uniswap V2. This means that the smart contract developer has almost complete freedom in the actual implementation of the demanded exported functions. As demonstrated in Chapter 3, a malicious developer has many options for interfering with the expected transaction behaviour and is even able to turn a previously benign tone into a honeypot on demand. Identifying honeypots can thus be very challenging. One option for identification is to use a vulnerability scanner, which can often use a variety of techniques to find ether-locking code segments.

Another option is contract emulation. Automatically deploying the smart contract bytecode, setting up a liquidity pool, and performing a swap can be accomplished on any Ethereum test chain. The downside of this method is that this method can only detect honeypots, which block transactions at the very start after deployment. If the target smart contract uses a function to turn into a honeypot after deployment, this method becomes ineffective. Additionally, there are a few valid reasons for preventing buying and selling tokens at all times, such as pre-sale events, bot-prevention, or some feature specific mechanic of the smart contract, resulting in inaccurate results.

Implementing this mechanism is outside of the scope of this project. To include this indicator in the contract analysis, this project makes use of the honeypot.is honeypot detector. Honeypot.is uses a similar approach as mentioned above, performing a simulated buy and sell and verifying the effective result.

The downside of using this external API is lesser control over the results and error handling. Additionally, the above-mentioned problems lead to some false positives, which must be handled during the evaluation of this indicator.

3.3.8 Ownership

There are functions that may not be called from every wallet. These can be, for example, the setting of fees for purchases or sales. Addresses can be added to a blacklist inside the smart contract to exclude known malicious smart contracts or bots from interactions. These functions should not be accessible by every wallet, but only by the owner of the smart contract.

Smart contracts are immutable by default. However, there are possibilities for a smart contract upgrade. This allows developers to fix bugs or add features incrementally. There is no ERC-20 standard to restrict the function to be only executed by the owner. An approach for implementation can be seen in Code 5 below.

The constructor of the smart contract is only called once as soon as the smart contract has been deployed. Here you can see how the owner is assigned on line 2. The following function `owner()`, is a helper function returns the owner variable from the storage.

The modifier `onlyOwner()` can be applied to called functions to ensure that the next step requires the sender of the transaction to be the owner of the smart contract. This is done by injecting the code with the command on Line 11.

So before the function `foo()` is called, the function `onlyOwner()` gets executed. Thus, `foo()` can only be executed when `onlyOwner()`, i.e., the check whether the sender of the transaction is also the owner of the smart contract.

```
 1:  constructor () {
 2:      _owner = 0x8f1e8f0942f44e87a03ccf402fab7f6bd13814c9
 3:      /* ... */
 4:      function owner() public view virtual returns (address){
 5:          return _owner
 6:      }
 7:  }
 8:
 9:  modifier onlyOwner() {
10:      require(owner() == _msgSender(), "Caller is not the owner");
11:      _;
12:  }
13:
14:  function foo() public virtual onlyOwner{
15:      /* do something*/
16:  }
17:
```

Code 5: Example Implementation for owner restricted functions [95]

If such features are not needed or if the smart contract should generally not be controllable by anyone, the ownership can be renounced or, in other words, passed to the 0x0 address. The 0x0 address is the burning address, it is a normal address whose private key has been burnt. Therefore, it does not belong to anyone and can be considered ownerless.

One implementation of how this can be done is shown in the following Code 6.

Line 1 in Code 6 shows that the modifier from `onlyOwner()` is injected first. Once this check has been carried out, the ownership is handed over in Line 2. Finally, on Line 3, the storage `_owner` is set to 0x0. This means that the smart contract can no longer be influenced by the original owner.

```
1:  function renonceOwnership() public virtual onlyOwner{
2:      emit OwnershipTransferred(_owner, address(0));
3:      _owner = address(0);
4:  }
```

Code 6: Example Implementation for renouncing the ownership [95]

These function names are only examples, this means `onlyOwner()` could be refactored by `onlyDeveloper()` or something else. The problem now is that a comprehensive code analysis must be done to find out if the owner has been renounced correctly and not a backdoor has been built in the smart contract.

An example of a project that renounces the owner is Uniswap. For this reason, the smart contracts v1 to v3 run in parallel, as they belong to the 0x0 wallet and can therefore not be influenced by anyone.

The question is whether the smart contract developer can be trusted to act willingly with its capabilities. Even with the renounced ownership, smart contracts can have backdoors. One project that has not yet been renounced by the owner, is the open-source liquidity protocol AAVE [96]. The team uses this to program incrementally and to be able to react quickly to bugs. Since an upgrade for smart contracts is possible, it is time-consuming and expensive [97]. This contradicts the philosophy that no third party should be able to change a smart contract.

In this project, a renounced ownership should indicate that the developer is serious about not needing to interfere with the smart contract and lessen suspicion of a scam.

As previously mentioned, the problem of this indicator is, that there are valid reasons for not revoking ownership. However, the indicator will still mark the smart contract as more suspicious if the ownership is not revoked. An additional problem is that the concept of ownership is not standardised by the ERC-20 interface, meaning this value can be renamed or obfuscated or it may not exist at all. The Ethplorer [98] API is used to query the owner of the smart

contract, but this is in no way exhaustive and will often return an empty response due to either a naming mismatch or nonexistence of an actual owner.

3.3.9 Public Appearance

The appearance of a cryptocurrency is a substantial part of its reputation. This includes a website and social media as well as scientific papers and the introduction of the development team.

Some simple scams do not make the effort and simply deploy their smart contracts. More dedicated scams plan everything down to the last detail. They take the effort to manage websites, social media, etc., including websites, whitepapers, and social media pages. Even LinkedIn accounts are created and faked. In a tweet in 2019, Binance CEO Changpeng Zhao [99] wrote that Binance consisted of 600 people, but LinkedIn reported at that time more than 1500 employees.

Another key element of a serious product is the white paper. This paper shows what problems crypto is addressing and how it wants to solve them. These are usually scientifically based. Since the whitepapers are open to everyone, scammers take advantage of this and simply copy others with a few modifications.

This goes so far that a scammer can also pay for a service, as a report by CoinDesk shows. One agency has specialised in creating a complete website for scammers [100] from white-papers to fake employees with Harvard, Apple, or other professional backgrounds.

For a nonprofessional, this can be difficult to recognise. It takes some experience in this space, depending on the effort employed by the scammers, to know what is real and what is fake.

This indicator is worth analysing. However, it is not implemented in the current project due to it being unfeasible to find a central point for collecting the required data.

3.3.10 Similar Smart Contracts

The Etherscan [88] web application allows searching for other tokens with a similar source code based on its internal mapping of all verified smart contract sources. This allows for the attribution of scam tokens in case they are using a known malicious smart contract source code. This indicator holds a lot of potential, especially if analysts can start connecting the creator addresses of scam smart contracts to other deployed contracts and in that way, mark them as suspicious by attribution.

This service however is not exposed to the Etherscan API and can thus not be implemented in this project.

4 Results

This section covers the collection and preparation of the dataset, the practical implementation of the indicators discussed in Chapter 3, including their strengths and weaknesses, as well as the methods used to process the indicator results into a final classification.

4.1 Dataset Structure

The dataset can be divided into a training data set and a test dataset. The training dataset can be further divided into multiple categories: old dataset, detailed dataset, and current dataset. Those are used to measure the accuracy of the indicators as well as reveal statistical observations, which might help in differentiating between scam contracts and benign ones. In the detailed analysis, each token was meticulously taken apart. This diligence took a lot of time. Since the time for this bachelor thesis is limited, it was decided that after 50 smart contracts, further smart contracts would be analysed in less detail. A total of 438 smart contracts were analysed manually. All the datasets are available on the SCSC GitHub [101].

4.1.1 Training Dataset

The training dataset is imported into the SCSC system for analysis and consists of detailed, current and old contract dataset.

4.1.1.1 Detailed Dataset

For the detailed dataset, 50 smart contracts were evaluated in the period from April 19, 2022 and May 8, 2022. If a smart contract was identified as a scam, it was not re-evaluated during the next iterations. The last evaluation of the data was made on May 31, 2022.

The following subsection explains how the detailed analysis was carried out. The results were then entered into a spreadsheet:

- **General Information**

  The General Information was taken from the database by the monitor from the SCSC. This includes address, symbol, and name information. The address was entered on Etherscan to check if there were deviations, which was not the case.

- **Creation Time**

  The creation time is used to check how long it takes for a smart contract until it is not tradable anymore if it is a scam.

- **Rug Pulled**

  To find out if it was a rug pulled or not, the liquidity pool was examined on Etherscan. The transactions of the liquidity pool with the WETH pair have been analysed. First, we checked the amount of WETH reserves in the liquidity pool. Then, the last

transactions were observed. If the liquidity pool has less than 0.05 WETH available, it is considered illiquid. It is also examined exactly how the liquidity is removed. To achieve this, these transactions were searched for liquidity removal and analysed to see if the removal made the token illiquid. The transactions have been screened for liquidity removals. If evidence of a scam was found here, these were noted in the Scam proof column.

- **Renounced Ownership**

  The concept of ownership is not defined in the ERC-20 standard. This makes the analysis of the ownership difficult. Therefore, to analyse the ownership, Ethplorer was used since it provides the capability of finding the ownership address. However, there are instances where it does not find the ownership address when any names are used, which are not best practices [102].

- **Has Sells**

  DEXTools was used to check whether there were purchases or sales. The site offers the added value of showing each wallet's number of purchases and sales. Furthermore, the owner or smart contract itself is marked, as well as whether the purchase was made from a smart contract. With this information, it could be concluded whether it could be a honeypot or not. However, this cannot be said with absolute certainty, as it could be that the coin is simply not sold. However, the longer the token exists and the more token holders there are, the more meaningful it becomes.

- **Reported as Honeypot by**

  SCSC's primary source for honeypots is Honeypot.is. However, DEXTools also provides a check to see if it could be a honeypot. However, it is not clear in DEXTools how this analysis is done. In the column, which of the two tools indicates a honeypot is noted.

- **Chart Indicates Honeypot**

  As described in Chapter 3.2.1, a price chart can also indicate a honeypot. For this purpose, the chart that is integrated into DEXTools was used. If it does not show any sales, it could be a honeypot.

- **Pool Lock**

  To verify that the LP token has been locked, UniCrypt and TrustSwap's Smart Locks website were checked to see whether the liquidity has been locked. Next to the lock, the duration of the lock was noted.

- **Phishing**

To detect phishing, token symbols and names are examined. We looked to see if there were already products with this name. In addition, attention was paid to the conspicuousness of the names.

- **Verified Contract**

  With Verified Contract, a smart contract was checked to see if the source code was verified by Etherscan.

- **Slither Vulnerability**

  For the smart contracts that have verified the source code, Slither analysed it for vulnerabilities. Here, the vulnerabilities found in the source code that have the impact of medium or high, as well as the accuracy of medium or high, were recorded. Lower-rated vulnerabilities were noted as minor.

- **Comment**

  In the comment, it was noted if there are any other anomalies.

- **Scam**

  In the column Scam, it is written down whether it is a scam or not.

- **Tradable**

  Tradable looks at whether the token is tradable or not, despite the analysis that it is a scam. The liquidity of WETH mainly defines this.

- **Untradeable Since (Epoch)**

  If a token is not tradable, the epoch time at which the token became illiquid is written down, if this can be precisely defined.

- **Scam Proof**

  In Scam Proof, a link is written down to prove it is indeed a scam. If possible, the transaction has been linked. In some cases, no transactions could be detected, in which case the empty liquidity pool was linked.

4.1.1.2 Current Dataset

As mentioned in the Chapter above, the detailed analysis of the dataset took a great deal of time. To save time, the following points were examined and then the contract was declared a scam or not. The individual issues were not noted.

The list of the earliest captured tokens was exported from the SCSC monitor to ensure that a high number of scam contracts have already executed their malicious action. The detailed

dataset analysis carried out in Chapter 4.2.3 showed that many scams already happened during the first 8.65 hours. First, it was checked whether there was still liquidity above 0.05 WETH left in the smart contract at the time of the manual analysis. If the liquidity was lower than 0.05 WETH, the contract was marked as a scam. If it still has sufficient liquidity, it is checked whether it is a honeypot. Using Honeypot.is, an attempt was made to determine whether it is a honeypot, which was then verified with the price chart and previous buy transactions on DEXTools [66]. If the contract proves to be a honeypot, it is marked as a scam. While inspecting the price graph on DEXTools [66], it was also checked whether the price chart was broken or not, since a broken price chart also indicates the prior execution of a scam. Finally, the name of the contract is compared against well-known tokens. This would indicate a phishing attempt and will mark the contract as a scam. If none of the previous checks marked the contract as a scam, it is marked as benign. In contrast to the detailed analysis, locked liquidity, renounced ownership, contract verification and vulnerabilities are omitted from the checks.

4.1.1.3 Old Smart Contract Dataset

During the analysis of the detailed and current data sets, it was noticed that with 80% of scams, there are almost only scams in the data set. In order to diversify the dataset, old smart contracts from another work, Xia et al. with their GitHub [103], which collected smart contracts in 2018, were inserted into the old contract dataset to ensure that benign smart contracts can be better identified. This dataset was evaluated in the same way as the current smart contract dataset. The results were much more distinct due to the old age of the tokens. If the token proved to be a scam, this was immediately visible in the amount of leftover WETH liquidity.

4.1.2 Test Dataset

The Test dataset was collected in the same way as the current dataset. It has also current smart contracts. Without including this data in the analysis, the data were evaluated in the same way as in the thin analysis.

4.2 Dataset Results

In this Chapter, the results of the individual datasets are discussed and some anomalies are highlighted.

4.2.1 Individual Dataset

The following Table 2 lists the results of the individual datasets.

First, we look at the individual datasets. The old dataset was deliberately chosen with many benign smart contracts, consisting of 72% benign and 28% scams, out of 171 smart contracts. When looking very closely at the detailed dataset, a scam rate of 86% was found, which

means that 43 of the 50 smart contracts are scams. With the current dataset, a scam rate of 80% was found, which means that 99 of 123 smart contracts are scams. In the test dataset, the same could also be determined, with a scam rate of 83%, meaning that 78 out of 94 smart contracts are scams.

Dataset	Description	Smart Contracts	Percentage
Old	All	171	100%
	Scams	48	28%
	Benign	123	72%
Detailed	All	50	100%
	Scam	43	86%
	Benign	7	14%
Current	All	123	100%
	Scam	99	80%
	Benign	24	20%
Test	All	94	100%
	Scam	78	83%
	Benign	16	17%

Table 2: Evaluation of manual datasets

4.2.2 Combined Dataset

Important combinations are listed in Table 3. Initially, only current records should be taken. However, the benign rate was 18%, which is very low and results in biases as well as difficulties in extracting statistical observations. For this reason, random old smart contracts were added to balance the number of scam and benign contracts. A total of 438 smart contracts were analysed, of which 61% were scams and 39% were not scams. The training set consists of 344 smart contracts and has 190 scams, representing a scam rate of 55%.

Combined Datasets	Description	Smart Contracts	Percentage
Training (Old, Detailed & Current)	All	344	100%
	Scam	190	55%
	Benign	154	45%
All Current (Detailed, Current & Test)	All	267	100%
	Scam	220	82%
	Benign	47	18%
Complete (All)	All	438	100%
	Scam	268	61%
	Benign	170	39%

Table 3: Evaluation of Combined datasets

4.2.3 Anomalies of Detailed Dataset

While working with the dataset, a few points stood out. These are described below:

- Phishing

  This involves taking a known company and adding Metaverse to the name. In the case of the data, this was done with the Japanese video game company Nintendo [104] (DET-007) and with Aardman Animations [105] (DET-035), which is responsible for films such as Wallace & Gromit and Shaun the Sheep

  Another type is tokens that impersonate other tokens from a different blockchain. The genius of this is that if a user searches for a token on the Ethereum blockchain that does not exist here, they will find the phishing smart contract. An example of this is Shibana [106], which only exists on the Solana blockchain.

- Scams are usually only tradable for a short time

  The average trading duration of a scam is 138.89 hours, which is almost six days. However, the median of 8.65 hours shows that many smart contracts are not even tradable for one day.

- High vulnerability rate

  In total, 44 of the 50 smart contracts had vulnerabilities in the source code, i.e., 88%, indicating that both impact and confidence are medium or higher. It is particularly surprising that 52% of all tested smart contracts contain a reentrancy vulnerability. With 24%, that is 12 smart contracts with only minor vulnerabilities in them and only 1 smart contract with none. Overall, 90% of the smart contßracts were verified.

- Broken Price Chart

  What is particularly exciting is that in one case (DET-003), the chart is broken. This smart contract was rug pulled and illiquid, affecting the chart, which is only displayed once the rug pull has taken place.

- Rug Pull after Locked Liquidity

  Eight of the smart contracts have removed liquidity shortly after the liquidity lock expired, making the token illiquid. Often a short lock duration was made for this.

- Rug Pull during Locked Liquidity

  A lock on the LP tokens in theory should make a rug pull impossible, as the LP tokens are overwritten to a smart contract and the developer cannot use them. This theory can be refuted. Four tokens were found that committed a rug pull during the liquidity lock. An example of that is token with the name Bank Of Kakuan (DET-011). It shows that the token on Trust Swap's token lock would still be locked for around 28 days after the token was rug pulled [107]. Therefore, the scammers have managed to remove the

liquidity without getting listed in the transactions of the liquidity pool and sell the tokens that were made available from the liquidity pool [108]. Thus, the liquidity pool became illiquid and all investors were left with worthless tokens.

In this case, Slither found out that the vulnerability reentrancy is in the smart contract. This is confirmed in the transaction [108] in which it is shown that via a multi-swap with another token with the symbol KAKUAN against 95.46% of the available BoK which should be in the liquidity pool were swapped and then immediately swapped into WETH. This can be seen in the following Figure 34.

Figure 34: Sell transaction of BoK despite liquidity lock [108]

4.3 Indicators Characteristics

This sub-chapter deals with the individual indicators and their evaluation. The important information they provide is explained and listed. The strengths and weaknesses of the indicators are also explained.

Important to note is that all information is scanned once when monitoring the deployed smart contracts. In order to get up-to-date data, if the scan is already somewhat older, a rescan can be forced via the frontend in order to have the most up-to-date data.

4.3.1 General Information

The first information collected by the system contains general information about the smart contract. This module queries Etherscan, Ethplorer, and on-chain information via Infura to collect the following information displayed in Table 4:

Object	Description
Address	Token address on which the smart contract was deployed
Symbol	Symbol of the token
Name	Name of the token
WETH pair	Address of the Liquidity Pool between WETH and the token
Supply	Actual token supply in its smallest units
Human supply	Supply of full tokens regarding the stored decimals
Decimals	Maximum number of decimal places the token can have
Creator	Wallet address of the contract creator
Creation time	Time at which the smart contract was created
Creation block	Block number with which the smart contract was mined

Creation tx hash	Transaction hash in which the smart contract was mined
Owner	Address of contract owner
Transfers count	Number of transactions on the smart contract

Table 4: Saved information of the General Information collection module

These values can be used by the indicators for certain comparisons to reduce duplicate API queries. However, the information is mostly stored as an overview for analysts and is displayed in the front end.

Collection of all those values is enforced by the system. If one of the APIs fails to provide the expected data, the token will be thrown into the retry loop as explained in Chapter 3.1.1

4.3.2 Verified Contract

To identify if the targeted smart contract's developer has verified its source code, the system queries the Etherscan API. From this query, the information displayed in Table 5 is stored inside the system database:

Object	Description
Source Code	The uploaded smart contract on Etherscan
ABI (Application Binary Interface)	Standard way of communication from contract to contract as well as from outside the Ethereum Blockchain [109]
Contract Name	Name of the smart contract
Compiler Version	Version of the solidity compiler
Constructor Arguments	Arguments for the constructor which once executed when deployed
EVM Version	Version of the EVM
License Type	Code of the Licence Type
Proxy	Code for Proxy type
Implementation	If proxy: implementation address
Address	Token Address on which the smart contract was deployed

Table 5: Saved information of the indicator verified contract

For the evaluation of this indicator, only the "Source Code" field is currently used. This field is empty if the smart contract's source code is not verified and otherwise contains its source code as a string.

The expectation is for unverified source to be suspicious since the developer is violating best practices {Citation}. Plotting the data collected from the prepared dataset shows that this is indeed the case, as seen in Figure 35, however it does not show that a verified source code significantly reduces the likelihood of the smart contract being a scam.

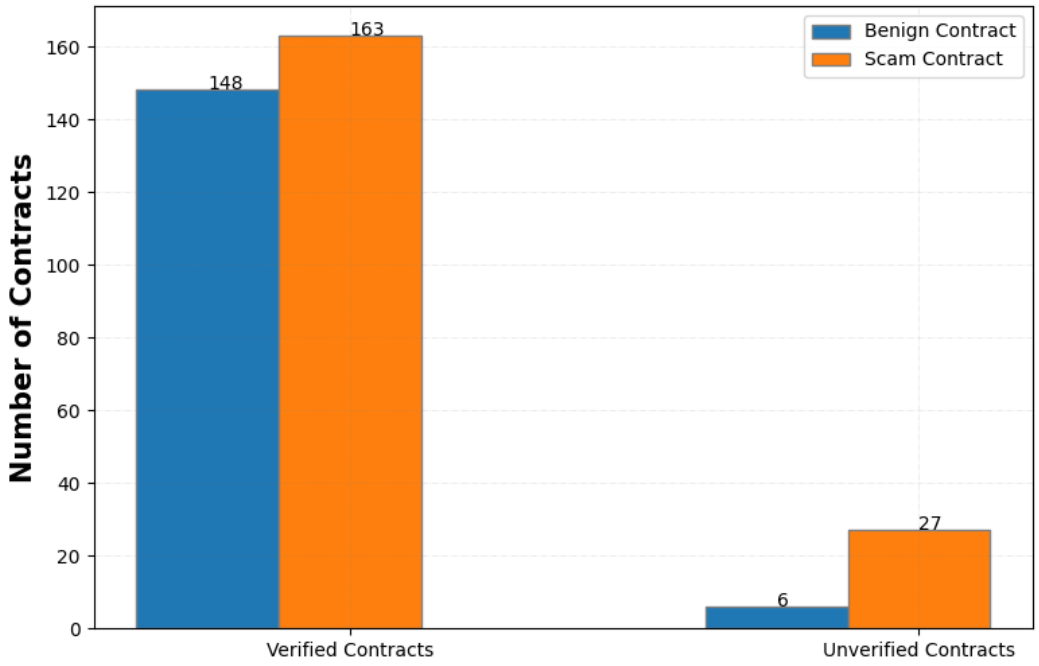


Figure 35: Distribution of Scam and Benign contracts with verified and unverified source code

According to the training dataset, an unverified contract is significantly more likely to be a scam than benign. This indicator marks unverified contracts as suspicious but does not decide on verified contracts.

The current confusion matrix base on the collected dataset is shown in Table 6.

	Indicator Scam	Indicator Benign
Actual Scam	27	0
Actual Benign	6	0

Table 6: Confusion matrix verified contracts

The indicator was not able to decide on 311 contracts of the dataset. These 311 contracts all have their source codes verified, which is shown to neither increase nor decrease the likelihood of being a scam.

4.3.3 Slither vulnerability scanning

If the targeted smart contract's source code is verified, the system runs the Slither vulnerability against its code. Otherwise, the indicator does not take a decision.

Slither searches for vulnerabilities in the smart contract source code and returns a list of its findings, which are then stored inside the database in the format listed in Table 7.

Object	Description
Address	Token Address on which the smart contract was deployed.
Error	Error Code
Vulnerabilities	An array in which the vulnerabilities are stored
Vulnerabilities: description	Raw text description of the found vulnerability
Vulnerabilities: check	Name of the check found in the smart contract
Vulnerabilities: impact	The impact of the found vulnerability can hold the following values: Optimization, Informational, Low, Medium, and High.
Vulnerabilities: confidence	The impact of the found vulnerability can hold the following values: Low, Medium, and High.

Table 7: Saved information of the indicator Slither

The collected dataset indicates that a few types of vulnerabilities are significantly more often present in scam contracts than in benign ones as seen in Figure 36:

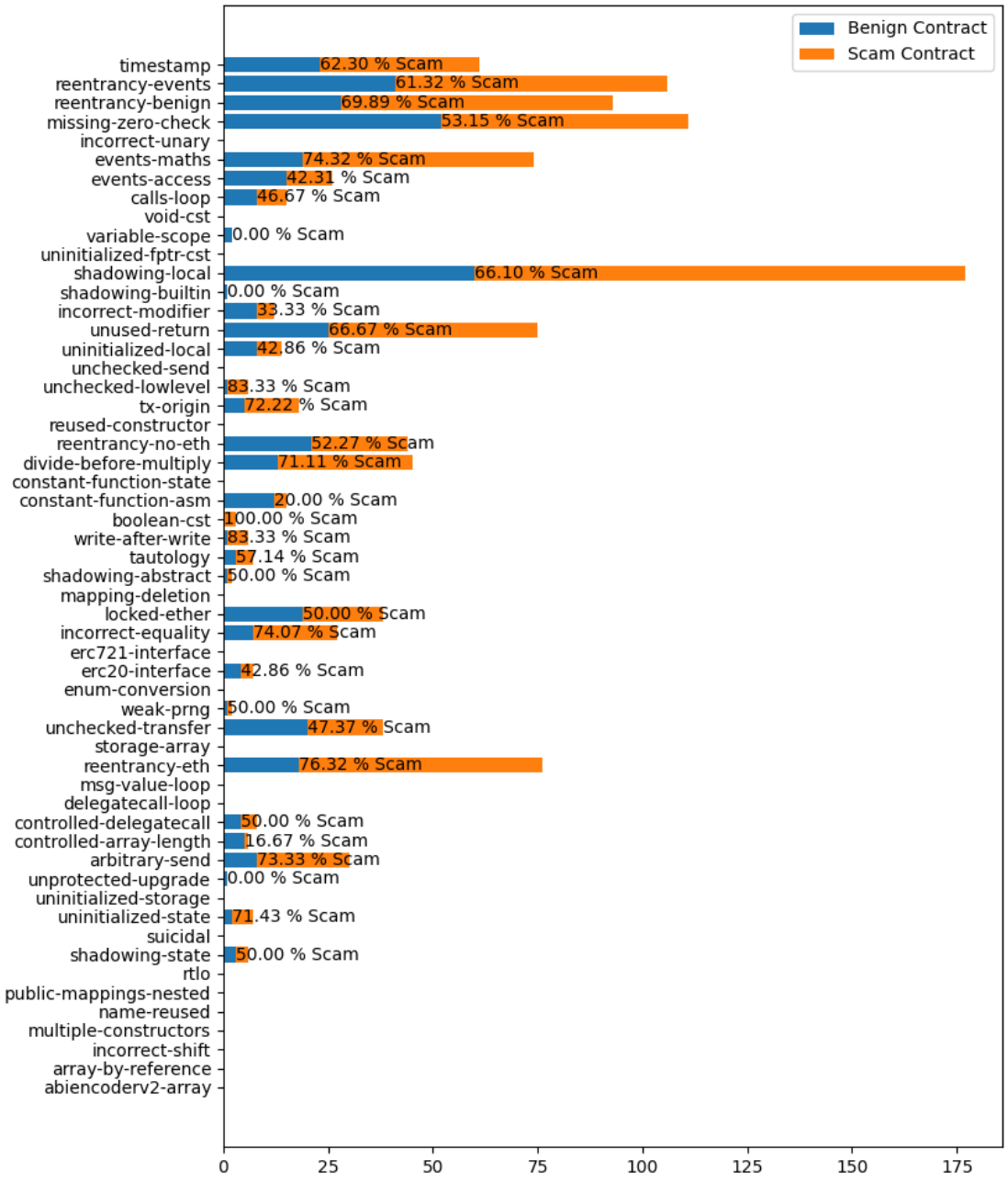


Figure 36: Distribution of Slither vulnerabilities and their scam likelihood

To evaluate a smart contract, the system will use the Slither checks in Table 8 to mark it as suspicious.

Detector	What it Detects	Impact	Confidence	Amount of Scams [%]
abiencoderv2-array	Storage abiencoderv2 array	High	High	-
array-by-reference	Modifying storage array by value	High	High	-
incorrect-shift	The order of parameters in a shift instruction is incorrect.	High	High	-

multiple-constructors	Multiple constructor schemes	High	High	-
name-reused	Contract's name reused	High	High	-
public-mappings-nested	Public mappings with nested variables	High	High	-
rtlo	Right-To-Left-Override control character is used	High	High	-
shadowing-state	State variables shadowing	High	High	50
suicidal	Functions allowing anyone to destruct the contract	High	High	-
uninitialized-state	Uninitialized state variables	High	High	71.42857
uninitialized-storage	Uninitialized storage variables	High	High	-
unprotected-upgrade	Unprotected upgradeable contract	High	High	0
arbitrary-send	Functions that send Ether to arbitrary destinations	High	Medium	65.51724
reentrancy-eth	Reentrancy vulnerabilities (theft of ethers)	High	Medium	68
incorrect-equality	Dangerous strict equalities	Medium	High	73.07692
locked-ether	Contracts that lock ether	Medium	High	50
shadowing-abstract	State variables shadowing from abstract contracts	Medium	High	50
write-after-write	Unused write	Medium	High	66.66667
boolean-cst	Misuse of Boolean constant	Medium	Medium	100
divide-before-multiply	Imprecise arithmetic operations order	Medium	Medium	65.90909
reentrancy-no-eth	Reentrancy vulnerabilities (no theft of ethers)	Medium	Medium	50
tx-origin	Dangerous usage of tx.origin	Medium	Medium	72.22222
unchecked-lowlevel	Unchecked low-level calls	Medium	Medium	83.33333
unused-return	Unused return values	Medium	Medium	60.81081
shadowing-local	Local variables shadowing	Low	High	63.06818
events-maths	Missing Events Arithmetic	Low	Medium	68.91892

Table 8: Slither vulnerabilities more common in scam contracts

The vulnerabilities, which are reported with high impact and confidence, are indicated as having a high likelihood to be scams. Vulnerabilities that show a clear discrepancy between occurrences in scam contracts and benign ones indicate that these might have been purposely placed, and the system marks those as suspicious with a high likelihood. Finally, some vulnerabilities could be purposely used to obfuscate mechanisms of the contract's source code. The system marks those as slightly suspicious.

If the targeted smart contract has not detected any vulnerabilities at all, the system marks the smart contract as more trusted.

The current confusion matrix based on the collected dataset is shown in Table 9.

	Indicator Scam	Indicator Benign
Actual Scam	139	18
Actual Benign	83	29

Table 9: Confusion matrix Slither vulnerabilities

The indicator was not able to decide on 75 contracts of the dataset due to their source code not being verified.

4.3.4 Liquidity Pool WETH Reserves

This indicator calculates the expected Uniswap V2 pair address with the targeted token and the WETH token address. It then proceeds to directly interface with the liquidity pool's smart contract and request its reserves of the WETH token. WETH is the most used trading currency on Uniswap V2 and, as described in Chapter 3, a certain amount of its reserves is required for the token to be tradable.

This indicator is not yet able to analyse other tokens than WETH, since the system cannot determine the value of the alternate token. It collects the information listed in Table 10 to store in the database.

Object	Description
Address	Token Address on which the smart contract was deployed.
WETH liquidity	Current Liquidity Pool reserve on WETH
LP Address	Address of the Liquidity Pool
Token 0	Token address of the first token in the pair
Token 1	Token address of the second token in the pair
Reserves 0	Amount of first token in the liquidity pool
Reserves 1	Amount of second token in the liquidity pool
Decimals	Maximum number of decimal places the token can have

Table 10: Saved information of the indicator Liquidity Pool Information

The most important information in this indicator is how much WETH reserves there are in the liquidity pool. The more illiquid the liquidity pool, the higher the price impact in a trade. This makes it unsafe for investors. In addition, a rug pull can be detected in retrospect if the ETH reserves are empty or only a few Gwei are available.

The WETH reserves collected from the used dataset look as follows. For visualisation purposes, WETH amounts higher than 100 have been truncated as seen in Figure 37:

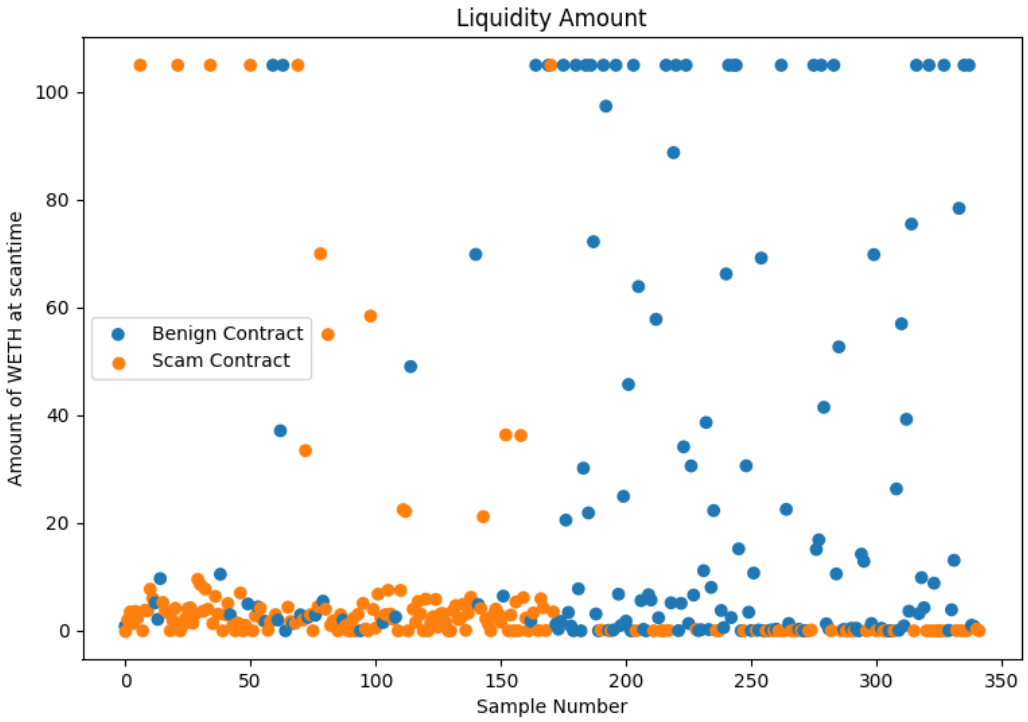


Figure 37: WETH amounts collected from the dataset

According to the collected dataset, a residual WETH amount less than 0.004 WETH heavily indicates that a malicious action has taken place, as seen in Figure 38. These tokens are marked as suspicious with a high importance, since the only expected false positive with this small amount of liquidity are migrated tokens, which also should not be considered completely benign.

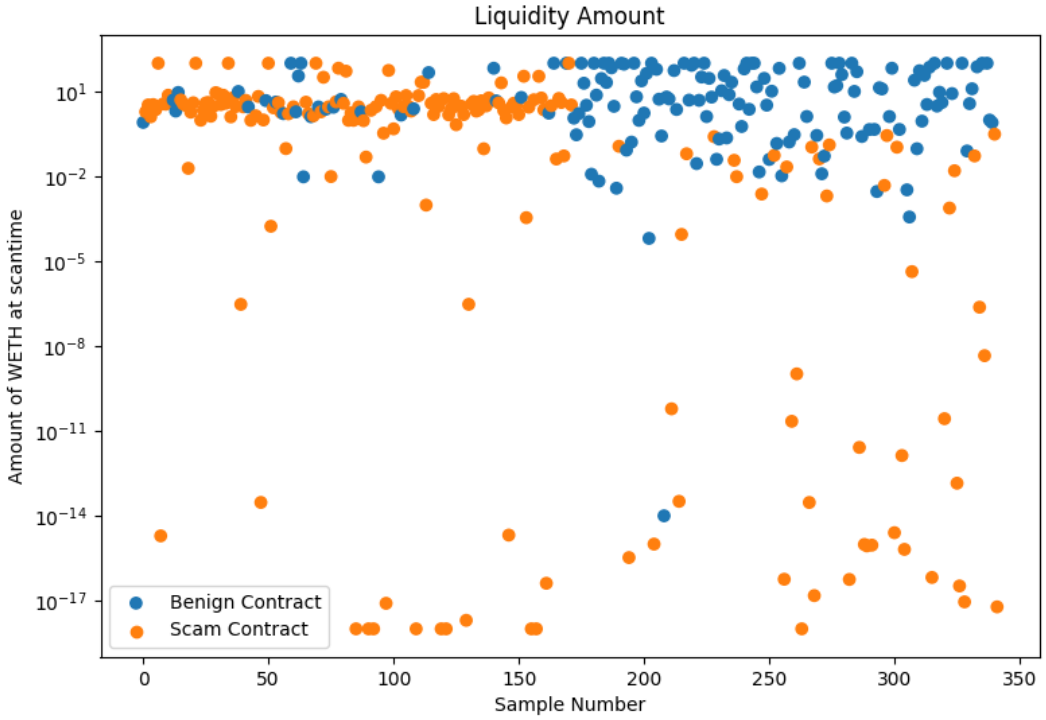


Figure 38: Logarithmic plot of WETH amounts

An amount below 0.5 WETH is still significantly more likely to be a scam than a benign contract and is used to mark targeted smart contracts as suspicious. Figure 39shows the distribution of scam contracts and benign contracts in the range of 0 to 0.5 WETH.

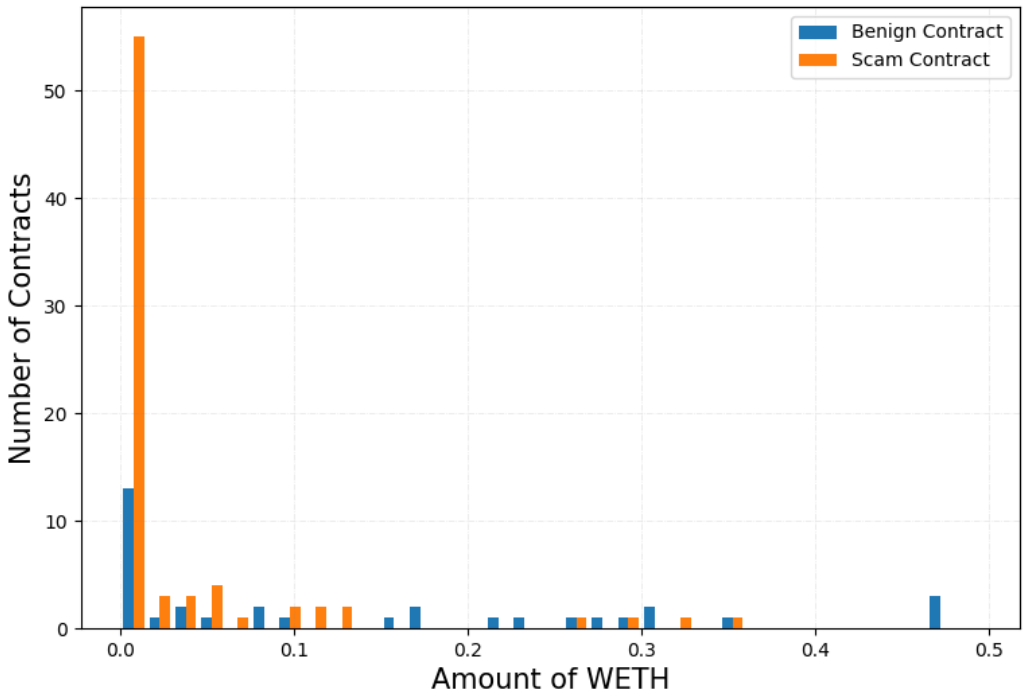


Figure 39: Overview of WETH amounts between 0 and 0.5

In contrast, a WETH amount higher than 40 is much more likely to be benign and only a few instances of scams have been collected in that range. This likelihood is factored in in favour of the targeted smart contract during the evaluation of this indicator.

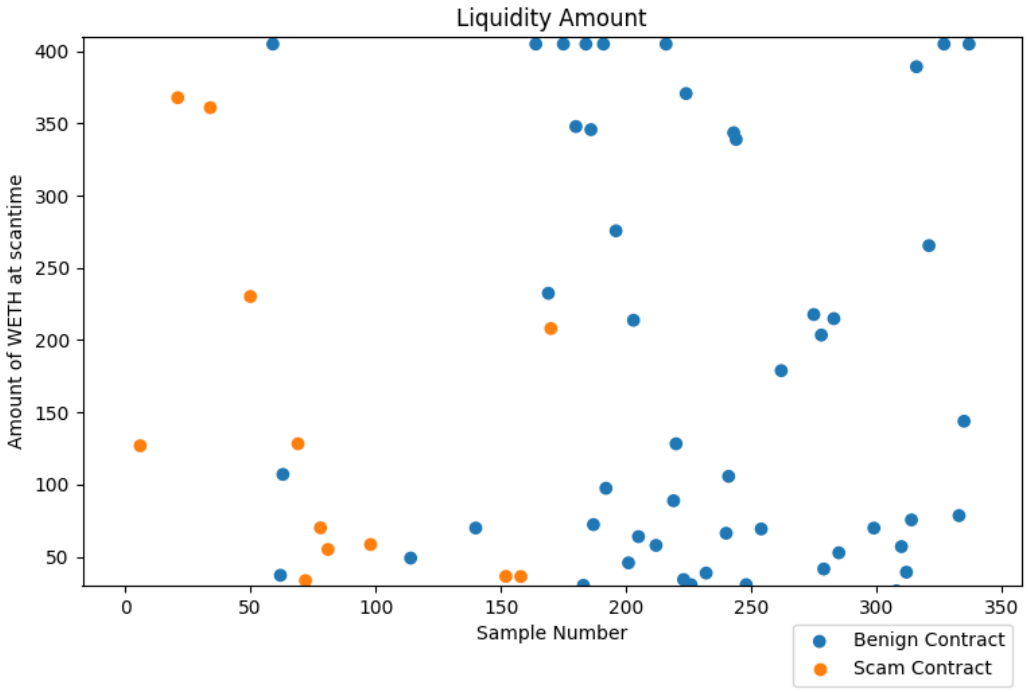


Figure 40: Liquidity amount plot between 40 and 400 WETH

This indicator is not very useful when the targeted smart contract has just been deployed. It is difficult to determine, whether it is a scam with little capital or a legitimate smart contract. This indicator is particularly strong to see if the token is illiquid or no longer tradable. In this case, it can be said with a very high probability that the project is a scam. However, this scam has often already been executed and is therefore only applicable post-mortem.

Running this indicator against the dataset results in a confusion matrix seen in Table 11:

	Indicator Scam	Indicator Benign
Actual Scam	77	9
Actual Benign	34	42

Table 11: Confusion matrix liquidity amount

Two smart contracts do not have a WETH pair and 180 contracts have WETH reserves between 0.5 and 40 WETH and thus are not decided on.

4.3.5 Top holders and whales

The wallets currently containing the highest market share can be calculated using the initial state of contract deployment and all the following transactions. However, Ethplorer's [98] API offers these results directly and is thus included in this indicator. The following fields in Table 13 are collected and stored from queries to Ethplorer:

Object	Description
Address	Token address on which the smart contract was deployed
Holders Count	Number of total token holders
Holders	A sorted array in which the top 100 holders are listed
Holders: address	The address of the holder
Holders: balance	The amount of the current token at scan time
Holders: share	The percentage of tokens from the maximum supply

Table 12: Saved information of the indicator Top Holders and Whales

To achieve results that are more accurate, the indicator performs some post-processing on this data. First, it gets the total market share from the general information module and subtracts all burnt tokens from the total. Otherwise, the percentual market share would deliver warped results. Afterwards, the indicator loops through the sorted list, ignoring burn addresses and the WETH pair of the token and returning the highest identified result.

This is the point where the current implementation of this indicator has a shortcoming. The system cannot currently distinguish between normal wallets and additional liquidity pools, thus if a token is included in more than one liquidity pool, the results of this indicator are slightly unreliable.

However, this indicator can still make a judgement on the targeted smart contract as seen in Figure 41 and Figure 42. Holders with shares above 100% existed and have been limited to an upper value of 105% to simplify visualization.

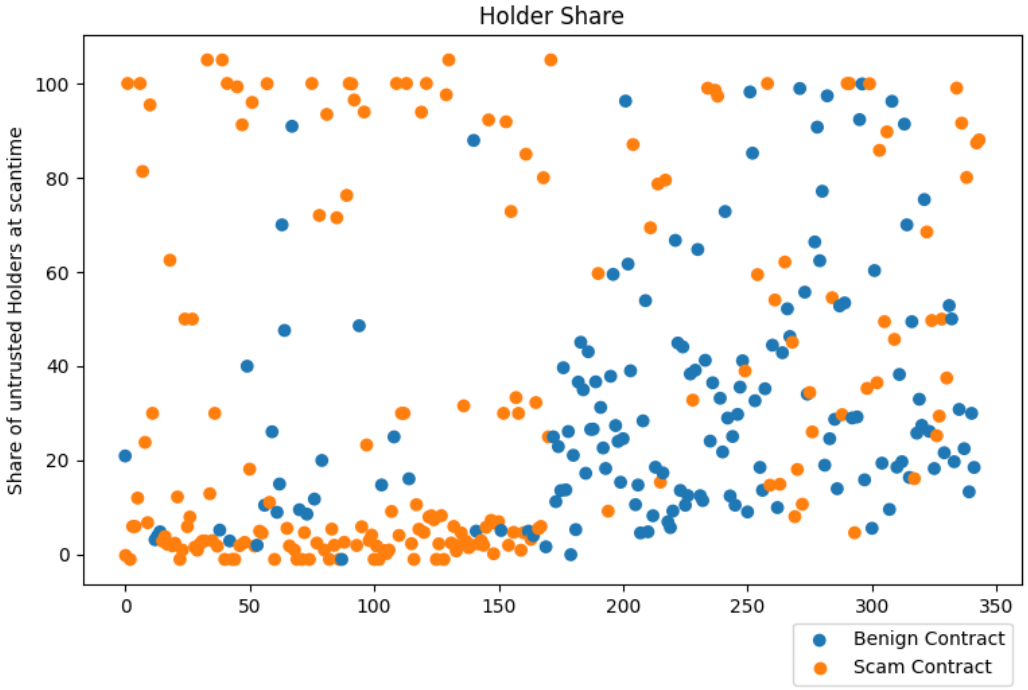


Figure 41: Overview of highest, unprotected market shares

The overview of the holders with the highest share in Figure 42 shows that a high amount of market share owned by an untrusted wallet increases the likelihood of the contract being a scam.

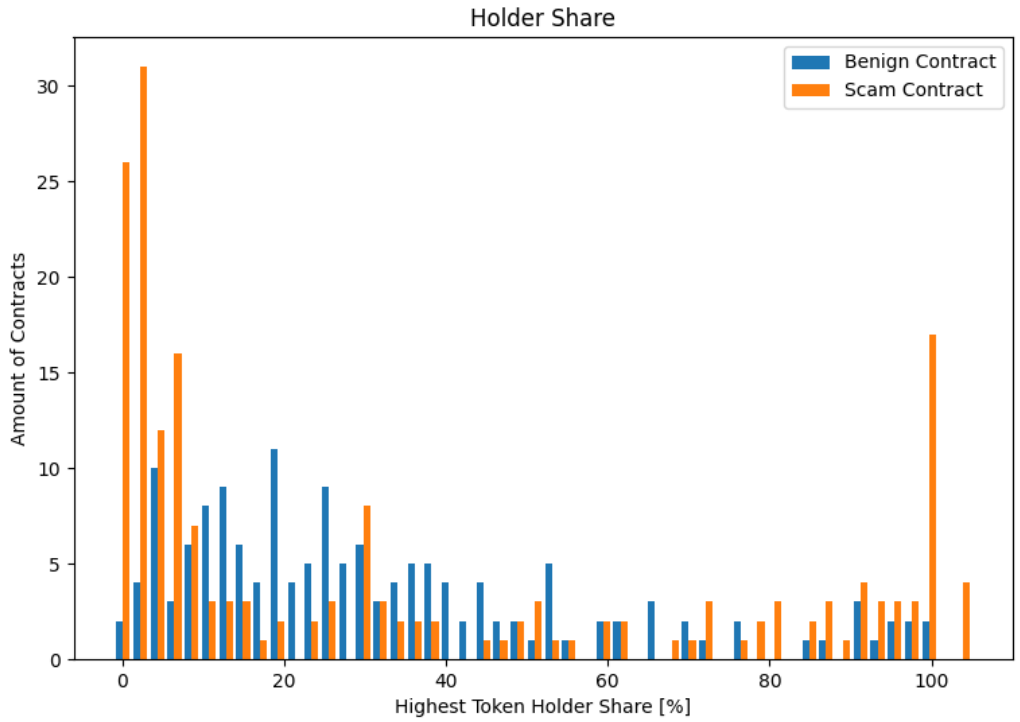


Figure 42: Distribution of Scam and Benign contracts according to the highest token holder

Wallets holding more than 70% of the circulating market share are significantly more likely to be scams according to our dataset and are thus marked as suspicious by the system. Wallets with the highest holder owning more than 61% of the market share are slightly more suspicious and marked as such by the system. Additionally, if the highest holder holds less than 5% of the market share the data indicates that the likelihood of the smart contract being a scam should increase. This, however, is most likely a false positive due to the bias inherent to the dataset, as seen in Figure 41: Overview of highest, unprotected market shares, where mostly the newly created contracts have holders with such small amounts of the share. During the collection period, most of the newly created smart contracts have been proven scams as described in Chapter 4 and the system was only able to collect a few legitimate newly created contracts. Thus, this statistical observation is not included in this indicator.

An additional observation was that the calculation of the holder share resulted in cases, where the highest token holder held more than 100% of the token's total supply. This is most likely due to new tokens being minted and, according to the dataset, only happened in scam contracts. Instances where this is the case are marked as suspicious by the indicator.

Running this indicator against the dataset results in the confusion matrix shown in Table 13.

	Indicator Scam	Indicator Benign
Actual Scam	53	0
Actual Benign	22	0

Table 13: Confusion matrix top token holders

The results shown in the confusion matrix do not consider the different significances of the indicators' output. For this confusion matrix, any increase in suspicion is treated as a positive finding. In later processing, the false positives will be reduced by the scaled value returned by the indicator.

4.3.6 Liquidity Pool Token Holders

As with the top token holders, the liquidity pool token holders are collected via the Ethplorer API and stored according to Table 14.

Object	Description
Address	Token Address on which the smart contract was deployed.
Pair	Address of the Liquidity pool
Holders Count	number of token holders
Holders	An array in which the liquidity holders are listed
Holders: Address	Address of the holder
Holders: Balance	LP-token balance
Holders: Share	Percentage share of total LP-token in supply

Table 14: Saved information of the indicator Liquidity Holders

This indicator struggles with a similar issue as the top token holder indicator, such that an early analysis of a smart contract does not result in a great variety of values. Most commonly, a newly created liquidity pool is either burnt, locked, or belongs to the smart contract creator as seen in Figure 43, where the samples on the left side are taken from currently monitored contracts and the higher samples belong to the older, matured contracts.

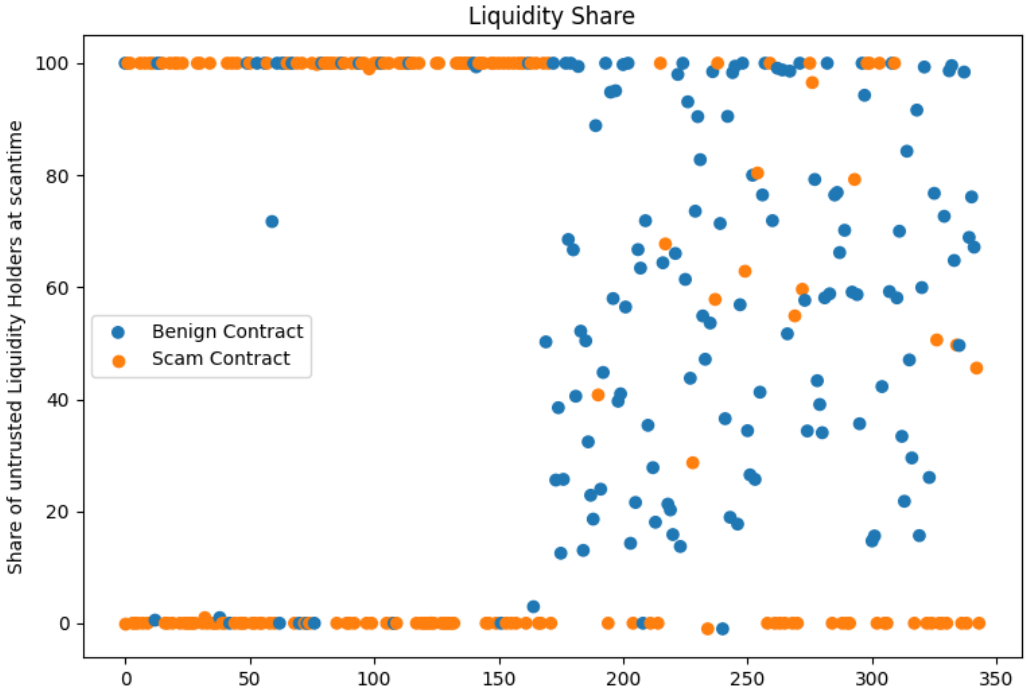


Figure 43: Highest unprotected liquidity holder share overview

This indicator can however still decide to mark smart contracts with most of the liquidity pool left unprotected as suspicious, since this behaviour is constant throughout the whole dataset as seen in Figure 44.

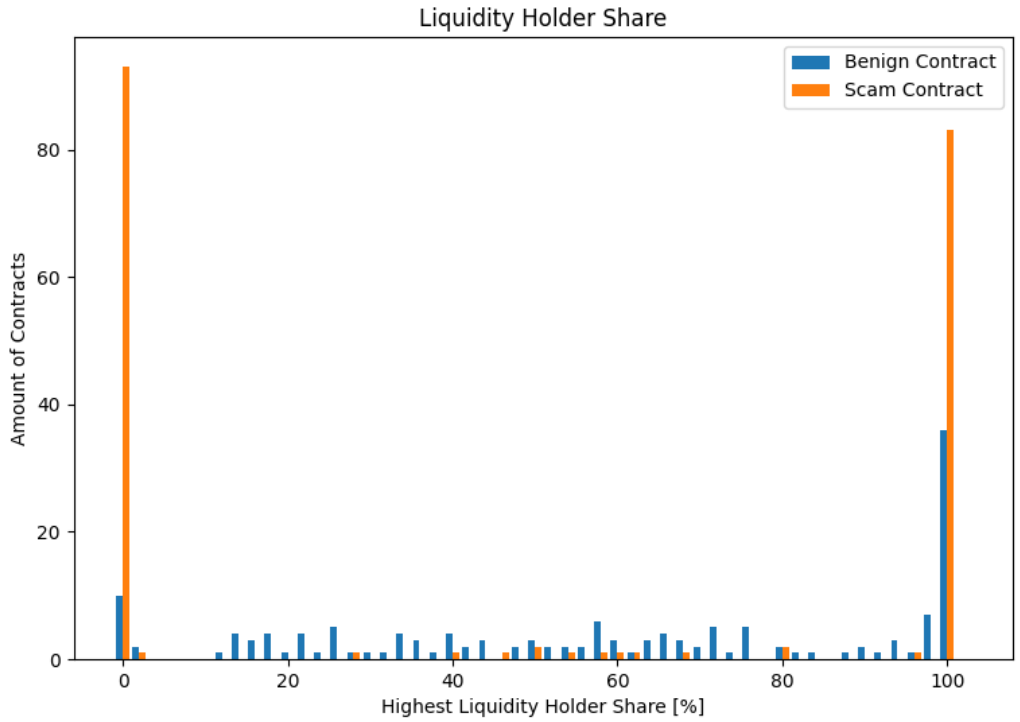


Figure 44: Distribution of highest unprotected liquidity holder share

The data in Figure 44 shows that if over 99% of the liquidity pool is unlocked or unburnt, the contract is more likely to be a scam. The indicator will mark these contracts as suspicious.

Same as with the top token holder, however, the data indicates that if all of the liquidity is burnt or locked, it is still more likely to be a scam. Some of those results are likely a false positive, resulting from an uneven amount of new benign and new scam contracts in the dataset, however as seen in Figure 43, the same observation also applies to older sample contracts. During manual analysis it was observed that locked tokens are very likely to be rug pulled as soon as the time of the lock runs out. Results with all their liquidity burnt or locked will thus be marked as slightly more suspicious.

Contracts with the highest liquidity pool holder of less than 5% are not decided on. Everything in between will be marked as more trusted.

This results in the following confusion matrix, based on the used dataset is shown in Table 15:

	Indicator Scam	Indicator Benign
Actual Scam	176	11
Actual Benign	61	89

Table 15: Confusion matrix for highest liquidity shareholders

The indicator was not able to make a decision on seven smart contracts, since they did not have a WETH pair. Even though there are some false positives listed in this confusion matrix, their impact is reduced by scaling them during further processing.

4.3.7 Honeypot.is

To find out whether a smart contract is a honeypot, Honeypot.is [72] is used. Honeypot.is simulates a purchase in a transaction and then a sale. This is to find out whether a sale is possible or not. Honeypot.is uses checks such as time jumps to minimise the number of false results. This indicator requests and stores the data shown in Table 16 from the Honeypot.is API:

Object	Description
Address	Token Address on which the smart contract was deployed
Is honeypot	The result if Honeypot.is thinks it is a honeypot or not as a Boolean
Error	Error Message from Honeypot.is
Buy Tax	The fees required for a buy in percent.
Sell Tax	The fees required for a sell in percent.
Buy Gas	Gas fees used to perform buying transaction
Sell Gas	Gas fees used to perform sell transaction

Table 16: Saved information of the indicator Honeypot.is

According to the used dataset, a smart contract marked by the honeypot indicator is significantly more likely to be a scam than benign. This is visualised in the distribution of scam and benign contracts in Figure 45.

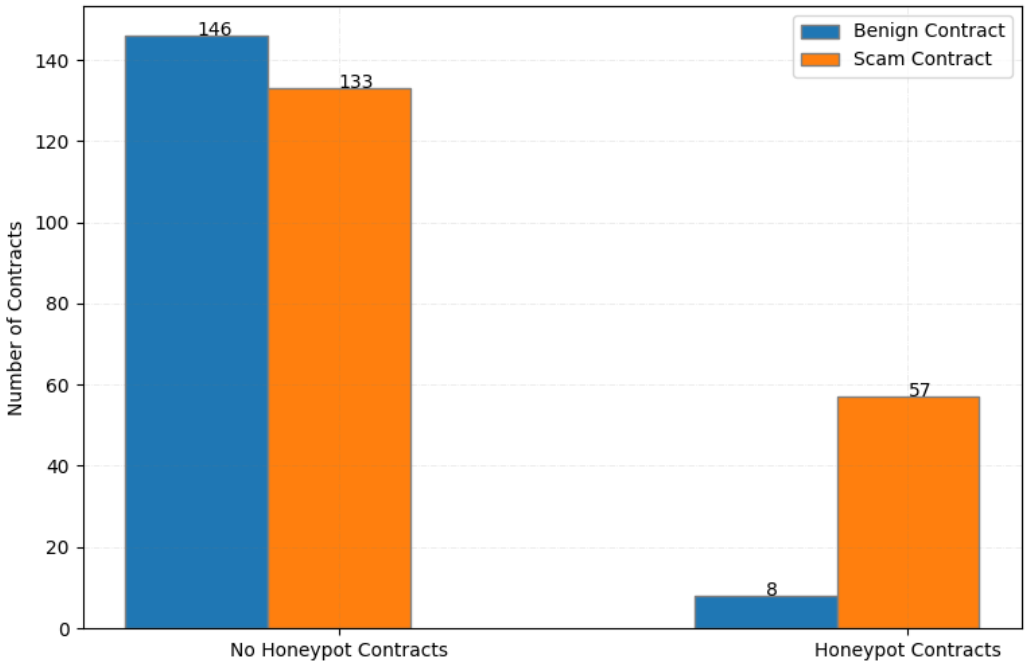


Figure 45: Distribution of Scam Contracts on Honeypot indicator

This indicator marks contracts identified as honeypot by the API as suspicious but does not take a decision on unmarked contracts.

	Indicator Scam	Indicator Benign
Actual Scam	57	0
Actual Benign	8	0

Table 17: Confusion matrix for honeypots

The indicator did not make a decision for 277 smart contracts of the dataset. Those were reported to be functional by the API and there is no significant increase in scam likelihood visible.

4.3.8 Ownership

As mentioned in Chapter 3, the concept of ownership is not standardised. During analysis of the dataset, it was observed that contrary to expectations, the legitimate contracts are more likely to keep the ownership and scam contracts revoke the ownership more often. Additionally, if no ownership information can be found, the smart contract is more likely to be a scam.

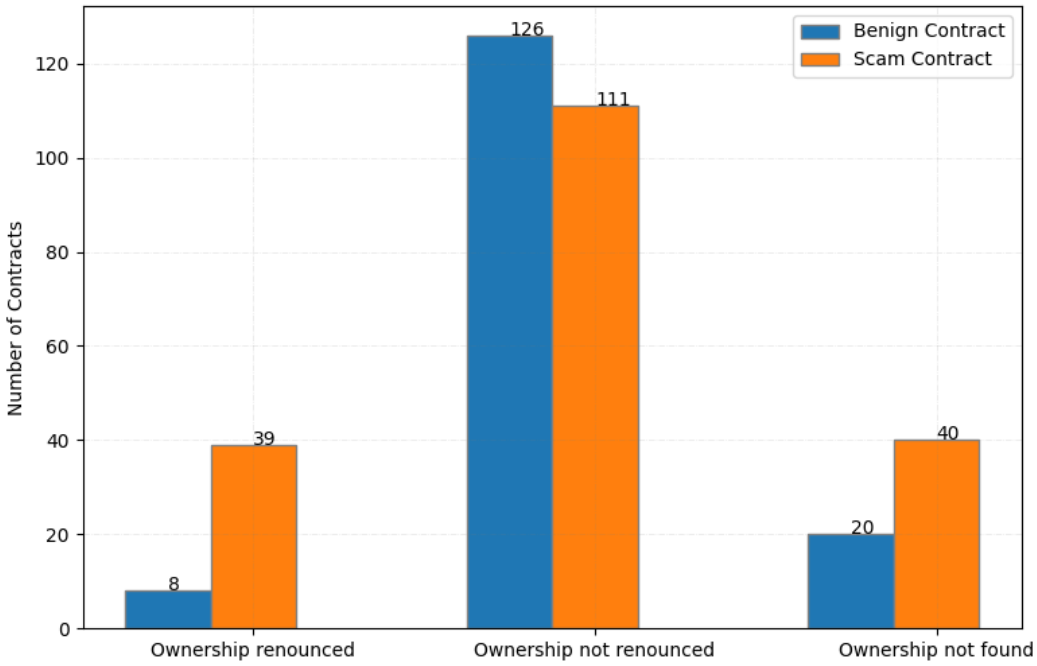


Figure 46: Distribution of smart contract ownerships

The system will mark contracts that have their ownership renounced or where no ownership information was found as slightly suspicious. It will not decide on contracts where the owner was identified but not actively renounced.

	Indicator Scam	Indicator Benign
Actual Scam	79	0
Actual Benign	28	0

Table 18: Confusion matrix for ownership results

The indicator did not decide for 236 smart contracts. Those 236 contracts did have an owner that was not renounced, but the likelihood of a scam did not decrease significantly enough to mark those as more benign.

4.4 SCSC Result

To calculate how well the system classifies targeted smart contracts, the F1-score algorithm is used. The F1-score is calculated from the following algorithms.

Accuracy shows how high the proportion of correctly identified smart contracts is. Neither false positives nor false negatives are taken into account, as seen in Equation 9:

$$Accuracy = \frac{True\ Positive}{Number\ of\ Total\ Vulnerabilities} \tag{9}$$

Precision indicates the percentage of positive identifications that are correct, which can be seen in Equation 10:

$$Precision = \frac{True\ Positives}{True\ Positive + False\ Positive} \tag{10}$$

Recall shows the proportion of true positives that were correctly identified:

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative} \tag{11}$$

The F1 Score brings Precision and Recall into a measure that shows the suitability of a model:

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall} \tag{12}$$

4.4.1 Custom Score Evaluation

During the development of the system, it was important to receive visual feedback from the indicators and the final score. As a result, the first approach for classifying the targeted smart contracts as scam or benign is based on a weighted average over all the received indicator scores. Each indicator returns a score between zero and one, were lower values suggest, that the targeted contract is more trustworthy and higher values mean, that the contract is more likely to be a scam. The neutral value 0.5 is used, if the indicator cannot decide.

The weight of the indicator's score is chosen, based on the accuracy of the respective indicator. An indicator with very few false positives must not be overshadowed by an indicator with a greater contract coverage, but which includes more false positives or false negatives.

During experimentation with different weights, it was possible to split the indicators into two categories.

The first category searches for very specific smart contract features, which are only present in some of the smart contracts inside the dataset. These indicators have a high accuracy and are weighted with a factor of two. The indicators inside this category are "verified contract", "honeypot", "liquidity amount", "ownership", and "top token holder".

The second category contains indicators that cover a broad range of smart contracts, but are subject to more misclassifications. The goal is that these indicators can affect the contracts in the first category sufficiently without overshadowing their more accurate results. The indicators in this category are "top liquidity holder" and "vulnerabilities" and are weighted with a factor of one.

With the scores and the weights determined, the SCSC systems multiplies each indicator score with the respective weight and divides the result with the sum of all weights. This results in a final score between zero and one. An overview of all the calculated scores of the training dataset is shown in Figure 47.

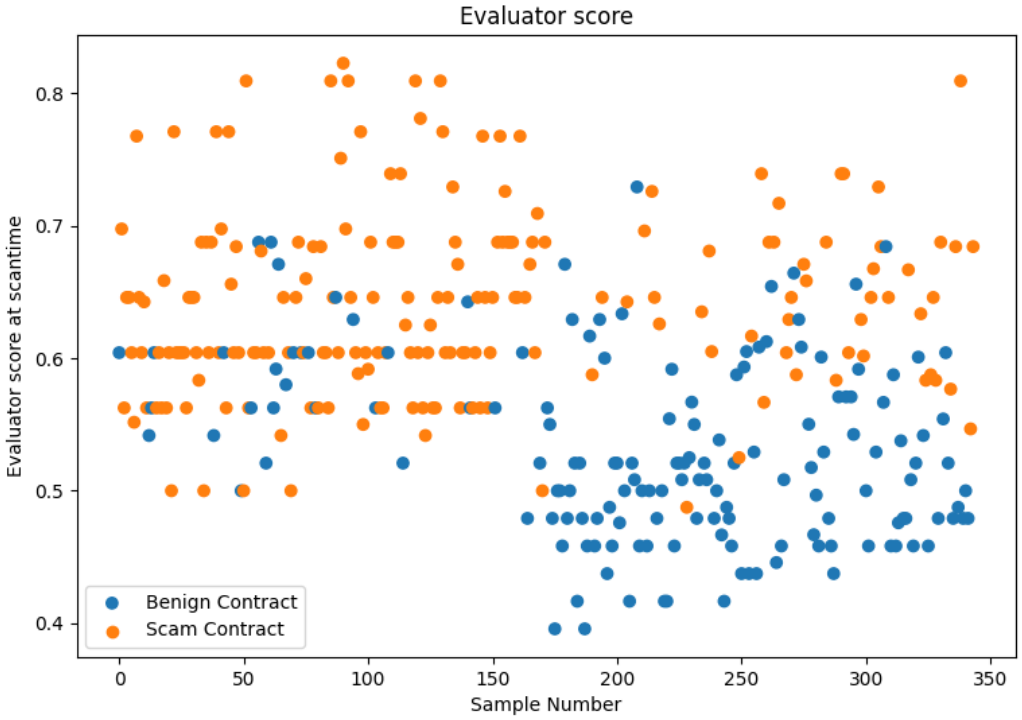


Figure 47: Overview of custom evaluation score

The result in Figure 47 shows that the indicators can successfully distinguish between legitimate and scam tokens to some extent. There are samples that can clearly be separated from each other. However, between the range of 0.5 and 0.6, there is still a significant overlap left, making it difficult to classify those. The distribution in Figure 48 shows at which point the cut-off point should be defined.

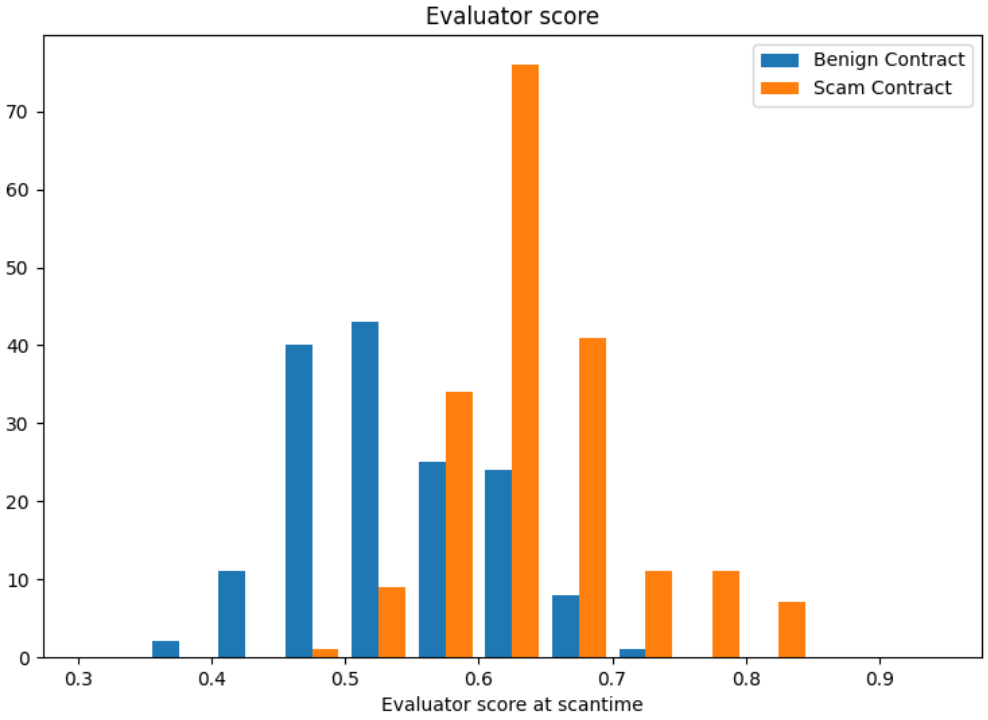


Figure 48: Distribution of Scam and Benign Contracts based on custom evaluator score

Based on these results, the cut-off value for the final classification was chosen at 0.58. Every contract which receives a final score higher than 0.58 will be marked as scam and every contract with a lower score marked as benign.

The confusion matrix of this classification on the training dataset looks as follows.

	Indicator Scam	Indicator Benign
Actual Scam	155	35
Actual Benign	39	115

Table 19: Confusion matrix for custom classification with manual cut-off

This results in an accuracy of 78%, a precision value of 80%, a recall value of 82%, and finally an F1-score of 81%.

Due to the high number of false positives and false negatives that originate from the overlapping data in the middle of Figure 48, the result should be displayed in form of a probability.

In the application, a Support Vector Machine with a linear kernel is used on this one-dimensional date, essentially automating the manual determination of the cut-off point. Additionally, this allows for an automatic calculation of a probability value for each possible evaluator score. The confusion matrix of the Support Vector Machine matches up very closely with the manually determined results and provides the following confusion matrix.

	Indicator Scam	Indicator Benign
Actual Scam	156	34
Actual Benign	44	110

Table 20: Confusion matrix for custom classification with SVM

Running this classifier against the test dataset resulted in the following confusion matrix:

	Indicator Scam	Indicator Benign
Actual Scam	61	17
Actual Benign	10	6

Table 21: Confusion matrix for custom classification on the test dataset

The calculated accuracy is 71%. It shows a precision of 86% with a recall of 78% resulting in an F1-score of 82%

The distribution of the custom classification scores is show in Figure 49 and shows that newly scanned smart contracts are difficult to classify as benign with this approach.

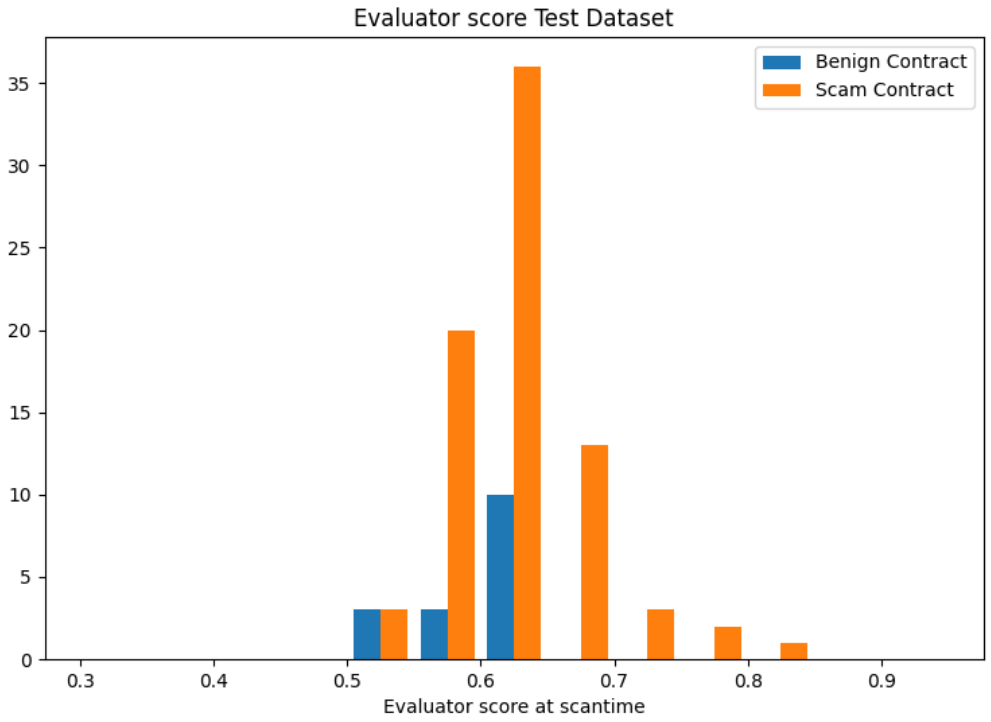


Figure 49: Distribution of the custom evaluator score on the test dataset

4.4.2 Support Vector Machine (SVM)

After analysing the results of the first method, the goal was to test if the classification accuracy can be increased further by using a Support Vector Machine [110] with the unprocessed indicator scores as features, instead of weighting them manually.

The Principal Component Analysis of the collected indicator scores shows that it should be possible to separate legitimate contracts from scams reasonably, as visualised in Figure 50.

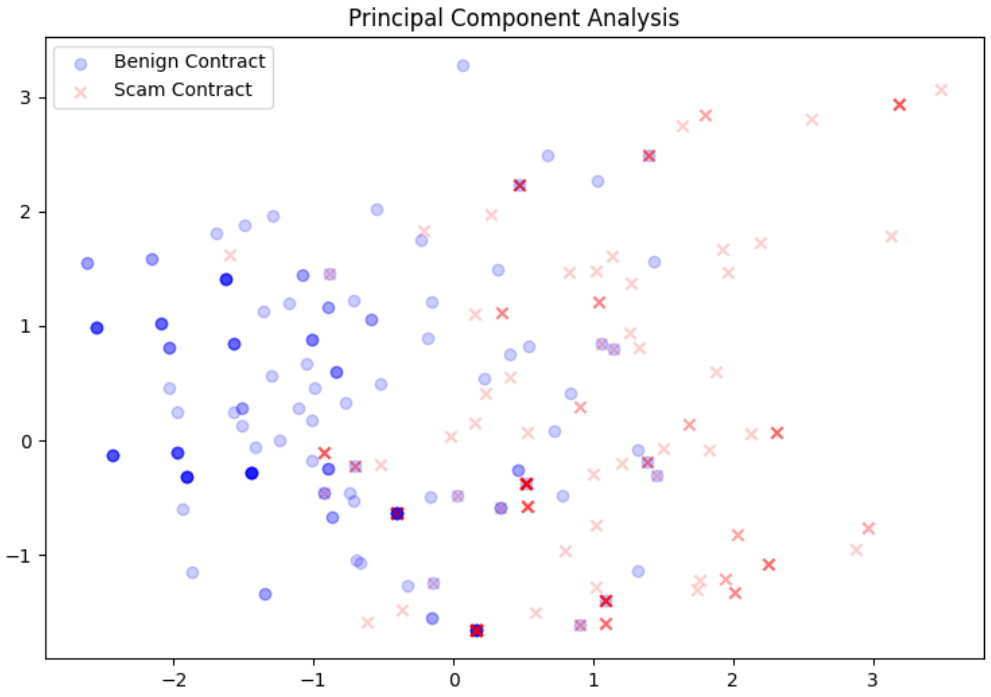


Figure 50: Principal Component Analysis of the current dataset

After trying out different kernels, the one offering the best results was the Radial Basis Function. After fitting the model onto the training data, it delivered better results than the custom weighted average classifier. The confusion matrix based on the training dataset is shown in Table 22.

	Indicator Scam	Indicator Benign
Actual Scam	179	11
Actual Benign	46	108

Table 22: Confusion matrix for SVM classifier with RBF kernel on the training dataset

On this dataset, the SVM classifier managed to get an F1-score of 86%.

Running this classifier against the test dataset resulted in the confusion matrix shown in Table 23.

	Indicator Scam	Indicator Benign
Actual Scam	75	3
Actual Benign	13	3

Table 23: Confusion matrix for SVM classifier with RBF kernel on the test dataset

The calculated accuracy is 83% and the F1-Score 90%. Due to the unbalanced test dataset, this is misleading. It only classifies three of the 16 legitimate contracts accurately. This problem is currently not addressed in this work but is planned after collecting a bigger test dataset and balancing it out with more legitimate, new smart contracts. It is adjusted for by returning a probability value instead of a binary classification.

4.4.3 XGBoost

The third method of classifying targeted smart contracts as benign or as scams was to use the XGBoost library [111]. The system currently contains seven weak indicators, which on their own cannot accurately classify smart contracts. This environment is suitable for boosting algorithms to perform well, even on small datasets.

The goal of this method was to compare the result of the XGBoost library to the other two methods and additionally to interpret which feature importance XGBoost considers best based on the collected dataset.

After training the XGBoost classifier on the training dataset, it calculated the importances per indicator show in Figure 51.

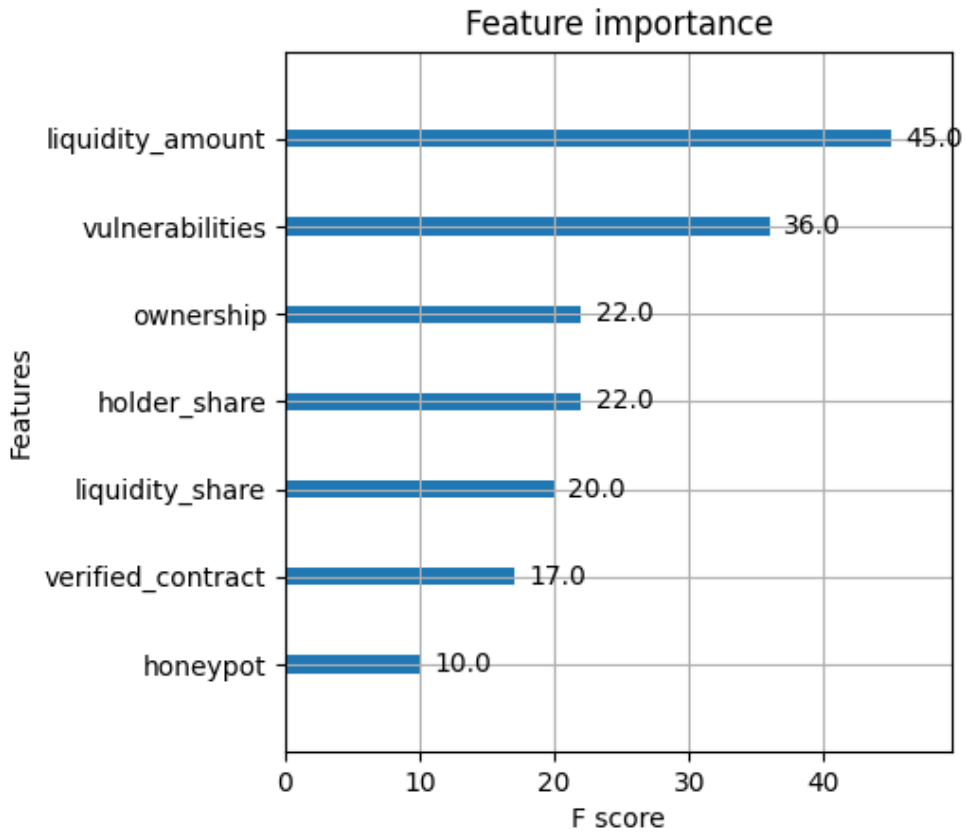


Figure 51: XGBoost feature importances

Running this method against the training dataset results in the confusion matrix shown inTable 24.

	Indicator Scam	Indicator Benign
Actual Scam	180	10
Actual Benign	38	116

Table 24: Confusion matrix for the XGBoost classifier on the training dataset

On this dataset, the XGBoost classifier managed to get an accuracy of 86% and an F1-score of 88%, which are currently the best results.

Running this classifier against the test dataset resulted in the confusion matrix shown in Table 25.

	Indicator Scam	Indicator Benign
Actual Scam	71	7
Actual Benign	12	4

Table 25: Confusion matrix for the XGBoost classifier on the test dataset

This classifier also delivered the best results on the test dataset, with an accuracy of 80% and an F1-score of 88%. Even though the F1-score is slightly lower than that from other classifiers,

this one managed to identify more of the legitimate smart contracts than the others did without producing too many false negatives.

4.4.4 Result Rescan

As an experiment, a manual rescan of the collected training dataset was performed 12 days after the initial monitoring results. Using the custom classifier to visualise the data showed the results in Figure 52:

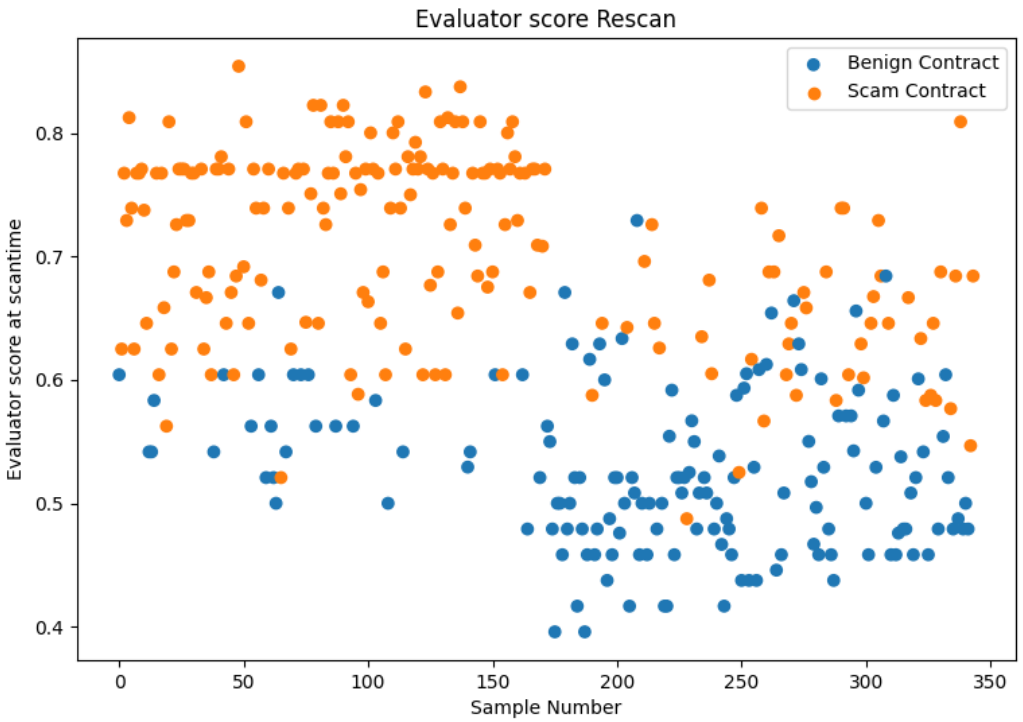


Figure 52: Overview of custom classifier score on rescanned dataset

This shows that a later analysis of the contract can lead to much better results, at the cost of some smart contracts already getting rug pulled during those few days of delay.

It also shows that our currently configured model is still applicable for rescanning tokens without any modifications to the model. This is important since the goal was to create a system capable of analysing both scenarios.

The distribution of the benign and scam contracts in Figure 53 also shows the same behaviour. However, the originally newer tokens are now properly classified, since some of the indicators now had sufficient information to mark them properly according to their logic.

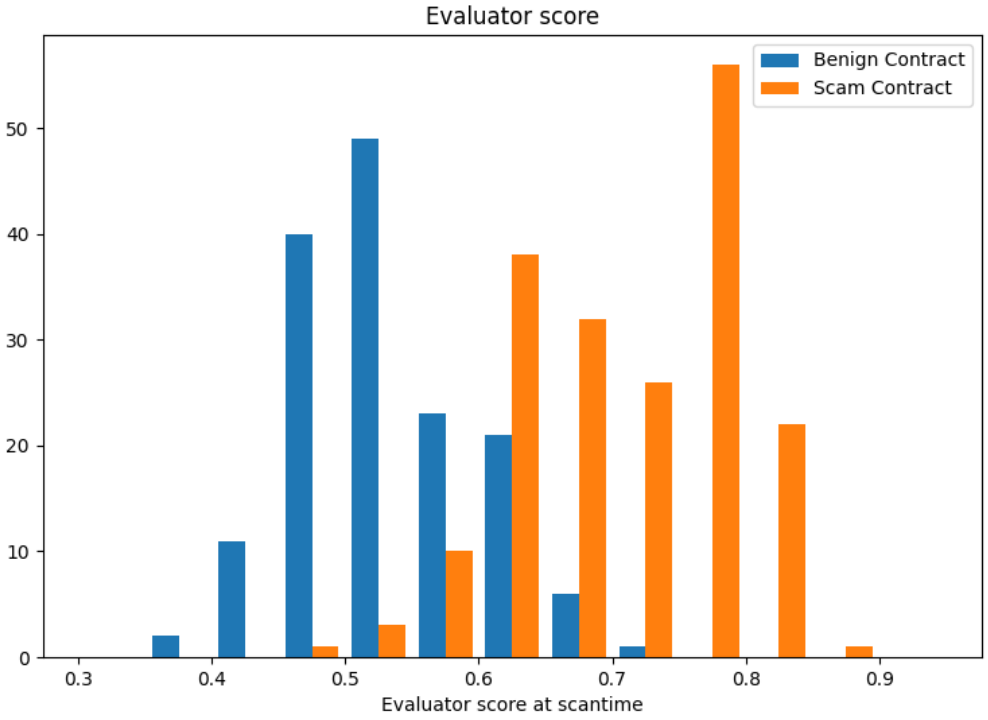


Figure 53: Distribution of scam and benign contracts after rescanning

5 Discussion

The system contains multiple elements working together, which in their current state show that it is possible to differentiate between benign and scam contracts with some certainty. However, each of those elements has its own remaining limitations after finishing this project. This chapter will reiterate the current state of the system and discuss potential improvements and limitations.

5.1 Data collection

Due to the limitation of using APIs to query the currently available information on smart contracts, the SCSC system is unable to collect historical data on tokens. This resulted in the necessity of collecting our own dataset, with the live gathered information as a data foundation as described in Chapter 4.1. The current approach for this project was to collect data as soon as possible, wait a sufficient amount of time until the scams are performed, and then label the tokens post-mortem. The downside of this approach is a relatively limited dataset, which could only be acquired during the creation of this work, as well as the downside of not being able to include and compare other existing datasets.

Additionally, the information on those tokens is collected after between 5 and 40 minutes of the `PairCreated` event. Currently, the system is not configured to periodically request additional data from the APIs, and in Chapter 4 as well as in the results of Xia et al. [2], it is shown that further analysis increases the accuracy of the results. It was a design decision to scan as early as possible, but for future work it might be beneficial to periodically rescan the tokens and compare the results further. However, a rescan later in the lifespan of a smart contract will give results that are more accurate in indicators where data can change such as token holders and liquidity pool holders. A rescan can detect in retrospective that a token is a scam, but there is no benefit for investors since it is too late. Therefore, there is this interesting detection time vs. accuracy trade-off for scam detection in this case.

Arguably, an option to mitigate both of these limitations would be not to use the APIs to query the necessary information and instead use an Ethereum node for collecting the data. There were sporadic problems with the APIs, especially Ethplorer, which repeatedly indicated that a token did not yet have a token holder, even though the smart contract had already been deployed on Uniswap and already had token holders. One assumption as to why this message comes up is that the blocks that come via Etherscan have not yet been processed by Ethplorer. Depending on the number of transactions in the blocks, this could lead to long calculations. This has led to many retries in the monitoring and thus a variance in the average scan time.

One way to process this data consistently and deterministically would be to host a separate Ethereum node. An advantage of a node would be the access to historical data down to the

block in which the transaction was mined. This would be especially useful for the verification of the system, so that any old data from smart contract could be accessed and checked for when and how these scams were carried out. This is also the reason why no foreign datasets were used as the one from Xia et al. [2]. If the data is queried via the APIs, only the already untradable smart contracts would be obtained and analysis of these would not give any meaningful data. Another advantage is that custom queries can be made to match the indicators so-called remote procedure calls (RPC) [112]. The main disadvantage is the implementation effort. For this, so-called remote procedure calls would have to be written that are very complex and costly to develop. In general, it is time-consuming to evaluate RPCs, the information from the blocks must be calculated in order to arrive at the desired result, which is computationally intensive for large amounts of data. Therefore, custom RPCs would have to be written that only output certain information.

5.2 Indicator performance

Currently the system relies on data processed by the indicators for any classification. The indicators are designed such that they return a categorical value. The reason behind those categorical values was the small dataset used for identifying those observed points. It might be advantageous to re-evaluate the indicators on a bigger dataset and potentially let them return a probabilistic value instead of categorical labels.

During analysis of each indicator's results, some trends in outliers were identified. It might be advantageous to further research those trends in future work and evaluate if these can be accurately identified and thus counteracted.

- **Migrated Tokens**

  If a token needs to fork to another contract or chain, the old resources are reused as often as possible. This leads to the old contract becoming untradable on a DEX. This is not a malicious action and the contract creators often offer up a way to transfer the tokens of each holder to the new contract. However, the current system will always mark those as scam and it does not have a way to distinguish between a migration and a rug pull. Further research is necessary to see if it is possible to distinguish between those two scenarios

- **No token holders at scan time**

  A conspicuous observation is that most tokens that have the highest token holder only having fractions of a percent as share are scams. It is assumed that legitimate tokens more often perform certain actions before they submit their token to a DEX. When evaluating the token holders, it became apparent that legitimate tokens give investors an opportunity to obtain tokens even before the official release. This is done with presales or airdrops. Presales can take place privately or publicly. The private presales can

only be obtained or purchased directly from the developers. In public presales, people can buy the tokens before releasing under certain conditions. Airdrop is a marketing method in which small amounts of new tokens are distributed for free. Sometimes these are tied to tasks or are given to token holders of other tokens on a pro-rata basis.

As such, it might be advantageous to research this behaviour further, because this would have the potential to become an additional indicator.

5.3 SCSC Accuracy

After analysing the results of all three classifiers, some of them produced better results than others did. The current main problem is the amount of labelled data that can be used to test the current system. While collecting the test dataset, most of the newly created smart contracts indeed turned out to be scams, heavily affecting the balance of the test dataset.

For future work, the size of the test dataset should be greatly increased so that it is possible to properly balance out the scam contracts and benign contracts. Right now, the F1-score appears to be very good, but the reported false positives are not at an expected level.

5.4 Period Discussion

During the evaluation of the data, several periodic factors stood out.

5.4.1 Ether Price impacts Smart Contract Deployment

While working with smart contracts, it was particularly noticeable that while Ether has fallen in price, many fewer smart contracts have been deployed. When the price recovered, the smart contracts were deployed again. This correlation was also confirmed in a paper by Bistarelli et al. [113] in which they overlaid the verified contracts on the Ether price. This makes sense, as it is more lucrative for developers to deploy a smart contract if the price of Ether plays along, as their own USD value is tied to Ether. It is suspected that many smart contracts are scams, as smart contracts do not place themselves in a bear market, which can endanger the project, but wait for a bull market or at least signs of an uptrend.

That is why it is assumed that if the smart contracts had been examined in the bull market of 2021, the ratio of scams would have been lower. Xia et al. [2] measured about 50% of scams, but they also have a dataset with roughly 20'000 scams.

5.4.2 Recent development

As shown in Chapter 3.2.2, recent news is used to promote scams. Empirically, at the end of February and in March 2022, it was noticed that the Ukraine war was used. The news in May that was used is the huge sell-off of LUNA [114] and the related stable coin UST [115] that could not keep its value at 1 USD. Between May 1, 2022 and May 13, 2022, Luna fell from 80 USD to slightly below $1 * 10^{-7}$ USD. The UST, which should be around 1 USD has fallen to

a little below 0.045 USD. This has also been covered in the mainstream media. Scammers have taken advantage of this and created many LUNA knockoffs. There are 182 out of 2426 entries with LUNA in the name in the database up to June 9, 2022.

5.5 Limitation

This bachelor's thesis only deals with smart contracts that are deployed on Uniswap v2, so only on Ethereum. Thus, only tokens that implement the ERC-20 standard are analysed. For this purpose, only tokens that can be traded against Wrapped Ether, i.e., WETH, are considered at this point.

It is also important to note that backdoors can be built-in by means of vulnerabilities that SCSC cannot detect, so a residual risk remains.

6 Outlook and Conclusion

This chapter looks at how the project could be extended. Then, a conclusion is made.

6.1 Outlook

The SCSC could be further developed in several ways.

The indicators in the project were written in such a way that they could be expanded with minimal effort. Thus, any new indicators designed and developed can be integrated into the system. An example of an extended indicator would be the evaluation of phishing and social dimensions. Since companies or similar are used to advertise the scams, one could search for websites, social media, and whitepapers. Another indicator that could be useful is the duration of locked liquidity. Here, a more precise analysis would have to be made regarding how long the lock is set because a short time on locked liquidity tends to be a scam.

Another direction that could make identification easier is the way of ETH's respective WETH. Serial scammers will accumulate ETH over time. Thus, the assets are reused for the scams. Since the blockchain is open, the transaction trail could be followed. Only a mixer like Tornado Cash [116], i.e., a system that deliberately sends the transactions over the network in order to disguise them at the destination, could make this analysis difficult. How far the mixer as a whole is recognisable and could be included as an indicator would have to be thoroughly investigated. In addition, the behaviour of scammers could be analysed to see if they are recognisable. This would be associated with the recognition of similar smart contracts, such as copy-and-paste smart contracts, which could be intercepted in this way. All this could be used to profile scammers, which will help the evaluation and then identification of scams.

The risks for rug pulls are not limited to the ERC-20 [20] standard. ERC-721 [21], i.e., NFTs, are also affected. A scam analysis would also be useful here to protect investors. A paper introduces the identification of security risks in NFTs. This can be taken further with the ERC-777 [117] standard, which complements the ERC-20 token, and ERC-1155 [118], which brings a multi-token standard.

Currently, only tokens based on the Ethereum platform were viewed on Uniswap v2. The sidechains were not considered. For expanding the coverage of SCSC in that direction, first, any candidate platform would have to be analysed how many smart contracts are deployed there to find out whether a development for the sidechains makes sense or not. An apparent and low-cost extension would be for the Binance Chain. Since the Binance Chain is a fork of Ethereum and therefore EVM compatible, the environment is very similar. PancakeSwap is a direct fork of Uniswap v2, the extension should not require much effort. The attachment to Etherscan can be replaced by BscScan [119], as they were created by the same team as Etherscan.

6.2 Conclusion

The original goal of the project was achieved by being able to score the trustworthiness of a submitted smart contract with an acceptable confidence and classifying them as scam or benign with a corresponding probability. Currently, the SCSC system is deployed as a web-based service and is constantly monitoring the Ethereum blockchain to collect information about every new Uniswap V2 pair and calculating its scam probability. Additionally, the system offers a way to submit smart contract addresses manually and if they are represented on the Uniswap V2 DEX, the calculated scam probabilities as well as details about the smart contract are returned in an intelligible manner.

Finally, the vulnerabilities contained in the scanned smart contract are made visible on the application, allowing the user to add his own conclusion to the automatically evaluated score.

In the case of a user wanting to rescan a result provided by the automatic monitoring, they can resubmit the token address and will receive a live updated result about the state of the token. Nevertheless, getting this information is currently only implemented as a manual operation and, as discussed in Chapter 5, is a possible improvement to the existing system.

7 Bibliography

7.1 Table of Abbreviations

Abbreviation	Explanation
AMM	Automated Market Maker
BTC	Bitcoin
CEX	Centralized exchange
DeFi	Decentralized finance
DEX	Decentralized exchange
EIP	Ethereum Improvement Proposal
ERC	Ethereum Request for Comments
ERC-20	A standard that defines smart contracts fungible tokens on Ethereum
ERC-721	A standard that defines smart contracts for non-fungible tokens on Ethereum
ETH	Ether, the currency on the Ethereum platform
EVM	Ethereum Virtual Machine
Gas	Usage to define
Gwei	Giga Wei, i.e., $10^9\ Wei$ or $10^{-9}\ ETH$
KYC	Know your customer
LP	Liquidity Provider
NFT	Non fungible tokens
P&D	Pump and Dump
SCSC	Acronym of this bachelor's thesis: Smart Contract Scam Checker for Decentralized Finance
SWC	Smart Contract Weakness Classification and Test Cases
Tx	Transaction
USDC	US-Dollar as a stable coin, which is calculated algorithmically
USDT	US-Dollar as a stable coin, backed by a company
Wei	The smallest unit possible at ETH, which is $10^{-18}\ ETH$.
WETH	Warped Ether
xCHF	Swiss Franc as a stable coin

7.2 List of figures

7.3 List of equations

7.4 List of codes

7.5 List of tables

7.6 List of sources

[1] P. Ackermann and S. Tassone, ‘Projektarbeit: Solidity Vulnerability Scanner’. Zurich University of Applied Sciences, Dec. 24, 2021.

[2] P. Xia *et al.*, ‘Trade or Trick? Detecting and Characterizing Scam Tokens on Uniswap Decentralized Exchange’, *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 5, no. 3, p. 39:1-39:26, Dezember 2021, doi: 10.1145/3491051.

[3] ‘Top DEX exchanges by Trading Volume’, *CoinGecko*. https://www.coingecko.com/en/dex (accessed Apr. 06, 2022).

[4] B. Mazorra, V. Adan, and V. Daza, ‘Do Not Rug on Me: Leveraging Machine Learning Techniques for Automated Scam Detection’, *Mathematics*, vol. 10, no. 6, Art. no. 6, Jan. 2022, doi: 10.3390/math10060949.

[5] N. El Ioini and C. Pahl, ‘A Review of Distributed Ledger Technologies’, in *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, Cham, 2018, pp. 277–288. doi: 10.1007/978-3-030-02671-4_16.

[6] S. Nakamoto, ‘Bitcoin: A Peer-to-Peer Electronic Cash System’, p. 9.

[7] V.-A. rund um VeChain, ‘Was ist Blockchain?’, *Medium*, May 10, 2019. https://vemag.medium.com/was-ist-blockchain-d128c770741b (accessed Oct. 28, 2021).

[8] V. Buterin, ‘Ethereum: A Next-Generation Smart Contract and Decentralized Application Platform.’ [Online]. Available: https://ethereum.org/669c9e2e2027310b6b3cdce6e1c52962/Ethereum_Whitepaper_-_Buterin_2014.pdf

[9] ‘Global Cryptocurrency Market Charts’, *CoinMarketCap*. https://coinmarketcap.com/charts/ (accessed May 11, 2022).

[10] ‘Ethereum ETH Netzwerk Hashrate Chart - 2Miners’. https://2miners.com/de/eth-network-hashrate (accessed May 11, 2022).

[11] ‘ethash’, *Ethereum Wiki*. https://eth.wiki/en/concepts/ethash/ethash (accessed Oct. 26, 2021).

[12] ‘51% Attack | Ethereum Glossary’, *ethereum.org*. https://ethereum.org (accessed Jun. 01, 2022).

[13] ‘Ethereum Glossary: Wei’, *ethereum.org*. https://ethereum.org/en/glossary/#wei (accessed Oct. 17, 2021).

[14] ‘What are Tether tokens and how do they work?’ https://tether.to/en/how-it-works (accessed May 11, 2022).

[15] ‘Ethereum (ETH) Mainnet - GasNow - etherchain.org - 2022’, *etherchain.org*. https://etherchain.org/tools/gasnow (accessed Apr. 23, 2022).

[16] A. M. Antonopoulos and D. G. Wood, *Mastering Ethereum*, 1. O’Reilly UK Ltd., 2021. Accessed: Oct. 31, 2021. [Online]. Available: https://github.com/ethereumbook/ethereumbook/blob/c5ddebd3dbec804463c86d0ae2de9f28fbafb83a/images/evm-architecture.png

[17] ‘Ethereum Virtual Machine (EVM)’, *ethereum.org*. https://ethereum.org (accessed Oct. 17, 2021).

[18] ‘Ethereum Improvement Proposals (EIPs)’, *ethereum.org*. https://ethereum.org (accessed May 12, 2022).

[19] ‘Compatible Competition - Empowering or Encroaching on Ethereum?’, *Bitcoin Suisse*, Oct. 12, 2021. https://www.bitcoinsuisse.com/research/decrypt/compatible-competition-empowering-or-encroaching-on-ethereum (accessed Apr. 26, 2022).

[20] ‘EIP-20: Token Standard’, *Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-20 (accessed May 23, 2022).

[21] ‘EIP-721: Non-Fungible Token Standard’, *Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-721 (accessed May 23, 2022).

[22] ‘ERC-20 Token Standard’, *ethereum.org*. https://ethereum.org/en/developers/docs/standards/tokens/erc-20 (accessed Oct. 27, 2021).

[23] ‘Tokens - OpenZeppelin Docs’. https://docs.openzeppelin.com/contracts/3.x/tokens (accessed Oct. 28, 2021).

[24] OpenSea, ‘CryptoPunk #9998 - CryptoPunks’, *OpenSea*. https://opensea.io/assets/0xb47e3cd837ddf8e4c57f05d70ab865de6e193bbb/9998 (accessed Dec. 18, 2021).

[25] S. Goldfeder, J. Bonneau, E. W. Felten, J. A. Kroll, and A. Narayanan, ‘Securing Bitcoin wallets via threshold signatures’, p. 11.

[26] ‘Where are my coins?’, *Ledger*. https://www.ledger.com/academy/crypto/where-are-my-coins (accessed May 11, 2022).

[27] ‘A crypto wallet & gateway to blockchain apps | MetaMask’. https://metamask.io/ (accessed Mar. 07, 2022).

[28] ‘Coinbase Wallet - Your key to the world of crypto’. https://www.coinbase.com/de/wallet (accessed Mar. 07, 2022).

[29] ‘Ledger Hardware Wallet - State-of-the-art security for crypto assets’, *Ledger*. https://www.ledger.com (accessed Mar. 07, 2022).

[30] 'Trezor Hardware Wallet | The original & most secure bitcoin wallet.' https://trezor.io/ (accessed Mar. 07, 2022).

[31] 'Ledger Docs: FIDO U2F', *Ledger Support*. https://support.ledger.com/hc/en-us/articles/115005198545-FIDO-U2F (accessed Mar. 07, 2022).

[32] 'Decentralized finance (DeFi)', *ethereum.org*. https://ethereum.org (accessed Jun. 01, 2022).

[33] 'Spot Archives | The Block Crypto (The Block Research)', *The Block*. https://www.theblockcrypto.com/data/crypto-markets/spot (accessed Apr. 05, 2022).

[34] 'Mt. Gox | CoinMarketCap', *CoinMarketCap Alexandria*. https://coinmarketcap.com/alexandria/glossary/mt-gox (accessed Mar. 01, 2022).

[35] S. Somraaj, 'Answer to "Is a $200 'network fee' normal for trades on uniswap?"', *Ethereum Stack Exchange*, Nov. 03, 2021. https://ethereum.stackexchange.com/a/112793 (accessed Jun. 02, 2022).

[36] 'Uniswap Interface'. https://app.uniswap.org/#/swap?chain=arbitrum&lng=en-US (accessed Mar. 09, 2022).

[37] 'Pancake Swap Docs'. https://docs.pancakeswap.finance/code/smart-contracts/pancakeswap-exchange/router-v2 (accessed Mar. 09, 2022).

[38] 'Serum Portal'. https://portal.projectserum.com/ (accessed May 11, 2022).

[39] 'DODO Home'. https://dodoex.io/ (accessed May 11, 2022).

[40] 'Decentralized Exchange (DEX) Trading Volume Charts | The Block Crypto (The Block Research)'. https://www.theblockcrypto.com/data/decentralized-finance/dex-non-custodial (accessed Apr. 02, 2022).

[41] 'EtherDelta'. https://etherdelta.com/ (accessed Apr. 02, 2022).

[42] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson, 'Uniswap Whitepaper v3', *Uniswap Whitepaper v3*. https://uniswap.org/whitepaper-v3.pdf

[43] 'What Is Uniswap? | Uniswap'. https://docs.uniswap.org//protocol/introduction (accessed Mar. 11, 2022).

[44] 'Our US Partner, Binance.US, Opens for Registration and Deposits | Binance'. https://www.binance.com/en/support/announcement/360033855351 (accessed Mar. 11, 2022).

[45] 'Updates to Daily Withdrawal Limits | Binance Support'. https://www.binance.com/en/support/announcement/9df8225c061b455da5c7cc293cd08a70 (accessed Mar. 14, 2022).

[46] ‘A short history of Uniswap’, *Uniswap Protocol*, Feb. 11, 2019. https://uniswap.org/blog/uniswap-history (accessed Mar. 14, 2022).

[47] ‘The Uniswap V1 Protocol | Uniswap v1’. https://docs.uniswap.org//protocol/V1/introduction (accessed Mar. 14, 2022).

[48] etherscan.io, ‘Uniswap v2 Smart Contract | Etherscan’, *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0xc31d7e7e85cab1d38ce1b8ac17e821ccd47dbde00f9d57f2bd8613bff9428396 (accessed Mar. 14, 2022).

[49] ‘Pools | Uniswap v2’. https://docs.uniswap.org//protocol/V2/concepts/core-concepts/pools (accessed Mar. 15, 2022).

[50] ‘Flash Swaps | Uniswap v2’. https://docs.uniswap.org//protocol/V2/guides/smart-contract-integration/using-flash-swaps (accessed Mar. 15, 2022).

[51] ‘SDK | Uniswap v2’. https://docs.uniswap.org//sdk/2.0.0/reference/getting-started (accessed Mar. 15, 2022).

[52] ‘Uniswap V3 Mainnet Launch!’, *Uniswap Protocol*, May 05, 2021. https://uniswap.org/blog/launch-uniswap-v3 (accessed Mar. 14, 2022).

[53] ‘Decentralized Finance Exploits | The Block Crypto (The Block Research)’. https://www.theblockcrypto.com/data/decentralized-finance/exploits (accessed Apr. 02, 2022).

[54] J. Mapperson, ‘How many DeFi projects still have “God Mode” admin keys? More than you think’, *Cointelegraph*. https://cointelegraph.com/news/how-many-defi-projects-still-have-god-mode-admin-keys-more-than-you-think (accessed Apr. 04, 2022).

[55] ‘SWC Registry | Smart Contract Weakness Classification and Test Cases’. http://swcregistry.io/ (accessed Apr. 03, 2022).

[56] ‘Binance charged Blockstack $250,000 prior to listing Stacks’, *The Block*. https://www.theblockcrypto.com/post/44837/binance-accepted-250000-to-list-blockstacks-token-despite-claiming-there-was-no-listing-fee (accessed Mar. 29, 2022).

[57] ‘Rug Pull | CoinMarketCap’, *CoinMarketCap Alexandria*. https://coinmarketcap.com/alexandria/glossary/rug-pull (accessed Mar. 28, 2022).

[58] ‘Mining’, *ethereum.org*. https://ethereum.org (accessed Apr. 20, 2022).

[59] ‘Uniswap Interface’. https://app.uniswap.org/#/swap?chain=mainnet (accessed Apr. 20, 2022).

[60] T. Li, D. Shin, and B. Wang, ‘Cryptocurrency Pump-and-Dump Schemes’, Social

Science Research Network, Rochester, NY, SSRN Scholarly Paper ID 3267041, Feb. 2021. doi: 10.2139/ssrn.3267041.

[61] 'Dogecoin price today, DOGE to USD live, marketcap and chart', *CoinMarketCap*. https://coinmarketcap.com/currencies/dogecoin/ (accessed Mar. 28, 2022).

[62] 'Cryptocurrency Prices, Charts, and Crypto Market Cap', *CoinGecko*. https://www.coingecko.com/ (accessed Apr. 04, 2022).

[63] 'FastAPI'. https://fastapi.tiangolo.com/ (accessed May 17, 2022).

[64] 'Minimal Free – Client & Admin Dashboard', *MUI Store*. https://mui.com/store/items/minimal-dashboard-free/ (accessed May 17, 2022).

[65] 'Ethereum API | IPFS API & Gateway | ETH Nodes as a Service', *Infura*. https://infura.io/ (accessed May 18, 2022).

[66] 'DEXTools.io'. https://www.dextools.io/ (accessed Apr. 09, 2022).

[67] 'Live New Pairs - DEXTools'. https://www.dextools.io/app/ether/pool-explorer (accessed Apr. 09, 2022).

[68] etherscan.io, 'Zilla (Zilla) Token Tracker | Etherscan', *Ethereum (ETH) Blockchain Explorer*. https://etherscan.io/token/0x8f1e8f0942f44e87a03ccf402fab7f6bd13814c9 (accessed Apr. 10, 2022).

[69] smart-contract-check, *Common Scam Analysis | smart-contract-check/smart-contract-check*. 2022. Accessed: Jun. 10, 2022. [Online]. Available: https://github.com/smart-contract-check/smart-contract-check/blob/00a8b5346ddc743f2906910b68daaf35d400e141/common_scam_analysis/Common%20Scam%20Analysis.xlsx

[70] etherscan.io, 'Zilla (Zilla) Token Tracker Read Contract | Etherscan', *Ethereum (ETH) Blockchain Explorer*. https://etherscan.io/token/0x8f1e8f0942f44e87a03ccf402fab7f6bd13814c9#readContract (accessed Apr. 10, 2022).

[71] 'Zilla $0.00001471 - Pair Explorer - DEXTools'. https://www.dextools.io/app/ether/pair-explorer/0xeaeb44b2d874fbfc9487913eb96ffc3edc20a3a9 (accessed Apr. 11, 2022).

[72] 'Ethereum | Honeypot Detector for ETH'. https://www.honeypot.is/ethereum (accessed Apr. 11, 2022).

[73] 'Zilla | Honeypot Detector for ETH'. https://www.honeypot.is/ethereum?address=0x8F1e8F0942f44E87A03cCf402fAb7f6bD13814C9 (accessed Apr. 11, 2022).

[74] etherscan.io, 'Zilla Similar Contracts Search | Etherscan', *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/find-similar-contracts (accessed Apr. 15, 2022).

[75] etherscan.io, 'AAVEE Ethereum Transaction Hash (Txhash) Details | Etherscan', *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0x1082865a74cf864926bd54c461a20b07687b748cda7bc177df9efa08be01b52c (accessed Apr. 15, 2022).

[76] 'Donate Uniswap'. https://donate.uniswap.org/#/swap (accessed Apr. 09, 2022).

[77] etherscan.io, 'Ukraine DAO | Etherscan', *Ethereum (ETH) Blockchain Explorer*. https://etherscan.io/token/0x4504BFbF4ae479f179453db2F5b65aBB9CcD5502 (accessed Apr. 09, 2022).

[78] etherscan.io, 'Add Liquidity Ukraine DAO | Etherscan', *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0xb171e3ee064797c4c84049becf3f6991db8aa2e962d63c2b22b26471fad9f643 (accessed Apr. 10, 2022).

[79] etherscan.io, 'Remove Liquidity Ukraine DAO | Etherscan', *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0xe6c7160cb4a4eb877af067e5b0a447a192e27d37d45cdaf4da1f654552de28f3 (accessed Apr. 10, 2022).

[80] 'Arbitrage and Frontrunning in DeFi', *Bitcoin Suisse*, Mar. 16, 2021. https://www.bitcoinsuisse.com/research/decrypt/arbitrage-and-frontrunning-in-defi (accessed Apr. 20, 2022).

[81] etherscan.io, 'Decentraland (MANA) Token Tracker | Etherscan', *Ethereum (ETH) Blockchain Explorer*. https://etherscan.io/token/0x0f5d2fb29fb7d3cfee444a200298f468908cc942 (accessed Apr. 20, 2022).

[82] etherscan.io, 'Altered State Machine Utility Token (ASTO) Token Tracker | Etherscan', *Ethereum (ETH) Blockchain Explorer*. https://etherscan.io/token/0x823556202e86763853b40e9cde725f412e294689 (accessed Apr. 21, 2022).

[83] 'ASTO - Pair Explorer - DEXTools'. https://www.dextools.io/app/ether/pair-explorer/0x11181bd3baf5ce2a478e98361985d42625de35d1 (accessed Apr. 08, 2022).

[84] P. Züst, 'Analyzing and Preventing Sandwich Attacks in Ethereum', ETH Zürich, 2021. [Online]. Available: https://pub.tik.ee.ethz.ch/students/2021-FS/BA-2021-07.pdf

[85] etherscan.io, 'Internal Transaction ASTO | Etherscan', *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0xcd2900e75a736a97499222d620aee7c8dadd4eb26f795670b367da98c5e3e368 (accessed Apr. 21, 2022).

[86] 'MEV Explore'. https://explore.flashbots.net/ (accessed Apr. 22, 2022).

[87] ‘Status - Ethereum Smart Contract Best Practices’. https://consensys.github.io/smart-contract-best-practices/development-recommendations/documentation/status/ (accessed May 28, 2022).

[88] etherscan.io, ‘Similar Contracts Search | Etherscan’, *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/find-similar-contracts (accessed Apr. 22, 2022).

[89] ‘Rekt - Ronin Network’, *rekt*. https://www.rekt.news/ (accessed Apr. 26, 2022).

[90] ‘Rekt - Leaderboard’, *rekt*. https://www.rekt.news/ (accessed Apr. 12, 2022).

[91] ‘Unicrypt • Multi chain decentralized protocols and services’, *Unicrypt Network*. https://unicrypt.network/ (accessed Apr. 22, 2022).

[92] ‘Token and Liquidity Locking | Team by TrustSwap’, *Team | Token Locking Made Easy*. https://team.finance/ (accessed Apr. 22, 2022).

[93] etherscan.io, ‘Unicrypt : Liquidity Lockers | Address 0x663a5c229c09b049e36dcc11a9b0d4a8eb9db214 | Etherscan’, *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/address/0x663a5c229c09b049e36dcc11a9b0d4a8eb9db214 (accessed Apr. 22, 2022).

[94] etherscan.io, ‘Unicrypt Example Transaction | Etherscan’, *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0x0912db17de195dcd3644c73c3d7508795611523ebac05840e0e2f08c14300b02 (accessed Apr. 22, 2022).

[95] GenTokens, *Does ‘Renounced Ownership’ make a contract safe?*, (Jul. 06, 2021). Accessed: Apr. 16, 2022. [Online Video]. Available: https://www.youtube.com/watch?v=AmNo_1UJfkk

[96] ‘Aave - Open Source Liquidity Protocol’. https://aave.com/ (accessed Apr. 17, 2022).

[97] ‘Upgrading smart contracts - OpenZeppelin Docs’. https://docs.openzeppelin.com/learn/upgrading-smart-contracts (accessed Apr. 16, 2022).

[98] ‘Ethereum explorer | Ethplorer’. https://ethplorer.io (accessed Apr. 15, 2022).

[99] CZ Binance [@cz_binance], ‘Scammers impersonating Binance | Twitter’, *Twitter*, Oct. 12, 2019. https://twitter.com/cz_binance/status/1182851509119139840 (accessed Apr. 16, 2022).

[100] W. Zhao, ‘$600 Fraud? Fake ICO White Papers Are Drawing Scrutiny’, Feb. 13, 2018. https://www.coindesk.com/markets/2018/02/13/600-fraud-fake-ico-white-papers-are-drawing-scrutiny/ (accessed Apr. 16, 2022).

[101] ‘All Datasets | smart-contract-check/smart-contract-check’, *GitHub*.

https://github.com/smart-contract-check/smart-contract-check (accessed Jun. 10, 2022).

[102] 'Access Control - OpenZeppelin Docs'. https://docs.openzeppelin.com/contracts/4.x/api/access (accessed May 29, 2022).

[103] P. Xia, H. Wang, B. Gao, W. Su, and Z. Yu, 'Trade or Trick? Dataset | GitHub'. https://raw.githubusercontent.com/T2Project/RugPullDetection/main/data/tokens.csv (accessed Jun. 09, 2022).

[104] 'Nintendo Official Site: Consoles, Games, News, and More'. https://www.nintendo.com/ (accessed May 22, 2022).

[105] 'Home | Aardman'. https://aardman.com// (accessed May 22, 2022).

[106] 'Shibana Coin', *Shibana Coin*. https://www.shibana.io (accessed May 22, 2022).

[107] 'BoK Token and Liquidity Locking | Team by TrustSwap', *Team | Token Locking Made Easy*. https://team.finance/ (accessed May 23, 2022).

[108] etherscan.io, 'BOK Sell Transaction Liquidity Pool despite Liquidity Lock | Etherscan', *Ethereum (ETH) Blockchain Explorer*. http://etherscan.io/tx/0x86ee2d47e1a84b44260a5ebaf3a91945ce2f3103657f6c5b40272b269dbd55c2 (accessed May 23, 2022).

[109] 'Contract ABI Specification — Solidity 0.8.14 documentation'. https://docs.soliditylang.org/en/v0.8.14/abi-spec.html (accessed May 26, 2022).

[110] '1.4. Support Vector Machines', *scikit-learn*. https://scikit-learn/stable/modules/svm.html (accessed Jun. 09, 2022).

[111] *eXtreme Gradient Boosting | GitHub*. Distributed (Deep) Machine Learning Community, 2022. Accessed: Jun. 09, 2022. [Online]. Available: https://github.com/dmlc/xgboost

[112] 'JSON-RPC API', *ethereum.org*. https://ethereum.org (accessed May 28, 2022).

[113] S. Bistarelli, G. Mazzante, M. Micheletti, L. Mostarda, and F. Tiezzi, 'Analysis of Ethereum Smart Contracts and Opcodes', DOI: 10.1007/978-3-030-15032-7_46, Jan. 2020. doi: 10.1007/978-3-030-15032-7_46.

[114] 'Terra price, LUNA chart, and market cap', *CoinGecko*. https://www.coingecko.com/en/coins/terra-luna (accessed May 20, 2022).

[115] 'TerraUSD price, UST chart, and market cap', *CoinGecko*. https://www.coingecko.com/en/coins/terra-usd (accessed May 20, 2022).

[116] 'Tornado.Cash'. https://tornado.cash (accessed Jun. 04, 2022).

[117] 'EIP-777: Token Standard', *Ethereum Improvement Proposals*.

https://eips.ethereum.org/EIPS/eip-777 (accessed May 23, 2022).

[118] ‘EIP-1155: Multi Token Standard’, *Ethereum Improvement Proposals*. https://eips.ethereum.org/EIPS/eip-1155 (accessed May 23, 2022).

[119] BscScan.com, ‘Binance (BNB) Blockchain Explorer’, *Binance (BNB) Blockchain Explorer*. http://bscscan.com/ (accessed May 23, 2022).

8 Appendices

8.1 Project agreement

The original project agreement was written in a mix of German and English. The original project agreement can be found under the following link:

https://tat.zhaw.ch/tpada/arbeit_vorschau.jsp?arbeitID=18530

Bachelorarbeit 2022 - FS: BA22_gueu_03

Allgemeines:

Titel:	Smart Contract Scam Checker for Decentralized Finance (SCSC)
Anzahl Studierende:	2
Durchführung in Englisch möglich:	Ja, die Arbeit kann vollständig in Englisch durchgeführt werden und ist auch für Incomings geeignet.

Betreuer:

HauptbetreuerIn: Gürkan Gür, gueu 

Zugeteilte Studenten:

Diese Arbeit ist vereinbart mit:
- Pascal Ackermann, ackerpas (IT)
- Stefano Tassone, tassoste (IT)

Fachgebiet:

IS Information Security

Studiengänge:

IT Informatik

Zuordnung der Arbeit :

InIT Institut für angewandte Informationstechnologie

Infrastruktur:

benötigt keinen zugeteilten Arbeitsplatz an der ZHAW

Interne Partner :

Es wurde kein interner Partner definiert!

Industriepartner:

Es wurden keine Industriepartner definiert!

Beschreibung:

Smart Contract Scam Checker for Decentralized Finance (SCSC) will be developed with Python and React.js, based on Slither which is also based on Python.

Requirements

You will work out the exact scope of functions and the user interface yourself in the course of your work. The project meetings, final report and presentation will be in English.

The following requirements apply as a current basis, although you will certainly add to or adapt these based on your findings during the work:

- The core of the work consists of Slither, that will be used for detecting vulnerabilities in open-source code on the Blockchain using the Etherscan API.
- The user receives helpful information on the vulnerabilities and scam risk level found. The results of the analysis are presented to the user in text and visual format on the front end.
- The written part of the work compares the literature and the security vulnerabilities found, and then draws a conclusion.

Tasks

The following points are to be dealt with in the work:

- Investigate existing security vulnerabilities that are visible from the outside with or without code.
- Integrate external intelligence data feeds and metrics for evaluation of the contracts for security vulnerabilities.

8.2 Meeting Protocol

Kick-off Meeting – Online – 18.02.2022

- Organisation of dates and meetings
- Revisit the goals of the work

Week 01 – Online – 24.02.2022

- Literature Resources
  - Uniswap
- Document Structure
- Case Study
  - First manual
  - Then automatic
- Showing Prepared UI Prototype

Week 02 – Online – 02.03.2022

- Create Meeting Summary
- Architecture Update
  - Define backend
  - ELK was not suitable
  - MongoDB
  - GUEU agreed to use it.
- ZHAW Server
  - We need a centralised machine in ZHAW Cloud for centralized testing/deployment
- Literature Update
- Started writing the theoretical part
  - Uniswap

GUEU:

- VM to Deploy Debian (10.03.2022)

Next Steps:

- Manually perform case study of a “rug pull” on the Ethereum blockchain and note down potential visible indicators.

Week 03 - Online– 10.03.2022

- On-site meeting 24.03.2022 → with Review of the document, which we will hand in.

Results:

- Research and BA text
- Case Study of recent big rug pull
    - Still need to write it down scientifically
- VM not here but not a Blocker

Next Steps:

- VM for next week
- Up to now we did Top-Down research, now we start with Bottom up by implementing the foundational API queries (Etherscan, web3, Uniswap API) and database connection.
- Write down the documentation on the case-study in the BA with some scientific facelift.
- Revising the text and rephrasing and restructure it.

GUEU:

- VM to Deploy Debian / Write the IT Department for it
- No Blockers

Week 04 – Asynchronous

Results:

- Theoretical Foundation to Uniswap is almost done.
- We can communicate with Uniswap v2 for new Smart Contracts. The monitoring works, but without persistence.
- Preparations for getting Smart Contracts by polling Infura with a limit of 100 ’000 Requests per day.
- We created a time plan on which we will try to stick as much as it makes sense and update week by week. The plan can be found in the Teams chat and later in the repo of our BA.

Next Steps:

- Finish Uniswap Theoretical Foundations and write Threat Landscape.
- Bind in Etherscan
- Persistence

GUEU:

- VM was delivered 16.03.22, the students can access it.
- Reminder next week we will meet physically if the circumstances allow it.
- We would hand in a text on Tuesday, which you could go through if it enough time for you.
- No Blocker

Week 05 – Online –24.04.2022

- On-site, meeting was not possible
- Showing current program
- Persistence with document-based with MongoDB is working
- Etherscan is not 100% done but should be done by the end of the week
- Indicators might be the most difficult task -> but we are on track
- Theoretical Foundations
- Next week on-site meeting, in GUEU

Week 06 – On-site - 31.03.2022

- New Case study for Locked Ether
- Mostly done with Theoretical Foundation
- Code Indicator
    - Liquidity Amount
    - Liquidity Share and Holders

Next

- Honeypot is and verified smart contract
- VM we need to figure out how to do it
- Methodology
- No Blocker

Week 07 – Online – 07.04.2022

- Delay on the text, we try to compensate it this week, if possible, due to illness
- Theoretical Foundation done, minor language changes
- Worker done
- Code Indicator
    - Verified Contracts → able to get the Source Code, in preparation on Slither that will be an indicator of the project

Next:

- Case Study URGENT! This week
- Dataset Organisation
- Coding Indicators, evaluate first the order of them

Week 08 – On-site – 14.04.2022

- Sandwich Attack / Front Running, new findings, and discussion with ex ETH Student
    - We we've meet and now we need to write it down
- Methodology mostly done
    - Code
    - 2 Indicators are done:
        - Slither with Verified Contract
        - Honeypot
- VM is collecting data
- Date for presentation
    - Preferably 31.06.22 or 01.07.22

Next:

- Experts
    - We could look for someone if we wanted to.
- Code
    - Efficient indicators first
- No suitable Datasets found to verify the indicators

Week 09 – Online- 21.04.2022

- Text
    - Added Sandwich attack in Theoretical Foundations
    - Methodology still not done since there were more Case studies
    - Questions about the Structure
    - We can have duplicate subchapters if the Chapters have clear names
- Code
    - Ownership, WIP
- VM is still collecting data on what we can rely on later

Next:

- Trying to get to an end on the first 3 Chapters
- Add more indicators
- Start Frontend to start evaluating the results

Week 10 – On-site -28.04.2022

- Research
    - Dataset
        - Not Found, we did not find any usable dataset
        - We will do our own dataset
    - Predict the Future
    - CSV → export will be published the
- Talk about presentation
    - Data Collection for the Presentations
    - Publishing until Presentation
- Text
    - Cleaned up
- Code
    - Ownership done

Next:

- Gueu will
- Monitor
- Next Week monitoring
- A Week later Dataset evaluation

Week 11 – On-site – 05.05.2022

- Discussed review of section 1 to 3 of the thesis document
- Slight restructuring of sentences and edits to some screenshots/images
- Add date to dynamic statements
- Shift general scope more towards DeFi
    - Rename the work to “Smart Contract Scam Checker for Decentralized Finance (SCSC)”
    - Refactor Case study
- Avoid dark mode

Code

- Get Contract creation time
- Implement Slither (maybe external?)
- Implement retry rule
- Just throw into queue again and mark the time

Week 12 – online – 15.05.2022

- Data
    - Since Crypto did a huge hit there are not as much Smart Contracts deployed on Uniswap
    - Try to migrate this weekend
- Frontend postponed
- Text
    - applied the review changes
    - Buy domain smart-contract-check.com
- Code
    - Try to handle all exception with clean logging

Next Steps:

- Clean-up Data
- Evaluators enhance the Accuracy
- Evaluation discussion
- Text
    - Finish in 2 Weeks if possible
- Frontend

Week 13 – On-site – 12.05.2022

- Architecture and Diagrams
- Dataset
    - “Light Dataset” we should do that so have 50 of 150 to analyse
- Evaluation
    - Walk-trough the code
    - Showed configuration
- Visualizing Results
    - Venn
    - Confusion matrix

Week 14 – Online - 24.05.2022

- Tables from Evaluating Indicators should we move them?
    - Take each indicator, talk about the result of that and at the end all together
- Structure in general is clear and written
- Abbreviations are written

Week 14 – Online - 27.05.2022

- Dataset is enhanced with around 170 manual annotations

- Text is not ready to hand out right now, we will hand it in as soon as we are ready

Week 15 – Online - 31.05.2022

- Try to find correlations between the charts
- Try to wrap up
- Dataset is closed
- Try to end it in the next few days

8.3 Time management

Week	Meeting	Research	Implementation	Documentation	Comments
Kick-off	18.02.22	Collect first Sources		Project agreement	
SW1	24.02.22	Initial paper research		Document skeleton	
SW2	02.03.2022	Manual scam analysis, reasearch into Uniswap		Document structure	
SW3	10.03.22	Case Study, Threat Landscape	Architecture Definition	Theoretical Foundation: General	
SW4	17.03.22	Case Study	Monitoring Uniswap for new pairs	Theoretical Foundation: DEX / Uniswap	
SW5	24.03.22	Identify Scam Indicators	Interface with Etherscan and DB	Theoretical Foundation: Threat Landscape	Important to get the necessary data from Etherscan.
SW6	31.03.22	Reserach into locked ether possibilities	Development of liquidity amount, liquidity share holder and token share holder indicators.	Methodology: Case Study Rug Pull	Milestone Meeting Physical with Document review. Access to the ZHAW VM received.
SW7	07.04.22	Searching Dataset for Scam Tokens	Development of verified contract indicator. Worker implementation done.	Methodology: Indicators	If no dataset is found, a decision must be made if we collect it ourselves. Delay in documentation of methodology.
SW8	14.04.22	Discussion with Ex-ETH student regarding discrepancies in sandwich attack.	Development of honeypot and Slither indicator.	Methodology: Indicators	No suitable datased found.
SW9	21.04.22		Development of ownership indicator. Finalizing Slither indicator.	Methodology: Indicators	Collect and label our own dataset based on collected data.
SW10	28.04.22	Evaluate Indicator and compare it with other's research.	Improving Slither indicator by using its check instead of the delivered impact and confidence	Methodology: Evaluation	Ensure that the indicators work with the collected dataset.
SW11	05.05.22	Buffer	Implement custom classifier based on weighted average	Refactoring case study	Collect and label a test dataset to verify the accuracy of the training model

SW12	12.05.22		Implement SVM classifier with the indicator scores as features. Jupyter Notebook to create plots and analyse the indicator characteristics	Results: Dataset Results: Indicator characteristics	
SW13	19.05.22	Compare SCSC results to other's research	Implement XGBoost calssifier wiht indicator scores as features. Jupyter Notebook to create plots and analyse overall SCSC score	Results: Indicator characteristics Results: SCSC Results	Identify correlations between the different indicator plots.
SW14	25.05.22	Done		Conclusion Outlook Results: SCSC Results	
PVW1	02.06.22			Abstact Zusammenfassung	
End	10.06.22			Review and final Grammar check	