Smart Contracts Exercise 06:
Fool the Oracle

1 Introduction

Oracles are essential components in decentralized applications that require external data. In this exercise, you will become familiar with both synchronous and asynchronous types of oracles, learn about the concept of decentralized exchanges, and understand their use as on-chain oracles. Finally, you will practically implement a price oracle manipulation attack using a flash loan. This will be by far the most challenging exercise in the series, with many new concepts introduced. So buckle up!

Project Setup

You have two options for working with this exercise: using Docker container or local installation. Choose the one that best fits your preferences.

1.1 Using Docker with VS Code

This option uses Docker to create a development environment with all the necessary tools and dependencies pre-installed.

Prerequisites:

- **Docker** - A platform for developing, shipping, and running applications in containers.
- **Visual Studio Code** - A lightweight but powerful source code editor.
- **Dev Containers** - An extension to VS Code that lets you use a Docker container as a full-featured development environment.

Setting Up the Project:

1. Visit the following GitLab repository and clone it to your local machine.
2. Open the repository folder in VS Code.
3. When prompted, click "Reopen in Container" or use the command palette (F1) and run `Dev Containers: Reopen in Container`.

1.2 Local Setup

If you prefer working directly on your machine without Docker, you can set up the development environment locally.

Prerequisites

- **Node.js**: `https://nodejs.org/en/` - An open-source, cross-platform, back-end JavaScript runtime environment that runs on the V8 engine and executes JavaScript code outside a web browser.
- **NPM**: Node Package Manager, which comes with Node.js.

Open your terminal and run the following commands to verify the installations:

```
$ node -v
$ npm -v
```

Both commands should return the installed version numbers of Node.js and NPM respectively. Node.js provides the runtime environment required to execute JavaScript-based tools like Hardhat, while NPM is used to manage the packages and dependencies needed for development.

Setting Up the Project

1. Visit the following GitLab repository and clone it to your local machine.
2. Open a terminal and navigate to the project directory.
3. Install the project dependencies by running `npm install`.

2 Oracles

Smart contracts cannot access data outside the blockchain on their own. They lack HTTP or similar network methods to access external sources. This is intentional to prevent non-deterministic behavior once a function is called. However, various DApp applications often need external information, such as lending platforms, insurance contracts, betting contracts, wrapped cryptocurrencies, synthetics, and others. Whenever a smart contract relies on external data to compute future states, an oracle pattern is required. The disadvantages of oracles include the cost in terms of gas consumption and the risk of dependence on third parties in terms of data manipulation and data availability.

Synchronous Oracles

In our example, we illustrate a situation where Alice wants to borrow USDC tokens using ETH as collateral. The problem is that the ETH/USDC price is volatile, and we need to know the current price. In a synchronous oracle, external data is periodically pushed into the oracle contract from an oracle controller. The smart contract that needs this external data—in our example, the lending platform—simply trusts that the data in the oracle smart contract is genuine and updated regularly. Look closely at the illustration in Figure 1.

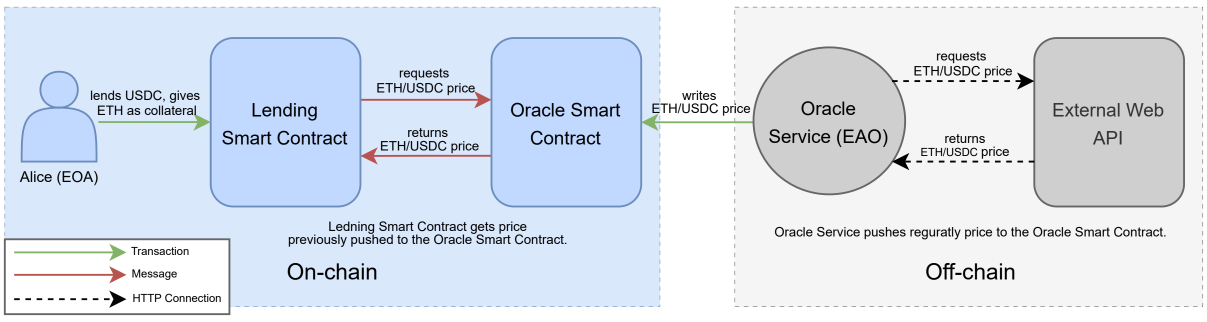


Figure 1: Synchronous Oracles

Asynchronous Oracles

An asynchronous oracle provides data in a separate transaction after an initial request. A smart contract requests data from the oracle, which then emits an event. The oracle service fetches the data from an API or another off-chain source and subsequently sends the data back to the contract in a separate transaction.

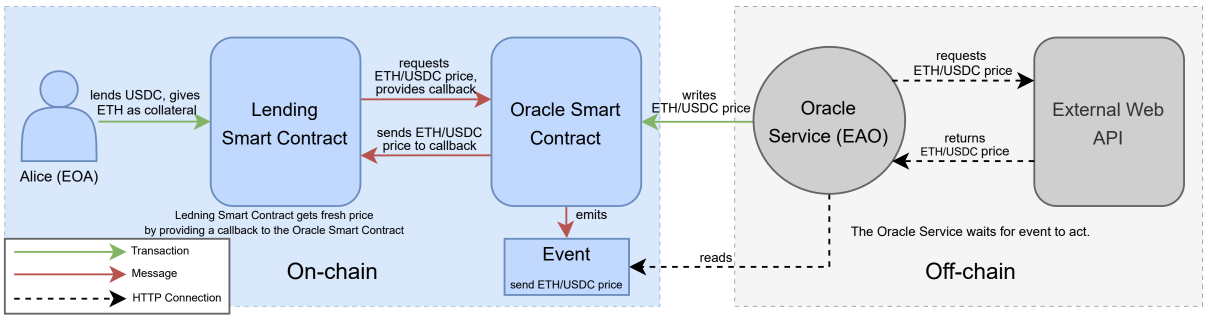

Alice (EOA)
lends USDC, gives ETH as collateral
Lending Smart Contract
requests ETH/USDC price, provides callback
sends ETH/USDC price to callback
Oracle Smart Contract
emits
Event
send ETH/USDC price
writes ETH/USDC price
reads
Oracle Service (EAO)
requests ETH/USDC price
returns ETH/USDC price
External Web API
Ledning Smart Contract gets fresh price by providing a callback to the Oracle Smart Contract
The Oracle Service waits for event to act.
Transaction
Message
HTTP Connection
On-chain
Off-chain

Figure 2: Asynchronous Oracles

DEXes as Price Oracles

DEXes like Uniswap or SushiSwap can serve as price on-chain price oracles because they contain pools of tokens with real-time price information determined by market forces. For example, an ETH/USDC pool on a DEX can be queried to determine the current market price of ETH in USDC. More about DEXes and AMMs is provided in the next section.

3 Automated Market Makers

Imagine you want to exchange your valuable token for another token or buy or sell some token with ETH. The service you are looking for is an exchange. Traditional centralized exchanges use order books where buyers and sellers are matched. Examples of such centralized services include Binance, Coinbase, Kraken, etc. However, the web3 world is about decentralization, right? What if we told you that you could exchange your assets simply with a smart contract? Decentralized Exchanges (DEXes) often use Automated Market Makers (AMMs), which are smart contracts that create markets for any pair of tokens by using liquidity pools. They rely on a mathematical formula to price assets instead of using an order book. The most commonly used formula is the **Constant Product**, which follows the equation:

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

where x is the quantity of Token A in the liquidity pool, y is the quantity of Token B in the liquidity pool, and k is a constant, representing the pool's total liquidity.

Token Swaps

When a trader swaps Token A for Token B, the new balance in the pool must still satisfy the constant product equation. If a trader adds Δx amount of Token A, the new state of the pool must satisfy:

$$(x + \Delta x)(y - \Delta y) = k \tag{2}$$

Solving for Δy, the amount of Token B the trader receives:

$$\Delta y = \frac{y \cdot \Delta x}{x + \Delta x} \tag{3}$$

Price Impact

The price of Token A in terms of Token B is given by the derivative of the invariant function:

$$\frac{dy}{dx} = -\frac{y}{x} \tag{4}$$

This means that as more of Token A is purchased, its price increases due to the decreasing reserve, illustrating **slippage**.

Liquidity Providers

Liquidity Providers are individuals who deposit tokens into AMM pools. In return for their deposits, they receive special liquidity provider (LP) tokens that represent their share of the pool. To incentivize liquidity providers, a fee f (e.g., 0.3%) is applied to each trade. This means the value of their LP tokens increases with every trade. Liquidity Providers can withdraw their liquidity and collect the accumulated fees at any time. The modified constant product formula with fees is:

$$(x + \Delta x \cdot (1 - f))(y - \Delta y) = k \tag{5}$$

For example, with a 0.3% fee, the effective input amount of the trader is reduced to 99.7% of the actual input. This can be represented as:

$$(x + \Delta x \cdot 0.997)(y - \Delta y) = k \tag{6}$$

Solving for Δy, the amount of Token B the trader receives:

$$\Delta y = \frac{y \cdot \Delta x \cdot 0.997}{x + \Delta x \cdot 0.997} \tag{7}$$

Slippage

A key characteristic of AMMs is that the larger the trade relative to the pool size, the more significant the price impact or “Slippage”. This is because removing a larger percentage of one token from the pool causes the price to move more dramatically according to the constant product formula.

Tip: Video explanation How do LIQUIDITY POOLS work?

Stablecoins and USDC

Stablecoins are cryptocurrencies designed to maintain a stable value relative to a specific asset, typically a fiat currency like the US dollar. They aim to combine the benefits of cryptocurrencies (fast transfers, programmability, global accessibility) with the stability of traditional currencies.

USDC is one of the most widely-used stablecoins in the crypto ecosystem. It is simply an ERC20 token. Each USDC token should be backed by $1 held in reserve. Circle, the company behind USDC, is regulated and publishes monthly attestations verifying these reserves. See USDC token contract on Etherscan.

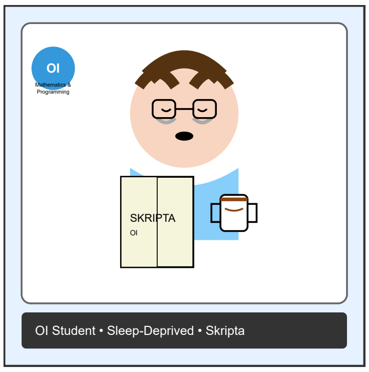
To learn more about DEXes, oracles and stablecoins we recommend Berkeley RDI Center on Decentralization & AI course Decentralized Finance:

- DeFi MOOC Lecture 5: DEX
- DeFi MOOC Lecture 7: Stablecoins
- DeFi MOOC Lecture 8: Oracles
- The full DeFi MOOC course with materials

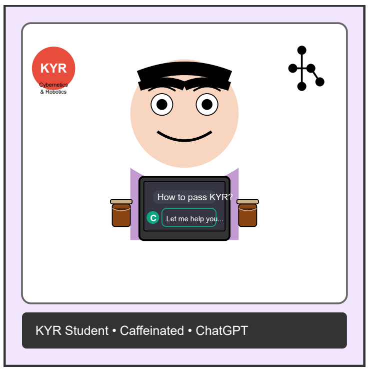
4 Task

Task 1: Fool the Oracle 01

The new rare NFT collection “FEL Students” is finally on the market! The first three unique pieces are now available: Sleep-deprived OI student learning from skripta for 5,000 USD, BIO pulling-all-nighter female student addicted to energy drinks for 8,000 USD, and Caffeinated KYR student who will graduate only thanks to ChatGPT for 3,000 USD. They are so unique and cool! You must have them all! Unfortunately, as a student, you can’t afford them and start with only 3,000 USDC tokens and 0.8 ETH, which is barely enough for the Caffeinated KYR student. Your goal, however, is to get them all!


OI
Mathematics & Programming
SKRIPTA
OI
OI Student • Sleep-Deprived • Skripta

BIO
Medical Electronics & Bioinformatics
3:45 AM
ENERGY
ENERGY
ENERGY
GENOMICS ANALYSIS
BIO Student • All-Nighter • Energy Drink



Your Mission: Buy all three NFTs that are listed on the market
Files that are relevant for this challenge:

- test/**FoolTheOracle01.js**: The test file where you should code your solution.
- contracts/**SimpleDEX.sol**: A decentralized exchange with a one liquidity pool containing 1 ETH and 2,000 USDC.
- contracts/**FELStudentNFT.sol**: A collection of unique CTU FEL Student NFTs with different traits.
- contracts/**NFTMarketplace.sol**: A marketplace for trading these NFTs. It uses SimpleDEX as its price oracle to convert USD (USDC) prices to ETH.
- contracts/**USDCToken.sol**: An ERC20 token representing simplified version of USDC.

Code your solution in the `test/FoolTheOracle01.js` file. Use only the player account. Verify your solution by running the following command:

```
$ npm run oracle01
```

Flash Loans

Flash loans are a unique DeFi primitive that allows users to borrow assets without collateral, as long as the borrowed amount (plus any fees) is returned within the same transaction. If the borrower fails to repay, the entire transaction is reverted, ensuring lenders never lose funds. This atomic property makes flash loans a powerful tool for arbitrage, collateral swaps, and other complex DeFi operations—but also creates potential for oracle manipulation attacks. Flash loans leverage Ethereum's transaction atomicity—either the entire transaction succeeds or it fails completely. The process follows these steps:

1. A user calls a flash loan provider to borrow tokens
2. The provider transfers tokens to the user
3. The user executes operations with the borrowed funds
4. The user returns the borrowed amount plus fees

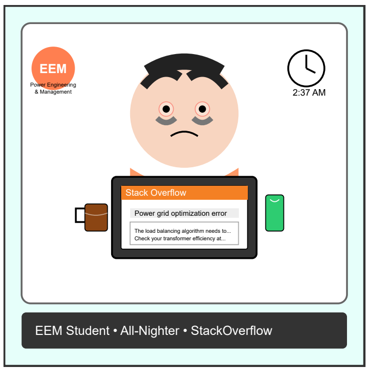
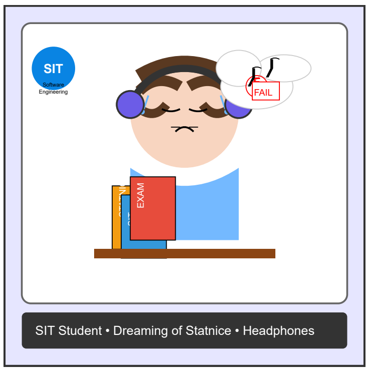
5. If repayment fails, the entire transaction reverts

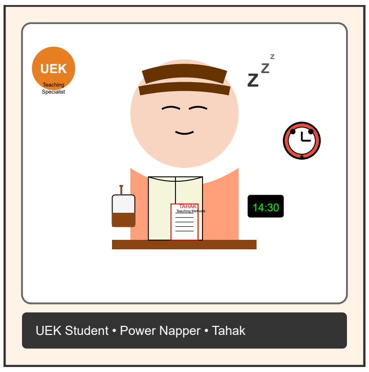
This pattern is implemented in our `FlashLoanProvider.sol` and `FlashLoanReceiver.sol` using the ERC-3156 standard.

Tip: Video explanation Flash Loans Explained

Task 2: Fool the Oracle 02

The success of the first FEL Student NFT collection was tremendous! The same marketplace now lists three new, even more valuable NFTs: EEM student pulling an all-nighter on Stackoverflow - 3,500 USD, SIT student dreaming of Statnice while listening to headphones - 7,500 USD and UEK power napper with a Tahak - 5,000 USD.


EEM
Power Engineering & Management
2:37 AM
Stack Overflow
Power grid optimization error
EEM Student • All-Nighter • StackOverflow




This time, the marketplace has become more resilient. The SimpleDEX now has a much larger liquidity pool containing 100 ETH and 200,000 USDC, making direct price manipulation much more difficult. At the current exchange rate of 1 ETH = 2,000 USDC, the total value of all NFTs is approximately 8 ETH. As a poor student, you only have 3 ETH on your wallet (and no USDC). Buying all three NFTs legitimately is impossible with your current funds. However, you have a secret weapon: Flash Loans! For this challenge, you can use the Flash Loan Provider, which has a pool of 200,000 USDC available for flash loans. The provider charges a minimal fee of just 0.01% (1 basis point) on all loans.

Your Mission: Acquire all three FEL Student NFTs despite your limited resources.
Files that are relevant for this challenge:

- test/**FoolTheOracle02.js**: The test file where you should code your solution.
- contracts/**FlashLoanProvider.sol**: The ERC-3156 compliant flash loan provider with 200,000 USDC.
- contracts/**FlashLoanReceiver.sol**: Template for creating a flash loan exploit contract.
- contracts/**SimpleDEX.sol**: The DEX with increased liquidity (100 ETH, 200,000 USDC).
- contracts/**FELStudentNFT.sol**: The NFT collection.
- contracts/**NFTMarketplace.sol**: The marketplace using SimpleDEX as price oracle.

- contracts/**USDCToken.sol**: An ERC20 token representing simplified version of USDC.

Code your solution in the `test/FoolTheOracle02.js` file and the `contracts/FlashLoanReceiver.sol` file. Use only the player account. Verify your solution by running the following command:

```
$ npm run oracle02
```