**Smart Contract Vulnerability Report**

**1. Introduction**

- Objective:

Smart contract is the backbone of the entire blockchain financial ecosystem, responsible for moving around enormous amount of money and capital. While inheriting all the merits of blockchain technology, immutability, transparency, and security, these characteristics also serve as attack vectors for malicious actors to exploit the underlying logic.

This report will cover one of the most common vulnerabilities, as well as provide a deep dive, and attack proof-of-concept. The report will also provide mitigation strategies and best practices to avoid these vulnerabilities.

**2. Glossary**

- EVM: Ethereum virtual machine, the engine behind smart contract technology to execute code

- Tx: Transaction, request for state update/computation. External function call are Tx.

- Miners: Computing resource providers, abide to the consensus to work together in a trustless manner

- Gas: Unit to calculate computational effort. Gas price are calculated differently by different chains, mainly on the computational effort of the last block.

- Accounts: Actors on ethereum-based blockchains, have balance and can be target of transactions

- EOA: externally owned accounts, controlled by private key

- Smart contract: controlled by code executed by a Tx

- Flash loan: A protocol for uncollateralized lending, the only restraint being that both borrowing and paying back is done in the same transaction.

**3. Background**

On EVM based blockchain, smart contracts and externally controlled accounts are both treated as accounts, holding custody over funds. Only difference is that smart contracts can operate on logic, programmable. As such, it is very important to keep in mind the capabilities of actors when designing a smart contract.

Public chains only complicate the matter, since everything is open to the public, with the only requirement being an internet connection. Malicious actors can analyze the code, find weaknesses, and exploit them. Therefore, a thorough understanding of potential vulnerabilities and careful audition is essential to ensure the security and robustness of smart contracts.

**4. Common attacks**

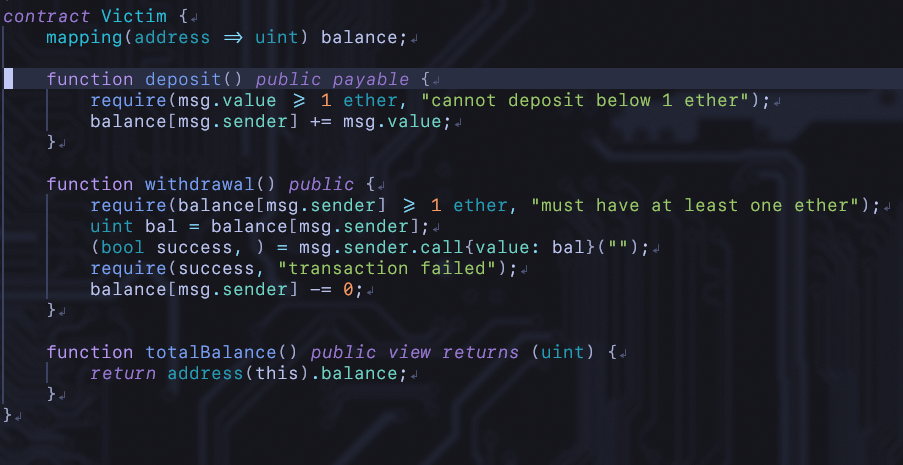
**4.1. Reentrancy**

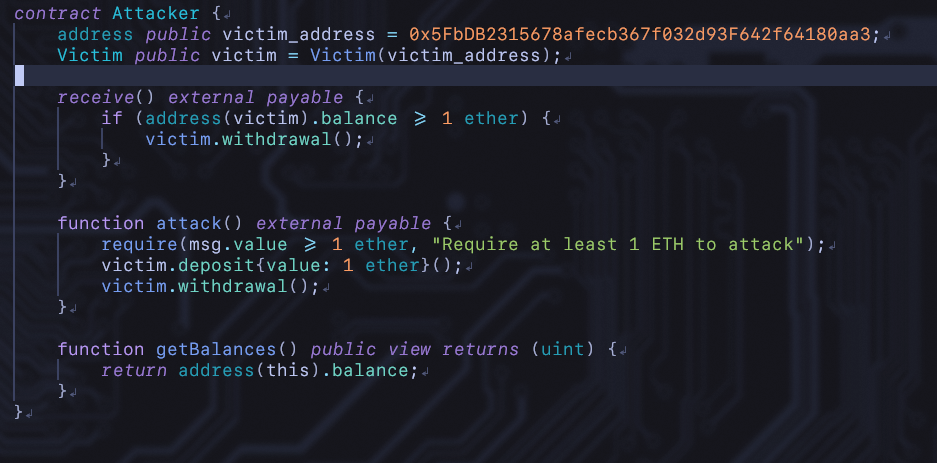
One of the most common exploits on smart contracts is known as reentrancy. It is the seizure of control flow by the calling contract.

Reentrancy takes advantage of improper state update of external functions, examples include using fallback functions to jump in the middle of logic execution before the victim contract properly updates its state.

**Impact:** draining fund that one does not have custody over. Large real-world example includes the Revest Finance hack in 2022, where the NFT platform lost ~2M USD to reentrancy, and EraLend, a zkSync lending platfrom, losing 3.4M USD to multiple flaw in its protocols

**Reentrancy Victim contract**



This is a basic victim contract that allows the deposit and withdraw of ETH. The vulnerablities lies at the fact that this contract lets caller withdraw fund before updating balance.  
  
**Reentrancy Attacker**  


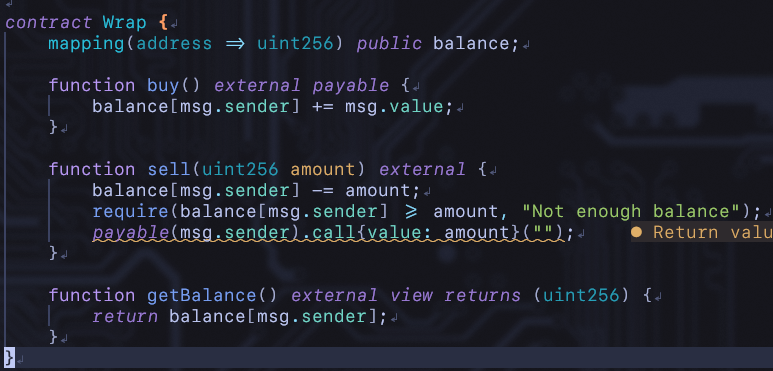
Attacker contract may exploit this vulnerability by using a receive() function that is triggered upon ETH receival, re-entering the victim to drain fund again before the balance is updated. This attack will fail if the victim simply update the balance before any fund leaves the contract.

**Mitigation**

Remedy includes changing state before external call; locking state with bool, and function modifier to prevent reentry

**4.2. Integer Underflow**

Unsigned integer data type in smart contracts created with solidity are fixed in size, as such, are vulnerable to underflow and overflow. Prior to version 0.8.0, smart contracts relies on openzeppelin’s safe math library in order to mitigate this issue, but this is not enabled by default, allowing overflow and underflow to happened, severly damaging the normal operation of the logic.



Above is an example of an ETH wrapper, built to expand functionalities of ETH, where users may fund some amount of ETH in exchange for balance in the contract.  
The withdraw function allows someone to withdraw more than their balance, triggering the value to underflow to very large value.

**Mitigation:**

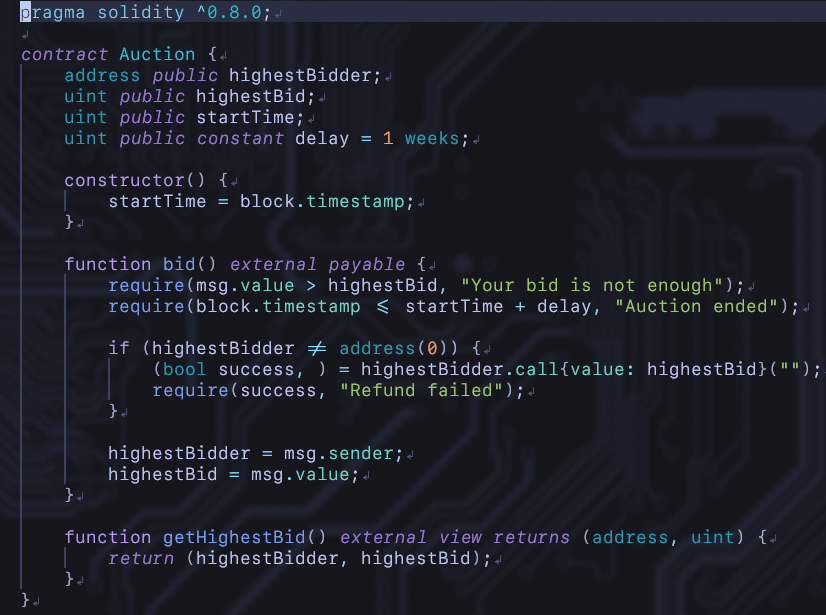
Use solidity version 0.8.0 and above, use safe math library

**4.3. Gas Griefing**

Gas griefing attacks are used to disrupt normal operation of the blockchain, usually targetting business logic. These can cause network congestion, inflate gas prices, or even allowing the attacker to gain unfair advantage.

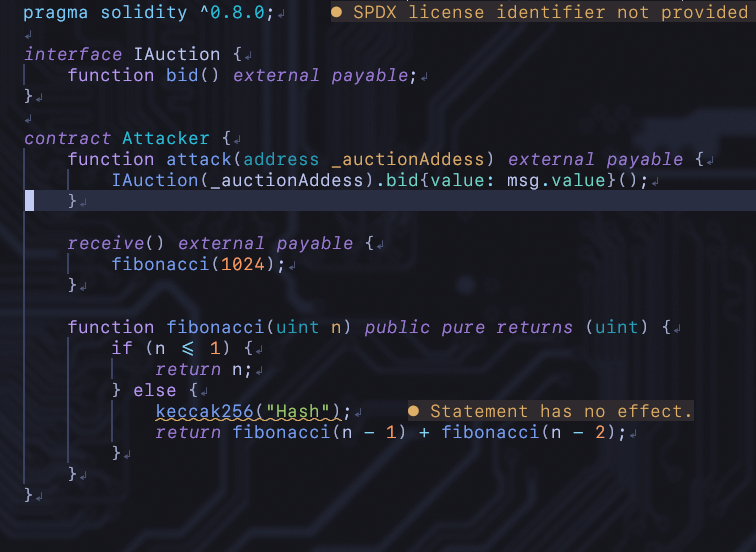
Similar to rentrancy where the fallback / receive funtion is ultilized, a malicious actor can use these to make the gas price increase pass the acceptable limit, potentially locking essential functionalities of a protocol.

**Vulnerable “Auction” contract:**



Above is a simple implementation of an auction, where user may bid with their ETH. When a higher bidder comes in, fund is returned to the previous one, and the state is updated.

**Gas griefer contract**



By modifying receive function to do very expensive calculation, an attacker may gain unfair advantage by making it impossible to pay back ETH by exceeding the gas limit. Thus, he is guarenteed to win the auction, despite putting in very litlle bid.

This specific attack be defeated by requiring bidder to withdraw their own fund instead of returning automatically when a new highest bidder comes in.

**Mitigation**

Use a manual withdrawal, were user must retake fund on their own; apply rate limits; economic incentives to reward good behavior and punish malicious actors

**4.4. Oracle manipulation**

Hybrid applications that combine off-chain data and on-chain calculation relies on Oracles to provide data for their code execution, making them essential parts of many finantial infrastructures.

Oracles are smart contracts that accept, verify then store data. This data can then be queried to execute business logics. Components of an oracle includes:  
- Ground truth: information to be queried

- Data source: source for information

- Data feeder: Extract the ground truth from source

- Data feeder selection

- Data aggregation

- Dispute phase: Incentive honest reporting and punish malicious actors

Oracle manipulation attack cause the oracle to report wrong or outdated data, allowing the attacker to steal fund, often in the hundred of thousands. This exploit is often performed as part of a larger attack on a protocol, usually targeting one of the components.

The data source is often targeted by malicious actors to perform an attack known as Spot Price Manipulation. Using flash loan, attacker may artificially inflate or deflate the value of a token on an exchange that is being used as a price oracle. A lending protocol using this oracle will receive this manipulated data, and allow the attacker to take out massively undercolateralized assets, at the lost of everyone else.

**POC**

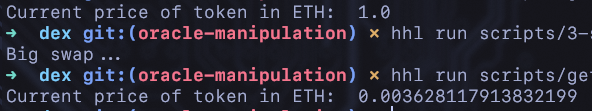
This POC includes a very simple oracle that uses a decentralized exchange for its spot prices. Due to some technical difficulties, instead of using flash loan to fund the attack, the upfront cost will come from the hardhat runtime environment.

- The [exchange contract](https://github.com/lacklusterer/KMSDEX/blob/4dfe06db750ddd4e715fe92fea55f2f965872794/contracts/exchange.sol) is an implementation of a decentralized exchange that ultilize the constant product market maker mechanism to calculate the value of exchanged assets, similar to uniswapv2 protocol. It maintains a liquidity pool of 2 assets (ETH or ERC-20 tokens) [[more on the exchange](https://github.com/lacklusterer/KMSDEX/blob/63d570654e9c8035e79d392fc4fb7c6803845789/README.md)]

**Price oracle**

****

This on-chain price oracle asks the exchange for value of assets in the liquidity pool.



A large swap can cause the value of assets reported by the oracle to derail massively, normally, this requires a large amount of upfront capital, but flash loan allows attacker to fund such attacks with almost zero risks.

**Mitigation**

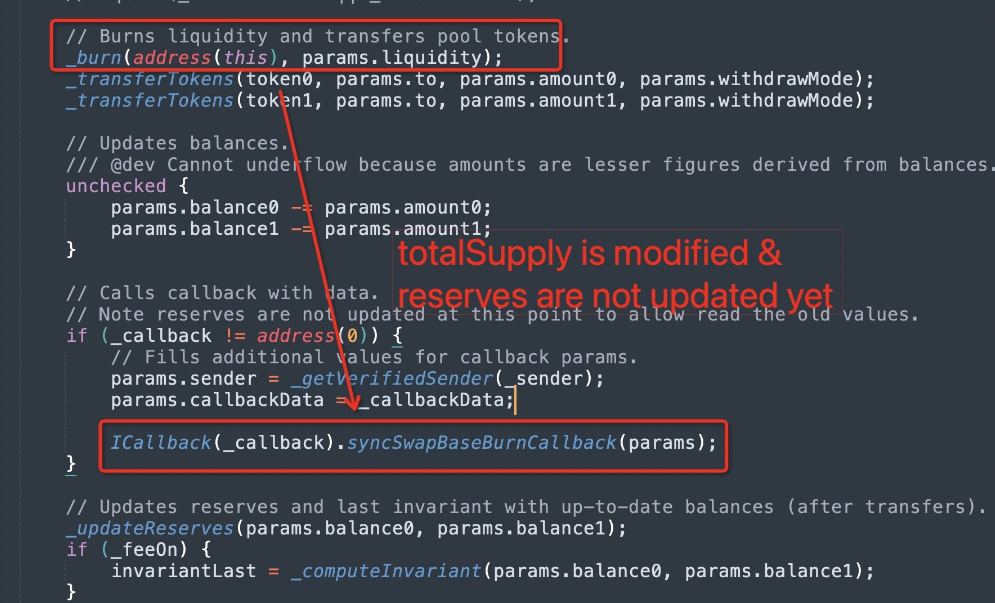
To bolster security of price oracle, it should instead use multiple sources for information, and select large CEX or DEX, as well as off chain sources with proper validation.

The UniswapV3 protocol was developed with option to fetch time-weighted average price and liquidity to prevent manipulation, however, these are not effective if used on exchang with low liquidity.

**5. Case study**

In practice, attacks rarely occurs on its own. Most high profile attacks combined a variety of different exploits. In 2023, [EraLend](https://rekt.news/eralend-rekt/) lost an estimated 3.4 million USD to a combination of flash loan, oracle manipulation and reentrancy vulnerability.

The attacker exploited a read-only reentrancy vulnerability within EraLend’s smart contracts. Using flash loan to obtain a large amount of liquidity, the attacker was able to burn the LP tokens then callback boefore the reserves were updated, causing the price oracle to calculate report a greatly inflated value of the assets. The attacker made away with ~2.7 million USD to EraLend’s ~3.4 loss.

****

***syncswap LP token code, including a comment to warn about reserves not being updated [image is courtesy of rekt.news]***

Exploiting transactions: [Tx1](https://explorer.zksync.io/tx/0x99efebacb3edaa3ac34f7ef462fd8eed85b46be281bd1329abfb215a494ab0ef); [Tx2](https://explorer.zksync.io/tx/0x7ac4da1ea1b0903dfabda56f713ea5e4a960a3fc34467a844d037f86ee8bfe98)

**6. Conclusion**

Ensuring the security of smart contracts is paramount for the integrity and trustworthiness of the blockchain financial ecosystem. As demonstrated, vulnerabilities such as reentrancy, integer underflow, gas griefing, and oracle manipulation can have devastating effects if not properly addressed. Proactive measures, best practices and thorough auditting are essential to mitigate these risks. By adopting these strategies, developers can build more resilient smart contracts, safeguarding assets and maintaining the confidence of users in decentralized platforms. The ongoing vigilance and commitment to security will be the cornerstone of the continued success and adoption of blockchain technology.

**7. Reference**

1. <https://owasp.org/www-project-smart-contract-top-10/>

2. <https://ethereum.org/en/developers/docs/smart-contracts/security/>

3. <https://scsfg.io/hackers/griefing/>

4. <https://www.getsecureworld.com/blog/smart-contract-gas-griefing-attack-the-hidden-danger/>

5. <https://scsfg.io/hackers/oracle-manipulation/>

6. <https://rekt.news/eralend-rekt/>

7.<https://www.certik.com/resources/blog/4NPEuNEiaUUcm6S3gdKKLP-eralend-incident-analysis>

8.<https://www.halborn.com/blog/post/what-is-oracle-manipulation-a-comprehensive-guide>