Investigating Arbitrageurs and Oracle Manipulators in Ethereum

Written by
Kevin Tjiam
Technische Universiteit Delft
k.c.tjiam@student.tudelft.nl

Under the supervision of
Prof. Dr. Kaitai Liang
Technische Universiteit Delft
kaitai.liang@tudelft.nl

Abstract

Smart contracts on Ethereum enable billions of dollars of value to be transacted in a decentralised, transparent and trustless environment. However, adversaries lie await in the Dark Forest, waiting to exploit any and all smart contract vulnerabilities in order to extract profits from unsuspecting victims in this new financial system. As the blockchain space moves at breakneck pace, exploits on smart contract vulnerabilities rapidly evolve, and existing research quickly becomes obsolete. It is imperative that smart contract developers stay up to date on the current most damaging vulnerabilities and countermeasures to ensure the security of users' funds, and to collectively ensure the future of Ethereum as a financial settlement layer. This research focuses on two smart contract vulnerabilities: *transaction-ordering dependency* and *oracle manipulation*. Combined, these two vulnerabilities have been exploited to extract hundreds of millions of dollars from smart contracts in the past year (2020-2021). For each vulnerability, this research presents: (1) a literary survey from recent (as of 2021) formal and informal sources; (2) a reproducible experiment as code demonstrating the vulnerability and, where applicable, countermeasures to mitigate the vulnerability; and (3) analysis and discussion of proposed countermeasures. To conclude, strengths, weaknesses and trade-offs of these countermeasures are summarised, presenting direction for future research.

1 Introduction

Blockchain technology provides a way to record transactions on a distributed ledger that is immutable, decentralised and cryptographically secure. While initially popularised by Bitcoin, the Ethereum network builds on this idea of an immutable public ledger with the ability to execute arbitrary programs in a decentralised manner, with results recorded on the blockchain, allowing for more than just simple peer-to-peer transactions to be made. As described in the original Ethereum whitepaper by Vitalik Buterin [5], these programs

Figure 1: Graph of Total Value Locked (USD) in DeFi from June 2020 to May 2021. (from DeFi Pulse)

are called smart contracts and are the basis of the recently-birthed DeFi[1] movement. Figure 1 illustrates how DeFi has shown explosive growth, achieving over $76B (USD) of TVL[2] as of May 2021 [9]. In the world of DeFi, code is law. Instead of needing to trust opaque entities such as banks, financial transactions are executed by smart contracts that are deployed onto the blockchain. Anyone can review and verify any smart contract deployed to the blockchain before interacting with it, providing a decentralised, transparent and trustless financial environment. As described in the paper by Atzei, Bartoletti, and Cimoli [3], smart contracts are written in a Turing-complete bytecode language called EVM[3] bytecode. These contracts are executed in a decentralised manner by *miners*, and the results of the execution are recorded immutably on the blockchain after the network reaches consensus. It is only through the successful interaction of these complex mechanisms that the Ethereum network is able to effectively maintain an immutable and trustless ledger.

Through the deployment of and interaction with smart contracts, DeFi replicates traditional financial instruments such as exchanges, yield-bearing assets and derivatives trading, while also introducing novel concepts such as AMMs[4][14] (this lifecycle is shown in figure 2). These smart contracts range from very simple storage of data,

[1] Decentralised Finance
[2] Total value locked
[3] Ethereum Virtual Machine
[4] Automated Market Makers

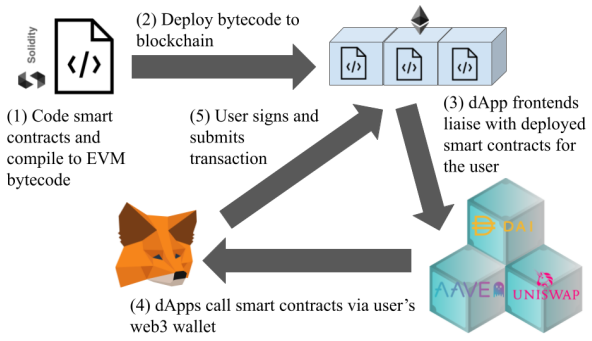

Figure 2: A diagram depicting the smart contract lifecycle that powers DeFi.

to standardised contracts such as ERC-20[5] tokens, to very complex smart contracts like Uniswap V3's capital-efficient AMM liquidity pools [26]. A vulnerability that prevents the correct execution of smart contracts may allow a bad actor to siphon value from legitimate users of the Ethereum network. Therefore, it is imperative to thoroughly investigate existing and potentially undiscovered vulnerabilities in order to evaluate whether we have effective solutions or mitigations in order to ensure the continued future of the Ethereum network as a settlement layer for billions of dollars in transacted value.

In-depth literature exists on the execution of smart contracts across many platforms, including Ethereum, Quorum and Hyperledger (as described by Hu et al. [15]). Furthermore, there exist recent surveys, like the one conducted by Khan and Namin [16], that catalogue known vulnerabilities in Ethereum smart contracts, placing them in useful categories such as inter-contractual vulnerabilities, arithmetic bugs, and gas-related issues, among others. Khan and Namin also include in their paper a brief section on available tools that help to mitigate these vulnerabilities, while other papers like the Sereum paper [21] propose specific countermeasures that mitigate certain classes of vulnerabilities (re-entrancy in Sereum's case).

While these papers are comprehensive in the vulnerabilities and countermeasures they cover, the explanations can be somewhat obtuse, such as in [16]. Furthermore, the provided implementations of vulnerabilities are limited to simplified, fictional code listings, which are not conducive to future research. To address this issue, this research contributes implementations of vulnerabilities and countermeasures based on real smart contracts that are actively used on the Ethereum Mainnet[6], where applicable.

As the rather young blockchain space also innovates at breakneck pace, some of these discussed vulnerabilities and countermeasures may no longer apply, or have very recent developments in improving said countermeasures.

This research investigates security and privacy vulnerabilities in Ethereum-based smart contracts that present the highest risks to the ecosystem, and presents up-to-date analyses of these vulnerabilities and state-of-the-art countermeasures in an easily digestible format. More importantly, this research demonstrates each vulnerability and its respective countermeasure(s) as a reproducible experiment, defined as code, so that they may be used as a starting point for future experiments. This code can be found in the accompanying GitHub repository located at https://github.com/kevincharm/arbitrageurs-and-oracle-manipulators. The research questions that will be answered in this paper are as follows.

1. What smart contract vulnerabilities present the highest risk to the Ethereum ecosystem, and what features of these vulnerabilities make them pose a higher risk compared to other vulnerabilities?
2. What are the security and privacy implications of transaction-ordering dependency vulnerabilities in Ethereum-based smart contracts?
   (a) How can transaction-ordering dependency vulnerabilities be mitigated? Are these countermeasures effective, or can we do better?
3. What are the security implications of using oracles in Ethereum-based smart contracts?
   (a) How can smart contracts resist oracle manipulation? Are these countermeasures effective, or can we do better?

First, the methodology of how this research and its experiments were carried out are explained in section 2. Then, section 3 enumerates the contributions of this research. Following on, section 4 gives a detailed run-through on the transaction-ordering dependency vulnerability and its countermeasures, as well as a discussion on the security and privacy implications. Similarly, section 5 describes in detail the oracle manipulation vulnerability and its countermeasures. Finally, the results of this research are discussed in section 6.

2 Methodology

The methodology used in carrying out this research is described in this section. A literary survey of prominent attack vectors that were concerned with privacy and security vulnerabilities on Ethereum smart contracts was conducted, focusing on more recently published materials. These published materials included not only formal research papers, but also many informal sources such as personal blogs and twitter feeds of prominent Ethereum security researchers like samczsun[7] and Igor Igamberdiev[8], and conversations that took place in Telegram channels where well-known DeFi developers congregate such as LobsterDAO[9].

After a literary survey was carried out, two specific vulnerabilities stood out as being current major problems

[5]ERC-20 Token Standard: https://ethereum.org/en/developers/docs/standards/tokens/erc-20

[6]Ethereum's production network is referred to as Mainnet; test networks (testnets) are named Ropsten, Kovan, Rinkeby, and Goerli.

[7]samczsun: https://samczsun.com

[8]FrankResearcher: https://twitter.com/FrankResearcher

[9]LobsterDAO: https://t.co/75UmHVB8lz

in the Ethereum ecosystem at the time of writing. These two vulnerabilities were Transaction-Ordering Dependency and Oracle Manipulation. Following more in-depth research on these selected vulnerabilities and their countermeasures, reproducible test cases were written, using real smart contracts and a locally forked mainnet - made possible by using a tool named Hardhat[10]. These test cases were written to demonstrate some of the ways the selected vulnerabilities have been exploited by adversaries. Countermeasures to these exploits were also proposed as code, derived from prior work, to exemplify potential solutions or mitigations to the problems caused by these exploited vulnerabilities. An evaluation and comparison of these selected vulnerabilities, along with other vulnerabilities encountered in the literary survey, their severities and categorisations have also been included in the discussion of results. Additionally, potential new solutions have been proposed as novel countermeasures to each of the selected vulnerabilities.

3 Contribution

This research presents an up-to-date representation of the major security and privacy-related vulnerabilities plaguing Ethereum-based smart contracts in the year 2021. As the DeFi space is still in its infancy, the most recent exploit analyses are published through more informal sources and this research aggregates these findings into reproducible code accompanied with easily digestible discussions and evaluations. The main contributions of this research are as follows.

- Insight into the current hot topics in vulnerabilities is given, to provide a base on which other researchers can look into vulnerabilities and hopefully develop better countermeasures to mitigate the effects of these exploited vulnerabilities.
- Background knowledge is provided for the reader to easily understand the complex concepts behind smart contract execution and how vulnerabilities are exploited.
- Code is provided, using real smart contracts on Ethereum mainnet, where appropriate and applicable, as examples. All code referred to in this research paper can be found in the GitHub repository located at https://github.com/kevincharm/arbitrageurs-and-oracle-manipulators.
- Vulnerabilities are evaluated and compared, looking at their features, categorisation, severities.
- Known countermeasures, their advantages, limitations, are discussed and illustrated, with real world examples.

4 Transaction-Ordering Dependence

Certain classes of smart contracts are prone to the transaction-ordering dependency (henceforth abbreviated as TOD) vulnerability. Namely, smart contracts powering AMM[11] DEXes[12] such as Uniswap fall under this classification.

[10]Hardhat: https://hardhat.org
[11]Automated Market Maker
[12]Decentralised Exchanges

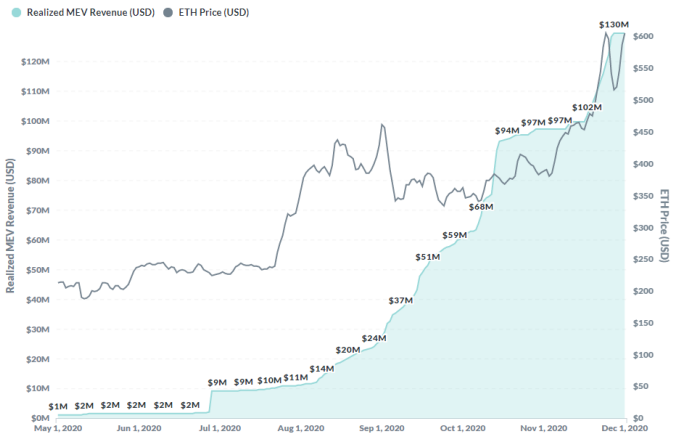

Figure 3: The exponential growth of MEV revenue in 2020 (from Paradigm Research) [18]

As described in the Uniswap V2 Core Whitepaper [1], AMM smart contracts manage large amounts of token pairs (so-called liquidity pools). Uniswap, being specifically a CPMM[13], enforces that the amount of tokens x and y in the pool maintains a constant product k such that $x \cdot y = k$. In other words, the price of each token is determined by the ratio of the tokens in the liquidity pool as users trade tokens into and out of the pool, while the smart contract enforces that the pool maintains the constant product k. It should be clear then, in this AMM mechanism, the price of a token at any given time depends on the order of buy and sell transactions.

The presence of this dependence on transaction-ordering has led to the currently on-going MEV crisis, first coined by Daian et al. [7] in their Flash Boys 2.0 paper, which gives deep insight into the numerous arbitrage opportunities on DEX smart contracts that have been employed by bots, and how the high gas fees paid by these bots pose significant consensus-layer security risks. MEV[14] is the measure of profit that is available to be exploited by miners, as they ultimately control the inclusion, exclusion and ordering of transactions within blocks that they mine, and is a direct result of the transaction-ordering dependency vulnerability in smart contracts [18]. Figure 3 shows the exponential growth in MEV revenue since the dawn of DeFi summer up until December 2020. Ergo, the exploitation of transaction-ordering dependency vulnerabilities is a security risk in Ethereum smart contracts as it can lead to honest users suffering monetary losses, and more severely, consensus-layer instability. It can also be considered a privacy risk, as the public and transparent nature of the Ethereum mempool is a feature of attacks exploiting this vulnerability.

4.1 Enter the Dark Forest

The Ethereum mempool is a Dark Forest, as Robinson [20] puts it, referencing a sci-fi novel written by Cixin Liu. The novel describes a Dark Forest as a dangerous environment where detection by advanced predators means

[13]Constant Product Market Maker
[14]Miner/Maximum Extractable Value

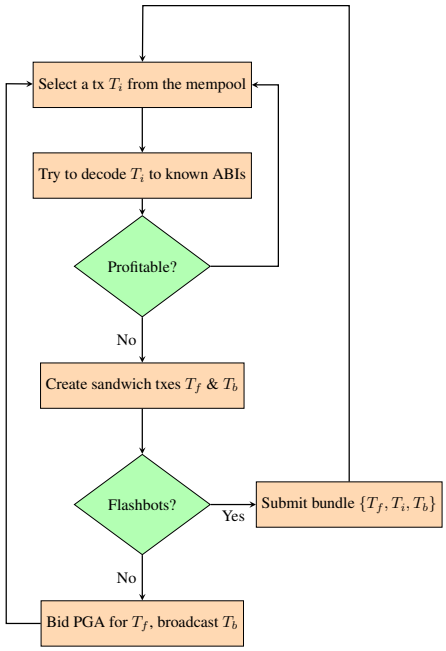

Figure 4: Flowchart of an indefatigable specialised frontrunner.

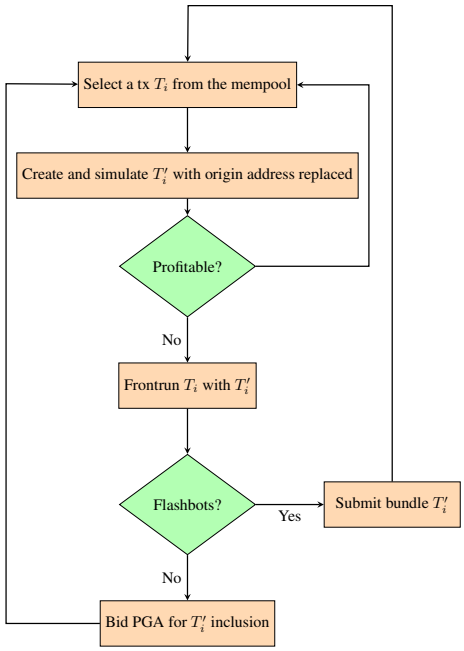

Figure 5: Flowchart of the *cosmic horror* lurking in the Dark Forest; the generalised frontrunner.

a swift death. This term indeed accurately describes the Ethereum mempool, where transactions are broadcasted before they are picked up by miners for inclusion in a block. Advanced predators in the form of frontrunning bots actively scan the mempool for any transactions with value to extract. The most common and low-level predators are *specialised frontrunners*, which are designed to detect transactions from specific types of contracts. Then there are *generalised frontrunners*, described by Daian et al. [7] and dubbed a "cosmic horror" by Robinson [20], which have the ability to scan the mempool for any profitable transactions to frontrun [17]. Flowcharts are presented in figures 4 and 5 depicting the indefatigable processes of these specialised and generalised frontrunners, respectively.

Frontrunning, as previously mentioned, is an exploit that is a direct result of the TOD vulnerability. The term *frontrunning* hails from the world of traditional finance, referring to the practice of running to the front of the queue after receiving information about a big incoming trade [8]. This analogy quite elegantly explains how the exploit works on the Ethereum network: as described by the *Dark Forest*, adversarial agents monitor the mempool for transactions with extractable value and then attempt to broadcast malicious transactions that are guaranteed to be included before the original transaction, effectively profiting from honest users of the network. This research primarily focuses on TOD vulnerabilities in DEX smart contracts, as these are amongst the biggest targets of MEV. One metric to support this claim is the fact that Uniswap is consistently at the top of the gas guzzlers rankings (dApps that consume the most gas) on Etherscan's Ethereum Gas Tracker[15] which paints it as a hotspot for high volumes of extractable value.

[15]Ethereum Gas Tracker: https://etherscan.io/gastracker

Adversarial agents employ numerous techniques to exploit smart contracts that depend on transaction ordering, which are discussed in the following subsections.

4.2 Sandwich Attack!

The simplest and most commonly encountered subtype of specialised frontrunning is the *sandwich attack*. Sandwich attacks take advantage of the high slippage tolerance required in large trades on AMM DEXes, especially for token pairs with low liquidity or high volatility. The amount of tokens that will be swapped by the smart contract (and therefore, the price of the token) depends on the token reserves in the liquidity pool for the token pair. Thus, it follows that there is room for the price to be manipulated before the trade, to extract profit, as long as the trade ultimately falls within the slippage tolerance. When a frontrunning bot sees a valuable trade like the one previously described, it will insert a buy order for the same tokens such that the price of the token increases but still within the trade's slippage tolerance, followed by a sell order (for the same amount of tokens bought) immediately after the user's trade, thereby making a profit. Shown in figure 6 is an annotated list of trades taken from ChartEx[16], a trading tool, exemplifying a sandwich attack that drained 0.09 ETH of value from frontrunning a buy order. Zhou et al. [27] found that a single arbitrageur has the ability to extract several thousands of USD per day from performing sandwich attacks on Uniswap. Furthermore, they describe a more complex variant of the sandwich attack that involves frontrunning to remove liquidity, then re-adding liquidity and selling after the victim's transaction.

[16]ChartEx: https://chartex.pro

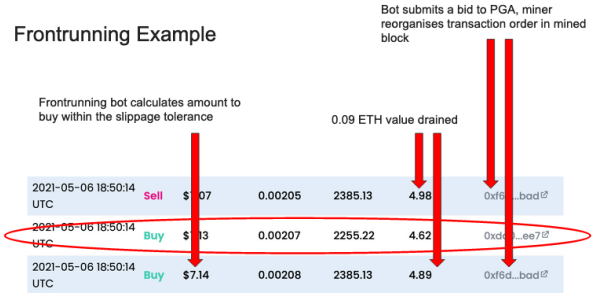

Figure 6: Sandwich attack that drained 0.09 ETH, shown on ChartEx

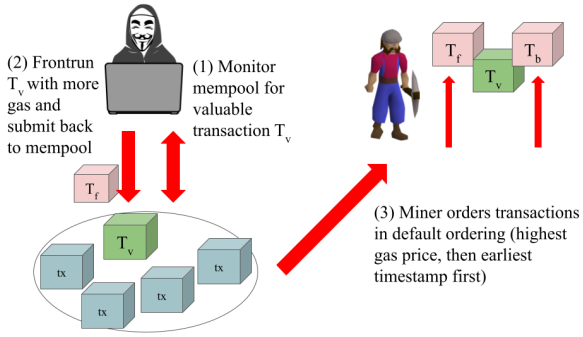

Figure 7: A diagram illustrating how adversaries guarantee transaction order by exploiting Geth's default transaction ordering algorithm.

4.3 Guaranteeing Transaction Order

In order to guarantee a profit from a sandwich attack, there are strict requirements about the transaction ordering within the mined block. That is, the malicious buy order transaction must be included immediately before the victim's transaction, followed by the malicious sell order transaction immediately after. One way to achieve this is by taking advantage of how certain Ethereum clients order transactions within a block. It was found by Zhou et al. [27] that out of analysing 388 days of trading on Uniswap, 79% of the transactions were ordered by gas price, which means that this percentage of Ethereum miners likely use the Geth[17] client, which orders transactions for inclusion by gas price. Users have the ability to specify how much to pay the miner (called the *gas price*) to have a transaction included in the block. The higher the gas price, the more likely it is that the transaction will be included in the next block. Listing 1 illustrates in code how an attacker would frontrun the victim's transaction by specifying a gas price that is explicitly higher for the buy order, and a gas fee that is equivalent to the victim's transaction for the sell order, as transactions with the same gas price are included by order of timestamp.

This particular technique of guaranteeing transaction order, as illustrated in figure 7, would create instant competition between adversarial agents: as long as there is still a margin of profit to be made, another frontrunning bot is likely to spot the attack and attempt to frontrun the current attacker using the same technique, kicking off a phenomenon called a Priority Gas Auction (abbreviated as PGA) [18]. Besides polluting the entire Ethereum network with high gas fees for regular users, it is clear to see that this particular technique is not the most effective technique to guarantee transaction ordering.

[17]Go Ethereum (Ethereum client implementation): https://geth.ethereum.org

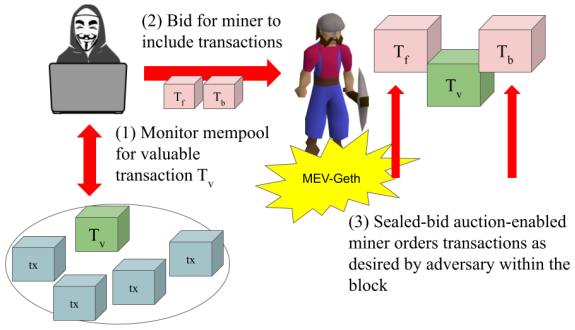

Figure 8: A diagram illustrating how adversaries guarantee transaction order by participating in sealed-bid auctions to compete for block space.

```
1  const frontrunTx = await uniswapV2Router.populateTransaction.
       swapExactETHForTokens(
2      /* ... */,
3      {
4          value: amountEthToSwap,
5          gasLimit: BigNumber.from(300000),
6          gasPrice: BigNumber.from(100).mul(BigNumber.from(10).pow
               (9)).add(tx.gasPrice),
7      }
8  )
9  const backrunTx = await uniswapV2Router.populateTransaction.
       swapExactTokensForETH(
10     /* ... */,
11     {
12         gasLimit: BigNumber.from(300000),
13         gasPrice: tx.gasPrice,
14     }
15 )
```

Listing 1: Example of a sandwich attack, written in Hardhat, utilising gas prices to guarantee transaction ordering. The gas prices used to frontrun and backrun can be found at lines 6 and 13, respectively. The variable `tx` in this example represents the valuable transation T_v to be exploited. (truncated, see GitHub repository for full example)

Since the inception of Flashbots, a research organisation formed with the purpose of solving the MEV Crisis, adversarial agents can participate in sealed-bid auctions by directly connecting to miners that support the Flashbots

MEV-Geth upgraded Ethereum client [19]. With this mechanism, adversarial agents are able to more effectively achieve their desired transaction orderings, without introducing network congestion by way of PGAs. This more advanced technique is described in figure 8.

4.4 Countering Arbitrageurs

Two protocols have recently been released to eliminate the dependency on transaction ordering in DEX smart contracts: Archerswap and CowSwap. These protocols exemplify the forefront of countermeasure techniques against TOD, mainly in an attempt to minimise MEV. Archerswap allows users to submit Uniswap and Sushiswap trades to the so-called Archer Relay, which negotiates for transactions to be included directly with co-operating miners, in order to bypass the mempool [24]. This protocol provides complete protection from sandwich attacks by putting regular users on a level playing field with adversaries that utilise direct miner connections. On the other hand, CowSwap uses Gnosis Protocol's batch auction mechanism to create a pseudo-order book system before sourcing liquidity from conventional DEXes. While CowSwap does not explicitly bypass the mempool, the privacy of each trade is preserved as many transactions are executed in batches by specialised third-parties (called *solvers*) with a single clearing price and tight slippage, effectively reducing sandwich attack risk.

This research proposes that the current best countermeasures to mitigating TOD vulnerability in Ethereum-based smart contracts are to adopt the following software design paradigms.

- Leverage off-chain computation and transaction batching to minimise any negative effects from being frontrun by adversaries.
- Provide a service to bypass the mempool, to protect smart contract users, by employing the same tactics that adversarial agents use, such as using Flashbots' sealed-bid auction mechanism.

While employing these design paradigms mitigates the effects of TOD, their adoption requires the evaluation of some trade-offs. Primarily, computations that are moved out of smart contracts onto off-chain services lose the property of being secured by the Ethereum network, and add a trust requirement to the party executing the off-chain computations. Additionally, leveraging off-chain services usually requires users to sign raw transactions, which is a bad security practice [24]. Table 2 shows a comparison of these design paradigms and their properties.

In addition to the two design paradigms previously mentioned, with the recent growth of layer-2 protocols on Ethereum, research has been started on commit-reveal schemes on ZK-rollups and optimistic rollups. Shadrach [23] suggests that it is possible to eliminate MEV through a system wherein block producers (rollup operators) must commit to including transactions in a known order prior to them being revealed. Some trade-offs have been identified; such as the ability for users to withhold revealing transactions and delaying the entire system.

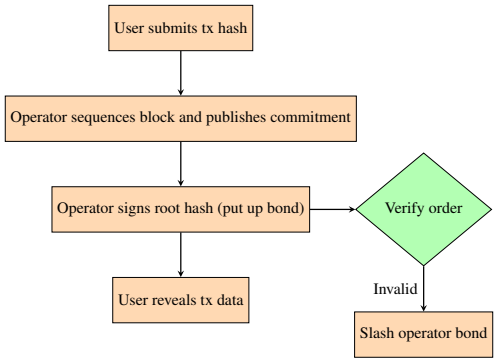

Figure 9: Commit-reveal scheme to prevent transaction-reordering for optimistic rollups [23]

Protocol	CowSwap	Archerswap
Design paradigm	Off-chain settlement, tx batching	Bypass mempool
Security model	Not trustless (off-chain)	Not trustless (off-chain), requires signing raw tx
Transaction privacy	Only under certain conditions	Complete privacy
TOD vulnerability	Almost complete protection	Complete protection

Table 1: Comparison of protocols countering transaction-ordering dependency in smart contracts

4.5 The Future of Frontrunning

Even with the dawn of Ethereum 2.0 and the network's transition from proof-of-work to proof-of-stake, transaction-ordering dependency, and in turn MEV, will remain a security and privacy vulnerability in Ethereum-based smart contracts [11]. As Vitalik mentioned in this interview, the ecosystem's best countermeasure to mitigate the consensus-layer instability and centralisation of block production arising from TOD and MEV aligns with the Flashbots project; which is to firewall the centralisation instead of trying to eliminate it. With the Flashbots MEV-Geth project, a marketplace is created between "dumb" miners (or validators in the case of proof-of-stake) and so-called *searchers* who bundle transactions and bid for block space. In this system, the consensus layer (miners and validators) are separated from the arbitrageurs, which prevents permissioned communication infrastructure between miners and arbitrageurs. Despite this countermeasure implemented by Flashbots and the upgrade to proof-of-stake on Ethereum 2.0, it is believed that frontrunning and MEV will still exist [10], and so the search must continue to find better countermeasures against TOD.

5 Oracle Manipulation

In computer science, an oracle is a black box device that provides a source of truth that can be used by other systems; such as providing an expected result for a test case. In the context of Ethereum smart contracts, oracles provide a source of truth for information that is external to the calling contract.

Design Paradigm	Off-chain computation & tx batching	Bypass mempool	Commit-reveal on rollups
Trade-offs	Loses security of Ethereum network	Requires miner co-operation	Users can delay the system
Effectiveness vs TOD	Minimises risk of MEV	Complete solution	Complete solution

Table 2: Comparison of proposed smart contract design paradigms as countermeasures to TOD

For example, a smart contract that allows users to bet on the next president of the United States would require an oracle to confirm the outcome of the elections in order to settle payments [13] - this is called an off-chain oracle. There also exist on-chain oracles that provide information using data only available on the blockchain.

As coined by Fridman and Nazarov [12], in the world of DeFi, *hybrid smart contracts* replace traditional contractual agreements found in the global financial system. This new format of contractual agreements offers two powerful advantages over the traditional contractual agreements:

- Transparency - As this new format of contracts are deployed as publicly-viewable code on the blockchain, anyone can inspect the inner workings of any financial products that they may have assets in; and

- Control - Any participant in DeFi interacting with these smart contracts are in control of their own assets, unlike in the traditional financial system where assets are controlled by banks, brokers, and other financial institutions.

These hybrid smart contracts rely on oracles as *bridges* between Ethereum and the real world, thus cementing oracles as essential building blocks for smart contract development in the DeFi landscape.

A common use case for an oracle is for a smart contract to receive a price feed of some asset. The price of this asset is then used in calculations by the smart contract for trading, lending, or borrowing. However, getting this price information accurately, consistently, and reliably is not as easy as it may seem. Depending on the architecture of the oracle system, and the way in which the smart contract uses the price information, it is possible for adversaries to manipulate this source of truth to maliciously redirect funds to themselves. Due to the immutable nature of the blockchain, loss of funds are irreversible [2].

5.1 Malicious Manipulation

The birth of DeFi in the Summer of 2020 saw a shift in smart contract exploits from typical re-entrancy attacks to more sophisticated attacks manipulating entire markets by way of flash loans. By December 2020, the total amount of funds irreversibly lost to these DeFi exploits amounted to approximately $100M (USD), an increase of 100%, from $50M from the beginning of the year [2]. It is clear that the presence of oracle manipulation vulnerabilities in Ethereum-based smart contracts poses a significant security risk to user funds locked in smart contracts. This research addresses this issue by empowering smart contract engineers with an aggregation of countermeasures against oracle manipulation attacks that have been demonstrably proven to be effective.

```
contract SimpleLendingProtocol is ILendingProtocol {
    // ...
    /**
     * Calculates the mid price of ETH (in DAI) from calculating the liquidity reserves.
     * This is vulnerable to instantaneous price movements as we rely solely on Uniswap
     * as an on-chain price oracle.
     */
    function getEthPrice() public view returns (uint256) {
        (uint112 daiReserve, uint112 ethReserve, ) =
            IUniswapV2Pair(daiEthPairAddress).getReserves();

        return FixedPoint.fraction(daiReserve, ethReserve).decode();
    }
    // ...
}
```

Listing 2: A smart contract that is vulnerable to an instantaneous oracle manipulation attack. (truncated, see GitHub repository for full example)

Listing 2 shows a concrete example of a (vulnerable) lending protocol that uses a Uniswap liquidity pool as a price oracle. It is clear to see in this example that the price returned by the getEthPrice function is directly calculated from the reserves in the liquidity pool. Thus, it follows that an adversary would be able to manipulate the ETH price in this smart contract by simply manipulating the reserves of this particular Uniswap liquidity pool (i.e., the price oracle). Listing 3 exemplifies this type of attack.

```
contract SimpleOracleAttack is Ownable {
    // ...
    function attack() external {
        // 1. Swap DAI -> ETH (This increases the ETH price on Uniswap)
        // ...
        univ2Router.swapExactTokensForETH(daiToSell, /* ... */);
        // 2. Deposit ETH (_NOT_ the ETH we just swapped) into lending protocol
        // ...
        lendingProtocol.depositCollateral{value: ethDeposit}();
        // 3. Borrow max DAI according to new mid-price that this lending protocol thinks it's at
        uint256 newEthPrice =
            (daiReserve + daiSold) / (wethReserve - wethBought);
        uint256 maxBorrow = (100 * newEthPrice * ethDeposit) / 150;
        lendingProtocol.borrowDai(maxBorrow);
        // At this point, we have more DAI than we started with
        // 4. Swap back ETH -> DAI
        // ...
        univ2Router.swapExactETHForTokens{value: wethBought}(
            (daiSold * 99) / 100, /* ... */);
        // ...
    }
}
```

Listing 3: A smart contract that manipulates the price oracle of a lending protocol to borrow more assets than is possible. (truncated, see GitHub repository for full example)

Indeed, oracle manipulation attacks take on this common form, as similarly described by CertiK [6]. Figure 10 presents a diagram depicting the general process of an oracle manipulation exploit. More explicitly, the features of this common form of attack are enumerated as follows.

1. Find a smart contract that uses an on-chain price oracle

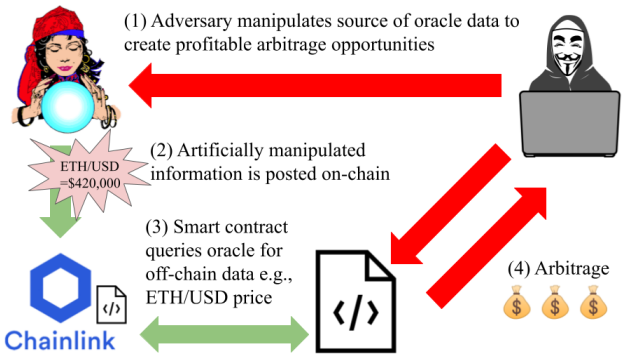

Figure 10: A diagram depicting the general process of an oracle manipulation exploit; involving the adversary, and the smart contract to be arbitraged which queries an oracle (Chainlink in this example).

(usually an AMM DEX like Uniswap or SushiSwap) as the source of truth for the price of token A against token B (or vice versa).

2. Utilise an undercollateralised flash loan to access a large amount of token A.
3. Sell token A on the AMM DEX in exchange for token B. This increases the reserves of token A while decreasing the reserves of token B in the liquidity pair, thus changing the price of token A against token B.
4. Drain the smart contract of its assets via a function that relies on the manipulated price oracle. Usually, this is a borrow function that now allows the adversary to take out more of an asset than would normally be possible.
5. Repay the flash loan plus some interest, and pocket the profit.

5.2 Resisting Oracle Manipulators

This research proposes that the best countermeasures against oracle manipulation attacks in Ethereum-based smart contracts are to adopt the following software paradigms when consulting oracles.

- Use TWAPs (Time-weighted Average Price) when consulting a price oracle. This simple, but effective, algorithm is employed by traders in the traditional financial markets and has been proven to provide resistance against flash loans attacks. An example of such an oracle is illustrated in listing 4.
- Consult M-of-N reporters within the oracle architecture, selecting the *best* M responses out of N reporters. Indeed, this is the approach taken by the MakerDAO [25] and Chainlink [22] protocols (with some variations).

These paradigms have been used successfully by major protocols such as MakerDAO and Compound [22] to guard against price oracle manipulation via flash loan attacks. However, as with the countermeasures previously proposed in section 4, they come with trade-offs. Using TWAPs is a very specialised solution that works only in the case of price feed oracles, and may prevent the smart contract from reacting quickly to price changes during times of high volatility in the market [22]. This countermeasure could theoretically be generalised for other types of oracles that provide numerical data. On the other hand, consulting M-of-N reporters can be applied to many types of oracles, at the expense of delegating trust to third parties. This countermeasure could be compared to leveraging off-chain computations in section 4.4: they both add a trust requirement to off-chain parties, which is counter to the trustless property of the Ethereum network.

```
1  contract DaiWethTwapPriceOracle {
2      uint256 public constant TWAP_PERIOD = 4 hours;
3      struct Observation {
4          uint256 timestamp;
5          uint256 cumPrice0;
6          uint256 cumPrice1;
7      }
8      uint8 private constant OBS_LEN = 6;
9      Observation[] private observations;
10
11     function updateTwap() public {
12         // ...
13         uint256 newCumulativePrice1 =
14             latestObservation.cumPrice1 +
15                 uint256(FixedPoint.fraction(daiReserve,
                       ethReserve).decode()) *
16                 timeElapsed;
17         recordObservation(Observation(timestamp,
                newCumulativePrice1));
18     }
19
20     function getEthTwap() external view returns (uint256) {
21         uint256 sumTwap = 0;
22         for (uint256 i = obs_head; i < (obs_head + OBS_LEN) - 1;
                i++) {
23             // ...
24             sumTwap += avgPrice;
25         }
26         return sumTwap / (OBS_LEN - 1);
27     }
28 }
```

Listing 4: A price oracle smart contract that records observations periodically and returns a time-weighted average price. The updateTwap() method is invoked periodically by keepers to record cumulative prices, and the getEthTwap() method returns the time-weighted average price. (truncated, see GitHub repository for full example)

5.3 Beyond Flash Loans

As the Ethereum network grows and smart contracts become ever more reliant on oracles for different kinds of information, new oracle manipulation attacks will undoubtedly emerge in the future. Thus, in addition to the adoption of the software paradigms presented in this research, *functional audits* from reputable smart contract security specialists (such as samczsun[18], Trail of Bits[19], and CertiK[20], among others) should be performed as part of the smart contract testing process. Regular audits usually include coverage on typical EVM pitfalls like re-entrancy and arithmetic issues which are largely detectable through the use of static analysers, while

[18] https://samczsun.com
[19] https://www.trailofbits.com/
[20] https://www.certik.io

functional audits would include more thorough coverage on smart contract logic; such as how usage of oracles could potentially be manipulated.

6 Conclusion

During the course of this research, from literary surveys, it was found that there exist many categories of smart contract vulnerabilities [16]. Some vulnerabilities, such as re-entrancy and unchecked returns, are preventable from exploit through the use of static analysers. Other approaches such as *Sereum* tackle re-entrancy by implementing a taint engine in a modified geth client [21]. A comparison of these vulnerabilities is shown in 3. This research focused on more complex smart contract vulnerabilities that present the highest risks to the Ethereum ecosystem as of 2021: transaction-ordering dependency and oracle manipulation. It was shown, through both literary surveys and through evaluation of experiments that implemented attacks on these vulnerabilities, that these vulnerabilities require the establishment and adherence to software design paradigms specific to Ethereum-based smart contracts. Future work should focus on enhancing the software design paradigms that have been presented as countermeasures in this research, in particular to improving the trade-offs in trustlessness and decentralisation.

7 Acknowledgments

The author would like to thank Prof. Dr. Kaitai Liang for supervising this research and providing valuable feedback and direction. The author would also like to thank Cheyenne Slager, Sara Op den Orth, Cathrine Paulsen, and Rado Stefanov for their numerous contributions including, but not limited to; peer reviews, project organisation, and moral support.

8 Responsible Research

8.1 Ethical Considerations for Releasing Vulnerability Implementations

While it was one of our main goals to contribute realistic implementations of smart contract vulnerability attacks, we realise that having these implementations publicly available might raise concerns about enabling new adversaries to participate in these attacks. However, the attacks implemented by this research show only the necessary code to illustrate smart contract vulnerabilities locally. For example, in a real-world frontrunning attack on Ethereum Mainnet, an adversary would be competing with much more advanced attackers, many with direct connections to miners - and so would not be able to use the attack provided by this research effectively without major modifications. We feel that this is a good trade-off in order to provide a useful base for others to develop more advanced countermeasures in the future.

8.2 Reproducibility of Results

According to research done by Baker [4], greater than half of researchers surveyed have failed when attempting to reproduce their own experiments, with an even greater number of researchers failing when attempting to produce other scientists' experiments. This has been dubbed the *reproducibility crisis*. Out of 1576 scientists surveyed in the study, more than 40% said that missing methods and code was a factor that contributed to irreproducible research.

As reproducibility was another of our main goals in this research, a lot of thought has been taken into robust experimental design, and best-practices have been implemented from the software engineering discipline. These practices include:

- Providing a thorough test suite with continuous integration;
- Delivering the codebase through a distributed version control system (Git);
- Using package managers and public package repositories for all external dependencies;
- Utilising common tooling favoured by the Ethereum community (Hardhat);
- Providing automated scripts and explicit documentation on how to build and run the attacks and countermeasures; and
- Including a Dockerfile definition and prebuilt Docker images.

The adoption of these software engineering practices should guarantee that any future researcher is able to reproduce the experiments described in this research. Prebuilt Docker images have also been provided as a last resort, in case the tooling required to build the experiment becomes unavailable in the future.

References

[1] Hayden Adams, Noah Zinsmeister, and Dan Robinson. *Uniswap v2 Core. 2020*. URL: https://uniswap.org/whitepaper.pdf.

[2] Zaryab Afser. *How $100M Got Stolen From DeFi in 2021: Price Oracle Manipulation And Flash Loan Attacks Explained*. URL: https://hackernoon.com/how-dollar100m-got-stolen-from-defi-in-2021-price-oracle-manipulation-and-flash-loan-attacks-explained-3n6q33r1.

[3] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. "A survey of attacks on ethereum smart contracts (sok)". In: *International conference on principles of security and trust*. Springer. 2017, pp. 164–186.

[4] Monya Baker. "Reproducibility crisis". In: *Nature* 533.26 (2016), pp. 353–66.

[5] V Buterin. *A Next Generation Smart Contract & Decentralized Application Platform*. 2009.

[6] CertiK. *Understanding Security Risks in DeFi: CertiK Foundation Blog*. URL: https://www.certik.org/blog/understanding-security-risks-in-defi.

[7] Philip Daian et al. "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges". In: *arXiv preprint arXiv:1904.05234* (2019).

Table 3: Comparison of vulnerabilities.

Vulnerability	Threat level	Possible implications	Countermeasures
Arithmetic overflow	EVM	contract malfunction, loss of funds	static analysis tools, geth modifications
Re-entrancy	EVM	contract malfunction, loss of funds	static analysis tools
TOD	consensus layer	consensus-layer instability, loss of funds	off-chain computation, bypass mempool
Oracle Manipulation	application layer	loss of funds	TWAPs, M-of-N reporters

[8] Eric Decourcy. *Protecting Against Front-Running and Transaction Reordering*. Sept. 2019. URL: https://forum.openzeppelin.com/t/protecting-against-front-running-and-transaction-reordering/1314.

[9] *DeFi Pulse: The DeFi Leaderboard: Stats, Charts and Guides*. URL: https://defipulse.com.

[10] William Foxley. *Yes, Front-Running Will Still Exist on Ethereum 2.0*. Mar. 2021. URL: https://www.coindesk.com/front-running-will-still-exist-ethereum-2-0-mev.

[11] Lex Fridman and Vitalik Buterin. *Vitalik Buterin: Ethereum 2.0 — Lex Fridman Podcast #188*. YouTube. May 2021. URL: https://www.youtube.com/watch?v=XW0QZmtbjvs.

[12] Lex Fridman and Sergey Nazarov. *Sergey Nazarov: Chainlink, Smart Contracts, and Oracle Networks — Lex Fridman Podcast #181*. YouTube. May 2021. URL: https://www.youtube.com/watch?v=TPXTmVdlyoc.

[13] Pierre Grimaud et al. *Oracles*. Apr. 2021. URL: https://ethereum.org/en/developers/docs/oracles/.

[14] Hasu. *Understanding Automated Market-Makers, Part 1: Price Impact*. Apr. 2021. URL: https://research.paradigm.xyz/amm-price-impact.

[15] Bin Hu et al. "A comprehensive survey on smart contract construction and execution: paradigms, tools, and systems". In: *Patterns* 2.2 (2021), p. 100179.

[16] Zulfiqar Ali Khan and Akbar Siami Namin. "A Survey on Vulnerabilities of Ethereum Smart Contracts". In: *arXiv preprint arXiv:2012.14481* (2020).

[17] Mario. *LobsterDAO*. URL: https://t.me/lobsters_chat/238257.

[18] Charlie Noyes. *MEV and Me*. Feb. 2021. URL: https://research.paradigm.xyz/MEV.

[19] Alex Obadia. *Flashbots: Frontrunning the MEV Crisis*. Nov. 2020. URL: https://medium.com/flashbots/frontrunning-the-mev-crisis-40629a613752.

[20] Dan Robinson. *Ethereum Is a Dark Forest*. Feb. 2021. URL: https://medium.com/@danrobinson/ethereum-is-a-dark-forest-ecc5f0505dff.

[21] Michael Rodler et al. "Sereum: Protecting existing smart contracts against re-entrancy attacks". In: *arXiv preprint arXiv:1812.05934* (2018).

[22] samczsun. *So you want to use a price oracle*. Nov. 2020. URL: https://samczsun.com/so-you-want-to-use-a-price-oracle/.

[23] Samuel Shadrach. *Off-chain commitments for rollups*. Apr. 2021. URL: https://ethresear.ch/t/off-chain-commitments-for-rollups/8993/4.

[24] Caleb Sheridan. *Scared of MEV? We share our proof of concept – Archer Swap – for @Uniswap and @SushiSwap traders to avoid being front-run.* Apr. 2021. URL: https://twitter.com/calebsheridan/status/1384811452402442240?lang=en.

[25] MakerDAO Community Development Team. *How it Works*. URL: https://community-development.makerdao.com/en/learn/Oracles/how-it-works/.

[26] Uniswap Team. *Introducing Uniswap V3*. Mar. 2021. URL: https://uniswap.org/blog/uniswap-v3.

[27] Liyi Zhou et al. "High-Frequency Trading on Decentralized On-Chain Exchanges". In: *arXiv preprint arXiv:2009.14021* (2020).