Zellic

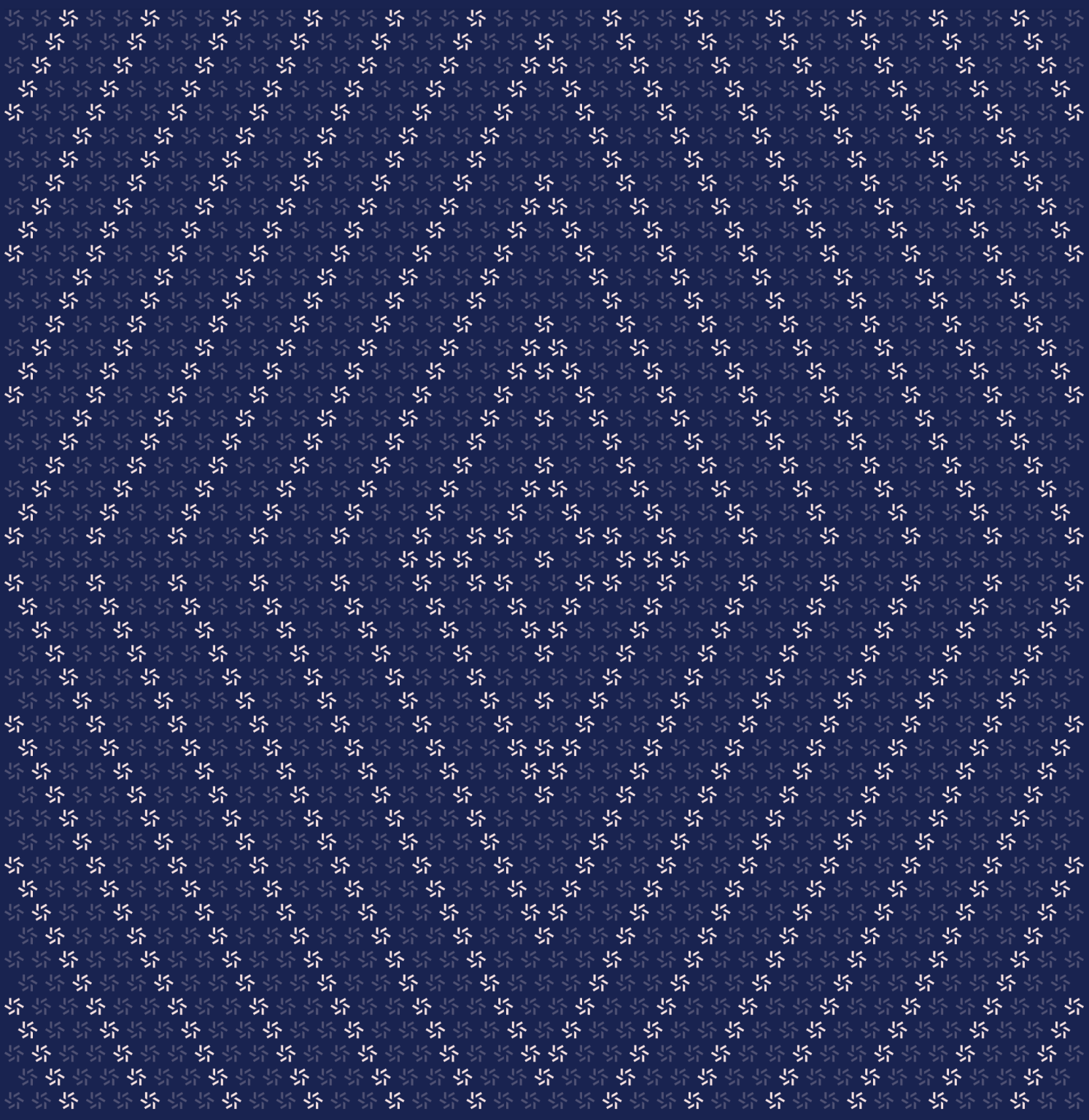
Prepared for
Mick de Graaf
Euler Labs

Prepared by
Aaron Esau
Jinseo Kim
Zellic

February 6, 2024

Fee Flow

Smart Contract Security Assessment



Contents

About Zellic

Zellic is a vulnerability research firm with deep expertise in blockchain security. We specialize in EVM, Move (Aptos and Sui), and Solana as well as Cairo, NEAR, and Cosmos. We review L1s and L2s, cross-chain protocols, wallets and applied cryptography, zero-knowledge circuits, web applications, and more.

Prior to Zellic, we founded the #1 CTF (competitive hacking) team ↗ worldwide in 2020, 2021, and 2023. Our engineers bring a rich set of skills and backgrounds, including cryptography, web security, mobile security, low-level exploitation, and finance. Our background in traditional information security and competitive hacking has enabled us to consistently discover hidden vulnerabilities and develop novel security research, earning us the reputation as the go-to security firm for teams whose rate of innovation outpaces the existing security landscape.

For more on Zellic's ongoing security research initiatives, check out our website zellic.io ↗ and follow @zellic_io ↗ on Twitter. If you are interested in partnering with Zellic, contact us at hello@zellic.io ↗.

1. Overview

1.1. Executive Summary

Zellic conducted a security assessment for Euler Labs on February 6th, 2024. During this engagement, Zellic reviewed Fee Flow's code for security vulnerabilities, design issues, and general weaknesses in security posture.

1.2. Goals of the Assessment

In a security assessment, goals are framed in terms of questions that we wish to answer. These questions are agreed upon through close communication between Zellic and the client. In this assessment, we sought to answer the following questions:

- Do auctions initialize properly?
- Are there edge-case bugs for extreme values?
- Is it possible for the auctions to be locked up (i.e., revert on call to `buy`)?

1.3. Non-goals and Limitations

We did not assess the following areas that were outside the scope of this engagement:

- Front-end components
- Infrastructure relating to the project
- Key custody

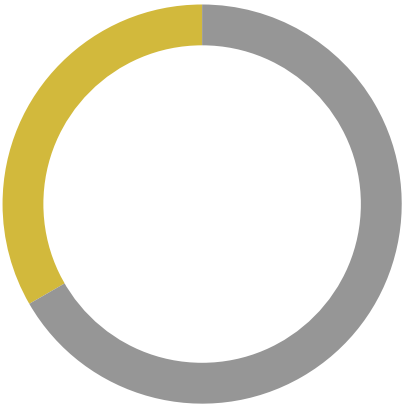
Due to the time-boxed nature of security assessments in general, there are limitations in the coverage an assessment can provide.

1.4. Results

During our assessment on the scoped Fee Flow contracts, we discovered three findings. No critical issues were found. One finding was of medium impact and the other findings were informational in nature.

Breakdown of Finding Impacts

Impact Level	Count
Critical	0
High	0
Medium	1
Low	0
Informational	2



2. Introduction

2.1. About Fee Flow

Fee Flow is a contract that runs continuous Dutch auctions to sell a flow of fees of multiple assets.

2.2. Methodology

During a security assessment, Zellic works through standard phases of security auditing, including both automated testing and manual review. These processes can vary significantly per engagement, but the majority of the time is spent on a thorough manual review of the entire scope.

Alongside a variety of tools and analyzers used on an as-needed basis, Zellic focuses primarily on the following classes of security and reliability issues:

> **Basic coding mistakes.** Many critical vulnerabilities in the past have been caused by simple, surface-level mistakes that could have easily been caught ahead of time by code review. Depending on the engagement, we may also employ sophisticated analyzers such as model checkers, theorem provers, fuzzers, and so on as necessary. We also perform a cursory review of the code to familiarize ourselves with the contracts.
>
> **Business logic errors.** Business logic is the heart of any smart contract application. We examine the specifications and designs for inconsistencies, flaws, and weaknesses that create opportunities for abuse. For example, these include problems like unrealistic tokenomics or dangerous arbitrage opportunities. To the best of our abilities, time permitting, we also review the contract logic to ensure that the code implements the expected functionality as specified in the platform's design documents.
>
> **Integration risks.** Several well-known exploits have not been the result of any bug within the contract itself; rather, they are an unintended consequence of the contract's interaction with the broader DeFi ecosystem. Time permitting, we review external interactions and summarize the associated risks: for example, flash loan attacks, oracle price manipulation, MEV/sandwich attacks, and so on.
>
> **Code maturity.** We look for potential improvements in the codebase in general. We look for violations of industry best practices and guidelines and code quality standards. We also provide suggestions for possible optimizations, such as gas optimization, upgradability weaknesses, centralization risks, and so on.

For each finding, Zellic assigns it an impact rating based on its severity and likelihood. There is no hard-and-fast formula for calculating a finding's impact. Instead, we assign it on a case-by-case basis based on our judgment and experience. Both the severity and likelihood of an issue affect its impact. For instance, a highly severe issue's impact may be attenuated by a low likelihood. We assign the following impact ratings (ordered by importance): Critical, High, Medium, Low, and Informational.

Zellic organizes its reports such that the most important findings come first in the document, rather than being strictly ordered on impact alone. Thus, we may sometimes emphasize an "Informational"

finding higher than a “Low” finding. The key distinction is that although certain findings may have the same impact rating, their *importance* may differ. This varies based on various soft factors, like our clients’ threat models, their business needs, and so on. We aim to provide useful and actionable advice to our partners considering their long-term goals, rather than a simple list of security issues at present.

2.3. Scope

The engagement involved a review of the following targets:

Fee Flow Contracts

Repository	https://github.com/euler-xyz/fee-flow ↗
Version	fee-flow: `a16608ef753194491b270ca2616cc2189f22aca1`
Programs	• FeeFlowController • MinimalEVCClient
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

Zellic was contracted to perform a security assessment with two consultants for a total of two person-days. The assessment was conducted over the course of one calendar day.

Contact Information

The following project manager was associated with the engagement:

Chad McDonald
Engagement Manager
chad@zellic.io

The following consultants were engaged to conduct the assessment:

Aaron Esau
Engineer
aaron@zellic.io

Jinseo Kim
Engineer
jinseo@zellic.io

2.5. Project Timeline

The key dates of the engagement are detailed below.

February 6, 2024	Kick-off call
February 6, 2024	Start of primary review period
February 6, 2024	End of primary review period

3. Detailed Findings

3.1. Front-running of the buy function

Target	FeeFlowController		
Category	Business Logic	**Severity**	Medium
Likelihood	Low	**Impact**	Medium

Description

The buy function initializes the next auction immediately after it processes the purchase.

```
function buy(address[] calldata assets, address assetsReceiver,
    uint256 deadline, uint256 maxPaymentTokenAmount)
    external nonReentrant returns(uint256 paymentAmount) {
    // ...

    // Setup new auction
    uint256 newInitPrice = paymentAmount * priceMultiplier
    / PRICE_MULTIPLIER_SCALE;

    if(newInitPrice > ABS_MAX_INIT_PRICE) {
        newInitPrice = ABS_MAX_INIT_PRICE;
    } else if(newInitPrice < minInitPrice) {
        newInitPrice = minInitPrice;
    }

    slot1Cache.initPrice = uint216(newInitPrice);
    slot1Cache.startTime = uint40(block.timestamp);

    // Write cache in single write
    slot1 = slot1Cache;

    // ...
}
```

An attacker can front-run the buy function call of a victim, forcing them to pay for the next auction with an increased price. Furthermore, because the attacker takes assets out first, the victim will not be able to receive the assets they expected.

Impact

By front-running the `buy` function call of a victim, an attacker can make the victim pay higher than the expected price and prevent the victim from receiving the expected assets.

It is possible to adjust the `maxPaymentTokenAmount` parameter, which is the upper limit of the payment amount, to mitigate this vulnerability. However, one should note that this mitigation can be nullified under certain circumstances. For instance, if the transaction from a user could not be included to the network promptly, the price of the current auction may decrease before the transaction is included so that the price of the next auction is below the value of the `maxPaymentTokenAmount` parameter.

Recommendations

Consider adding a unique, incrementing auction ID for each auction and checking it in the `buy` function in order to confirm that a user intends to purchase in the current running auction.

Remediation

This issue has been acknowledged by Euler Labs, and a fix was implemented in commit a42879c0 ↗.

3.2. Read-only reentrancy with `getPrice` during `buy`

Target	FeeFlowController		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The `buy` function calls the untrusted external contracts before it updates the state of the auction:

```
function buy(address[] calldata assets, address assetsReceiver,
    uint256 deadline, uint256 maxPaymentTokenAmount)
    external nonReentrant returns(uint256 paymentAmount) {
    // ...

    for(uint256 i = 0; i < assets.length; i++) {
        // Transfer full balance to buyer
        uint256 balance = ERC20(assets[i]).balanceOf(address(this));
        ERC20(assets[i]).safeTransfer(assetsReceiver, balance);
    }

    // ...

    slot1Cache.initPrice = uint216(newInitPrice);
    slot1Cache.startTime = uint40(block.timestamp);

    // Write cache in single write
    slot1 = slot1Cache;

    // ...
}
```

The `buy` function prevents the reentrancy by using the `nonReentrant` modifier. However, this still allows reentering to other functions that do not have the `nonReentrant` modifier, such as `getPrice`.

Impact

By exploiting reentrancy, on-chain contracts can be presented with the stale price from the return value of the `getPrice` function.

Recommendations

Consider transferring assets after updating the state variable.

Remediation

This issue has been acknowledged by Euler Labs, and a fix was implemented in commit `c789fcaa`↗.

3.3. Incorrect size for `ABS_MAX_INIT_PRICE`

Target	FeeFlowController		
Category	Coding Mistakes	**Severity**	Informational
Likelihood	N/A	**Impact**	Informational

Description

The following error message suggests that `ABS_MAX_INIT_PRICE` should be the maximum `uint216` value:

```
if(minInitPrice_ > ABS_MAX_INIT_PRICE) revert MinInitPriceExceedsUint216();
```

However, according to its definition, it holds the `uint192` maximum value:

```
uint256 constant public ABS_MAX_INIT_PRICE = type(uint192).max; // chosen so
    that initPrice * priceMultiplier does not exceed uint256
```

Impact

This finding simply documents an inconsistency in the code.

Recommendations

Consider renaming the error or changing the value of `ABS_MAX_INIT_PRICE`.

Remediation

This issue has been acknowledged by Euler Labs, and a fix was implemented in commit `8c2767f5` ↗.

4. Threat Model

This provides a full threat model description for various functions. As time permitted, we analyzed each function in the contracts and created a written threat model for some critical functions. A threat model documents a given function's externally controllable inputs and how an attacker could leverage each input to cause harm.

Not all functions in the audit scope may have been modeled. The absence of a threat model in this section does not necessarily suggest that a function is safe.

4.1. Module: FeeFlowController.sol

Function: `buy(address[] assets, address assetsReceiver, uint256 deadline, uint256 maxPaymentTokenAmount)`

Allows a user to buy assets by transferring payment tokens and receiving the assets.

Inputs

- `assets`
  - **Control**: Full.
  - **Constraints**: Must be a nonempty array of addresses of contracts with a callable `transfer(address,uint256)` function.
  - **Impact**: The addresses of the assets to be bought.
- `assetsReceiver`
  - **Control**: Full.
  - **Constraints**: None.
  - **Impact**: The address that will receive the bought assets.
- `deadline`
  - **Control**: Full.
  - **Constraints**: Must be a future timestamp.
  - **Impact**: The deadline timestamp for the purchase.
- `maxPaymentTokenAmount`
  - **Control**: Full.
  - **Constraints**: Must be greater than or equal to the payment amount.
  - **Impact**: The maximum amount of payment tokens the user is willing to spend.

Branches and code coverage

Intended branches

- Successfully executes purchase of fee tokens and calculates new price, capping at `ABS_MAX_INIT_PRICE`.
  - ☑ Test coverage

Negative behavior

- `assets` is empty.
  - ☑ Negative test
- `deadline` passed.
  - ☑ Negative test
- `paymentAmount` exceeds `maxPaymentTokenAmount`.
  - ☑ Negative test
- Function cannot reenter.
  - ☑ Negative test

5. Assessment Results

At the time of our assessment, the reviewed code was not deployed to the Ethereum Mainnet.

During our assessment on the scoped Fee Flow contracts, we discovered three findings. No critical issues were found. One finding was of medium impact and the other findings were informational in nature. Euler Labs acknowledged all findings and implemented fixes.

5.1. Disclaimer

This assessment does not provide any warranties about finding all possible issues within its scope; in other words, the evaluation results do not guarantee the absence of any subsequent issues. Zellic, of course, also cannot make guarantees about any code added to the project after the version reviewed during our assessment. Furthermore, because a single assessment can never be considered comprehensive, we always recommend multiple independent assessments paired with a bug bounty program.

For each finding, Zellic provides a recommended solution. All code samples in these recommendations are intended to convey how an issue may be resolved (i.e., the idea), but they may not be tested or functional code. These recommendations are not exhaustive, and we encourage our partners to consider them as a starting point for further discussion. We are happy to provide additional guidance and advice as needed.

Finally, the contents of this assessment report are for informational purposes only; do not construe any information in this report as legal, tax, investment, or financial advice. Nothing contained in this report constitutes a solicitation or endorsement of a project by Zellic.