Zellic

Prepared for

Bracket Labs Group SA

Prepared by
Nipun Gupta
Ayaz Mammadov
Zellic

April 19, 2024

Bracket Fi Escrow

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 Bracket Labs Group SA from April 16th and April 18th, 2024. During this engagement, Zellic reviewed Bracket Fi Escrow'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:

- Are there any issues in the validation of the Merkle tree proofs?
- Can funds be drained in the escrow contract?
- Does the escrow interact correctly with the contract?

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.

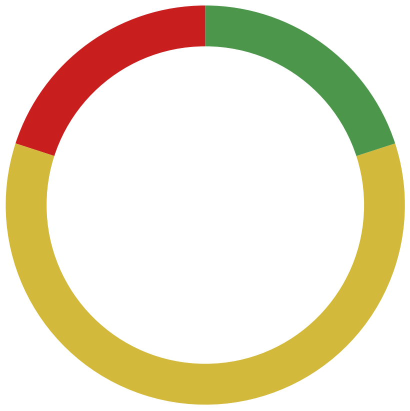
1.4. Results

During our assessment on the scoped Bracket Fi Escrow contracts, we discovered five findings. One critical issue was found. Three were of medium impact and one was of low impact.

Additionally, Zellic recorded its notes and observations from the assessment for Bracket Labs Group SA's benefit in the Discussion section (4. ↗).

Breakdown of Finding Impacts

Impact Level	Count
🟥 Critical	1
🟧 High	0
🟨 Medium	3
🟩 Low	1
⬜ Informational	0



2. Introduction

2.1. About Bracket Fi Escrow

Bracket Labs Group SA contributed the following description of Bracket Fi Escrow:

> Bracket's Escrow contracts aims to collect Liquid Staking Tokens (LSTs) in an escrow on both Layer 1 (L1) Ethereum and Layer 2 (L2) Arbitrum for a designated period in order to accumulate points for users until an escrow break event occurs.
>
> Users are permitted to deposit and withdraw their funds any time until the escrow is broken. Once escrow is broke, the funds will be used to mint brktETH, where they will serve as collateral, backing brktETH's value, users will be able to burn brktETH in exchange for the collateral any time they want.
>
> brktETH will then be deposited into the escrow contract, where user will be able to claim an amount of brktETH with the same value of their deposits through a merkle tree distribution.

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.

Finally, Zellic provides a list of miscellaneous observations that do not have security impact or are not directly related to the scoped contracts itself. These observations — found in the Discussion (4. ↗) section of the document — may include suggestions for improving the codebase, or general recommendations, but do not necessarily convey that we suggest a code change.

2.3. Scope

The engagement involved a review of the following targets:

Bracket Fi Escrow Contracts

Repository	https://github.com/bracket-fi/escrow_contract ↗
Version	escrow_contract: `cf207cc18097500a283a1f89a98d5de42523819a`
Programs	• BridgeEscrow.sol • EscrowBase.sol • MainEscrow.sol
Type	Solidity
Platform	EVM-compatible

2.4. Project Overview

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

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:

Nipun Gupta
Engineer
nipun@zellic.io ↗

Ayaz Mammadov
Engineer
ayaz@zellic.io ↗

2.5. Project Timeline

The key dates of the engagement are detailed below.

April 16, 2024	Kick-off call
April 16, 2024	Start of primary review period
April 18, 2024	End of primary review period

3. Detailed Findings

3.1. Reentrancy in withdrawals

Target	EscrowBase		
Category	Business Logic	Severity	Critical
Likelihood	High	Impact	Critical

Description

The function `withdraw` is responsible for the withdrawals of tokens from the escrow contract. The function first checks if there is enough balance in the `usersBalance` for the caller, then unwraps the token if `unwrap` is true, and finally decreases the amount from the `usersBalance` mapping and `totalStaked`.

If the rebase token is `ETH_ADDRESS`, the function will unwrap WETH to ETH and send the tokens to the user via the `msg.sender.call{value: amount}("");` call. In this function, the checks-effects-interactions pattern is broken, as the interaction (i.e., the transfer of the amount to the caller) is happening before the effects (i.e., the deduction of the amount). Furthermore, the amount is deducted in an `unchecked` block, which leads to underflow in the balance, and thus the function does not revert when the amount subtracted is more than the balance.

```
function withdraw(address token, uint256 amount, bool unwrap)
    external onlyNotBroke returns (uint256) {
    if (amount == 0) revert ZeroAmount();

    EscrowBaseStorage storage s = _getStorage();

    uint256 balance = s.usersBalance[msg.sender][token];
    if (amount > balance) revert NotEnoughAmountStaked(balance); //[1] <-
    checks

    uint256 finalAmt = amount;
    if (unwrap) {
        //...
        } else if (tokenInfo.rebase == ETH_ADDRESS) {
            _unwrapETH(token, amount);
            (bool success,) = msg.sender.call{value: amount}(""); //[2] <-
    interaction
            if (!success) revert ETHSendFailed();

    //...
    unchecked {
        s.usersBalance[msg.sender][token] = balance - amount; //[3] <- effects
```

```
            s.tokens[token].totalStaked -= amount;
        }
        //...
    }
```

Impact

An attacker could drain the WETH tokens available in the contract.

Recommendations

We recommend following the checks-effects-interactions pattern or adding a `nonReentrant` modifier to stop the reentrancy in this function.

Remediation

This issue has been acknowledged by Bracket Labs Group SA, and a fix was implemented in commit 6b7b4b1b ↗.

Bracket Labs Group SA provided the following response:

> Unchecking the balance subtraction was introduced while debugging an issue with rebase tokens and the development team forgot to remove it before the audit commit, and was originally meant to revert upon reentrancy. This was fixed immediately after the beginning of the audit, and nonReentrant modifier was added as an additional safeguard mechanism.

3.2. Centralization Risk

Target	MainEscrow, BridgeEscrow		
Category	Protocol Risks	**Severity**	High
Likelihood	Low	**Impact**	Medium

Description

The MainEscrow and BridgeEscrow contracts have certain centralization risks, which could result in a single point of failure or grant excessive control over the tokens stored in the escrow to a single entity.

For example `withdrawEscrow` allows owner to withdraw all the tokens from the escrow

```
function withdrawEscrow(address[] calldata tokens)
    external onlyOwner onlyBroke {
        uint256 length = tokens.length;

        for (uint256 i; i < length; ++i) {
            IERC20 token = IERC20(tokens[i]);
            uint256 balance = token.balanceOf(address(this));
            if (balance != 0) {
                token.safeTransfer(msg.sender,
    token.balanceOf(address(this)));
            }
        }
    }
```

and the bridge functions in BridgeEscrow allow owner to transfer tokens to any address on the L2.

```
function bridgeTokenArb(address token, address arbEscrow, uint256 amount,
    uint256 maxGas, uint256 gasPrice)
        external
        onlyOwner
        onlyBroke
    {
        IERC20(token).safeIncreaseAllowance(address(bridgeRouter), amount);
        bridgeRouter.outboundTransferCustomRefund(token, msg.sender, arbEscrow,
    amount, maxGas, gasPrice, bytes(""));
    }
```

```
function bridgeTokenConnext(address token, address arbEscrow,
 uint256 amount, uint256 slippage, uint256 relayerFee)
    external
    onlyOwner
    onlyBroke
{
    IERC20(token).safeIncreaseAllowance(address(renzoLockbox), amount);

    IXERC20 xToken = renzoLockbox.XERC20();
    renzoLockbox.deposit(amount);

    IERC20(address(xToken)).safeIncreaseAllowance(address(connext),
 amount);
    connext.xcall{value: relayerFee}(
        1634886255, // _destination: Domain ID of the destination chain
        arbEscrow, // _to: address receiving the funds on the destination
        address(xToken), // _asset: address of the token contract
        msg.sender, // _delegate: address that can revert or forceLocal on
 destination
        amount, // _amount: amount of tokens to transfer
        slippage, // _slippage: the maximum amount of slippage the user
 will accept in BPS (e.g. 30 = 0.3%)
        bytes("") // _callData: empty bytes because we're only sending funds
    );
}
```

Impact

If the owner's keys are compromised, the impact on the protocol could be significant as the contract allows the owner to directly transfer user's assets to another address.

Recommendations

To address this risk, and increase user confidence and security, we recommend implementing measures that remove reliance on a single point of failure. Here are a few recommendations to do that:

- Utilize a multi-signature address wallet with multiple signers. This approach would prevent an attacker from causing economic harm if a private key were compromised.
- Secure critical functions behind a timelock. This would provide users with sufficient time to withdraw funds from the protocol if any malicious executions are scheduled.
- Provide documentation on privileged functions so that users are aware of the potential risks when depositing their tokens into the protocol.

Remediation

This issue has been acknowledged by Bracket Labs Group SA.

Bracket Labs Group SA provided the following response:

> Because brktETH will be actively developed during the escrow phase, a certain level of centralization is required as assets need to be exchanged for brktETH once the escrow breaks. However, privileged actions can only be performed by the owner, which will be a big multisig to increase decentralization. Moreover, assets cannot be pulled before the escrow breaks, and users are free to deposit and withdraw before then. Once the escrow breaks, the assets will be exchanged for brktETH and users will be able to claim brktETH in the escrow which will hold an equal value to that of their assets at the moment of break.

3.3. Nonpayable `bridgeTokenConnext` function

Target	BridgeEscrow		
Category	Coding Mistakes	**Severity**	Low
Likelihood	High	**Impact**	Low

Description

The function `bridgeTokenConnext`, which is used to bridge tokens via connext bridge, calls the following function with the value passed as `relayerFee`:

```
function bridgeTokenConnext(address token, address arbEscrow, uint256 amount,
    uint256 slippage, uint256 relayerFee)
    external
    onlyOwner
    onlyBroke
{
    //...
    connext.xcall{value: relayerFee}(
        1634886255, // _destination: Domain ID of the destination chain
        arbEscrow, // _to: address receiving the funds on the destination
        address(xToken), // _asset: address of the token contract
        msg.sender, // _delegate: address that can revert or forceLocal on
    destination
        amount, // _amount: amount of tokens to transfer
        slippage, // _slippage: the maximum amount of slippage the user will
    accept in BPS (e.g. 30 = 0.3%)
        bytes("") // _callData: empty bytes because we're only sending funds
    );
}
```

While the function calls `connext.xcall` with some value, `bridgeTokenConnext` is not marked as `payable`. Therefore, the `relayerFee` would be subtracted from the ETH balance available in the contract (i.e., the user funds). Adding a `payable` keyword would allow ETH to be sent to this function call.

Impact

The `relayerFee` sent to connext bridge would be deducted from the user funds.

Recommendations

We recommend making the function `payable`. Furthermore, we also recommend adding a check that the `msg.value` passed to the function is equal to the `relayerFee`.

Remediation

This issue has been acknowledged by Bracket Labs Group SA, and a fix was implemented in commit `e2377e01` ↗.

3.4. Nonpayable `bridgeTokenArb` function

Target	BridgeEscrow		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The function `outboundTransferCustomRefund` called in `bridgeTokenArb` is a payable function, while `bridgeTokenArb` is not. Therefore, ETH could not be sent in the `outboundTransferCustomRefund` call.

```
function bridgeTokenArb(address token, address arbEscrow, uint256 amount,
    uint256 maxGas, uint256 gasPrice)
    external
    onlyOwner
    onlyBroke
{
    IERC20(token).safeIncreaseAllowance(address(bridgeRouter), amount);
    bridgeRouter.outboundTransferCustomRefund(token, msg.sender, arbEscrow,
    amount, maxGas, gasPrice, bytes(""));
}
```

The `msg.value` passed along this function is used as the gas fees on the L2 after the tokens are bridged. If this value is not sent, the gas passed along would be `0`, and hence the bridging might fail.

Impact

Bridging of tokens might fail due to no gas fee on the L2.

Recommendations

We recommend making the function `payable` and sending the `msg.value` to the function call `outboundTransferCustomRefund`.

Remediation

This issue has been acknowledged by Bracket Labs Group SA, and a fix was implemented in commit `9d8ba151` ↗.

3.5. Allowance given to incorrect address

Target	BridgeEscrow		
Category	Coding Mistakes	**Severity**	Medium
Likelihood	High	**Impact**	Medium

Description

The function `bridgeTokenArb` first increases the allowance of `bridgeRouter` by the value `amount` and then calls `outboundTransferCustomRefund` to start the bridging of the tokens.

```
function bridgeTokenArb(address token, address arbEscrow, uint256 amount,
    uint256 maxGas, uint256 gasPrice)
    external
    onlyOwner
    onlyBroke
{
    IERC20(token).safeIncreaseAllowance(address(bridgeRouter), amount);
    bridgeRouter.outboundTransferCustomRefund(token, msg.sender, arbEscrow,
    amount, maxGas, gasPrice, bytes(""));
}
```

As per the Arbitrum Docs ↗, the approval of the tokens should be given to the L1ERC20Gateway and not the IL1GatewayRouter (or `bridgeRouter`) contract.

The correct address of the L1ERC20Gateway could be retrieved by calling the method `getGateway` function in the IL1GatewayRouter contract.

Impact

Bridging of tokens will fail due to the incorrect allowance.

Recommendations

We recommend calling `safeIncreaseAllowance` on the correct contract address. The correct address can be retrieved by calling `getGateway` as mentioned above.

Remediation

This issue has been acknowledged by Bracket Labs Group SA, and a fix was implemented in commit `9d8ba151` ↗.

4. Discussion

The purpose of this section is to document miscellaneous observations that we made during the assessment. These discussion notes are not necessarily security related and do not convey that we are suggesting a code change.

4.1. Multiple Merkle tree leaves

In the case of a user having multiple Merkle tree leaves, they would not be able to claim all the tokens they have proofs for, as the interaction with the claim mapping would result in the claimed amount subtracting the amount in the second leaf.

The team stated, however, that the Merkle tree will only have one leaf per user and that the claimable mapping is used in case a redistribution of tokens has to be made.

5. 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.

5.1. Module: BridgeEscrow.sol

Function: **`bridgeTokenArb(address token, address arbEscrow, uint256 amount, uint256 maxGas, uint256 gasPrice)`**

The function is called by the owner after the break timestamp to bridge tokens to the L2.

Inputs

- `token`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Address of the token contract.
- `arbEscrow`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Address receiving the funds on the destination.
- `amount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Amount of tokens to transfer.
- `maxGas`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Max gas deducted from user's L2 balance to cover the execution in L2.
- `gasPrice`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Gas price for the execution in L2.

Branches and code coverage

Intended branches

- Increases the allowance of the `bridgeRouter` contract and starts the bridging of the tokens.
    - ☐ Test coverage

Negative behavior

- N/A.

Function call analysis

- `SafeERC20.safeIncreaseAllowance(IERC20(token), address(this.bridgeRouter), amount)`
    - **What is controllable?** `token` and `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.
- `this.bridgeRouter.outboundTransferCustomRefund(token, msg.sender, arbEscrow, amount, maxGas, gasPrice, bytes(""))`
    - **What is controllable?** `token`, `msg.sender`, `arbEscrow`, `amount`, `maxGas`, and `gasPrice`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.

Function: `bridgeTokenConnext(address token, address arbEscrow, uint256 amount, uint256 slippage, uint256 relayerFee)`

The function is called by the owner after the break timestamp to bridge Renzo restaked ETH tokens to the L2.

Inputs

- `token`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Address of the token contract.
- `arbEscrow`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Address receiving the funds on the destination.
- `amount`

    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: Amount of tokens to transfer.
- `slippage`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: The maximum amount of slippage the user will accept in BPS (e.g., 30 = 0.3%).
- `relayerFee`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: The fee payed to the relayer.

Branches and code coverage

Intended branches

- Increases the allowance for the `token` of the `renzoLockbox` contract and deposits the token amount to that contract.
    - ☐ Test coverage
- Increases the allowance for the `xtoken` of the `connext` contract and calls `connext.xcall` to start the bridging.
    - ☐ Test coverage

Negative behavior

- N/A.

Function call analysis

- `SafeERC20.safeIncreaseAllowance(IERC20(token), address(this.renzoLockbox), amount)`
    - **What is controllable?** `token` and `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.
- `this.renzoLockbox.deposit(amount)`
    - **What is controllable?** `amount`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.
- `SafeERC20.safeIncreaseAllowance(IERC20(address(xToken)), ad-`

`dress(this.connext), amount)`

- **What is controllable?** `amount`.
- **If the return value is controllable, how is it used and how can it go wrong?** N/A.
- **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.

- `this.connext.xcall{value: relayerFee}(1634886255,arbEscrow,address(xToken),msg.s`
    - **What is controllable?** `arbEscrow`, `msg.sender`, `amount`, and `slippage`.
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.

5.2. Module: EscrowBase.sol

Function: `depositETH()`

The function can be utilized for depositing ETH into the escrow contract.

Branches and code coverage

Intended branches

- Wrap the deposited ETH to WETH, then increase the `totalStaked` value of WETH, and increase the value of `usersBalance` of user for that token.
    - ☑ Test coverage

Negative behavior

- Revert if `msg.value` is 0.
    - ☑ Negative test

Function call analysis

- `this._getStorage()`
    - **What is controllable?** N/A.
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the storage slot.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `depositToken(address token, uint256 amount)`

The function can be utilized for depositing tokens into the escrow contract.

Inputs

- `token`
    - **Control**: Fully controlled by caller.
    - **Constraints**: Should be whitelisted.
    - **Impact**: This token is deposited in the contract.
- `amount`
    - **Control**: Fully controlled by caller.
    - **Constraints**: Should not be `0`.
    - **Impact**: The amount of token to be deposited.

Branches and code coverage

Intended branches

- If the wrapped token is a non-zero address, wrap the token first and then deposit the token to the contract.
    - ☑ Test coverage
- If the wrapped token is address(0), directly deposit the token to the contract.
    - ☑ Test coverage
- Increase the `totalStaked` value of the token, and increase the value of `usersBalance` of user for that token.
    - ☑ Test coverage

Negative behavior

- Revert if the token used is `ETH_ADDRESS`.
    - ☑ Negative test
- Revert if amount is `0`.
    - ☑ Negative test
- Revert if there is a balance mismatch after the transfer of token from caller to the contract.
    - ☑ Negative test
- Revert if token is not added or not whitelisted.
    - ☑ Negative test
- Revert if the function is called after break time has reached.
    - ☑ Negative test

Function call analysis

- this._getStorage()
    - **What is controllable?** N/A
    - **If the return value is controllable, how is it used and how can it go wrong?** Returns the storage slot.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A

- IERC20(wrapped).balanceOf(address(this))
    - **What is controllable?** N/A
    - **If the return value is controllable, how is it used and how can it go wrong?** The return value is the balance of the wrapped token of this contract
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A
- SafeERC20.safeTransferFrom(IERC20(token), msg.sender, address(this), amount)
    - **What is controllable?** `token`, `msg.sender` and `amount`
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert; no re-entrancy scenario.
- SafeERC20.safeIncreaseAllowance(IERC20(token), wrapped, amount)
    - **What is controllable?** `token` and `amount`
    - **If the return value is controllable, how is it used and how can it go wrong?** N/A
    - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert; no re-entrancy scenario.
- IERC20(token).balanceOf(address(this))
    - **What is controllable?** N/A
    - **If the return value is controllable, how is it used and how can it go wrong?** The return value is the balance of the `token` of this contract.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A

Function: `withdraw(address token, uint256 amount, bool unwrap)`

The function can be utilized for withdrawing tokens from the escrow contract.

Inputs

- `token`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: No constraints.
    - **Impact**: This token is withdrawn from the contract.
- `amount`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Should not be `0`, and user's deposit balance should be greater than or equal to this value.
    - **Impact**: The amount of token to be withdrawn.
- `unwrap`
    - **Control**: Fully controlled by the caller.
    - **Constraints**: Should be a boolean.
    - **Impact**: Unwraps and then transfers the token to the user if true; transfers di-

rectly without unwrapping if false.

Branches and code coverage

Intended branches

- If `unwrap` is true, then unwrap the token and transfer the unwrapped token to the caller.
  - ☑ Test coverage
- If `unwrap` is false, directly transfer the token to the caller.
  - ☑ Test coverage
- Decrease the `usersBalance` of the user and `totalStaked` for that token by `amount`.
  - ☑ Test coverage

Negative behavior

- Revert if amount is `0`.
  - ☐ Negative test
- Revert if balance of the user is less than the withdrawal `amount` requested.
  - ☐ Negative test
- Revert if `unwrap` is true and the address of `rebase` for that token is `address(0)`.
  - ☑ Negative test
- Revert if rebase token is ETH and the transfer of ETH to `msg.sender` fails.
  - ☐ Negative test

Function call analysis

- `this._getStorage()`
  - **What is controllable?** N/A.
  - **If the return value is controllable, how is it used and how can it go wrong?** Returns the storage slot.
  - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `SafeERC20.safeTransfer(IERC20(tokenInfo.rebase), msg.sender, finalAmt)`
  - **What is controllable?** `msg.sender` and `finalAmt` (partially controllable).
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.
- `SafeERC20.safeTransfer(IERC20(token), msg.sender, amount)`
  - **What is controllable?** `token`, `msg.sender`, and `amount`.
  - **If the return value is controllable, how is it used and how can it go wrong?** N/A.
  - **What happens if it reverts, reenters or does other unusual control flow?** If it reverts, the entire transaction would revert — no reentrancy scenario.

5.3. Module: MainEscrow.sol

Function: `claimTokens(address token, uint256 amount, byte[32][] proof)`

Claim tokens.

Inputs

- `token`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: The token.
- `amount`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: Amount.
- `proof`
    - **Control**: Full.
    - **Constraints**: Length > 0.
    - **Impact**: The proof.

Branches and code coverage

Intended branches

- Verify that the proof is valid.
    - ☑ Test coverage
- User cannot claim amount twice.
    - ☑ Test coverage

Negative behavior

- Another token cannot be used to redeem the proof of a different token.
    - ☑ Negative test

Function call analysis

- `MerkleProof.verify(proof, root, leaf)`
    - **What is controllable?** `proof` and `leaf`.
    - **If the return value is controllable, how is it used and how can it go wrong?** No.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `SafeERC20.safeTransfer(IERC20(token), msg.sender, claimable)`

- **What is controllable?** `token`.
- **If the return value is controllable, how is it used and how can it go wrong?** Nothing.
- **What happens if it reverts, reenters or does other unusual control flow?** N/A.

Function: `withdrawEscrow(address[] tokens)`

Withdraw escrow tokens (`onlyOwner`).

Inputs

- `tokens`
    - **Control**: Full.
    - **Constraints**: None.
    - **Impact**: The tokens to withdraw.

Branches and code coverage

Intended branches

- Tokens are withdrawn.
    - ☑ Test coverage

Negative behavior

- Empty tokens are not transferred.
    - ☐ Negative test

Function call analysis

- `token.balanceOf(address(this))`
    - **What is controllable?** Full.
    - **If the return value is controllable, how is it used and how can it go wrong?** Check if there are tokens.
    - **What happens if it reverts, reenters or does other unusual control flow?** Nothing.
- `SafeERC20.safeTransfer(token, msg.sender, token.balanceOf(address(this)))`
    - **What is controllable?** `token`.
    - **If the return value is controllable, how is it used and how can it go wrong?** Nothing.
    - **What happens if it reverts, reenters or does other unusual control flow?** N/A.
- `token.balanceOf(address(this))`
    - **What is controllable?** `token`.

- **If the return value is controllable, how is it used and how can it go wrong?** Amount to send.
- **What happens if it reverts, reenters or does other unusual control flow?** N/A.

6. Assessment Results

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

During our assessment on the scoped Bracket Fi Escrow contracts, we discovered five findings. One critical issue was found. Three were of medium impact and one was of low impact.

6.1. 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.