our shielding · Your smart contracts, our shielding · Your smart

shieldify

# Berally Pass

SECURITY REVIEW

Date: 6 February 2025

# CONTENTS

## 1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

## 2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

## 3. About Berally - Pass

Pass is a social token on Berally that grants users access to private group chats and crowdfunding vaults. It functions similarly to Friendtech's key but integrates a unique Proof-of-Liquidity (POL) mechanism powered by Berachain's design. This means that every BERA spent on purchasing a Pass is automatically staked into the Berachain reward vault, earning POL rewards in BGT tokens.

## 4. Risk Classification

| Severity | Impact: High | Impact: Medium | Impact: Low |
|---|---|---|---|
| **Likelihood: High** | Critical | High | Medium |
| **Likelihood: Medium** | High | Medium | Low |
| **Likelihood: Low** | Medium | Low | Low |

### 4.1 Impact

- **High** - results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** - results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** - losses will be limited but bearable - and covers vectors similar to griefing attacks that can be easily repaired

### 4.2 Likelihood

- **High** - almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** - still relatively likely, although only conditionally possible
- **Low** - requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

## 5. Security Review Summary

The security review lasted 2 days with a total of 64 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified one high-severity issue, two medium-severity issues, and one low-severity vulnerability. The vulnerabilities primarily stem from fee theft risk, slippage and deployment misconfigurations.

The Berally team has done a great job with the development and has been highly responsive to the Shieldify research team's inquiries and promptly implemented all recommendations.

### 5.1 Protocol Summary

| **Project Name** | **Berally - Pass** |
|---|---|
| **Repository** | smartcontract-passes |
| **Type of Project** | Social Token |
| **Audit Timeline** | 2 days |
| **Review Commit Hash** | 73ec1b6ee0b2d9d8fc645529cf3ce10d00792422 |
| **Fixes Review Commit Hash** | 54f1f20622cc68536f7c7713d0a82aa3a0197b3c |

### 5.2 Scope

The following smart contracts were in the scope of the security review:

| File | nSLOC |
|---|---|
| Passes.sol | 215 |
| ICustomBerachainRewardsVault.sol | 3 |
| **Total** | **218** |

## 6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Critical & High** issues: **1**
- **Medium** issues: **2**
- **Low** issues: **1**

| ID | Title | Severity | Status |
|---|---|---|---|
| [H-01] | Users Can Steal Part of Protocol Fees Through Referrals to Their Own Addresses | High | Fixed |
| [M-01] | Slippage Check Before Fee Deduction Allows for Users Getting Less Than Desired ‘minPrice‘ | Medium | Fixed |
| [M-02] | The ‘Passes‘ Contract Initialization Can Be Permanently Dossed, Forcing Redeployment | Medium | Fixed |
| [L-01] | ETH Sent to Contract Through ‘sellPasses()‘ Will Be Lost Forever | Low | Fixed |

# 7. Findings

## [H-01] Users Can Steal Part of Protocol Fees Through Referrals to Their Own Addresses

### Severity

High Risk

### Description

Users can steal part of the protocol fee by setting `referral` to addresses they control. `buyPasses` and `sellPasses` allows the caller to specify the `referral` address where the `referralFee` is paid to.

### Location of Affected Code

File: Passes.sol

```
@>  function sellPasses(address manager, uint256 amount, uint256 minPrice
    , address referral) public payable {
        // code
        bool success4 = true;
        if(referralFee > 0) {
            (success4, ) = referral.call{value: referralFee}("");
        }

        require(success1 && success2 && success3 && success4, "Unable to
            send funds");
    }

...

@>  function buyPasses(address manager, uint256 amount, uint256 factor,
    address referral) public payable {
        // code
        bool success3 = true;
        if(referralFee > 0) {
            (success3, ) = referral.call{value: referralFee}("");
        }

        // code
    }
```

### Impact

Part of the protocol fees can be stolen.

### Proof of Concept

The `referralFee` is calculated as a portion of the protocol fee in both `buyPasses` and `sellPasses`.

```
uint256 referralFee = 0;
if(referral != address(0)) {
    referralFee = price * referralFeePercent / 1 ether;
    protocolFee -= referralFee;
}
```

So by setting `referral` to a user-controlled address, the user can easily recover part of the `protocolFee`.

### Recommendation

Create an admin-controlled function to specify a referral address. The address can then distribute the fees to the required referrals.

### Team Response

Fixed.

## [M-01] Slippage Check Before Fee Deduction Allows for Users Getting Less Than Desired `minPrice`

### Severity

Medium Risk

### Description

The `sellPasses()` function doesn't take fees into consideration when comparing the price against slippage, causing users to get less than their specified `minPrice`.

### Location of Affected Code

File: Passes.sol

```
function sellPasses(address manager, uint256 amount, uint256 minPrice,
    address referral) public payable {
     uint256 supply = passesSupply[manager];
     require(supply > amount, "Cannot sell the last pass");
     require(msg.sender != manager || passesBalance[manager][msg.sender] >
         amount, "Cannot sell the first pass");
     require(passesBalance[manager][msg.sender] >= amount, "Insufficient
        passes");

     uint factor = factors[manager];
@>       uint256 price = getPrice(supply - amount, amount, factor);
@>       require(price >= minPrice, "Lower than minimum price");
//..
```

```
@>       (bool success1, ) = msg.sender.call{value: price - protocolFee -
    managerFee - referralFee}("");
     (bool success2, ) = treasury.call{value: protocolFee}("");
     (bool success3, ) = manager.call{value: managerFee}("");

     bool success4 = true;
     if(referralFee > 0) {
         (success4, ) = referral.call{value: referralFee}("");
     }

     require(success1 && success2 && success3 && success4, "Unable to send
         funds");
}
```

### Impact

In many cases, users will get less than their specified `minPrice` since fees are deducted after comparing against slippage.

### Proof of Concept

1. The `getPrice` function returns a price of 10 eth;
2. The user specified a `minPrice` of 10 eth;
3. Without accounting for fees, the function passes the slippage check;
4. Protocol fee and manager fee are deducted, reducing the price to 8 eth;
5. The user gets 8 eth rather than the expected 10;

### Recommendation

Query the `getSellPriceAfterFee()` function and compare its result against the `minPrice` instead.

### Team Response

Fixed.

## [M-02] The `Passes` Contract Initialization Can Be Permanently Dossed, Forcing Redeployment

### Severity

Medium Risk

### Description

Initialization can be dossed by directly creating a rewards vault in the factory. Since CREATE is used to deploy the `PassesStakingToken` in the initializer of Passes.sol, the potential address of the stakingToken can be calculated based on the `Passes.sol` contract address and nonce (number of deployed contracts the factory has previously deployed). In this case, the nonce will be 0. See here

> The destination address is calculated as the rightmost 20 bytes (160 bits) of the Keccak-256 hash of the rlp encoding of the sender address followed by its nonce. That is: address = keccak256(rlp([sender_address,sender_nonce]))[12:]

As a result, upon contract deployment, before its initialization, an attacker can calculate the address of the `PassesStakingToken` and then directly call the `createRewardsVault` in the RewardsVaultFactory.

By calling this, the vault for the staking token is deployed and stored, and subsequent calls to create a rewards vault with the same staking token will fail due to a check in the `createRewardsVault` function.

### Location of Affected Code

File: Passes.sol

```
function initialize() public initializer {
    __Ownable_init();
    treasury = owner();

    protocolFeePercentage = 5e16;
    managerFeePercentage = 5e16;
    referralFeePercent = 1e16;

    defaultFactors[500] = true;
    defaultFactors[100] = true;
    defaultFactors[30] = true;

    address vaultFactory = 0x2B6e40f65D82A0cB98795bC7587a71bfa49fBB2B;
@>      stakingToken = new PassesStakingToken(address(this));
    polVault = IBerachainRewardsVault(
@>          IBerachainRewardsVaultFactory(vaultFactory).
   createRewardsVault(
            address(stakingToken)
        )
    );
}
```

File: BerachainRewardsVaultFactory.sol

```
function createRewardsVault(address stakingToken) external returns (
   address) {
@>      if (getVault[stakingToken] != address(0)) VaultAlreadyExists.
   selector.revertWith();

    // Use solady library to deploy deterministic beacon proxy.
    bytes32 salt = keccak256(abi.encode(stakingToken));
    address vault = LibClone.deployDeterministicERC1967BeaconProxy(beacon
       , salt);

    // Store the vault in the mapping and array.
@>      getVault[stakingToken] = vault;
    allVaults.push(vault);
    emit VaultCreated(stakingToken, vault);

    // Initialize the vault.
    BerachainRewardsVault(vault).initialize(bgt, distributor, berachef,
       owner(), stakingToken);

    return vault;
}
```

### Impact

Upon initialization front run and stakingToken reward vault deployment, subsequent attempts to deploy the same stakingToken reward vault will always fail causing contract initialization to be dossed.

#### Proof of Concept

1. Protocol deploys Passes.sol;
2. The attacker uses the Passes.sol contract address to calculate the potential address of the `PassesStakingToken` contract;
3. He then front runs the call to initialize Passes.sol, to call the `createRewardsVault` function in the factory;
4. This creates and sets the vault for the staking token;
5. The protocol's call to initialize Passes.sol will execute, calling the `createRewardsVault` function again, which will fail.
6. Initialization is dossed, forcing redeployment.

#### Recommendation

Recommend wrapping the `createRewardsVault` in a try-catch mechanism. If the call fails, catch the error and use the `predictRewardsVaultAddress` function to set the vault address as `polVault`
.

#### Team Response

Fixed.

### [L-01] ETH Sent to Contract Through `sellPasses()` Will Be Lost Forever

#### Severity

Low Risk

#### Description

Sell passes should not be payable, as it has no logic to process incoming ETH.

#### Location of Affected Code

File: Passes.sol

```
function sellPasses(address manager, uint256 amount, uint256 minPrice,
    address referral) public payable {
```

#### Impact

Any ETH sent through the function will be lost forever.

#### Recommendation

Remove the payable functionality.

#### Team Response

Fixed.

our shielding · Your smart contracts, our shielding · Your smart

shieldify

# Thank you!