BadgerDAO Zaps contest Findings & Analysis Report

2021-01-05

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 code contest is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the code contest outlined in this document, C4 conducted an analysis of the BadgerDAO Zaps smart contract system written in Solidity. The code contest took place between November 14—November 16 2021.

Wardens

14 Wardens contributed reports to the BadgerDAO Zaps contest:

1. WatchPug (jtp and ming)
2. gzeon
3. Ruhum
4. 0x0x0x
5. MetaOxNull
6. defsec
7. pmerkleplant
8. fatima_naz
9. ksk2345
10. pants

11. yeOlde
12. TomFrenchBlockchain
13. GiveMeTestEther

This contest was judged by leastwood.

Final report assembled by moneylegobatman and CloudEllie.

Summary

The C4 analysis yielded an aggregated total of 14 unique vulnerabilities and 40 total findings. All of the issues presented here are linked back to their original finding.

Of these vulnerabilities, 1 received a risk rating in the category of HIGH severity, 6 received a risk rating in the category of MEDIUM severity, and 7 received a risk rating in the category of LOW severity.

C4 analysis also identified 7 non-critical recommendations and 19 gas optimizations.

Scope

The code under review can be found within the C4 BadgerDAO Zaps contest repository, and is composed of 4 smart contracts written in the Solidity programming language.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities according to a methodology based on OWASP standards.

Vulnerabilities are divided into three primary risk categories: high, medium, and low.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic

- Gas use

Further information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on [the C4 website](#).

High Risk Findings (1)

[H-01] `setGuardian()` Wrong implementation

*Submitted by WatchPug, also found by MetaOxNull, gzeon, fatima*naz, 0x0x0x, and ksk2345_

`IbbtcVaultZap.sol` [L116-L119](#)

```
    function setGuardian(address _guardian) external {
        _onlyGovernance();
        governance = _guardian;
    }
```

`SettToRenIbbtcZap.sol` [L130-L133](#)

```
    function setGuardian(address _guardian) external {
        _onlyGovernance();
        governance = _guardian;
    }
```

`governance = _guardian` should be `guardian = _guardian`.

Medium Risk Findings (6)

[M-01] Improper implementation of slippage check

Submitted by WatchPug, also found by gzeon

`Zap.sol` [L216-L238](#)

```
function redeem(IERC20 token, uint amount, uint poolId, int128 i
    external
    defend
    blockLocked
    whenNotPaused
    returns(uint out)
{
    ibbtc.safeTransferFrom(msg.sender, address(this), amount);

    Pool memory pool = pools[poolId];
    if (poolId < 3) { // setts
        settPeak.redeem(poolId, amount);
        pool.sett.withdrawAll();
        pool.deposit.remove_liquidity_one_coin(pool.lpToken.bala
    } else if (poolId == 3) { // byvwbtc
        byvWbtcPeak.redeem(amount);
        IbyvWbtc(address(pool.sett)).withdraw(); // withdraws al
    } else {
        revert("INVALID_POOL_ID");
    }
    out = token.balanceOf(address(this));
    token.safeTransfer(msg.sender, out);
}
```

In the current implementation of. `Zap.sol#redeem()` , the `outAmount` of `IbyvWbtc.withdraw()` is not controlled by `minOut` .

Recommendation

Consider implementing the `minOut` check in between L236 and L237.

```
...
out = token.balanceOf(address(this));
require(out >= _minOut, "Slippage Check");
token.safeTransfer(msg.sender, out);
}
```

GalloDaSballo (BadgerDAO) confirmed:

> Agree with the finding, not having slippage check at end means people can get rekt, we'll add as suggested

[M-02] Missing `_token.approve()` to `curvePool` in `setZapConfig`

Submitted by WatchPug

`SettToRenIbbtcZap.sol` **L162-L183**

```
    function setZapConfig(
        uint256 _idx,
        address _sett,
        address _token,
        address _curvePool,
        address _withdrawToken,
        int128 _withdrawTokenIndex
    ) external {
        _onlyGovernance();

        require(_sett != address(0));
        require(_token != address(0));
        require(
            _withdrawToken == address(WBTC) || _withdrawToken == adc
        );

        zapConfigs[_idx].sett = ISett(_sett);
        zapConfigs[_idx].token = IERC20Upgradeable(_token);
        zapConfigs[_idx].curvePool = ICurveFi(_curvePool);
        zapConfigs[_idx].withdrawToken = IERC20Upgradeable(_withdraw
        zapConfigs[_idx].withdrawTokenIndex = _withdrawTokenIndex;
    }
```

In the current implementation, when `curvePool` or `token` got updated, `token` is not approved to `curvePool`, which will malfunction the contract and break minting.

Recommendation

Change to:

```
    function setZapConfig(
        uint256 _idx,
        address _sett,
```

```
        address _token,
        address _curvePool,
        address _withdrawToken,
        int128 _withdrawTokenIndex
    ) external {
        _onlyGovernance();

        require(_sett != address(0));
        require(_token != address(0));
        require(
            _withdrawToken == address(WBTC) || _withdrawToken == adc
        );

        if (zapConfigs[_idx].curvePool != _curvePool && _curvePool !
            IERC20Upgradeable(_token).safeApprove(
                _curvePool,
                type(uint256).max
            );
        }

        zapConfigs[_idx].sett = ISett(_sett);
        zapConfigs[_idx].token = IERC20Upgradeable(_token);
        zapConfigs[_idx].curvePool = ICurveFi(_curvePool);
        zapConfigs[_idx].withdrawToken = IERC20Upgradeable(_withdraw
        zapConfigs[_idx].withdrawTokenIndex = _withdrawTokenIndex;
    }
```

GalloDaSballo (BadgerDAO) confirmed:

> Agree with the finding, it should be noted that adding a pool does handle for the scenario, this would break the pool in case we update it or change the token

[M-03] Zap contract's `redeem()` function doesn't check which token the user wants to receive

Submitted by Ruhum

Impact

In the `redeem()` function, the user can pass a token address. That's the token they receive in return for the ibbtc they give back. Because of missing address checks the user can provide any possible ERC20 token here without the function reverting.

Although it's not strictly specified in the code I expect that the user should only be able to redeem wBTC or renBTC tokens since they should also only be able to deposit those.

Proof of Concept

`Zap.sol` L216-L238

Tools Used

Manual Analysis

Recommended Mitigation Steps

Verify that the passed token address is either wBTC or renbtc

tabshaikh (BadgerDAO) disagreed with severity:

> best practice to add wBTC or renbtc in require, disagree on the severity

GalloDaSballo (BadgerDAO) commented:

> Agree with the finding since only user can rekt themselves I believe this to be a medium severity finding we'll mitigate by adding a slippage check at the end of the function

[M-04] Excessive `require` makes the transaction fail unexpectedly

Submitted by WatchPug

The check for `RENCRV_VAULT.blockLock` is only needed when `if (_amounts[1] > 0 || _amounts[2] > 0)`.

However, in the current implementation, the check is done at the very first, making transactions unrelated to `RENCRV_VAULT` fail unexpectedly if there is a prior transaction involved with `RENCRV_VAULT` in the same block.

`IbbtcVaultZap.sol` L149-L199

```
function deposit(uint256[4] calldata _amounts, uint256 _minOut)
    public
    whenNotPaused
{
    // Not block locked by setts
    require(
        RENCRV_VAULT.blockLock(address(this)) < block.number,
        "blockLocked"
    );
    require(
        IBBTC_VAULT.blockLock(address(this)) < block.number,
        "blockLocked"
    );

    uint256[4] memory depositAmounts;

    for (uint256 i = 0; i < 4; i++) {
        if (_amounts[i] > 0) {
            ASSETS[i].safeTransferFrom(
                msg.sender,
                address(this),
                _amounts[i]
            );
            if (i == 0 || i == 3) {
                // ibbtc and sbtc
                depositAmounts[i] += _amounts[i];
            }
        }
    }

    if (_amounts[1] > 0 || _amounts[2] > 0) {
        // Use renbtc and wbtc to mint ibbtc
        // NOTE: Can change to external zap if implemented
        depositAmounts[0] += _renZapToIbbtc([_amounts[1], _amour
    }
    // ...
}
```

shuklaayush (BadgerDAO) confirmed

GalloDaSballo (BadgerDAO) commented:

> Agree with the finding, we would have to check for those locks only under specific condition, not doing so opens up to unnecessary reverts

GalloDaSballo (BadgerDAO) patched:

> We have mitigated by following the advice of the warden

[M-05] No slippage control on `deposit` of `IbbtcVaultZap.sol`

Submitted by gzeon, also found by WatchPug

Impact

There is no slippage control on `deposit` of `IbbtcVaultZap.sol`, which expose user to sandwich attack.

Proof of Concept

`IbbtcVaultZap.sol` L174 Any deposit can be sandwiched, especially when the pool is not balanced.

Recommended Mitigation Steps

Add a `minOut` in line with the mint function of other contacts, and pass it as a parameter on L174

[M-06] `calcMint` always return poolId=0 and idx=0

Submitted by gzeon

Impact

`calcMint` in Zap.sol always return poolId=0 and idx=0, while the docstring specified it should return the most optimal route instead. This will lead to suboptimal zap.

Proof of Concept

- `Zap.sol` L156

GalloDaSballo (BadgerDAO) commented:

> Given the context that the warden has, the finding is valid, we're missing two functions for `calcMint`

> As for us, we have shifted to only using pool 0 as such the code works fine for us

Low Risk Findings (7)

- [L-01] Wrong comment on `SettToRenIbbtcZap.sol` and `IbbtcVaultZap.sol` *Submitted by 0x0x0x*
- [L-02] `Zap.sol#redeem()` Lack of input validation *Submitted by WatchPug*
- [L-03] Arithmetic operations without using SafeMath may over/underflow *Submitted by WatchPug*
- [L-04] Use `safeTransfer` / `safeTransferFrom` consistently instead of `transfer` / `transferFrom` *Submitted by defsec*
- [L-05] Zap contract's `mint()` allows minting ibbtc tokens for free *Submitted by Ruhum*
- [L-06] Missing overflow protection *Submitted by pmerkleplant*
- [L-07] `blockLock` of `RENCRV_SETT` makes transactions likely to fail as only 1 transaction is allowed in 1 block *Submitted by WatchPug*

Non-Critical Findings (7)

- [N-01] Critical changes should use two-step procedure *Submitted by WatchPug, also found by defsec*
- [N-02] Modifier should be used instead of functions to write modifier in ibBTC VaultZap.sol *Submitted by fatimanaz_*
- [N-03] use modifier keyword to write modifier not function In `SettToRenIbbtcZap.sol` line no - 105 and 109 *Submitted by fatimanaz_*
- [N-04] Missing events for critical operations *Submitted by WatchPug, also found by pants, 0x0x0x, and defsec*

- [N-05] Open TODOs *Submitted by pants, also found by MetaOxNull, GiveMeTestEther, and yeOlde*
- [N-06] Redundant type casting *Submitted by WatchPug*
- [N-07] named return issue - Zap.sol `calcMint` *Submitted by pants*

Gas Optimizations (19)

- [G-01] For `uint` use `!= 0` instead of `> 0` *Submitted by 0x0x0x*
- [G-02] SLOAD pools.length for Every Loop is Waste of Gas *Submitted by MetaOxNull*
- [G-03] Avoiding Initialization of Loop Index If It Is 0 *Submitted by MetaOxNull*
- [G-04] Zap.sol declares unused variable `_ren` in `calcRedeemInRen` among other functions *Submitted by TomFrench*
- [G-05] Adding `recipient` parameter to mint functions can help avoid unnecessary token transfers and save gas *Submitted by WatchPug*
- [G-06] `Zap.sol#mint()` Validation of `poolId` can be done earlier to save gas *Submitted by WatchPug*
- [G-07] Avoid unnecessary read of array length in for loops can save gas *Submitted by WatchPug*
- [G-08] Unused local variables *Submitted by WatchPug, also found by GiveMeTestEther, yeOlde, and pmerkleplant*
- [G-09] Avoid unnecessary code execution can save gas *Submitted by WatchPug*
- [G-10] Avoid unnecessary arithmetic operations can save gas *Submitted by WatchPug*
- [G-11] `Zap.sol#mint()` Check `blockLock` earlier can save gas *Submitted by WatchPug*
- [G-12] Gas Optimization on the Public Function *Submitted by defsec*
- [G-13] Gas optimization: Use else if for mutually exclusive conditions *Submitted by gzeon, also found by WatchPug*
- [G-14] Gas optimization: Unreachable code in Zap.sol *Submitted by gzeon*
- [G-15] Gas optimization: Unnecessary ops *Submitted by gzeon*

- [G-16] Unnecessary `SLOAD`s / `MLOAD`s / `CALLDATALOAD`s in for-each loops *Submitted by pants*
- [G-17] Zap.sol init for loop - uint default value is 0 *Submitted by pants*
- [G-18] public function that could be set external instead *Submitted by pants*
- [G-19] ibbtcCurveLP can be simplified *Submitted by ye0lde*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Contests incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Contest submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top