Security Assessment

BakerDAO

Jun 18th, 2021

Table of Contents

Summary

This report has been prepared for BakerDAO smart contracts, to discover issues and vulnerabilities in the source code of their Smart Contract as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Static Analysis and Manual Review techniques.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases given they are currently missing in the repository;
- Provide more comments per each function for readability, especially contracts are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

Overview

Project Summary

Project Name	BakerDAO
Platform	BSC
Language	Solidity
Codebase	https://github.com/BakerDAO-Fi/bss https://github.com/BakerDAO-Fi/yield-farming-contracts
Commit	f3417fb2ee1c21828ff10d21e7e3835e79b0967b d72b50f82bb740fcfd8837e72455ddcac8161343 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

Audit Summary

Delivery Date	Jun 18, 2021
Audit Methodology	Static Analysis, Manual Review
Key Components	

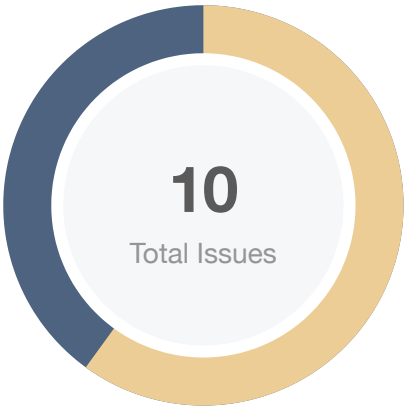
Vulnerability Summary

Total Issues	10
Critical	0
Major	0
Medium	0
Minor	6
Informational	4
Discussion	0

Audit Scope

ID	file	SHA256 Checksum
DOP	bss-bsc-deployment/src/cat.sol	8511e7a26788ef0ef4dea685c75ba004822a99e637849de2fc6d8e71643192e7
DCK	bss-bsc-deployment/src/dai.sol	d2c2bdb043071e1150b14313e4d5664ca8877110b7bddcfeb6ffff84a155ada6
DCP	bss-bsc-deployment/src/end.sol	d9c249613c738c0059fd359252199ec18270afde27488f406985db1e40c2d2a4
DKP	bss-bsc-deployment/src/flap.sol	9dfb9c6ffff1dba5102bce1e31bbeb66619911c8a57883ff74afe086c63fd62c
AOC	bss-bsc-deployment/src/flip.sol	aa9ffac92ff488f1905ed94383016a323a661ee323b1c34af0edf6bd2ee30fe0
AOK	bss-bsc-deployment/src/flop.sol	362bf141a3961f1388dfc4aff89e6bb9617fa750d1ac480379ff2d97ec34034f
AOP	bss-bsc-deployment/src/join.sol	d2fcf54aa72a9afd42ed6749d1a56e7aa8d8bcf54105e9c2845ce97916581f2a
ACK	bss-bsc-deployment/src/jug.sol	7944ee58946d92e729830ae0d4918c629d9b91f1627faabc9c714b15b469add6
ACP	bss-bsc-deployment/src/lib.sol	b25a4d1cfda61fd90033512663dbd2c8115e335510b0c1e8d071d4007ee58b41
AKP	bss-bsc-deployment/src/pot.sol	441f36a166035b5e03f96ca95fffb1a52692c7c52b61419ff984b5dad8edcfc4
OCK	bss-bsc-deployment/src/spot.sol	15db43511ea43c8f1a5602ad130dd042cba1003142e333e9e9b72894602c0df8
VAS	bss-bsc-deployment/src/vat.sol	1c4bfc29e5a258ad0b564020637d33dbfb2718cadb4dc27f573fc9ac27c7b4cd
VOS	bss-bsc-deployment/src/vow.sol	ab59ff47044f0d5b88e0bccda6139c81aaff1e2a306952854b66bd59a07a327b
MCD	yield-farming-contracts-main/src/MasterChef.sol	03c34361c6d649ea83b471abadde755bbcadceb94eddc246f8b2375c48cce451
TAF	yield-farming-contracts-main/src/TokenAmountFromDex.sol	120e39e543955c8f968dea19076f7e3e188965633f9e922b4be375818c4ea074

Findings



Critical	**0** (0.00%)
Major	**0** (0.00%)
Medium	**0** (0.00%)
Minor	**6** (60.00%)
Informational	**4** (40.00%)
Discussion	**0** (0.00%)

ID	Title	Category	Severity	Status
MCD-01	Check-effect-interaction Pattern Violation	Logical Issue	Minor	Resolved
MCD-02	Check-effect-interaction Pattern Violation	Logical Issue	Minor	Resolved
MCD-03	Proper Usage of `Public` and `External`	Gas Optimization	Informational	Resolved
MCD-04	`SafeMath` Not Used	Mathematical Operations	Minor	Resolved
MCD-05	`SafeMath` Not Used	Mathematical Operations	Minor	Resolved
MCD-06	Unhandled Return Value of `transferFrom` and `transfer`	Logical Issue	Minor	Resolved
MCD-07	Recommended Explicit Pool Validity Checks	Logical Issue	Informational	Resolved
MCD-08	Incompatibility With Deflationary Tokens	Logical Issue	Minor	Resolved
TAF-01	Proper Usage of `Public` and `External`	Gas Optimization	Informational	Resolved
TAF-02	Check Reserve Greater Than 0	Logical Issue	Informational	Resolved

MCD-01 | Check-effect-interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	yield-farming-contracts-main/src/MasterChef.sol: 766~767	Resolved

Description

The update of `pool.amount` is after the call to `transferMainnetToken`, which violates the check-effect-interaction pattern

Recommendation

A recommended revision is to put `pool.amount` before a potential external function call for example:

```
function withdraw(uint256 _pid, uint256 _amount) public payable {
        PoolInfo storage pool = poolInfo[_pid];
        UserInfo storage user = userInfo[_pid][msg.sender];
        require(block.number >= pool.startBlock,"this pool is not start!");
        require(user.amount >= _amount, "withdraw: not good");
        harvest(_pid, msg.sender);
        user.amount = user.amount.sub(_amount);
        user.rewardDebt =
user.amount.mul(pool.accSushiPerShare).div(ACC_SUSHI_PRECISION);
        pool.amount = pool.amount.sub(_amount);
        if(pool.lpToken != IERC20(0)){
            pool.lpToken.transfer(msg.sender, _amount);
        }else{//if pool is HT
            transferMainnetToken(msg.sender, _amount);
        }
        emit Withdraw(msg.sender, _pid, _amount);
    }
```

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-02 | Check-effect-interaction Pattern Violation

Category	Severity	Location	Status
Logical Issue	Minor	yield-farming-contracts-main/src/MasterChef.sol: 784~787	Resolved

Description

Updating storage variables after transfer violates the check-effect-interaction.

Recommendation

We advise the client to revise the function `emergencyWithdraw()` with following snippet:

```
function emergencyWithdraw(uint256 _pid) public {
    PoolInfo storage pool = poolInfo[_pid];
    UserInfo storage user = userInfo[_pid][msg.sender];
    pool.amount = pool.amount.sub(user.amount);
    uint256 oldAmount = user.amount;
    user.amount = ZERO;
    user.rewardDebt = ZERO;
    if(pool.lpToken != IERC20(0)){
        pool.lpToken.transfer(msg.sender, oldAmount);
    }else{//if pool is HT
        transferMainnetToken(msg.sender, oldAmount);
    }
    emit EmergencyWithdraw(msg.sender, _pid, oldAmount);
}
```

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-03 | Proper Usage of `Public` and `External`

Category	Severity	Location	Status
Gas Optimization	Informational	yield-farming-contracts-main/src/MasterChef.sol	Resolved

Description

In public functions, Solidity immediately copies array arguments to memory, while external functions can read directly from `calldata`. Memory allocation is expensive, whereas reading from `calldata` is cheap. For best practices, the visibility of function should use external if the function is expected to only ever be called externally, and use public if the specific function needs to be called both externally and internally. There is a list of functions in this contract that can be declared external to save gas:

- `owner()`
- `renounceOwnership()`
- `transferOwnership()`
- `setTokenAmountContract()`
- `setHarvestFeeRatio()`
- `add()`
- `setPoolInfo()`
- `setAllPoolOperationFee()`
- `setAllPoolHarvestFee()`
- `deposit()`
- `withdraw()`
- `emergencyWithdraw()`
- `updateDev1Address()`
- `updateDev2Address()`
- `updateDev3Address()`
- `updateBuyAddress()`
- `addRewardForPool()`

Recommendation

We advise the client to consider using `external` to decorate the functions listed above to save gas.

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-04 | `SafeMath` Not Used

Category	Severity	Location	Status
Mathematical Operations	Minor	yield-farming-contracts-main/src/MasterChef.sol: 493	Resolved

Description

This expression does not check arithmetic overflow. Such unsafe math operation may cause unexpected behavior if unusual parameters are given.

Recommendation

We advise the client to consider using `SafeMath` library of Openzeppelin library to prevent overflow.

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-05 | `SafeMath` Not Used

Category	Severity	Location	Status
Mathematical Operations	Minor	yield-farming-contracts-main/src/MasterChef.sol: 729	Resolved

Description

This expression does not check arithmetic overflow. Such unsafe math operation may cause unexpected behavior if unusual parameters are given.

Recommendation

We advise the client to consider using `SafeMath` library of Openzeppelin library to prevent underflow.

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-06 | Unhandled Return Value of `transferFrom` and `transfer`

Category	Severity	Location	Status
Logical Issue	Minor	yield-farming-contracts-main/src/MasterChef.sol: 869~871, 930, 847, 831~836, 780, 762, 741~743, 739, 712, 527	Resolved

Description

Function `transferFrom()`'s and `transfer()`'s' return values are not checked in the function `addRewardForPool()`. This could be a problem since `transferFrom()` or `transfer()` could fail, and therefore the user's fund is not deposited to the protocol, causing financial loss to the project.

Recommendation

We advise the client change to add `require()` to check the return values of `transferFrom()` and `transfer()`, or use `safeTransferFrom()` function and `safeTransfer()` function from `SafeERC20` library in Openzeppelin Library: `https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/ERC20/utils/SafeERC20.sol`

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-07 | Recommended Explicit Pool Validity Checks

Category	Severity	Location	Status
Logical Issue	Informational	yield-farming-contracts-main/src/MasterChef.sol: 533, 567, 587, 609, 657, 701, 753, 776, 791, 856, 908	Resolved

Description

There's no sanity check to validate if a pool is existing.

Recommendation

We advise the client to adopt following modifier `validatePoolByPid` to functions `set()`, `migrate()`, `deposit()`, `withdraw()`, `emergencyWithdraw()`, `pendingSushi()` and `updatePool()`.

```
1 modifier validatePoolByPid(uint256 _pid) {
2     require (_pid < poolInfo . length , "Pool does not exist") ;
3     _;
4 }
```

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

MCD-08 | Incompatibility With Deflationary Tokens

Category	Severity	Location	Status
Logical Issue	Minor	yield-farming-contracts-main/src/MasterChef.sol: 701, 753	Resolved

Description

When users add or remove LP tokens into the router, and the `mint` and `burn` operations are performed. When transferring standard ERC20 deflationary tokens, the input amount may not be equal to the received amount due to the charged transaction fee. As a result, the amount inconsistency will occur and the transaction may fail due to the validation checks.

Recommendation

We advise the client to regulate the set of LP tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support deflationary tokens.

Alleviation

`[BakerDAO]`: BakerDAO will not support any deflationary tokens

TAF-01 | Proper Usage of `Public` and `External`

Category	Severity	Location	Status
Gas Optimization	Informational	yield-farming-contracts-main/src/TokenAmountFromDex.sol	Resolved

Description

In public functions, Solidity immediately copies array arguments to memory, while external functions can read directly from `calldata`. Memory allocation is expensive, whereas reading from `calldata` is cheap. For best practices, the visibility of function should use external if the function is expected to only ever be called externally, and use public if the specific function needs to be called both externally and internally. There is a list of functions in this contract that can be declared external to save gas:

- `owner()`
- `renounceOwnership()`
- `transferOwnership()`
- `setPairContract()`

Recommendation

We advise the client to consider using `external` to decorate the functions listed above to save gas.

Alleviation

`[BakerDAO]`: The client heeded the advice and fixed the issue in the commit 51a6b086d9aa0b29f6e7a09dd7c4e2c5ad5759e4

TAF-02 | Check Reserve Greater Than 0

Category	Severity	Location	Status
Logical Issue	Informational	yield-farming-contracts-main/src/TokenAmountFromDex.sol: 287~296	Resolved

Description

There is no check for both `reserve(reserve1, reserve0) greater than 0`. It would be better if we check the reserve and output an error if one of them is not greater than zero, this would let the user know that this pool currently does not have enough liquidity. A similar function in Uniswap V2 called `getAmountOut()` implements such checks and is a good industry practice.

Recommendation

We advise the client to change line 287-296 into:

```
(uint112 reserve0, uint112 reserve1, ) = pairContract.getReserves();
require(reserve0 > 0 && reserve1 > 0, 'UniswapV2Library: INSUFFICIENT_LIQUIDITY');

if(_token0 == pairContract.token0()){
    return _token1Amount.mul(reserve0).div(reserve1);
}else{
    return _token1Amount.mul(reserve1).div(reserve0);
}
```

Alleviation

`[BakerDAO]`: In the current design of BakerDAO, function `getTokenAmount()` will return 0 when there's no sufficient liquidity in the uniswap pool, i.e. reserve0 == 0 or reserve1 == 0. This will benefit the user as the `getHarvestFee()` will return 0 for harvest fee for user.

Appendix

Finding Categories

Gas Optimization

Gas Optimization findings do not affect the functionality of the code but generate different, more optimal EVM opcodes resulting in a reduction on the total gas cost of a transaction.

Mathematical Operations

Mathematical Operation findings relate to mishandling of math formulas, such as overflows, incorrect operations etc.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block.timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to the Company in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes without CertiK's prior written consent.

This report is not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any "product" or "asset" created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK's position is that each company and individual are responsible for their own due diligence and continuous security. CertiK's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

About

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.