bamidefi

BamiDeFi

Security Assessment

April 12th, 2021

[Preliminary Report]

Audited By:
Adrian Hetman @ CertiK
adrian.hetman@certik.org
Reviewed By:
Alex Papageorgiou @ CertiK
alex.papageorgiou@certik.org

CERTIK

Disclaimer

CertiK reports are not, nor should be considered, an "endorsement" or "disapproval" of any particular project or team. These reports are 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 review.

CertiK Reports do 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.

CertiK Reports should not be used in any way to make decisions around investment or involvement with any particular project. These reports in no way provide investment advice, nor should be leveraged as investment advice of any sort.

CertiK Reports represent an extensive auditing 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.

What is a CertiK report?

- A document describing in detail an in depth analysis of a particular piece(s) of source code provided to CertiK by a Client.
- An organized collection of testing results, analysis and inferences made about the structure, implementation and overall best practices of a particular piece of source code.
- Representation that a Client of CertiK has completed a round of auditing with the intention to increase the quality of the company/product's IT infrastructure and or source code.

Overview

CERTIK

Project Summary

Project Name	bamidefi - BamiDeFi
Description	Fork of a ParaSwap and SushiSwap with enchanced features.
Platform	Ethereum; Solidity, Yul
Codebase	GitHub Repository
Commits	1. c920f5b4b3569adaa4be2d1460ac4a4d7af3a49d

Audit Summary

Delivery Date	April 12th, 2021
Method of Audit	Static Analysis, Manual Review
Consultants Engaged	1
Timeline	April 8th, 2021 - April 12th, 2021

Vulnerability Summary

Total Issues	13
Total Critical	0
Total Major	0
Total Medium	0
Total Minor	4
Total Informational	9

Executive Summary

Bami Defi is a combination of forks from ParaSwap and SushiSwap. It improves upon main contracts with added `nonReentrant` modifiers, following checks-effects-pattern or disallowing adding twice the same liquidity pool token. Code quality is good with said improvements.

All major issues that are in the Pancakeswap and SushiSwap are also available here. During our audit and review of the code we only focused on the difference between forked code and the original code.

Below is the list of contracts and from which project they are forked:

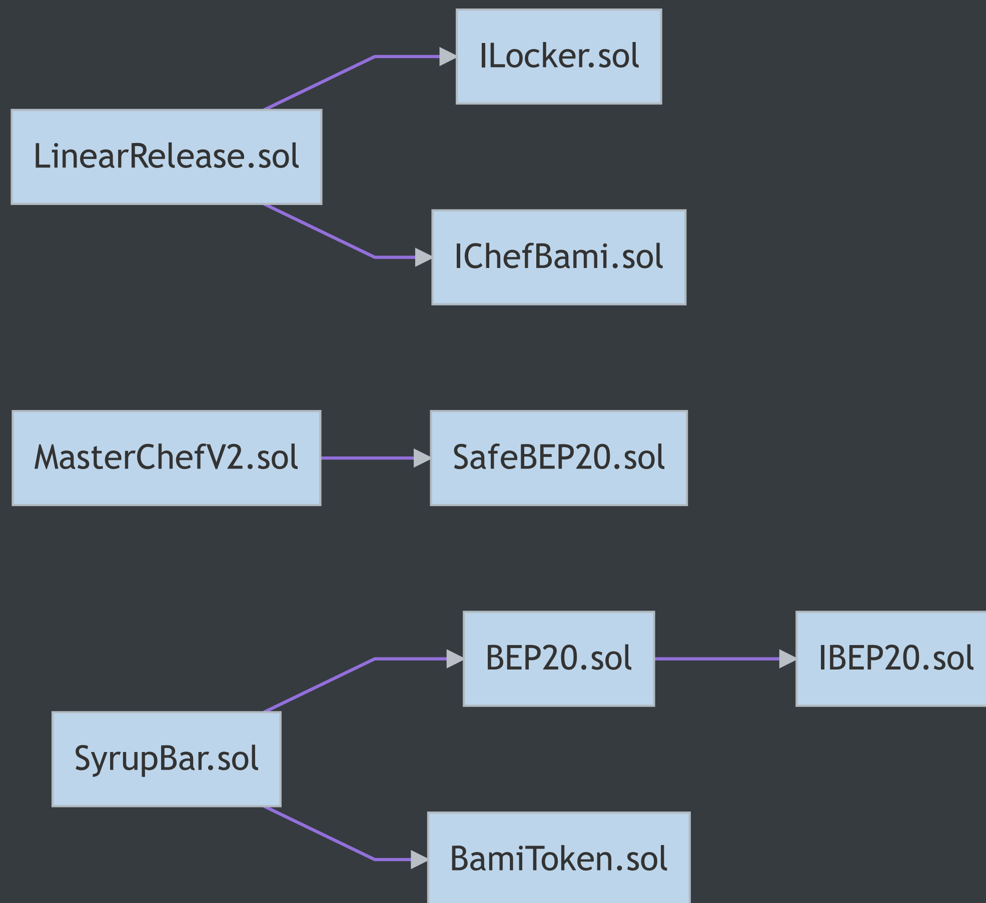
1. BamiToken is forked from SushiToken, no major changes.
2. SyrupBar is forked from SyrupBar from Pancakeswap, no major changes.
3. Timelock is forked from Timelock from Pancakeswap.
4. ChefBami is forked from MasterChef from SushiSwap, harvest functionality was added based on harvest from MasterChefV2 and locker contract is utilised.
5. MasterChefV2 is slightly modified ChefBami without harvest functionality. More in par with MasterChef from Pancakeswap.
6. LinearRelease is a new contract that adds locking functionality where tokens are being locked for specific timeframe.

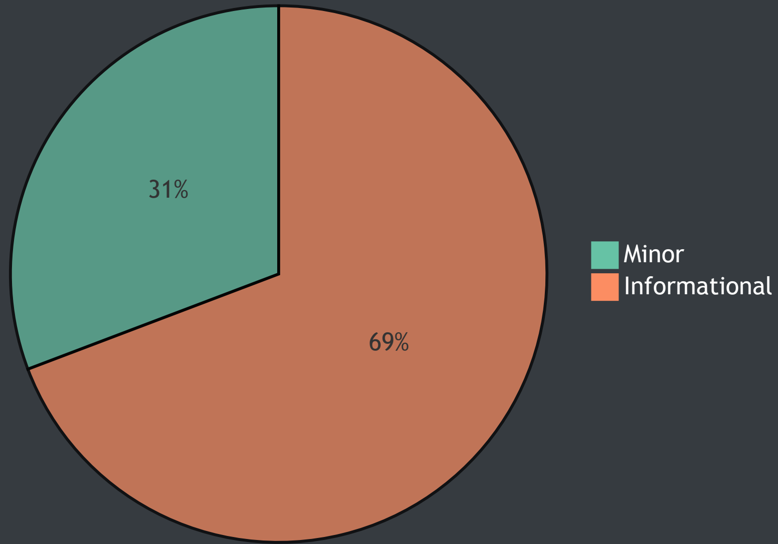
Most issues found are with LinearRelease contract. Changes and alteration to the forked code didn't introduced any vulnerabilities.

Files In Scope

ID	Contract	Location
BTN	BamiToken.sol	contracts/BamiToken.sol
CBI	ChefBami.sol	contracts/ChefBami.sol
LRE	LinearRelease.sol	contracts/LinearRelease.sol
MCV	MasterChefV2.sol	contracts/MasterChefV2.sol
SBR	SyrupBar.sol	contracts/SyrupBar.sol
TIM	Timelock.sol	contracts/Timelock.sol
BEP	BEP20.sol	contracts/libs/BEP20.sol
IBE	IBEP20.sol	contracts/libs/IBEP20.sol
ICB	IChefBami.sol	contracts/libs/IChefBami.sol
ILR	ILocker.sol	contracts/libs/ILocker.sol
MUL	Multicall.sol	contracts/libs/Multicall.sol
SBE	SafeBEP20.sol	contracts/libs/SafeBEP20.sol

File Dependency Graph




Finding Summary
31%
69%
Minor
Informational

Manual Review Findings

ID	Title	Type	Severity	Resolved
LRE-01M	Checks-effects-pattern not applied	Volatile Code	Minor	ⓘ
LRE-02M	LinearRelease should inherit from ILocker	Volatile Code	Informational	ⓘ
LRE-03M	Function returns arrays with single element	Coding Style	Informational	ⓘ
BEP-01M	Out-dated solidity version used	Volatile Code	Minor	ⓘ
BEP-02M	Unlocked Compiler Version	Language Specific	Informational	ⓘ
BEP-03M	BEP20 is based on old version of OpenZeppelin	Coding Style	Informational	ⓘ
IBE-01M	Unlocked Compiler Version	Language Specific	Informational	ⓘ
ILR-01M	ILocker is missing an event from LinearRelease.sol	Inconsistency	Informational	ⓘ
ILR-02M	Contract name is different from LinearRelease	Coding Style	Informational	ⓘ
MUL-01M	Arbitrary external calls	Volatile Code	Minor	ⓘ
MUL-02M	Unlocked Compiler Version	Language Specific	Informational	ⓘ
SBE-01M	Unlocked Compiler Version	Language Specific	Informational	ⓘ

Static Analysis Findings

ID	Title	Type	Severity	Resolved
CBI-01S	Unchecked Value of ERC-20 `approve()` Call	Volatile Code	Minor	ⓘ

LRE-01M: Checks-effects-pattern not applied

Type	Severity	Location
Volatile Code	Minor	LinearRelease.sol L58-L69, L97-L107

Description:

State variables are changed after transfer call to msg.sender.

Recommendation:

It is recommended to follow checks-effects-interactions pattern for cases like this.
It shields public functions from re-entrancy attacks. It's always a good practie to follow this pattern. `checks-effects-interaction` pattern also applies to ERC20 tokens as they can inform the recipient of a transfer in certain implementations.

LRE-02M: LinearRelease should inherit from ILocker

Type	Severity	Location
Volatile Code	Informational	LinearRelease.sol L13

Description:

LinearRelease contract should inherit from ILocker interface as its implements the same function.

Recommendation:

Contracts should iherit from interfaces the contract implements.

LRE-03M: Function returns arrays with single element

Type	Severity	Location
Coding Style	Informational	LinearRelease.sol L71

Description:

`pendingTokens` function returns `IERC20[] memory, uint256[] memory` when it could only return single IERC and uint256 variable. Before returning the values, _rewardTokens and _rewardAmounts variables are added to array of lenght 1 and then returned.

Recommendation:

We would recommend to simplify the return statements and only return basic variables instead of arrays in this case.

BEP-01M: Out-dated solidity version used

Type	Severity	Location
Volatile Code	Minor	BEP20.sol L3

Description:

solc frequently releases new compiler versions. Using an old version prevents access to new Solidity security checks.

Recommendation:

Deploy with any of the following Solidity versions:

- 0.5.16 - 0.5.17
- 0.6.11 - 0.6.12
- 0.7.5 - 0.7.6 Use a simple pragma version that allows any of these versions

BEP-02M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	BEP20.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

BEP-03M: BEP20 is based on old version of OpenZeppelin

Type	Severity	Location
Coding Style	Informational	BEP20.sol General

Description:

BEP20 is based on old version of OpenZeppelin 2.5.0. Currently newer version are available with support of newer versions of Solidity and it's features.

Recommendation:

We would recommend using ERC20 from OpenZeppelin from version 3.4 and up.

IBE-01M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	IBEP20.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

ILR-01M: ILocker is missing an event from LinearRelease.sol

Type	Severity	Location
Inconsistency	Informational	ILocker.sol L14-L15

Description:

ILocker only defines `Lock` event and not `Claim` event like LinearRelease contract is defining.

Recommendation:

Add `Claim` event to be compliant with LinearRelease contract

ILR-02M: Contract name is different from LinearRelease

Type	Severity	Location
Coding Style	Informational	ILocker.sol L7

Description:

Interface name is different than the contract that it is based off i.e. LinearRelease.

Recommendation:

We would recommend to standarize the interface and contract names to be the same when contract is based off an interface.

MUL-01M: Arbitrary external calls

Type	Severity	Location
Volatile Code	Minor	Multicall.sol L14-L22

Description:

This function can make any external call to any address and it's open for anybody to call it.

Recommendation:

We would restrict the option of calling this function by onlyOwner or governance only.

MUL-02M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	Multicall.sol L1

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

SBE-01M: Unlocked Compiler Version

Type	Severity	Location
Language Specific	Informational	SafeBEP20.sol L3

Description:

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to differing compiler version numbers. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation:

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

CBI-01S: Unchecked Value of ERC-20 `approve()` Call

Type	Severity	Location
Volatile Code	Minor	ChefBami.sol L302, L305

Description:

The linked `approve()` invocations do not check the return value of the function call which should yield a `true` result in case of a proper ERC-20 implementation.

Recommendation:

Return statement should be checked to be sure that the call was succesfull or not. We can also recommend using `safeApprove()` from OpenZeppelin SafeERC20 implementation that SafeBEP20 is also using.

Appendix

Finding Categories

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Language Specific

Language Specific findings are issues that would only arise within Solidity, i.e. incorrect usage of `private` or `delete`.

Coding Style

Coding Style findings usually do not affect the generated byte-code and comment on how to make the codebase more legible and as a result easily maintainable.

Inconsistency

Inconsistency findings refer to functions that should seemingly behave similarly yet contain different code, such as a `constructor` assignment imposing different `require` statements on the input variables than a setter function.