Ballena.io

DeFi Project Security Audit

Version 2: re-deployment of StratPancakeLpV2.sol

Audit tools used:

Audit date: April - May 2021

Auditor: Javier Calatrava (linkedin.com/in/javiercalatrava)

VERSION CONTROL

Version	DATE	Main Changes	Author
0.1	12th April 2021	First audit, Initial Draft	JC
1	21st May 2021	Initial version, Migration phase	JC
2	28th May 2021	Re-deployment of 2nd version of Strategy smart contracts	JC
3	31st May 2021	Typo corrections	JC

CONTENT TABLE

Content table

ABSTRACT

This is the executive summary of the audit report based on performed analysis, in accordance with best cybersecurity and code development practices as at the date of this report, in relation to cybersecurity, vulnerabilities and issues in the framework and algorithms based on smart contracts and DeFi platform development.

This report must not be considered, under any circunstance, as an investment advice, as its only intention is to asure the security of the platform and the strongness of provided Smart Contracts.

Ballena.io started as an upgraded fork from beefy project (original v1 of the project), with specific Smart Contract developed within construction phase.

Ballena.io community has developed a DeFi platform for the community. The management is a DAO (Decentralized Autonomous Organization) structure, which is better to avoid fraudulent actions from the project owners.

On top of this, Ballena.io has applied multi-sign to the treasury and pools management smart contracts, avoiding the single point of failure or a weakness point of trustworthy, due to have a sole owner of the smart contracts.

Ballena.io is using the Gnosis Safe multi-signature system on the Binance Smart Chain with three (3) different wallets, depending on the requirements (https://docs.ballena.io/dao-organizacion/gnosis):

- Governance protocol. The most critical wallet out of the Ballena.io Gnosis system. This will be in charge of daily actions within the platform and will be allowed to operate with all the smart contracts. It will use a 6/9 model, so 6 out of 9 signatures are required to confirm an action.
- Operations protocol. This will be in charge of less critical operations but still requiring a minimum number of signatures to not compromise the project security level. Actions like change the rewarding multiplier, re-start a vault or agregate a new vault are part of the normal operations protocol. It will use a 6/9 model, so 6 out of 9 signatures are required to confirm an action.
- Security emergency protocol. This will be reserved for emergency actions, but not transcendental actions impacting the project. Pause a vault or the token rewards are the actions that this wallet can confirm, mainly to avoid a dangerous situation like an attack or any other issue that could be mitigated by pausing the vaults or rewards. As said, is only pausing, so can't delete

vaults, create new vaults or any other action that must be approved by several administrators. Only 1 out of 9 signatures is required for these actions.

- Last but not least, there is also a Donations protocol to manage the donations. This wallet can't interact with any contract within the project. It will follow as well a 6/9 model. 6 signatures out of 9.

By implementing the multi signature governance model, Ballena.io is providing a higher level of trustworthy from their administration community. It is a community project for the community users.

DISCLAIMER / LEGAL NOTICE

The data on this report is for your information only. It does not constitute investment advice, or advice on tax or legal matters.

Any information provided on this website does not constitute investment advice or investment recommendation nor does it constitute an offer to buy or sell or a solicitation of an offer to buy or sell shares or units in any of the investment funds or other financial instruments described on this website. In addition, the information provided on this website does not contain any offer, recommendation or incitement to conclude any contracts for financial services (e.g. wealth management) or to conclude any other kind of contract (e.g. Family Office agreements). In particular, this information should not be used as a substitute for suitable investment and product-related advice. Unless expressly stated otherwise, all pricing information is non-binding.

Should you have any doubts about the meaning of the information provided herein, please contact your financial advisor or any other independent professional advisor.

Background

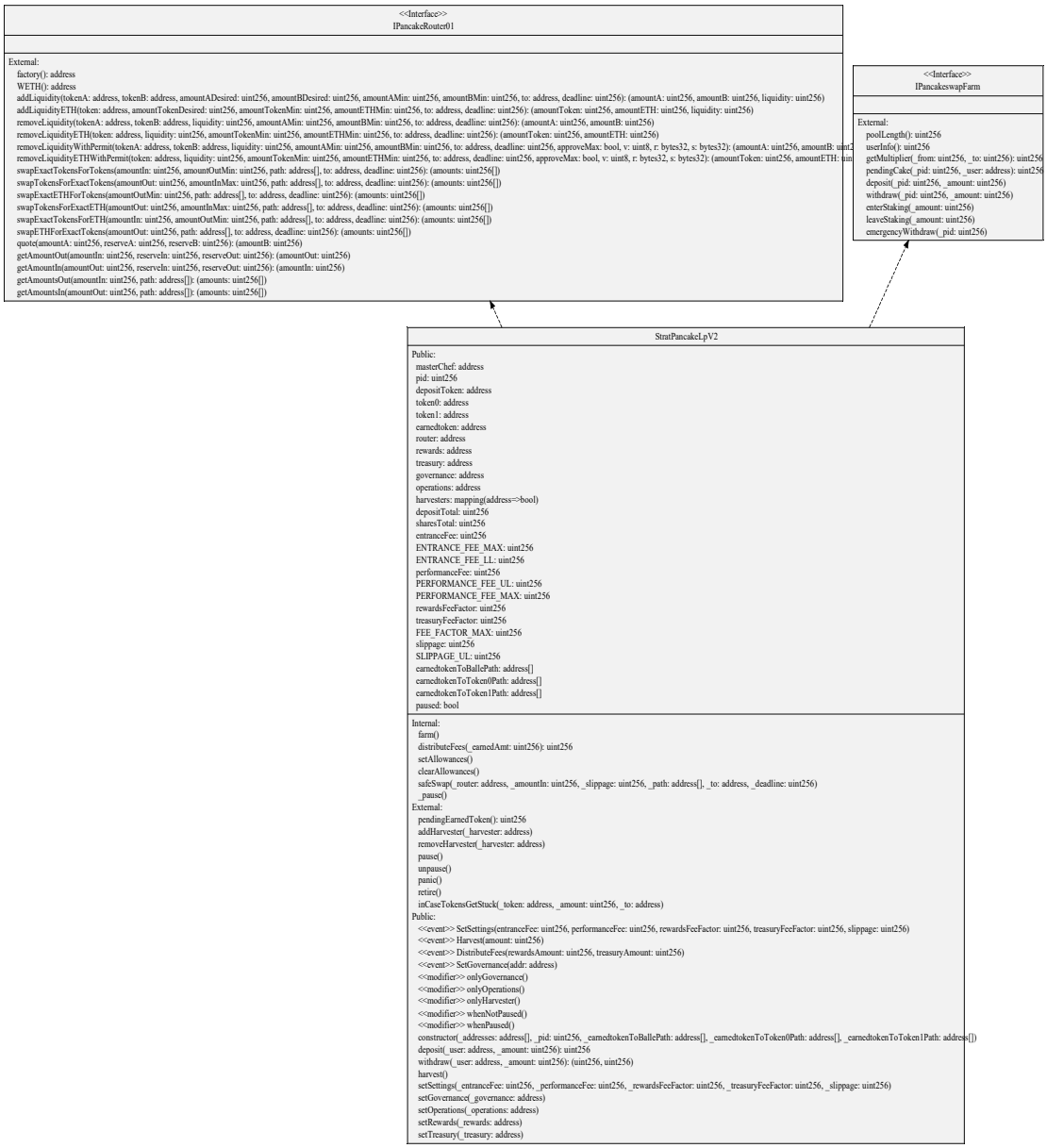
Following smart contracts have been audited within the scope of this audit, any new contract or change in audited contracts will be documented in future versions of this document:

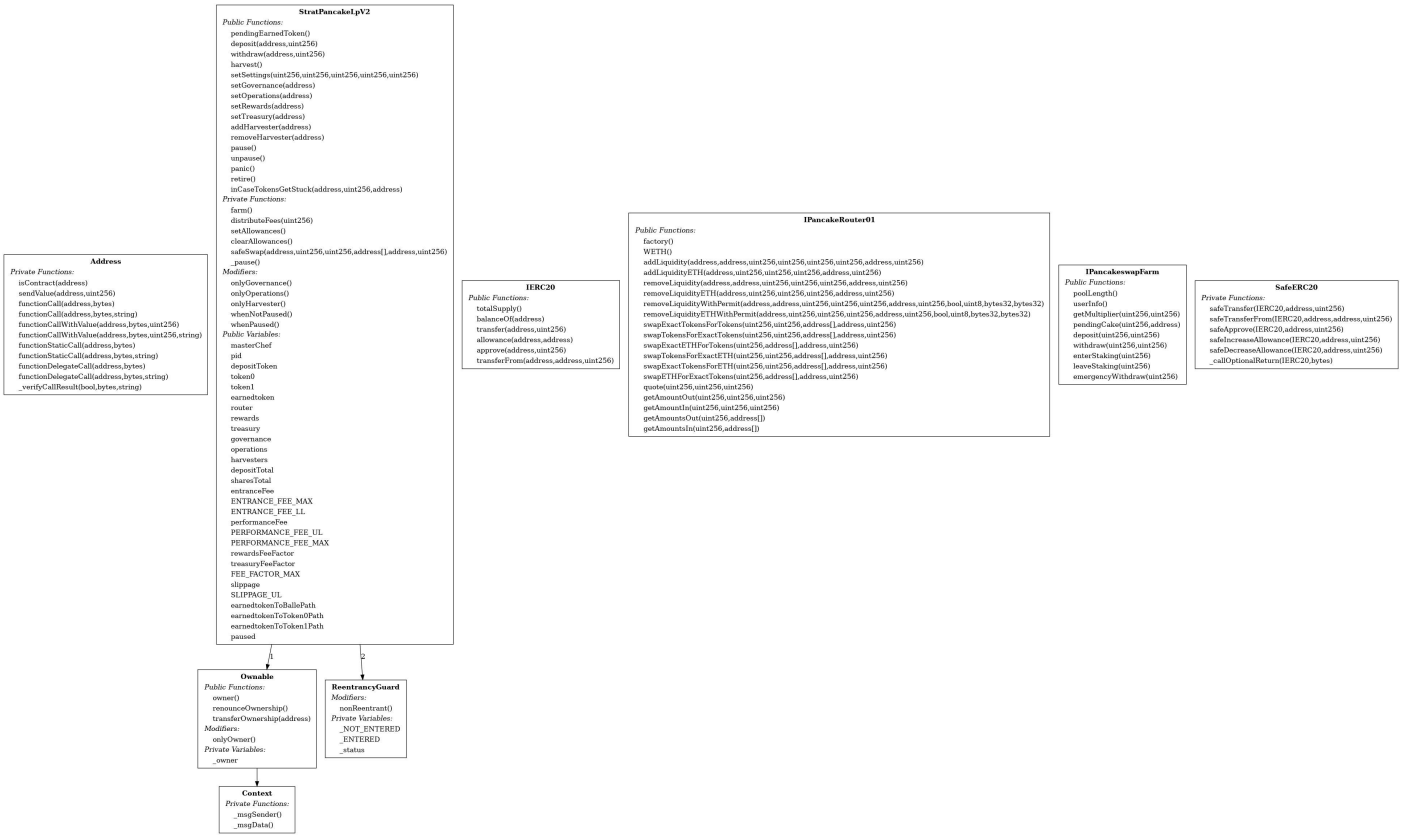
- ✔ *https://github.com/ballena-io/ballena-protocol/blob/master/contracts/strategies/StratPancakeLpV2.sol*

  Corresponding SC:
  https://bscscan.com/address/0x5034919684b3BEC9A4b0b134beF4b68660E36220

StratPancakeLpV2.sol

This is the PancakeSwap Strategy smart contract for Ballena.io, the first vaults strategies' contract. It is interacting with the main BalleMaster contract, which has been also audited , and also with other libraries from PancakeSwap and OpenZeppeling, as shown in the following draw:





Developers has decided to used “battle tested” code instead of create the specific piece of code in the main contracts, this could be considered as a potential issue, but once the contract is compiled and deployed, won't be impacting anymore. Otherwise, specific pieces of code, even if it is created with the same intention or purpose, could have new vulnerabilities, like recent cases where a small different in a single code line, has provoqued the drain of investors' funds in well know projects.

The following information is a quick summary of **StratPancakeLpV2**.sol contract, showing the interaction with the specific functions on the other contracts linked to it.

While deploying PancakeSwap vaults, all the vault's contract will have the same StratPancakeLpV2.sol code.

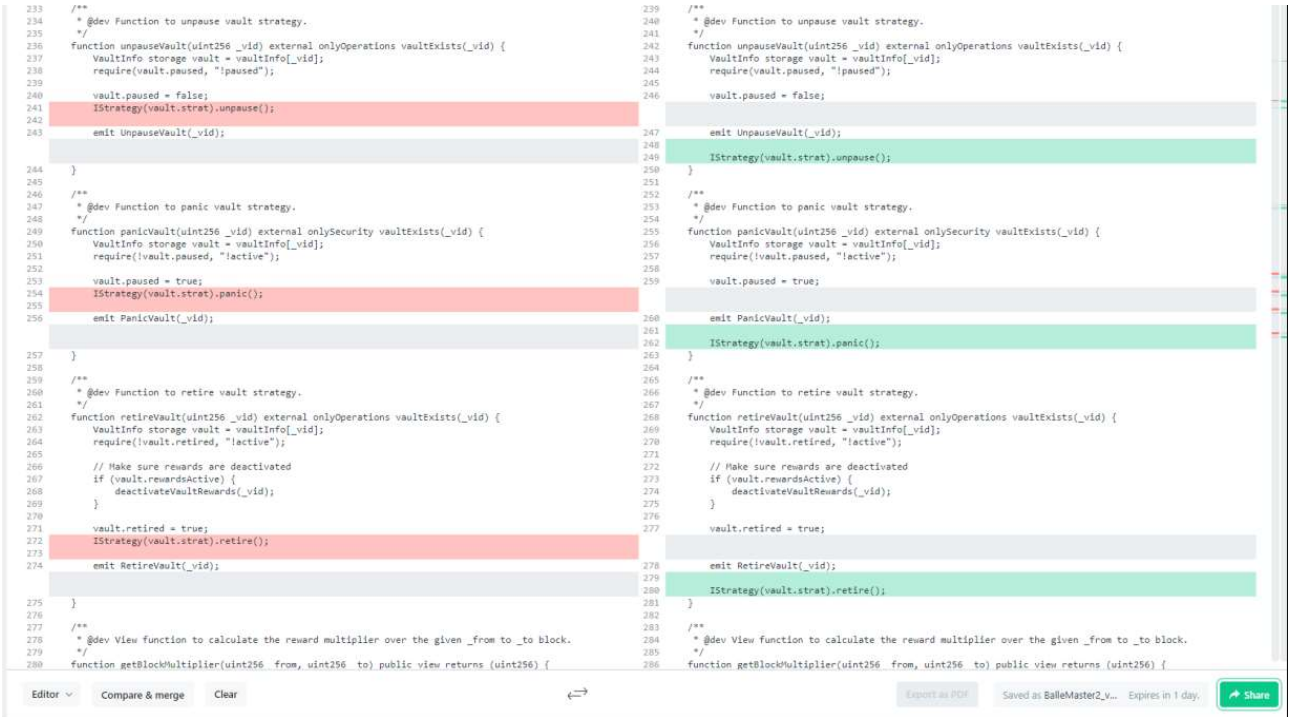
```
+ Contract Address (Most derived contract)
  - From Address
    - _verifyCallResult(bool,bytes,string) (private)
    - functionCall(address,bytes) (internal)
    - functionCall(address,bytes,string) (internal)
    - functionCallWithValue(address,bytes,uint256) (internal)
    - functionCallWithValue(address,bytes,uint256,string) (internal)
    - functionDelegateCall(address,bytes) (internal)
    - functionDelegateCall(address,bytes,string) (internal)
    - functionStaticCall(address,bytes) (internal)
    - functionStaticCall(address,bytes,string) (internal)
    - isContract(address) (internal)
    - sendValue(address,uint256) (internal)

+ Contract Context
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)

+ Contract IERC20 (Most derived contract)
  - From IERC20
    - allowance(address,address) (external)
    - approve(address,uint256) (external)
    - balanceOf(address) (external)
    - totalSupply() (external)
    - transfer(address,uint256) (external)
    - transferFrom(address,address,uint256) (external)

+ Contract IPancakeRouter01 (Most derived contract)
  - From IPancakeRouter01
    - WETH() (external)
    - addLiquidity(address,address,uint256,uint256,uint256,uint256,address,uint256) (external)
    - addLiquidityETH(address,uint256,uint256,uint256,address,uint256) (external)
    - factory() (external)
    - getAmountIn(uint256,uint256,uint256) (external)
    - getAmountOut(uint256,uint256,uint256) (external)

+ Contract IPancakeswapFarm (Most derived contract)
  - From IPancakeswapFarm
    - deposit(uint256,uint256) (external)
    - emergencyWithdraw(uint256) (external)
    - enterStaking(uint256) (external)
    - getMultiplier(uint256,uint256) (external)
    - leaveStaking(uint256) (external)
    - pendingCake(uint256,address) (external)
    - poolLength() (external)
    - userInfo() (external)
    - withdraw(uint256,uint256) (external)

+ Contract Ownable
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)

+ Contract StratPancakeLpV2 (Most derived contract)
  - From ReentrancyGuard
    - constructor() (internal)
  - From Ownable
    - owner() (public)
    - renounceOwnership() (public)
    - transferOwnership(address) (public)
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)
  - From StratPancakeLpV2
    - _pause() (internal)
    - addHarvester(address) (external)
    - clearAllowances() (internal)
    - constructor(address[],uint256,address[],address[],address[]) (public)
    - deposit(address,uint256) (public)
    - distributeFees(uint256) (internal)
    - farm() (internal)
    - harvest() (public)
    - inCaseTokensGetStuck(address,uint256,address) (external)
    - panic() (external)
    - pause() (external)
    - pendingEarnedToken() (external)
    - removeHarvester(address) (external)
    - retire() (external)
    - safeSwap(address,uint256,uint256,address[],address,uint256) (internal)
    - setAllowances() (internal)
    - setGovernance(address) (public)
    - setOperations(address) (public)
    - setRewards(address) (public)
    - setSettings(uint256,uint256,uint256,uint256,uint256) (public)
    - setTreasury(address) (public)
    - unpause() (external)
    - withdraw(address,uint256) (public)
```

Audit process has been done in several phases, initially to allow the Ballena.io development team to solve the potential code issues. Later, to check the “almost final” version and finally, to ensure that code is robust and there are no issues impacting the users and their investments.

Within the initial phase of audit, few minor issues were found on javascript code, these issues were solved for the pre-production testing phase audit.

Version 2 of the audited smart contract was created after the resolution of these audit findings and re-deployment due to small discrepancy with the BALLE token rewards.

Some of these issues were solved by changing some code order as shown in the following picture:



Regarding the Smart Contracts, only few issues were found and all the recommendations have been applied.

Testing plan

Ballena.io development team has deeply tested their code with different functional and non functional test cases to ensure the code is working propertly.

- Test cases within the Ballena.io testing plan has been passes successfully.

They have simulated different potential attacks and they found the code and smart contracts are robust and safe to these kind of scenarios.

- Reentracy attacks
- Fast loan attacks
- Governance impersonation attacks

Ballena.io is not vulnerable to Reentrancy attacks due to two main reasons, the first one is the usage of ReentrancyGuard function, intended to avoid these kind os scenarios. The second one is the updated version of compiler, which is not anymore vulnerable to this kind of atack.

Regarding the Fast loans attack, Ballena.io is not a farming project and the BALLE token is minted in the same amount regardless the investment amount of one single user. The minted amount is later on distributed between all the vaults investors, proportionally to their investments, so in the even one investor is using fast loans to invest in Ballena.io vaults, th TVL in the project will rapidly increase and that will report a benefit to the other investors, as the deposit fee will be spread within the previous investors, the potential attacker won't obtain an extra payment not directly derived from the investment, as it is not based on variable minting.

Regarding the Governance impersonation, as the project governance wallet is a Gnosis wallet, the potential attack should obtain access to a 66% of the users with signature power on the Gnosis Governance multisig, so this will difficult the attack.

Audit scope

- Audit of smart contracts to avoid negative impact on users.
- Audit of application code to avoid potential security vulnerabilities.

This audit is verifying the following audits points (with current status):

- Deny of Service attacks based on REVERT actions. CORRECT
- Deny of Service attacks based on gas limitation. CORRECT
- Warnings at compilation time. CORRECT
- Reentrancy and race conditions issues. CORRECT
- Cross-functions race conditions issues. CORRECT
- Overflow and underflow issues. CORRECT
- Logic of the economy model. CORRECT
- Exchange rate impact. CORRECT
- Oracle calls. CORRECT
- Timestamp dependace. CORRECT
- Potential delays in data delivery. CORRECT
- Permissions on execution methods. CORRECT
- Private data leakage prevention. CORRECT

Audit findings

With MythX, each smart contract is compiled individually and checked for a range of security issues using static and dynamic analysis. The following table lists the bug classes that were tested for. A checkmark in the "pass" column indicates that no issues were detected in the category. An "X" indicates that one or more issues in the category were found.

SWC ID	Bug class	Pass
SWC-100	Function Default Visibility	✔
SWC-101	Integer Overflow and Underflow	✔
SWC-102	Outdated Compiler Version	✖
SWC-103	Floating Pragma	✔
SWC-104	Unchecked Call Return Value	✔
SWC-105	Unprotected Withdrawal	✔
SWC-106	Unprotected SELFDESTRUCT Instruction	✔
SWC-107	Reentrancy	✔
SWC-108	State Variable Default Visibility	✔
SWC-109	Uninitialized Storage Pointer	✔
SWC-110	Assert Violation	✔
SWC-111	Use of Deprecated Solidity Functions	✔
SWC-112	Delegatecall to Untrusted Callee	✔
SWC-113	DoS with Failed Call	✔
SWC-114	Transaction Order Dependence	✔
SWC-115	Authorization through tx.origin	✔
SWC-116	Timestamp Dependence	✔
SWC-118	Incorrect Constructor Name	✔
SWC-119	Shadowing State Variables	✔

SWC-120	Weak Sources of Randomness	✔
SWC-123	Requirement Violation	✔
SWC-124	Write to Arbitrary Storage Location	✔
SWC-127	Arbitrary Jump	✔
SWC-128	Gas Exhaustion	✔
SWC-129	Typographical Error	✔
SWC-130	Right-To-Left-Override control character	✔

High Severity Findings

- No high severity issues found.

Medium Severity Findings

- No medium severity issues found.

Low Severity Findings and Recommendations

- No low severity issues found. Three recommendations.

1. **inCaseTokensGetStuck functions**

```
525
526     /**
527      * @dev Function to use from Governance Gnosis Safe multisig only in case tokens get stuck.
528      * This is to be used if someone, for example, sends tokens to the contract by mistake.
529      * There is no guarantee governance will vote to return these.
530      */
531     function inCaseTokensGetStuck(
532         address _token,
533         uint256 _amount,
534         address _to
535     ) external onlyGovernance {
536         require(_to != address(0), "zero address");
537         require(_token != earnedtoken, "!safe");
538         require(_token != depositToken, "!safe");
539
540         IERC20(_token).safeTransfer(_to, _amount);
541     }
542 }
```

StratPancakeLpV2.sol "inCaseTokensGetStuck" function

Ballena.io smart contracts have the “inCaseTokenGetStuck” function, that could be considered as a dangerous function, but as can be seen in the code above, it is created to not allow the withdrawal of neither the earned nor the deposited tokens in the StratPancakeLpV2.sol contract.

The reason to have these functions is to be able to return tokens sent by mistake to the Ballena.io contracts, by users that could make these mistakes.
On top of this, only the Gnosis Governance wallet will be allowed to use these functions, and as described in the Ballena.io documents, the DAO must appove first the action.

2. **Zero address checking**
   Recommendation:
   Add zero address checking functions. Developer has added before deployment of contracts

3. **Prefer External to public function declaration**
   Recommendation:
   Use the external visibility modifier for functions never called from the contract via internal call. Most of the functions have been updated to external functions.

Compiler version is a recurrent informational finding in these contracts. After discussion with developer, he has stated that the compiler version used for the development is not the last one, because the previous version is battle tested and due to this, he can apply better coding and avoid potential unknown issues to the newer version.

The contracts are compiling propertly and on top of this, developer has agreed to allow compiler versions above the version he has used at coding time.

There is another finding not related to code but related with the Gnosis multi signarute system usage. It is documented that main protocols will follow a 6/9 model.

The governance multisig is controlled by 9 community members and requires 6 signatures. This means that no transactions can be processed until the 9 signers have reached at least 66.67% consensus.

But at the moment of this audit, the Gnosis system is working under a 3/5 model due to a technical limitation in the Binance deployment of Gnosis, which avoid the community to increase the number of signatures from 5 to 9.

After confirmation with developers, they are aware of this and they already have an

open ticket with Binance for the resolution of this limitation. This point will be updated in following versions of this document.

Conclusion

At the moment of conducting these audits, current deployed Ballena.io smart contracts, do not contain any known security issue. All minor issues and recommendations have been solved or implemented before deployment.

On top of this, code and functionalities have been deeply tested by the community users, following a testing plan, before the final deployment of contracts.

Ballena.io application code is secure and do not have any security issue before live version.