HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Powerbomb Finance
Date: May 03rd, 2022

Hacken OÜ
Parda 4, Kesklinn, Tallinn,
10151 Harju Maakond, Eesti,
Kesklinna, Estonia
support@hacken.io

Document

Name	Smart Contract Code Review and Security Analysis Report for Powerbomb Finance.
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC20 token depositing
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.powerbomb.finance/
Timeline	18.04.2022 - 03.05.2022
Changelog	26.04.2022 - Initial Review 29.04.2022 - Second Review 03.05.2022 - Third Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Powerbomb Finance (Customer) to conduct a Smart Contract Code Review and Security Analysis. This report presents the findings of the security assessment of the Customer's smart contracts.

Scope

The scope of the project is smart contracts in the repository:

Initial review scope

Repository:
https://github.com/Powerbomb-Finance/MVP/blob/develop
Commit:
5dcdb944b8c320aa4e804cd0709acbc0ebc48484
Technical Documentation: Yes
(https://github.com/Powerbomb-Finance/MVP/blob/develop/hardhat/docs/powerbombAvaxAnc.md#technical-explanation)
JS tests: Yes
Contracts:

```
File: ./hardhat/contracts/PowerBombAvaxAnc.sol
SHA3: 9304ff6212ab678b47d260bbbca54aaf6640e30832937d003893eda1068ece84

File: ./hardhat/contracts/PowerBombAvaxAncSAVAX.sol
SHA3: 2cd6351be3bd85ce020aa4323b4c87ece420516a964cf6938d47784ac760e381

File: ./hardhat/contracts/PowerBombAvaxAncHelper.sol
SHA3: c31dc869092b7b0d308a85bfda95425386c9d4508200950b14106928f895a85f
```

Second review scope

Repository:
https://github.com/Powerbomb-Finance/MVP/blob/develop
Commit:
db121bd2b4baccec4dc53ac9439be751a04f82ba
Technical Documentation: Yes
(https://github.com/Powerbomb-Finance/MVP/blob/develop/hardhat/docs/powerbombAvaxAnc.md#technical-explanation)
JS tests: Yes
Contracts:

```
File: ./hardhat/contracts/PowerBombAvaxAnc.sol
SHA3: 59f7f1c3f5cb37dca998bf921f25d9cf465bedf61328f900050a378e9283710d

File: ./hardhat/contracts/PowerBombAvaxAncSAVAX.sol
SHA3: e00b0888752f775ae4912b81d28344f43c5e37bc512ae6db534599aa3f0443dc
```

Second review scope

Repository:
https://github.com/Powerbomb-Finance/MVP/blob/develop
Commit:
f899e5fe224973ff7433f1d26f5541f4819b9aaf
Technical Documentation: Yes
(https://github.com/Powerbomb-Finance/MVP/blob/develop/hardhat/docs/powerbombAvaxAnc.md#technical-explanation)

```
JS tests: Yes
Contracts:
      File: ./hardhat/contracts/PowerBombAvaxAnc.sol
      SHA3: 3cc24437681cf142f9c15510b317d921ada5ef6a9ce4bc170bfcabaaac0d9448

      File: ./hardhat/contracts/PowerBombAvaxAncSAVAX.sol
      SHA3: babd911ae92603ce9228502e66a0173665d1ac45f9b50983f3ecc4fd31cbdfb0
```

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
High	High-level vulnerabilities are difficult to exploit; however, they also have a significant impact on smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
Low	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

Executive Summary

The score measurements details can be found in the corresponding section of the methodology.

Documentation quality

The Customer provided technical information and detailed comments in the contracts and no functional requirements. The total Documentation Quality score is **5** out of **10**.

Code quality

The total CodeQuality score is **7** out of **10**. Unit tests were provided.

As a result of the second review, the total CodeQuality is **10** out of **10**.

Architecture quality

The architecture quality score is **10** out of **10**. The architecture is clear.

Security score

As a result of the initial review, security engineers found **1** critical, **4** high, **2** medium, and **7** low severity issues. The security score is **0** out of **10**.

As a result of the second review, security engineers found **1** new high and **1** low severity issues. **1** critical, **3** high, **2** medium, and **2** low severity issues from the previous revision were fixed. **1** high and **3** low severity issues from the previous revision were mitigated. As a result, the code contains **1** high and **3** low severity issues. The security score is **5** out of **10**

As a result of the third review, security engineers found **no** new severity issues. **1** high and **2** low severity issues from the previous revision were fixed. As a result, the code contains **1** low severity issue. The security score is **10** out of **10**

All found issues are displayed in the “Findings” section.

Summary

According to the assessment, the Customer's smart contract has the following score: **9.5**.

1
2
3
4
5
6
7
8
9
10
You are here

Checked Items

We have audited provided smart contracts for commonly known and more specific vulnerabilities. Here are some of the items that are considered:

Item	Type	Description	Status
Default Visibility	SWC-100 SWC-108	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	SWC-101	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	SWC-102	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	SWC-103	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	SWC-104	The return value of a message call should be checked.	Passed
Access Control & Authorization	CWE-284	Ownership takeover should not be possible. All crucial functions should be protected. Users could not affect data that belongs to other users.	Passed
SELFDESTRUCT Instruction	SWC-106	The contract should not be destroyed until it has funds belonging to users.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Uninitialized Storage Pointer	SWC-109	Storage type should be set explicitly if the compiler version is < 0.5.0.	Not Relevant
Assert Violation	SWC-110	Properly functioning code should never reach a failing assert statement.	Not Relevant
Deprecated Solidity Functions	SWC-111	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	SWC-112	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	SWC-113 SWC-128	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed

Race Conditions	SWC-114	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization through tx.origin	SWC-115	tx.origin should not be used for authorization.	Passed
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	SWC-117 SWC-121 SWC-122	Signed messages should always have a unique id. A transaction hash should not be used as a unique id.	Not Relevant
Shadowing State Variable	SWC-119	State variables should not be shadowed.	Passed
Weak Sources of Randomness	SWC-120	Random values should never be generated from Chain Attributes.	Not Relevant
Incorrect Inheritance Order	SWC-125	When inheriting multiple contracts, especially if they have identical functions, a developer should carefully specify inheritance in the correct order.	Passed
Calls Only to Trusted Addresses	EEA-Level-2 SWC-126	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	SWC-131	The code should not contain unused variables if this is not justified by design.	Passed
EIP standards violation	EIP	EIP standards should not be violated.	Not Relevant
Assets integrity	**Custom**	Funds are protected and cannot be withdrawn without proper permissions.	Passed
User Balances manipulation	**Custom**	Contract owners or any other third party should not be able to access funds belonging to users.	Passed
Data Consistency	**Custom**	Smart contract data should be consistent all over the data flow.	Passed
Flashloan Attack	**Custom**	When working with exchange rates, they should be received from a trusted source and not be vulnerable to short-term rate changes that can be achieved by using flash loans. Oracles should be used.	Not Relevant
Token Supply manipulation	**Custom**	Tokens can be minted only according to rules specified in a whitepaper or any other documentation provided by the customer.	Not Relevant

Gas Limit and Loops	**Custom**	Transaction execution costs should not depend dramatically on the amount of data stored on the contract. There should not be any cases when execution fails due to the block gas limit.	Passed
Style guide violation	**Custom**	Style guides and best practices should be followed.	Passed
Requirements Compliance	**Custom**	The code should be compliant with the requirements provided by the Customer.	Passed
Repository Consistency	**Custom**	The repository should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Passed
Tests Coverage	**Custom**	The code should be covered with unit tests. Tests coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	Passed

System Overview

Powerbomb Avalanche is a system for deposits with the following contracts:

- PowerBombAvaxAnc — a contract that allows users to deposit UST tokens and receive a reward in the token specified by the owner. Invested UST tokens are deposited to the Anchor pool (exchanged for aUST tokens).

  Tokens withdrawal consists of calling withdraw function (aUST tokens are redeemed from Anchor to UST (transferred to the contract)) by user and funds transferring through calling repayWithdraw function. A fee is charged for withdrawing funds. If withdrawing within 1 week, the fee is calculated by the defined percent(0.1%), the minimal fee is 2 UST, and the maximal fee is 100 UST. If withdrawing after 1 week, the fee amount is 2 UST.

  The reward amount depends on the size of the harvest collected by the contract. Harvesting can be initiated by anyone. The UST available in the contract is exchanged for a WAVAX token, and the award is collected from AAVE in WAVAX tokens. Then the total amount of WAVAX tokens with the fee removed (5%) is taken into account in calculating rewards for users, exchanged for a reward token, and deposited with AAVE. When users receive rewards, reward tokens are withdrawn from Aave.

  There is pausing functionality in the contract. Pausing stops depositing into contracts.
- PowerBombAvaxAncSAVAX — a contract inherited from PowerBombAvaxAnc, but the harvesting process differs. The UST available in the contract is exchanged for a WAVAX token and is staked into Benqi. The staked amount with the removed fee is taken into account of rewards sizes for users. The rewards are paid in the sAVAX tokens.

Privileged roles

- The owners of PowerBombAvaxAnc and PowerBombAvaxAncSAVAX contracts can set admins and treasury (the address where collected fees are sent) addresses, set the limit of the total deposited amount. The owners can pause and unpause contracts.

Findings

🟥🟥🟥🟥 Critical

An assignment to undeclared variable.

The contract attempts to assign a value to a “depositFeePerc” variable, which has not previously been declared.

Therefore, the contract cannot be compiled.

Contracts: PowerBombAvaxAncSAVAX.sol

Functions: initialize

Recommendation: remove the assignment to the undeclared variable.

Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

🟨🟨🟨 High

1. **Highly permissive owner and admin access.**

   The functionality allows owners and admins of contracts to redeem any amount of aUST tokens through the Anchor pool.

   This means that users' funds (aUST) can be exchanged for UST.

   Contracts: PowerBombAvaxAnc.sol, PowerBombAvaxAncSAVAX

   Functions: PowerBombAvaxAnc.initializeHarvest, PowerBombAvaxAncSAVAX.initializeHarvest

   Recommendation: ensure that users' funds are safe when redeeming extra UST.

   Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

2. **Not guaranteed users' funds withdrawal.**

   One of the stages of withdrawing users' funds is claiming aUST tokens from PowerBombAvaxAnc/PowerBombAvaxAncSAVAX contract to PowerBombAvaxAncHelper contract by the owner or user with “admin” role.

   Users are not guaranteed to withdraw their funds.

   Contracts: PowerBombAvaxAncHelper.sol, PowerBombAvaxAnc.sol, PowerBombAvaxAncSAVAX.sol

   Functions: PowerBombAvaxAncHelper.executeWithdraw, PowerBombAvaxAnc.withdraw

   Recommendation: ensure that users will withdraw their funds.

   Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

3. **Fees percentage changing.**

   The owner can change “_withdrawFeePerc”, “_yieldFeePerc” anytime.

   Some users may disagree with the new rules.

   Contracts: PowerBombAvaxAnc.sol

   Functions: setFeePerc

   Recommendation: make a two-step percentage change or take away the right to change percentages anytime.

   Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

4. **No rewards for users.**

   The contract stakes WAVAX tokens into Benqi, but funds for awards are not withdrawn from there and are not exchanged for reward tokens.

   This means that users will not get the rewards.

   Contracts: PowerBombAvaxAncSAVAX.sol

   Functions: harvest, claimReward

   Recommendation: ensure that users will withdraw their funds.

   Status: Mitigated. The Customer comment: “Rewards are in sAVAX token directly”

5. **Insufficient funds and early withdrawal of funds from Anchor.**

   The contract allows redeeming funds from Anchor for fees an unlimited number of times before they are paid.

   Funds from Anchor will be withdrawn prematurely. There will be fewer of them on the Anchor than expected and because of this users will not be able to withdraw their funds.

   Contracts: PowerBombAvaxAncSAVAX.sol

   Functions: claimFees

   Recommendation: limit the redeeming for fees, ensure that users will withdraw their funds.

   Status: Fixed (Revised commit: f899e5fe224973ff7433f1d26f5541f4819b9aaf)

■■ Medium

1. **Check-Effect-Interaction pattern violation.**

   The state variables are changed after the external calls are executed.

   Executing external calls after changing the state protects the contract from re-entrance and race conditions if calling ambiguous contracts.

Contracts: PowerBombAvaxAnc.sol, PowerBombAvaxAncSAVAX

Functions: PowerBombAvaxAnc.harvest, PowerBombAvaxAnc.claimReward, PowerBombAvaxAncSAVAX.harvest

Recommendation: ensure all internal state changes are performed before the external calls are executed or use a reentrancy lock.

Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

2. **Possibility to exceed the UST limit specified in the contract.**

   When depositing funds, it is checked whether the limit has not been exceeded without taking into account the amount to be invested now.

   This means that the previously deposited amount together with the newly added one may exceed the set limit.

   Contracts: PowerBombAvaxAnc

   Functions: deposit

   Recommendation: ensure that the limit will not be exceeded.

   Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

■ Low

1. **Unused variable.**

   Field “_gap” is never used.

   Contracts: PowerBombAvaxAnc.sol

   Function: -

   Recommendation: remove unused variable.

   Status: Mitigated. The customer comment: “Refer Openzeppelin (Using with Upgrades - OpenZeppelin Docs)”

2. **No caller verification in the initialization function.**

   There is no restriction that only the owner can call the initialization function.

   Anyone can call the initialization function.

   Contracts: PowerBombAvaxAncHelper.sol, PowerBombAvaxAnc.sol, PowerBombAvaxAncSAVAX.sol

   Functions: PowerBombAvaxAncHelper.initialize, PowerBombAvaxAnc.initialize, PowerBombAvaxAncSAVAX.initialize

   Recommendation: it is better to add only owner access to the initialization function.

Status: Mitigated. The customer comment: “onlyOwner modifier can't be used before __Ownable_init() due to OpenZeppelin upgradeable contract pattern”

3. **Using state variables default visibility.**

   The visibility of PowerBombAvaxAncHelper.UST, PowerBombAvaxAncHelper.aUST, PowerBombAvaxAncHelper.pool, PowerBombAvaxAncHelper.admin, PowerBombAvaxAncSAVAX.sAVAX, PowerBombAvaxAnc.UST, PowerBombAvaxAnc.WAVAX, PowerBombAvaxAnc.lpToken, PowerBombAvaxAnc.pool, PowerBombAvaxAnc.router, PowerBombAvaxAnc.admin, PowerBombAvaxAnc.basePool, PowerBombAvaxAnc.lendingPool, PowerBombAvaxAnc.incentivesController variables are not labeled.

   It is recommended to label visibility to catch incorrect assumptions about who can access the variable.

   Contracts: PowerBombAvaxAncHelper.sol, PowerBombAvaxAnc.sol, PowerBombAvaxAncSAVAX.sol

   Functions: PowerBombAvaxAnc.harvest, PowerBombAvaxAnc.claimReward, PowerBombAvaxAncSAVAX.harvest

   Recommendation: explicitly mark visibility for state variables.

   Status: Fixed (Revised commit: f899e5fe224973ff7433f1d26f5541f4819b9aaf)

4. **Functions that can be declared as external.**

   There are public functions in the contracts that are not used internally.

   “External” visibility uses less gas.

   Contracts: PowerBombAvaxAnc.sol, PowerBombAvaxAncSAVAX.sol

   Functions: PowerBombAvaxAnc.harvest, PowerBombAvaxAncSAVAX.harvest

   Recommendation: replace the visibilities to “external”.

   Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

5. **Redundant functions.**

   The “getUserBalance”, “getUserBalanceInUSD” functions do the same. The “getAllPoolInUSD” function returns the result of the “getAllPool” function calling.

   Contracts: PowerBombAvaxAnc.sol

   Functions: getUserBalance, getUserBalanceInUSD, getAllPool, getAllPoolInUSD

   Recommendation: remove the redundant functions.

   Status: Reported

6. **Exchange rate defining.**

The owner or admin defines the exchange rate for UST/aUST.

This means that the rate may be incorrect.

Contracts: PowerBombAvaxAncHelper.sol

Functions: executeWithdraw, claimFees

Recommendation: it is recommended to receive the exchange rate from a trusted source.

Status: Fixed (Revised commit: db121bd2b4baccec4dc53ac9439be751a04f82ba)

7. **Using block numbers for time calculations**

   The contract uses block.timestamp for time calculations. It is not precise and safe.

   Contracts: PowerBombAvaxAnc.sol

   Functions: deposit

   Recommendation: it is recommended to avoid using block.timestamp. Alternatively, it is safe to use oracles.

   Status: Mitigated. The Customer comment: “No exact timestamp needed”

8. **Redundant functionality.**

   The role “admin” is never used. The function of setting the admin and “admin” variable are redundant.

   Contracts: PowerBombAvaxAnc.sol

   Functions: initialize, setAdmin

   Recommendation: remove the redundant code.

   Status: Fixed (Revised commit: f899e5fe224973ff7433f1d26f5541f4819b9aaf)

Disclaimers

Hacken Disclaimer

The smart contracts given for audit have been analyzed by the best industry practices at the date of this report, with cybersecurity vulnerabilities and issues in smart contract source code, the details of which are disclosed in this report (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

The audit makes no statements or warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other contract statements. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only — we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit cannot guarantee the explicit security of the audited smart contracts.