



**HACKEN**

# SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

**Customer:** PowerBomb

**Date:** July 01<sup>st</sup>, 2022

This report may contain confidential information about IT systems and the intellectual property of the Customer, as well as information about potential vulnerabilities and methods of their exploitation.

The report can be disclosed publicly after prior consent by another Party. Any subsequent publication of this report shall be without mandatory consent.

## Document

<b>Name</b>	Smart Contract Code Review and Security Analysis Report for PowerBomb
<b>Approved By</b>	Evgeniy Bezuglyi   SC Audits Department Head at Hacken OU
<b>Type</b>	Uniswap V3 Arbitrum
<b>Platform</b>	EVM
<b>Network</b>	Arbitrum
<b>Language</b>	Solidity
<b>Methods</b>	Manual Review, Automated Review, Architecture review
<b>Website</b>	<a href="https://www.powerbomb.finance/">https://www.powerbomb.finance/</a>
<b>Timeline</b>	20.06.2022 - 01.07.2022
<b>Changelog</b>	29.06.2022 - Initial Review 01.07.2022 - Second Review



## Table of contents

Introduction	4
Scope	4
Severity Definitions	5
Executive Summary	6
Checked Items	7
System Overview	10
Findings	11
Disclaimers	12

## Introduction

Hacken OÜ (Consultant) was contracted by PowerBomb (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/powerbomb-foundry/blob/univ3>

#### Commit:

157f70349723a238485ae1e1c7b38846144502e5

#### Technical Documentation:

Type: ReadMe

[Link](#)

#### Integration and Unit Tests: Yes

##### Contracts:

File: ./src/PbUniV3.sol

SHA3: 664a27e83855271596a5cf457ce2103d3b77d532c935a798f1f7342a1d5390d3

File: ./src/PbUniV3Proxy.sol

SHA3: 42717dafa30672ba6625e90448ee813209258b535ab7173134d0cb16a4d76f15

File: ./src/PbUniV3Reward.sol

SHA3: c237b7db89102d3720c374d71e2373b9f00e20fa33708b4f04a018cb83f85227

### Second review scope

#### Repository:

<https://github.com/Powerbomb-Finance/powerbomb-foundry/tree/univ3>

#### Commit:

9698e633d11e0549de41630e2f79e8b4badb2819

#### Technical Documentation:

Type: ReadMe

[Link](#)

#### Integration and Unit Tests: Yes

##### Contracts:

File: ./src/PbUniV3.sol

SHA3: 8e56c4fc5aa43a4794db659bdc890abc6bfbd081753e6913d3af0c7ff3b96f12

File: ./src/PbUniV3Proxy.sol

SHA3: 42717dafa30672ba6625e90448ee813209258b535ab7173134d0cb16a4d76f15

File: ./src/PbUniV3Reward.sol

SHA3: c237b7db89102d3720c374d71e2373b9f00e20fa33708b4f04a018cb83f85227

## Severity Definitions

Risk Level	Description
<b>Critical</b>	Critical vulnerabilities are usually straightforward to exploit and can lead to assets loss or data manipulations.
<b>High</b>	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.
<b>Medium</b>	Medium-level vulnerabilities are important to fix; however, they cannot lead to assets loss or data manipulations.
<b>Low</b>	Low-level vulnerabilities are mostly related to outdated, unused, etc. code snippets that cannot have a significant impact on execution.

## Executive Summary

The score measurement details can be found in the corresponding section of the [methodology](#).

### Documentation quality

The total Documentation Quality score is **8** out of **10**. Functional and technical descriptions are provided in GitHub Readme. NatSpec comments are superficial.

### Code quality

The total CodeQuality score is **6** out of **10**. Code follows the Style guide. Tests were provided but did not run.

### Architecture quality

The architecture quality score is **9** out of **10**. The contracts do not implement interfaces.

### Security score

As a result of the audit, the code contains **no** issues. 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: **9.3**.



## 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	<a href="#">SWC-100</a> <a href="#">SWC-108</a>	Functions and state variables visibility should be set explicitly. Visibility levels should be specified consciously.	Passed
Integer Overflow and Underflow	<a href="#">SWC-101</a>	If unchecked math is used, all math operations should be safe from overflows and underflows.	Passed
Outdated Compiler Version	<a href="#">SWC-102</a>	It is recommended to use a recent version of the Solidity compiler.	Passed
Floating Pragma	<a href="#">SWC-103</a>	Contracts should be deployed with the same compiler version and flags that they have been tested thoroughly.	Passed
Unchecked Call Return Value	<a href="#">SWC-104</a>	The return value of a message call should be checked.	Passed
Access Control & Authorization	<a href="#">CWE-284</a>	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	<a href="#">SWC-106</a>	The contract should not be self-destructible while it has funds belonging to users.	Passed
Check-Effect-Interaction	<a href="#">SWC-107</a>	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
Assert Violation	<a href="#">SWC-110</a>	Properly functioning code should never reach a failing assert statement.	Passed
Deprecated Solidity Functions	<a href="#">SWC-111</a>	Deprecated built-in functions should never be used.	Passed
Delegatecall to Untrusted Callee	<a href="#">SWC-112</a>	Delegatecalls should only be allowed to trusted addresses.	Passed
DoS (Denial of Service)	<a href="#">SWC-113</a> <a href="#">SWC-128</a>	Execution of the code should never be blocked by a specific contract state unless it is required.	Passed
Race Conditions	<a href="#">SWC-114</a>	Race Conditions and Transactions Order Dependency should not be possible.	Passed
Authorization	<a href="#">SWC-115</a>	tx.origin should not be used for	Passed

through tx.origin		authorization.	
Block values as a proxy for time	<a href="#">SWC-116</a>	Block numbers should not be used for time calculations.	Passed
Signature Unique Id	<a href="#">SWC-117</a> <a href="#">SWC-121</a> <a href="#">SWC-122</a> <a href="#">EIP-155</a>	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifier should always be used. All parameters from the signature should be used in signer recovery	Passed
Shadowing State Variable	<a href="#">SWC-119</a>	State variables should not be shadowed.	Passed
Weak Sources of Randomness	<a href="#">SWC-120</a>	Random values should never be generated from Chain Attributes or be predictable.	Passed
Incorrect Inheritance Order	<a href="#">SWC-125</a>	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	<a href="#">EEA-Leve1-2</a> <a href="#">SWC-126</a>	All external calls should be performed only to trusted addresses.	Passed
Presence of unused variables	<a href="#">SWC-131</a>	The code should not contain unused variables if this is not <a href="#">justified</a> by design.	Passed
EIP standards violation	<a href="#">EIP</a>	EIP standards should not be violated.	Passed
Assets integrity	Custom	Funds are protected and cannot be withdrawn without proper permissions.	Not Relevant
User Balances manipulation	Custom	Contract owners or any other third party should not be able to access funds belonging to users.	Not Relevant
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.	Passed
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	Passed



		data stored on the contract. There should not be any cases when execution fails due to the block Gas limit.	
<b>Style guide violation</b>	<b>Custom</b>	Style guides and best practices should be followed.	<b>Failed</b>
<b>Requirements Compliance</b>	<b>Custom</b>	The code should be compliant with the requirements provided by the Customer.	<b>Passed</b>
<b>Environment Consistency</b>	<b>Custom</b>	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	<b>Failed</b>
<b>Secure Oracles Usage</b>	<b>Custom</b>	The code should have the ability to pause specific data feeds that it relies on. This should be done to protect a contract from compromised oracles.	Not Relevant
<b>Tests Coverage</b>	<b>Custom</b>	The code should be covered with unit tests. Test coverage should be 100%, with both negative and positive cases covered. Usage of contracts by multiple users should be tested.	<b>Failed</b>
<b>Stable Imports</b>	<b>Custom</b>	The code should not reference draft contracts, that may be changed in the future.	<b>Passed</b>

## System Overview

Powerbomb finance is a crypto swap system with rewards accumulation. It consists of the contracts:

- PbUniV3.sol - handles interaction with Uniswap V3
- PbUniV3Proxy.sol - proxy contract
- PbUniV3Reward.sol - manages distribution of rewards for users

## Privileged roles

Roles defined in the system, e.g.:

- Owner: can set the Vault SC, yield fee percent, treasury for Reward SC, pause or unpause Vault SC, set Reward SC, bot.
- Vault: can record deposits, withdrawals, and harvest tokens.
- Bot: can reinvest and update price range of positions.

## Findings

### ■■■■ Critical

No critical severity issues were found.

### ■■■ High

**Owner can change important contract addresses.**

The functionality allows the owner to change addresses of vault, treasury, bot, and reward contracts.

This can lead to a loss of users' funds.

**File:** PbUniV3Reward.sol, PbUniV3.sol

**Contract:** PbUniV3Reward, PbUniV3

**Functions:** setVault, setTreasury, setBot, setReward

**Recommendation:** Add highly permissive functionality to the documentation.

**Status:** Fixed (revised commit: 9698e63)

### ■■ Medium

No medium severity issues were found.

### ■ Low

**Insufficient check for *tokenIn*.**

The check assumes that *tokenIn* is always one of the tokens from the Uniswap pool but does not consider the possibility of an invalid token address.

**File:** PbUniV3.sol

**Contract:** PbUniV3

**Functions:** reinvest

**Recommendation:** Check if *tokenIn* is *token0* or *token1*

**Status:** Fixed (revised commit: 9698e63)

## 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 report contains no statements or warranties on the identification of all vulnerabilities and security of the code. The report covers the code submitted to and reviewed, so it may not be relevant after any modifications. Do not consider this report as a final and 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.

English is the original language of the report. The Consultant is not responsible for the correctness of the translated versions.

### 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, Consultant cannot guarantee the explicit security of the audited smart contracts.