SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: PAID Network
Date: July 6th, 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 PAID Network
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	Launchpad
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	PAID Network
Timeline	25.05.2022 - 06.07.2022
Changelog	31.05.2022 - Initial Review 06.07.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by PAID Network (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/PAIDNetwork/launchpad-sc-multipool/commit/04d60aafcb86857aecd817a67b87ef3bb92620be

Commit:
04d60aafcb86857aecd817a67b87ef3bb92620be

Technical Documentation: No

Integration and Unit Tests: Yes (in “test” directory)

Contracts:

```
File: ./contracts/IgnitionAccess.sol
SHA3: 3de8e3abf85cf06370bdcbcc7a99b4aacc498a5856c09c72dcfa6fd24b3e5519

File: ./contracts/IgnitionCore.sol
SHA3: 5d5f3e111f707fe93d74557eed67a116b78ce8353dda49a14133fa2625fd4240

File: ./contracts/IgnitionEvents.sol
SHA3: 76c40e57258a1f1802c8f2dc9ec42d80be7c0e51f9c6ca8f8e2af41f2025a68d

File: ./contracts/IgnitionIDO.sol
SHA3: d7d96c2fee90d14549362b037ca98516d4f92358378617eb1a893a832f09b522

File: ./contracts/IgnitionPools.sol
SHA3: 0bd77b0ce3e0e57a691e5ce023a3784d2e6508d614a42b3dec44b3b7e032f202

File: ./contracts/IgnitionTransfers.sol
SHA3: 7cb9ed3c7bff4434d03e41adf54a7fd97da02e91e175e1e5a790845602bf5c19

File: ./lib/BitOpMapPool.sol
SHA3: 39c9ec2c51e0f957dfd96129d4e72ca38a33d0ef21a2a2dd4e87ea21db80479b

File: ./lib/Data.sol
SHA3: 35de6759fb3462da3eff6db51dab3474b5d272abda8f9c9d69d102d2750dc288

File: ./lib/LibPool.sol
SHA3: 020ad70ff2b8095e65c292684bddd3c9a5193633a080aacd8ea047fa2f2ad3f2

File: ./lib/Math.sol
SHA3: 931e6559ed50e2b95b3a51586bb7edd7a15d04d57d6b720200afecffdfb46dba
```

Second review scope

Repository:
https://github.com/PAIDNetwork/launchpad-sc-multipool/commit/60d0d5c1e755edc4c8c63b81aa058201babbfc63

Commit:
60d0d5c1e755edc4c8c63b81aa058201babbfc63

Technical Documentation: Yes
Type: Repository Readme
Link

Integration and Unit Tests: Yes (in “test” directory)
Contracts:

```
File: ./contracts/ERC20Token.sol
SHA3: f55053ce1d6812954394502a1b1a206e9efbf96985b9f896098ac76b3817739e

File: ./contracts/IgnitionAccess.sol
SHA3: 13b564d0f13485bd58b586ca1e8db0427556730d7ba9d59967d62c4670e067b2

File: ./contracts/IgnitionCore.sol
SHA3: 6555025cb4750d51ec4e933ea4d9f3fa3f34ef46c36e75529ae36722211c2b15

File: ./contracts/IgnitionIDO.sol
SHA3: 380941087ff75cebb75bffd31f8487fee072d3247177083698ab858df40be64a

File: ./contracts/IgnitionPools.sol
SHA3: 3ea89dec59298c5350ed6175963ab1989eaac7ffdd9ca122d660afdf53f61477

File: ./contracts/IgnitionTransfers.sol
SHA3: 13f4be674ea61a607cb82de7084549726646a9ec2831dbf0a0acdace9d39850f

File: ./contracts/PplToken.sol
SHA3: 440e60f45b6a6ee96a3fab40ad71c5079e45caad1fa2388c0037fa4295f8eee0

File: ./contracts/SampleToken.sol
SHA3: a729f0c5ab2caa400f7fd1975ae2683d0fc14f570378a439ebd06764983fe23d

File: ./contracts/SndToken.sol
SHA3: bc07f36f2dba6b189ee45bfb0ea2c6a52051e6bd0c4693ebf2107fc368ef2e94
```

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 measurement details can be found in the corresponding section of the methodology.

Documentation quality

The total Documentation Quality score is **10** out of **10**. Functional and technical description provided in repository Readme. Code is followed by NatSpec comments.

Code quality

The total Code Quality score is **8** out of **10**. Deployment and basic user interactions are covered with tests not available for running due to dependency errors.

Architecture quality

The architecture quality score is **4** out of **10**. Smart contracts fail to follow single responsibility principle, therefore creating deep multi-level inheritance. It is advised to separate the code into different files according to logical concern; to use *abstract* keyword for parental smart contracts. Several template code patterns were found.

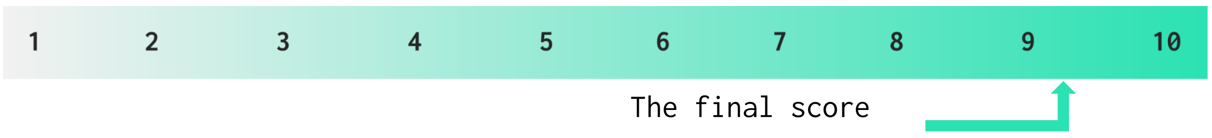
Security score

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: **9.2**



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 self-destructible while it has funds belonging to users.	Passed
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.	Passed
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	SWC-114	Race Conditions and Transactions Order	Passed

Conditions		Dependency should not be possible.	
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.	Passed
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 or be predictable.	Passed
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.	Passed
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	Passed

		should not be any cases when execution fails due to the block Gas limit.	
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.	Not Relevant
Environment Consistency	**Custom**	The project should contain a configured development environment with a comprehensive description of how to compile, build and deploy the code.	Failed
Tests Coverage	**Custom**	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.	Passed
Stable Imports	**Custom**	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

PAID Network is a launchpad contract system with the following contracts:

- *IgnitionAccess* – a contract with modifiers that control smart contract roles and managers; checks for user whitelist.
- *IgnitionCore* - a core contract that stores information about IDOs and pools; has ownership and pausing functionality.
- *IgnitionEvents* - a contract that declares all events used by the core contract.
- *IgnitionIDO* - IDO pool of the launchpad.
- *IgnitionPool* - the pool of project's tokens.
- *IgnitionTransfers* - a contract with the functionality of buying, redeeming, and withdrawing tokens.

Privileged roles

- The owner and the admin can finalize the pool, set whitelist, revert finalizing, add tokens to pool, set rate in pool, base tier, start and end date. They can pause and unpause the pool, set the total amount of tokens, transfer pool, set private pool, quote assets, and disable the pool.
- The owner and rbac manager can get the address of the project and set the admin wallet of it.

Findings

🟥🟥🟥🟥 Critical

No critical severity issues were found.

🟨🟨🟨 High

Highly permissive role

The owner can disable the pool, set ratez and base tier, change the pool's start and end date, pause and unpause token buying functions.

This can lead to token manipulation.

Contracts: IgnitionCore, IgnitionPools, IgnitionTransfers

Functions: pause, pausePool, buyTokensETH, buyTokensQuoteAsset, redeemTokens, disablePool, setStartDate, setEndDate, setRate, setBaseTier

Recommendation: Add highly permissive functionality to the documentation.

Status: Fixed (revised commit: 60d0d5c)

🟩🟩 Medium

1. **Code documentation contradiction**

   Some statements in the code documentation are opposite to code functionality. Documentation of the *IgnitionTransfers* contract declares the contract as the events contract, but the actual events contract is the *IgnitionEvents*.

   This makes code hard to read and evaluate

   Contract: -

   Functions: -

   Recommendation: Refactor the code project documentation and add the code requirements.

   Status: Fixed (revised commit: 60d0d5c)

2. **Check-effect interaction pattern violation**

   It is required to follow the pattern even when calling to the trusted contract.

   Contract: IgnitionTransfers

   Functions: redeemTokens

   Recommendation: Implement the Check-Effect Interaction pattern.

   Status: Fixed (revised commit: 60d0d5c)

3. Unchecked call return value

The return value of a *transfer* function call is not checked. Execution will resume even if the called contract throws an exception.

This may result in a loss of funds.

Contract: IgnitionTransfers

Functions: withdrawUnsoldTokens

Recommendation: Use *safeTransfer* method instead of *transfer*

Status: Fixed (revised commit: 60d0d5c)

■ Low

1. Unused variable

Event *LogAdminOnPoolToken* is declared and never used.

Contract: IgnitionEvents

Functions: -

Recommendation: Remove unused field.

Status: Fixed (revised commit: 60d0d5c)

2. Functions can be declared as external

Public functions that are never called in the contract should be marked *external*.

This increases transaction Gas.

Contracts: IgnitionCore, IgnitionPools

Functions: initialize, mintToWallet, calculateAmount, setEndDate

Recommendation: Declare mentioned functions as external.

Status: Fixed (revised commit: 60d0d5c)

3. State variable default visibility

The explicit visibility makes it easier to catch incorrect assumptions about who can access the variable.

Contracts: IgnitionAccess, IgnitionCore, IgnitionTransfers

Variables: merkleRoots

Recommendation: Specify variables as *public*, *internal,* or *private*. Explicitly define visibility for all state variables.

Status: Fixed (revised commit: 60d0d5c)

4. Redundant casting

Unnecessary *uint8* casting makes code harder to read and consumes more Gas.

Contracts: IgnitionIDO

Functions: finalize, revert_finalize

Recommendation: Remove redundant casting.

Status: Fixed (revised commit: 60d0d5c)

5. Incorrect data location of function arguments

Functions use *memory* data location for saving function arguments.

This increases transaction Gas.

Contracts: IgnitionIDO, IgnitionPools, LibPool

Functions: setWhiteList, addTokenToPool, generatePackage

Recommendation: Use *calldata* as data location because it will avoid copies and ensures that the data cannot be modified.

Status: Fixed (revised commit: 60d0d5c)

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.