

HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Apymon

Date: November 28th, 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

Name	Smart Contract Code Review and Security Analysis Report for Apymon
Approved By	Evgeniy Bezuglyi SC Audits Department Head at Hacken OU
Type	Vault
Platform	EVM
Language	Solidity
Methodology	Link
Website	
Changelog	15.11.2022 - Initial Review 28.11.2022 - Second Review



Table of contents

Introduction	4
Scope	4
Severity Definitions	6
Executive Summary	7
Checked Items	8
System Overview	11
Findings	12
Disclaimers	13

Introduction

Hacken OÜ (Consultant) was contracted by Apymon (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/apymon-team/Apymon-Vault-Contracts
Commit	5987091a3b91d9c0c8572957cad725a1fbf8645c
Functional Requirements	Attached for audit
Contracts	<p>File: ./contracts/IPunks.sol SHA3: 193be507d14453e9538c0dd6730e799f319787bfb282193b433887b16c165340</p> <p>File: ./contracts/IVault.sol SHA3: cac2550cf69b4f81d6cb718833d0b60357fb4e62efc1dcdd8c1ca9a2adcbd6d8</p> <p>File: ./contracts/IVaultFactory.sol SHA3: ed02cef3ccf7d531dbc4acd8ae8695d4bf2aabe629c7c362651391c48e9fa63c</p> <p>File: ./contracts/tokens/CryptoPunksToken.sol SHA3: 4e23eb26c46ffefaa116fe7242b245f2aaceb0213e3477157b069a9f575408a4</p> <p>File: ./contracts/tokens/ERC1155Token.sol SHA3: c20303a3b0d1804156958d2516cbabb6beeb8a00ed1dc43f56dc63120ab51481</p> <p>File: ./contracts/tokens/ERC20Token.sol SHA3: c647834c811c40c8edb6a66345b32284e2283656de107225fdf3432a7d5eaced</p> <p>File: ./contracts/tokens/ERC721Token.sol SHA3: a33b1c4df651ccd1a394c094becf88c554520e0450803b6b44898de100c8c470</p> <p>File: ./contracts/Vault.sol SHA3: 92fcfb96cb17dfd75e0b71ca0d418cda3105a7e7ba15c82ae93095dece59e783</p> <p>File: ./contracts/VaultFactory.sol SHA3: 6ad2db1967d72df11c5df6becfedfe5db501c49a3f7e7a4c3c496cc1f4c6737d</p>

Second review scope

Repository	https://github.com/apymon-team/Apymon-Vault-Contracts
Commit	59424c168a4c4cc9b1936cb66d8b1ddafe20ffca
Functional Requirements	Attached for audit
Contracts	<p>File: ./contracts/IPunks.sol SHA3: 193be507d14453e9538c0dd6730e799f319787bfb282193b433887b16c165340</p> <p>File: ./contracts/IVault.sol SHA3: fcffc76acfa3e54c7887572afd0296442d727f2fd39ec7cb9e0594da45a22a4a</p> <p>File: ./contracts/IVaultFactory.sol</p>



	<p>SHA3: ed02cef3ccf7d531dbc4acd8ae8695d4bf2aabe629c7c362651391c48e9fa63c</p> <p>File: ./contracts/Vault.sol</p> <p>SHA3: 20c9bb606db8c2cb631ddba31409073fab457dac9ba6c61d91c1850399a4196e</p> <p>File: ./contracts/VaultFactory.sol</p> <p>SHA3: 1903c295b266ff0dfb30deb2f9ecc6fac01ff583b08e59b6aa3e5f151833ad5c</p>
--	--

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 [scoring methodology](#).

Documentation quality

The total Documentation Quality score is **7** out of **10**.

- Functional requirements are partially missed.
- NatSpec comments are partially missed.

Code quality

The total Code Quality score is **7** out of **10**.

- The development environment is configured.
- Code violates Style guide.

Test coverage

Test coverage of the project is **96.67%** (branch coverage).

- Deployment and basic user interactions are covered with tests.
- Negative cases coverage is missed.

Security score

As a result of the audit, the code contains **1** medium and **1** low severity issues. The security score is **9** 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: **8.3**.

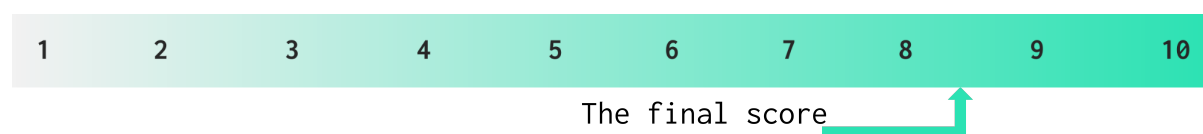


Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
14 November 2022	5	4	4	0
28 November 2022	1	1	0	0

Checked Items

We have audited the Customers' smart contracts for commonly known and more specific vulnerabilities. Here are some items 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.	Failed
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.	Not Relevant
Check-Effect-Interaction	SWC-107	Check-Effect-Interaction pattern should be followed if the code performs ANY external call.	Passed
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 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 EIP-155	Signed messages should always have a unique id. A transaction hash should not be used as a unique id. Chain identifiers should always be used. All parameters from the signature should be used in signer recovery	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 or be predictable.	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.	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.	Passed

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.	Failed
Style guide violation	Custom	Style guides and best practices should be followed.	Failed
Requirements Compliance	Custom	The code should be compliant with the requirements provided by the Customer.	Passed
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
Secure Oracles Usage	Custom	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
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.	Failed
Stable Imports	Custom	The code should not reference draft contracts, which may be changed in the future.	Passed

System Overview

Apymon is a mixed-purpose system with the following contracts:

- IPunks.sol - Interface for punks contract.
- IVault.sol - Interface for vault contract.
- IVaultFactory - Interface for vault factory.
- Vault.sol - Vault that uses NFT ownership as a key.
- VaultFactory.sol - Factory to clone vault contracts.

Privileged roles

- The owner of the Vault key has access to vault functionality.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

1. Unoptimized Loops Usage

Contracts use loops without optimization. Optimizing loops will lower Gas taxes.

Path: ./contracts/Vault.sol : withdrawMultiple()

Recommendation: Cache arrays in a loop, save state variables to local memory, iterate the loop and save changes to the state after the loop finishes.

Status: Reported

■ Low

1. Floating Pragma

Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, an outdated compiler version that might introduce bugs that affect the contract system negatively.

Paths: ./contracts/IPunks.sol, ./contracts/IVault.sol,
./contracts/IVaultFactory.sol

Recommendation: Consider locking the pragma version whenever possible and avoid using a floating pragma in the final deployment.

Status: Reported

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.