SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Vault Hill
Date: Sep 22th, 2022

Document

Name	Smart Contract Code Review and Security Analysis Report for Vault Hill
Approved By	Evgeniy Bezuglyi | SC Audits Department Head at Hacken OU
Type	ERC1155 with ERC2981 token
Platform	EVM
Network	Ethereum, Polygon
Language	Solidity
Methods	Manual Review, Automated Review, Architecture Review
Website	https://www.vaulthill.io
Timeline	29.08.2022 - 22.09.2022
Changelog	31.08.2022 - Initial Review 22.09.2022 - Second Review

Table of contents

Introduction

Hacken OÜ (Consultant) was contracted by Vault Hill (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/Vault-Hill/VHC1155
Commit:
cc44701a9179cb9480a18b17aa93d9c4552700c3
Documentation:
Whitepaper

Technical description

Integration and Unit Tests: Yes
Contracts:

```
File: ./interfaces/IERC2981Royalties.sol
SHA3: 81145063c657fbad24c7d2ebb1f306ee7a3d4897ca2f570d0fb64e40c13b4fb5

File: ./contracts/ERC2981Base.sol
SHA3: 3ff14b7d7a14f1e76d7cba04e8c4837e54b59e849a5d46e17d4f35ab16c18a81

File: ./contracts/ExposedVHC1155.sol
SHA3: c202135359d95743591e04ecd8887fde388bb5bd3a7d79a9c9c8f83581e8fc24

File: ./contracts/ERC2981PerTokenRoyalties.sol
SHA3: 20e3c9cb44e17a31b95484a13962a5b07325cd1aa9f1bf46c5453bdf60288765

File: ./contracts/VHC1155.sol
SHA3: 6d0e68c211f01a02d6f67647c09f118acbb820f525fb9800523fc000c2d95e12
```

Second review scope

Repository:
https://github.com/Vault-Hill/VHC1155
Commit:
f0da520a32e1c5ba581cf1f547028802617a1840
Documentation:
Whitepaper

Technical description

Integration and Unit Tests: Yes
Contracts:

```
File: ./interfaces/IERC2981Royalties.sol
SHA3: 2af768987dab411e82d4971ab66920104b1427741578f2bfcc95d6868b9ef82a

File: ./contracts/ERC2981Base.sol
SHA3: ec87f628c72e54cb2fcf3851861a17f0efee5560cf64712142fb9cbbdf2bd28c

File: ./contracts/ExposedVHC1155.sol
```

```
        SHA3: 41bcb03b7dc19a5b9d8e61bb87a520aa6cd6d3c3a7bd4950a239de3b6916d4ec

        File: ./contracts/ERC2981PerTokenRoyalties.sol
        SHA3: d079e3ba45c593ca104ecdcd398818ea55b4c960396276476d06b54c12f12e54

        File: ./contracts/VHC1155.sol
        SHA3: 88c70bd2747dde1a8b0dccf7d2f2b80dae61415724f54d8f34862cb12c800066
```

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 requirements are provided and cover the scope of the audit.

Code quality

The total CodeQuality score is **10** out of **10**. Deployment and user interactions are covered with tests. **Test coverage is 96.43%.**

Architecture quality

The architecture quality score is **8** out of **10**. Code follows the single responsibility principle, well-formatted and organized. It is recommended to use standardized interfaced from OpenZeppelin instead of copy-pasting them into the code.

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: **9.8**.

1
2
3
4
5
6
7
8
9
10
The final score

Table. The distribution of issues during the audit

Review date	Low	Medium	High	Critical
30 August 2022	4	1	0	0
21 September 2022	0	0	0	0

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.	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.	Not Relevant
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	SWC-115	tx.origin should not be used for	Not Relevant

through tx.origin		authorization.	
Block values as a proxy for time	SWC-116	Block numbers should not be used for time calculations.	Not Relevant
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 identifier 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.	Not Relevant
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	Passed

		data stored on the contract. There 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.	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.	Passed
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.	Passed
Stable Imports	Custom	The code should not reference draft contracts, that may be changed in the future.	Passed

System Overview

Vault Hill aims to create a metaverse called Vault Hill City (VHC) - a collection of virtual shared spaces, including the sum of all virtual worlds and the Internet. In VHC, users can interact with computer-generated imagery (CGI) and other users. It consists of the following contracts:

- *VHC1155* – the contract that implements ERC1155 and ERC2981 standards. It allows to mint tokens by users without limits.
- *ERC2981Base* – the contract used to add ERC2981 support to ERC721 and 1155.
- *ExposedVHC1155* - the contract that extends *VHC1155* and allows to retrieve token owner by index.
- *ERC2981PerTokenRoyalties* - the contract used to add ERC2981 support to ERC721 and 1155 allowing for royalties to be set on a per token ID basis.

Privileged roles

- The owner of the *VHC1155* contract can change token URI.

Risks

- No risks were found.

Findings

■■■■ Critical

No critical severity issues were found.

■■■ High

No high severity issues were found.

■■ Medium

1. **Invalid calculations**

   Solidity round division result towards zero, so *royaltyInfo* function could return unexpected values.

   Path: ./interfaces/ERC2981PerTokenRoyalties.sol: royaltyInfo()

   Recommendation: Check if the division logic is correct.

   Status: Mitigated

■ Low

1. **Floating Pragma**

   Contracts should be deployed with the same compiler version and flags that have been tested thoroughly. 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/ERC2981Base.sol, ./interfaces/IERC2981Royalties.sol, ./contracts/ExposedVHC1155.sol, ./contracts/ERC2981PerTokenRoyalties.sol, ./contracts/VHC1155.sol

   Recommendation: Use a fixed version of the compiler (^ symbol should be removed from Pragma).

   Status: Fixed (f0da520a32e1c5ba581cf1f547028802617a1840)

2. **The public function could be declared external**

   Public functions that are never called by the contract should be declared external to save Gas.

   Paths: ./contracts/ExposedVHC1155.sol : getLatestTokenOwner(),

   ./contracts/VHC1155.sol : setURI(), nextTokenId(),

   Recommendation: Use the external attribute for functions never called from the contract.

   Status: Fixed (f0da520a32e1c5ba581cf1f547028802617a1840)

3. **Duplicate code**

updateTokenRoyalty function implements the same code as *_setTokenRoyalty* function. Duplication of code may lead to unnecessary Gas consumption.

Path: ./contracts/ERC2981PerTokenRoyalties.sol: updateTokenRoyalty()

Recommendation: Reuse *_setTokenRoyalty* function.

Status: Fixed (f0da520a32e1c5ba581cf1f547028802617a1840)

4. **Missing validation**

mintBatch does not validate *amounts* size, so it is possible to pass amounts array with a length that differs from other parameters.

Path: ./contracts/VHC1155.sol: mintBatch()

Recommendation: Validate *amounts* size.

Status: Fixed (f0da520a32e1c5ba581cf1f547028802617a1840)

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.