HACKEN

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

Customer: Alt Platform
Date: June 28th, 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 Alt Platform
Approved By	Evgeniy Bezuglyi | SC Department Head at Hacken OU
Type	ERC-1155 system
Platform	EVM
Language	Solidity
Methods	Architecture Review, Functional Testing, Computer-Aided Verification, Manual Review
Website	https://www.onlyalt.com/
Timeline	19.05.2022 - 28.06.2022
Changelog	25.05.2022 - Initial Review 07.06.2022 - Second Review 21.06.2022 - Third Review 28.06.2022 - Fourth Review

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

Table of contents

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

Introduction

Hacken OÜ (Consultant) was contracted by Alt Platform (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/onlyalt/alt-smart-contracts
Commit:
3e49353f8f866f706b0adaf8655fd1da9c034670
Technical Documentation:
Type: Functional and technical description
Link: https://github.com/onlyalt/alt-smart-contracts/blob/main/README.md
JS tests: Yes
Contracts:
File: ./contracts/burners/AltBurner.sol
SHA3: b6a1e070ad5e44033b20c42d12122a9a947220aaed56415cd3268f9219d2fac0

File: ./contracts/interfaces/IERC2981.sol
SHA3: bd23b66af4d739d9be43d4b9a7c7bd3e1ec7294422ea20faff97d33d68109588

File: ./contracts/interfaces/IRoyalty.sol
SHA3: 5dd03bd484d074e82dd8b42a0de5060696ff2f10ca1219fb916d42d56e61b5e9

File: ./contracts/interfaces/ISupply.sol
SHA3: 362c9a1c36c97c1b13f67fe72315372fdc99486ada3aa272cc1c0538a604f388

File: ./contracts/interfaces/IVault.sol
SHA3: 2f615d800eec417b966b7527b8eba6542d77aaa2bbdaf8e621877bb4889524eb

File: ./contracts/minters/pack-drop/AltRepack.sol
SHA3: 8c88cad7f979a46787812072201fa72280b0e64f9a80f6bfbe9e8024531247c4

File: ./contracts/minters/PackDropBase.sol
SHA3: cd850b4ee47fa8f1982967d49ea261712e86e4e8e5cd2d38c8d2d7b05bf7b955

File: ./contracts/minters/TokenMintingPool.sol
SHA3: c04f1dc616158ad1377fb10caca6ba3a16a42cb03f3d4322a6bd81e97a4880d5

File: ./contracts/SignatureAccessControl.sol
SHA3: d9683ee47075076fdce956d67d1a692bad7fa7e2083816ea896e8126322bfa7e

File: ./contracts/test/contracts/AltRepackTest.sol
SHA3: 31870b18d320c4b9b0b8d5e7ee7b1cebd1c3414a5d5f753af8f0f8dc36069c8b

File: ./contracts/test/contracts/DelegateMinter.sol
SHA3: 71c18a3a03ee9442d96a470e92c8792ea947ab3da3dbb06784b160a295835522

File: ./contracts/test/contracts/LinkToken.sol
SHA3: 4c697667b5136470ec0356b9da311ee5ffa5e70c5fb267479cda1ebf7f937db4

File: ./contracts/test/contracts/MockVRFCoordinator.sol
SHA3: b9ea5a2d778528e415b81b2a2bc97cabaf63421efad775adb118b93289a79281

File: ./contracts/test/contracts/VaultTest.sol
SHA3: 74dd6b9bc38cddc5aa94e5c1b8f851e8cdec3207616481f87a27ef13f2097b5a

File: ./contracts/test/interfaces/IAltRepack.sol
SHA3: a75a68726ea7a6bf0bb0baa0156c0edb663d317006b03c28618bdc0d4172ca08

File: ./contracts/vault/ERC2981Base.sol
SHA3: 0ef9285c6756dc47cb2d6615e29e4997e25da289f49f05d6b173428a1578ee8a

File: ./contracts/vault/Roles.sol
SHA3: 8704d54bc32fe7a31c9542a0849c5ca08391f300127fb79e03fc346b753ced5f

File: ./contracts/vault/Royalty.sol
SHA3: 59e59d23483ddecff0f371ffdeaf7552cf9b5ba59587758c381beb548ff93f69

File: ./contracts/vault/Supply.sol
SHA3: 4b455889f9de6089aa11ef7d9abed1ef4f90d373876129c405c4ef84d2a2f960

File: ./contracts/vault/VaultUpgradeable.sol
SHA3: eb129982876066a5edd3ef4f39e2cc8361cb97f87130e93b95327d4923c16246

Second review scope

Repository:
https://github.com/onlyalt/alt-smart-contracts
Commit:
9a2d597deee3ea0c2dd004426871cd0112658b10
Technical Documentation:
Type: Functional and technical description
Link: https://github.com/onlyalt/alt-smart-contracts/blob/main/README.md
JS tests: Yes
Contracts:
File: ./contracts/interfaces/ISupply.sol
SHA3: f39d7c461534d3f8282903c58daa76ac9672a236c8b27e5834b699b830e1165b

File: ./contracts/interfaces/IVault.sol
SHA3: efe5043ac91757270f547da9465f6ec4d88d646950a9e5601663c1d05581f220

File: ./contracts/minters/pack-drop/AltRepack.sol
SHA3: c5782a036782cb967598937cd7b7358e8f72faeffe3ac513c628acc7bcfe55cb

File: ./contracts/minters/PackDropBase.sol
SHA3: 2b4c40706217bd4123df5ff2dfe1a90ddb54beb10243a7cab7e574f2d9c2a214

File: ./contracts/minters/TokenMintingPool.sol
SHA3: 07fe4600b8e24c881de778db85deb710532ef785b6388ccef84ffe2763347bc9

File: ./contracts/SignatureAccessControl.sol
SHA3: 7ef695a82034d327a3a35689657e3138b14b6d77e1d0a93ad60112026c704074

File: ./contracts/test/contracts/AltRepackTest.sol
SHA3: 407b58e1c6485aa2ef289530aab095a4ad3c9538b223b277774347548154c1ad

File: ./contracts/test/contracts/DelegateMinter.sol
SHA3: 6778a5f82833785478d53b19425b18907522215f2b235aee0840fe709f171ee2

File: ./contracts/test/contracts/MockVRFCoordinator.sol
SHA3: b8b694870dcaf4e695a5c6f778cf93cfc2a44ed9977766cd012ea344dc4948dd

File: ./contracts/test/contracts/VaultTest.sol

SHA3: 60f18b1b09be093d709614c7bf8824fba0278bad6d5763037c9a747c383d889a

File: ./contracts/test/interfaces/IAltRepack.sol
SHA3: a75a68726ea7a6bf0bb0baa0156c0edb663d317006b03c28618bdc0d4172ca08

File: ./contracts/vault/Roles.sol
SHA3: df87d8a9419719b35f9bb13c4111e805f4b9c0562bebc9be779c85fb5596b49d

File: ./contracts/vault/Supply.sol
SHA3: ae47377667266084d27d55b26eecb474c98256bf284d73a58874aaba333dbcc5

File: ./contracts/vault/VaultUpgradeable.sol
SHA3: 5dc9d7f79e9cc38cfaf3d2dfb33ec81fb46311b998dabb927fcb5eff874e3b53

Third review scope

Repository:
https://github.com/onlyalt/alt-smart-contracts
Commit:
6ec3da7cf88e4fcdef970847d843b6c09d35c488
Technical Documentation:
Type: Functional and technical description
Link: https://github.com/onlyalt/alt-smart-contracts/blob/main/README.md
JS tests: Yes
Contracts:
File: ./contracts/interfaces/ISupply.sol
SHA3: f39d7c461534d3f8282903c58daa76ac9672a236c8b27e5834b699b830e1165b

File: ./contracts/interfaces/IVault.sol
SHA3: efe5043ac91757270f547da9465f6ec4d88d646950a9e5601663c1d05581f220

File: ./contracts/minters/pack-drop/AltMint.sol
SHA3: 808947a7f3687ed5433304cc93cfe537fa876928a3c488abd437b77c0f013459

File: ./contracts/minters/PackDropBase.sol
SHA3: 4e6d6f960382a8a8fea94496a52a8ebb9a36f09e94bcf1cdd05b8de8c1a714b2

File: ./contracts/minters/TokenMintingPool.sol
SHA3: 55930b23778bde5955387fc3f927c9213b36b5eb45c226d78f9dc6ef95844b35

File: ./contracts/NoDelegateCall.sol
SHA3: 2858138165f431a7db70d4d022b69e9f7a70de3cafdb64eeabd06e561224ed83

File: ./contracts/proof_of_integrity/AltProofOfIntegrity.sol
SHA3: 2af7167b0bb12a5c07e67e124f021e2fd9e3a557eecd073d0b7f8d45e44c0650

File: ./contracts/SignatureAccessControl.sol
SHA3: 7ef695a82034d327a3a35689657e3138b14b6d77e1d0a93ad60112026c704074

File: ./contracts/SignatureAccessControlUpgradeable.sol
SHA3: 05d861445fdc80a66e3d782c2ee2e7602692d19582858b524853bada57d6ef66

File: ./contracts/test/contracts/AltMintTest.sol
SHA3: 748d3f739cef306ef84f9f3fc321785225f81bbd8aef0d433793b1fe064990fc

File: ./contracts/test/contracts/DelegateMinter.sol
SHA3: 6778a5f82833785478d53b19425b18907522215f2b235aee0840fe709f171ee2

File: ./contracts/test/contracts/MockVRFCoordinator.sol
SHA3: 793c4c5c30653235e7d5c2492fb47424ebef9d55d1840ac21264b091a193963e

File: ./contracts/test/contracts/Relayer.sol
SHA3: 5fa0a556d0862b6b6dcdbe68bac69ddabb5be8c90f73239cf16b955d5bc00090

File: ./contracts/test/contracts/VaultTest.sol
SHA3: 60f18b1b09be093d709614c7bf8824fba0278bad6d5763037c9a747c383d889a

File: ./contracts/test/interfaces/IAltRepack.sol
SHA3: a75a68726ea7a6bf0bb0baa0156c0edb663d317006b03c28618bdc0d4172ca08

File: ./contracts/vault/Roles.sol
SHA3: df87d8a9419719b35f9bb13c4111e805f4b9c0562bebc9be779c85fb5596b49d

File: ./contracts/vault/Supply.sol
SHA3: ae47377667266084d27d55b26eecb474c98256bf284d73a58874aaba333dbcc5

File: ./contracts/vault/VaultUpgradeable.sol
SHA3: 5dc9d7f79e9cc38cfaf3d2dfb33ec81fb46311b998dabb927fcb5eff874e3b53

Fourth review scope

Repository:
https://github.com/onlyalt/alt-smart-contracts
Commit:
192ea3f18681ece22265bebcf743fb46ac9b57e5
Technical Documentation:
Type: Functional and technical description
Link: https://github.com/onlyalt/alt-smart-contracts/blob/main/README.md
JS tests: Yes
Contracts:
File: ./contracts/interfaces/ISupply.sol
SHA3: f39d7c461534d3f8282903c58daa76ac9672a236c8b27e5834b699b830e1165b

File: ./contracts/interfaces/IVault.sol
SHA3: efe5043ac91757270f547da9465f6ec4d88d646950a9e5601663c1d05581f220

File: ./contracts/minters/pack-drop/AltMint.sol
SHA3: 808947a7f3687ed5433304cc93cfe537fa876928a3c488abd437b77c0f013459

File: ./contracts/minters/PackDropBase.sol
SHA3: 4e6d6f960382a8a8fea94496a52a8ebb9a36f09e94bcf1cdd05b8de8c1a714b2

File: ./contracts/minters/TokenMintingPool.sol
SHA3: 55930b23778bde5955387fc3f927c9213b36b5eb45c226d78f9dc6ef95844b35

File: ./contracts/NoDelegateCall.sol
SHA3: 2858138165f431a7db70d4d022b69e9f7a70de3cafdb64eeabd06e561224ed83

File: ./contracts/proof_of_integrity/AltProofOfIntegrity.sol
SHA3: b4b24acc112fd41296eff630bcaeb04ac25645c5e16f780f0d6dc47ee31a6552

File: ./contracts/SignatureAccessControl.sol
SHA3: 7ef695a82034d327a3a35689657e3138b14b6d77e1d0a93ad60112026c704074

File: ./contracts/SignatureAccessControlUpgradeable.sol
SHA3: 05d861445fdc80a66e3d782c2ee2e7602692d19582858b524853bada57d6ef66

File: ./contracts/test/contracts/AltMintTest.sol
SHA3: 748d3f739cef306ef84f9f3fc321785225f81bbd8aef0d433793b1fe064990fc

File: ./contracts/test/contracts/DelegateMinter.sol
SHA3: 6778a5f82833785478d53b19425b18907522215f2b235aee0840fe709f171ee2

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

```
File: ./contracts/test/contracts/MockVRFCoordinator.sol
SHA3: 793c4c5c30653235e7d5c2492fb47424ebef9d55d1840ac21264b091a193963e

File: ./contracts/test/contracts/Relayer.sol
SHA3: 5fa0a556d0862b6b6dcdbe68bac69ddabb5be8c90f73239cf16b955d5bc00090

File: ./contracts/test/contracts/VaultTest.sol
SHA3: 60f18b1b09be093d709614c7bf8824fba0278bad6d5763037c9a747c383d889a

File: ./contracts/test/interfaces/IAltRepack.sol
SHA3: 77b754c71a3ec102eff56b80c63d56c3dd806b04d1beb766031f50cb9e558d90

File: ./contracts/vault/Roles.sol
SHA3: df87d8a9419719b35f9bb13c4111e805f4b9c0562bebc9be779c85fb5596b49d

File: ./contracts/vault/Supply.sol
SHA3: ae47377667266084d27d55b26eecb474c98256bf284d73a58874aaba333dbcc5

File: ./contracts/vault/VaultUpgradeable.sol
SHA3: 5dc9d7f79e9cc38cfaf3d2dfb33ec81fb46311b998dabb927fcb5eff874e3b53
```

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

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.

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

Executive Summary

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

Documentation quality

The Customer provided functional requirements and technical requirements. The total Documentation Quality score is **10** out of **10**.

Code quality

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

Architecture quality

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

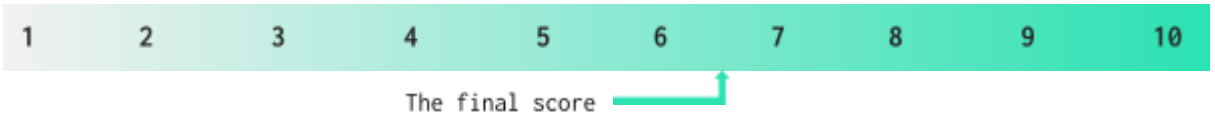
Security score

As a result of the audit, the code contains **1** high severity issue. The security score is **5** 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: **6.5**.



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

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.	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.	Failed
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.	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. 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

Alt NFT is an NFT token system with the following contracts:

- *VaultUpgradeable* — is a contract that stores ERC-1155 token. Allows to mint and burn tokens. Supply data and limits are described in the *Supply* contract. It has pausing functionality.
- *Supply* — is a contract that manages supply of *VaultUpgradeable*. Allows to retrieve supply information (if the token exists, current and maximal supply), set maximal supply for each token, and set the default supply.
- *Roles* — is a contract for roles and access functionality.
- *SignatureAccessControl* - is a contract for allowlist functionality. The contract checks callers` signatures.
- *SignatureAccessControlUpgradeable* - is an upgradable contract for signatures verifications.
- *TokenMintingPool* - is a contract for minting tokens from VaultUpgradeable. Uses Chainlink to get random value for choosing tokens to be minted. Allows to mint only after the random from Chainlink is retrieved.
- *PackDropBase* - is a contract for selling tokens from VaultUpgradeable using the TokenMintingPool. There are presale and public selling phases. Users should pay the defined price in native coins and be allowlisted to buy NFTs (up to the limit amount users can buy per one transaction and up to the limit per one user) in both phases.
- *AltMint* - is a *PackDropBase* repack on Rinkeby.
- *ISupply* - is an interface for the *Supply* contract.
- *IVault* - is an interface for the *VaultUpgradeable* contract.
- *AltMintTest* - is a testing *PackDropBase* repack on Rinkeby.
- *DelegateMinter* - is a testing contract implementing a delegate call to AltRepack.claimToken.
- *MockVRFCoordinator* - is a testing mock VRF Coordinator contract.
- *VaultTest* - is a testing contract for Vault with minting and burning functionality included.
- *IAltRepack* - is a testing interface for *AltRepack* used in *DelegateMinter.*
- *NoDelegateCall* - is a contract for preventing delegatecall to a contract.
- *AltProofOfIntegrity* - is a contract for generating and verifying proofs of integrity.
- *Relayer* - is a minimal forwarder contract meant for testing.

Privileged roles

- The “DEFAULT_ADMIN_ROLE” role in the *VaultUpgradeable* contract allows setting base URI value, pausing, and unpausing contract.
- The “MINTER_ROLE” role in the *VaultUpgradeable* contract allows token minting.

- The “DEFAULT_ADMIN_ROLE” role in the *Supply* contract allows setting maximal supplies for tokens and default maximal supply.
- The “DEFAULT_ADMIN_ROLE” role in the *Roles* contract allows granting and revoking roles, transferring the “DEFAULT_ADMIN_ROLE” to the other address.
- The “ESCROW_ROLE” role in the *PackDropBase* contract allows the withdrawal of native coins from the contract.
- The “MINTING_ADMIN_ROLE” role in the *PackDropBase* contract allows updating the minting price before the selling start, adding and removing tokens available for minting, and locking total supply (sending request for randomness to the Chainlink), and opening selling phases.
- The “DEFAULT_ADMIN_ROLE” role in the *VaultTest* contract allows setting base URI value, pausing, and unpausing contract.
- The “MINTER_ROLE” role in the *VaultTest* contract allows tokens minting.

Risks

- *AltMint, AltMintTest, DelegateMinter* contracts use deployed contract addresses that cannot be verified.
- The functionality allows the user with “DEFAULT_ADMIN_ROLE” to pause the *VaultUpgradeable* contract.

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

Findings

Critical

1. **Missing royalty payment and incorrect royalty receiver**

   The royalty is never paid. The “_beforeMint” function, which is marked as a royalty helper in the comments, does not transfer any funds to the royalty receiver.

   Therefore, the defined royalty receiver will never get the royalty.

   There is a defined royalty recipient for each token (RoyaltyInfo.recipient), but there is “royaltyReceiver” value in the Royalty contract.

   An incorrect royalty recipient address can be used (“royaltyReceiver” value instead of separately defined RoyaltyInfo.recipient for each token).

   Contracts: VaultUpgradeable.sol, Royalty.sol

   Recommendation: add royalty transferring and transfer royalty to the correct recipient.

   Functions: VaultUpgradeable._beforeMint

   Status: Fixed. Functionality removed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

2. **Insufficient signature verification**

   Used signatures are not stored and checked for uniqueness.

   This may lead to Signature Replay Attacks.

   The signatures are not separated by the functionality they give access to.

   Therefore, users who have access to some functionality using the signature will take access to all the protected functionality. For example, allowlisted for NFT selling (“PackDropBase” contract, “claimTokens'' function), users may burn (“AltBurner” contract, “burnTokensAllowList” function) users` tokens using the same signature.

   Contracts: SignatureAccessControl.sol

   Recommendation: store the used signatures and check the new ones for uniqueness. Separate the signatures by the functionality they give access to.

   Functions: _hasAccess

   Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

High

1. **Predictable randomness**

When minting tokens, the random index is calculated using the remaining supply count, a publicly accessible and random value from Chainlink that is the one per sale period.

Therefore, users can calculate the random value from Chainlink ("_randomSeed" variable) and call the minting at a profitable moment.

Contracts: TokenMintingPool.sol, PackDropBase.sol

Functions: _mintTokens

Recommendation: use a newly generated random value from Chainlink per each token minting.

Status: Partially fixed (Revised commit: 6ec3da7cf88e4fcdef970847d843b6c09d35c488). Random values("_randomSeeds" values) can be calculated and used for profit.

2. **Incorrect allowlist functionality**

According to the comments in the code, the presale should be available for users with presale access, and this access expires. However, there is the same access check for presale and public sale, and the access does not expire.

Therefore, not allowed users can have access to the presale, or there is a redundant check for public sale.

Contracts: PackDropBase.sol

Functions: claimTokens

Recommendation: add the separate access check for the presale or remove the check for public sale, according to the contracts` requirements.

Status: Mitigated. The Customer comment: "There is indeed no distinction between PreSale and Public Sale signature mechanism on-chain. Changes happen in our backend: during the presale, we have an allowlist (csv file) and only sign addresses in this allowlist ; after public release is triggered (pre defined date), we will compute the signature for any address without any check."

3. **Highly permissive burning functionality**

Admin and users from the allowlist can burn tokens from any account.

Users` tokens can be burnt without their allowance.

Contracts: AltBurner.sol, VaultUpgradeable.sol

Functions: AltBurner.burnTokens, AltBurner.burnTokensAllowList, VaultUpgradeable.burn, VaultUpgradeable.burnBatch

Recommendation: remove the possibility to burn users` tokens without their allowance.

Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

4. **Insufficient signature verification**

The signatures do not include the CHAIN_ID values.

This may lead to the Signature Replay Attacks when the same signature is used on different chains.

Contracts: SignatureAccessControl.sol

Recommendation: use the CHAIN_ID value in the signatures verifications. (EIP-155)

Functions: _hasAccess

Status: Mitigated. The Customer comment: “Using different signer addresses on different chains should allow us to avoid this replay attack vector”

Medium

1. **Not found artifact for tests**

   The code in Test_Vault.js contains getting and deploying the “Relayer” contract factory that cannot be found.

   Therefore, the tests cannot pass.

   Contract: -

   Functions: -

   Recommendation: remove getting and deploying the “Relayer” contract (27 and 29 lines).

   Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

2. **Possible minting limit per account exceeding**

   When processing minting tokens, there is a check if a user has not already reached the minting limit per wallet, and the newly claimed tokens are not considered in the calculation.

   Therefore, the minting limit per wallet may be exceeded.

   Contract: PackDropBase.sol

   Functions: _processMintRequest

   Recommendation: check if the sum of already minted tokens and newly claimed ones do not exceed the minting limit per wallet.

   Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

3. **Difference between testing and production contracts**

   The VaultTest contract, used for testing, differs from VaultUpgradeable contract. The testing contract includes the minting functionality not included in the production contract. The minting functionality in the test contract sets the royalty; the production version misses this step.

This may lead to erroneous testing of contracts.

Contract: VaultTest.sol

Functions: _beforeMint, mintPackDrop

Recommendation: ensure that testing and production versions of contracts have the same functionality.

Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

4. **Transfer can fail**

   The “transfer” function has a built-in Gas limit.

   Execution will fail if the receiver is a contract with fallback functionality.

   Contract: PackDropBase.sol

   Functions: withdraw

   Recommendation: ensure the receiver address is not a contract or transfer using “call”.

   Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

5. **Usage of hardcoded parameters**

   Hardcoded addresses are used in the constructor.

   Testing of the contract is complicated.

   Contract: AltRepack.sol

   Functions: constructor

   Recommendation: use constructor parameters instead of hardcoded values.

   Status: Mitigated. The Customer comment: “This is a design choice from our team to have hardcoded addresses in the constructor for ease of visibility in the smart contract itself.”

6. **Tests failing**

   16 tests are failing with the reason: “This address does not have access to the drop.”

   Contract: -

   Functions: -

   Recommendation: ensure that all tests pass.

   Status: Fixed (Revised commit: 192ea3f18681ece22265bebcf743fb46ac9b57e5)

Low

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

1. **Redundant modifier**

   Modifier “onlyPublicSale” is never used.

   Contract: PackDropBase.sol

   Functions: -

   Recommendation: remove the redundant modifier.

   Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

2. **Unlocked pragma**

   Contracts with unlocked pragmas may be deployed by the latest compiler, which may have higher risks of undiscovered bugs.

   Contracts should be deployed with the same compiler version they have been tested thoroughly.

   Contracts: AltBurner.sol, IERC2981.sol, IRoyalty.sol, ISupply.sol, IVault.sol, AltRepack.sol, PackDropBase.sol, TokenMintingPool.sol, AltRepackTest.sol, DelegateMinter.sol, LinkToken.sol, MockVRFCoordinator.sol, VaultTest.sol, IAltRepack.sol, ERC2981Base.sol, Roles.sol, Royalty.sol, Supply.sol, VaultUpgradeable.sol, SignatureAccessControl.sol, AltProofOfIntegrity.sol

   Function: -

   Recommendation: lock pragma to a specific compiler version.

   Status: Fixed (Revised commit: 192ea3f18681ece22265bebcf743fb46ac9b57e5)

3. **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: AltBurner.sol, PackDropBase.sol, Roles.sol, VaultUpgradeable.sol, VaultTest.sol

   Function: AltBurner.burnTokens, AltBurner.burnTokensAllowList, PackDropBase.updateMintPrice, PackDropBase.addTokens, PackDropBase.removeTokens, PackDropBase.selfCheckPresaleAccess, PackDropBase.lockTokenSupply, PackDropBase.openPresale, PackDropBase.openPublicSale, PackDropBase.saleStatus, PackDropBase.selfCheckTokensMinted, Roles.transferAdmin, Roles.grantMinterRole, Roles.grantBurnerRole, Roles.revokeMinterRole, Roles.revokeBurnerRole, VaultUpgradeable.uri, VaultTest.uri

   Recommendation: replace the visibilities to “external”.

   Status: Fixed (Revised commit: 6ec3da7cf88e4fcdef970847d843b6c09d35c488)

4. **No events on state variables changes**

There are no events emitted on important state variables changes.

Contracts: AltBurner.sol, PackDropBase.sol, TokenMintingPool.sol

Function: AltBurner.setBurningFee, PackDropBase.openPresale, PackDropBase.openPublicSale, PackDropBase.updateMintPrice, TokenMintingPool._addTokens, TokenMintingPool._removeTokens,

Recommendation: emit events on important state variables changings.

Status: Fixed (Revised commit: 9a2d597deee3ea0c2dd004426871cd0112658b10)

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

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.