KnownOrigin Audit

MintingFactoryV2, BaseUpgradableMarketplace & KODAV3UpgradableGatedMarketplace

April 2022

By CoinFabrik

Introduction

CoinFabrik was asked to audit the contracts for the KnownOrigin project. First we will provide a summary of our discoveries and then we will show the details of our findings.

Scope

The contracts audited are from the https://github.com/knownorigin/known-origin-contracts-v3.git git repository. The audit is based on the commit `d592c5f4fa4e0b6fc65a1fce43e302706aedf607`.

The audited files are:

- `/contracts/marketplace/KODAV3UpgradableGatedMarketplace.sol`: Contains the contract used to deploy gated marketplaces.
- `/contracts/marketplace/BaseUpgradableMarketplace.sol`: Contains the `BaseUpgradableMarketPlace` contract. This contract can be used to make upgradable marketplaces, including several utilities that simplify its creation.
- `/contracts/minter/MintingFactoryV2.sol`: Contains the `MintingFactoryV2` contract. This contract is an upgradeable glue contract that binds several contracts to facilitate the minting process. It is intended to replace the `MintingFactory` contract while allowing for gated marketplaces.
- `/contracts/access/IKOAccessControlsLookup.sol`: It contains the interface definition for the contract responsible for handling authorization of different roles in the system.
- `/contracts/core/IKODAV3.sol`: It contains the interface definition for the `KODAV3` token.
- `/contracts/marketplace/IKODAV3Marketplace.sol`: Contains interface definitions for different kinds of marketplace contracts.
- `/contracts/core/IKODAV3Minter.sol`: Contains the interface definition for the KODA Minter contract (version 3).
- `/contracts/collab/ICollabRoyaltiesRegistry.sol`: Contains the interface definition for the royalties registry.

The scope of the audit is limited to those files. No other files in this repository were audited. Its dependencies are assumed to work according to their documentation. Also, no tests were reviewed for this audit.
Fixes were checked on commit `3bcd94f66e5d0f6b38881fd52971c13dd08b6974`.

After that the development team fixed a bug regarding royalties distribution. We checked that fix on commit `9cd490474667bf021c7b0c1e8b3c697fc3072f3a` and no new security issues were found in the audited files.

Analyses

Without being limited to them, the audit process included the following analyses:

- Arithmetic errors
- Outdated version of Solidity compiler
- Race conditions
- Reentrancy attacks
- Misuse of block timestamps
- Denial of service attacks
- Excessive gas usage
- Missing or misused function qualifiers
- Needlessly complex code and contract interactions
- Poor or nonexistent error handling
- Insufficient validation of the input parameters
- Incorrect handling of cryptographic signatures
- Centralization and upgradeability

Summary of Findings

We found no critical or medium issues. Several minor issues were found. Also, several enhancements were proposed.

All security issues were either resolved, mitigated or acknowledged by the development team. Some enhancements were implemented by the development team.

Security Issues

ID	Title	Severity	Status
MI-01	Possible Reentrancy Issues	Minor	Acknowledged
MI-02	Outdated Solidity Version	Minor	Acknowledged
MI-03	Unchecked Transfer	Minor	Resolved

ID	Title	Severity	Status
MI-04	Possible DOS on KODAV3UpgradableGatedMarketplace.mint()	Minor	Mitigated

Privileged Roles

These are the privileged roles that we identified on each of the audited contracts.

MintingFactoryV2

Admin

An address with the admin role can:

- Upgrade the contract.
- Set and unset the frequency override for an address.
- Set the minting period.
- Set the royalties registry.
- Set the maximum mints in a period.

The `accessControls` contract controls if an address has the admin role via the `hasAdminRole()` function.

Verified Artist

An address with the verified artist role can mint batches for itself using the following functions:

- `mintBatchEdition()`
- `mintBatchEditionGatedOnly()`
- `mintBatchEditionGatedAndPublic()`
- `mintBatchEditionOnly()`

Minting is governed by the restrictions set by the admin.

The `accessControls` contract controls if an address has the verified artist role via the `isVerifiedArtist()` function. The required cryptographic proof is forwarded to this function.

Verified Artist Proxy

An address with the verified artist proxy role can mint batches on behalf of a creator it represents using the following functions:

- `mintBatchEditionAsProxy()`
- `mintBatchEditionGatedOnlyAsProxy()`
- `mintBatchEditionGatedAndPublicAsProxy()`
- `mintBatchEditionOnlyAsProxy()`

Minting is governed by the restrictions set by the admin.

The `accessControls` contract controls if an address is a verified artist proxy for a creator via the `isVerifiedArtistProxy()` function.

BaseUpgradableMarketplace

Admin

An address with the admin role can:

- Upgrade the contract.
- Change the `accessControls` contract used in the contract via the `updateAccessControls()` function. In order to do so, it needs to have the admin role in both the old and new `accessControls` contracts.
- Transfer away ERC20 tokens owned by the contract via the `recoverERC20()` function.
- Transfer ether away owned by the contract the `recoverStuckETH()` function.
- Update the platform commision via the `updatePlatformPrimaryComission()` function. This value is not used in this contract but may be used in a derived contract.
- Update the `modulo` value via the `updateModulo()` function. This value is only used in the private and non-invoked `_handleSaleFunds()` function and may also be used in a derived contract.
- Update the `minBidAmount` via the `updateMinBidAmount()` function. This value is not used in this contract but may be used in a derived contract.
- Update the `bidLockupPeriod` via the `updateBidLockupPeriod()` function. This value is only used in the private and non-invoked `_getLockupTime()` function and may also be used in a derived contract.

- Update the `platformAccount` value via the `updatePlatformAccount()` function. This value is only used in the private and non-invoked `_handleSaleFunds()` function and may also be used in a derived contract.
- Pause and resume the contract via the `pause()` and `unpause()` functions.

The `accessControls` contract controls if an address has the admin role via the `hasAdminRole()` function.

Other functions in derived contracts may be allowed to an admin only via the `onlyAdmin()` modifier and/or to the admin and other roles via the `onlyCreatorContractOrAdmin()` modifier.

Contract and Creator

While there are no functions marked with the `onlyCreatorContractOrAdmin()` modifier in this contract, this modifier can be used to restrict the execution of functions to either an admin, a contract or the issuer of the edition passed as a parameter. Checking that an address has a contract or admin role is made via the `accessControls.hasContractOrAdminRole()` function. To check if an address corresponds to an editionId the `koda.getCreatorOfEdition()` function is used.

KODAV3UpgradableGatedMarketplace

This contract inherits all the roles and capabilities defined in the `BaseUpgradableMarketplace` contract. No new roles are added, but the original roles get additional capabilities.

Admin

Besides the capabilities defined in `BaseUpgradableMarketplace` contract, an address with the admin role can:

- Change the `fundsReceiver` of a sale via the `updateFundsReceiver()` function.
- Change the `maxEditionId` of a sale via the `updateMaxEditionId()` function.
- Update the creator of a sale via the `updateCreator()` function.
- Override the commission for a sale via the `setKoCommissionOverrideForSale()` function.

Admin and Contract

An address with the admin and/or contract roles can:

- Create sales for any edition via the `createSale()` and `createSaleWithPhases()` functions.
- Create and remove phases of sales for any edition via the `createPhase()`, `createPhases()` and `removePhase()` functions.
- Pause or resume any sale via the `toggleSalePause()` function.

Creator

An address with the admin and/or contract roles can:

- Create sales for any edition whose `editionId` corresponds to them via the `createSale()` and `createSaleWithPhases()` functions.
- Create and remove phases of sales for any edition whose `editionId` corresponds to them via the `createPhase()`, `createPhases()` and `removePhase()` functions.
- Pause or resume any sale whose `editionId` corresponds to them via the `toggleSalePause()` function.

Security Issues Found

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. They must be fixed **immediately**.
- **Medium:** These are potentially exploitable issues. Even though we did not manage to exploit them or their impact is not clear, they might represent a security risk in the near future. We suggest fixing them **as soon as possible**.
- **Minor:** These issues represent problems that are relatively small or difficult to take advantage of but can be exploited in combination with other issues. These kinds of issues do not block deployments in production environments. They should be taken into account and be fixed **when possible**.

Issues Status

An issue detected by this audit can have four distinct statuses:

- **Unresolved**: The issue has not been resolved.
- **Acknowledged**: The issue remains in the code but is a result of an intentional decision.

- **Resolved**: Adjusted program implementation to eliminate the risk.
- **Partially resolved**: Adjusted program implementation to eliminate part of the risk. The other part remains in the code but is a result of an intentional decision.
- **Mitigated**: Implemented actions to minimize the impact or likelihood of the risk

Critical Severity Issues

No issues found.

Medium Severity Issues

No issues found.

Minor Severity Issues

MI-01 Possible Reentrancy Issues

Location:

- `contracts/minter/MintingFactoryV2.sol`

While `accessControls`, `koda`, `marketplace`, `gatedMarketplace` and `royaltiesRegistry` contracts are considered trustworthy, a crafty attacker may eventually find a way to use those to trigger a reentrancy attack in the `MintingFactoryV2` contract.

Recommendation

Mark all the `MintingFactoryV2.mint*()` functions as non-reentrant using the `nonReentrant` modifier defined in the OpenZeppelin library. Reentrancy issues are not easy to spot and the checks-effects-interaction pattern cannot be trivially applied in this contract.

Status

Acknowledged.

MI-02 Outdated Solidity Version

The solidity version declared in the audited contracts is 0.8.4. The latest version released at the time of this audit is 0.8.13.

Recommendation

Test all contracts with the latest version and change the `pragma solidity` statements.

Status

Acknowledged.

MI-03 Unchecked Transfer

Location:

- `contracts/marketplace/BaseUpgradableMarketplace.sol:92`

When the admin recovers ERC20 tokens in the `BaseUpgradableMarketplace` contract, the contract does not control if the transfer to the new address is made or not.

This issue is minor because:

- Only the admin can recover ERC20 tokens.
- If the check fails, the only difference is that a `AdminRecoverERC20` event is emitted when it should not be.

Recommendation

Use OpenZeppelin's `safeTransfer()` function to transfer ERC20s.

Status

Resolved. Checked on commit `3bcd94f66e5d0f6b38881fd52971c13dd08b6974`.

MI-04 Possible DOS on KODAV3UpgradableGatedMarketplace.mint()

Location:

- `contracts/marketplace/BaseUpgradableMarketplace.sol:149,152`

When `KODAV3UpgradableGatedMarketplace.mint()` is called, eventually `BaseUpgradableMarketplace._handleSalesFunds()` is called several times. In this function, 2 ether transfers are made, one to the `platformAddress` and the other to the `fundsReceiver` of the sale.

If any of those addresses rejects the transfer, then the minting is aborted.

The severity of this issue was lowered because neither the platform nor the fund receiver has clear economic incentives to do this and the DOS is transient.

Recommendation

Use the withdrawal pattern[1] to transfer funds to those addresses. This may also result in gas savings for multiple mints, as a single transfer would be required for each address.

Status

Mitigated. The development team informed us that they don't want to introduce the pull pattern at this point due to UX complexity. mitigation can be achieved by an admin running the `KODAV3UpgradableGatedMarketplace.updateFundsReceiver()` function. The development team also informs that the funds handler is derived from the collab registry, so they can also disable minting access to the malicious artists if found and also update the funds handler in this scenario.

Enhancements

These items do not represent a security risk. They are best practices that we suggest implementing.

Table

ID	Title	Status
EN-01	NameTime Lock Mechanism to Transfer Funds and Upgrade	Not implemented
EN-02	Missing Non-Zero Checks	Implemented
EN-03	Commissions Calculations	Partially implemented

Details

EN-01 Time Lock Mechanism to Transfer Funds and Upgrade

Location:

- `contracts/marketplace/BaseUpgradableMarketplace.sol:87-100`

If an attacker takes control of an address with the admin role, it can steal all the funds belonging to a `BaseUpgradableMarketplace` derived contract, including

[1] https://docs.soliditylang.org/en/v0.8.13/common-patterns.html#withdrawal-from-contracts

KODAV3UpgradableGatedMarketplace. This operation may be made instantly, giving the real admins no recourse to stop the operation.

Recommendation

Make the operations to recover ERC20s and ether, and also the upgrade of the contract, time locked. This would give the possibility to prevent this attack from happening.

Status

Not implemented. The development team acknowledges this recommendation but will not implement it.

EN-02 Missing Non-Zero Checks

- contracts/marketplace/BaseUpgradableMarketplace.sol:78,96,131

The BaseUpgradableMarketplace contract is missing the checks for zero addresses, which are implemented in its child contract KODAV3UpgradableGatedMarketplace.

This may lead to using an uninitialized address as target for a transfer, losing those funds.

Recommendation

Add the non-zero checks to the following function parameters:

- _platformAccount in the BaseUpgradableMarketplace.initialize() function.
- _recipient in the BaseUpgradableMarketplace.recoverStuckETH() function.
- _newPlatformAccount in the BaseUpgradableMarketplace.updatePlatformAccount() function.

Status

Implemented. Checked on commit 3bcd94f66e5d0f6b38881fd52971c13dd08b6974.

EN-03 Commissions Calculations

Location:

- contracts/marketplace/BaseUpgradableMarketplace.sol:29-32, 58-67, 113-117, 146-154
- contracts/marketplace/KODAV3UpgradableGatedMarketplace.sol:70, 102-105, 306-311, 396-401

There are several things that can be improved in the commissions calculations:

1. Decouple modulo for each commision. Now if the admin changes the modulo then all the commissions would be altered. It would be better for each commission to have its own modulo.
2. The value for koCommissionOverrideForSale[_saleId].koCommission (set in KODAV3UpgradableGatedMarketplace.sol) and platformPrimaryCommission (set in BaseUpgradableMarketplace.sol) should be less than its corresponding modulo.
3. Given that the current supply of wei is about 1.2e26[2], the multiplication in line 147 of BaseUpgradableMarketplace.sol would not overflow if written as msg.value * _platformCommission / modulo, given that _platform commission is less than 10^{50}. Checking that is less than 10^{40} would ensure that it would not overflow for millennia. And by multiplying before dividing the calculation of the commission would be more precise. If other blockchains are added (different from ethereum), these numbers should be revised accordingly to avoid overflows.

Status

Partially implemented. Hereunder there is the split of this recommendation implementation:

1. **Not implemented.**
2. **Not implemented.**
3. **Implemented.** Checked on commit 3bcd94f66e5d0f6b38881fd52971c13dd08b6974.

Other Considerations

The considerations stated in this section are not right or wrong. We do not suggest any action to fix them. But we consider that they may be of interest for other stakeholders of the project, including users of the audited contracts, owners or project investors.

Trusted Contracts

When analyzing the MintingFactoryV2 contract we assumed that the accessControls, koda, marketplace, gatedMarketplace and

[2] Consulted from https://ycharts.com/indicators/ethereum_supply while writing this report.

`royaltiesRegistry` contracts are trustworthy. Those contracts are set in the initialization phase, which is invoked during the deployment of the contract.

When analyzing the `KODAV3UpgradableGatedMarketplace` contract and its parent, `BaseUpgradableMarketplace`, we assumed that the `accessControls` and `koda` contracts are trustworthy. Those contracts are set in the initialization phase, which is invoked during the deployment of the contract. The `accessControls` contract can also be changed by an `admin`, using the `BaseUpgradableMarketplace.updateAccessControls()` function.

Centralization and Upgrades

The contracts `MintingFactoryV2`, `KODAV3UpgradableGatedMarketplace` and `BaseUpgradableMarketplace` are quite centralized. In particular, an address with the admin role may trigger an upgrade, so being able to run any code. An administrator may also transfer any ERC20 or ether belonging to a `BaseUpgradableMarketplace` derived contract, including `KODAV3UpgradableGatedMarketplace`, to any arbitrary address.

In the `KODAV3UpgradableGatedMarketplace` contract, addresses with the contract role also have significant capabilities to interfere with any sale.

Changelog

- 2022-04-12 – Initial report based on commit `d592c5f4fa4e0b6fc65a1fce43e302706aedf607`.
- 2022-04-18 – Check fixes on commit `3bcd94f66e5d0f6b38881fd52971c13dd08b6974`.
- 2022-04-28 – Check bug fix on commit `9cd490474667bf021c7b0c1e8b3c697fc3072f3a`

Disclaimer: This audit report is not a security warranty, investment advice, or an approval of the KnownOrigin project since CoinFabrik has not reviewed its platform. Moreover, it does not provide a smart contract code faultlessness guarantee.