VIBE SECURITY AUDIT REPORT

June 30, 2023

MixBytes()

TABLE OF CONTENTS

1. INTRODUCTION

1.1 Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, investment advice, endorsement of the platform or its products, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only. The information presented in this report is confidential and privileged. If you are reading this report, you agree to keep it confidential, not to copy, disclose or disseminate without the agreement of the Client. If you are not the intended recipient(s) of this document, please note that any disclosure, copying or dissemination of its content is strictly forbidden.

1.2 Security Assessment Methodology

A group of auditors are involved in the work on the audit. The security engineers check the provided source code independently of each other in accordance with the methodology described below:

1. Project architecture review:

- Project documentation review.
- General code review.
- Reverse research and study of the project architecture on the source code alone.

Stage goals

- Build an independent view of the project's architecture.
- Identifying logical flaws.

2. Checking the code in accordance with the vulnerabilities checklist:

- Manual code check for vulnerabilities listed on the Contractor's internal checklist. The Contractor's checklist is constantly updated based on the analysis of hacks, research, and audit of the clients' codes.
- Code check with the use of static analyzers (i.e Slither, Mythril, etc).

Stage goal

Eliminate typical vulnerabilities (e.g. reentrancy, gas limit, flash loan attacks etc.).

3. Checking the code for compliance with the desired security model:

- Detailed study of the project documentation.
- Examination of contracts tests.
- Examination of comments in code.
- Comparison of the desired model obtained during the study with the reversed view obtained during the blind audit.
- Exploits PoC development with the use of such programs as Brownie and Hardhat.

Stage goal

Detect inconsistencies with the desired model.

4. Consolidation of the auditors' interim reports into one:

- Cross check: each auditor reviews the reports of the others.
- Discussion of the issues found by the auditors.
- Issuance of an interim audit report.

Stage goals

- Double-check all the found issues to make sure they are relevant and the determined threat level is correct.
- Provide the Client with an interim report.

5. Bug fixing & re-audit:

- The Client either fixes the issues or provides comments on the issues found by the auditors. Feedback from the Customer must be received on every issue/bug so that the Contractor can assign them a status (either "fixed" or "acknowledged").
- Upon completion of the bug fixing, the auditors double-check each fix and assign it a specific status, providing a proof link to the fix.
- A re-audited report is issued.

Stage goals

- Verify the fixed code version with all the recommendations and its statuses.
- Provide the Client with a re-audited report.

6. Final code verification and issuance of a public audit report:

- The Customer deploys the re-audited source code on the mainnet.
- The Contractor verifies the deployed code with the re-audited version and checks them for compliance.
- If the versions of the code match, the Contractor issues a public audit report.

Stage goals

- Conduct the final check of the code deployed on the mainnet.
- Provide the Customer with a public audit report.

Finding Severity breakdown

All vulnerabilities discovered during the audit are classified based on their potential severity and have the following classification:

Severity	Description
Critical	Bugs leading to assets theft, fund access locking, or any other loss of funds.
High	Bugs that can trigger a contract failure. Further recovery is possible only by manual modification of the contract state or replacement.
Medium	Bugs that can break the intended contract logic or expose it to DoS attacks, but do not cause direct loss funds.
Low	Bugs that do not have a significant immediate impact and could be easily fixed.

Based on the feedback received from the Customer regarding the list of findings discovered by the Contractor, they are assigned the following statuses:

Status	Description
Fixed	Recommended fixes have been made to the project code and no longer affect its security.
Acknowledged	The Customer is aware of the finding. Recommendations for the finding are planned to be resolved in the future.

1.3 Project Overview

The Vibe project can be described as an NFT sale factory. Users can deploy new collections and at the same time create sales of these collections. It is possible to set prices for tokens, select the ERC20 token as a payment token, and even create NFT collections with different tiers.

1.4 Project Dashboard

Project Summary

Title	Description
Client	Vibe
Project name	Vibe
Timeline	June 12 2023 - June 30 2023
Number of Auditors	4

Project Log

Date	Commit Hash	Note
12.06.2023	d08057edbaf83b00d94dcaca2a05e3c44a45e4d9	Commit for the audit

Date	Commit Hash	Note
21.06.2023	985a4265c300124ada0a6ec557f64ff2c9526ac1	Commit for the re-audit

Project Scope

The audit covered the following files:

File name	Link
NFTMintSaleMultiple.sol	NFTMintSaleMultiple.sol
NFTMintSaleWhitelisting.sol	NFTMintSaleWhitelisting.sol
NFTMintSaleWhitelistingMultiple.sol	NFTMintSaleWhitelistingMultiple.sol
NFTMintSale.sol	NFTMintSale.sol
MintSaleBase.sol	MintSaleBase.sol
SimpleFactory.sol	SimpleFactory.sol
RoyaltyReceiver.sol	RoyaltyReceiver.sol
VibeERC721.sol	VibeERC721.sol

Deployments

Ethereum

File name	Contract deployed on mainnet	Comment
NFTMintSale.sol	0xa218aecA6Ea21f151fB9b1ee77a6f77B65efD1F9	
NFTMintSaleMultiple.sol	0xE24b47E9490898237a7209F13d2BbD8440499743	

File name	Contract deployed on mainnet	Comment
NFTMintSaleWhitelisting.sol	0x644E56Fc5027Fce48ae922078beB4708e69d1EC1	
NFTMintSaleWhitelistingMultiple.sol	0x4b11A37B8EE1cA2e331A22f7D5dbb08840C9FcAf	
RoyaltyReceiver.sol	0xbBD235F96E1a2DcB52083c25E09b5833E4C66C10	
SimpleFactory.sol	0x5AFc0A0be968b9Ff72e8e11cA8b76518Fd687F21	
VibeERC721.sol	0x9F53A09524bb782C23Ec352B20688820F270d28d	

Arbitrum

File name	Contract deployed on mainnet	Comment
NFTMintSale.sol	0x5372F72eC96591E1Cc477c8Fb4e2194d646D9BC2	
NFTMintSaleMultiple.sol	0x116FCdaC02D2FFE7663fCdBDF3aC3Eae383AEb6b	
NFTMintSaleWhitelisting.sol	0x50b077A696E58075c53899223D653cFa9Cf11B1A	
NFTMintSaleWhitelistingMultiple.sol	0x40AEc802260bb4047BE2E545ec6381b01b785150	
RoyaltyReceiver.sol	0xa2FB7c0Da7fb819B0FC6e80d2d6b19fDe874E02F	
SimpleFactory.sol	0x6cb83598b3D2CBc86CCcD5B037FCc5889e24a23b	
VibeERC721.sol	0xA3e5cfaF294b415663776908CB06c105d364f050	

1.5 Summary of findings

Severity	# of Findings
Critical	1
High	2
Medium	10
Low	21

ID	Name	Severity	Status
C-1	Unrestricted access to the `setVibeFees` function in `MintSaleBase`	Critical	Fixed
H-1	Potential reinitialization of the `VibeERC721` contract	High	Fixed
H-2	`NFTMintSaleMultiple` won't work if other minter mints tokens with an id intersected with the id range on sale	High	Acknowledged
M-1	Lack of checks in the `setVibesFee` function	Medium	Fixed
M-2	Lack of checks in the `init` function of `NFTMintSale` and `NFTMintSaleMultiple` contracts	Medium	Fixed
M-3	External mints affect the number of tokens available for sale in `NFTMintSale`	Medium	Acknowledged
M-4	Lack of checks on the payed amount in `buyNFT` and `buyMultipleNFT` functions if `paymentToken` is `WETH`	Medium	Fixed
M-5	Lack of checks in the `setMerkleTree` function of `NFTMintSaleWhitelistingMultiple`	Medium	Acknowledged
M-6	Lack of checks of tier range in the `init` function of the `NFTMintSaleMultiple` contract	Medium	Fixed

M-7	Lack of checks within the `init` function of `RoyaltyReceiver`	Medium	Acknowledged
M-8	Ownership is not set in `createERC721` in some cases	Medium	Acknowledged
M-9	Royalty parameters are not checked	Medium	Fixed
M-10	`OPERATOR_FILTER_REGISTRY` can be missing in a new network	Medium	Acknowledged
L-1	`NON_WHITELISTED_MAX_PER_USER` of `NFTMintSaleWhitelistingMultiple` is shared between the tiers	Low	Acknowledged
L-2	Unused variables	Low	Fixed
L-3	Lack of event emission in setter functions	Low	Fixed
L-4	`paymentToken` argument within the `getPayment` function can be removed	Low	Fixed
L-5	Unused function parameter	Low	Fixed
L-6	Repetition of time range checks in the `buyMultipleNFT` functions	Low	Fixed
L-7	Multiple `getPayment` calls within the `buyMultipleNFT` function of `NFTMintSaleMultiple` can be simplified	Low	Fixed
L-8	The `NON_WHITELISTED_MAX_PER_USER` parameter is taking effect only if `merkleRoot` is not set	Low	Acknowledged
L-9	Inconsistent order of `name` and `symbol` arguments in creation functions	Low	Acknowledged
L-10	`SimpleFactory` can be drained out of ether	Low	Fixed
L-11	Dust can be left on the contract because of the rounding	Low	Acknowledged
L-12	Unnecessary inheritance from `Ownable`	Low	Fixed
L-13	Missing zero address and zero total checks in `claimEarnings`	Low	Fixed

L-14	`renounceMinter` can be called by anyone	Low	Fixed
L-15	The missing check for `newEndTime`	Low	Fixed
L-16	`totalSupply` doesn't represent the actual token supply	Low	Acknowledged
L-17	`require` checks without messages	Low	Fixed
L-18	The current minter status is not checked	Low	Fixed
L-19	`claimed` can be updated for non-existing tiers	Low	Acknowledged
L-20	Fees are not distributed at the beginning of `RoyaltyReceiver.setRecipientsAndBPS()`	Low	Acknowledged
L-21	NFT-owner can run minting again after `MintSaleBase.removeTokensAndReclaimOwnership()` was called	Low	Acknowledged

1.6 Conclusion

During the audit process 1 CRITICAL, 2 HIGH, 10 MEDIUM, and 21 LOW severity findings were spotted.
After working on the reported findings, all of them were acknowledged or fixed by the client.
Most findings can be classified as "Unrestricted input parameters value". We recommend adding this kind of check to further development of the protocol. It will increase protocol security because more restrictions on input parameters always mean fewer possible ways of breaking smart contracts logic.

2.FINDINGS REPORT

2.1 Critical

C-1	Unrestricted access to the `setVibeFees` function in `MintSaleBase`
Severity	Critical
Status	Fixed in 985a4265

Description

The`setVibeFees` function within the `MintSaleBase` contract is protected by the `onlyMasterContractOwner` modifier. However, anyone can call this function if the contract is the implementation contract itself, meaning it has no master contract. An exploiter via this vulnerability can monitor the deployment transactions of the `MintSaleBase` inheritor contracts and submit their transaction before deployment, thereby setting the `vibeTreasury` field to their address and arbitrarily increasing the value of the `feeTake` field. Consequently, the owner of the `masterContract` must reset the values back and re-establish the `fee` parameters of the affected copy.

Recommendation

We recommend adding the following require statement to the `onlyMasterContractOwner` modifier that verifies if the contract doesn't have a master contract. In such cases, only the owner of the contract should be allowed to pass this modifier. We propose adding the following line here: [MintSaleBase.sol#L69]

```
require(msg.sender == owner())
```

2.2 High

H-1	Potential reinitialization of the `VibeERC721` contract
Severity	High
Status	Fixed in 985a4265

Description

The `baseURI` field can be set to an empty string using the `changeBaseURI` function VibeERC721.sol#L223. It allows to call the `init` function by anyone and subsequently claim the ownership of the entire `VibeERC721` contract VibeERC721.sol#L79.

Recommendation

We recommend adding the following check to the `changeBaseURI` function:
`require(bytes(baseURI)_.length != 0);`

H-2	`NFTMintSaleMultiple` won't work if other minter mints tokens with an id intersected with the id range on sale
Severity	High
Status	Acknowledged

Description

If an external minter of `nft` mints a token with an id that falls within the range of ids specified for sale, the functionality of `NFTMintSaleMultiple` can be disrupted. When such an intersecting mint occurs, the `buyNFT` function will fail to execute as the `nft.mintWithId(recipient, id)` NFTMintSaleMultiple.sol#L77 call will revert due to the conflicting id.

Recommendation

We recommend allowing to use only `mint` or `mintWithId` in a specific instance of the VibeERC721 contract.

Client's commentary

> Expected behavior, UI should verify

2.3 Medium

M-1	Lack of checks in the `setVibesFee` function
Severity	Medium
Status	Fixed in 985a4265

Description

The `setVibesFee` function currently lacks checks to ensure that the `vibeTreasury_` parameter is not set to the zero address and that the `feeTake_` parameter does not exceed the `BPS`. It can potentially lead to a DoS scenario if these parameters are mistakenly set in such a way.

Recommendation

We recommend setting the following checks within the `setVibesFee` function MintSaleBase.sol#L76

```
require(vibeTreasury_ != address(0));
require(feeTake_ <= BPS);
```

M-2	Lack of checks in the `init` function of `NFTMintSale` and `NFTMintSaleMultiple` contracts
Severity	Medium
Status	Fixed in 985a4265

Description

The `init` function in both the `NFTMintSale` and `NFTMintSaleMultiple` contracts currently lacks checks to prevent certain scenarios that can lead to unintended consequences. Firstly, there is no check to ensure that the `proxy` argument is not set to the zero address which can potentially lead to the reinitalization of the contract with the complete loss of ownership. Additionally, there are no checks to verify that `endTime >= beginTime` and that `beginTime > block.timestamp`. These checks are essential to maintain the integrity of the contract and prevent violations of the specified invariants.

Recommendation

We recommend implementing the stated checks within the `init` function of both the `NFTMintSale` NFTMintSale.sol#L31 and `NFTMintSaleMultiple` NFTMintSaleMultiple.sol#L30 contracts.

M-3	External mints affect the number of tokens available for sale in `NFTMintSale`
Severity	Medium
Status	Acknowledged

Description

If an external minter of `nft` mints a token, it affects the `nft.totalSupply()` value, reducing the number of available tokens for sale in the `NFTMintSale` contract.

Recommendation

We recommend using a local counter of minted tokens within the `NFTMintSale` contract instead of relying solely on the `nft.totalSupply()` NFTMintSale.sol#L59. This approach will prevent interference between external mints and the availability of tokens for sale.

Client's commentary

> This is intended behavior, there should not be external mints otherwise NFTMintSaleMultiple should be used for sales of specific ranges

M-4	Lack of checks on the payed amount in `buyNFT` and `buyMultipleNFT` functions if `paymentToken` is `WETH`
Severity	Medium
Status	Fixed in 985a4265

Description

The `buyNFT` and `buyMultipleNFT` functions currently lack checks to verify that `msg.value` matches the expected payment amount when the `paymentToken` is set to `WETH`. If users accidentally send more ether than the intended payment amount, the funds that exceed the payment amount will be locked within the contract forever, as there is no mechanism to redeem ether from the contract.

Recommendation

We recommend adding the following checks to `buyNFT` (NFTMintSale.sol#L69, NFTMintSaleMultiple.sol#L76)
and `buyMultipleNFT` (NFTMintSale.sol#L80, NFTMintSaleMultiple.sol#L86) functions:

```
if (paymentToken == WETH) {
    require(msg.value == payedAmount)
} else {
    require(msg.value == 0)
}
```

Here, the `payedAmount` is the sum of prices for the tokens being purchased.

M-5	Lack of checks in the `setMerkleTree` function of `NFTMintSaleWhitelistingMultiple`
Severity	Medium
Status	Acknowledged

Description

The `setMerkleTree` function in the `NFTMintSaleWhitelistingMultiple` contract currently lacks checks to ensure that the `merkleRoot` values are set for all tiers. It allows the contract owner to accidentally set the `merkleRoot` of higher tiers to `bytes32(0)`, effectively allowing anyone to mint up to the `NON_WHITELISTED_MAX_PER_USER` amount of tokens from those tiers.

Recommendation

We recommend adding the following check within `setMerkleRoot` NFTMintSaleWhitelistingMultiple.sol#L28 function:

```
require(
    merkleRoot_.length == tiers.length &&
    merkleRoot_.length == externalURI_.length
);
```

It ensures that the `merkleRoot` values are set intentionally for all the tiers.

Client's commentary

> Client: This is an intended behavior to allow non whitelisted tiers
> MixBytes(): if merkleRoot_.length < tiers.length unintentionally, then a user will incorrectly set parameters for the last tiers

M-6	Lack of checks of tier range in the `init` function of the `NFTMintSaleMultiple` contract
Severity	Medium
Status	Fixed in 985a4265

Description

The `init` function within the `NFTMintSaleMultiple` contract currently verifies that the id ranges of different tiers do not intersect with each other and appear in increasing order. However, there is a missing check for the last tier with index `0`.
NFTMintSaleMultiple.sol#L62

Recommendation

We recommend adding a corresponding check for the last tier in the `init` function of the `NFTMintSaleMultiple` contract.

M-7	Lack of checks within the `init` function of `RoyaltyReceiver`
Severity	Medium
Status	Acknowledged

Description

The `init` function within the `RoyaltyReceiver` contract currently lacks a check to ensure that the lengths of the `recipients_` and `recipientsBPS_` arrays are equal. It leaves the contract exposed to a potential DoS until the owner of the contract calls `setRecipientsAndBPS` again to rectify the mismatched lengths.

Recommendation

We recommend adding a check within the init function to verify that `recipients_.length == recipientsBPS_.length`.
RoyaltyReceiver.sol#L19

M-8	Ownership is not set in `createERC721` in some cases
Severity	Medium
Status	Acknowledged

Description

The user can call the `createERC721` function declared here - VibeFactory.sol#L352. If the user set the `owner` parameter to `address(0)` then `SimpleFactory` would own the `VibeERC721` contract and anyone would be able to trigger `transferOwnership` here - SimpleFactory.sol#L35. If the user wanted to transfer ownership to themself in the second step, then their second transaction could be frontrunned by an attacker, a new minter can be set and then ownership can be given back to `SimpleFactory` so that users tx would succeed.

Recommendation

We recommend transferring ownership to `msg.sender` in the `createERC721` function if the `owner` parameter is equal to `address(0)`. There are cases when `createERC721` is intentionally called with `address(0)` to leave `factory` as an owner for future initialization. Instead of `address(0)` - `address(factory)` can be passed.

Client's commentary

> The frontend should handle these errors and flag invalid deployments

M-9	Royalty parameters are not checked
Severity	Medium
Status	Fixed in 985a4265

Description

Royalty parameters are used to calculate how much fees should be paid to the royalty receiver VibeERC721.sol#L205-L210. If `royaltyRate_` is greater than `BPS` then royalty payment will revert.

Recommendation

We recommend adding checks for royalty parameters.

M-10	`OPERATOR_FILTER_REGISTRY` can be missing in a new network
Severity	Medium
Status	Acknowledged

Description

`VibeERC721` contract uses `onlyAllowedOperatorApproval` and `onlyAllowedOperator` modifiers that required deployed `OPERATOR_FILTER_REGISTRY` for correct work VibeERC721.sol#L132 But it can be not deployed on a new networks.

Recommendation

We recommend adding a check that `OPERATOR_FILTER_REGISTRY` is deployed on the network that will be used for the protocol deployment.

Client's commentary

> New deployments can be verified and are currently not planned for networks where this contract is not deployed

2.4 Low

L-1	`NON_WHITELISTED_MAX_PER_USER` of `NFTMintSaleWhitelistingMultiple` is shared between the tiers
Severity	Low
Status	Acknowledged

Description

The `NON_WHITELISTED_MAX_PER_USER` parameter is currently shared across all tiers in the `NFTMintSaleWhitelistingMultiple` contract NFTMintSaleWhitelistingMultiple.sol#L53. This means that non-whitelisted users are allowed to purchase the same maximum amount of tokens for each tier.

Recommendation

We recommend modifying the parameter `NON_WHITELISTED_MAX_PER_USER` to be an array that describes the maximum amount of tokens non-whitelisted users are allowed to mint for each tier.

Client's commentary

> This variable comes only in play if a tier is not whitelisted so one variable should be sufficient

L-2	Unused variables
Severity	Low
Status	Fixed in 985a4265

Description

`masterNFT` within `MintSaleBase` MintSaleBase.sol#LL41 is unused.

Recommendation

We recommend removing the `masterNFT` variable.

L-3	Lack of event emission in setter functions
Severity	Low
Status	Fixed in 985a4265

Description

The following setter functions do not emit any events:

- `setVibesFee` within `NFTMintSaleBase` MintSaleBase.sol#L76,
- `setMerkleRoot` within `NFTMintSaleWhitelisting` NFTMintSaleWhitelisting.sol#L26,
- `setMerkleRoot` within `NFTMintSaleWhitelistingMultiple` NFTMintSaleWhitelistingMultiple.sol#L27.

Recommendation

We recommend adding `emit event` statements within the listed functions.

L-4	`paymentToken` argument within the `getPayment` function can be removed
Severity	Low
Status	Fixed in 985a4265

Description

The `getPayment` internal function within the `NFTMintSaleBase` contract currently includes the `paymentToken` argument MintSaleBase.sol#L80. All the occurrences of this function are called with the value of`paymentToken` field of the `NFTMintSaleBase` contract.

Recommendation

We recommend removing the `paymentToken` argument from the `getPayment` function.

L-5	Unused function parameter
Severity	Low
Status	Fixed in 985a4265

Description

At line RoyaltyReceiver.sol#L64 the `distribute` function is declared. It has an `amount` as an input parameter, but its value is never used.

Recommendation

We recommend removing the `amount` parameter and not passing the `total - fee` value here - MintSaleBase.sol#L99.

L-6	Repetition of time range checks in the `buyMultipleNFT` functions
Severity	Low
Status	Fixed in 985a4265

Description

The `buyMultipleNFT` function in both the `NFTMintSale` and `NFTMintSaleMultiple` contracts currently calculates check that the sale is still ongoing multiple times. It results in redundant calculations of the same expression NFTMintSale.sol#L59, NFTMintSaleMultiple.sol#L75.

Recommendation

We recommend moving this time range check to the beginning of the corresponding `buyNFT` and `buyMultipleNFT` functions ensuring this check is performed only once and refactoring the base logic of the `buyNFT` function of the `NFTMintSaleMultiple` contract to the internal `_buyNFT` function.

L-7	Multiple `getPayment` calls within the `buyMultipleNFT` function of `NFTMintSaleMultiple` can be simplified
Severity	Low
Status	Fixed in 985a4265

Description

The `buyMultipleNFT` function in the `NFTMintSaleMultiple` contract currently makes multiple calls to `buyNFT` in a loop NFTMintSaleMultiple.sol#L84, which results in multiple calls to `getPayment` for each minted token. This can be simplified by using an accumulator variable to keep track of the total price of the minted tokens. With this accumulator sum, you can make a single call to the `getPayment` function after all the tokens have been minted.

Recommendation

We recommend using one `getPayment` call here using the accumulator sum variable representing the total sum price.

L-8	The `NON_WHITELISTED_MAX_PER_USER` parameter is taking effect only if `merkleRoot` is not set
Severity	Low
Status	Acknowledged

Description

The current implementation of the `_preBuyCheck` function in the `NFTMintSaleWhitelisting` contract relies on the `merkleRoot` being set to `bytes32(0)` to make the `NON_WHITELISTED_MAX_PER_USER` parameter the maximum limit of tokens to mint NFTMintSaleWhitelisting.sol#L40. This means that if the `merkleRoot` is set, the `NON_WHITELISTED_MAX_PER_USER` parameter has no effect, making it impossible to create a sale that allows whitelisted users to have a different maximum mint amount than non-whitelisted users.

Recommendation

We recommend adjusting the `require` statement within the `_preBuyCheck` functions (NFTMintSaleWhitelisting.sol#L33, NFTMintSaleWhitelistingMultiple.sol#L36) as follows:

```
require(
    claimed[msg.sender].claimed < claimed[msg.sender].max ||
    claimed[msg.sender].claimed < NON_WHITELISTED_MAX_PER_USER
);
```

With this adjustment, the `require` statement ensures that a user can buy up to the `NON_WHITELISTED_MAX_PER_USER` amount of NFT tokens if they haven't called `initUser`. This change allows for more fine-grained control over the maximum mint amount for whitelisted and non-whitelisted users.

After making this adjustment, the `else` branch in the `initUser` function becomes redundant and can be removed (NFTMintSaleWhitelisting.sol#L50, NFTMintSaleWhitelistingMultiple.sol#L53). Additionally, this change allows the owner of the sale to set `NON_WHITELISTED_MAX_PER_USER` to `0` if they don't want to sell tokens to non-whitelisted users.

Client's commentary

> This behavior is as expected and per design

L-9	Inconsistent order of `name` and `symbol` arguments in creation functions
Severity	Low
Status	Acknowledged

Description

The order of the `name` and `symbol` arguments in the creation functions of the `VibeFactory` contract (`createERC721` VibeFactory.sol#L353, `createNFTMintSaleMultiple` VibeFactory.sol#L279, `createNFTMintSaleMultipleWhitelisting` VibeFactory.sol#L238, `createNFTMintSaleWhitelisting` VibeFactory.sol#L165, `createNFTMintSale` VibeFactory.sol#L120) differs, causing potential confusion for users who interact with these functions. Since both arguments have the same type, inconsistent ordering may lead to unintended parameter assignments and mistakes.

Recommendation

We recommend using the same order for the `name` and `symbol` arguments in all the creation functions of the `VibeFactory` contract.

Client's commentary

> Will be potentially addressed at a later date

L-10	`SimpleFactory` can be drained out of ether
Severity	Low
Status	Fixed in 985a4265

Description

The `SimpleFactory` contract inherits from `BoringFactory` and `BoringBatchable`. `BoringBatchable` acts as a multicall and `BoringFactory` uses `msg.value` inside the `deploy` function. It is possible for `SimpleFactory` to store ether as it has an `exec` function which is not `payable`, but uses `value` with a call. An attacker can provide their contract to the `deploy` function and create multiple deployments using `batch`. Together with a batch attacker can provide a small `msg.value` which will be duplicated inside the `deploy` function on `init`. The attackers' contract would send the `msg.value` amount to a pre-defined address in the `init` function. It will lead to `SimpleFactory` being drained out of funds.

Recommendation

We recommend changing the `exec` function and making it `payable` here - SimpleFactory.sol#L39 so that in future there is no need to pre-fund `SimpleFactory` with ether.

L-11	Dust can be left on the contract because of the rounding
Severity	Low
Status	Acknowledged

Description

At line RoyaltyReceiver.sol#L58 transferrable `amount` is calculated based on `totalAmount` and pre-defined `recipientBPS[i]`. In some cases, there will be a residue here `totalAmount * recipientBPS[i] / BPS` which will be stuck in the contract.

Recommendation

We recommend sending dust to any of the recipients.

Client's commentary

> Dust is insignificant

L-12	Unnecessary inheritance from `Ownable`
Severity	Low
Status	Fixed in 985a4265

Description

`NFTMintSale` and `NFTMintSaleMultiple` contracts are inherited from `MintSaleBase` and `Ownable`. But `MintSaleBase` is already inherited from `Ownable`.

Recommendation

We recommend removing unnecessary inheritance from `Ownable` in `NFTMintSale` and `NFTMintSaleMultiple` contracts.

L-13	Missing zero address and zero total checks in `claimEarnings`
Severity	Low
Status	Fixed in 985a4265

Description

There is a `claimEarnings` function here - MintSaleBase.sol#LL88. It accepts the `proceedRecipient` address as a parameter and uses the `total` variable which represents the contract balance in pre-defined `paymentToken`. In cases when the `total` is zero ,`claimEarnings` call would still be completed. Also, it is possible to transfer funds to a zero address.

Recommendation

We recommend adding checks that `proceedRecipient` is not a zero address and `total` is not equal to zero.

L-14	`renounceMinter` can be called by anyone
Severity	Low
Status	Fixed in 985a4265

Description

The `renounceMinter` function is declared here - VibeERC721.sol#L68. That function can be called by anyone and it emits the `LogMinterChange` event. Such events can be misleading for some external tools.

Recommendation

We recommend adding the `require(isMinter[msg.sender])` check at the beginning of the function. It will ensure that the caller is an actual minter.

L-15	The missing check for `newEndTime`
Severity	Low
Status	Fixed in 985a4265

Description

There is an `extendEndTime` function here - MintSaleBase.sol#L124. It allows contract owner to set new `endTime` for sale. But there is no check that `newEndTime` is bigger than `beginTime`. `beginTime` may be in the future, so it is possible to make a mistake.

Recommendation

We recommend adding a check that `newEndTime` is bigger than `beginTime`. Also, check that `nft.isMinter(address(this)) == true` can be added to exclude cases when the current minter was renounced.

L-16	`totalSupply` doesn't represent the actual token supply
Severity	Low
Status	Acknowledged

Description

The `totalSupply` variable is used here - VibeERC721.sol#L87. It helps to define the next minted token id. But in cases of `burn` or `mintWithId` that variable doesn't represent actual total token supply.

Recommendation

We recommend using different naming for the variable which is used to define the next token id.

Client's commentary

> In case only mint() is used the variable represents the totalSupply

L-17	`require` checks without messages
Severity	Low
Status	Fixed in 985a4265

Description

There are a few `require` checks without corresponding messages - RoyaltyReceiver.sol#L21, RoyaltyReceiver.sol#L32, RoyaltyReceiver.sol#L42, RoyaltyReceiver.sol#L49, VibeERC721.sol#L59, VibeERC721.sol#L79, VibeERC721.sol#L187, VibeERC721.sol#L200, MintSaleBase.sol#L67, MintSaleBase.sol#L125, NFTMintSale.sol#L59, NFTMintSaleMultiple.sol#L75.

Recommendation

We recommend adding necessary messages to the mentioned checks.

L-18	The current minter status is not checked
Severity	Low
Status	Fixed in 985a4265

Description

Status can be changed to the same one here VibeERC721.sol#L64.

Recommendation

We recommend adding a check that the new status differs from the previous one.

L-19	`claimed` can be updated for non-existing tiers
Severity	Low
Status	Acknowledged

Description

It is possible to update `claimed` mapping for non-existing tiers NFTMintSaleWhitelistingMultiple.sol#L56

Recommendation

We recommend adding a check that the `claimed` mapping cannot be updated for non-existing tiers.

Client's commentary

> That is fine

L-20	Fees are not distributed at the beginning of `RoyaltyReceiver.setRecipientsAndBPS()`
Severity	Low
Status	Acknowledged

Description

If `RoyaltyReceiver.setRecipientsAndBPS()` is called, collected to that moment royalties are not distributed between old recipients. So, old royalties will be collected by new recipients.

Recommendation

We recommend distributing already collected royalties at the beginning of `RoyaltyReceiver.setRecipientsAndBPS()`.

Client's commentary

> Client: should be to be decided by the owner
> MixBytes(): It is better to allow call of the function only for the owner

L-21	NFT-owner can run minting again after `MintSaleBase.removeTokensAndReclaimOwnership()` was called
Severity	Low
Status	Acknowledged

Description

After `MintSaleBase.removeTokensAndReclaimOwnership()` was called by NFT-owner, they can make a minting contract minter again with `VibeERC721.setMinter()`:
VibeERC721.sol#L63
So, multiple `SaleEnded` or `SaleEndedEarly` events can be emitted for the same minting. If these events are used somewhere, it can lead to problems.

Recommendation

We recommend adding some flags so that the minting is finished.

Client's commentary

> That is correct but intended

3. ABOUT MIXBYTES

MixBytes is a team of blockchain developers, auditors and analysts keen on decentralized systems. We build opensource solutions, smart contracts and blockchain protocols, perform security audits, work on benchmarking and software testing solutions, do research and tech consultancy.

Contacts

https://github.com/mixbytes/audits_public

https://mixbytes.io/

hello@mixbytes.io

https://twitter.com/mixbytes