Mintra Audit

September 2023

By CoinFabrik

v202307

Executive Summary

CoinFabrik was asked to audit the contracts for the Mintra project.

During this audit we found no critical issues, one medium issue and several minor issues. Also, several enhancements were proposed.

The medium issue and most of the minor issues were resolved by the development team. The rest of the minor issues are mitigated. All the enhancement proposals were either implemented or partially implemented.

Scope

The audited files are from the git repository located at https://gitlab.com/gerawrdog/mintra-solidity.git.

The scope for this audit includes and is limited to the following files:

- `contracts/feesplitter/FeeSplitter.sol`: The `FeeSplitter` contract distributes transaction fees among addresses, performs token swaps, burns, and manages rewards, ensuring equitable distribution and proper ecosystem incentives.
- `contracts/marketplace/ERC1155Marketplace.sol`: Marketplace for ERC1155 assets.
- `contracts/marketplace/ERC721Marketplace.sol`: Marketplace for ERC721 assets.
- `contracts/marketplace/Marketplace.sol`: It contains the base abstract contract used to implement marketplaces.
- `contracts/msi/libraries/NFTDescriptor.sol`: It contains a library used to generate the token uri for the MSI token.

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.

The following commits were reviewed, in order, as part of this audit:

1. `7d59c9f492c9d95f479320b41ddfe064ba45b7e9`
2. `664f4434721671eb567cdc55da332f732fc9289a`
3. `4fa19210a8a8f42fd9b9bca0119b64c84c68fedb`
4. `c53135eb77234891c0de62a5b40651d58027c035`
5. `e523b30716a5568177dce28c5d85e98613fe5ab8`
6. `51f53e5a31b20ad6ae988a1b6a541441f87b828d`
7. `42a8714a66542781d28a0563372c1051f363f4ea`
8. `ead5ca13fdc216f0b0613ecd782509faae0601cb`
9. `35f52b6fa93b097cb767b168991095659f9a33cc`

10. `2d2493ab1077924a8b519b51b4a765df3c714b34`
11. `6a493d28af6945a824b3904b71e7adedf6b505d5`
12. `11023b21151ca1af63156c1fb90cd8a6ec439524`
13. `b7f13c1d8bb6b12e9fcf34090dc29248d597c5f5`

The final status of all the issues and enhancements stated in this report corresponds to the latest commit in this list. All the listed commits were at the top of the `main` branch when we used them to conduct the audit.

Methodology

CoinFabrik was provided with the source code, including automated tests that define the expected behavior. Our auditors spent four weeks auditing the source code provided, which includes understanding the context of use, analyzing the boundaries of the expected behavior of each contract and function, understanding the implementation by the development team (including dependencies beyond the scope to be audited) and identifying possible situations in which the code allows the caller to reach a state that exposes some vulnerability. 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

All the issues and enhancement proposals were communicated to the development team in the course of the audit and they provided the corresponding fixes. After each fix was provided we checked that the fix was properly applied and that no additional issues were introduced. This report includes the final status of all the reported issues and enhancement proposals.

Findings

In the following table we summarize the security issues we found in this audit. The severity classification criteria and the status meaning are explained below. This table does not include the enhancements we suggest to implement, which are described in a specific section after the security issues.

ID	Title	Severity	Status
ME-01	No Safe Transfers	Medium	Resolved
MI-01	Use EnumerableMap	Minor	Resolved
MI-02	Unrestrained ERC20 Allowances	Minor	Resolved
MI-03	Marketplace Payout Denial of Service	Minor	Mitigated
MI-04	`FeeSplitter.flush()` Denial of Service	Minor	Mitigated
MI-05	FeeSplitter DEXes Derived Addresses	Minor	Resolved

Severity Classification

Security risks are classified as follows:

- **Critical:** These are issues that we manage to exploit. They compromise the system seriously. Blocking bugs are also included in this category. 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 might 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 has one of the following 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

ME-01 No Safe Transfers

Found on commit: `7d59c9f492c9d95f479320b41ddfe064ba45b7e9`
Location:

- `contracts/marketplace/ERC1155Marketplace.sol: 242`
- `contracts/marketplace/ERC721Marketplace.sol: 192`
- `contracts/marketplace/Marketplace.sol: 383,386,533`

Some ERC20 tokens return `false` instead of raising an error when the operation fails. This possibility is not handled in any of the marketplaces code and may lead to transferring assets to an attacker without obtaining its payment.

This issue's severity was lowered, given that all the ERC20 tokens used to do payments in the marketplaces are vetted by the wizard.

Recommendation

Use the OpenZeppelin SafeERC20 library[1] to interact with the ERC20 token contracts. The SafeERC20 library is designed to transparently handle this problem. This is especially important when doing token transfers.

[1] https://docs.openzeppelin.com/contracts/4.x/api/token/erc20#SafeERC20

Status
Resolved. Fixed according to the recommendation. Fix checked on commit `4fa19210a8a8f42fd9b9bca0119b64c84c68fedb`.

Minor Severity Issues

MI-01 Use EnumerableMap

Found on commit: `7d59c9f492c9d95f479320b41ddfe064ba45b7e9`
Location:

- `contracts/marketplace/Marketplace.sol: 77-78,581-597,599-602`

The `allowedTokenAddresses` and `allowedTokenAddressesArray` variables, defined in lines 77-78 of `contracts/marketplace/Marketplace.sol`, show a reimplementation of OpenZeppelin's `EnumerableMap` library[2]. But this reimplementation is not ideal. In particular, the implementation of the `addTokenAddress()` function in lines 581-597 should not need the array iteration in lines 586-590 and it would be trivial if using `EnumerableMap`. On the other hand, it may not be necessary at all given that neither the `allowedTokenAddressesArray` contents nor the `allowedAssets()` function are not being used anywhere else in the code, so it is likely that instead of using `EnumerableMap`, just eliminating the `allowedTokenAddressesArray` variable is good enough.
This is marked as a minor issue since if the `allowedTokenAddressesArray` grows too much then no new token addresses can be added, as gas may be exhausted.

Recommendation
Evaluate if it is required to iterate on the elements of the `allowedTokenAddresses` mapping. If so, use `EnumerableMap`. If not, eliminate the `allowedTokenAddressesArray` variable and its uses.

Status
Resolved. The `allowedTokenAddressesArray` variable was removed. Fix checked on commit `664f4434721671eb567cdc55da332f732fc9289a`.

MI-02 Unrestrained ERC20 Allowances

Found on commit: `7d59c9f492c9d95f479320b41ddfe064ba45b7e9`
Location:

- `contracts/feesplitter/FeeSplitter.sol: 439-440`

[2] https://docs.openzeppelin.com/contracts/4.x/api/utils#EnumerableMap

In the `FeeSplitter.approveERC20()` function the `plsxRouterV1` and `plsxRouterV2` addresses are awarded an infinite allowance for the `erc20Address` token on behalf of the contract. This is not a good security practice, as the minimum required capabilities should be given to external contracts.

Recommendation
Only give enough allowance to the addresses, and only when the operation is about to occur. In the case of smart contracts, allowance should be given just prior to the function invocation and revoked afterwards.

Status
Resolved. Now the approval is made in the `processErc20()` function for the funds that are being exchanged. The `approveErc20()` function was removed. Fix checked on commit `42a8714a66542781d28a0563372c1051f363f4ea`.

MI-03 Marketplace Payout Denial of Service

Found on commit: `664f4434721671eb567cdc55da332f732fc9289a`
Location:

- `contracts/marketplace/Marketplace.sol: 360,367`

The payout process, defined in the `Marketplace.payout()` function, can be interrupted by the `feeSplitter` when `executing payable(feeSplitterAddress).call{value: marketFee}("")` in line 360. It might also be interrupted if the `IERC20(_tokenAddress).transfer(feeSplitterAddress, marketFee);` in line 367 fails.

This issue severity was lowered because:

- the `feeSplitter` is set in the constructor, and cannot be changed afterwards.
- the ERC20 token used in line 367 is whitelisted by the wizard.

This issue is related to the ME-01 issue of the June 2023 audit.

Recommendation
Use the withdrawal pattern to transfer funds to the `feeSplitter`.

Status
Mitigated. The development team informed us that the `feeSplitter` passed in the contract's constructor corresponds to the `FeeSplitter` contract, also reviewed in this audit.

MI-04 `FeeSplitter.flush()` Denial of Service

Found on commit: `c53135eb77234891c0de62a5b40651d58027c035`
Location:

- `contracts/feesplitter/FeeSplitter.sol: 208,212`

The withdrawal pattern was not used to transfer funds to the `rootAddress` and the `mintStakingAddress` in the `FeeSplitter.flush()` function. This makes the flush procedure susceptible to being interrupted by the `rootAddress` or the `mintStakingAddress` accounts.
This issue's severity was lowered given that the `rootAddress` and the `mintStakingAddress` are set in the contract's constructor and cannot be changed afterwards. This issue is related to MI-01 in the June 2023 audit.

Recommendation
Use the withdrawal pattern to transfer funds to the `rootAddress` and the `mintStakingAddress` accounts.

Status
Mitigated. The development team informed us that they control the `rootAddress` and the `mintStakingAddress` accounts.

MI-05 FeeSplitter DEXes Derived Addresses

Found on commit: `e523b30716a5568177dce28c5d85e98613fe5ab8`
Location:

- `contracts/feesplitter/FeeSplitter.sol: 156-157,160-161`

Both the factories (`pulseXFactoryV1`, `pulseXFactoryV2`) and the MINT pairs (`plsMintPairV1`, `plsMintPairV2`) can be obtained from the routers (`plsxRouterV1`, `plsxRouterV2`). A failure to set the proper addresses may lead to an incorrect contract behavior.
This issue is considered minor as we consider it similar to a lack of zero-check in a constructor.

Recommendation
Instead of receiving the factories and routers via constructor parameters, obtain them from the routers instead.

Status
Resolved. Now the factory and pair are obtained from the router. Fix checked on commit `42a8714a66542781d28a0563372c1051f363f4ea`.

Enhancements

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

ID	Title	Status
EN-01	Royalties Coupling	Implemented

ID	Title	Status
EN-02	Configurable Routers	Implemented
EN-03	`FeeSplitter.flush()` Improvements	Partially implemented

EN-01 Royalties Coupling

Found on commit: `664f4434721671eb567cdc55da332f732fc9289a`
Location:

- `contracts/marketplace/Marketplace.sol: 410-411`

When setting a new receiver via the `Marketplace.createOrUpdateRoyality()` function, the royalties basis points need to be decreased as well, as this is enforced in the function code.

Recommendation

Change the `_royaltyInBasisPoints < royalties[_collectionAddress].basisPoints` check in line 410 to be `_royaltyInBasisPoints <= royalties[_collectionAddress].basisPoints`.

Status

Implemented. Fixed according to the recommendation. Fix checked on commit `4fa19210a8a8f42fd9b9bca0119b64c84c68fedb`.

EN-02 Configurable Routers

Found on commit: `e523b30716a5568177dce28c5d85e98613fe5ab8`
Location:

- `contracts/feesplitter/FeeSplitter.sol`

The `FeeSplitter` contract has the logic to handle 2 routers, because currently there are 2 different routers in the PLS blockchain. But it is not a given that those 2 will be the only 2 trustworthy routers in the blockchain.

Recommendation

Allow the vulcan to configure other routers to be used in the `FeeSplitter` contract. When doing so, also change the the `approveERC20()` function to receive the router where the token is approved.

Status

Implemented. New routers can be configured via the `changeRouterAddresses()` function. The `approveERC20()` function was removed as part of the MI-02 resolution. Fix checked on commit `42a8714a66542781d28a0563372c1051f363f4ea`.

EN-03 `FeeSplitter.flush()` Improvements

Found on commit: `51f53e5a31b20ad6ae988a1b6a541441f87b828d`
Location:

- `contracts/feesplitter/FeeSplitter.sol`

Currently the `FeeSplitter.flush()` function has some drawbacks that can be addressed. Those are:

1. There is no incentive for a caller to pass a non-zero value in the `plsAmountToBuyAndBurnMintWith` parameter. Passing 0 effectively makes the buy and burn part of the process void.
2. The `amounts` parameter is not needed, as the slippage calculation is done anyway.
3. The `burn()` call in the `processErc20s()` function is not needed (see line 243), as MINT transferred to the contract will be burned anyway in the `buyBackAndBurn()` function also invoked by `FeeSplitter.flush()` (see line 384).
4. In the `FeeSplitter.flush()` documentation, the fact that the MINT tokens are not used to generate funds for the `rootAddress`, the `mintStakingAddress` or the msg.sender is not clear.

Recommendation

1. Calculate the amounts to exchange for each ERC20. This should not be more expensive (in gas) than the current logic given that the slippage calculations are done anyway.
2. Calculate the amounts used to buy MINT for burning. This should also not be more expensive (in gas) than the current logic given that the slippage calculations are done anyway.
3. Give some rewards using a small part of the funds used to buy and burn, to incentivize that procedure.
4. Remove the unnecessary parameters (`plsAmountToBuyAndBurnMintWith` and `amounts`).
5. Document how MINT funds transferred to the contract are used in the `FeeSplitter.flush()` documentation.
6. Remove `mint()` call in line 243.

Status

Partially implemented. The `burn()` call in `processErc20s()` function was removed. The documentation for the `FeeSplitter.flush()` function was updated. Changes checked on commit `42a8714a66542781d28a0563372c1051f363f4ea`.

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 to other stakeholders of the project, including users of the audited contracts, token holders or project investors.

Previous Audit

Older versions of the contracts audited here, along with other contracts of this project, were audited by Coinfabrik in the June 2023 audit.

Only the difference with the last commit of the first audit[3] was audited for the `contracts/msi/MSIFactory.sol` file.

The rest of the files included in this audit (`contracts/feesplitter/FeeSplitter.sol`, `contracts/marketplace/ERC1155Marketplace.sol`, `contracts/marketplace/ERC721Marketplace.sol` and `contracts/marketplace/Marketplace.sol`) were audited in full.

Centralization

The `Marketplace` wizard role and the `FeeSplitter` vulcan role show some degree of centralization, but there is a conscious effort made by the development team to keep their power to a minimum. Please see the Privileged Roles section for details.

There are no admin-like roles in any of the audited contracts with the capabilities to upgrade the deployed code, stop the contracts functionality or withdraw the contract funds. This is a double edged sword. While as a user of the contracts we can be sure that the code will not be changed by any malicious attacker or administrator, if a critical bug is found after the deployment of any of the audited contracts there is no way an administrator can rescue the funds or assets locked in the contracts nor stop users from using the contracts.

Upgrades

There are no mechanisms to upgrade the audited contracts.

Privileged Roles

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

[3] `48188721057fd632c5b324a5521eb5e672f4136a`

Marketplace

Wizard

The wizard can:

- Set the market fee percentage via the `setMarketPercent()` function. This value is used when the transacting token is not the MINT token. The value can be set between 0% and 3.69%. The initial value is 2.25%.
- Add a new ERC20 token to pay for items sold in the marketplace via the `addTokenAddress()` function.

The account associated with this role is set in the contract's constructor (passed via the `_wizard` parameter) and cannot be changed afterwards.

Collection Owner

A collection owner can set the royalty charged when sales are conducted through the contract and what address receives those royalties if the collection does not implement the ERC2981 by calling the `createOrUpdateRoyality()` function. The collection owner is determined by calling the `owner()` function in the collection.

FeeSplitter

Vulcan

The vulcan can:

- change the maximum slippage via the `changeSlippage()` function. The slippage is generated when tokens are burnt. The maximum tolerated slippage can be set between 0.1% and 15%.
- allow the `plsxRouterV1` and `plsxRouterV2` contracts to operate on behalf of this contract for an ERC20 address via the `approveERC20()` function. This functionality was removed as part of the resolution of MI-02.
- configure the routers used to do the exchanges via the `changeRouterAddresses()`. This functionality was added following the EN-02 recommendation.
- configure the fees for each router via the changeRouterFees() function. This functionality was added while this audit was made and checked on commit `ead5ca13fdc216f0b0613ecd782509faae0601cb`.

The account associated with this role is set in the contract's constructor (passed via the `_vulcan` parameter) and cannot be changed afterwards.

EOA

Only an EOA can execute the `flush()` function. This function implements the process used to give out the funds accumulated in the contract. These funds are split in equal parts and given to the `mintStakingAddress`, the `rootAddress` and the caller of the function.

Marketplace Items

An item represents an item to be sold in any of the marketplaces. In the `ERC721Marketplace` contract is identified by the (contract address, token id) pair of the underlying token. But in the `ERC1155Marketplace` the (contract address, token id, seller address) triplet is used given that the same token may have multiple instances.

There are 3 types of items:

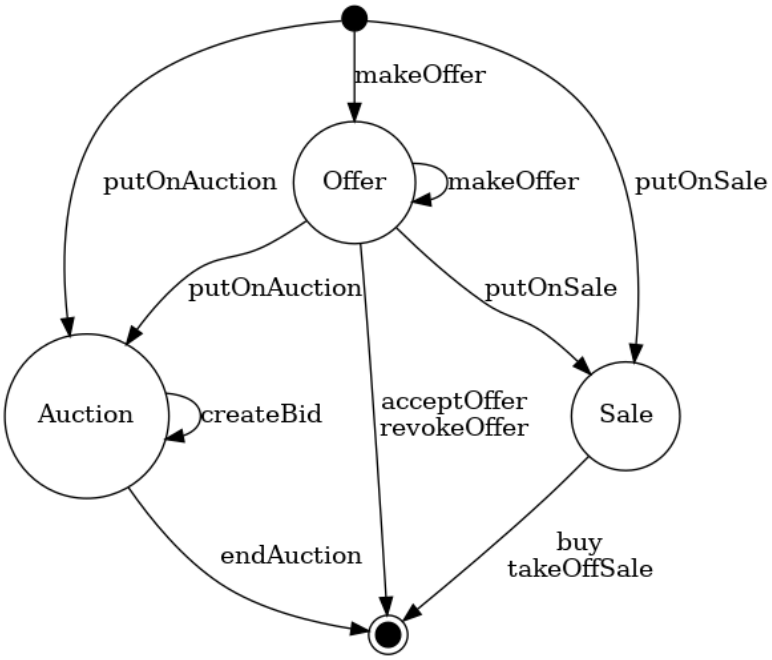
- Offer: It means that someone offered to buy a token not being sold in the store.
- Auction: The owner of the token sells it via an auction handled by the marketplace.
- Sale: The owner of the tokens sells it at a predetermined price.

Items can be put on sale or on auction on a single token at any given time. Offers for items can only be made in PLS[4].

It must be noted that an ERC1155 offer item is also identified by a seller address, so ERC1155 offers are for a single seller.

This state diagram shows the possible item states, and applies to both the `ERC721Marketplace` and the `ERC1155Marketplace` contracts. The edges are noted with the names of the external or public functions used to change the item state.

[4] PLS is the native token in the Pulse blockchain.



The `bulk*()` functions in the marketplace contracts just iterate on the single item actions, so they are not included in this diagram for simplicity.

`Marketplace` Royalties

The `Marketplace` contract gives royalties when operations are performed on tokens.

If the token implements ERC2981, checked by probing the token's `royaltyInfo()` function, then royalties are sent to the royalty receiver according to the declaration made by the token contract.

If the token does not implement ERC2981, then the token needs to implement the `owner()` function, and royalties are distributed according to the settings made via the `Marketplace.createOrUpdateRoyality()` function.

If neither of those conditions is respected, then no royalties are distributed.

It must be noted that royalties can be increased and exceed the maximum royalty percentage (stored in the `Marketplace.maxRoyaltyBasisPoints` variable) for ERC2981-compliant tokens but not for other tokens, as the `createOrUpdateRoyality()` function ensures that the royalty points are not increased on successive function invocations.

For commit `b7f13c1d8bb6b12e9fcf34090dc29248d597c5f5` a check was added to the payout so the sale transaction will be reverted if the royalty is too high. See `contracts/marketplace/Marketplace.sol:312`.

`FeeSplitter` Slippage Calculation

In the June 2023 audited code there was a price slippage calculation implemented in the `BuyAndBurn.calculateSlippage()` function. This calculation directly measured the price variation.

As part of the development work made after that audit, the `FeeSplitter` contract and the `BuyAndBurn` contract were fused into the current `FeeSplitter` contract. So the slippage calculation was moved to the `FeeSplitter.calculateSlippage()` function.

But this new function does not calculate the price slippage. Instead, it calculates the reserve variation increase in the DEX.

Given that the development team informed us that they are going to be using the pulse routers, deployed at `0x1715a3E4A142d8b698131108995174F37aEBA10D` and `0x165C3410fC91EF562C50559f7d2289fEbed552d9` in the PulseChain production network, and that those contracts follow the standard way for DEXes to calculate the exchanges by keeping the the multiplication of the reserves constant, we know that constraining the variation on the reserves of the in-token in an exchange will also limit the price variation, but this price variation limit will be bigger than the reserves variation.

For example, if the in-reserve increases 15%[5] the exchange rate will increase 32% approximately.

Changelog

- 2023-08-16 – Reported MI-01.
- 2023-08-22 – Reported ME-01, MI-02.
- 2023-08-25 – Checked fix for MI-01. Suggested EN-01. Reported mitigated issue MI-03.
- 2023-08-31 – MI-02 was acknowledged by the development team. Checked fix for ME-01. Checked implementation for EN-01. Reported mitigated issue MI-04.
- 2023-09-06 – Reported MI-05. Suggested EN-02, EN-03.
- 2023-09-07 – Checked fixes for MI-02, MI-05. Check implementations for EN-02, EN-03.
- 2023-09-08 – Add Executive Summary, Scope, Methodology, and Other Considerations sections.
- 2023-09-12 – Add to scope commit for non-security bug fix.
- 2023-09-13 – Add to scope commit for another non-security bug fix.
- 2023-09-18 – Add to scope commit for another non-security bug fix.
- 2023-10-19 – Add to scope commit for other non-security bug fixes. Document new restriction on royalties.

[5] This is the maximum possible in-reserve increase configurable.

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