OpenZeppelin | security

Freeverse Crypto Payments Audit

21st of July, 2022

This security assessment was prepared by **OpenZeppelin**, protecting the open economy.

Table of Contents

Summary

The OpenZeppelin security services team audited the crypto-payments project of the Freeverse team. This escrow project is about bidding on or buying assets that are referenced through a `paymentId`, where all payment details are determined and signed by an arbitrator, defined as the `operator` role.

Type	DeFi
Timeline	From 2022-06-20 To 2022-07-06
Languages	Solidity

Total Issues	15 (14 resolved, 1 partially resolved)
Critical Severity Issues	0 (0 resolved)
High Severity Issues	0 (0 resolved)
Medium Severity Issues	3 (2 resolved, 1 partially resolved)
Low Severity Issues	7 (7 resolved)
Notes & Additional Information	5 (5 resolved)

Scope

The Freeverse team asked us to audit the freeverseio/crypto-payments repository, which we reviewed for the `956a98e53ef540f7b2b40f3b923d0b0e3e5d32ef` commit over the course of 2.5 weeks.

In scope were the following contracts:

```
contracts/
├── auction
│   ├── AuctionERC20.sol
│   ├── AuctionNative.sol
│   ├── IAuctionERC20.sol
│   ├── IAuctionNative.sol
│   └── base
│       ├── AuctionBase.sol
│       ├── EIP712VerifierAuction.sol
│       ├── IAuctionBase.sol
│       ├── IEIP712VerifierAuction.sol
│       └── ISignableStructsAuction.sol
└── buyNow
    ├── BuyNowERC20.sol
    ├── BuyNowNative.sol
    ├── IBuyNowERC20.sol
    ├── IBuyNowNative.sol
    └── base
        ├── BuyNowBase.sol
        └── IBuyNowBase.sol
```

System Overview

Freeverse's `crypto-payments` project implements a marketplace that enables users to sell assets from Freeverse universes. This audit reviewed both the existing `BuyNow` and the new `Auction` capabilities of the marketplace.

A critical part of this marketplace are the arbitrators. Arbitrators are called `operators` across the contracts and are responsible for most of the steps throughout sales. They are responsible for signing listings, signing bids, and finalizing sales. Any and all validation, excluding relayed bids, is done by the `operator` off-chain and verified on-chain.

According to the Freeverse team, the motivation behind the extensive verification, such as requiring each bid to be signed, is due to KYC requirements. Since this marketplace is expected to be used throughout the world, the team must comply with every jurisdiction, resulting in this strict verification scheme.

Each universe will be able to assign an `operator`, which enables them to moderate asset sales in the marketplace. There will be an instance of this contract for each currency type, be it the native currency or any supported ERC20s. Since the `operator` roles are saved directly on the contracts, a universe can have a unique `operator` for each currency, and will have to set new ones as new ERC20s are supported.

There are contract wide configurations, set by the owner of the contract, as well as universe wide configurations, which are set by the `operators`. The settings available are around various bidding and buy-now configurations.

When bidding on or buying a listing, a user will need to have their offer signed by the `operator` before submitting it on-chain. The Freeverse team mentioned that in order to facilitate fast bidding, the signing is done via a service currently run by the team themselves, likely via the default operator role.

The two payment modes, buy-now and auction, are implemented in their own contracts, `BuyNowBase` and `AuctionBase`, where `AuctionBase` inherits from `BuyNowBase`. The contract is further split into ERC20 and native currency implementations. The `AuctionNative` contract is deployed to support the blockchain's native currency and `AuctionERC20` contracts are deployed for each supported ERC20. Each `Auction` version of the contract contains the functionality of the `BuyNow` counterpart.

Trust Assumptions

The contracts in scope heavily rely on the `Operators` contract, which further inherits the OpenZeppelin `Ownable` contract. The contract owner is in charge of the following functions for the contracts in scope:

- `setToLocalBalanceOnOutBid`
- `setDefaultAuctionConfig`
- `setUniverseAuctionConfig`
- `removeUniverseAuctionConfig`
- `setEIP712`
- `setPaymentWindow`
- `setIsSellerRegistrationRequired`

In addition, the contract owner can set a default operator and set or remove a universe specific operator. For any buy or bid action, it is required to provide the operator's signature over the buy/bid information. This is a very powerful role as the operator could, for instance, maliciously set the seller address in their favor before publishing the listing. Similarly, the operator

determines if payment returns are allowed at the end of a sale, granting them the ability to steal either assets or money if they collude with a buyer or seller respectively.

Therefore, we have to assume that the operator acts as intended and the contract owner verifies whoever is in charge of the universe. Hence, any attacks or vulnerabilities targeting this part of the system were not considered throughout this audit.

Since this marketplace is supposed to enable users to buy and sell assets within universes, which are owned by arbitrary people and not necessarily trusted individuals, this is an incredible assumption. This could lead to issues where universes are created specifically for phishing, where buyers may be phished into purchasing assets in the wrong universe, owned by a malicious operator. Depending on the level of KYC requirements that operators must adhere to, this attack could be more or less easy to carry out.

Further, the ERC20 tokens in use are expected to be audited and safe in the context of this project.

Medium Severity

M-01 Misconfiguration could lead to blocking bids

In the `_processBid` function of the `AuctionBase` contract the time details of the auction are checked. These times are:

- `extendableUntil` - The time until an auction may be extended through late bids.
- `expirationTime` - The time until an asset must be transferred. Otherwise, the payment can be refunded by the user afterwards.

The requirement of line 118 checks the following:

```
extendableUntil + _SAFETY_TRANSFER_WINDOW < expirationTime
```

Here, the `_SAFETY_TRANSFER_WINDOW` is a constant value of two hours. Therefore, the auction must expire after the longest possible duration, including enough time to transfer the asset. Further, substituting the values of `extendableUntil` and `expirationTime` and simplifying the inequation we receive:

```
universeExtendableBy(bidInput.universeId) + _SAFETY_TRANSFER_WINDOW < _paymentWindow
```

Both values, `universeExtendableBy()` and `_paymentWindow`, are configurable by the contract owner. In the case that the system accidentally is configured to never satisfy the above condition, the requirement would always fail and so would calls to bid. No bids on new auctions would be processed, shutting down all upcoming auctions.

Consider moving this requirement and adding additional checks when setting these configurations.

Update: Fixed with commit `ef597b1` of PR#16.

M-02 Previous operator power is not revoked

During the process of a buy action within `_processBuyNow` and bid action within `_processBid`, the payment details are saved on-chain using the `_payments` mapping. This

includes the current operator address at the time of action, fetched through the `universeOperator` function within the `BuyNowERC20.buyNow`, `BuyNowERC20.relayedBuyNow`, `BuyNowNative.buyNow`, `AuctionERC20.bid`, `AuctionERC20.relayedBid`, and `AuctionNative.bid` functions.

In the case of an operator change, after the payment details have been written on-chain, there is no way to revoke the power of an operator. This means, if an operator's key was compromised and that operator is saved as the operator in charge of a listing in the `_payments` mapping, they could still finalize a payment or auction with an untruthful `transferResult.wasSuccessful` outcome even after changing the current operator for the universe.

For instance, after the first bid on an auction and a consecutive operator change, the previous operator can bid and win the auction to then get the asset transferred. But during finalization they are able to sign that the asset transfer was not successful to also get the refund. Similarly, if the operator knows they are exchanged in the future, they can buy an asset or win an auction to then maliciously finalize the payment the same way.

Consider revoking the power of any previous operator, e.g., by fetching the current operator through the `universeOperator` function inside the `_finalize` function. The returned address can then be checked against the provided signature to assure that only the current operator can finalize a payment. In addition, remove the operator from the `Payments` struct.

Update: Fixed with commit `ff4ff3b` of PR#15.

M-03 Seller's signature not required

In this protocol, listing an asset is simply having the operator sign either a `BidInput` or a `BuyNowInput` struct. From there, once a buyer either places a bid or completes a buy-it-now listing, the struct will be saved on-chain. When processing either type of struct, the only requirements around signing is a valid signature from the operator.

Nowhere in the process is it required for the seller to signal that they intend to sell their asset. Requiring the seller's signature over the listing would ensure that the origin of the listing was the seller itself. If a signature was not required, users could create phony listings that will never complete.

Since the seller is required to actually transfer an asset at the end of a listing, the worst situation that should arise would be a large number of listings that do not complete after they end, resulting in users' funds being locked for small periods of time.

Consider requiring the seller's signature along with the operator's when listing assets.

Update: Partially fixed with commit `097fa4c` of PR#18. Seller validation is done with a signature over the `paymentId`, which does not remove the need for trust between operator and seller. However, the upgradability allowed via the EIP712 implementation contract allows for future updates which remedy this.

The Freeverse team states:

> Following your suggestion, we decided to start, at least, with what we already have, and require the already-existing L2 sellerSignature to be provided to the payments contract. The data that makes paymentId is available off chain.
>
> However, we also agreed with your point and decided that it'll be nicer to move to a pattern where the seller signature is much closer to the others, so that the payments contract can verify the seller's intention pre-digest.
>
> That's why the PR intentionally provides an interface for verifySellerSignature where future upgrades of the verification contract (via the setEIP712 method) can include implementations where the verification of the sellerSig makes use of the entire BuyNow/Auction input struct, as suggested. (note that verifySellerSignature already takes as inputs the entire bidInput/buynowInput struct).
>
> Since changing the L2 current signature will take time (we first need to fill in details such as the not 1-to-1 mapping, and upgrades require coordination with nodes, etc.), we plan to do it carefully, and incorporate the suggestion using the setEIP712 method in due time.

Low Severity

L-01 Events lack previous values

There are a couple instances of events which are emitted when the state of the contract is changed. Among these events, some only record the new value and omit recording the previous value.

Here are instances of this issue:

- `event EIP712`
- `event PaymentWindow`
- `event OnlyUserCanWithdraw`
- `event DefaultFeesCollector`
- `event UniverseFeesCollector`
- `event DefaultOperator`
- `event UniverseOperator`
- `event DefaultAuctionConfig`
- `event UniverseAuctionConfig`

When using events to record state changes, it is recommended to record the previous value as well to document the entirety of the change. Consider including the previous state to any event that changes the state of the contract.

Update: Fixed with commit `6e02cc3` of PR#14.

L-02 Loose restriction on fees

Currently, fees on buys and won auctions are not restricted and can be arbitrarily set for up to 100% via the `assertBidInputsOK` function in the `BuyNowBase` and `AuctionBase` contracts. Such `feeBPS` value is given through the input data that is signed by the operator.

Despite the trust assumption towards the operator, consider limiting the fee to a lower percentage or setting it on a contract level similar to the `_paymentWindow`. By doing so it is less likely that the seller gets less funds than intended due to an unexpectedly high fee.

Update: Fixed with commit `bfa52af` of PR#13.

L-03 Withdraw with `finalBalance` adds unnecessary complexity

In the `BuyNowBase` contract, the `_withdrawAmount` function takes three arguments: The `recipient`, who's balance is withdrawn, the `amount`, and a `finalBalance`. `finalBalance` is used to set the balance state of the recipient, which adds unnecessary complexity and is error prone.

Consider updating the balance by subtracting the withdrawn amount like `_balanceOf[recipient] -= amount;` in order to simplify the function.

Update: Fixed with commit `d9bd95a` of PR#12.

L-04 `_acceptedCurrency` could be misleading

Due to the support of ERC20s and blockchain native currencies, payments are settled through the currency indicated by the `_acceptedCurrency` variable that is set during deployment in the `BuyNowBase` constructor.

Since this variable is passed in rather than obtained via calling the `name` or `symbol` functions of the ERC20s, it is possible to have a different underlying currency than the one claimed to be supported, possibly resulting in an uncareful user accidentally paying an unexpected price.

Consider implementing the `_acceptedCurrency` through the symbol or name view function of the underlying ERC20 contract with a fallback to the native currency, as well as maintaining a manual method for cases of ERC20s that do not support those functions.

Update: Fixed with commit `bd26283` of PR#17.

L-05 Gas optimization

In the `_processBid` function of the `AuctionBase` contract, the `BidInput` struct is processed from `memory`. Further, that struct is propagated to the `assertBidInputsOk` function, where the inputs are sent via `memory` again, while they are defined as `calldata` in the `IAuctionBase` interface.

Consider changing both locations from `memory` to `calldata` to save gas.

In the `assertBidInputsOk` function of the `AuctionBase` contract, the `bidInput` requirements are checked. These checks include, whether the `bidInput.deadline` is met

and if the `bidInput.seller` should be registered. Further checks are based on the state of a payment that is fetched from the state of the blockchain.

Consider moving line 229, to line 240 to save some gas getting the payment state if one of the two mentioned checks fail.

Update: Fixed with commit `00705cc` of PR#11.

L-06 Inaccurate revert string

The `withdrawTo` function will revert with the message `tx sender not authorized to withdraw on recipients behalf` in the case that the given `recipient` is set to false in the `onlyUserCanWithdraw` mapping.

This function should not revert if the passed in `recipient` address is the same as the address calling the function.

Consider updating the logic to never revert when a user attempts to withdraw their own funds with the `withdrawTo` function.

Update: Fixed with commit `b5db284` of PR#10.

L-07 Redundant check of unused and signed data

In the `assertBidInputsOk` function of the `AuctionBase` contract, incoming bid data is checked and verified based on the state of the auction. In the case of `State.Auctioning`, it is required that the `bidInput.feeBPS` and `bidInput.endsAt` values align with the values that were saved during the initial bid. Both checks are redundant for two reasons:

1. The provided call values `bidInput.feeBPS` and `bidInput.endsAt` are not being used.
2. The operator signed these values so they are expected to match. This is contradictory and undermines the given strong trust assumption.

The checks are for user experience only, mainly to guarantee on a contract level that the user does not provide different values than the initial data. This could lead to a user paying a different fee than expected.

Consider moving these checks off-chain to save some gas and simplify the code.

Update: Fixed with commit `715f28a` of PR#9.

Notes & Additional Information

N-01 Inconsistent `buyer` terminology

Throughout the codebase, there are two terms for the user who wants to acquire an asset: `buyer` and `payer`.

Consider sticking to the more frequent term `buyer` to be more explicit and precise about the involved roles.

Update: Fixed with commit `345c3c9` of PR#7.

N-02 Mismatch between documentation and implementation

In the `AuctionBase` contract, the `_processBid` function is documented to take parameter `bidInput` first and `operator` second, while the parameters are ordered the other way around in the actual function.

Consider swapping the order of the parameters in the documentation to match the implementation.

Update: Fixed with commit `f6a2da5` of PR#5.

N-03 Revert strings lack explicitness

Throughout the codebase, the revert strings could be more explicit by following the pattern of `"<Contract-name>::<function-name>: <message>`.

Consider updating the revert strings to know exactly where a transaction fails and ease debugging.

Update: Fixed with commit `9e604c1` of PR#4.

N-04 Typographical errors

During the audit of the codebase, some typographical errors were identified. For instance:

- "msg.sender must the the recipient" should be "msg.sender must be the recipient" in lines 209 and 234 of `IBuyNowBase`.
- "curretEndsAt" should be "currentEndsAt" in line 173 of `AuctionBase`.
- "exisiting" should be "existing" in line 40 and 57 of `IAuctionERC20`.
- "While the funds provided can by less than the bid amount" should be "While the funds provided can be less than the bid amount" in line 46 of `AuctionNative`.

Consider correcting above errors and any other typographical errors throughout the codebase to guarantee explicitness and correctness in the documentation.

Update: Fixed with commit `dab036d` of PR#3 and commit `0543d30` of PR#19.

N-05 Unintuitive naming of `withdrawTo`

The `withdrawTo` function of the `BuyNowBase` contract enables users to withdraw funds of another user on their behalf. For example, user `A` calls the contract so the funds of user `B` get withdrawn to user `B`.

Unfortunately, the naming of `withdrawTo` is rather unintuitive as it suggests that the funds are withdrawn from user `A` to user `B`, which could lead to confusion and unexpected reverts.

Consider renaming the function to `relayedWithdraw` or `withdrawOnBehalfOf` to better match the naming with the implementation.

Update: Fixed with commit `56b6bb0` of PR#2.

Conclusion

The code of the Freeverse team is clean, thus no major issues were found. The use of simple and self contained functions and contracts really contributed to this outcome. We found some lower severity issues and suggested some best practices that could further contribute to the overall health of the codebase. We also communicated some architectural concerns, which were acknowledged, but unchanged. We enjoyed working with Freeverse as they provided thorough explanations for any questions that we had.

Appendix

Architectural Recommendation

The current design of the payments contracts plans on having one `AuctionERC20` (including `BuyNowERC20`) contract per supported currency. These contracts allow the contract owner to set universe specific operators and auction configurations. This managing capability is fully coupled to that contract, so as soon as a new currency is supported, the whole setup of a universe would have to be reapplied to the new contract. While the design is simple in favor of readability and security, it does not scale very well.

It was suggested to decouple the managing functionality from the contract or to introduce some logic that supports multiple currencies from one contract. The concern was acknowledged by the Freeverse team.

Monitoring Recommendation

Due to the strong trust assumption towards the contract owner and operator role, we recommend implementing monitoring for all sensitive actions. For instance, to look out for suspicious inputs of the following functions:

- `BuyNowERC20.buyNow`
- `BuyNowERC20.relayedBuyNow`
- `BuyNowNative.buyNow`
- `AuctionERC20.bid`
- `AuctionERC20.relayedBid`
- `AuctionNative.bid`

Suspicious activity could include an unusually high `amount` or `feeBPS` or repetitively selling to the same `seller`. The following functions are also dependent on `operator` signatures:

- `BuyNowBase.finalize`
- `BuyNowBase.finalizeAndWithdraw`

Their `transferResult` input could be manipulated by the operator to maliciously claim that an asset was successfully traded when it wasn't. Thus, the input could be checked against off-chain data to verify the correctness of the provided data. Any suspicious activity could be an

indication that the operator might be compromised. Further, all functions that are maintainable by the contract owner should be monitored, to also keep track of operator and configuration changes. The affected functions are listed under *Trust Assumptions*.