SECURITY REVIEW OF FANTASY

Contents

Summary

About 0xWeiss
0xWeiss is an independent smart contract security researcher, Co-Founder of Enigma Dark and SR at Spearbit. He also serves as an in-house security in Tapioca DAO and Ambit Finance.

Fantasy Top v1.1
Fantasy is a Trading Card Game in which players collect cards featuring crypto influencers to compete and earn ETH, BLAST, more cards, and FAN Points.

Engagement Overview

Over the course of 4 days, 0xWeiss conducted a security review of the Fantasy v1.1 protocol via the Hyacinth platform.

The following repositories were reviewed at the specified commits:

Repository	Commit
fantasy-top/fantasy-core-audit	`28fb2b10b629ff5474eec417c693fce7cfaf4261`

Risk Classification

Severity	Description
High	Exploitable, causing loss or manipulation of assets or data.
Medium	Risk of future exploits that may or may not impact the smart contract execution.
Low	Minor code errors that may or may not impact the smart contract execution.

Vulnerability Summary

Severity	Count	Fixed	Acknowledged
High	1	1	0
Medium	2	2	0
Low	4	3	1
Informational	1	1	0

Findings

Index	Issue Title	Status
H-01	buy orders can be executed with revoked approvals	Fixed
M-01	Whitelist not checked in `setCollectionForMintConfig` and `newMintConfig`	Fixed
M-02	Users can burn from non-whitelisted collections	Fixed
L-01	`batchBuy` allows to send more ether than required but does not refund	Acknowledged
L-02	Missing buy and burn event emission	Fixed
L-03	Token whitelist is not being checked	Fixed
L-04	Missing check for array lengths	Fixed
I-01	Informational recopilation	Fixed

Detailed Findings

High Risk

H-01 - buy orders can be executed with revoked approvals

Severity: High Risk

Technical Details:

Inside the buy function, if buyers decide to set `burnAfterPurchase` as true, the NFT will be burned from the owner.

```
 if (burnAfterPurchase) {
            executionDelegate.burnFantasyCard(sellOrder.collection,
sellOrder.tokenId);
        } else {
            _executeTokenTransfer(sellOrder.collection, sellOrder.trader,
msg.sender, sellOrder.tokenId);
        }
```

There is a logical problem where it shows a flaw in the usage of the approval system. The seller can revoke authorization of sell orders by setting `revokedApproval[from] == false` in the execution delegate contract. Which in v1, it reverted if someone tried to buy when that mapping returned true.

When introducing the burn method, if the buyer decides to burn, even if the approval is revoked, they will still burn the NFT from the user as it is not checked for approvals

Impact:

Buy orders can be executed with revoked approvals

Recommendation:

Add a `revokedApproval` when burning. You can add it directly in the ExecutionDelegate contract so it gets always checked:

```
 function burnFantasyCard(address collection, uint256 tokenId) external
whenNotPaused approvedContract {
+  require(revokedApproval[from] == false, "User has revoked approval");
        IFantasyCards(collection).burn(tokenId);
    }
```

or if the protocol was expecting to use such functionality often themselves on behalf of the user, they could call the mapping from the `ExecutionDelegate` directly in the buy function.

Developer Response:

Fixed at commit: https://github.com/fantasy-top/fantasy-core-audit/pull/44

Medium Risk

M-01 - Whitelist not checked in `setCollectionForMintConfig` and `newMintConfig`

Severity: Medium Risk

Technical Details:

The `setCollectionForMintConfig` and

```
 function setCollectionForMintConfig(uint256 mintConfigId, address
collection) public onlyRole(MINT_CONFIG_MASTER) {
        require(mintConfigId < mintConfigIdCounter, "Invalid mintConfigId");
        require(collection != address(0), "Collection address cannot the
zero address");

        MintConfig storage config = mintConfigs[mintConfigId];
        require(!config.cancelled, "Mint config cancelled");
        config.collection = collection;

        emit CollectionUpdatedForMintConfig(mintConfigId, collection);
    }
```

the `newMintConfig` functions do not check that a collection is actually whitelisted:

```
  if (requiresWhitelist) {
           require(merkleRoot != 0, "missing merkleRoot");
        }

        MintConfig storage config = mintConfigs[mintConfigIdCounter];
        config.collection = collection;
```

Impact:

Whitelist not checked in `setCollectionForMintConfig` and `newMintConfig`

Recommendation:

Check that the collection being used is whitelisted:

`require(whitelistedCollections[collection], "Collection is not whitelisted");`

Developer Response:

Fixed at commit: https://github.com/fantasy-top/fantasy-core-audit/pull/43/files

M-02 - Users can burn from non-whitelisted collections

Severity: Medium Risk

Technical Details:

`sellOrder.collection` is not checked against the whitelist in the case the buyer sets `burnAfterPurchase` as true. it is just checked inside the `_executeTokenTransfer` function:

```
 if (burnAfterPurchase) {
            executionDelegate.burnFantasyCard(sellOrder.collection,
sellOrder.tokenId);
        } else {
            _executeTokenTransfer(sellOrder.collection, sellOrder.trader,
msg.sender, sellOrder.tokenId);
        }
```

Impact:

Users can burn from non-whitelisted collections

Recommendation:

Check the whitelist status of the collection also when burning.

Developer Response:

Fixed at commit: https://github.com/fantasy-top/fantasy-core-audit/pull/42/files

Low Risk

L-01 - `batchBuy` allows to send more ether than required but does not refund

Severity: Low Risk

Technical Details:

`batchBuy` allows to send more ether than required but does not refund:

```
 if (sellOrders[i].paymentToken == address(0)) {
                totalEthSpending += sellOrders[i].price;
                require(totalEthSpending <= msg.value, "Insufficient ETH
sent");
            }
```

Impact:

When more ether than required is sent, it will be lost

Recommendation:

Add a refund mechanism at the end of the function

Developer Response:

Acknowledged

L-02 - Missing buy and burn event emission

Severity: Low Risk

Technical Details:

When the standard `buy` function gets called, there is no distinction between the event emitted when the token gets burned vs when it doesn't, it always emits the following event: `emit Buy(msg.sender, sellOrder, sellOrderHash);`

In the batch buy function, it does make a distinction whether the user burns or not by emitting `BatchBuyAndBurn` :

```
    if (burnAfterPurchase) {
            emit BatchBuyAndBurn(msg.sender, sellOrders, sellerSignatures);
        } else {
            emit BatchBuy(msg.sender, sellOrders, sellerSignatures);
        }
```

Impact:

State is not tracked properly

Recommendation:

Add a `BuyAndBurn` event on the standard `buy` function

```
  if (burnAfterPurchase) {
            executionDelegate.burnFantasyCard(sellOrder.collection,
sellOrder.tokenId);
+       emit BuyAndBurn(msg.sender, sellOrder, sellOrderHash);
        } else {
            _executeTokenTransfer(sellOrder.collection, sellOrder.trader,
msg.sender, sellOrder.tokenId);
        }
        emit Buy(msg.sender, sellOrder, sellOrderHash);
    }
```

Developer Response: TODO

L-03 - Token whitelist is not being checked

Severity: Low Risk

Technical Details:

The `setMinimumPricePerPaymentToken` does not check that the `paymentToken` is whitelisted:

```
  function setMinimumPricePerPaymentToken(address paymentToken, uint256
minimuPrice) public onlyOwner {
        _setMinimumPricePerPaymentToken(paymentToken, minimuPrice);
    }
```

while it should check the whitelist status of the token:
`whitelistedPaymentTokens[_paymentToken]`

Impact:

State can be modified for un-whitelisted tokens

Recommendation:

Add the `whitelistedPaymentTokens` check to the function

Developer Response:

Fixed at commit: https://github.com/fantasy-top/fantasy-core-audit/pull/39

L-04 - Missing check for array lengths

Severity: Low Risk

Technical Details:

In the function `batchMintCardsTo` the check to make sure that `merkleProofs` and recipients` have the same length is missing:

```
function batchMintCardsTo(uint256 configId, bytes32[][] calldata
merkleProofs, uint256 maxPrice, address[] calldata recipients) public
payable nonReentrant onlyEOA onlyRole(MINT_CONFIG_MASTER) {
        for (uint i = 0; i < recipients.length; i++) {
            _mintCardsTo(configId, merkleProofs[i], maxPrice,
recipients[i]);
        }
    }
```

Impact:

Missing important check

Recommendation:

Check that the length between `merkleProofs` and `recipients` is the same

Developer Response:
FIxed at commit: a7377dc37d6e632334b41c22deeec15c77887ed0

Informational

I-01 - Informational recopilation

Severity: Informational

Technical Details:

- The NATSPEC in the `batchMintCardsTo` function has a typo, should be `to mint` instead of `to mints` : `* @notice Admin function to mints packs based on the specified mint configuration to multiple recipients` .
- There could be a standardized internal `_mint` function to be called inside `batchMintCardsTo` and `mint` as both function do basically the same. The only difference is the `to` parameter, which could be set to `msg.sender` in the `mint` function
- The `batchBurn` does an unnecessary loop, you could burn just after the check:

```
      for (uint i = 0; i < tokenIds.length; i++) {
            require(IFantasyCards(collection).ownerOf(tokenIds[i]) ==
msg.sender, "caller does not own one of the tokens");
+         executionDelegate.burnFantasyCard(address(collection),
tokenIds[i]);
        }

-        for (uint i = 0; i < tokenIds.length; i++) {
-            executionDelegate.burnFantasyCard(address(collection),
tokenIds[i]);
-        }
```

Impact:

Informational issues

Recommendation:

Fix the above issues accordingly

Developer Response:

Fixed at commit: https://github.com/fantasy-top/fantasy-core-audit/pull/41/files

Disclaimer

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts 0xWeiss to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology. Blockchain technology and cryptographic assets present a high level of ongoing risk.

My position is that each company and individual are responsible for their own due diligence and continuous security. My goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze. Therefore, I do not guarantee the explicit security of the audited smart contract, regardless of the verdict.