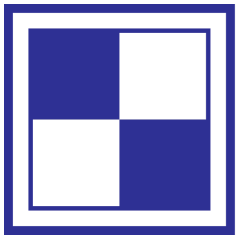

Perfect Abstractions

Tokentrust Canvas Audit

Table of Contents

III Low Risk

50 Disclaimer

1 Tokentrust Canvas Audit

Perfect Abstractions conducted a smart contract audit of Tokentrust's Canvas Contracts from October 27th to November 10th, 2022.

The git commit hash used for the audit is `3aafa6e3c42dc5509d2e9c33bdd3780bc9006522`.

Auditors:

- Gašper Pregelj

Audit report reviewed by Nick Mudge.

1.1 Overview

The Tokentrust Canvas contracts provide a configurable token creation protocol with various sales mechanisms, distribution methods and output customizability. There is an option to create a single NFT, which is described as 1/1 or `isOne` in the code, or a collection of multiple NFTs. The term `canvas` is used to describe a single NFT in case of 1/1, or a collection.

The main contract `Canvas.sol` provides the entrypoint interface for transactions and view functions, while `Schema.sol` outlines a data storage schema which is shared across the entire protocol. It also defines utility structs and errors.

The `Modules` folder contains library contracts that utilize and modify the storage, which allows robust functionality within a single contract address.

The `Collection` folder contains an ERC721 factory contract that deploys a new ERC721 contract for each new Canvas collection. 1/1 Canvases (single NFTs) don't need new ERC721 contracts to be deployed. They are all contained within two ERC721 contracts. One is permissionless (`canvasOne`) and the other is curated (`canvasOneCurated`). To create an NFT in the curated one, you need to be permitted by the curator. To create it in the permissionless one, you do not need any permission.

The `Chainlink` folder contains contracts that depend on an external oracle network for verifiable decentralized on-chain randomness (`VRF.sol`) and contract automation (`Keeper.sol`).

A small subset of the protocol has dependencies on two separate contract repositories, `TokenRegistry` and `LicenseRegistry`. These were not part of the audit.

1.1.1 Objectives

1. Find bugs, inefficiencies, design flaws and security vulnerabilities in the code base.
2. Report and make recommendations concerning what was found.

1.1.2 Scope

The following files were audited:

- contracts/Schema.sol
- contracts/Canvas.sol
- contracts/Utils/Parser.sol
- contracts/Modules/View.sol
- contracts/Modules/URI.sol
- contracts/Modules/Minting.sol
- contracts/Modules/Funds.sol
- contracts/Modules/Customize.sol
- contracts/Modules/Create.sol
- contracts/Modules/Core.sol
- contracts/Collection/CanvasCreator.sol
- contracts/Collection/CanvasCollection.sol
- contracts/Chainlink/VRF.sol
- contracts/Chainlink/Keeper.sol

I. High Risk

2 Changes after sale enable theft of funds

High Risk

In Create.sol there is `updateCanvas` function that is used to update a canvas. You can update `saleStart` and `saleEnd` as long as the sale has not started, or the sale ended, and it is not a refundable dutch. But changes to `saleStart` and `saleEnd` after sale ends enables other changes to also be made. This enables stealing of funds.

2.1 Examples

2.1.1 Example 1

1. User A creates a canvas and sets it up for a sale with `saleStart` set to current date, 7am and `saleEnd` current date, 8am. Price is 10 ETH
2. He buys an NFT that is part of the canvas for 10 ETH. He receives the 10 ETH back because he is buying from himself.
3. It is now 9am and the sale expired.
4. He updates canvas and sets new `saleStart` to 10am and `saleEnd` to 11am.
5. Now the `saleStarted` check is false. So user A can again execute a canvas update and set it as refundable dutch with end price of 1 ETH.
6. He buys another NFT through the refundable dutch for 1 ETH.
7. He executes the `claimDutchRefund` for the first NFT (that he bought on a regular sale actually) and receives 9 ETH.
8. He executes the `payoutAction` and receives `canvas.totalQuantity * ds.canvasSystem[canvasId].dutchEndPrice` = `2 * 1 ETH` = 2 ETH.
9. He waits a year to go by and executes the `claimExpiredRefund`.
10. The amount that he can claim is calculated `ds.canvasSystem[canvasId].revenue - (canvas.totalQuantity * ds.canvasSystem[canvasId].dutchEndPrice)) - (ds.canvasSystem[canvasId].refundSum) / 2` = `11 ETH - (2 * 1 ETH) - 9 ETH / 2` = `4.5 ETH`

2.1.2 Example 2

1. User A creates a canvas and sets it up for a sale with `saleStart` set to current date, 7am and `saleEnd` current date, 8am. Price is 1 ETH
2. He buys an NFT that is part of the canvas for 1 ETH. He receives the 1 ETH back because he is buying from himself.
3. It is now 9am and the sale expired.
4. He updates canvas and sets new `saleStart` to 10am and `saleEnd` to 11am.

5. Now the `saleStarted` check is false. So user A can again execute a canvas update and set it as refundable dutch with end price of 10 ETH.
6. He buys another NFT through the refundable dutch for 10 ETH.
7. He executes the `payoutAction` and receives `canvas.totalQuantity * ds.canvasSystem[canvasId].dutchEndPrice` = `2 * 10 ETH` = 20 ETH.

In first example user started with 10 ETH, bought his own NFT which means he received the 10 ETH back. Bought his own NFT again for 1 ETH. Then he got paid 2 ETH from the contract. And in the end he claimed 4.5 ETH and ended up with 15.5 ETH.

In second example user started with 10 ETH, bought his own NFT for 1 ETH which he received back. Then he bought his own NFT for 10 ETH and got paid from the contract 20 ETH. So he ended up with 20 ETH. This is not taking into account the protocol fee, which can be 0 - 30 %.

2.2 Recommendation

Either prevent changes to `saleStart` (and consequently type of sale and price) after there has already been buying/selling done on that canvas, or prevent a canvas that was previously on a sale to be marked for refundable dutch. The first option would be preferred, because changing `saleStart` has many side effects and that opens up more possibilities for something to be vulnerable.

3 Claiming refunds on ongoing auctions make NFTs free

High Risk

In Funds.sol there is `claimDutchRefund` function that is used to claim the refund from a refundable dutch auction. Anyone who bought an NFT on this kind of auction, can execute this function any time. This can lead to people getting free NFTs.

3.1 Example

1. User A creates a canvas and sets it up for a refundable dutch. Starting price is 10 ETH, ending price is 5 ETH and total quantity is 10.
2. User B buys one NFT at the beginning of the auction and pays 10 ETH.
3. He can then immediately execute the `claimDutchRefund` function.
4. His refund will be calculated like this `purchaseTracker.spend - (purchaseTracker.quantity * (ds.canvasSystem[canvasId].dutchEndPrice))` = `10 ETH - (1 * 0)` = 10 ETH

So the user just bought an NFT for free. The `dutchEndPrice` is 0, because it gets set only when the amount of NFTs sold reaches the total quantity (in the above case 10). This example is not taking into account the protocol fee, which can be 0 - 30 %.

3.2 Recommendation

Since the reason for above exploit is that auction has not ended, meaning `ds.canvasSystem[canvasId].dutchEndPrice` is not set, you can simply check if it is zero and revert (`if(ds.canvasSystem[canvasId].dutchEndPrice == 0) revert`). Notice that you are only allowing claiming of funds after amount of NFTs sold reaches the total quantity (that is when `ds.canvasSystem[canvasId].dutchEndPrice` is set). This raises another issue that is described in Never ending refundable dutch. It includes a solution that takes into account both issues, and it would be recommended to use it.

4 Canvas one can be refundable dutch

High Risk

In Create.sol there are `createCanvas` and `updateCanvas` functions through which canvas can be created/updated. In both cases, someone can set a canvas to `refundableDutch` = `true` and `isOne` = `true`, although a refundable auction does not make sense for 1/1 canvases. Also, this can lead to people getting free NFTs.

4.1 Example

1. User A creates a canvas and marks it `refundableDutch` and `isOne`. Price is set to 10 ETH.
2. User B buys one NFT by executing `mint` function and pays 10 ETH. The sale is treated as a regular canvas 1/1 sale with the exception that ETH is not sent to the seller, but it is kept in the contract.
3. User B can then execute the `claimDutchRefund` function, because `refundableDutch` is `true`.
4. His refund will be calculated like this `purchaseTracker.spend - (purchaseTracker.quantity * (ds.canvasSystem[canvasId].dutchEndPrice))` = `10 ETH - (1 * 0)` = 10 ETH

`dutchEndPrice` is 0 because when canvas `isOne`, it never gets set.

4.2 Recommendation

Add a check in `createCanvas` and `updateCanvas` that prevents the above scenario. You can add for example: `if(canvas.refundableDutch && canvas.isOne) revert`.

5 Reserve Auction can be bypassed

High Risk

In Minting.sol there is `metaMint` function that is used for gasless minting. It states that it can only be used for free mints, as signature data does not handle transfer of funds from signer. However, that is not true. ERC20 tokens can be transferred from the signer if he gives the Canvas contract approval. That by itself is not problematic, since there is nothing wrong if minting with signature enables non-free mints also. The problem is that even if the canvas is on reserve auction, it executes `defaultMint` instead of `mintReserveAuction`.

5.1 Example

1. User A creates a canvas and sets it up for a reserve auction.
2. User B signs a mint request for that canvas and calls `metaMint`, which executes `defaultMint`.
3. User B buys the canvas at `salePrice`, which is the "starting bid price" of the auction.
4. Because there was no bid made, user C can still make a bid on the auction. But he will not be able to mint, because `oneMinted` is already `true`.

5.2 Recommendation

In `metaMint` add a check if `reserveAuction` is `true`. You can then disable reserve auctions by reverting, or you can execute `mintReserveAuction`.

```
Schema.MintRequest memory mintRequest = getMintRequest(data);

if (!canvasStorage().canvas[canvasId].reserveAuction) {
    defaultMint(msgSender, canvasId, 0, mintRequest);
} else {
    mintReserveAuction(msgSender, canvasId, mintRequest.selectedTraits);
    emit OneMinted(msgSender, canvasId, mintRequest);
}
```

6 Reserve Auction mints wrong canvas

High Risk

In Minting.sol there is `mintReserveAuction` function that is used for minting a canvas that was on reserve auction. On line 600 we can see that no mater if canvas is curated or not, the `mint` function is called on a contract with address `canvasOneAddress`. However, in case that a canvas is curated, `mint` should be called on a contract with address `canvasOneCuratedAddress`. The effect of this is not only that a token is minted in the wrong contract, but also wrong mappings of canvases to their traits, wrong `ds.canvasOne` and `ds.canvasOneCurated` mappings of token ID to canvas ID and wrong counters of canvasOne and canvasOneCurated.

```
586  uint256 tokenId;
587  if (ds.canvasSystem[canvasId].isCurated) {
588      ds.canvasOneCurated[ds.contractInfo.canvasOneCuratedCounter] = canvasId;
589      tokenId = ds.contractInfo.canvasOneCuratedCounter;
590      ds.customization[ds.contractInfo.canvasOneCuratedAddress][tokenId].selectedTraits =
591  selectedTraits;
592      unchecked { ds.contractInfo.canvasOneCuratedCounter++; }
593  } else {
594      ds.canvasOne[ds.contractInfo.canvasOneCounter] = canvasId;
595      tokenId = ds.contractInfo.canvasOneCounter;
596      ds.customization[ds.contractInfo.canvasOneAddress][tokenId].selectedTraits =
597  selectedTraits;
598      unchecked { ds.contractInfo.canvasOneCounter++; }
599  }
600
      ds.canvasSystem[canvasId].oneMinted = true;
      INFT(ds.contractInfo.canvasOneAddress).mint(msgSender, 1);
```

6.1 Recommendation

Change the code in a way that calls `mint` on `canvasOneCuratedAddress` when canvas is curated and `canvasOneAddress` when it is not.

```
586  uint256 tokenId;
587  address canvasAddress;
588  if (ds.canvasSystem[canvasId].isCurated) {
589      tokenId = ++ds.contractInfo.canvasOneCuratedCounter;
590      canvasAddress = ds.contractInfo.canvasOneCuratedAddress;
591      ds.canvasOneCurated[tokenId] = canvasId;
592  } else {
593      tokenId = ++ds.contractInfo.canvasOneCounter;
594      canvasAddress = ds.contractInfo.canvasOneAddress;
595      ds.canvasOne[tokenId] = canvasId;
596  }
597
598  ds.customization[canvasAddress][tokenId].selectedTraits = selectedTraits;
599  ds.canvasSystem[canvasId].oneMinted = true;
600  INFT(canvasAddress).mint(msgSender, 1);
```

*This example also includes fixes regarding some gas inefficiencies and a bug from Wrong token ID in mintReserveAuction.

7 Wrong token ID in mintReserveAuction

High Risk

In Minting.sol there is `mintReserveAuction` function that is used for minting a canvas that was on reserve auction. On lines 588 - 591 and 593 - 596 we can see that the `tokenId` is assigned the value of count of token IDs and only after that, the count of token IDs is incremented. This would be okay if token IDs would start at 0. However, they start at 1, which can be seen in CanvasCollection.sol. This means that the traits and canvas ID will not match the token. It will also override the traits and canvas ID of the last minted token.

```
586  uint256 tokenId;
587  if (ds.canvasSystem[canvasId].isCurated) {
588      ds.canvasOneCurated[ds.contractInfo.canvasOneCuratedCounter] = canvasId;
589      tokenId = ds.contractInfo.canvasOneCuratedCounter;
590      ds.customization[ds.contractInfo.canvasOneCuratedAddress][tokenId].selectedTraits =
591  selectedTraits;
592      unchecked { ds.contractInfo.canvasOneCuratedCounter++; }
593  } else {
594      ds.canvasOne[ds.contractInfo.canvasOneCounter] = canvasId;
595      tokenId = ds.contractInfo.canvasOneCounter;
596      ds.customization[ds.contractInfo.canvasOneAddress][tokenId].selectedTraits =
597  selectedTraits;
         unchecked { ds.contractInfo.canvasOneCounter++; }
     }
```

7.1 Recommendation

Change the code so that the token ID is equal to last token ID + 1. The solution was already included in ReserveAuction mints wrong canvas.

8 Refundable dutch with zero dutchEndTime enables theft of funds

High Risk

In Create.sol there are `createCanvas` and `updateCanvas` functions through which canvas can be created/updated. In both cases, someone can set a canvas to be refundable dutch, even though it is not a dutch auction (`refundableDutch` = `true` and `dutchEndTime` = `0`). This enables stealing of funds from the contract. Canvas creator has the ability to change `saleToken` as stated in Change of saleToken can result in wrong revenue. So he can use this in combination with `payoutAuction` to steal any token from the contract.

8.1 Example

1. User A creates a canvas and sets `refundableDutch` = `true` and `dutchEndTime` = `0`. The price is 1 USDT and total quantity is 10.
2. He buys 10 NFTs and pays 10 USDT.
3. He then executes a canvas update and sets `saleToken` to ETH, which he can because `dutchEndTime` is 0.
4. He can then execute the `payoutAuction` function.
5. His revenue will be calculated like this `canvas.totalQuantity.mul(ds.canvasSystem[canvasId].dutchEndPrice)` = `10 * 1` = 10
6. He will receive 10 ETH

He paid 10 USDT and received 10 ETH (which is at the time worth more than 1000x that).

8.2 Recommendation

Consider adding a check in `createCanvas` and `updateCanvas` that prevents the above scenario. You can add for example: `if(canvas.refundableDutch && canvas.dutchEndTime == 0) revert`.

II. Medium Risk

9 Change of saleToken can result in wrong revenue

> ⚠ **Medium Risk**

In Create.sol there is `updateCanvas` function that is used to update a canvas. You can also update `saleToken` as long as it is not on reserve auction or dutch auction. If it is on a regular sale it can be updated. In case someone has already purchased an NFT, which is part of a canvas, and then `saleToken` is changed, the revenue will be wrong. Because there is no way to know how much was sold for which `saleToken`.

9.1 Example

1. User A creates a canvas and sets it up for a regular sale. `saleToken` is ETH (address(0)) and price is 1e18.
2. User B mints one NFT that is part of that canvas and pays 1 ETH.
3. Revenue for that canvas updates to `ds.canvasSystem[canvasId].revenue += totalPrice` -> 1e18.
4. User A executes a canvas update. He sets `saleToken` to USDT and price to 100e6 (USDT has 6 decimals).
5. User C mints one NFT that is part of that canvas and pays 100 USDT.
6. Revenue for that canvas updates to `ds.canvasSystem[canvasId].revenue += totalPrice` -> 1e18 + 100e6 = 10000000001e8.

Now the revenue is 10000000001e8, but that does not represent how much ETH was earned or how much USDT was earned.

9.2 Recommendation

Prevent a change of `saleToken` if any kind of sale is started. You can add this line: `if(saleStarted && canvas.saleToken != update.saleToken) revert`

10 Collection to canvas mapping does not work for canvas one

Medium Risk

There are multiple instances where canvas ID is being read from `collectionToCanvas` mapping. We can see that the only time this relation is set is in `approveCanvas` function in Core.sol and that it is set only for canvases that don't have `isOne` = `true`. That means, that for 1/1 canvases, ID must be retrieved another way. This makes sense, because all the 1/1 canvases are part of the same collection/contract. But in URI.sol there are 3 instances where that is ignored. This can result in wrong traits being included in the tokenURI for 1/1 canvases.

1. Line 87 - `getRefUri`
2. Line 244 - `getSelectedTraits`
3. Line 411 - `getTraitsAndParams`

This is how canvas ID is retrieved in the above cases:

```
uint256 canvasId = ds.collectionToCanvas[collectionAddress];
```

This is how canvas ID is retrieved the right way in `tokenURI` function:

```
uint256 canvasId;
if (collectionAddress == ds.contractInfo.canvasOneAddress) canvasId = ds.canvasOne[tokenId];
else if (collectionAddress == ds.contractInfo.canvasOneCuratedAddress) canvasId =
ds.canvasOneCurated[tokenId];
else canvasId = ds.collectionToCanvas[collectionAddress];
```

10.1 Recommendation

Since you are already getting canvas ID the right way in `tokenURI` function and all the other functions are only called within it, just pass the canvas ID as an argument. This way the right ID will be used and also gas consumption will be lower, since there will be less reading from storage.

11 Canvas can have unlimited supply even in case of dutch auction

Medium Risk

In Create.sol there are `createCanvas` and `updateCanvas` functions through which canvas can be created/updated. In both cases, someone can set a canvas to have `totalQuantity` = `0` (which is considered unlimited) and be a dutch auction. Which means that the number of minted tokens will never equal the `totalQuantity` and therefore the line 539 in Minting.sol will never be executed. This means that `canvasSystem[canvasId].dutchEndPrice` will not be saved, and the seller will not be able to withdraw the funds if it is a refundable dutch. It also does not make sense to have a dutch auction that has unlimited supply. Since all the buyers can wait for the price to reach the end price without the risk of not getting a mint, which is the point of a dutch auction.

```
535  if (ds.canvas[canvasId].totalQuantity > 0 &&
536      tokenIdCounter.add(quantity) > ds.canvas[canvasId].totalQuantity)
537  revert SoldOut();
538  else if (tokenIdCounter.add(quantity) == ds.canvas[canvasId].totalQuantity) {
539      ds.canvasSystem[canvasId].dutchEndPrice = price;
540  }
```

11.1 Recommendation

Add a check that prevents `totalQuantity` to be `0` in case of a dutch auction. For example: `if(canvas.totalQuantity == 0 && canvas.dutchEndTime != 0) revert`.

12 VRF subscription cannot be cancelled

Medium Risk

In VRF.sol there is a `cancelSubscription` function. It is used to cancel a subscription of a canvas to VRF. But the code has a bug that prevents the actual subscription from being canceled. The mapping `canvasIdToSubscription[canvasId]` is cleared first and after that it calls `cancelSubscription` on `vrfCoordinator`. In the parameters of `cancelSubscription` it forwards the `canvasIdToSubscription[canvasId]`, which is already cleared and therefore does not match the subscription that should be canceled.

```
function cancelSubscription(uint256 canvasId) external {
    address admin = ICanvas(canvasAddress).getVRFHelper(canvasId).admin;
    if (msg.sender != admin) revert NoPermission();

    canvasIdToSubscriptionId[canvasId] = 0;
    vrfCoordinator.cancelSubscription(canvasIdToSubscriptionId[canvasId], admin);
    ICanvas(canvasAddress).setVrfSubscription(canvasId, 0);
}
```

12.1 Recommendation

Either cache the subscription ID before deleting it from the mapping and use the cached ID in `vrfCoordinator.cancelSubscription`, or execute the call first and delete the subscription ID after.

```
function cancelSubscription(uint256 canvasId) external {
    address admin = ICanvas(canvasAddress).getVRFHelper(canvasId).admin;
    if (msg.sender != admin) revert NoPermission();

    vrfCoordinator.cancelSubscription(canvasIdToSubscriptionId[canvasId], admin);
    delete canvasIdToSubscriptionId[canvasId];
    ICanvas(canvasAddress).setVrfSubscription(canvasId, 0);
}
```

13 Never ending refundable dutch

Medium Risk

A canvas can be put on refundable dutch auction. When setting up an auction, the seller defines `totalQuantity`. The auction is considered to be over when the amount of tokens minted matches the `canvas.totalQuantity`. However, that value can be set so high (intentionally or unintentionally) that the auction never ends. Or there simply is not enough interested buyers. Since the auction is considered as ongoing, funds cannot be withdrawn from contract.

13.1 Recommendation

Allow withdrawing even if `totalQuantity` is not reached, as long as `dutchEndTime` is. In that case you can be sure that the end price will equal `canvas.dutchEndPrice`. So you can take `canvas.dutchEndPrice` instead of `ds.canvasSystem[canvasId].dutchEndPrice` (which is set when last token is minted) to calculate how much was earned so far and how much can be refunded. Notice that in such case, it would also be important to track how much was paid out so far. It would be recommended to limit how long an auction can last, otherwise if `canvas.dutchEndTime` is set too far in the future, this solution will not help.

In `payoutAction` you can do it like this:

```
if (canvas.refundableDutch) {
    uint endPrice = ds.canvasSystem[canvasId].dutchEndPrice
    if(endPrice == 0) {
        if(canvas.dutchEndTime > block.timestamp) revert SaleOngoing();
        endPrice = canvas.dutchEndPrice;
    }

    uint tokenIdCounter = ds.canvasSystem[canvasId].tokenIdCounter;
    revenue = (tokenIdCounter - ds.canvasSystem[canvasId].payoutCounter) * endPrice;

    ds.canvasSystem[canvasId].payoutCounter = tokenIdCounter;
    if(tokenIdCounter == canvas.totalQuantity) {
        ds.canvasSystem[canvasId].auctionPayout = true;
    }
}
```

Notice that `ds.canvasSystem[canvasId].payoutCounter` is added.

In `claimRefundableDutch` you can do it like this:

```
uint endPrice = ds.canvasSystem[canvasId].dutchEndPrice

if(endPrice == 0) {
    if(canvas.dutchEndTime > block.timestamp) revert SaleOngoing();
    endPrice = canvas.dutchEndPrice;
}

uint256 refund = purchaseTracker.spend - (purchaseTracker.quantity * endPrice);
```

III. Low Risk

14 Lack of selectedTraits validation in mintReserveAuction

Low Risk

In Minting.sol there is `mintReserveAuction` function used to mint a canvas that was on reserve auction. It receives an uint array `selectedTraits` and saves it to storage for the canvas that is being minted. But the traits validity is never checked like it is in `defaultMint`. There can be more traits than are defined for the selected canvas. Also, the traits values can be out of range.

14.1 Recommendation

In `mintReserveAuction` check that selected traits match the traits definition for selected canvas, like you do in `defaultMint` with `handleTraits` function.

15 Lack of validation for time settings

> **Low Risk**

In Create.sol there are `createCanvas` and `updateCanvas` functions through which canvas can be created/updated. In both cases, there is a lack of validation for time related settings.

- `dutchEndTime` can be set to less than current timestamp and less than `saleStart`, which will result in price always being the same.
- Both `saleStart` and `saleEnd` can be less than current timestamp.
- Both `saleStart` and `saleEnd` can be as far in the future as possible.

15.1 Recommendation

Add a check in `createCanvas` and `updateCanvas` that prevents the above and limits the duration of sale. Consider using constants to represent minimum and maximum duration for auction/sale.

```
if(canvas.dutchEndTime != 0 && (canvas.dutchEndTime - canvas.saleStart > MAX_DUTCH_DURATION ||
canvas.dutchEndTime - canvas.saleStart < MIN_DUTCH_DURATION)) revert
if(canvas.saleStart != 0 && (canvas.saleStart < block.timestamp || canvas.saleEnd -
canvas.saleStart > MAX_SALE_DURATION || canvas.saleEnd - canvas.saleStart < MIN_SALE_DURATION))
revert
```

16 Canvas that is not one can have reserveAuction flag

Low Risk

Only canvases that are 1/1 can be put on reserve auction. However, in Create.sol there is a `createCanvas` function that receives a `Schema.Canvas` struct. This struct is then saved to storage as a new canvas. Therefore, someone can create a canvas with `isOne` marked `false` and `reserveAuction` marked `true`. This canvas will be unusable, because you can't make a bid on it (it reverts because it is not 1/1), and you can't buy it either.*

There is also another way to achieve such setup. In `updateCanvas` function, when changing `reserveAuction` property, it is checked if `isOne` is `false`. In that case it does not allow change of `reserveAuction` to `true`. But the opposite scenario is never considered. Canvas can have `reserveAuction` previously marked `true`, and then change `isOne` to `false` in the next update.

16.1 Recommendation

Add a check in `createCanvas` if `reserveAuction` is `true` and `isOne` is `false` and revert. You can add this line for example: `if(canvas.reserveAuction && !canvas.isOne) revert InvalidSetup();`.

In `updateCanvas` you can check if `reserveAuction` is `true` where updating `isOne` property to `false` and revert like in the example bellow. It would be even better to place all revert cases at the top of the function. That way more gas would be returned when reverting.

```
157  if (canvas.isOne != update.isOne) {
158      if(!update.isOne && canvas.reserveAuction) revert InvalidSetup();
159      canvas.isOne = update.isOne;
160  }
```

*You actually can buy such canvas through `metaMint`, because of an issue that is described in ReserveAuction can be bypassed

17 Canvas can be a reserve auction and a dutch auction

Low Risk

In Create.sol there are `createCanvas` and `updateCanvas` functions through which canvas can be created/updated. In both cases, someone can set a canvas to have both a reserve auction, dutch auction and refundable dutch, although there is no support for this kind of setup in the rest of the code. Such canvas will be treated as reserve auction.

17.1 Recommendation

Add a check that prevents `reserveAuction` to be `true` at the same time that `refundableDutch` is `true` or `dutchEndTime` is set.

18 externalUrlSlash cannot be updated

Low Risk

In `Canvas` struct, there is a boolean `externalUrlSlash`. The value can be set when canvas is created, but there is no way to change it. Maybe this is intended, but based on that all other values can be updated, we assume that it should also be possible to update `externalUrlSlash`.

18.1 Recommendation

Consider adding a way to update the `externalUrlSlash`. You can add `canvas.externalUrlSlash = update.externalUrlSlash;` to the end of `updateCanvas` function.

19 There can be more purchaseIdentifiers than tokens minted

Low Risk

Whenever someone is minting through the `defaultMint` function, they can specify `purchaseIdentifiers` which is a list of custom identifiers by minter. On lines 553 - 556 in Minting.sol we can see that each of these identifiers is assigned to one of the tokens that he is minting. However, the minter can specify more identifiers than the amount of tokens he is minting. This means that he can assign an identifier to a token that is not yet minted.

```
for (uint256 i = 0; i < mintRequest.purchaseIdentifiers.length;) {
    ds.customization[collectionAddress][tokenId + i].purchaseIdentifier =
mintRequest.purchaseIdentifiers[i];
    unchecked { i++; }
}
```

19.1 Recommendation

Consider adding a check that the length of `mintRequest.purchaseIdentifiers` matches the `mintRequest.quantity`, otherwise revert. This way minter can specify only identifiers for the tokens he is minting.

20 Same trait can appear twice in URI

Low Risk

In URI.sol, there is `getTraitsAndParams` function that combines selected traits with static or random traits. In case that some traits are static and some are selected, `getStaticTraits` will be executed first to get the static traits and `getSelectedTraits` will be executed after to get the selected traits and combine them. In `getSelectedTraits` a trait will be assigned only if it is `selectable` and the selected value is not zero (meaning trait is not selected). In `getStaticTraits` all possible traits will be assigned. No matter if they are `selectable` or not. This results in the traits that are selected appearing twice. Once with the selected value and once with the default (first in array) value.

20.1 Recommendation

Add an if statement in `getStaticTraits` function, that checks if trait is `selectable` and only add the ones that aren't to `encodedTraits` string. If the intended use case of the selected trait zero value is that it is not displayed, then also add a check if the selected value is zero.

```
for (uint256 i = 0; i < canvasTraits.length;) {
    if (!canvasTraits[i].selectable) {
        encodedTraits = string(abi.encodePacked(
        encodedTraits,
        '{"trait_type":"',
        canvasTraits[i].trait,
        '", "value":"',
        canvasTraits[i].values[0],
        '"},')
        );
    }
    unchecked { i++; }
}
```

21 Partner must send more ETH than needed

Low Risk

Each canvas can have "partners". These are addresses that have a discount on tokens that they purchase. In Minting.sol there is `handleFunds` function that handles the distribution of funds for a given sale. We can see on line 463, that if the buyer is a partner and the sale is in ETH, he is returned the amount of ETH that he is discounted for. This means that if a partner has 30% discount, he must send 1 ETH when buying even if it costs him only 0.7 ETH. He will receive the 0.3 ETH back, but it can be inconvenient.

```
460  if (canvas.saleToken == address(0)) {
461      if (!canvas.refundableDutch) {
462          if (protocolFee > 0) sendETH(ds.contractInfo.protocolFeeRecipient, protocolFee);
463          if (partnerFee > 0) sendETH(msgSender, partnerFee);
464          sendETH(ds.canvas[canvasId].feeRecipient, remainder);
465      }
466
467      uint256 excessAmount = ethValue.sub(totalPrice);
468      if (excessAmount > 0) sendETH(msgSender, excessAmount);
469  }
```

21.1 Recommendation

Allow partners to send the amount of ETH needed for purchase without requiring the full price. Only send back ETH that exceeds the required amount.

```
460  if (canvas.saleToken == address(0)) {
461      uint256 excessAmount = ethValue - protocolFee - remainder;
462
463      if (!canvas.refundableDutch) {
464          if (protocolFee > 0) sendETH(ds.contractInfo.protocolFeeRecipient, protocolFee);
465          sendETH(ds.canvas[canvasId].feeRecipient, remainder);
466      } else {
467          excessAmount -= partnerFee;
468      }
469
470      if (excessAmount > 0) sendETH(msgSender, excessAmount);
471  }
```

22 Lack of subscription validation in VRF

Low Risk

In VRF.sol there are multiple functions that get a subscription ID from the `canvasIdToSubscription` mapping and execute some actions related to that subscription. But they lack validating that the subscription for selected canvas actually exists. These functions are: `cancelSubscription`, `onTokenTransfer`, `resetVRF` and `requestVRF`

22.1 Recommendation

Consider adding a check that `canvasIdToSubscription` is not equal to `0`. `if(canvasIdToSubscription[canvasId] == 0) revert`.

23 updatePurchaseIds can overwrite purchaseIdentifiers

Low Risk

Buyers can set a purchase ID for each token when they buy it. But these identifiers can be overwritten by canvas admin. In Customize.sol there is `updatePurchaseIds` function that receives an array of token IDs and purchase IDs. These purchase IDs then get assigned to tokens given in the token IDs array. This allows the canvas admin to change purchase IDs that were previously set by buyers. This could break some external functionality that would be dependent on purchase IDs. Especially because there is no event emitted that would signal a change. Furthermore, there is no validation, that given token IDs actually exist.

Similar problem exists in `updateChipIds` function. But we assume that this function is part of PBT protocol, which is not implemented yet, and therefore its functionality is not complete. It is important to notice that in this function there is no check that each chip belongs to exactly one address and vice versa.

23.1 Recommendation

You can add an event to signal a change in purchase IDs and validate that given token IDs exist.

24 Storage changes after external calls

Low Risk

There are multiple instances where storage is updated after external calls are made. This opens up possibility for reentrancy related exploits. These functions are wrapped in `nonReentrant` modifier which fixes that to some degree. However, the modifier prevents only the functions that use it from being entered, where other functions could still be entered (`updateCanvas` for example). Although we did not notice an opportunity to exploit that, it would still be recommended to follow the "check effects interactions" pattern to make the code more secure.

24.1 Recommendation

Make storage changes before executing external calls.

Minting.sol

- Move line 484 before the transfer of ETH (L459 for example).
- In `defaultMint` do `fulfillMint` first and `handleFunds` second.

Funds.sol

- Move line 104 before the transfer of ETH (L81 for example).
- Move line 173 before the transfer of ETH (L154 for example).
- Move line 228 before the transfer of ETH (L209 for example).
- Move line 286 before the transfer of ETH (L264 for example).

25 Looping through big arrays can cause failure

Low Risk

There is a lot of looping through arrays in this code base. Since there are no bounds to sizes of these arrays, they can grow indefinitely. This can cause the functions that are looping through them to become too expensive or even fail.

Functions that could break if arrays grow too large:

Customize.sol

- `setTokenRefs`

Minting.sol

- `processMintPass`
- `handleTraits`
- `getPrice`

URI.sol

- `getRefUri`
- `getStaticTraits`
- `getSelectedTraits`
- `getRandomTraits`
- `getTraitsAndParams`

25.1 Recommendation

Do extensive tests on all the functions that are looping through arrays and test the limits.

26 Loops do not break when possible

Low Risk

Looping through arrays can be expensive, that is why it is important to break the loop as soon as possible. There are multiple cases in the code base, where loops go through whole array even though it already has the information that it needs.

26.1 Examples

26.1.1 Example 1

`verifyTraits` in Create.sol goes through whole `canvasTraits` array, although the loop could be exited whenever `valid` is set to `false`.

```
278  bool valid = true;
279  for (uint256 i = 0; i < canvasTraits.length;) {
280      if (canvasTraits[i].traitsChance > 10_000) valid = false;
281      if (canvasTraits[i].values.length != canvasTraits[i].chancesOrQuantity.length) valid =
282  false;
283      unchecked { i++; }
284  }
285
     if (!valid) revert InvalidTraits();
```

26.1.2 Solution 1

```
278  uint256 len = canvasTraits.length;
279  for (uint256 i = 0; i < len;) {
280      if(canvasTraits[i].traitsChance > 10_000 || canvasTraits[i].values.length !=
281  canvasTraits[i].chancesOrQuantity.length)
282          revert InvalidTraits();
283      unchecked { i++; }
     }
```

26.1.3 Example 2

`setTokenRefs` in Customize.sol goes through whole `ds.allowedRefs[canvasId]` array, although the loop could be exited when `allowed` is set to `true`.

```
110  bool allowed = false;
111  for (uint256 j = 0; j < ds.allowedRefs[canvasId].length;) {
112      if (ds.allowedRefs[canvasId][j] == refs[i].tokenAddress) allowed = true;
113      unchecked { j++; }
114  }
115  if (!allowed) revert InvalidReference();
```

26.1.4 Solution 2

```
bool allowed = false;
uint256 len = ds.allowedRefs[canvasId].length;
for (uint256 j = 0; j < len) {
    if (ds.allowedRefs[canvasId][j] == refs[i].tokenAddress) {
        allowed = true;
        break;
    }
    unchecked { j++; }
}
if (!allowed) revert InvalidReference();
```

27 Poor sequence of actions in updateCanvas

Low Risk

In Create.sol the `updateCanvas` function is used to update a canvas. When updating, it stores `isImmutable` property first, then it reads it from storage to check if it is immutable, and only if it is not, other properties can be updated. This can cause you to lock wrongly configured canvas.

27.1 Example

1. User A has a canvas with `feeBps` equal to `100`.
2. He wants to change `feeBps` to `200` and lock the canvas.
3. He executes `updateCanvas` with `feeBps` = `200` and `isImmutable` = `true`.
4. The `isImmutable` property gets stored first, therefore the canvas is now immutable.
5. Because canvas is immutable, `feeBps` is not saved.

User A locked his canvas, but `feeBps` is still `100`. He can't change `feeBps`, unless the DAO unlocks his canvas.

27.2 Recommendation

Move part of the code, that changes canvas to immutable, to the end of the function. This way you are locking the canvas after making all the other changes.

IV. Informational

28 Same validation is being executed in Keeper and VRF

Informational

In Keeper.sol there is a `performUpkeep` function, which makes a call to `requestVRF` function that is in VRF.sol. Both of these functions perform the same check, which can be seen in `performUpkeep` on lines 34 - 40, and in `requestVRF` on lines 102 - 107. There is no need to perform these checks in both of the functions, because of their dependence on each other. If `requestVRF` fails, `performUpkeep` fails too.

```
32  function performUpkeep(bytes calldata performData) external override {
33      uint256 canvasId = abi.decode(performData, (uint256));
34      Schema.VRFHelper memory info = ICanvas(canvasAddress).getVRFHelper(canvasId);
35
36      if (
37        !info.vrfPending &&
38        info.tokenIdCounter - info.randomizedTraitsCounter > 0 &&
39        info.vrfLastRunTimestamp + (info.vrfMinuteInterval * 60) < block.timestamp
40      ) {
41        CanvasVRF(vrfAddress).requestVRF(canvasId);
42      }
43  }
```

```
101   function requestVRF(uint256 canvasId) external {
102       Schema.VRFHelper memory info = ICanvas(canvasAddress).getVRFHelper(canvasId);
103
104       if (info.vrfPending ||
105           info.tokenIdCounter - info.randomizedTraitsCounter == 0 ||
106           info.vrfLastRunTimestamp + (info.vrfMinuteInterval * 60) > block.timestamp)
107         revert VRFNotNeeded();
108
109       uint256 requestId = vrfCoordinator.requestRandomWords(keyHash,
110   canvasIdToSubscriptionId[canvasId], 7, 300000, 1);
111
112       vrfToCanvasId[requestId] = canvasId;
113       ICanvas(canvasAddress).setVrfResult(canvasId, 0, "requested");
      }
```

28.1 Recommendation

Remove the validation from `performUpkeep` function and just execute the call to `requestVRF`.

```
32  function performUpkeep(bytes calldata performData) external override {
33      uint256 canvasId = abi.decode(performData, (uint256));
34      CanvasVRF(vrfAddress).requestVRF(canvasId);
35  }
```

Or if you do not want `performUpkeep` to revert you can do it like this:

```
32  function performUpkeep(bytes calldata performData) external override {
33      uint256 canvasId = abi.decode(performData, (uint256));
```

```
    vrfAddress.call(
      abi.encodeWithSignature('requestVRF(uint256)'),
      canvasId
    ));
}
```

29 Events don't describe changes clearly

Informational

Events are used to communicate changes in state to front ends. In this code base we can see a lot of events that do not give clear information on what has changed. In such cases, the front end or a back end service that keeps database up to date, has to figure out what has changed using other methods that make the process more complex.

29.1 Example

Event `NewCurator()` is emitted when curator is changed. But it does not give information on whom the new curator is. It would be good to add the new curator's address to the event.

There are more events like this one that do not give information on what has changed:

- `ContractUpdate()` - no information on what was updated.
- `NewDao()` - no information on what the new dao address is.
- `NewAdmin()` - no information on what the new admin address is.
- `CanvasCreate(uint256 indexed canvasId)` - no information on what kind of canvas was created.
- `CanvasUpdate(uint256 indexed canvasId)` - no information on what was updated.
- `CanvasAsset(uint256 indexed canvasId)` - no information on what the new asset version is, new `thumbnailUri` and `baseUri`.
- `CanvasCreators(uint256 indexed canvasId)` - no information on new creators addresses.
- `CanvasRefs(uint256 indexed canvasId)` - no information on new refs addresses.
- `OneMinted(address indexed msgSender, uint256 indexed canvasId, Schema.MintRequest indexed quantity, uint256[] selectedTraits)` - no information about the cost.
- `ApproveCanvas(uint256 indexed canvasId)` - is used when curating and approving. No information on whether it is approved or curated.

29.2 Recommendation

Add information to events, that would enable a back end service to reconstruct the contracts state just by looking at events. In some cases, you can make separate events to represent different changes. For example in `createCanvasOne` function you could change `ContractUpdate` event to something like `CanvasOneCreated(address canvasOne, address canvasOneCurated)`.

30 Logging timestamp in events is redundant

Informational

Event `AuctionBid` includes a parameter for timestamp, which represents the time when the bid was made and event was emitted. However, for each event it is known part of which block it is, and each block has a known timestamp. Therefore, it is not needed to have a timestamp also included in the event parameters. But this might be intended by developers for convenience or similar reasons.

30.1 Recommendation

Consider removing the timestamp from `AuctionBid` event.

31 Event definitions in Canvas are redundant

Informational

There are multiple events defined in Canvas.sol, but none of them are emitted in this contract. These are the same events that are emitted in libraries, which are called from this contract. But the libraries also include definitions of the events they emit, and they do not require the contract executing the call (`Canvas.sol`) to have them too. Therefore, the definitions in `Canvas.sol` are unnecessary.

31.1 Recommendation

Remove events from Canvas.sol.

32 Storing and reading contract address uses more gas

Informational

The contract address is stored to `ds.contractInfo.canvasStorage`. It is then read from storage multiple times in libraries. Reading from storage costs 2100 gas, whereas using `address(this)` uses just 2 gas. It is stated, that the address is stored, so libraries can use it. Since calls to libraries are essentially executed as `delegateCall`, using `address(this)` in a library contract will give the address of the contract which is executing the code. That address is the same as `ds.contractInfo.canvasStorage`.

32.1 Recommendation

Remove the `ds.contractInfo.canvasStorage` state variable and use `address(this)` where you need the contract address instead.

33 Unused state variables

Informational

There are multiple state variables that are never used within the contracts. Some of these might be stored and read from the contract, but not actually used in any of the functionality. This might be intentional, but we want to point them out anyway.

- `ds.contractInfo.linkTokenAddress`
- `ds.contractInfo.keeperRegistrarAddress`
- `ds.contractInfo.keeperAddress`
- `ds.contractInfo.canvasOpenCounter`
- `ds.canvas[id].isPBT`
- `ds.customization[contractAddress][tokenId].chipIdentifier`
- `ds.chipToTokenId`

33.1 Recommendation

Remove the state variables that are not needed.

34 Redundant usage of msg.sender and msg.value

Informational

In Canvas.sol there are multiple cases where libraries are called with `msg.sender` and `msg.value` forwarded as arguments. Since a call to a library is a `delegateCall`, there is no need for that. Reading `msg.sender` or `msg.value` within the library will give the same result as reading it in the contract that is calling the library.

34.1 Example

Code in Canvas.sol

```
function mint(
    uint256 canvasId,
    bytes calldata data
  ) external payable {
    Minting.mint(
      msg.sender,
      msg.value,
      canvasId,
      data
    );
  }
```

Code in Minting.sol

```
function mint(
    address msgSender,
    uint256 ethValue,
    uint256 canvasId,
    bytes calldata data
  ) external nonReentrant {
    Schema.Storage storage ds = canvasStorage();
    Schema.Canvas storage canvas = ds.canvas[canvasId];

    Schema.MintRequest memory mintRequest = getMintRequest(data);

    if (!canvas.reserveAuction) defaultMint(msgSender, canvasId, ethValue, mintRequest);
    else {
      mintReserveAuction(msgSender, canvasId, mintRequest.selectedTraits);
      emit OneMinted(msgSender, canvasId, mintRequest);
    }
  }
```

34.2 Recommendation

Remove `msg.sender` and `msg.value` forwarding. Use `msg.sender` and `msg.value` directly where needed in libraries.

35 Variable shadowing

Informational

There are two cases of variable shadowing.

35.1 Case 1

In VRF.sol there is `vrfCoordinator` state variable that shadows same name variable inherited from `VRFConsumerBaseV2`. Both of these store the same address.

35.1.1 Recommendation

Remove the `vrfCoordinator` variable from `CanvasVRF`. Since the same address is stored in `VRFConsumerBaseV2`, which is inherited by `CanvasVRF`, there is no need to store it. The variable defined in `CanvasVRF` was public and was of type `VRFCoordinatorV2Interface`, while the one defined in `VRFConsumerBaseV2` is private and of type `address`, meaning the code will need additional changes. You need to add a getter function if you want to be able to get the address from contract. Additionally, when making calls to `vrfCoordinator` you will have to use it like this: `VRFCoordinatorV2Interface(vrfCoordinator)`. The gas consumption when using it like this is the same, since the difference of defining address as contract type and casting an address to a contract type is purely visual. Furthermore, this solution should use less gas because the `vrfCoordinator` defined in `VRFConsumerBaseV2` is immutable, meaning it becomes a part of the code on deployment and does not require reading from storage.

35.2 Case 2

In CanvasCollection.sol there is `initialize` function that takes `name` and `symbol` as arguments. The names of these two arguments are the same as the names of functions defined in `ERC721AUpgradable`, which is inherited by `CanvasCollection`.

35.2.1 Recommendation

Rename `name` and `symbol` in `initialize` function to `_name` and `_symbol`.

36 Initialize can be external

Informational

In CanvasCollection.sol there is `initialize` function that is defined as public. Public functions can be called from within the contract and externally. However, `initialize` is never called from within the contract, and therefore it would be clearer if it would be defined external.

36.1 Recommendation

Change `initialize` function from public to external.

37 checkTransfer is always true

Informational

In View.sol there is a `checkTransfer` function that returns a bool value. However, this value is always `true`, otherwise the function reverts. In CanvasCollection.sol there are several instances where `checkTransfer` is called. Each time it is done in an if statement to verify that the return value is `true`. However since the result is always `true` (otherwise it reverts), the check is unnecessary.

37.1 Example

If we look at this code form View.sol, we can see that `checkTransfer` returns `!canvas.soulbound`. But that will always result in `true`, because we can see few lines above, that if `canvas.soulbound` is `true`, `revert NonTransferable()` will be executed.

```
363   function checkTransfer(
364       address collectionAddress,
365       address from,
366       address to,
367       uint256 tokenId
368     ) external returns (
369       bool
370     ) {
371       Schema.Storage storage ds = canvasStorage();
372
373       uint256 canvasId;
374       if (collectionAddress == ds.contractInfo.canvasOneAddress) canvasId =
375   ds.canvasOne[tokenId];
376       else if (collectionAddress == ds.contractInfo.canvasOneCuratedAddress) canvasId =
377   ds.canvasOneCurated[tokenId];
378       else canvasId = ds.collectionToCanvas[collectionAddress];
379
380       Schema.Canvas storage canvas = ds.canvas[canvasId];
381       if (!ds.canvasSystem[canvasId].approved) revert InvalidToken();
382       if (canvas.soulbound) revert NonTransferable();
383
384       emit Transfer(collectionAddress, from, to, tokenId);
          return !canvas.soulbound;
        }
```

Taking a look at code of CanvasCollection.sol, we can see the if statement that is unnecessary.

```
if (ICanvas(canvasAddress).checkTransfer(from, to, tokenId)) {
  super.transferFrom(from, to, tokenId);
}
```

37.2 Recommendation

In View.sol remove the return statement from `checkTransfer`, because there is no upside to returning a boolean that is always `true`.

```
363   function checkTransfer(
364       address collectionAddress,
365       address from,
366       address to,
367       uint256 tokenId
368     ) external {
369       Schema.Storage storage ds = canvasStorage();
370
371       uint256 canvasId;
372       if (collectionAddress == ds.contractInfo.canvasOneAddress) canvasId =
373   ds.canvasOne[tokenId];
374       else if (collectionAddress == ds.contractInfo.canvasOneCuratedAddress) canvasId =
375   ds.canvasOneCurated[tokenId];
376       else canvasId = ds.collectionToCanvas[collectionAddress];
377
378       Schema.Canvas storage canvas = ds.canvas[canvasId];
379       if (!ds.canvasSystem[canvasId].approved) revert InvalidToken();
380       if (canvas.soulbound) revert NonTransferable();
381
          emit Transfer(collectionAddress, from, to, tokenId);
        }
```

Remove the if statements in CanvasCollection.sol and just call the `checkTransfer` function.

```
ICanvas(canvasAddress).checkTransfer(from, to, tokenId)
super.transferFrom(from, to, tokenId);
```

38 Unused code in Customize.sol

Informational

In Customize.sol there is a `getCustomization` function that is empty and was probably left there by accident.

38.1 Recommendation

Remove the `getCustomization` function.

39 Redundant ERC20 approval

Informational

In Funds.sol there are 3 cases (L92, L211, L267) where the contract is approving itself for ERC20 transfer. But a contract does not need approval to transfer its own funds. Therefore, these approvals are not needed.

39.1 Recommendation

Remove ERC20 approvals (`IERC20(canvas.saleToken).safeApprove`) on lines: L92, L211, L267.

40 Redundant ERC20 balance check

Informational

In Minting.sol on line 470 it is checked if the sender has enough ERC20 tokens to execute the transfer. If he doesn't, it reverts. However, if the sender would not have enough balance, the transfer would revert anyway, so there is no need for such checks.

40.1 Recommendation

Remove line 470.

41 Double checking requireLicense

Informational

In URI.sol there is an if statement on line 105 that includes a check of `canvas.requireLicense`. But at that part of the code, the value will always be true, because it is already enforced a few lines above (L102).

```
102   if (canvas.requireLicense) {
103       bool isValid;
104       (isValid, uri) = checkValidLicense(collectionAddress, tokenId, refs[i]);
105       if ((canvas.requireLicense && !isValid) || keccak256(bytes(uri)) ==
106   keccak256(bytes(''))) { unchecked { i++; } continue; }
      }
```

41.1 Recommendation

Remove `canvas.requireLicense` check from line 105 like this:

```
102   if (canvas.requireLicense) {
103       bool isValid;
104       (isValid, uri) = checkValidLicense(collectionAddress, tokenId, refs[i]);
105       if (!isValid || keccak256(uri) == keccak256('')) { unchecked { i++; } continue; }
106   }
```

42 ContractOwner is not an interface

Informational

The file IOwnable.sol includes an interface `OwnableInterface` and also a contract called `ContractOwner`. This can be confusing since the file is located in `Interfaces` directory, and is named like an interface, but it includes a contract.

42.1 Recommendation

Consider moving the contract part out of the file and create a separate file called `ContractOwner.sol`.

43 Usage of same functions for different actions

Informational

Throughout the code base there are multiple instances where one function is used to perform different actions. This makes the execution more expensive, since it requires multiple additional parameters and if statements to chose which action to perform. Also, each action in the function can require a different set of parameters.

43.1 Example

In Core.sol there is `changeOwnership` function used to change either curator address, dao address or canvas admin address. It is used both to propose a new owner and to accept ownership. This makes the function complex and requires additional parameters. Also in case the canvas admin is being changed, `canvasId` is needed, but if curator is being changed the `canvasId` parameter is not used. This makes gas consumption for each of these actions higher.

43.2 Recommendation

I would recommend splitting the `changeOwnership` function, and making a different function for each action. You could make `changeCurator`, `acceptCurator`, `changeDao`, `acceptDao`, `changeCanvasAdmin`, `acceptCanvasAdmin`. This requires 5 more functions than before to be made, but the code is more readable and consumes less gas when changing ownership. Of course this requires `changeOwnership` function in Canvas.sol to be split into 6 as well.

43.2.1 Other examples

Create.sol

- `setCreatorsOrRefs` can be split into `setCreators` and `setRefs`
- `createOrUpdateCanvas` can be split into `createCanvas`, `updateCanvas`, `updateTraits` and `updateMintPass`

Core.sol

- `changeProtocolParameter`
- `approveCanvas` can be split into `curateCanvas` and `approveCanvas`
- `setVrfResult` can be split into `setVrfResult`, `setVrfPending` and `resetVrf`

Customize.sol

- `customizeToken` can be split into `updateTitle`, `updateSubtitle` and `updateRefs`
- `updateIdentifiers` can be split into `updateChipIds` and `updatePurchaseIds`

Funds.sol

- `payoutAuctionOrDutchRefund` can be split into `claimExpiredRefund`, `payoutAuction` and `claimDutchRefund`

44 Checks if a value is being changed use more gas

Informational

In Create.sol the `updateCanvas` function receives `Schema.Canvas` struct named `update`. It then compares every value of this struct to current configuration of canvas to see if it is different. In case it is, it saves it to storage. However, this does not save any more gas compared to just overwriting storage variable without checking. Overwriting a storage variable consumes almost the same amount of gas as comparing it to the stored value. Furthermore, if a value is changed, you are both checking and storing which consumes around 1000 gas more than just storing. Also, you are saving only 40 gas if a variable is not changed.* So we can expect that overall checking for changes consumes more gas than necessary and also makes the code less readable.

44.1 Recommendation

Consider removing checking for changes. Only leave the checks that are needed to validate that the change follows the rules.

For example:

```
if (canvas.editioned != update.editioned)
        canvas.editioned = update.editioned;
```

change the above code to:

```
canvas.editioned = update.editioned;
```

*Notice: The gas amounts are derived from my testing.

45 Usage of SafeMath is redundant

Informational

The `SafeMath` library is used to perform arithmetic operations. In older Solidity versions it was used to prevent overflow/underflow. However, starting with Solidity 8.0 the compiler has built in overflow checking. Meaning that `SafeMath` adds no additional safety to this code base. It is just wrapping arithmetic operations with no additional checks. But maybe the intention of using it is purely visual or to avoid rewriting older code.

45.1 Recommendation

Remove `SafeMath` library.

46 Poor logic in domainSeparator

Informational

In Minting.sol there is `_domainSeparatorV4` function used to get the domain separator, needed for signature verification. On line 71 there is a check that `ds.contractInfo.canvasAddress` equals `ds._CACHED_THIS` and `block.chainid` equals `ds._CACHED_CHAIN_ID`. If it is `true`, `_CACHED_DOMAIN_SEPARATOR` is returned. However, the if statement will always be `true` because `ds._CACHED_THIS` and `ds._CACHED_CHAIN_ID` are set to exactly those values in the `initialize` function and are never changed. In case that the result would somehow be `false`, the other line of code that would be executed would not work either, because it needs `ds._TYPE_HASH`, `ds._HASHED_NAME` and `ds._HASHED_VERSION`, which are only set in the `initialize` function as well. So you are counting on the contract being initialized in both cases. This means that if one option works, the other does as well, and if one does not work, the other does not either. We can see the `constructor` in Canvas.sol executes the `initialize` function, so we can be sure that all the needed values are set. Therefore, `_CACHED_DOMAIN_SEPARATOR` can be returned and there is no need for the if statement.

```
52  function initialize(string memory name, string memory version) external {
53      Schema.Storage storage ds = canvasStorage();
54
55      bytes32 hashedName = keccak256(bytes(name));
56      bytes32 hashedVersion = keccak256(bytes(version));
57      bytes32 typeHash = keccak256(
58          "EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)"
59      );
60      ds._HASHED_NAME = hashedName;
61      ds._HASHED_VERSION = hashedVersion;
62      ds._CACHED_CHAIN_ID = block.chainid;
63      ds._CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(typeHash, hashedName, hashedVersion);
64      ds._CACHED_THIS = ds.contractInfo.canvasAddress;
65      ds._TYPE_HASH = typeHash;
66  }
67
68  function _domainSeparatorV4() internal view returns (bytes32) {
69      Schema.Storage storage ds = canvasStorage();
70
71      if (ds.contractInfo.canvasAddress == ds._CACHED_THIS && block.chainid ==
72  ds._CACHED_CHAIN_ID) {
73        return ds._CACHED_DOMAIN_SEPARATOR;
74      } else {
75        return _buildDomainSeparator(ds._TYPE_HASH, ds._HASHED_NAME, ds._HASHED_VERSION);
76      }
    }
```

46.1 Recommendation

Consider removing the if statement in `_domainSeparatorV4` and just return `ds._CACHED_DOMAIN_SEPARATOR`. Only store the `_CACHED_DOMAIN_SEPARATOR` variable and remove other variables, because they are not needed.

```
52
```

```
53
54  function initialize(string memory name, string memory version) external {
55      Schema.Storage storage ds = canvasStorage();
56      ds._CACHED_DOMAIN_SEPARATOR = _buildDomainSeparator(
57          keccak256("EIP712Domain(string name,string version,uint256 chainId,address
58  verifyingContract)"),
59          keccak256(bytes(name)),
60          keccak256(bytes(version))
61          );
62  }
63
    function _domainSeparatorV4() internal view returns (bytes32) {
        return canvasStorage()._CACHED_DOMAIN_SEPARATOR;
    }
```

47 Redundant conversions in views

Informational

In View.sol there are multiple functions that return a struct from storage. In the process of doing that, the struct is first saved to memory, converted to bytes, and then the bytes are returned. That is unnecessary, since you can just return the struct and the return value will be the same. These functions are: `contractInfo`, `getCanvasData` and `getCanvasSystem`.

47.1 Example

```
function contractInfo()
  external view returns (
    bytes memory
  ) {
    Schema.ContractInfo memory info = canvasStorage().contractInfo;
    return bytes.concat(
      abi.encode(
        info.daoAddress,
        info.newDaoRequest,
        info.curatorAddress,
        info.newCuratorRequest,
        info.canvasAddress,
        info.licenseRegistryAddress,
        info.tokenRegistryAddress,
        info.vrfAddress,
        info.keeperAddress,
        info.keeperRegistrarAddress,
        info.linkTokenAddress,
        info.collectionContractAddress
      ),
      abi.encode(
        info.canvasOneAddress,
        info.canvasOneCuratedAddress,
        info.protocolFeeRecipient,
        info.protocolFeeBps,
        info.protocolDutchRefundFeeBps,
        info.canvasCounter,
        info.canvasCuratedCounter,
        info.canvasOpenCounter,
        info.canvasOneCounter,
        info.canvasOneCuratedCounter,
        info.openCanvas,
        info.openCanvasOne
      )
    );
}
```

47.2 Recommendation

Change the code like this:

```
function contractInfo()
  external view returns (
    Schema.ContractInfo memory
  ) {
    return canvasStorage().contractInfo;
}
```

48 Usage of strings increases gas cost

Informational

Throughout the code base, there are multiple cases where strings are used to choose what action we want to perform. However, these strings can use more memory compared to other types and must be hashed to compare. This adds unnecessary gas cost, when you could use a small uint, bytes or an enum.

Functions that use strings: `changeProtocolParameter`, `changeOwnership`, `setVrfResult`, `customizeToken`

48.1 Example

In Core.sol there is `changeProtocolParameter` function that takes string `parameter`. It uses this string to choose which parameter should be changed.

```
function changeProtocolParameter(
    address msgSender,
    string memory parameter,
    address addressVal,
    uint256 numberVal,
    string memory stringVal
  ) external {
    Schema.Storage storage ds = canvasStorage();
    if (msgSender != ds.contractInfo.daoAddress) revert NoPermission();

    if (keccak256(bytes(parameter)) == keccak256(bytes('vrf'))) {
      ds.contractInfo.vrfAddress = addressVal;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('keeper'))) {
      ds.contractInfo.keeperAddress = addressVal;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('tokenRegistryAddress'))) {
      ds.contractInfo.tokenRegistryAddress = addressVal;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('licenseRegistryAddress'))) {
      ds.contractInfo.licenseRegistryAddress = addressVal;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('openCanvas'))) {
      ds.contractInfo.openCanvas = numberVal == 0 ? false : true;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('openCanvasOne'))) {
      ds.contractInfo.openCanvasOne = numberVal == 0 ? false : true;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('protocolFeeRecipient'))) {
      ds.contractInfo.protocolFeeRecipient = payable(addressVal);
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('protocolFeeBps'))) {
      if (numberVal > 3000) revert InvalidFee(); // Maximum 30% protocol fee possible
      ds.contractInfo.protocolFeeBps = numberVal;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('protocolDutchRefundFeeBps'))) {
      if (numberVal > 3000) revert InvalidFee(); // Maximum 30% protocol fee possible
      ds.contractInfo.protocolDutchRefundFeeBps = numberVal;
      emit ContractUpdate();
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('unlockCanvas'))) {
      if (ds.canvas[numberVal].unlockRequest) ds.canvas[numberVal].isImmutable = false;
      emit CanvasUpdate(numberVal);
    } else if (keccak256(bytes(parameter)) == keccak256(bytes('externalBaseUrl'))) {
      ds.externalBaseUrl = stringVal;
    }
}
```

48.2 Recommendation

Change string to uint or enum. Also consider splitting the function like described in Usage of same functions for different actions.

```
enum ProtocolParameter{
  vrf,
  keeper,
  tokenRegistryAddress,
  licenseRegistryAddress,
  openCanvas,
  openCanvasOne,
  protocolFeeRecipient,
  protocolFeeBps,
  protocolDutchRefundFeeBps,
  unlockCanvas,
  externalBaseUrl
  }

function changeProtocolParameter(
    address msgSender,
    ProtocolParameter parameter,
    address addressVal,
    uint256 numberVal,
    string memory stringVal
  ) external {
    Schema.Storage storage ds = canvasStorage();
    if (msgSender != ds.contractInfo.daoAddress) revert NoPermission();

    if (parameter == ProtocolParameter.vrf) {
      ds.contractInfo.vrfAddress = addressVal;
      emit ContractUpdate();
    } else if (parameter ==  ProtocolParameter.keeper) {
      ds.contractInfo.keeperAddress = addressVal;
      emit ContractUpdate();
    } else if (parameter ==  ProtocolParameter.tokenRegistryAddress) {
      ds.contractInfo.tokenRegistryAddress = addressVal;
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.licenseRegistryAddress) {
      ds.contractInfo.licenseRegistryAddress = addressVal;
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.openCanvas)) {
      ds.contractInfo.openCanvas = numberVal == 0 ? false : true;
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.openCanvasOne) {
      ds.contractInfo.openCanvasOne = numberVal == 0 ? false : true;
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.protocolFeeRecipient) {
      ds.contractInfo.protocolFeeRecipient = payable(addressVal);
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.protocolFeeBps) {
      if (numberVal > 3000) revert InvalidFee(); // Maximum 30% protocol fee possible
      ds.contractInfo.protocolFeeBps = numberVal;
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.protocolDutchRefundFeeBps) {
      if (numberVal > 3000) revert InvalidFee(); // Maximum 30% protocol fee possible
      ds.contractInfo.protocolDutchRefundFeeBps = numberVal;
      emit ContractUpdate();
    } else if (parameter == ProtocolParameter.unlockCanvas) {
      if (ds.canvas[numberVal].unlockRequest) ds.canvas[numberVal].isImmutable = false;
      emit CanvasUpdate(numberVal);
    } else if (parameter == ProtocolParameter.externalBaseUrl) {
      ds.externalBaseUrl = stringVal;
    }
}
```

* Notice: When using enums, the values correspond to uint8 values 0, 1, 2 and so on in the order they are defined in. In case that you call the function with an out of range value, a `panic` exception `0x21` will be generated, like it is described in the documentation.

49 Other small inefficiencies

Informational

49.1 Inefficiency 1

In CanvasCreator.sol there is `_IMPLEMENTATION_SLOT` constant. In its definition it is given a value of `0x360894a13ba1a3210667c828492db98dca3e2076cc3735a920a3ca505d382bbc`. In constructor it is then validated that it equals to keccak-256 hash of "eip1967.proxy.implementation" subtracted by 1. This validation is unnecessary, because it will always be right, unless you change the code.

49.1.1 Solution

Remove line 12 `assert(_IMPLEMENTATION_SLOT == bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1));`. If you want to be sure that the value is right, and you are not sure in the hardcoded hex value, then just replace it with `bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1)`. It will give the same result.

```
bytes32 internal constant _IMPLEMENTATION_SLOT =
bytes32(uint256(keccak256("eip1967.proxy.implementation")) - 1);
```

49.2 Inefficiency 2

In Core.sol, in the `setVrfResult` function `tokenCounter` is subtracted by `randomizedTraitCounter` to get the `remainder`. If the `remainder` is higher than 0, then `randomizedTraitCounter` is added up with `remainder` and saved. There is no point to making this calculation since `randomizedTraitCounter + remainder` will always equal `tokenCounter`.

```
310   uint256 tokenCounter = ds.canvasSystem[canvasId].tokenIdCounter;
311   uint256 remainder = tokenCounter.sub(ds.canvasSystem[canvasId].randomizedTraitsCounter);
312
313   if (ds.canvasSystem[canvasId].vrfPending && remainder > 0) {
314     ds.randomizedTraits[canvasId].push(Schema.RandomTraits({
315       randomizedAtTokenId: tokenCounter,
316       randomWord: randomWord
317     }));
318
319     ds.canvasSystem[canvasId].randomizedTraitsCounter =
320   ds.canvasSystem[canvasId].randomizedTraitsCounter.add(remainder);
321     ds.canvasSystem[canvasId].vrfPending = false;
322     ds.canvasSystem[canvasId].vrfLastRunTimestamp = block.timestamp;
323     emit VRFResult(canvasId, tokenCounter, randomWord);
    }
```

49.2.1 Solution

Remove the calculation (`ds.canvasSystem[canvasId].randomizedTraitsCounter.add(remainder)`) and replace it with `tokenCounter`.

```
310   uint256 tokenCounter = ds.canvasSystem[canvasId].tokenIdCounter;
311
312   if (ds.canvasSystem[canvasId].vrfPending &&
313   tokenCounter.sub(ds.canvasSystem[canvasId].randomizedTraitsCounter) > 0) {
314     ds.randomizedTraits[canvasId].push(Schema.RandomTraits({
315       randomizedAtTokenId: tokenCounter,
316       randomWord: randomWord
317     }));
318
319     ds.canvasSystem[canvasId].randomizedTraitsCounter = tokenCounter;
320     ds.canvasSystem[canvasId].vrfPending = false;
321     ds.canvasSystem[canvasId].vrfLastRunTimestamp = block.timestamp;
322     emit VRFResult(canvasId, tokenCounter, randomWord);
    }
```

49.3 Inefficiency 3

In Create.sol there is `createCanvas` function that on line 114 stores a value to storage and then reads it multiple times. Reading from storage is more expensive than reading from memory. So caching the value to memory and reading it from there would be more optimal.

```
114   ds.contractInfo.canvasCounter = ds.contractInfo.canvasCounter.add(1);
115   ds.canvas[ds.contractInfo.canvasCounter] = canvas;
116   ds.canvas[ds.contractInfo.canvasCounter].admin = msgSender;
117   ds.creators[ds.contractInfo.canvasCounter] = new address[](1);
118   ds.creators[ds.contractInfo.canvasCounter][0] = msgSender;
119   if (canvas.feeRecipient == address(0)) ds.canvas[ds.contractInfo.canvasCounter].feeRecipient
120   = payable(msgSender);
121
      return ds.contractInfo.canvasCounter;
```

49.3.1 Solution

Cache the value to memory and read it from there.

```
114   uint256 canvasId = ++ds.contractInfo.canvasCounter;
115   ds.canvas[canvasId] = canvas;
116   ds.canvas[canvasId].admin = msgSender;
117   ds.creators[canvasId] = new address[](1);
118   ds.creators[canvasId][0] = msgSender;
119   if (canvas.feeRecipient == address(0)) ds.canvas[canvasId].feeRecipient =
120   payable(msgSender);
121
      return canvasId;
```

In above example, `feeRecipient` and `admin` are changed when canvas is already saved to storage. You can also make those changes in memory first and store the canvas after.

```
114
```

```
115
116   uint256 canvasId = ++ds.contractInfo.canvasCounter;
117   canvas.admin = msgSender;
118   if (canvas.feeRecipient == address(0)) canvas.feeRecipient = payable(msgSender);
119   ds.canvas[canvasId] = canvas;
120   ds.creators[canvasId] = new address[](1);
121   ds.creators[canvasId][0] = msgSender;

      return canvasId;
```

49.4 Inefficiency 4

In Create.sol on line 359, a new struct is created and saved to storage. The struct includes `mintCounter` variable, which must stay the same as it was before. This means that the value must be read from storage and then included in the new struct being saved. This is unnecessarily complex, when you can just update the values that are being changed.

```
359   ds.partners[canvasId][partners[i]] = Schema.Partnership({
360     discountBps: discounts[i],
361     allocation: allocation[i],
362     mintCounter: ds.partners[canvasId][partners[i]].mintCounter
363   });
```

49.4.1 Solution

Only update the `discountBps` and `allocation`.

```
359   ds.partners[canvasId][partners[i]].discountBps = discounts[i];
360   ds.partners[canvasId][partners[i]].allocation = allocation[i];
```

49.5 Inefficiency 5

In Minting.sol the `mint` function receives `data` and then decodes it using `getMintRequest` to get `MintRequest` struct. However, the function could receive `MintRequest` by default. Also, in that case the `mintRequest` would stay as calldata and wouldn't need to use memory.

```
function mint(
    address msgSender,
    uint256 ethValue,
    uint256 canvasId,
    bytes calldata data
  ) external nonReentrant {
    Schema.Storage storage ds = canvasStorage();
    Schema.Canvas storage canvas = ds.canvas[canvasId];

    Schema.MintRequest memory mintRequest = getMintRequest(data);

    if (!canvas.reserveAuction) defaultMint(msgSender, canvasId, ethValue, mintRequest);
    else {
      mintReserveAuction(msgSender, canvasId, mintRequest.selectedTraits);
      emit OneMinted(msgSender, canvasId, mintRequest);
    }
}
```

49.5.1 Solution

Remove conversion of `data` to `MintRequest` and accept `MintRequest` as argument.

```
function mint(
    uint256 canvasId,
    Schema.MintRequest calldata mintRequest
  ) external nonReentrant {
    Schema.Canvas storage canvas = canvasStorage().canvas[canvasId];

    if (!canvas.reserveAuction) defaultMint(msg.sender, canvasId, msg.value, mintRequest);
    else {
      mintReserveAuction(msg.sender, canvasId, mintRequest.selectedTraits);
      emit OneMinted(msg.sender, canvasId, mintRequest);
    }
}
```

*Notice: in this solution `msgSender` and `ethValue` were also removed, as stated in Redundant use of msg.sender and msg.value.

49.6 Inefficiency 6

In Minting.sol the `checkCanMint` function gets storage of a canvas `Schema.Canvas storage canvas = ds.canvas[canvasId]`, but then proceeds to access it multiple times using `ds.canvas[canvasId]` instead of `canvas`.

```
231   Schema.Canvas storage canvas = ds.canvas[canvasId];
232
233   if (canvas.bulkMax > 0 && mintRequest.quantity > canvas.bulkMax)
234     revert OverMintLimit(canvas.bulkMax);
235   if (ds.partners[canvasId][msgSender].discountBps > 0) {
236     if (ds.partners[canvasId][msgSender].mintCounter + mintRequest.quantity >
237   ds.partners[canvasId][msgSender].allocation)
238       revert OverAllowance(ds.partners[canvasId][msgSender].allocation);
239     ds.partners[canvasId][msgSender].mintCounter += mintRequest.quantity;
240   } else if (canvas.walletMax > 0 && ds.purchaseTracker[canvasId][msgSender].quantity +
241   mintRequest.quantity > canvas.walletMax)
```

```
    revert OverMintLimit(canvas.walletMax - ds.purchaseTracker[canvasId][msgSender].quantity);


  if (ds.canvas[canvasId].presaleActive && !ds.canvas[canvasId].saleActive) {
    if (ds.canvas[canvasId].mintPassPerQuantity > 0 &&
        ds.mintPassTokens[canvasId].length > 0)
      processMintPass(msgSender, canvasId, mintRequest);
    else checkAllowList(msgSender, canvasId, mintRequest);
  } else {
    if (canvas.saleStart > 0 &&
        (canvas.saleStart > block.timestamp || canvas.saleEnd < block.timestamp) ||
        !ds.canvas[canvasId].saleActive)
      revert SaleInactive();

    if (ds.canvas[canvasId].mintPassPerQuantity > 0 &&
        ds.mintPassTokens[canvasId].length > 0)
      processMintPass(msgSender, canvasId, mintRequest);
    else if (ds.canvas[canvasId].restrictToAllowList)
      checkAllowList(msgSender, canvasId, mintRequest);
  }
```

49.6.1 Solution

Replace `ds.canvas[canvasId]` with `canvas`.

```
  Schema.Canvas storage canvas = ds.canvas[canvasId];

  if (canvas.bulkMax > 0 && mintRequest.quantity > canvas.bulkMax)
    revert OverMintLimit(canvas.bulkMax);
  if (ds.partners[canvasId][msgSender].discountBps > 0) {
    if (ds.partners[canvasId][msgSender].mintCounter + mintRequest.quantity >
  ds.partners[canvasId][msgSender].allocation)
      revert OverAllowance(ds.partners[canvasId][msgSender].allocation);
    ds.partners[canvasId][msgSender].mintCounter += mintRequest.quantity;
  } else if (canvas.walletMax > 0 && ds.purchaseTracker[canvasId][msgSender].quantity +
  mintRequest.quantity > canvas.walletMax)
    revert OverMintLimit(canvas.walletMax - ds.purchaseTracker[canvasId][msgSender].quantity);


  if (canvas.presaleActive && !canvas.saleActive) {
    if (canvas.mintPassPerQuantity > 0 &&
        ds.mintPassTokens[canvasId].length > 0)
      processMintPass(msgSender, canvasId, mintRequest);
    else checkAllowList(msgSender, canvasId, mintRequest);
  } else {
    if (canvas.saleStart > 0 &&
        (canvas.saleStart > block.timestamp || canvas.saleEnd < block.timestamp) ||
        !canvas.saleActive)
      revert SaleInactive();

    if (canvas.mintPassPerQuantity > 0 &&
        ds.mintPassTokens[canvasId].length > 0)
      processMintPass(msgSender, canvasId, mintRequest);
    else if (canvas.restrictToAllowList)
      checkAllowList(msgSender, canvasId, mintRequest);
  }
```

49.7 Inefficiency 7

In URI.sol the `getSelectedTraits` function gets `canvasStorage()` two times unnecessarily. The first time it is saved to `ds` variable, meaning there is no need to get it the second time, because it can be accessed using `ds`.

```
Schema.Storage storage ds = canvasStorage();
uint256 canvasId = ds.collectionToCanvas[collectionAddress];
Schema.CanvasTraits[] storage canvasTraits = canvasStorage().canvasTraits[canvasId];
```

49.7.1 Solution

Replace the second `canvasStorage` with `ds`.

```
Schema.Storage storage ds = canvasStorage();
uint256 canvasId = ds.collectionToCanvas[collectionAddress];
Schema.CanvasTraits[] storage canvasTraits = ds.canvasTraits[canvasId];
```

49.8 Inefficiency 8

In URI.sol the `getSelectedTraits` function overwrites `encodedTraits` with its own value in case `hasTrait` is `false`.

```
encodedTraits = hasTrait ?
  string(abi.encodePacked(
    keccak256(bytes(encodedTraits)) != keccak256(bytes('')) ?
    selectedTraits
    : Parser.substring(selectedTraits, 0, bytes(selectedTraits).length.sub(1)),
    encodedTraits
  )) : encodedTraits;
```

49.8.1 Solution

Change the code so you only update `encodedTraits` if `hasTrait` is `true`.

```
if(hasTrait) {
  encodedTraits = string(
    abi.encodePacked(
      keccak256(bytes(encodedTraits)) != keccak256(bytes('')) ?
      selectedTraits
      : Parser.substring(selectedTraits, 0, bytes(selectedTraits).length.sub(1)),
      encodedTraits
    ));
}
```

49.9 Inefficiency 9

In URI.sol the `tokenURI` function is making same hashing and comparison two times (L489 and L505), same conversion of uint to string three times (L484, L490 and L499) and there is also a redundant `encodePacked` operation on line 496. All of these increase gas cost.

```
478   string memory animationUrl = string(abi.encodePacked(
479     ds.canvasAssets[canvasId][versionIndex].baseUri,
480     getRefUri(collectionAddress, tokenId),
481     specialParams,
482     getCustomTitles(collectionAddress, tokenId),
483     '&id=',
484     Parser.uint2str(tokenId)
485   ));
486
487   string memory tokenAddressString = string(abi.encodePacked(
488     Parser.addressToString(collectionAddress),
489     keccak256(bytes(canvas.externalUrl)) != keccak256(bytes('')) && canvas.externalUrlSlash ?
490   "/" : ":",
491     Parser.uint2str(tokenId)
492   ));
493
494   return string(abi.encodePacked(
495     "data:application/json;base64,",
496     Base64.encode(bytes(abi.encodePacked(
497       '{"name":"', string(abi.encodePacked(
498         canvas.name,
499         canvas.editioned ? ' #' : '',
500         canvas.editioned ? Parser.uint2str(tokenId) : '')),
501       '", "description":"', usageDescription(canvasId, collectionAddress, tokenId),
502       '","image":"',ds.canvasAssets[canvasId][versionIndex].thumbnailUri,
503       '","animation_url":"',animationUrl,
504       '","collection_url":"', canvas.collectionUri,
505       '","external_url":"',
506         keccak256(bytes(canvas.externalUrl)) != keccak256(bytes('')) ?
507           canvas.externalUrl :
508           ds.externalBaseUrl
509         ,tokenAddressString,
510       '","attributes": [', encodedTraits,'] }'
511     ))))
    );
```

49.9.1 Solution

Cache the result of conversion (`Parser.uint2str(tokenId)`) and comparison (`keccak256(bytes(canvas.externalUrl)) != keccak256(bytes(''))`) to memory. Remove the redundant `encodePacked` operation.

```
478   string tokenIdStr = Parser.uint2str(tokenId);
479   bool hasExternalUrl = keccak256(bytes(canvas.externalUrl)) != keccak256(bytes(''));
480
481   string memory animationUrl = string(abi.encodePacked(
482     ds.canvasAssets[canvasId][versionIndex].baseUri,
483     getRefUri(collectionAddress, tokenId),
484     specialParams,
485     getCustomTitles(collectionAddress, tokenId),
486     '&id=',
487     tokenIdStr
488   ));
489
490   string memory tokenAddressString = string(abi.encodePacked(
491     Parser.addressToString(collectionAddress),
492     hasExternalUrl && canvas.externalUrlSlash ? "/" : ":",
```

```
    tokenIdStr
  ));

  return string(abi.encodePacked(
    "data:application/json;base64,",
    Base64.encode(bytes(abi.encodePacked(
      '{"name":"',
        canvas.name,
        canvas.editioned ? ' #' : '',
        canvas.editioned ? tokenIdStr : '',
      '", "description":"', usageDescription(canvasId, collectionAddress, tokenId),
      '","image":"',ds.canvasAssets[canvasId][versionIndex].thumbnailUri,
      '","animation_url":"',animationUrl,
      '","collection_url":"', canvas.collectionUri,
      '","external_url":"',
        hasExternalUrl ?
          canvas.externalUrl :
          ds.externalBaseUrl
        ,tokenAddressString,
      '","attributes": [', encodedTraits,'] }'
    ))))
  );
```

50 Disclaimer

Perfect Abstractions LLC receives payment from clients (the “Clients”) for reviewing code and writing these reports (the “Reports”).

The Reports are not an accusation or endorsement of any project or team, and the Reports do not guarantee the security of any project. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. To remove any doubt, this Report is not investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the security of the project.

The Reports are created for Clients and published with their consent. The scope of our review is limited to the code or files that are specified in this report. The Solidity language remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond specified code that could present security risks.