SPEARBIT

Astaria Security Review

Auditors

Sawmon and Natalie, Lead Security Researcher

Noah Marconi, Lead Security Researcher

Zach Obront, Security Researcher

Blockdev, Associate Security Researcher

Report prepared by: Pablo Misirov

February 9, 2023

Contents

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Astaria is a NFT Collateralized Lending Market leveraging a novel 3AM Model.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of astaria-core and astaria-GPL according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 15 days in total, Astaria engaged with Spearbit to review the astaria-core protocol and astaria-GPL contracts. In this period of time a total of **183** issues were found.

Summary

Project Name	Astaria
Repository	astaria-core
Repository	astaria-GPL
Commit astaria-core	7e957460...1ee480eb
Commit astaria-GPL	afdd1dfb...a84c2244
Type of Project	Borrowing/Lending, NFT
Audit Timeline	Nov 22 - Dec 12
Two week fix period	Dec 12 - Dec 26

Issues Found

Severity	**Count**
Critical Risk	6
High Risk	24
Medium Risk	17
Low Risk	28
Gas Optimizations	30
Informational	78
Total	**183**

5 Findings

5.1 Critical Risk

5.1.1 `LienToken.transferFrom` does not update a public vault's bookkeeping parameters when a lien is transferred to it.

Severity: Critical Risk

Context: LienToken.sol#L303

Description: When `transferFrom` is called, there is not check whether the `from` or `to` parameters could be a public vault. Currently, there is no mechanism for public vaults to transfer their liens.

But private vault owners who are also owners of the vault's lien tokens, they can call `transferFrom` and transfer their liens to a public vault. In this case, we would need to make sure to update the bookkeeping for the public vault that the lien was transferred `to`.

On the `LienToken` side, `s.LienMeta[id].payee` needs to be set to the address of the public vault. And on the `PublicVault` side, `yIntercept`, `slope`, `last`, `epochData` of `VaultData` need to be updated (this requires knowing the lien's end). However, private vaults do not keep a record of these values, and the corresponding values are only saved in stacks off-chain and validated on-chain using their hash.

Recommendation:

- Either to block transferring liens to public vaults or
- Private vaults or the `LienToken` would need to have more storage parameters that would keep a record of some values for each lien so that when the time comes to transfer a lien to a public vault, the parameters mentioned in the Description can be updated for the public vault.

5.1.2 Anyone can take a valid commitment combined with a self-registered private vault to steal funds from any vault without owning any collateral

Severity: Critical Risk

Context: VaultImplementation.sol#L279, VaultImplementation.sol#L227

Description: The issue stems from the following check in `VaultImplementation._validateCommitment(params, receiver)`:

```
if (
  msg.sender != holder &&
  receiver != holder &&
  receiver != operator &&
  !ROUTER().isValidVault(receiver) // <-- the problematic condition
) { ...
```

In this `if` block if `receiver` is a valid vault the body of the `if` is skipped. A valid vault is one that has been registered in `AstariaRouter` using `newVault` or `newPublicVault`. So for example any supplied private vault as a `receiver` would be allowed here and the call to `_validateCommitment` will continue without reverting at least in this `if` block.

If we backtrack function calls to `_validateCommitment`, we arrive to `3` exposed endpoints:

- `commitToLiens`
- `buyoutLien`
- `commitToLien`

A call to `commitToLiens` will end up having the `receiver` be the `AstariaRouter`. A call to `buyoutLien` will set the `receiver` as the `recipient()` for the vault which is either the vault itself for public vaults or the `owner` for private vaults. So we are only left with `commitToLien`, where the caller can set the value for the `receiver` directly.

A call to `commitToLien` will initiate a series of function calls, and so receiver is only supplied to _validateCommitment to check whether it is allowed to be used and finally when transferring safeTransfer) `wETH`.

This opens up exploiting scenarios where an attacker:

1. Creates a new private vault by calling `newVault`, let's call it V.
2. Takes a valid commitment C and combines it with V and supply those to `commitToLien`.
3. Calls `withdraw` endpoint of V to withdraw all the funds.

For step 2. the attacker can source valid commitments by doing either of the following:

1. Frontrun calls to `commitToLiens` and take all the commitments $C_0, \cdots, C_n$ and supply them one by one along with V to `commitToLien` endpoint of the vault that was specified by each C_i.
2. Frontrun calls to `commitToLien` endpoints of vaults, take their commitment C and combine it with V to send to `commitToLien`.
3. Backrun the either scenarios from the above points and create a new commitment with new lien request that tries to max out the potential debt for a collateral while also keeping other inequalities valid (for example, the inequality regarding `liquidationInitialAsk`).

Recommendation: If the `commitToLien` endpoint is only supposed to be called by `AstariaRouter` make sure to apply that restriction.

Or if it is allowed to be called by anyone, the `!ROUTER().isValidVault(receiver)` condition would need to be modified. As an example, one can use the `commitment.lienRequest.strategy.vault` (let's call it V_s) and make sure that when `ROUTER().isValidVault(receiver) === true` then $R = V_s$ (where R is the `receiver`, note some parameters might be redundant in this case) and also we would need to take into consideration that either the vault `owners` or `delegates` also signing V_s or the strategy validators would take this value into consideration when `validateAndParse` is called. This 2nd recommendation might interfere with what `commitment.lienRequest.strategy.vault` would need to represent in other places (the vault that the amount is borrowed from, not sent to).

5.1.3 Collateral owner can steal funds by taking liens while asset is listed for sale on Seaport

Severity: Critical Risk

Context: LienToken.sol#L368-372

Description: We only allow collateral holders to call `listForSaleOnSeaport` if they are listing the collateral at a price that is sufficient to pay back all of the liens on their collateral.

When a new lien is created, we check that `collateralStateHash != bytes32("ACTIVE_AUCTION")` to ensure that the collateral is able to accept a new lien.

However, calling `listForSaleOnSeaport` does not set the `collateralStateHash`, so it doesn't stop us from taking new liens.

As a result, a user can deposit collateral and then, in one transaction:

- List the asset for sale on Seaport for 1 wei.
- Take the maximum possible loans against the asset.
- Buy the asset on Seaport for 1 wei.

The 1 wei will not be sufficient to pay back the lenders, and the user will be left with the collateral as well as the loans (minus 1 wei).

Recommendation: Either set the `collateralStateHash` when an item is listed for sale on Seaport, or check the `s.collateralIdToAuction` variable before allowing a lien to be taken.

Astaria: `listForSaleOnSeaport` has been removed in the following PR and that resolves the issue PR 206.

Spearbit: Verified.

5.1.4 validateStack allows any stack to be used with collateral with no liens

Severity: Critical Risk

Context: LienToken.sol#L225-232

Description: The `validateStack` modifier is used to confirm that a stack entered by a user matches the stateHash in storage.

However, the function reverts under the following conditions:

```
if (stateHash != bytes32(0) && keccak256(abi.encode(stack)) != stateHash) {
    revert InvalidState(InvalidStates.INVALID_HASH);
}
```

The result is that any collateral with `stateHash == bytes32(0)` (which is all collateral without any liens taken against it yet) will accept any provided stack as valid.

This can be used in a number of harmful ways. Examples of vulnerable endpoints are:

- `createLien`: If we create the first lien but pass a stack with other liens, those liens will automatically be included in the stack going forward, which means that the collateral holder will owe money they didn't receive.
- `makePayment`: If we make a payment on behalf of a collateral with no liens, but include a stack with many liens (all owed to me), the result will be that the collateral will be left with the remaining liens continuing to be owed
- `buyoutLien`: Anyone can call `buyoutLien(...)` and provide parameters that are spoofed but satisfy some constraints so that the call would not revert. This is currently possible due to the issue in this context. As a consequence the caller can
  - `_mint` any unminted liens which can DoS the system.
  - `_burn lienIds` that they don't have the right to remove.
  - manipulate any public vault's storage (if it has been set as a `payee` for a lien) through its `handleBuyoutLien`. It seems like this endpoint might have been meant to be a restricted endpoint that only registered vaults can call into. And the caller/user is supposed to only call into here from `VaultImplementation.buyoutLien`.

Recommendation:

```
 modifier validateStack(uint256 collateralId, Stack[] memory stack) {
   LienStorage storage s = _loadLienStorageSlot();
   bytes32 stateHash = s.collateralStateHash[collateralId];
+  if (stateHash == bytes32(0) && stack.length != 0) {
+    revert InvalidState(InvalidStates.EMPTY_STATE);
+  }
   if (stateHash != bytes32(0) && keccak256(abi.encode(stack)) != stateHash) {
     revert InvalidState(InvalidStates.INVALID_HASH);
   }
   _;
 }
```

This will also require adding the `InvalidStates.EMPTY_STATE` to the enum.

Astaria: PR 194.

Spearbit: Confirmed that this is fixed in the following PR 194.

5.1.5 A borrower can list their collateral on Seaport and receive almost all the listing price without paying back their liens

Severity: Critical Risk

Context: LienToken.sol#L480

Description: When the collateral is listed on `SeaPort` by the borrower using `listForSaleOnSeaport`, `s.auctionData` is not populated and thus, if that order gets fulfilled/matched and `ClearingHouse`'s fallback function gets called since `stack.length` is `0`, this loop will not run and no payment is sent to the lending vaults.

The rest of the `payment` is sent to the borrower. And the collateral token and its related data gets burnt/deleted by calling `settleAuction`. The lien tokens and the vaults remain untouched as though nothing has happened.

So basically a borrower can:

1. Take/borrow liens by offering a collateral.
2. List their collateral on `SeaPort` through the `listForSaleOnSeaport` endpoint.
3. Once/if the `SeaPort` order fulfills/matches, the borrower would be paid the listing price minus the amount sent to the `liquidator` (`address(0)` in this case, which should be corrected).
4. Collateral token/data gets burnt/deleted.
5. Lien token data remains and the loans are not paid back to the vaults.

And so the borrower could end up with all the loans they have taken plus the listing price from the `SeaPort` order.

Note that when a user lists their own collateral on `Seaport`, it seems that we intentionally do not kick off the auction process:

- Liens are continued.
- Collateral state hash is unchanged.
- `liquidator` isn't set.
- Vaults aren't updated.
- Withdraw proxies aren't set, etc.

Related issue 88.

Recommendation: Be careful and also pay attention that listing by a borrower versus auctioning by a liquidator take separate return/payback paths. It is recommended to separate the listing and liquidating logic and make sure auction funds and distributed appropriately. Most importantly, the auction stack must be set.

Astaria: We've removed the ability for self-listing on seaport as the fix for v0, will add this feature this in a future release.

Spearbit: Fixed in the following PR by removing the `listForSaleOnSeaport` endpoint PR 206.

5.1.6 Phony signatures can be used to forge any strategy

Severity: Critical Risk

Context: VaultImplementation.sol#L249

Description: In `_validateCommitment()`, we check that the merkle root of the strategy has been signed by the strategist or delegate.

After the signer is recovered, the following check is performed to validate the signature:

```
recovered != owner() && recovered != s.delegate && recovered != address(0)
```

This check seems to be miswritten, so that any time `recovered == address(0)`, the check passes.

When `ecrecover` is used to check the signed data, it returns `address(0)` in the situation that a phony signature is submitted. See this example for how this can be done.

The result is that any borrower can pass in any merkle root they'd like, sign it in a way that causes `address(0)` to return from `ecrecover`, and have their commitment validated.

Recommendation:

```diff
    if (
-     recovered != owner() && recovered != s.delegate && recovered != address(0)
+     (recovered != owner() && recovered != s.delegate) || recovered == address(0)
    ) {
      revert IVaultImplementation.InvalidRequest(
        InvalidRequestReason.INVALID_SIGNATURE
      );
    }
```

Astaria: Fixed in PR 209.

Spearbit: Verified.

5.2 High Risk

5.2.1 Inequalities involving `liquidationInitialAsk` and `potentialDebt` can be broken when `buyoutLien` is called

Severity: High Risk

Context:

- LienToken.sol#L102
- VaultImplementation.sol#L305
- LienToken.sol#L377-L378
- LienToken.sol#L427
- AstariaRouter.sol#L542

Description: When we commit to a new lien, the following gets checked to be true for all $j \in 0, \cdots, n-1$:

$$o_{new} + o_{n-1} + \cdots + o_j \leq L_j$$

where:

parameter	description
o_i	`_getOwed(newStack[i], newStack[i].point.end)`
o_{new}	`_getOwed(newSlot, newSlot.point.end)`
n	`stack.length`
L_i	`newStack[i].lien.details.liquidationInitialAsk`
L'_k	`params.encumber.lien.details.liquidationInitialAsk`
k	`params.position`
A'_k	`params.encumber.amount`

so in a `stack` in general we should have the:

$$\cdots + o_{j+1} + o_j \leq L_j$$

But when an old lien is replaced with a new one, we only perform the following checks for L'_k:

$$L'_k \geq A'_k \land L'_k > 0$$

And thus we can introduce:

- $L'_k \ll L_k$ or
- $o'_k \gg o_k$ (by pushing the lien duration)

which would break the inequality regarding o_i s and L_i .

If the inequality is broken, for example, if we buy out the first lien in the stack, then if the lien expires and goes into a `Seaport` auction the auction's starting price L_0 would not be able to cover all the potential debts even at the beginning of the auction.

Recommendation: When `buyoutLien` is called, we would need to loop over j and check the inequalities again:

$$\cdots + o_{j+1} + o_j \leq L_j$$

5.2.2 `VaultImplementation.buyoutLien` can be DoSed by calls to `LienToken.buyoutLien`

Severity: High Risk

Context:

- LienToken.sol#L102
- LienToken.sol#L121
- VaultImplementation.sol#L305

Description: Anyone can call into `LienToken.buyoutLien` and provide `params` of the type `LienActionBuyout`:

`params.incoming` is not used, so for example vault signatures or strategy validation is skipped. There are a few checks for `params.encumber`.

Let's define the following variables:

parameter	value
i	`params.position`
k_j	`params.encumber.stack[j].point.position`
t_j	`params.encumber.stack[j].point.last`
e_j	`params.encumber.stack[j].point.end`
e'_i	$t_{now} + D'_i$
l_j	`params.encumber.stack[j].point.lienId`
l'_i	$h(N'_i, V'_i, S'_i, c'_i, (A'^{max}_i, r'_i, D'_i, P'_i, L'_i))$ where h is the `keccak256` of the encoding
r_j	`params.encumber.stack[j].lien.details.rate` : old rate
r'_i	`params.encumber.lien.details.rate` : new rate
c	`params.encumber.collateralId`

parameter	value
c_j	`params.encumber.stack[j].lien.collateralId`
c'_i	`params.encumber.lien.collateralId`
A_j	`params.encumber.stack[j].point.amount`
A'_i	`params.encumber.amount`
A_j^{max}	`params.encumber.stack[j].lien.details.maxAmount`
$A_i'^{max}$	`params.encumber.lien.details.maxAmount`
R	`params.encumber.receiver`
N_j	`params.encumber.stack[j].lien.token`
N'_i	`params.encumber.lien.token`
V_j	`params.encumber.stack[j].lien.vault`
V'_i	`params.encumber.lien.vault`
S_j	`params.encumber.stack[j].lien.strategyRoot`
S'_i	`params.encumber.lien.strategyRoot`
D_j	`params.encumber.stack[j].lien.details.duration`
D'_i	`params.encumber.lien.details.duration`
P_j	`params.encumber.stack[j].lien.details.maxPotentialDebt`
P'_i	`params.encumber.lien.details.maxPotentialDebt`
L_j	`params.encumber.stack[j].lien.details.liquidationInitialAsk`
L'_i	`params.encumber.lien.details.liquidationInitialAsk`
I_{min}	`AstariaRouter.s.minInterestBPS`
D_{min}	`AstariaRouter.s.minDurationIncrease`
t_{now}	`block.timestamp`
b_i	`buyout`
o	`_getOwed(params.encumber.stack[params.position], block.timestamp)`
o_j	`_getOwed(params.encumber.stack[j], params.encumber.stack[j].point.end)`
n	`params.encumber.stack.length`
$O = o_0 + o_1 + \cdots + o_{n-1}$	`_getMaxPotentialDebtForCollateral(params.encumber.stack)`
s_j	`params.encumber.stack[j]`
s'_i	`newStack`

Let's go over the checks and modifications that `buyoutLien` does:

1. `validateStack` is called to make sure that the hash of `params.encumber.stack` matches with `s.collateralStateHash` value of c. This is not important and can be bypassed by the exploit even after the fix for Issue 106.

2. `_createLien` is called next which does the following checks: 2.1. c is not up for auction. 2.2. We haven't reached max number of liens, currently set to 5. 2.3. $L'_i \geq A'_i$ and $L'_i > 0$ 2.4. If `params.encumber.stack` is not empty then $c'_i = c_0$ 2.5. We `_mint` a new lien for R with id equal to $h(N'_i, V'_i, S'_i, c'_i, (A_i'^{max}, r'_i, D'_i, P'_i, L'_i))$ where h is the hashing mechanism of encoding and then taking the `keccak256`. 2.6 The new stack slot and

the new lien id is returned.

3. `isValidRefinance` is called which performs the following checks: 3.1. checks $c'_i = c_0$ 3.2. checks either

$$(r'_i < r_i - I_{min}) \land (e'_i \geq e_i)$$

or

$$(r'_i \leq r_i) \land (e'_i \geq e_i + D_{min})$$

4. check where c'_i is in auction by checking `s.collateralStateHash`'s value.
5. check $O \leq P'_i$.
6. check $A'^{max}_i \geq o$.
7. send `wETH` through `TRANSFER_PROXY` from `msg.sender` to `payee` of l_i with the amount of b_i.
8. if `payee` of l_i is a public vault, do some book keeping by calling `handleBuyoutLien`.
9. call `_replaceStackAtPositionWithNewLien` to:

- 9.1. replace s_i with s'_i in `params.encumber.stack`.
- 9.2. `_burn` l_i.
- 9.3. `delete s.lienMeta` of l_i.

So in a nutshell the important checks are:

- c, c_i are not in auction (not important for the exploit)
- $c'_i = c_0$
- n is less than or equal to max number of allowed liens (5 currently) (not important for the exploit)
- $L'_i \geq A'_i$ and $L'_i > 0$
- $O \leq P'_i$
- $A'^{max}_i \geq o$

$$(r'_i < r_i - I_{min}) \land (e'_i \geq e_i)$$

or

$$(r'_i \leq r_i) \land (e'_i \geq e_i + D_{min})$$

Exploit

An attacker can DoS the `VaultImplementation.buyoutLien` as follows:

1. A vault decides to buy out a collateral's lien to offer better terms and so signs a commitment and someone on behalf of the vault calls `VaultImplementation.buyoutLien` which if executed would call `LienToken.buyoutLien` with the following parameters:

```
LienActionBuyout({
  incoming: incomingTerms,
  position: position,
  encumber: ILienToken.LienActionEncumber({
    collateralId: collateralId,
    amount: incomingTerms.lienRequest.amount,
    receiver: recipient(),
    lien: ROUTER().validateCommitment({
      commitment: incomingTerms,
      timeToSecondEpochEnd: _timeToSecondEndIfPublic()
    }),
    stack: stack
  })
})
```

2. The attacker fronrun the call from step 1. and instead provide the following modified parameters to `LienToken.buyoutLien`

```
LienActionBuyout({
  incoming: incomingTerms, // not important, since it is not used and can be zeroed-out to save tx gas
  position: position,
  encumber: ILienToken.LienActionEncumber({
    collateralId: collateralId,
    amount: incomingTerms.lienRequest.amount,
    receiver: msg.sender, // address of the attacker
    lien: ILienToken.Lien({ // note that the lien here would have the same fields as the original
↪   message by the vault rep.
        token: address(s.WETH),
        vault: incomingTerms.lienRequest.strategy.vault, // address of the vault offering a better term
        strategyRoot: incomingTerms.lienRequest.merkle.root,
        collateralId: collateralId,
        details: details // see below
    }),
    stack: stack
  })
})
```

Where `details` provided by the attacker can be calculated by using the below snippet:

```
uint8 nlrType = uint8(_sliceUint(commitment.lienRequest.nlrDetails, 0));
(bytes32 leaf, ILienToken.Details memory details) = IStrategyValidator(
  s.strategyValidators[nlrType]
).validateAndParse(
  commitment.lienRequest,
  s.COLLATERAL_TOKEN.ownerOf(
    commitment.tokenContract.computeId(commitment.tokenId)
  ),
  commitment.tokenContract,
  commitment.tokenId
);
```

The result is that:

- The `newLienId` that was supposed to be `_mint`ed for the ‘recipient()‘ of the vault, gets minted for the `attacker`.
- The call to `VaultImplementation.buyoutLien` would fail, since the `newLienId` is already minted, and so the vault would not be able to receives the interests it had anticipated.
- When there is a payment or `Seaport` auction settlement, the attacker would receive the funds instead.

- The attacker can intorduces a malicous contract into the protocol that would be `LienToken.ownerOf(newLienId)` without needing to register for a vault.

To execute this attack, the attacker would need to spend the `buyout` amount of assets. Also the attacker does not necessarily need to front run a transaction to buyout a lien. They can pick their own hand-crafted parameters that would satisfy the conditions in the analysis above to introduce themselves in the protocol.

Recommendation: There are multiple ways to mitigate this issue.

1. We can restrict `LienToken.buyoutLien` endpoint to be only called by the registered vaults in `AstariaRouter`.
2. In `LienToken.buyoutLien`, use `params.incoming` to validate the signatures and lien details.

The above 2 solutions would prevent an attacker introducing/minting a new lien id using parameters from a different vault without themselves registering a vault.

Spearbit: This is resolved in the following commit by restricting the `buyoutLien` of the `LienToken` to only valid/registered vaults: commit 24da50.

5.2.3 `VaultImplementation.buyoutLien` does not update the new public vault's parameters and does not transfer assets between the vault and the borrower

Severity: High Risk

Context:

- VaultImplementation.sol#L305
- LienToken.sol#L102
- LienToken.sol#L116
- LienToken.sol#L165-L174

Description: `VaultImplementation.buyoutLien` does not update the accounting for the `vault` (if it's public). The `slope`, `yIntercept`, and `s.epochData[...].liensOpenForEpoch` (for the new lien's end epoch) are not updated. They are updated for the `payee` of the swapped-out lien if the `payee` is a public vault by calling `handleBuyoutLien`.

Also, the buyout amount is paid out by the vault itself. The difference between the new lien amount and the buyout amount is not worked out between the `msg.sender` and the new vault.

Recommendation:

1. If the vault that `VaultImplementation.buyoutLien` endpoint was called into is a public vault, make sure to update its `slope`, `yIntercept`, and `s.epochData[...].liensOpenForEpoch` (for the new lien's end epoch) when the new lien is created.
2. The difference between the new lien amount and the buyout amount is not worked out between the `msg.sender` that called `VaultImplementation.buyoutLien` and the vault. If the buyout amount is higher than the new lien amount, we need to make sure the `msg.sender` also transfers some assets (`wETH`) to the vault. And the other way around, if the new lien amount is higher than the buyout amount, the vault needs to transfer some assets (`wETH`) to the borrower / `msg.sender`.

5.2.4 setPayee doesn't update y intercept or slope, allowing vault owner to steal all funds

Severity: High Risk

Context:

- LienToken.sol#L868-878
- LienToken.sol#L165-173

Description: When `setPayee()` is called, the payment for the lien is no longer expected to go to the vault. However, this change doesn't impact the vault's y-intercept or slope, which are used to calculate the vault's `totalAssets()`.

This can be used maliciously by a vault owner to artificially increase their `totalAssets()` to any arbitrary amount:

- Create a lien from the vault.
- SetPayee to a non-vault address.
- Buyout the lien from another vault (this will cause the other vault's y-int and slope to increase, but will not impact the y-int and slope of the original vault because it'll fail the check on L165 that payee is a public vault.
- Repeat the process again going the other way, and repeat the full cycle until both vault's have desired `totalAssets()`.

For an existing vault, a vault owner can withdraw a small amount of assets each epoch. If, in any epoch, they are one of the only users withdrawing funds, they can perform this attack immediately before the epoch is processed. The result is that the withdrawal shares will by multiplied by `totalAssets() / totalShares()` to get the withdrawal rate, which can be made artificially high enough to wipe out the entire vault.

Recommendation: Adjust the y-intercept and slope of the old payee and the new payee immediately upon the payee being set.

Astaria: We're thinking of removing the ability for the owner to change the payee altogether. There's no clear benefit to having this in the first place, since the payee would have no guarantees on receiving funds since we reset payee on LienToken transfers. We can just lock setPayee() to only be callable by a WithdrawProxy (if it needs auction funds), which is the primary use case anyways.

Spearbit: Confirmed, removing the setPayee function in the following PR PR 205 solves the issue.

5.2.5 `settleAuction()` doesn't check if the auction was successful

Severity: High Risk

Context: CollateralToken.sol#L600

Description: `settleAuction()` is a privileged functionality called by `LienToken.payDebtViaClearingHouse()`. `settleAuction()` is intended to be called on a successful auction, but it doesn't verify that that's indeed the case.

Anyone can create a fake Seaport order with one of its considerations set as the `CollateralToken` as described in Issue 93.

Another potential issue is if the Seaport orders can be "Restricted" in future, then there is a possibility for an authorized entity to force settleAuction on CollateralToken, and when SeaPort tries to call back on the zone to validate it would fail.

Recommendation: The following validations can be performed:

- `CollateralToken` doesn't own the underlying NFT.
- `collateralIdToAuction[collateralId]` is active.

Now, `settleAuction()` can only be called on the success of the Seaport auction created by Astaria protocol.

5.2.6 Incorrect auction end validation in `liquidatorNFTClaim()`

Severity: High Risk

Context: CollateralToken.sol#L119

Description: `liquidatorNFTClaim()` does the following check to recognize that Seaport auction has ended:

```
if (block.timestamp < params.endTime) {
  //auction hasn't ended yet
  revert InvalidCollateralState(InvalidCollateralStates.AUCTION_ACTIVE);
}
```

Here, `params` is completely controlled by users and hence to bypass this check, the caller can set `params.endTime` to be less than `block.timestamp`.

Thus, a possible exploit scenario occurs when `AstariaRouter.liquidate()` is called to list the underlying asset on Seaport which also sets `liquidator` address. Then, anyone can call `liquidatorNFTClaim()` to transfer the underlying asset to `liquidator` by setting `params.endTime < block.timestamp`.

Recommendation: The parameter passed to `liquidatorNFTClaim()` should be validated against the parameters created for the Seaport auction. To do that:

- `collateralIdToAuction` mapping which currently maps `collateralId` to a boolean value indicating an active auction, should instead map from `collateralId` to Seaport order hash.
- All usages of `collateralIdToAuction` should be updated. For example, `isValidOrder()` and `isValidOrderIncludingExtraData()` should be updated:

```
  return
  -   s.collateralIdToAuction[uint256(zoneHash)]
  +   s.collateralIdToAuction[uint256(zoneHash)] == orderHash
        ? ZoneInterface.isValidOrder.selector
        : bytes4(0xffffffff);
```

- `liquidatorNFTClaim()` should verify that hash of `params` matches the value stored in `collateralIdToAuction` mapping. This validates that `params.endTime` is not spoofed.

Astaria: Fixed in PR 210.

Spearbit: Verified.

5.2.7 Typed structured data hash used for signing commitments is calculated incorrectly

Severity: High Risk

Context:

- VaultImplementation.sol#L150-L151
- VaultImplementation.sol#L172-L176
- IVaultImplementation.sol#L41

Description: Since

```
STRATEGY_TYPEHASH == keccak256("StrategyDetails(uint256 nonce,uint256 deadline,bytes32 root)")
```

The hash calculated in `_encodeStrategyData` is incorrect according to EIP-712. `s.strategistNonce` is of type `uint32` and the `nonce` type used in the type hash is `uint256`.

Also the struct name used in the typehash collides with `StrategyDetails` struct name defined as:

```
struct StrategyDetails {
  uint8 version;
  uint256 deadline;
  address vault;
}
```

Recommendation: We suggest the followings:

1. Update the `STRATEGY_TYPEHASH` to reflect the correct type `uint32` for the `nonce`.
2. Keep the `STRATEGY_TYPEHASH` using the non-inlined version below since the compiler would inline the value off-chain:

```
bytes32 public constant STRATEGY_TYPEHASH = keccak256("StrategyDetails(uint32 nonce,uint256
↪  deadline,bytes32 root)");
```

3. To avoid name collision for the 2 structs, rename one of the `StrategyDetails` (even though one is not defined directly).

5.2.8 `makePayment` doesn't properly update stack, so most payments don't pay off debt

Severity: High Risk

Context: LienToken.sol#615-635

Description: As we loop through individual payment in `_makePayment`, each is called with:

```
(newStack, spent) = _payment(
    s,
    stack,
    uint8(i),
    totalCapitalAvailable,
    address(msg.sender)
);
```

This call returns the updated stack as `newStack` but then uses the function argument stack again in the next iteration of the loop.

The `newStack` value is unused until the final iterate, when it is passed along to `_updateCollateralStateHash()`. This means that the new state hash will be the original state with only the final loan repaid, even though all other loans have actually had payments made against them.

Recommendation:

```
  uint256 n = stack.length;
+ newStack = stack;
  for (uint256 i; i < n; ) {
    (newStack, spent) = _payment(
      s,
-     stack,
+     newStack,
      uint8(i),
      totalCapitalAvailable,
      address(msg.sender)
  );
```

This fixes the issue above, but the solution must also take into account the fix for the loop within `_payment` outlined here in Issue 134.

If you follow the suggestion in that issue, then this function should return an extra value (`elementRemoved`) and use that to dictate whether the loop iterates forward, or remains at the same `i` for the next run.

The final result should look like:

```
function _makePayment(
    LienStorage storage s,
    Stack[] calldata stack,
    uint256 totalCapitalAvailable
) internal returns (Stack[] memory newStack, uint256 spent) {
    newStack = stack;
    bool elementRemoved = false;
    for (uint256 i; i < newStack.length; ) {
        (newStack, spent, elementRemoved) = _payment(
        s,
        newStack,
        uint8(i),
        totalCapitalAvailable,
        address(msg.sender)
     );
    totalCapitalAvailable -= spent;

     // if stack is updated, we need to stay at the current index
     // to process the new element on the same index.
     if (!elementRemoved) unchecked { ++i };

    _updateCollateralStateHash(s, stack[0].lien.collateralId, newStack);
}
```

Astaria: Checked if newStack changed length instead of returning an elementRemoved bool because of stack too deep error.

5.2.9 `_removeStackPosition()` always reverts

Severity: High Risk

Context: LienToken.sol#L823-L828

Description: `removeStackPosition()` always reverts since it calls `stack` array for an index beyond its length:

```
for (i; i < length; ) {
  unchecked {
    newStack[i] = stack[i + 1];
    ++i;
  }
}
```

Notice that for `i==length-1`, `stack[length]` is called. This reverts since `length` is the length of `stack` array.

Additionally, the intention is to delete the element from `stack` at index `position` and shift left the elements appearing after this index. However, an addition increment to the loop index `i` results in `newStack[position]` being empty, and the shift of other elements doesn't happen.

Recommendation: Apply this diff to LienToken.sol#L823-L831:

```
-  unchecked {
-    ++i;
-  }
-  for (i; i < length; ) {
+  for (i; i < length-1; ) {
   unchecked {
     newStack[i] = stack[i + 1];
     ++i;
   }
 }
```

Note: This issue has to be considered in conjunction with the following issue:

- `makePayment` doesn't properly update stack, so most payments don't pay off debt

Astaria: Fixed in PRs 202 and 265.

Spearbit: Verified.

5.2.10 Refactor `_paymentAH()`

Severity: High Risk

Context: LienToken.sol#L571

Description: `_paymentAH()` has several vulnerabilities:

- `stack` is a memory parameter. So all the updates made to `stack` are not applied back to the corresponding storage variable.
- No need to update `stack[position]` as it's deleted later.
- `decreaseEpochLienCount()` is always passed `0`, as `stack[position]` is already deleted. Also `decreaseEpochLienCount()` expects epoch, but `end` is passed instead.
- This if/else block can be merged. `updateAfterLiquidationPayment()` expects `msg.sender` to be `LIEN_TOKEN`, so this should work.

Recommendation: Apply this diff:

```
  function _paymentAH(
    LienStorage storage s,
    uint256 collateralId,
-   AuctionStack[] memory stack,
+   AuctionStack[] storage stack,
    uint256 position,
    uint256 payment,
    address payer
  ) internal returns (uint256) {
    uint256 lienId = stack[position].lienId;
    uint256 end = stack[position].end;
    uint256 owing = stack[position].amountOwed;
    //checks the lien exists
    address owner = ownerOf(lienId);
    address payee = _getPayee(s, lienId);

-   if (owing > payment.safeCastTo88()) {
-   stack[position].amountOwed -= payment.safeCastTo88();
-   } else {
+   if (owing < payment.safeCastTo88()) {
      payment = owing;
    }
    s.TRANSFER_PROXY.tokenTransferFrom(s.WETH, payer, payee, payment);

    delete s.lienMeta[lienId]; //full delete
    delete stack[position];
    _burn(lienId);

    if (_isPublicVault(s, payee)) {
-     if (owner == payee) {
        IPublicVault(payee).updateAfterLiquidationPayment(
          IPublicVault.LiquidationPaymentParams({lienEnd: end})
        );
-   } else {
-      IPublicVault(payee).decreaseEpochLienCount(stack[position].end);
-     }
```

```
    }
    emit Payment(lienId, payment);
    return payment;
  }
```

Also note other issues related to `_paymentAH()`:

- *Avoid shadowing variables*
- *Comment or remove unused function parameters*

Astaria: Fixed in PR 201.

Spearbit: Verified.

5.2.11 `processEpoch()` needs to be called regularly

Severity: High Risk

Context:

- PublicVault.sol#L247
- PublicVault.sol#L320

Description: If the `processEpoch()` endpoint does not get called regularly (especially close to the epoch boundaries), the updated `currentEpoch` would lag behind the actual expected value and this will introduce arithmetic errors in formulas regarding epochs and timestamps.

Recommendation: Thus public vaults need to create a mechanism so that the `processEpoch()` gets called regularly maybe using relayers or off-chain bots.

Also if there are any outstanding withdraw reserves, the vault needs to be topped up with assets (and/or the current withdraw proxy) so that the full amount of withdraw reserves can be transferred to the withdraw proxy from the epoch before using `transferWithdrawReserve`, otherwise, the processing of epoch would be halted. And if this halt continues more than one epoch length, the inaccuracy in the epoch number will be introduced in the system.

Another mechanism that can be introduced into the system is of incrementing the current epoch not just by one but by an amount depending on the amount of time passed since the last call to the `processEpoch()` or the timestamp of the current epoch.

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.2.12 Can create lien for collateral while at auction by passing spoofed data

Severity: High Risk

Context: LienToken.sol#L368-372

Description: In the `createLien` function, we check that the collateral isn't currently at auction before giving a lien with the following check:

```
if (
  s.collateralStateHash[params.collateralId] == bytes32("ACTIVE_AUCTION")
) {
  revert InvalidState(InvalidStates.COLLATERAL_AUCTION);
}
```

However, `collateralId` is passed in multiple places in the params: both in `params` directly and in `params.encumber.lien`.

The `params.encumber.lien.collateralId` is used everywhere else, and is the final value that is used. But the check is performed on `params.collateralId`.

As a result, we can set the following:

- `params.encumber.lien.collateralId`: collateral that is at auction.
- `params.collateralId`: collateral not at auction.

This will allow us to pass this validation while using the collateral at auction for the lien.

Recommendation:

```
    if (
-      s.collateralStateHash[params.collateralId] == bytes32("ACTIVE_AUCTION")
+      s.collateralStateHash[params.encumber.liencollateralId] == bytes32("ACTIVE_AUCTION")
    ) {
      revert InvalidState(InvalidStates.COLLATERAL_AUCTION);
    }
```

Astaria: We can remove collateralId entirely from the encumber call as its inside lien, the fix is to update to use the lien.collateralId everywhere vs encumber.collateralId

Spearbit: Agreed, that seems like the best fix and gets rid of an unneeded parameter. Confirmed the following PR 214 resolves the issue.

5.2.13 stateHash isn't updated by buyoutLien function

Severity: High Risk

Context: LienToken.sol#L102-187

Description: We never update the collateral state hash anywhere in the `buyoutLien` function. As a result, once all checks are passed, payment will be transferred from the buyer to the seller, but the seller will retain ownership of the lien in the system's state.

Recommendation: We should save the return value of the `_replaceStackAtPositionWithNewLien` function call and use it to call:

```
s.collateralStateHash[collateralId] = keccak256(abi.encode(newUpdatedStack));
```

Spearbit: Confirmed, the following commit fixes this issue.

5.2.14 If a collateral's liquidation auction on Seaport ends without a winning bid, the call to `liquidatorNFTClaim` does not clear the related data on `LienToken`'s side and also for `payee`s that are public vaults

Severity: High Risk

Context: CollateralToken.sol#L107

Description: If/when a liquidation auction ends without being fulfilled/matched on `Seaport` and afterward when the current `liquidator` calls into `liquidatorNFTClaim`, the storage data (`s.collateralStateHash`, `s.auctionData`, `s.lienMeta`) on the `LienToken` side don't get reset/cleared and also the lien token does not get burnt. That means:

- `s.collateralStateHash[collateralId]` stays equal to `bytes32("ACTIVE_AUCTION")`.
- `s.auctionData[collateralId]` will have the past auction data.
- `s.lienMeta[collateralId].atLiquidation` will be `true`.

That means future calls to `commitToLiens` by holders of the same collateral will revert.

Recommendation: Make sure to clear related storage data on `LienToken`'s side and `payee`s that are public vaults when `liquidatorNFTClaim` is called.

5.2.15 `ClearingHouse` cannot detect if a call from `Seaport` comes from a genuine listing or auction

Severity: High Risk

Context: ClearingHouse.sol#L21

Description: Anyone can create a `SeaPort` order with one of the considerations' recipients set to a `ClearingHouse` with a `collateralId` that is genuinely already set for auction. Once the spoofed order settles, `SeaPort` calls into this `fallback` function and causes the genuine Astaria auction to settle.

This allows an attacker to set random items on sale on `SeaPort` with funds directed here (small buying prices) to settle genuine Astaria auctions on the protocol.

This causes:

- The Astaria auction `payee`s and the `liquidator` would not receive what they would expect that should come from the auction. And if `payee` is a public vault it would introduce incorrect parameters into its system.
- Lien data (`s.lienMeta[lid]`) and the lien token get deleted/burnt.
- Collateral token and data get burnt/deleted.
- When the actual genuine auction settles and calls back to here, it will revert due to `s.collateralIdToAuction[collateralId]` check.

Recommendation: Astaria needs to introduce a mechanism so that `Seaport` would send more data to `ClearingHouse` to check the genuineness of the fallback calls.

Astaria: In a change yet to be merged, we have the `ClearingHouse` setup with checks to enforce that it has received enough of a payment in the right asset to complete the txn, we ultimately do not care where the txn came from as long as we are indeed offering the payment, and are getting everything that the auction should cost. Will mark it as acknowledged and tag this ticket with the updates when merged.

Spearbit: Acknowledged.

5.2.16 `c.lienRequest.strategy.vault` is not checked to be a registered vault when `commitToLiens` is called

Severity: High Risk

Context: AstariaRouter.sol#L680-L683

Description: From when `commitToLiens` is called till when we end up calling `IVaultImplementation(c.lienRequest.strategy.vault).commitToLien( ... )` and after the value of `c.lienRequest.strategy.vault` is not checked whether it is a registered vault within the system (by checking `s.vaults`). The caller can set this value to any address they would desire and potentially perform some unwanted actions.

For example, the user could spoof all the values in `commitments` so that the later dependant contracts' checks are skipped and lastly we end up transferring funds:

```
s.TRANSFER_PROXY.tokenTransferFrom(
  address(s.WETH),
  address(this), // <--- AstariaRouter
  address(msg.sender),
  totalBorrowed
);
```

Not that since all checks are skipped, the caller can also indirectly set `totalBorrowed` to any value they would desire. And so, if `AstariaRouter` would hold any `wETH` at any point in time. Anyone can craft a payload to `commitToLiens` to drain its `wETH` balance.

Recommendation: Check that the value of `s.vaults[c.lienRequest.strategy.vault]` is not `address(0)` before calling `c.lienRequest.strategy.vault`'s `commitToLien` endpoint.

Astaria: Solved in PR 197.

Spearbit: Verified.

5.2.17 Anyone can take a loan out on behalf of any collateral holder at any terms

Severity: High Risk

Context: VaultImplementation.sol#L225

Description: In the `_validateCommitment()` function, the initial checks are intended to ensure that the caller who is requesting the lien is someone who should have access to the collateral that it's being taken out against.

The caller also inputs a `receiver`, who will be receiving the lien. In this validation, this `receiver` is checked against the collateral holder, and the validation is approved in the case that `receiver == holder`.

However, this does not imply that the collateral holder wants to take this loan.

This opens the door to a malicious lender pushing unwanted loans on holders of collateral by calling `commitToLien` with their collateralId, as well as their address set to the receiver. This will pass the `receiver == holder` check and execute the loan.

In the best case, the borrower discovers this and quickly repays the loan, incurring a fee and small amount of interest. In the worst case, the borrower doesn't know this happens, and their collateral is liquidated.

Recommendation: Only allow calls from the holder or operator to lead to valid commitments:

```
    address holder = CT.ownerOf(collateralId);
    address operator = CT.getApproved(collateralId);

    if (
      msg.sender != holder &&
-     receiver != holder &&
-     receiver != operator &&
-     !ROUTER().isValidVault(receiver)
+     msg.sender != operator &&
+     CT.isApprovedForAll(holder, msg.sender)
    ) {
-     if (operator != address(0)) {
-       require(operator == receiver);
-     } else {
-       require(CT.isApprovedForAll(holder, receiver));
+    revert NotApprovedForBorrow();
      }
    }
```

5.2.18 Strategist Interest Rewards will be 10x higher than expected due to incorrect divisor

Severity: High Risk

Context: PublicVault.sol#L564

Description: `VAULT_FEE` is set as an immutable argument in the construction of new vaults, and is intended to be set in basis points. However, when the strategist interest rewards are calculated in `_handleStrategistInterestReward()`, the `VAULT_FEE` is only divided by 1000.

The result is that the fee calculated by the function will be 10x higher than expected, and the strategist will be dramatically overpaid.

Recommendation:

```
      unchecked {
-       uint256 fee = x.mulDivDown(VAULT_FEE(), 1000);
+       uint256 fee = x.mulDivDown(VAULT_FEE(), 10000);
        s.strategistUnclaimedShares += convertToShares(fee).safeCastTo88();
      }
```

Astaria: Resolved based on the following PR 203.

Spearbit: Verified.

5.2.19 The lower bound for `liquidationInitialAsk` for new lines needs to be stricter

Severity: High Risk

Context:

- LienToken.sol#L376-L381
- AstariaRouter.sol#L516

Description: `params.lien.details.liquidationInitialAsk` (L_{new}) is only compared to `params.amount` (A_{new}) whereas in `_appendStack` `newStack[j].lien.details.liquidationInitialAsk` (L_j) is compared to `potentialDebt`. `potentialDebt` is the aggregated sum of all potential owed amount at the end of each position/lien. So in `_appendStack` we have:

$$o_{new} + o_n + \cdots + o_j \leq L_j$$

Where o_j is `_getOwed(newStack[j], newStack[j].point.end)` which is the amount for the stack slot plus the potential interest at the end of its term.

So it would make sense to enforce a stricter inequality for L_{new}:

$$(1 + \frac{r(t_{end} - t_{now})}{10^{18}})A_{new} = o_{new} \leq L_{new}$$

The big issue regarding the current lower bound is when the borrower only takes one lien and for this lien `liquidationInitialAsk == amount` (or they are close). Then at any point during the lien term (maybe very close to the end), the borrower can atomically self `liquidate` and settle the `Seaport` auction in one transaction. This way the borrower can skip paying any interest (they would need to pay OpenSea fees and potentially royalty fees) and plus they would receive liquidation fees.

Recommendation: Make sure the following stricter lower bound is used instead:

$$(1 + \frac{r(t_{end} - t_{now})}{10^{18}})A_{new} = o_{new} \leq L_{new}$$

5.2.20 `commitToLiens` transfers extra assets to the borrower when protocol fee is present

Severity: High Risk

Context:

- AstariaRouter.sol#L417-L422
- VaultImplementation.sol#L392

Description: `totalBorrowed` is the sum of all `commitments[i].lienRequest.amount` But if `s.feeTo` is set, some of funds/assets from the vaults get transefered to `s.feeTo` when `_handleProtocolFee` is called and only the remaining is sent to the `ROUTER()`. So in this scenario, the total amount of assets sent to `ROUTER()` (so that it can be transferred to `msg.sender`) is up to rounding errors:

$$(1 - \frac{n_p}{d_p})T$$

Where:

- T is the `totalBorrowed`
- n_p is `s.protocolFeeNumerator`
- d_p is `s.protocolFeeDenominator`

But we are transferring T to `msg.sender` which is more than we are supposed to send,

Recommendation: Make sure only $(1 - \frac{n_p}{d_p})T$ is transfered to the borrower.

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.2.21 Withdraw proxy's `claim()` endpoint updates public vault's `yIntercept` incorrectly.

Severity: High Risk

Context:

- WithdrawProxy.sol#L235-L261
- WithdrawProxy.sol#L239

Description: Let

parameter	description
y_0	the `yIntercept` of our public vault in the question.
n	the current epoch for the public vault.
E_{n-1}	the `expected` storage parameter of the previous withdraw proxy.
B_{n-1}	the asset balance of the previous withdraw proxy.
W_{n-1}	the `withdrawReserveReceived` of the previous withdraw proxy.
S_{n-1}	the total supply of the previous withdraw proxy.
S_v	the total supply of the public vault when `processEpoch()` was last called on the public vault.
B_v	the total balance of the public vault when `processEpoch()` was last called on the public vault.
V	the public vault.

parameter	description
P_{n-1}	the previous withdraw proxy.

Then y_0 is updated/decremented according to the formula (up to rounding errors due to division):

$$y_0 = y_0 - \max(0, E_{n-1} - (B_{n-1} - W_{n-1}))(1 - \frac{S_{n-1}}{S_v})$$

Whereas the amount (A) of assets transfered from P_{n-1} to V is

$$A = (B_{n-1} - W_{n-1})(1 - \frac{S_{n-1}}{S_v})$$

And the amount (B) of asset left in P_{n-1} after this transfer would be:

$$B = W_{n-1} + (B_{n-1} - W_{n-1})\frac{S_{n-1}}{S_v})$$

$(B_{n-1} - W_{n-1})$ is supposed to represent the payment withdrawal proxy receives from Seaport auctions plus the amount of assets transferred to it by external actors. So A represents the portion of this amount for users who have not withdrawn from the public vault on the previous epoch and it is transferred to V and so y_0 should be compensated positively. Also note that this amount might be bigger than E_{n-1} if a lien has a really high `liquidationInitialAsk` and its auction fulfills/matches near that price on Seaport. So it is possible that $E_{n-1} < A$.

The current update formula for updating the y_0 has the following flaws:

- It only considers updating y_0 when $E_{n-1} - (B_{n-1} - W_{n-1}) > 0$ which is not always the case.
- Decrements y_0 by a portion of E_{n-1}.

The correct updating formula for y_0 should be:

$$y_0 = y_0 - E_{n-1} + (B_{n-1} - W_{n-1})(1 - \frac{S_{n-1}}{S_v})$$

Also note, if we let $B_{n-1} - W_{n-1} = X_{n-1} + \epsilon$, where X_{n-1} is the payment received by the withdraw proxy from Seaport auction payments and ϵ (if W_{n-1} updated correctly) be assets received from external actors by the previous withdraw proxy. Then:

$$B = W_{n-1} + (X_{n-1} + \epsilon)\frac{S_{n-1}}{S_v}) = \left[\max(0, B_v - E_{n-1}) + X_{n-1} + \epsilon\right]\frac{S_{n-1}}{S_v})$$

The last equality comes from the fact that when the withdraw reserves is fully transferred from the public vault and the current withdraw proxy (if necessary) to the previous withdraw proxy the amount W_{n-1} would hold should be $\max(0, B_v - E_{n-1})\frac{S_{n-1}}{S_v})$.

Related Issue.

Recommendation: Make sure y_0 is updated in `claim()` according to the following formula:

$$y_0 = y_0 - E_{n-1} + (B_{n-1} - W_{n-1})(1 - \frac{S_{n-1}}{S_v})$$

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.2.22 Public vault's `yIntercept` is not updated when the full amount owed is not paid out by a Seaport auction.

Severity: High Risk

Context: LienToken.sol#L587

Description: When the full `amountOwed` for a lien is not paid out during the callback from Seaport to a collateral's `ClearingHouse` and if the `payee` is a public vault, we would need to decrement the `yIntercept`, otherwise the `payee.totalAssets()` would reflect a wrong value.

Recommendation: When the above scenario happens make sure to call `decreaseYIntercept` with the difference of `amountOwed` and the payment received from the Seaport auction sale.

Astaria: Solved by PR 219.

Spearbit: Verified.

5.2.23 `LienToken` `payee` not reset on transfer

Severity: High Risk

Context: LienToken.sol#L303-L313

Description: `payee` and `ownerOf` are detached in that owners may set `payee` and owner may transfer the `LienToken` to a new owner. `payee` does not reset on transfer.

Exploit scenario:

- Owner of a `LienToken` sets themselves as `payee`
- Owner of `LienToken` sells the lien to a new owner
- New owner does not update `payee`
- Payments go to address set by old owner

Recommendation: Reset payee on transfer.

```
  function transferFrom(
    address from,
    address to,
    uint256 id
  ) public override(ERC721, IERC721) {
    LienStorage storage s = _loadLienStorageSlot();
    if (s.lienMeta[id].atLiquidation) {
      revert InvalidState(InvalidStates.COLLATERAL_AUCTION);
    }
+   delete s.lienMeta[id].payee;
+   emit PayeeChanged(id, address(0));
    super.transferFrom(from, to, id);
  }
```

5.2.24 `WithdrawProxy` allows redemptions before `PublicVault` calls `transferWithdrawReserve`

Severity: High Risk

Context: WithdrawProxy.sol#L172-L175

Description: Anytime there is a withdraw pending (i.e. someone holds `WithdrawProxy` shares), shares may be redeemed so long as `totalAssets() > 0` and `s.finalAuctionEnd == 0`.

Under normal operating conditions `totalAssets()` becomes greater than `0` when then `PublicVault` calls `transferWithdrawReserve`.

`totalAssets()` can also be increased to a non zero value by anyone transferring WETH to the contract.

If this occurs and a user attempts to redeem, they will receive a smaller share than they are owed.

Exploit scenario:

- Depositor `redeems` from `PublicVault` and receives `WithdrawProxy` shares.
- Malicious actor deposits a small amount of WETH into the `WithdrawProxy`.
- Depositor accidentally `redeems`, or is tricked into `redeem`ing, from the `WithdrawProxy` while `totalAssets()` is smaller than it should be.
- `PublicVault` properly processes epoch and full `withdrawReserve` is sent to `WithdrawProxy`.
- All remaining holders of `WithdrawProxy` shares receive an outsized share as the previous shares we `redeem`ed for the incorrect value.

Recommendation:

- Option 1:

Consider being explicit in opening the `WithdrawProxy` for redemptions (`redeem`/`withdraw`) by requiring `s.withdrawReserveReceived` to be a non zero value:

```diff
- if (s.finalAuctionEnd != 0) {
+ if (s.finalAuctionEnd != 0 || s.withdrawReserveReceived == 0) {
  // if finalAuctionEnd is 0, no auctions were added
  revert InvalidState(InvalidStates.NOT_CLAIMED);
  }
```

Astaria notes there is a second scenario where funds are sent to the `WithdrawProxy`: auction payouts. For the above recommendation to be complete, auction payouts or claiming MUST also set `withdrawReserveReceived`.

- Option 2:

Instead of inferring when it is safe to withdraw based on `finalAuctionEnd` and `withdrawReserveReceived`, consider explicitly marking the withdraws as `open` when it is both safe to withdraw (i.e. expected funds deposited) and the vault has claimed its share.

5.3 Medium Risk

5.3.1 `Point.position` is not updated for `stack` slots in `_removeStackPosition`

Severity: Medium Risk

Context:

- LienToken.sol#L402
- LienToken.sol#L809

Description: In `_createLien`, when a new stack slot is created, the `newSlot.point.position` is set to `uint8(params.stack.length)` which would be its index in the `stack`.

When `_removeStackPosition` is called to remove a slot from the `stack` at index `position`, the `newStack[i].point.position` is not updated for indexes that are greater than `position` in the original `stack`.

Also `slot.point.position` is only used when we emit `AddLien` and `LienStackUpdated` events. In both of those cases, we could have used `params.stack.length`

Recommendation: If it is necessary to keep `slot.point.position` due to future upgrades, make sure to update `_removeStackPosition` so that it updates the positions as well.

Otherwise, `slot.point.position` can be removed.

Astaria: Issue is fixed in commit fa175c by removing the `slot.point.position`.

Spearbit: Verified.

5.3.2 `unchecked` may cause under/overflows

Severity: Medium Risk

Context: LienToken.sol#L424, LienToken.sol#L482, PublicVault.sol#L376, PublicVault.sol#L422, PublicVault.sol#L439, PublicVault.sol#L490, PublicVault.sol#L578, PublicVault.sol#L611, PublicVault.sol#L527, PublicVault.sol#L544, PublicVault.sol#L563, PublicVault.sol#L640, VaultImplementation.sol#L401, WithdrawProxy.sol#L254, WithdrawProxy.sol#L293

Description: `unchecked` should only be used when there is a guarantee of no underflows or overflows, or when they are taken into account. In absence of certainty, it's better to avoid `unchecked` to favor correctness over gas efficiency.

For instance, if by error, `protocolFeeNumerator` is set to be greater than `protocolFeeDenominator`, this block in `_handleProtocolFee()` will underflow:

```
unchecked {
  amount -= fee;
}
```

However, later this reverts due to the ERC20 transfer of an unusually high amount. This is just to demonstrate that unknown bugs can lead to under/overflows.

Recommendation: Reason about each `unchecked` and remove them in absence of absolute certainty of safety.

Astaria: Acknowledged. We'll put checks on setting protocol values to not cross unintended boundaries.

Spearbit: Acknowledged.

5.3.3 Multiple ERC4626Router and ERC4626RouterBase functions will always revert

Severity: Medium Risk

Context:

- ERC4626Router.sol#L49-58
- ERC4626RouterBase.sol#L47
- ERC4626RouterBase.sol#L60

Description: The intention of the `ERC4626Router.sol` functions is that they are approval-less ways to deposit and redeem:

> // For the below, no approval needed, assumes vault is already max approved

As long as the user has approved the TRANSFER_PROXY for WETH, this works for the `depositToVault` function:

- WETH is transferred from user to the router with `pullTokens`.
- The router approves the vault for the correct amount of WETH.
- vault.deposit() is called, which uses `safeTransferFrom` to transfer WETH from router into vault.

However, for the `redeemMax` function, it doesn't work:

- Approves the vault to spend the router's WETH.
- vault.redeem() is called, which tries to transfer vault tokens from the router to the vault, and then mints withdraw proxy tokens to the receiver.

This error happens assuming that the vault tokens would be burned, in which case the logic would work. But since they are transferred into the vault until the end of the epoch, we require approvals.

The same issue also exists in these two functions in ERC4626RouterBase.sol:

- `redeem()`: this is where the incorrect approval lives, so the same issue occurs when it is called directly.
- `withdraw()`: the same faulty approval exists in this function.

Recommendation: `redeemMax` should follow the same flow as deposit to make this work:

- `redeemMax` should `pullTokens` to pull the vault tokens from the user.
- The router should approve the vault to spend its own tokens, not WETH.
- Then we can call vault.redeem() and it will work as intended.

Both the `ERC4626RouterBase` functions should change the approval to be vault tokens rather than WETH:

```
- ERC20(vault.asset()).safeApprove(address(vault), amount);
+ vault.safeApprove(address(vault), amount);
```

5.3.4 UniV3 tokens with fees can bypass strategist checks

Severity: Medium Risk

Context: UNI_V3Validator.sol#L117-119

Description: Each UniV3 strategy includes a value for `fee` in `nlrDetails` that is used to constrain their strategy to UniV3 pools with matching fees.

This is enforced with the following check (where `details.fee` is the strategist's set fee, and `fee` is the fee returned from Uniswap):

```
if (details.fee != uint24(0) && fee != details.fee) {
  revert InvalidFee();
}
```

This means that if you set details.fee to 0, this check will pass, even if the real fee is greater than zero.

Recommendation: If this is the intended behavior and you would like strategists to have a number they can use to accept all fee levels, I would recommend choosing a number other than zero (since it's a realistic value that strategists may want to set fees for).

Otherwise, adjust the check as follows:

```
- if (details.fee != uint24(0) && fee != details.fee) {
+ if (fee != details.fee) {
  revert InvalidFee();
}
```

For more flexibility, you could also allow all fees lower than the strategist set fee to be acceptable:

```
- if (details.fee != uint24(0) && fee != details.fee) {
+ if (fee > details.fee) {
  revert InvalidFee();
}
```

5.3.5 If auction time is reduced, withdrawProxy can lock funds from final auctions

Severity: Medium Risk

Context: WithdrawProxy.sol#L295

Description: When a new liquidation happens, the withdrawProxy sets `s.finalAuctionEnd` to be equal to the new incoming auction end.

This will usually be fine, because new auctions start later than old auctions, and they all have the same length.

However, if the auction time is reduced on the Router, it is possible for a new auction to have an end time that is sooner than an old auction. The result will be that the WithdrawProxy is claimable before it should be, and then will lock and not allow anyone to claim the funds from the final auction.

Recommendation: Replace this with a check like:

```
uint40 auctionEnd = (block.timestamp + finalAuctionDelta).safeCastTo40();
if (auctionEnd > s.finalAuctionEnd) s.finalAuctionEnd = auctionEnd;
```

Astaria: Fixed in commit 050487.

Spearbit: Verified.

5.3.6 claim() will underflow and revert for all tokens without 18 decimals

Severity: Medium Risk

Context: WithdrawProxy.sol#238-244

Description: In the `claim()` function, the amount to decrease the Y intercept of the vault is calculated as:

```
(s.expected - balance).mulWadDown(10**ERC20(asset()).decimals() - s.withdrawRatio)
```

`s.withdrawRatio` is represented as a WAD (18 decimals). As a result, using any token with a number of decimals under 17 (assuming the withdraw ratio is greater than 10%) will lead to an underflow and cause the function to revert.

In this situation, the token's decimals don't matter. They are captured in the `s.expected` and `balance`, and are also the scale at which the vault's y-intercept is measured, so there's no need to adjust for them.

Note: I know this isn't a risk in the current implementation, since it's WETH only, but since you are planning to generalize to accept all ERC20s, this is important.

Recommendation:

```
  if (balance < s.expected) {
    PublicVault(VAULT()).decreaseYIntercept(
      (s.expected - balance).mulWadDown(
-       10**ERC20(asset()).decimals() - s.withdrawRatio
+       1e18 - s.withdrawRatio
      )
    );
  }
```

5.3.7 Call to Royalty Engine can block NFT auction

Severity: Medium Risk

Context: CollateralToken.sol#L481

Description: `_generateValidOrderParameters()` calls `ROYALTY_ENGINE.getRoyaltyView()` twice. The first call is wrapped in a try/catch. This lets Astaria to continue even if the `getRoyaltyView()` reverts. However, the second call is not safe from this.

Both these calls have the same parameters passed to it except the price (`startingPrice` vs `endingPrice`). In case they are different, there exists a possibility that the second call can revert.

Recommendation: Wrap the second call in a try/catch. In case of a revert, the execution will be transferred to an empty `catch` block. Here is a sample:

```
if (foundRecipients.length > 0) {
  try
    s.ROYALTY_ENGINE.getRoyaltyView(
      underlying.tokenContract,
      underlying.tokenId,
      endingPrice
    ) returns (, uint256[] memory foundEndAmounts) {
    recipients = foundRecipients;
    royaltyStartingAmounts = foundAmounts;
    royaltyEndingAmounts = foundEndAmounts;
  } catch {}
}
```

Astaria: Acknowledged. We have a change pending that removes the royalty engine as apart of multi token.

Spearbit: Acknowledged.

5.3.8 Expired liens taken from public vaults need to be liquidated otherwise processing an epoch halts/reverts

Severity: Medium Risk

Context: PublicVault.sol#L275-L277

Description: `s.epochData[s.currentEpoch].liensOpenForEpoch` is decremented or is supposed to be decremented, when for a lien with an end that falls on this epoch:

- The full payment has been made,
- Or the lien is bought out by a lien that is from a different vault or ends at a higher epoch,
- Or the lien is liquidated.

If for some reason a lien expires and no one calls `liquidate`, then `s.epochData[s.currentEpoch].liensOpenForEpoch > 0` will be true and `processEpoch()` would revert till someones calls `liquidate`.

Note that a lien's end falling in the `s.currentEpoch` and `timeToEpochEnd() == 0` imply that the lien is expired.

Recommendation: Astaria would need to have a monitoring solution setup to make sure the `liquidate` endpoint gets called for expired liens without delay.

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.3.9 `assets < s.depositCap` invariant can be broken for public vaults with non-zero deposit caps

Severity: Medium Risk

Context:

- PublicVault.sol#L207-L208
- PublicVault.sol#L231-L232

Description: The following check in `mint` / `deposit` does not take into consideration the new `shares` / `amount` supplied to the endpoint, since the `yIntercept` in `totalAssets()` is only updated after calling `super.mint(shares, receiver)` or `super.deposit(amount, receiver)` with the `afterDeposit` hook.

```
uint256 assets = totalAssets();
if (s.depositCap != 0 && assets >= s.depositCap) {
  revert InvalidState(InvalidStates.DEPOSIT_CAP_EXCEEDED);
}
```

Thus the new `shares` or `amount` provided can be a really big number compared to `s.depositCap`, but the call will still go through.

Recommendation: To have the inequality `assets < s.depositCap` to be always correct, we would need to calculate the to be updated value of `assets` beforehand and then perform the check.

5.3.10 `redeemFutureEpoch` transfers the shares from the `msg.sender` to the vault instead of from the `owner`

Severity: Medium Risk

Context: PublicVault.sol#L143

Description: `redeemFutureEpoch` transfers the vault shares from the `msg.sender` to the vault instead of from the `owner`.

Recommendation: The 1st parameter passed to the `ERC20(address(this)).safeTransferFrom` needs to be the `owner`:

```
- ERC20(address(this)).safeTransferFrom(msg.sender, address(this), shares);
+ ERC20(address(this)).safeTransferFrom(owner, address(this), shares);
```

Astaria: Fixed in `443b0e01263755a64c98e3554b43a8fbfa1de215`.

Spearbit: Verified.

5.3.11 Lien buyouts can push maxPotentialDebt over the limit

Severity: Medium Risk

Context: LienToken.sol#L143-148

Description: When a lien is bought out, `_buyoutLien` calls `_getMaxPotentialDebtForCollateral` to confirm that this number is lower than the `maxPotentialDebt` specified in the lien.

However, this function is called with the existing stack, which hasn't yet replaced the lien with the new, bought out lien.

Valid refinances can make the rate lower or the time longer. In the case that a lien was bought out for a longer duration, `maxPotentialDebt` will increase and could go over the limit specified in the lien.

Recommendation: Perform this check after the old lien has been replaced by the new lien in the stack.

Astaria: Fixed in PR 211.

Spearbit: Verified.

5.3.12 Liens cannot be bought out once we've reached the maximum number of active liens on one collateral

Severity: Medium Risk

Context: LienToken.sol#L373-375

Description: The `buyoutLien` function is intended to transfer ownership of a lien from one user to another. In practice, it creates a new lien by calling `_createLien` and then calls `_replaceStackAtPositionWithNewLien` to update the stack.

In the `_createLien` function, there is a check to ensure we don't take out more than `maxLiens` against one piece of collateral:

```
if (params.stack.length >= s.maxLiens) {
  revert InvalidState(InvalidStates.MAX_LIENS);
}
```

The result is that, when we already have `maxLiens` and we try to buy one out, this function will revert.

Recommendation: Move this check from `_createLien` into the `_appendStack` function, which is only called when new liens are created rather than when they are bought out.

Astaria: Fixed in PR 213.

Spearbit: Verified.

5.3.13 First vault deposit can cause excessive rounding

Severity: Medium Risk

Context: ERC4626-Cloned.sol#L130

Description: Aside from storage layout/getters, the context above notes the other major departure from Solmate's ERC4626 implementation. The modification requires the initial `mint` to cost 10 full WETH.

```
  function mint(
    uint256 shares,
    address receiver
  ) public virtual returns (uint256 assets) {
+   // assets is 10e18, or 10 WETH, whenever totalSupply() == 0
    assets = previewMint(shares); // No need to check for rounding error, previewMint rounds up.

    // Need to transfer before minting or ERC777s could reenter.
+   // minter transfers 10 WETH to the vault
    ERC20(asset()).safeTransferFrom(msg.sender, address(this), assets);

+   // shares received are based on user input
    _mint(receiver, shares);

    emit Deposit(msg.sender, receiver, assets, shares);

    afterDeposit(assets, shares);
  }
```

Astaria highlighted that the code diff from Solmate is in relation to this finding from the previous Sherlock audit.

However, `deposit` is still unchanged and the initial deposit may be 1 wei worth of WETH, in return for 1 wad worth of vault shares.

Further, the previously cited issue may still surface by calling `mint` in a way that sets the price per share high (e.g. 10 shares for 10 WETH produces a price per of 1:1e18). Albeit, at a higher cost to the minter to set the initial price that high.

Recommendation: Revert the hardcoding of 10e18 in `previewMint` and `previewWithdraw`, this will require the first `minting` to be 1:1 asset to share price.

Prevent share price manipulation, add a condition in each of `mint` and `deposit` reverting if assets (when `depositing`) or shares (when `minting`) not above the minimum asset amount when `totalSupply() == 0`.

This comes at the cost of a duplicate storage read.

For WETH vaults, minimum asset amount for initial deposit can be a small amount, such as `100 gwei` so long as shares are issued 1:1 for the first mint/deposit.

5.3.14 When the collateral is listed on `SeaPort` by the borrower using `listForSaleOnSeaport`, when settled the liquidation fee will be sent to `address(0)`

Severity: Medium Risk

Context: LienToken.sol#L472-L477

Description: When the collateral is listed on `SeaPort` by the borrower using `listForSaleOnSeaport`, `s.auctionData[collateralId].liquidator` (`s.auctionData` in general) will not be set and so it will be `address(0)` and thus the `liquidatorPayment` will be sent to `address(0)`.

Recommendation: Before calculating and transferring the liquidation fee make sure that the `liquidator` is not `address(0)`.

Astaria: Fixed in PR 206.

Spearbit: Verified.

5.3.15 `potentialDebt` is not compared against a new lien's `maxPotentialDebt` in `_appendStack`

Severity: Medium Risk

Context: LienToken.sol#L435-L439

Description: In `_appendStack`, we have the following block:

```
newStack = new Stack[](stack.length + 1);
newStack[stack.length] = newSlot;

uint256 potentialDebt = _getOwed(newSlot, newSlot.point.end);
...
if (
  stack.length > 0 && potentialDebt > newSlot.lien.details.maxPotentialDebt
) {
  revert InvalidState(InvalidStates.DEBT_LIMIT);
}
```

Note, we are only performing a comparison between `newSlot.lien.details.maxPotentialDebt` and `potentialDebt` when `stack.length > 0`. If `_createLien` is called with `params.stack.length == 0`, we would not perform this check and thus the input `params` is not fully checked for misconfiguration.

Recommendation: Make sure to perform this check in either `_createLien` or here in `_appendStack` by removing the `stack.length > 0` condition:

```
// `potentialDebt` needs to be calculated in `_createLien`
if ( potentialDebt > params.lien.details.maxPotentialDebt ) {
  revert InvalidState(InvalidStates.DEBT_LIMIT);
}
```

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.3.16 Previous withdraw proxy's `withdrawReserveReceived` is not updated when assets are drained from the current withdraw proxy to the previous

Severity: Medium Risk

Context: PublicVault.sol#L378-L381

Description: When `drain` is called, we don't update the `s.epochData[s.currentEpoch - 1]`'s `withdrawReserveReceived`, this is in contrast to when withdraw reserves are transferred from the public vault to the withdraw proxy. This would unlink the previous withdraw proxy's `withdrawReserveReceived` storage parameter to the total amount of assets it has received from either the public vault or the current withdraw proxy.

An actor can manipulate $B_{n-1} - W_{n-1}$'s value by sending assets to the public vault and the current withdraw proxy before calling `transferWithdrawReserve` (B_{n-1} is the previous withdraw proxy's asset balance, W_{n-1} is previous withdraw proxy's `withdrawReserveReceived` and n is public vault's epoch). $B_{n-1} - W_{n-1}$ should really represent the sum of all near-boundary auction payment's the previous withdraw proxy receives plus any assets that are transferred to it by an external actor.

Related Issue 46.

Recommendation: The current behavior of draining assets from the current withdraw proxy to the previous is inconsistent compared to when assets are transferred from the public vault to the previous withdraw proxy which:

- Updates the public vault's `s.withdrawReserve`.
- Transfers the assets.
- Updates the previous withdraw proxy's `withdrawReserveReceived`.

In the case of the `drain` the first two points are performed but the last one is missing. Based on the behavior and other calculations it seems that `withdrawReserveReceived` would also need to be updated.

5.3.17 Update solc version and use `unchecked` in Uniswap related libraries

Severity: Medium Risk

Context: FullMathUniswap.sol, LiquidityAmounts.sol, TickMath.sol

Description: The highlighted libraries above are referenced from Uniswap codebase which is intended to work with Solidity compiler `<0.8`. These older versions have unchecked arithmetic by default and the code takes it into account.

Astaria code is intended to work with Solidity compiler >=0.8 which doesn't have unchecked arithmetic by default. Hence, to port the code, it has to be turned on via `unchecked` keyword.

For example, `FullMathUniswap.mulDiv(type(uint).max, type(uint).max, type(uint).max)` reverts for v0.8, and returns `type(uint).max` for older version.

Recommendation:

- Update the pragma of all the three files to:

```
pragma solidity ^0.8.4;
```

- Wrap all the function bodies in `unchecked`.

Astaria: Fixed in PR 9.

Spearbit: Verified.

5.4 Low Risk

5.4.1 `buyoutLien` is prone to race conditions

Severity: Low Risk

Context:

- LienToken.sol#L102
- VaultImplementation.sol#L305

Description: `LienToken.buyoutLien` and `VaultImplementation.buyoutLien` are both prone to race conditions where multiple vaults can try to front-run each others' `buyoutLien` call to end up registering their own lien.

Also note, due to the storage values `s.minInterestBPS` and `s.minDurationIncrease` being used in the `isValidRefinance`, the winning `buyoutLien` call does not necessarily have to have the best `rate` or `duration` among the other candidates in the race.

Recommendation: Make sure to document this. The current race between the vaults is not in an ideal condition for vaults and only sometimes in favor of picking the best liens for the borrower.

A better mechanism to avoid the race condition would be to introduce an auctioning process to buy out a lien and make sure the auction's picking strategy is sound.

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.4.2 `ERC20-Cloned` allows certain actions for `address(0)`

Severity: Low Risk

Context:

- ERC20-Cloned.sol#L39
- ERC20-Cloned.sol#L99
- ERC20-Cloned.sol#L76
- ERC20-Cloned.sol#L167
- ERC20-Cloned.sol#L180
- ERC20-Cloned.sol#L46

Description: In `ERC20-Cloned`, `address(0)` can be used as the:

- `spender` (`spender`)
- `to` parameter of `transferFrom`.
- `to` parameter of `transfer`.
- `to` parameter of `_mint`.
- `from` parameter of `_burn`.

As an example, one can `transfer` or `transferFrom` to `address(0)` which would turn the amount of tokens unusable but those not update the total supply in contrast to if `_burn` was called.

Recommendation: The decision to not check these addresses to make sure they cannot be assigned to `address(0)` is against the OpenZepplin implementation of ERC20. It is recommended to have these checks to avoid introducing quirks regarding `address(0)`.

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.4.3 `BEACON_PROXY_IMPLEMENTATION` and `WETH` cannot be updated for `AstariaRouter`

Severity: Low Risk

Context:

- IAstariaRouter.sol#L67
- IAstariaRouter.sol#L72

Description: There is no update mechanism for `BEACON_PROXY_IMPLEMENTATION` and `WETH` in `AstariaRouter`. It would make sense that one would want to keep `WETH` as not upgradable (unless we provide the wrong address to the constructor). But for `BEACON_PROXY_IMPLEMENTATION` there could be possibilities of potentially upgrading it.

Recommendation: If there is no plan to add an upgrading mechanism to these storage parameters, they can be defined as immutables. Also if there is a plan to use a diamond/facet pattern for the `AstariaRouter` and related contract, it might be best to document it so that it is more clear the reasoning for the current storage structures.

And finally, other implementation parameters have been made upgradable, it would make sense to also have `BEACON_PROXY_IMPLEMENTATION` to be upgradable or document why it is not so currently.

Astaria: `wETH` being a storage variable is removed in an open PR, so will acknowledge the lack of a setter, it came from a previously immutable design, the beacon proxy is not designed to be updated either in its current form, as any changes to the underlying or new features would leave older proxies in the dust.

Spearbit: Acknowledged by Astaria.

5.4.4 Incorrect key parameter type is used for `s.epochData`

Severity: Low Risk

Context: PublicVault.sol#/*

Description: In `PublicVault`, whenever the `epoch` key provided is to the mapping `s.epochData` its type is `uint64`, but the type of `s.epochData` is `mapping(uint256 => EpochData)`

Recommendation: Since the `epoch`s have `uint64` type, it would be best to define `VaultData` as:

```
struct VaultData {
  uint88 yIntercept;
  uint48 slope;
  uint40 last;
  uint64 currentEpoch; // <-- pay attention to the type of epochs
  uint88 withdrawReserve;
  uint88 liquidationWithdrawRatio;
  uint88 strategistUnclaimedShares;
  mapping(uint64 => EpochData) epochData; // <-- changed line
}
```

5.4.5 `buyoutLien`, `canLiquidate` and `makePayment` have different notion of expired liens when considering edge cases

Severity: Low Risk

Context:

- VaultImplementation.sol#L305
- LienToken.sol#L731
- AstariaRouter.sol#L509

Description: When swapping a lien that is just expired (lien's end t_{end} equals to the current timestamp t_{now}), one can call `buyoutLien` to swap it out. But when $t_{now} > t_{end}$, `buyoutLien` reverts due to the underflow in `_getRemainingInterest` when calculating the buyout amount. This is in contrast to `canLiquidate` which allows a lien with $t_{now} = t_{end}$ to `liquidate` as well.

`makePayment` also only considers $t_{end} < t_{now}$ as expired liens.

So the expired/non-functional liens time ranges for different endpoints are:

endpoint	expired range
`buyoutLien`	(t_{end}, ∞)
`canLiquidate`	$[t_{end}, \infty)$
`makePayment`	(t_{end}, ∞)

Recommendation: Make sure the edge case of $t_{now} = t_{end}$ is treated consistently for expired liens across the 3 endpoints in this **context**.

Astaria: All the ranges have been unified to consider $[t_{end}, \infty)$ as the expired range in commit 36ceb.

Spearbit: Verified.

5.4.6 Ensure all ratios are less than 1

Severity: Low Risk

Context: AstariaRouter.sol#L212-L227

Description: Although, numerators and denominators for different fees are set by admin, it's a good practice to add a check in the contract for absurd values. In this case, that would be when numerator is greater than denominator.

Recommendation: Add a `require` check for each numerator highlighted in **Context**:

```
require(numerator < denominator, "MAX_FEE_EXCEEDED");
```

You can also use custom errors instead.

Astaria: Fixed in commits b43317 and a883a3.

Spearbit: Verified.

5.4.7 Factor out `s.slope` updates

Severity: Low Risk

Context:

- PublicVault.sol#L422
- PublicVault.sol#L491
- PublicVault.sol#L528
- PublicVault.sol#L579
- PublicVault.sol#L615

Description: Slope updates occur in multiple locations but do not emit events.

Recommendation: Emit an event when updating slope. For ease of testing consider moving slope updates to an internal function and emit an event when called.

5.4.8 External `call` to arbitrary address

Severity: Low Risk

Context: AstariaRouter.commitToLiens, AstariaRouter. _executeCommitment

Description: The Router has a convenience function to commit to multiple liens `AstariaRouter.commitToLiens`. This function causes the router to receive WETH and allows the caller to supply an arbitrary vault address `lienRequest.strategy.vault` which is called by the router.

This allows the potential for the caller to re-enter in the middle of the loop, and also allows them to drain any WETH that happens to be in the Router.

In our review, no immediate reason for the Router to have WETH outside of `commitToLiens` calls was identified and therefore the severity of this finding is low.

Recommendation: To protect against potential malicious calls, `isValidVault` should be checked against any calls to vaults.

5.4.9 Astaria's Seaport orders may not be listed on OpenSea

Severity: Low Risk

Context: CollateralToken.sol#L524-L530

Description: To list Seaport orders on OpenSea, the order should pass certain validations as described here(see *OpenSea Order Validation*). Currently, Astaria orders will fail this validation. For instance, `zone` and `zoneHash` values are not set as suggested.

Recommendation: Either follow the guidelines completely to list the orders on Opensea. If that's not the intention, `OS_FEE_PAYEE` can be removed from consideration items.

Astaria: Acknowledged. We're going to change our approach and wait for seaport 1.2.

Spearbit: Acknowledged.

5.4.10 Any ERC20 held in the Router can be stolen using ERC4626RouterBase functions

Severity: Low Risk

Context: ERC4626RouterBase.sol#L15-65

Description: All four functions in ERC4626RouterBase.sol take in a `vault` address, a `to` address, a `shares` amount, and a `maxAmountIn` for validation. The first step is to read `vault.asset()` and then approve the vault to spend the ERC20 at whatever address is returned for the given amount.

```
function mint(
  IERC4626 vault,
  address to,
  uint256 shares,
  uint256 maxAmountIn
) public payable virtual override returns (uint256 amountIn) {
  ERC20(vault.asset()).safeApprove(address(vault), shares);
  if ((amountIn = vault.mint(shares, to)) > maxAmountIn) {
    revert MaxAmountError();
  }
}
```

In the event that the Router holds any ERC20, a malicious user can design a contract with the following functions:

```
function asset() view pure returns (address) {
  return [ERC20 the router holds];
}

function mint(uint shares, address to) view pure returns (uint) {
  return 0;
}
```

If this contract is passed as the vault, the function will pass, and the router will approve this contract to control its holdings of the given ERC20.

Recommendation: These functions should validate that the vault being passed into the contract is a legitimate Astaria vault.

Astaria: Accepted reccomendations, fixed in PR 253.

Spearbit: Confirmed, the following commit fixes this issue by overriding the functions and adding a `validVault()` modifier to them, so only valid vaults can be used in the function.

5.4.11 Inconsistency in byte size of maxInterestRate

Severity: Low Risk

Context: AstariaRouter.sol#L246

Description: In RouterStorage, `maxInterestRate` has a size of uint88. However, when being set from `file()`, it is capped at uint48 by the `safeCastTo48()` function.

Recommendation: Make sure these two values align.

Astaria: Fixed in PR 246.

Spearbit: Verified.

5.4.12 Router#file has update for nonexistent MinInterestRate variable

Severity: Low Risk

Context: AstariaRouter.sol#L247-248

Description: One of the options in the `file()` function is to update `FileType.MinInterestRate`. There are two problems here:

1) If someone chooses this FileType, the update actually happens to `s.maxInterestRate`.
2) There is no `minInterestRate` storage variable, as `minInterestBPS` is handled on L235-236.

Recommendation: Remove the `else if` block on L247-248.

Astaria: Fixed in PR 230.

Spearbit: Verified.

5.4.13 `getLiquidationWithdrawRatio()` and `getYIntercept()` have incorrect return types

Severity: Low Risk

Context:

- PublicVault.sol#L170-L172
- PublicVault.sol#L174-L176

Description: `liquidationWithdrawRatio` and `yIntercept` like other amount-related parameters are of type `uint88` (`uint88`) and they are the returned values of `getLiquidationWithdrawRatio()` and `getYIntercept()` respectively. But the return type of `getLiquidationWithdrawRatio()` and `getYIntercept()` are defined as `uint256`.

Recommendation: Change the return types of `getLiquidationWithdrawRatio()` and `getYIntercept()` to `uint88`:

```
function getLiquidationWithdrawRatio() public view returns (uint88) {
  return _loadStorageSlot().liquidationWithdrawRatio;
}
```

```
function getYIntercept() public view returns (uint88) {
  return _loadStorageSlot().yIntercept;
}
```

5.4.14 The modified implementation of `redeem` is omitting a check to make sure not to redeem `0` assets.

Severity: Low Risk

Context:

- PublicVault.sol#L108
- PublicVault.sol#L141

Description: The modified implementation of `redeem` is omitting the check

```
// Check for rounding error since we round down in previewRedeem.
require((assets = previewRedeem(shares)) != 0, "ZERO_ASSETS");
```

You can see a trail of it in `redeemFutureEpoch`.

Recommendation: It is recommended to introduce the check back to make sure that `0` assets are not allowed to be redeemed. Or document the decision as to why the check was omitted.

Astaria: Fixed in PR 227.

Spearbit: Verified.

5.4.15 `PublicVault`'s `redeem` and `redeemFutureEpoch` always returns `0` assets.

Severity: Low Risk

Context:

- PublicVault.sol#L133
- PublicVault.sol#L148
- PublicVault.sol#L114

Description: `assets` returned by `redeem` and `redeemFutureEpoch` will always be `0`, since it has not been set in `redeemFutureEpoch`. Also `Withdraw` event emits an incorrect value for `asset` because of this.

The issue stems from trying to consolidate some of the logic for `redeem` and `withdraw` by using `redeemFutureEpoch` for both of them.

Recommendation: Make sure the amount of `assets` is calculated for these endpoints and pay extra attention to the `2` different routing of `withdraw` and `redeem`.

Astaria: Fixed in PR 227.

Spearbit: Verified.

5.4.16 `OWNER()` cannot be updated for private or public vaults

Severity: Low Risk

Context: AstariaVaultBase.sol#L22-L24

Description: `owner()` is an immutable data for any `ClonesWithImmutableArgs.clone` that uses `AstariaVault-Base`. That means for example if there is an issue with the current hardcoded `owner()` there is no way to update it and liquidities/assets in the public/private vaults would also be at risk.

Recommendation: It might be best to allow change of ownership for vaults. The upgradeability for the owner might be more important than the router as mentioned in PR 107.

Astaria: Acknowledged. We don't want strategists to be able to assign someone else to their vaults, this is working as intended.

Spearbit: Acknowledged.

5.4.17 `ROUTER()` can not be updated for private or public vaults

Severity: Low Risk

Context: AstariaVaultBase.sol#L14-L16

Description: `ROUTER()` is an immutable data for any `ClonesWithImmutableArgs.clone` that uses `AstariaVault-Base`. That means for example if there is an issue with the current hardcoded `ROUTER()` or that it needs to be upgraded, the current public/private vaults would not be able to communicate with the new `ROUTER`.

Recommendation: This is something to keep in mind regarding the architecture of the protocol as the upgradability of the router can break the connection between it and the vaults.

Astaria: Acknowledged, this is working as intended. We have a planned update that makes LienToken/CollateralToken/AstariaRouter all upgradeable proxies.

Spearbit: Acknowledged.

5.4.18 Wrong return parameter type is used for `getOwed`

Severity: Low Risk

Context:

- LienToken.sol#L693
- LienToken.sol#L701

Description: Both variations of `getOwed` use `_getOwed` and return `uint192`. But `_getOwed` returns a `uint88`.

Recommendation: The return types of both `getOwed` variations need to be changed to `uint88`. Also, note that most parameters that deal with lien amounts have the `uint88` type.

5.4.19 Document and reason about which functionalities should be frozen on protocol pause

Severity: Low Risk

Context: PublicVault.sol#L247, PublicVault.sol#L338

Description: On protocol pause, a few functions are allowed to be called. Some instances are noted above. There is no documentation on why these functionalities are allowed while the remaining functions are frozen.

Recommendation: Some guidelines should be provided which specifies which functionalities should work when the protocol is paused.

Astaria: Acknowledged. There is no harm letting people withdraw their money through the epoch system, but there is immense harm in allowing for any deposits to come in or for new loans to go out, as the pause would be due to some emergency or something that requires a contract update.

When crystalizing the protocol we would remove the pause from the implementation before bricking upgrades.

Spearbit: Acknowledged.

5.4.20 Wrong parameter type is used for `s.strategyValidators`

Severity: Low Risk

Context:

- IAstariaRouter.sol#L82
- AstariaRouter.sol#L254
- AstariaRouter.sol#L345
- AstariaRouter.sol#L349

Description: `s.strategyValidators` is of type `mapping(uint32 => address)` but the provided `TYPE` in the **context** is of type `uint8`.

Recommendation: Make sure the typing is correct either use `mapping(uint32 => address)` or `mapping(uint8 => address)`

Astaria: Fixed by changing the type of `s.strategyValidators` to `mapping(uint8 => address)` in PR 241.

Spearbit: Verified.

5.4.21 Some functions do not emit events, but they should

Severity: Low Risk

Context:

- AstariaRouter.sol#L268
- VaultImplementation.sol#L195
- PublicVault.sol#L579
- PublicVault.sol#L528
- PublicVault.sol#L491
- PublicVault.sol#L565

Description: AstariaRouter.sol#L268 : Other filing endpoints in the same contract and also `CollateralToken` and `LienToken` emit `FileUpdated(what, data)`. But `fileGuardian` does not.

Recommendation: Make sure these endpoints emit events as some off-chain agents might be monitoring the protocol for these events.

Astaria: Solved in PR 240.

Spearbit: Verified.

5.4.22 `setNewGuardian` can be changed to a 2 or 3 step transfer of authority process

Severity: Low Risk

Context:

- AstariaRouter.sol#L262-L266
- AstariaRouter.sol#L268

Description: The current `guardian` might pass a wrong `_guardian` parameter to `setNewGuardian` which can break the upgradability of the `AstariaRouter` using `fileGuardian`.

Recommendation: It might be best to convert the transfer of guardianship into a 2 or 3 step process.

Astaria: This is intentional, we want to be able to remove all permissions if we decide to crystalize the protocol.

Spearbit: Renouncing the `guardian` can have its own separate endpoint possibly.

5.4.23 There are no range/value checks when some parameters get `file`ed

Severity: Low Risk

Context:

- AstariaRouter.sol#L196
- AstariaRouter.sol#L268
- CollateralToken.sol#L191
- LienToken.sol#L77

Description: There are no range/value checks when some parameters get `file`ed. For example:

- There are no hardcoded range checks for the `...Numerator`s and `...Denominator`s, so that the protocol's users can trustlessly assume the authorized users would not push these values into ranges seemed unacceptable.
- When an `address` get updated, we don't check whether the value provided is `address(0)` or not.

Recommendation: Apply value/range checks for the parameters that can be updated using the `file` endpoints.

5.4.24 Manually constructed storage slots can be chosen so that the pre-image of the hash is unknown

Severity: Low Risk

Context:

- AstariaRouter.sol#L54-L55
- LienToken.sol#L46-L48
- CollateralToken.sol#L69-L70
- PublicVault.sol#L58-L59
- VaultImplementation.sol#L46-L47
- WithdrawProxy.sol#L48-L49
- Pausable.sol#L20-L21
- ERC20-Cloned.sol#L14
- ERC721.sol#L13-L14

Description: In the codebase, some storage slots are manually constructed using `keccak256` hash of a string `xyz.astaria. ...`. The pre-images of these hashes are known. This can allow in future for actors to find a potential path to those storage slots using the `keccak256` hash function in the codebase and some crafted payload.

Recommendation:

1. Subtract `1` from these hashes so that the pre-image would be unknown / less obvious.
2. Keep them as it is withouting manually calculating the hash and inlining them as the compiler does that for us off-chain.

So in general:

```
uint256 private constant NAMED_SLOT = uint256(keccak256("xyz.astaria.<PATH>")) - 1
```

And for

- WithdrawProxy.sol#L48-L49
- Pausable.sol#L20-L21
- ERC20-Cloned.sol#L14

- ERC721.sol#L13-L14

specifically, what should be used:

```
bytes32 constant NAMED_SLOT = bytes32(uint256(keccak256("xyz.astaria.<PATH>")) - 1);
```

5.4.25 Avoid shadowing variables

Severity: Low Risk

Context: LienToken.sol#L583

Description: The highlighted line declares a new variable `owner` which has already been defined in `Auth.sol` inherited by `LienToken`:

```
 address owner = ownerOf(lienId);
```

Recommendation: Rename `owner` to `lienOwner` at LienToken.sol#L583.

Astaria: Fixed in PR 251.

Spearbit: Verified.

5.4.26 `PublicVault.accrue` is manually inlined rather than called

Severity: Low Risk

Context: PublicVault.sol#L438-L448, PublicVault.sol#L611-L617

Description: The `_accrue` function locks in the implied value of the `PublicVault` by calculating, then adding to yIntercept, and finally emitting an event.

This calculation is duplicated in 3 separate locations in `PublicVault`:

- In totalAssets
- In _accrue
- And in updateVaultAfterLiquidation

Recommendation: The calculation itself can be factored out into a view function:

Which can be reused in both `totalAssets` and `accrue`:

```
  function totalAssets()
    public
    view
    virtual
    override(ERC4626Cloned)
    returns (uint256)
  {
    VaultData storage s = _loadStorageSlot();
-   uint256 delta_t = block.timestamp - s.last;
-   return uint256(s.slope).mulDivDown(delta_t, 1) + uint256(s.yIntercept);
+   return _totalAssets(s);
  }
  function _accrue(VaultData storage s) internal returns (uint256) {
    unchecked {
-     s.yIntercept += uint256(block.timestamp - s.last)
-       .mulDivDown(uint256(s.slope), 1)
-       .safeCastTo88();
+     s.yIntercept = (_totalAssets(s)).safeCastTo88();
      s.last = block.timestamp.safeCastTo40();
    }
    emit YInterceptChanged(s.yIntercept);

    return s.yIntercept;
  }
```

and finally, accrue may be used in `updateVaultAfterLiquidation` so that emitting the event is not missed:

```
  function updateVaultAfterLiquidation(
    uint256 maxAuctionWindow,
    AfterLiquidationParams calldata params
  ) public returns (address withdrawProxyIfNearBoundary) {
    require(msg.sender == address(LIEN_TOKEN())); // can only be called by router
    VaultData storage s = _loadStorageSlot();

    unchecked {
-     s.yIntercept += uint256(s.slope)
-       .mulDivDown(block.timestamp - s.last, 1)
-       .safeCastTo88();
+     _accrue(s);
      s.slope -= params.lienSlope.safeCastTo48();
-     s.last = block.timestamp.safeCastTo40();
    }

    ...snip...

  }
```

5.4.27 `CollateralToken.flashAction` reverts with incorrect error

Severity: Low Risk

Context: CollateralToken.sol#L272

Description: Reverts with `InvalidCollateralStates.AUCTION_ACTIVE` when the address is not `flashEnabled`.

Recommendation: Revert using `InvalidCollateralStates.FLASH_DISABLED`:

```
  function flashAction(
    IFlashAction receiver,
    uint256 collateralId,
    bytes calldata data
  ) external onlyOwner(collateralId) {

    ...snip...
    if (!s.flashEnabled[addr]) {
-     revert InvalidCollateralState(InvalidCollateralStates.AUCTION_ACTIVE);
+     revert InvalidCollateralState(InvalidCollateralStates.FLASH_DISABLED);
    }
    ...snip...
  }
```

5.4.28 `AstariaRouter` has unnecessary access to `setPayee`

Severity: Low Risk

Context: LienToken.sol#L872

Description: `setPayee` is never called from `AstariaRouter`, but the router has access to call `LienToken.setPayee`.

Recommendation: Remove unneeded access:

```
 function setPayee(Lien calldata lien, address newPayee) public {

 ...snip...

   require(
-    msg.sender == ownerOf(lienId) || msg.sender == address(s.ASTARIA_ROUTER)
+    msg.sender == ownerOf(lienId)
   );

 ...snip...
 }
```

5.5 Gas Optimization

5.5.1 ClearingHouse can be deployed only when needed

Severity: Gas Optimization

Context: CollateralToken.sol#L632-640

Description: When collateral is deposited, a Clearing House is automatically deployed. However, these Clearing Houses are only needed if the collateral goes to auction at Seaport, either through liquidation or the collateral holder choosing to sell them.

The Astaria team has indicated that this behavior is intentional in order to put the cost on the borrower, since liquidations are already expensive.

I'd suggest the perspective that all pieces of collateral will be added to the system, but a much smaller percentage will ever be sent to Seaport. The aggregate gas spent will be much, much lower if we are careful to only deploy these contract as needed.

Further, let's look at the two situations where we may need a Clearing House:

1) The collateral holder calls `listForSaleOnSeaport()`: In this case, the borrower is paying anyways, so it's a no brainer.

2) Another user calls `liquidate()`: In this case, they will earn the liquidation fees, which should be sufficient to justify a small increase in gas costs.

Recommendation: Move the creation of the Clearing House to `liquidate()` and `listForSaleOnSeaport()` and remove it from `onERC721Received()`.

5.5.2 `PublicVault.claim()` can be optimized

Severity: Gas Optimization

Context:

- PublicVault.sol#L479
- PublicVault.sol#L483

Description: For `claim` not to revert we would need to have `msg.sender == owner()`. And so when the following is called:

```
_mint(owner(), unclaimed);
```

Instead of `owner()` we can use `msg.sender` since reading the immutable `owner()` requires some `calldata` lookup.

Recommendation: Use `msg.sender` instead of `owner()` when minting in `claim()`:

```
_mint(msg.sender, unclaimed);
```

5.5.3 Can remove incoming terms from LienActionBuyout struct

Severity: Gas Optimization

Context: ILienToken.sol#L88

Description: Incoming terms are never used in the LienActionBuyout struct. The general flow right now is:

- incomingTerms are passed to `VaultImplementation#buyoutLien`.
- These incoming terms are validated and used to generate the lien information.
- The lien information is encoded into a `LienActionBuyout` struct.
- This is passed to `LienToken#buyoutLien`, but then the incoming terms are never used again.

Recommendation: Pass the `incomingTerms` to `VaultImplementation#buyoutLien`, validate them, use them to generate the lien information, and then pass that to `LienToken#buyoutLien` without the incoming terms.

This will allow you to edit the LienActionBuyout struct to:

```
  struct LienActionBuyout {
-   IAstariaRouter.Commitment incoming;
    uint8 position;
    LienActionEncumber encumber;
 }
```

Astaria: The following commit fixes this issue: commit 9dcfc4.

Spearbit: Verified.

5.5.4 Refactor `updateVaultAfterLiquidation` to save gas

Severity: Gas Optimization

Context: PublicVault.sol#L604-637

Description: In `updateVaultAfterLiquidation`, we check if we're within `maxAuctionWindow` of the end of the epoch. If we are, we call `_deployWithdrawProxyIfNotDeployed` and assign `withdrawProxyIfNearBoundary` to the result.

We then proceed to check if `withdrawProxyIfNearBoundary` is assigned and, if it is, call `handleNewLiquidation`.

Instead of checking separately, we can include this call within the block of code executed if we're within `maxAuctionWindow` of the end of the epoch. This is true because (a) withdraw proxy will always be deployed by the end of that block and (b) withdraw proxy will never be deployed if timeToEnd >= maxAuctionWindow.

Recommendation:

```
    uint256 timeToEnd = timeToEpochEnd(lienEpoch);
    if (timeToEnd < maxAuctionWindow) {
      _deployWithdrawProxyIfNotDeployed(s, lienEpoch);
      withdrawProxyIfNearBoundary = s.epochData[lienEpoch].withdrawProxy;
-    }
-
-   if (withdrawProxyIfNearBoundary != address(0)) {
      WithdrawProxy(withdrawProxyIfNearBoundary).handleNewLiquidation(
        params.newAmount,
        maxAuctionWindow
      );
    }
```

Astaria: commit 1a64b3.

Spearbit: Verified.

5.5.5 Use `collateralId` to set `collateralIdToAuction` mapping

Severity: Gas Optimization

Context: CollateralToken.sol#L595

Description: `_listUnderlyingOnSeaport()` sets `collateralIdToAuction` mapping as follows:

```
s.collateralIdToAuction[uint256(listingOrder.parameters.zoneHash)] = true;
```

Since this function has access to `collateralId`, it can be used instead of using `zoneHash`.

Recommendation: Consider using `collateralId` to set `collateralIdToAuction`:

```
s.collateralIdToAuction[collateralId] = true;
```

Astaria: Fixed in commit 8fc32b.

Spearbit: Verified.

5.5.6 Storage packing

Severity: Gas Optimization

Context:

- IAstariaRouter.RouterStorage
- ILienToken.LienStorage
- IPublicVault.VaultData

Description: RouterStorage:

The `RouterStorage` struct represents state managed in storage by the `AstariaRouter` contract. Some of the packing in this struct is sub optimal.

1. `maxInterestRate` and `minInterestBPS`:

These two values pack into a single storage slot, however, are never referenced together outside of the constructor. This means, when read from storage, there are no gas efficiencies gained.

2. Comments denoting storage slots do not match implementation. The comment //slot 3 + for example occurs far after the 3rd slot begins as the addresses do not pack together.

LienStorage:

3. The `LienStorage` struct packs `maxLiens` with the `WETH` address into a single storage slot. While gas is saved on the constructor, extra gas is spent in parsing `maxLiens` on each read as it is read alone.

VaultData:

4. `VaultData` packs `currentEpoch` into the struct's first slot, however, it is more commonly read along with values from the struct's second slot.

Recommendation: Note, some recommendations incur a greater one time gas cost on write to net a lower cost on reads.

1. Pack `minInterestBPS` and `minDurationIncrease` together as they are both read in AstariaRouter.isValidRefinance

1b. Consider packing `maxInterestRate` with one of the addresses (`COLLATERAL_TOKEN` or `WETH`). This means both may be read in a single `sload` in `_validateCommitment`, however, does incur a smaller increase in gas when the stored address is referenced in other functions.

2. Update or removed the comments. If packing, updating is preferred.
3. Give `maxLiens` a type of `uint256`.
4. Consider moving `currentEpoch` to the second storage slot.

5.5.7 ClearingHouse fallback can save WETH address to memory to save gas

Severity: Gas Optimization

Context: ClearingHouse.sol#L21-34

Description: The fallback function reads `WETH()` from `ROUTER` three times. It would save gas to read the value once and save to memory for the future calls.

Recommendation:

```
  fallback() external payable {
    IAstariaRouter ASTARIA_ROUTER = IAstariaRouter(_getArgAddress(0));
    require(msg.sender == address(ASTARIA_ROUTER.COLLATERAL_TOKEN().SEAPORT()));
+   WETH weth = WETH(payable(address(ASTARIA_ROUTER.WETH())))
-   WETH(payable(address(ASTARIA_ROUTER.WETH()))).deposit{value: msg.value}();
+   weth.deposit{value: msg.value}();
-   uint256 payment = ASTARIA_ROUTER.WETH().balanceOf(address(this));
+   uint256 payment = weth.balanceOf(address(this));
-   ASTARIA_ROUTER.WETH().safeApprove(
+   weth.safeApprove(
      address(ASTARIA_ROUTER.TRANSFER_PROXY()),
      payment
    );
...
```

5.5.8 CollateralToken's onlyOwner modifier doesn't need to access storage

Severity: Gas Optimization

Context: CollateralToken.sol#254-259

Description: The `onlyOwner` modifier calls to `ownerOf()`, which loads storage itself to check ownership. We can save a storage load since we don't need to load the storage variables in the modifier itself.

Recommendation:

```
  modifier onlyOwner(uint256 collateralId) {
-   CollateralStorage storage s = _loadCollateralSlot();

    require(ownerOf(collateralId) == msg.sender);
    _;
  }
```

Astaria: Fixed in commit a52a05.

Spearbit: Verified.

5.5.9 Can stop loop early in _payDebt when everything is spent

Severity: Gas Optimization

Context: LienToken.sol#L480-488

Description: When a loan is sold on Seaport and `_payDebt` is called, it loops through the auction stack and calls `_paymentAH` for each, decrementing the remaining `payment` as money is spent.

This loop can be ended when `payment == 0`.

Recommendation:

```
    for (uint256 i = 0; i < stack.length; i++) {
+     if (payment == 0) break;
      uint256 spent;
      unchecked {
        spent = _paymentAH(s, collateralId, stack, i, payment, payer);
        totalSpent += spent;
        payment -= spent;
      }
    }
  }
```

Astaria: Fixed in commit 795c0c.

Spearbit: Verified.

5.5.10 Can remove initializing allowList and depositCap for private vaults

Severity: Gas Optimization

Context: AstariaRouter.sol#L653-660

Description: Private Vaults do not allow enabling, disabling, or editing the allow list, and don't enforce a deposit cap, so seems strange to initialize these variables.

Delegates are still included in the _validateCommitment function, so we can't get rid of this entirely.

Recommendation: Change the init function in the underlying vault to only set the delegate. Move the allowList and depositCap setup to a separate function that lives in `PublicVault.sol`, and is called separately in the event that a public vault is being created.

5.5.11 `ISecurityHook.getState` can be modified to return `bytes32` / hash of the state instead of the state itself.

Severity: Gas Optimization

Context: CollateralToken.sol#L304-L308

Description / Recommendation: Since only the `keccak256` of `preTransferState` is checked against the `keccak256` hash of the returned security hook state, we could change the design so that `ISecurityHook.getState` returns `bytes32` to save gas. Unless there is a plan to use the `bytes memory preTransferState` in some other form as well.

5.5.12 Define an endpoint for `LienToken` that only returns the `liquidator`

Severity: Gas Optimization

Context: CollateralToken.sol#L113

Description: It would save a lot of gas if `LienToken` had an endpoint that would only return the `liquidator` for a `collateralId`, instead of all the auction data.

Recommendation: An endpoint like the following can be defined for `LienToken`:

```
function getAuctionLiquidator(uint256 collateralId)
  external
  view
  returns (address liquidator)
{
  return _loadLienStorageSlot().auctionData[collateralId].liquidator;
}
```

`getAuctionLiquidator` perhaps can revert if `liquidator` is `address(0)`.

Also, note `getAuctionLiquidator` is more useful than `getAuctionData`. Since if above is implemented, `getAuctionData` could only potentially be used for off-chain purposes.

5.5.13 Setting uninitialized stack variables to their default value can be avoided.

Severity: Gas Optimization

Context: LienToken.sol#L687

Description: Setting uninitialized stack variables to their default value adds extra gas overhead.

```
T t = <DEFAULT_VALUE>;
```

Recommendation: Assignment of default value right after the declaration of the variable can be removed unless it is there for better readability (but that can also be included as a comment).

```
T t;
```

5.5.14 Simplify / optimize `for` loops

Severity: Gas Optimization

Context:

- LienToken.sol#L688-L690
- LienToken.sol#L816-L822
- AstariaRouter.sol#L191-L193
- AstariaRouter.sol#L272-L290
- AstariaRouter.sol#L405-L413
- CollateralToken.sol#L182-L184
- LienToken.sol#L258-L290
- LienToken.sol#L480-L487
- VaultImplementation.sol#L189-L191

Description: In the codebase, sometimes there are `for` loops of the form:

```
for (uint256 i = 0; <CONDITION>; i++) {
   <BODY>
}
```

These `for` loops can be optimized.

Recommendation: Transform the `for` loops mentioned into the following form:

```
uint256 i;
for (; <CONDITION>;) {
   <BODY>
   unchecked {
      ++i;
   }
}
```

5.5.15 `calculateSlope` can be more simplified

Severity: Gas Optimization

Context: LienToken.sol#L649-L656

Description: `calculateSlope` can be more simplified:

`owedAtEnd` would be:

$$\texttt{owedAtEnd} = \texttt{amt} + (t_{end} - t_{last}) r \frac{\texttt{amt}}{10^{18}}$$

where:

- `amt` is `stack.point.amount`
- t_{end} is `stack.point.end`
- t_{last} is `stack.point.last`
- r is `stack.lien.details.rate`

and so the returned value would need to be $r\frac{\texttt{amt}}{10^{18}}$.

Recommendation: The simplified version of `calculateSlope` would be:

```
function calculateSlope(Stack memory stack) public pure returns (uint256) {
  return
    stack.lien.details.rate.mulWadDown(
      stack.point.amount
    );
}
```

5.5.16 Break out of `_makePayment` `for` loop early when `totalCapitalAvailable` reaches `0`

Severity: Gas Optimization

Context: LienToken.sol#L629

Description: In `_makePayment` we have the following `for` loop:

```
for (uint256 i; i < n; ) {
  (newStack, spent) = _payment(
    s,
    stack,
    uint8(i),
    totalCapitalAvailable,
    address(msg.sender)
  );
  totalCapitalAvailable -= spent;
  unchecked {
    ++i;
  }
}
```

When `totalCapitalAvailable` reaches `0` we still call `_payment` which costs a lot of gas and it is only used for transferring `0` assets, removing and adding the same slope for a lien owner if it is a public vault and other noops.

Recommendation: Break out of the `for` loop when `totalCapitalAvailable == 0`.

5.5.17 `_buyoutLien` can be optimized by reusing `payee`

Severity: Gas Optimization

Context: LienToken.sol#L154-L164

Description: `payee` in `_buyoutLien` can be reused to save some gas

Recommendation: Define `payee` earlier in the `_buyoutLien` body and use this stack variable to avoid unnecessary computation:

```
address payee = _getPayee(
  s,
  params.encumber.stack[params.position].point.lienId
);

s.TRANSFER_PROXY.tokenTransferFrom(
  s.WETH,
  address(msg.sender),
  payee,  // <--- replaced value
  buyout
);
```

5.5.18 `isValidRefinance` and related storage parameters can be moved to `LienToken`

Severity: Gas Optimization

Context:

- LienToken.sol#L123-L127
- AstariaRouter.sol#L593
- IAstariaRouter.sol#L74
- IAstariaRouter.sol#L81

Description: `isValidRefinance` is only used in `LienToken` and with the current implementation it requires reading `AstariaRouter` from the storage and performing a cross-contract call which would add a lot of overhead gas cost.

Recommendation: We can move `isValidRefinance` function from `AstariaRouter` to `LienToken`. This would remove the need to first read the `ASTARIA_ROUTER` from storage and then call an external contract. It would also simplify the codebase.

This would also mean defining the storage variables `minInterestBPS` and `minDurationIncrease` here instead of in `AstariaRouter`. Note that both of these variables are used only in `isValidRefinance`, besides when filing to update them.

Astaria: We did this in an effort to concentrate protocol values into the same contract / space.

5.5.19 `auctionWindowMax` can be reused to optimize `liquidate`

Severity: Gas Optimization

Context: AstariaRouter.sol#L541

Description: There are mutiple instances of `s.auctionWindow + s.auctionWindowBuffer` in the `liquidate` function which would make the function to read from the storage twice each time. Also there is already a stack variable `auctionWindowMax` defined as the sum which can be reused.

Recommendation: Reuse `auctionWindowMax` for other extra instances of `s.auctionWindow + s.auctionWindowBuffer`.

```
  listedOrder = s.COLLATERAL_TOKEN.auctionVault(
    ICollateralToken.AuctionVaultParams({
      settlementToken: address(s.WETH),
      collateralId: stack[position].lien.collateralId,
-     maxDuration: uint256(s.auctionWindow + s.auctionWindowBuffer),
+     maxDuration: auctionWindowMax
      startingPrice: stack[0].lien.details.liquidationInitialAsk,
      endingPrice: 1_000 wei
    })
  );
```

5.5.20 `fileBatch()` does `requiresAuth` for each file separately

Severity: Gas Optimization

Context: CollateralToken.sol#L181-L191

Description: `fileBatch()` does a `requiresAuth` check and then for each element in the input array calls `file()` which does another `requiresAuth` check.

```
function fileBatch(File[] calldata files) external requiresAuth {
  for (uint256 i = 0; i < files.length; i++) {
    file(files[i]);
  }
}
...
function file(File calldata incoming) public requiresAuth {
```

This wastes gas as if the `fileBatch()`'s `requiresAuth` pass, `file()`'s check will pass too.

Recommendation: Create an internal function `_file()` without the check. Update `fileBatch()` and `file()` to call `_file()`.

Astaria: Fixed in PR 1931.

Spearbit: Verified.

5.5.21 `_sliceUint` can be optimized

Severity: Gas Optimization

Context: AstariaRouter.sol#L315

Description: `_sliceUint` can be optimized

Recommendation: Cheaper to use a named return parameter, cache calculating the end and use custom errors:

```
// cast --to-bytes32 $(cast sig "OutOfBoundError()")
uint256 private constant OUTOFBOUND_ERROR_SELECTOR =
↪  0x571e08d100000000000000000000000000000000000000000000000000000000;
uint256 private constant ONE_WORD = 0x20;
...
function _sliceUint(bytes memory bs, uint256 start)
  internal
  pure
  returns (uint256 x)
{
  uint256 length = bs.length;

  assembly {
    let end := add(ONE_WORD, start)

    if lt(length , end) {
      mstore(0, OUTOFBOUND_ERROR_SELECTOR)
      revert(0, ONE_WORD)
    }

    x := mload(add(bs, end))
  }
}
```

Also add unit/differential tests for this function.

5.5.22 Use basis points for ratios

Severity: Gas Optimization

Context: IAstariaRouter.sol#L61, IAstariaRouter.sol#L65, IAstariaRouter.sol#L78-L79

Description: Fee ratios are represented through two state variables for numerator and denominator. Basis point system can be used in its place as it is simpler (denominator always set to `10_000`), and gas efficient as denominator is now a constant.

Recommendation: Use basis point system to represent ratios. Remove denominator state variables and use `10_000` as a constant variable in its place.

5.5.23 No Need to Allocate Unused Variable

Severity: Gas Optimization

Context:

- LienToken.sol#L619-L622
- LienToken.sol#L553
- VaultImplementation.sol#L315

Description: `LienToken._makePayment()` returns two values: `(Stack[] memory newStack, uint256 spent)`, but the second value is never read:

```
(newStack, ) = _makePayment(_loadLienStorageSlot(), stack, amount);
```

Also, if this value is planned to be used in future, it's not a useful value. It is equal to the payment made to the last lien. A more meaningful quantity can be the total payment made to the entire stack.

Additional instances noted in `Context` above.

Recommendation: Only return `newStack` from `_makePayment()`.

5.5.24 Cache Values to Save Gas

Severity: Gas Optimization

Context:

1. CollateralToken.sol#L286-L307
2. ROUTER().LIEN_TOKEN() VaultImplementation.sol#L329
3. LienToken.sol#L511-L512
4. AstariaRouter.sol#L345
5. PublicVault.sol#L380

Description: Calls are occurring, same values are computed, or storage variables are being read, multiple times; e.g. `CollateralToken.sol#L286-L307` reads the storage variable `s.securityHooks[addr]` four times. It's better to cache the result in a stack variable to save gas.

Recommendation:

1. Cache `s.securityHooks[addr]` in a stack variable. Replace all the current usages of `s.securityHooks[addr]` with this new variable;
2. Cache `ROUTER().LIEN_TOKEN()` and reuse the value;
3. Compute `lienId` one time only;
4. Cache `s.strategyValidators[nlrType]` and reuse the value;
5. Make use of the existing `currentWithdrawProxy` variable (requires this recommendation to be adopted first);

Depending on the optimizer settings used, the optimizer itself can eliminate the duplicate `sload`s.

5.5.25 `RouterStorage.vaults` can be a boolean mapping

Severity: Gas Optimization

Context: AstariaRouter.sol#L662, AstariaRouter.sol#LL295, AstariaRouter.sol#LL590, IAstariaRouter.sol#LL85

Description: `RouterStorage.vaults` is of type `mapping(address => address)`. A key-value is stored in the mapping as:

```
s.vaults[vaultAddr] = msg.sender;
```

However, values in this mapping are only used to compare against `address(0)`:

```
if (_loadRouterSlot().vaults[msg.sender] == address(0)) {
...
return _loadRouterSlot().vaults[vault] != address(0);
```

It's better to have `vaults` as a boolean mapping as the assignment of `msg.sender` as value doesn't carry a special meaning.

Recommendation: To save gas, make `RouterStorage.vaults` of type `mapping(address => bool)`. Assign it as:

```
- s.vaults[vaultAddr] = msg.sender;
+ s.vaults[vaultAddr] = true;
```

Instead of comparing it against `address(0)`, check if the value is true:

```
- if (_loadRouterSlot().vaults[msg.sender] == address(0)) {
+ if (!_loadRouterSlot().vaults[msg.sender]) {
...
- return _loadRouterSlot().vaults[vault] != address(0);
+ return _loadRouterSlot().vaults[vault];
```

Astaria: Fixed in commit 2f5856.

Spearbit: Verified.

5.5.26 `isValidReference()` should just take an array element as input

Severity: Gas Optimization

Context: AstariaRouter.sol#L596-L602

Description: `isValidRefinance()` takes `stack` array as an argument but only uses `stack[0]` and `stack[position]`:

```
function isValidRefinance(
 ILienToken.Lien calldata newLien,
 uint8 position,
 ILienToken.Stack[] calldata stack
) public view returns (bool) {
```

The usage of `stack[0]` can be replaced with `stack[position]` as `stack[0].lien.collateralId == stack[position].lien.collateralId`:

```
if (newLien.collateralId != stack[0].lien.collateralId) {
  revert InvalidRefinanceCollateral(newLien.collateralId);
}
```

To save gas, it can directly take that one element as input.

Recommendation:

- Update the function to take `stack[position]` as input:

  ```
  function isValidRefinance(
   ILienToken.Lien calldata newLien,
   ILienToken.Stack calldata stack // notice the type change of `stack`
  ) public view returns (bool) {
  ```

- Replace all usages of `stack[position]` and `stack[0]` with `stack`.
- Replace all calls to `isValidReference()` to use the new function signature.

Astaria: Acknowledged. Will be automatically fixed through the issue: "*`isValidRefinance` and related storage parameters can be moved to `LienToken`*"

5.5.27 Functions can be made `external`

Severity: Gas Optimization

Context:

- WithdrawProxy.sol#LL211
- PublicVault.sol#L486
- VaultImplementation.sol#L42-L44
- VaultImplementation.sol#L66

Description: If `public` function is not called from within the contract, it should made `external` for clarity, and can potentially save gas.

Recommendation: Convert all highlighted functions to `external`.

Astaria: Fixed in the commit cfa6e0.

Spearbit: Verified.

5.5.28 Store bytes known at compile time as `constant` variables

Severity: Gas Optimization

Context: LienToken.sol#L134, LienToken.sol#L291, LienToken.sol#L369, VaultImplementation.sol#L140-L143

Context points to several instance which can be converted into `constant` variables. For instance, `bytes32("ACTIVE_AUCTION")` and `keccak256("0")`. This is less error-prone and saves gas as `keccak`'s value will be inlined by the compiler.

Recommendation: Convert all highlighted code to `constant` variables.

Astaria: Verified in commit 655728.

Spearbit: Verified.

5.5.29 `a.mulDivDown(b,1)` is equivalent to `a*b`

Severity: Gas Optimization

Context: LienToken.sol#L220, LienToken.sol#L733

Description: Highlighted code above follow the pattern of `a.mulDivDown(b, 1)` which is equivalent to `a*b`.

Recommendation: Replace all instances of `a.mulDivDown(b, 1)` with `a*b` to save gas.

Astaria: Fixed in commits f56f41 and 5af9c8.

Spearbit: Verified.

5.5.30 Use scratch space for keccak

Severity: Gas Optimization

Context: CollateralLookup.sol#L21

Description: `computeId()` function computes and returns `uint256(keccak256(abi.encodePacked(token, tokenId)))`. Since the data being hashed fits within 2 memory slots, scratch space can be used to avoid paying gas cost on memory expansion.

Recommendation: Consider rewriting `computeId()` function as:

```
function computeId(address token, uint256 tokenId)
  internal
  pure
  returns (uint256 hash)
{
  assembly {
    mstore(0, token) // sets the right most 20 bytes in the first memory slot.
    mstore(0x20, tokenId) // stores tokenId in the second memory slot.
    hash := keccak256(12, 52) // keccak from the 12th byte up to the entire second memory slot.
  }
}
```

Astaria: Fixed in commit 47e7a6.

Spearbit: Verified.

5.6 Informational

5.6.1 Define a named constant for the return value of `onFlashAction`

Severity: Informational

Context: ClaimFees.sol#L38

Description: `onFlashAction` returns:

```
keccak256("FlashAction.onFlashAction")
```

Recommendation: This value can be turned into a named constant:

```
bytes32 private constant FLASH_ACTION_MAGIC = keccak256("FlashAction.onFlashAction");
```

The constant value can be used as the return value instead.

5.6.2 Define a named constant for `permit` typehash in `ERC20-cloned`

Severity: Informational

Context: ERC20-Cloned.sol#L121-L123

Description: In `permit`, the following type hash has been used:

```
keccak256(
  "Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
)
```

Recommendation: It would be best to define a named constant for the hash above and replace its usage with the named constant:

```
bytes32 private constant PERMIT_TYPEHASH = keccak256(
  "Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)"
);
```

5.6.3 Unused `struct`, `enum` and `storage` fields can be removed

Severity: Informational

Context:

- IAstariaRouter.sol#L38
- IAstariaRouter.sol#L41
- ICollateralToken.sol#L67
- ILienToken.sol#L22
- ICollateralToken.sol#L49
- ICollateralToken.sol#L42

Description: The `struct`, `enum` and `storage` fields in this **context** have not been used in the project.

Recommendation: If there is no plan to use these fields, it would be best to remove them to simplify/clean the code base.

5.6.4 `WPStorage.expected`'s comment can be made more accurate

Severity: Informational

Context: WithdrawProxy.sol#L53

Description: In `WPStorage`'s definition we have:

```
uint88 expected; // Expected value of auctioned NFTs. yIntercept (virtual assets) of a PublicVault are
↪  not modified on liquidation, only once an auction is completed.
```

The comment for `expected` is not exactly accurate.

The accumulated value in `expected` is the sum of all auctioned NFTs's `amountOwed` when (the timestamp) the `liquidate` function gets called.

Whereas the NFTs get auctioned starting from their first stack's element's `liquidationInitialAsk` to `1_000 wei`

Recommendation: The comment for `expected` can be modified to emphasis that:

> The accumulated value in `expected` is the sum of all auctioned NFTs's `amountOwed` when (the timestamp) the `liquidate` function gets called.

5.6.5 Leave comment that in `WithdrawProxy.claim()` the calculation of `balance` cannot underflow

Severity: Informational

Context: WithdrawProxy.sol#L235-L236

Description: There is this following line in `claim()` where `balance` is initialised:

```
uint256 balance = ERC20(asset()).balanceOf(address(this)) -
  s.withdrawReserveReceived;
```

With the current `PublicVault` implementation of `IPublicVault`, this cannot underflow since the increase in `withdrawReserveReceived` (using `increaseWithdrawReserveReceived`) is synced with increasing the asset balance by the same amount.

Recommendation: The claim in the description needs to be doubled-checked. But the general recommendation is to leave a comment/explanation as in the description.

5.6.6 Shared logic in `withdraw` and `redeem` functions of `WithdrawProxy` can be turned into a shared modifier

Severity: Informational

Context:

- WithdrawProxy.sol#L146-L152
- WithdrawProxy.sol#L169-L175

Description: `withdraw` and `redeem` both start with the following lines:

```
WPStorage storage s = _loadSlot();
// If auction funds have been collected to the WithdrawProxy
// but the PublicVault hasn't claimed its share, too much money will be sent to LPs
if (s.finalAuctionEnd != 0) {
  // if finalAuctionEnd is 0, no auctions were added
  revert InvalidState(InvalidStates.NOT_CLAIMED);
}
```

Since they have this shared logic at the beginning of their body, we can consolidate the logic into a modifier.

Recommendation: Refactor the shared logic into a modifier that can be used by both functions.

```
modifier onlyWhenNoActiveAuction() {
  WPStorage storage s = _loadSlot();
  // If auction funds have been collected to the WithdrawProxy
  // but the PublicVault hasn't claimed its share, too much money will be sent to LPs
  if (s.finalAuctionEnd != 0) {
    // if finalAuctionEnd is 0, no auctions were added
    revert InvalidState(InvalidStates.NOT_CLAIMED);
  }
  _;
}
```

5.6.7 StrategyDetails version can only be used in custom implementation of IStrategyValidator, requires documentation

Severity: Informational

Context: IAstariaRouter.sol#L103

Description: StrategyDetails.version is never used in the current implementations of the validators.

- If the intention is to avoid replays across different versions of Astaria, we should add a check for it in commitment validation functions.
- A custom implementation of IStrategyValidator can make use of this value, but this needs documentation as to exactly what it refers to.

Recommendation: Include documentation on how StrategyDetails.version will be used, and add a check to the validators if needed.

5.6.8 Define helper functions to tag different pieces of cloned data for `ClearingHouse`

Severity: Informational

Context:

- ClearingHouse.sol#L22
- ClearingHouse.sol#L31

Description: `_getArgAddress(0)` and `_getArgUint256(21)` are used as the `ROUTER()` and `COLLATERAL_ID()` in the `fallback` implementation for `ClearingHouse` was `Clone` derived contract.

Recommendation: It would be best to define endpoints for these different parameters to tag the pieces of cloned data like other similiar `Clone` contracts in the project. So we can define:

```
function ROUTER() public pure returns (IAstariaRouter) {
  return IAstariaRouter(_getArgAddress(0));
}
```

```
function COLLATERAL_ID() public pure returns (uint256) {
  return _getArgAddress(21);
}
```

and (to be consistent with other cloned contracts)

```
function IMPL_TYPE() public pure returns (uint8) {
  return _getArgUint8(20);
}
```

and the `fallback` can be changed to:

```
fallback() external payable {
  IAstariaRouter ASTARIA_ROUTER = IAstariaRouter(ROUTER());
  require(msg.sender == address(ASTARIA_ROUTER.COLLATERAL_TOKEN().SEAPORT()));
  WETH(payable(address(ASTARIA_ROUTER.WETH()))).deposit{value: msg.value}();
  uint256 payment = ASTARIA_ROUTER.WETH().balanceOf(address(this));
  ASTARIA_ROUTER.WETH().safeApprove(
    address(ASTARIA_ROUTER.TRANSFER_PROXY()),
    payment
  );
  ASTARIA_ROUTER.LIEN_TOKEN().payDebtViaClearingHouse(
    COLLATERAL_ID(),
    payment
  );
}
```

5.6.9 A new modifier `onlyVault()` can be defined for `WithdrawProxy` to consolidate logic

Severity: Informational

Context:

- WithdrawProxy.sol#L212
- WithdrawProxy.sol#L271
- WithdrawProxy.sol#L281
- WithdrawProxy.sol#L291

Description: The following `require` statement has been used in multiple functions including `increaseWithdrawReserveReceived`, `drain`, `setWithdrawRatio` and `handleNewLiquidation`.

```
require(msg.sender == VAULT(), "only vault can call");
```

Recommendation: We can transform the `require` statement into a modifier and use the modifier instead for the functions in this context.

```
modifier onlyVault() {
   require(msg.sender == VAULT(), "only vault can call");
   _;
}
```

5.6.10 Inconsistant pragma versions and floating pragma versions can be avoided

Severity: Informational

Context: Description: Most contracts in the project use `pragma solidity ^0.8.17`, but there are other variants as well:

```
pragma solidity ^0.8.16;  // src/Interfaces/IAstariaVaultBase.sol
pragma solidity ^0.8.16;  // src/Interfaces/IERC4626Base.sol
pragma solidity ^0.8.16;  // src/Interfaces/ITokenBase.sol

pragma solidity ^0.8.15;  // src/Interfaces/ICollateralToken.sol

pragma solidity ^0.8.0;   // src/Interfaces/IERC20.sol
pragma solidity ^0.8.0;   // src/Interfaces/IERC165.sol
pragma solidity ^0.8.0;   // src/Interfaces/IERC1155.sol
pragma solidity ^0.8.0;   // src/Interfaces/IERC1155Receiver.sol
pragma solidity ^0.8.0;   // src/Interfaces/IERC721Receiver.sol
pragma solidity ^0.8.0;   // src/utils/Math.sol

pragma solidity >=0.8.0;  // src/Interfaces/IERC721.sol
pragma solidity >=0.8.0;  // src/utils/MerkleProofLib.sol
```

And they all have floating version pragmas.

- In `hardhat.config.ts`, `solidity: "0.8.13"` is used.
- In `.prettierrc` settings we have `"compiler": "0.8.17"`
- In `.solhint.json` we have `"compiler-version": ["error", "^0.8.0"]`
- `foundry.toml` does not have a `solc` setting

Recommendation: Unless a file is used as a library or used by other projects, it would be best to not have floating pragmas and assign the same pragma version to all the main files and use that fixed prgama version in other settings as well to make sure that the code gets compiled deterministally using the same compiler version. In this case the fixed pragma version would be `=0.8.17`.

5.6.11 `IBeacon` is missing a compiler version pragma

Severity: Informational

Context: IBeacon.sol#L11

Description: `IBeacon` is missing a compiler version pragma.

Recommendation: It is recommend to add the compiler version pragma to this file:

```
pragma solidity x.y.z;
```

5.6.12 `zone` and `zoneHash` are not required for fully open Seaport orders

Severity: Informational

Context: CollateralToken.sol#L524-L530

Description: As per Seaport's documentation,`zone` and `zoneHash` are not required for `PUBLIC` orders:

> The zone of the order is an optional secondary account attached to the order with two additional privileges:
>
> - The zone may cancel orders where it is named as the zone by calling cancel. (Note that offerers can also cancel their own orders, either individually or for all orders signed with their current counter at once by calling incrementCounter).
> - "Restricted" orders (as specified by the order type) must either be executed by the zone or the offerer, or must be approved as indicated by a call to an isValidOrder or isValidOrderIncludingExtraData view function on the zone.

This order isn't "Restricted", and there is no way to cancel a Seaport order once created from this contract.

Recommendation: You can consider removing `zone` and `zoneHash` if the plan is to keep Seaport orders fully open.

Note: If applying this recommendation, applying this issue's fix is mandatory to Issue 150.

5.6.13 Inconsistent treatment of delegate setting

Severity: Informational

Context: VaultImplementation.sol#L195

Description: Private vaults include delegate in the allow list when deployed through the Router. Public vaults do not. The `VaultImplementation`, when mutating a delegate, sets them on allow list.

Recommendation: Make consistent or note decision to have different deploy behaviors.

5.6.14 `AstariaRouter` does not adhere to EIP1967 spec

Severity: Informational

Context: AstariaRouter.sol#L48

Description: The Router serves as an implementation `Beacon` for proxy contracts, however, does not adhere to the EIP1967 spec.

Recommendation: Inherit compliant `IBeacon` (e.g. OpenZeppelin) and make conform to the spec.

5.6.15 Type mismatch `stack.point.end`

Severity: Informational

Context: LienToken.sol#L757, PublicVault.sol#L509

Description: `stack.point.end` is `uint40` but used elsewhere as `uint64`.

5.6.16 Receiver of bought out lien must be approved by msg.sender

Severity: Informational

Context: LienToken.sol#L107-111

Description: The `buyoutLien` function requires that either the receiver of the lien is `msg.sender` or is an address approved by `msg.sender`:

```
if (msg.sender != params.encumber.receiver) {
  require(
    _loadERC721Slot().isApprovedForAll[msg.sender][params.encumber.receiver]
  );
}
```

This check seems unnecessary and in some cases will block users from buying out liens as intended.

Recommendation: Remove this check.

Once it is removed, we can also remove the update at VaultImplementation.sol#L331-336 for private vaults to approve the vault owner before buyoutLien() is called:

```
if (
  recipient() != address(this) &&
  !lienToken.isApprovedForAll(address(this), recipient())
) {
  lienToken.setApprovalForAll(recipient(), true);
}
```

Astaria: Fixed in PR 204.

Spearbit: Verified, updates in this PR 276solve the issue.

5.6.17 A new modifer `onlyLienToken()` can be defined to refactor logic

Severity: Informational

Context:

- PublicVault.sol#L487
- PublicVault.sol#L525
- PublicVault.sol#L574
- PublicVault.sol#L588
- PublicVault.sol#L604

Description: The following `require` statement has been used in multiple locations in `PublicVault`:

```
require(msg.sender == address(LIEN_TOKEN()));
```

Locations used:

- `beforePayment`
- `afterPayment`
- `handleBuyoutLien`
- `updateAfterLiquidationPayment`
- `updateVaultAfterLiquidation`

Recommendation: It would be best to consolidate this condition into a `modifier`.

```
modifier onlyLienToken() {
  require(msg.sender == address(LIEN_TOKEN()));
  _;
}
```

5.6.18 A redundant `if` block can be removed from `PublicVault._afterCommitToLien`

Severity: Informational

Context:

- PublicVault.sol#L428-L430
- PublicVault.sol#L443
- PublicVault.sol#L443

Description: In `PublicVault._afterCommitToLien`, we have the following `if` block:

```
if (s.last == 0) {
  s.last = block.timestamp.safeCastTo40();
}
```

This `if` block is redundant, since regardless of the value of `s.last`, a few lines before `_accrue(s)` would update the `s.last` to the current timestamp.

Recommendation: The `if` block above can be removed. And perhaps to emphasis that the `s.last` value is update in `_accrue(s)`, we can leave a comment for it on a line above in `_afterCommitToLien` body.

5.6.19 Private vaults' `deposit` endpoints can be potentially simplifed

Severity: Informational

Context: Vault.sol#L74-L77

Description: A private vault's `deposit` function can be called directly or indirectly using the `ROUTER()` (either way by anyone) and we have the following `require` statement:

```
require(
  s.allowList[msg.sender] ||
    (msg.sender == address(ROUTER()) && s.allowList[receiver])
);
```

If the `ROUTER()` is the `AstariaRouter` implementation of `IAstariaRouter`, then it inherits from `ERC4626RouterBase` and `ERC4626Router` which allows anyone to call into `deposit` of this private vault using:

- `depositToVault`
- `depositMax`
- `ERC4626RouterBase.deposit`

Thus if anyone of the above functions is called through the `ROUTER()`, `msg.sender == address(ROUTER()` will be true. Also, note that when private vaults are created using the `newVault` the `msg.sender/owner` along the `delegate` are added to the `allowList` and allowlist is enabled.

And since there is no bookkeeping here for the `receiver`, except only the `require` statement, that means

- Only the `owner` or the `delegate` of this private vault can call directly into `deposit` or
- Anyone else can set the `address to` parameter of one of those `3` endpoints above to `owner` or `delegate` to deposit assets (`wETH` in the current implementation) into the private vault.

And all the assets can be `withdrawn` by the `owner` only.

Recommendation: We suggest documenting the above paths.

The restriction on the `receiver` would make sense to make sure end users are aware that they are sending assets through the `ROUTER()` to an allowed listed user of the private vault and to avoid potential `deposit` mistakes. It would make sense to simplify the `require` statement though:

```
require(s.allowList[receiver]));
```

The current form of the `require` statement is also acceptable, although it would be best to document why this choice was taken (gas saving for the `owner/delegate` to set the `receiver` to `address(0)`)

5.6.20 The `require` statement in `decreaseEpochLienCount` can be more strict

Severity: Informational

Context: PublicVault.sol#L498

Description: `decreaseEpochLienCount` has the following `require` statement that limits who can call into it:

```
require(
  msg.sender == address(ROUTER()) || msg.sender == address(LIEN_TOKEN())
);
```

So only, the `ROUTER()` and `LIEN_TOKEN()` are allowed to call into. But `AstariaRouter` never calls into this function.

Recommendation: If you are not planning to add new functionalities to `AstariaRouter` that would call into a public vault's `decreaseEpochLienCount` endpoint, it is recommended to make this `require` statement more strict:

```
require(msg.sender == address(LIEN_TOKEN()));
```

5.6.21 `amount` is not used in `_afterCommitToLien`

Severity: Informational

Context: PublicVault.sol#L414

Description: `amount` is not used in `_afterCommitToLien` to update/decrement `s.yIntercept`, because even though assets have been transferred out of the vault, they would still need to be paid back and so the net effect on `s.yIntercept` (that is used in the calculation of the total virtual assets) is `0`.

Recommendation: Comment out `amount` and also maybe provide an explanation in a NatSpec `@dev` of the reason that it has not been used.

```
function _afterCommitToLien(
  uint40 lienEnd,
  uint256 lienId,
  uint256 /* amount */,
  uint256 lienSlope
) internal virtual override {
```

Also since `_afterCommitToLien` is only overridden with an actual implementation in this scenario, if `amount` is not needed it can be removed from the function signature completely.

5.6.22 Use modifier

Severity: Informational

Context: AstariaRouter.sol#L270, AstariaRouter.sol#L264, VaultImplementation.sol#L67-L99, VaultImplementation.sol#L131, VaultImplementation.sol#L196

Description: Highlighted code have `require` checks on `msg.sender` which can be converted to modifiers. For instance:

```
require(address(msg.sender) == s.guardian);
```

Recommendation: Replace these checks with modifiers.

5.6.23 Prefer `SafeCastLib` for typecasting

Severity: Informational

Context: AstariaRouter.sol#L95-L106

Description: Highlighted code above does typecasting of several constant values. In case, some value doesn't fit in the type, this typecasting will silently ignore the higher order bits although that's currently not the case, but it may pose a risk if these values are changed in future.

Recommendation: Consider using `SafeCastLib` for all typecastings.

Astaria: Acknowledged.

Spearbit: Acknowledged.

5.6.24 Rename `Multicall` to `Multidelegatecall`

Severity: Informational

Context: Multicall.sol

Description: `Multicall.sol` lets performs multiple `delegatecalls`. Hence, the name `Multicall` is not suitable. The contract and the file should be named `Multidelegatecall`.

Recommendation: Update the name to `Multidelegatecall`, or replace `delegatecall` with `call`.

5.6.25 `safeTransferFrom()` without the `data` argument can be used

Severity: Informational

Context: CollateralToken.sol#LL347

Description: Highlighted code above sends empty data over an external call via `ERC721.safeTransferFrom(from, to, tokenId, data)`:

```
IERC721(underlyingAsset).safeTransferFrom(
  address(this),
  releaseTo,
  assetId,
  ""
);
```

`data` can be removed since `ERC721.safeTransferFrom(from, to, tokenId)` sets empty data too.

Recommendation: Remove the empty `data` argument from `safeTransferFrom()`.

Astaria: Fixed in commit f63183.

Spearbit: Verified.

5.6.26 Fix documentation that `updateVaultAfterLiquidation` can be called by LIEN_TOKEN, not ROUTER

Severity: Informational

Context: PublicVault.sol#L608

Description: The function has the correct validation that it can only be called by `LIEN_TOKEN()`, but the comment says it can only be called by `ROUTER()`.

```
require(msg.sender == address(LIEN_TOKEN())); // can only be called by router
```

Recommendation: Update the comment to say `lien token` or remove the comment entirely.

5.6.27 Declare event and constants at the beginning

Severity: Informational

Context: CollateralToken.sol#L598, VaultImplementation.sol#L150

Description: Events and constants are generally declared at the beginning of a smart contract. However, for the highlighted code above, that's not the case.

Recommendation: Move event and constant declarations at the beginning of the contract.

Astaria: Fixed in commit b55058.

Spearbit: Verified.

5.6.28 Rename `Vault` to `PrivateVault`

Severity: Informational

Context: Vault.sol#L34

Description: `Vault` contract is used to represent private vaults.

Recommendation: To distinguish private and public vaults better, it would be best to rename `Vault` contract/file to `PrivateVault`. Note that public vaults are already called `PublicVault`.

5.6.29 Remove comment

Severity: Informational

Context: WithdrawProxy.sol#L229

Description: Comment at line WithdrawProxy.sol#L229 can be removed:

```
if (
  block.timestamp < s.finalAuctionEnd
  // || s.finalAuctionEnd == uint256(0)
) {
```

The condition in comments is always false as the code already reverts in that case.

Recommendation: Remove comment.

Astaria: Fixed in commit 861bf.

Spearbit: Verified.

5.6.30 WithdrawProxy and PrivateVault symbols are missing hyphens

Severity: Informational

Context:

- WithdrawProxy.sol#L113
- Vault.sol#L55

Description: The symbol for the WithdrawProxy token is missing a hyphen after the W, which will make the name `AST-W0x...` instead of `AST-W-0x...`.

Similarly, the symbol for the Private Vault token (in Vault.sol) is missing a hyphen after the V.

Recommendation:

```
- string(abi.encodePacked("AST-W", owner(), "-", ERC20(asset()).symbol()));
+ string(abi.encodePacked("AST-W-", owner(), "-", ERC20(asset()).symbol()));
```

```
- string(abi.encodePacked("AST-V", owner(), "-", ERC20(asset()).symbol()));
+ string(abi.encodePacked("AST-V-", owner(), "-", ERC20(asset()).symbol()));
```

5.6.31 Lien cannot be bought out after `stack.point.end`

Severity: Informational

Context: LienToken.sol#L731

Description: The `_getRemainingInterest` function reverts with `Panic(0x11)` when `block.timestamp > stack.point.end`.

Recommendation: If intentional, include explicit check and informative error message, or include documentation to note decision.

5.6.32 Inconsistent strictness of inequalities in isValidRefinance

Severity: Informational

Context: AstariaRouter.sol#L593-612

Description: In `isValidRefinance`, we check that either: a) newRate < maxNewRate && newEnd >= oldEnd b) newEnd - oldEnd >= minDurationIncrease && newRate <= oldRate

We should be consistent in whether we're enforcing the changes are strict inequalities or non-strict inequalities.

Recommendation: Use strict inequalities (for slightly better gas performance) for the variables that need to improve. In other words, keep `newRate < maxNewRate` as is, but change it so that `newEnd - oldEnd > minDurationIncrease`.

Alternatively, you could use non-strict inequalities and change it so that `newRate <= maxNewRate`.

Astaria: Fixed in commit 0d2e24c.

Spearbit: Verified.

5.6.33 Lengthy comments

Severity: Informational

Context: AstariaRouter.sol#L58, LienToken.sol#L38

Context provides a few examples of lengthy comments. For readability, it's better to restrict comment length per line. Solidity guidelines suggests 120 characters.

Recommendation: Convert lengthy single line comments into multiple lines. To identify these comments, run `awk 'length>120' *sol` in `src`.

5.6.34 Clarify comments

Severity: Informational

Context: CollateralToken.sol#L524-L532

Description: Few comments are not clear on what they are referring to:

```
zone: address(this), // 0x20
...
conduitKey: s.CONDUIT_KEY, // 0x120
```

Recommendation: Elaborate these comments or remove if they are not useful.

5.6.35 Remove unused files

Severity: Informational

Context: CallUtils.sol

Description: `CallUtils.sol` is not used anywhere in the codebase.

Recommendation: Remove `CallUtils.sol`.

5.6.36 Document privileges and entities holding these privileges

Severity: Informational

Context: astaria-core

Description: There are certain privileged functionalities in the codebase (recognized through `requiresAuth` modifier). Currently, we have to refer to tests to identify the setup.

Recommendation: Document these privileges along with the entities holding these privileges. Also document the process of setting up `Auth/Authority` for the contracts.

5.6.37 Document and ensure that maximum number of liens should not be set greater than 256

Severity: Informational

Context: LienToken.sol#L625, LienToken.sol#L373-L375, LienToken.sol#L64

Description: Maximum number of liens in a stack is currently set to 5. While paying for a lien, the index in the stack is casted to uint8. This makes the implicit limit on maximum number of liens to be `256`.

Recommendation: Document this restriction and in case of any upgrade to this cap, ensure that all the related components are upgraded together to avoid the safe where casting down to `uint8` becomes unsafe.

5.6.38 `transferWithdrawReserve()` can return early when the current epoch is `0`

Severity: Informational

Context:

- PublicVault.sol#L338
- PublicVault.sol#L341
- PublicVault.sol#L380
- PublicVault.sol#L372

Description: If `s.currentEpoch == 0`, `s.currentEpoch - 1` will wrap around to `type(uint256).max` and we will most probably will `drain` assets into `address(0)` in the following block:

```
unchecked {
  s.withdrawReserve -= WithdrawProxy(withdrawProxy)
    .drain(
      s.withdrawReserve,
      s.epochData[s.currentEpoch - 1].withdrawProxy
    )
    .safeCastTo88();
}
```

But this cannot happen since in the outer `if` block the condition `s.withdrawReserve > 0` indirectly means that `s.currentEpoch > 0`.

The indirect implication above regarding the 2 conditions stems from the fact that `s.withdrawReserve` has only been set in `transferWithdrawReserve()` function or `processEpoch()`. In `transferWithdrawReserve()` function it assumes a positive value only when `s.currentEpoch > uint64(0)` and in `processEpoch()` at the end we are incrementing ‘s.currentEpoch‘.

Recommendation: It would be more readable to rewrite this function so that the `if (s.currentEpoch > uint64(0))` is turned into an early exit since it is a directly or indirectly implied condition for the whole function.

```
if (s.currentEpoch == uint64(0)) {
  return;
}
```

5.6.39 2 of the inner `if` blocks of `processEpoch()` check for a condition that has already been checked by an outer `if` block

Severity: Informational

Context: PublicVault.sol#L247

Description: The following 2 `if` block checks are redundant:

```
if (address(currentWithdrawProxy) != address(0)) {
  currentWithdrawProxy.setWithdrawRatio(s.liquidationWithdrawRatio);
}

uint256 expected = 0;
if (address(currentWithdrawProxy) != address(0)) {
  expected = currentWithdrawProxy.getExpected();
}
```

Since the condition `address(currentWithdrawProxy) != address(0)` has already been checked by an outer `if` block.

Recommendation: The 2 `if` blocks can be transformed into:

```
currentWithdrawProxy.setWithdrawRatio(s.liquidationWithdrawRatio);
uint256 expected = currentWithdrawProxy.getExpected();
```

5.6.40 General formatting suggestions

Severity: Informational

Context: PublicVault.sol#L283

Description:

- PublicVault.sol#L283 : there are extra sourounding paranthesis

Recommendation:

- PublicVault.sol#L283 : extra parenthesis can be removed:

```
if (address(currentWithdrawProxy) != address(0)) {
```

5.6.41 Identical collateral check is performed twice in _createLien

Severity: Informational

Context: LienToken.sol#L383-393

Description: In `_createLien`, a check is performed that the collateralId of the new lien matches the collateralId of the first lien on the stack.

```
if (params.stack.length > 0) {
  if (params.lien.collateralId != params.stack[0].lien.collateralId) {
    revert InvalidState(InvalidStates.COLLATERAL_MISMATCH);
  }
}
```

This identical check is performed twice (L383-387 and L389-393).

Recommendation: Delete L389-393.

Astaria: Fixed in commit 81a542.

Spearbit: Verified.

5.6.42 `checkAllowlistAndDepositCap` modifer can be defined to consolidate some of the `mint` and `deposit` logic for public vaults

Severity: Informational

Context:

- PublicVault.sol#L202-L210
- PublicVault.sol#L226-L234

Description: The following code snippet has been used for both `mint` and `deposit` endpoints of a public vault:

```
VIData storage s = _loadVISlot();
if (s.allowListEnabled) {
  require(s.allowList[receiver]);
}

uint256 assets = totalAssets();
if (s.depositCap != 0 && assets >= s.depositCap) {
  revert InvalidState(InvalidStates.DEPOSIT_CAP_EXCEEDED);
}
```

Recommendation: We can transform the snippet above into a `modifier checkAllowlistAndDeposit-Cap(address receiver)` to consolidate some of the logic. It would also simplify the codebase.

5.6.43 Document why `bytes4(0xffffffff)` is chosen when `CollateralToken` acting as a `Seaport` zone to signal invalid orders

Severity: Informational

Context:

- CollateralToken.sol#L148
- CollateralToken.sol#L163

Description: `CollateralToken's` `isValidOrder` and `isValidOrderIncludingExtraData` return `bytes4(0xffffffff)` to indicate a `Seaport` order using this zone is not a valid order.

Recommendation: Document why the specific value of `bytes4(0xffffffff)` was chosen to indicate the invalidity of an order using `CollateralToken` as a zone.

`Seaport` uses this value in one of its test files. Also `EIP-165` uses this magic value to indicate an unsupported interface.

5.6.44 `CollateralToken.onERC721Received`'s use of `depositFor` stack variable is redundant

Severity: Informational

Context: CollateralToken.sol#L652-L665

Description: If we follow the logic of assigning values to `depositFor` in `CollateralToken.onERC721Received`, we notice that it will end up being `from_`. So its usage is redundant.

Recommendation: Remove `depositFor` from `CollateralToken.onERC721Received` and replace its usage with `from_`:

```
_mint(from_, collateralId);

s.idToUnderlying[collateralId] = Asset({
  tokenContract: msg.sender,
  tokenId: tokenId_
});

emit Deposit721(msg.sender, tokenId_, collateralId, from_);
```

Also, if the above change is applied `operator` will not be used and so it can be commented out in the function signature:

```
function onERC721Received(
  address /* operator_ */,
  address from_,
  uint256 tokenId_,
  bytes calldata data_
) external
```

5.6.45 `onlyOwner` modifier can be defined to simplify the codebase

Severity: Informational

Context:

- CollateralToken.sol#L323-L326
- CollateralToken.sol#L254-L259

Description: `releaseToAddress` checks whether the `msg.sender` is an owner of a collateral. `CollateralToken` already has a modifier `onlyOwner(...)`, so the initial check in `releaseToAddress` can be delegated to the modifier.

Recommendation: `releaseToAddress` can be changed to the following to take advantage of the `onlyOwner` modifier:

```
function releaseToAddress(uint256 collateralId, address releaseTo)
  public
  releaseCheck(collateralId)
  onlyOwner(collateralId) // <-- added modifier
{
  CollateralStorage storage s = _loadCollateralSlot();
  _releaseToAddress(s, collateralId, releaseTo);
}
```

The only difference is that the `releaseToAddress` uses a custom error to revert, but the modifier uses a `require` statement.

5.6.46 Document `liquidator`'s role for the protocol

Severity: Informational

Context:

- AstariaRouter.sol#L519
- CollateralToken.sol#L107
- LienToken.sol#L472-L477

Description: When a lien's term end (`stack.point.end <= block.timestamp`), anyone can call the `liquidate` on `AstariaRouter`. There is no restriction on the `msg.sender`. The `msg.sender` will be set as the `liquidator` and if:

- The `Seaport` auction ends (3 days currently, set by the protocol), they can call `liquidatorNFTClaim` to claim the NFT.
- Or if the `Seaport` auction settles, the `liquidator` receives the liquidation fee.

Recommendation: Document `liquidator`'s role for the protocol covering the points mentioned in the Description.

5.6.47 Until `ASTARIA_ROUTER` gets filed for `CollateralToken`, `CollateralToken` can not receive `ERC721`s safely.

Severity: Informational

Context: CollateralToken.sol#L72-L78

Description: `ASTARIA_ROUTER` is not set in the `CollateralToken`'s constructor. So till an entity with an authority would `file` for it, `CollateralToken` is unable to safely receive an `ERC721` token (`whenNotPaused` and `onERC721Received` would revert).

Recommendation: If this is part of the design decision, it might be useful to have it documented.

5.6.48 `_getMaxPotentialDebtForCollateral` might have meant to be an `internal` function

Severity: Informational

Context: LienToken.sol#L667-L668

Description: `_getMaxPotentialDebtForCollateral` is defined as a `public` function. Its name starts with `_`-underscore which as a convention usually is used for `internal` or `private` functions.

Recommendation: If it is necessary to use this `pure` function off-chain, it is recommended to chain its name to not start with `_` or if its meant to be an `internal` function, make sure to change its visibility to `internal`.

`_getMaxPotentialDebtForCollateral` is made `internal` in the following PR 102.

5.6.49 `return` keyword can be removed from `stopLiens`

Severity: Informational

Context: LienToken.sol#L240-L247

Description: `_stopLiens` does not return any values but in `stopLiens` the `return` statement is used along with the non-existent return value of `_stopLiens`.

Recommendation: `stopLiens` can be changed to:

```
-    return
      _stopLiens(
        _loadLienStorageSlot(),
        collateralId,
        auctionWindow,
        stack,
        liquidator
      );
```

5.6.50 LienToken's constructor does not set ASTARIA_ROUTER which makes some of the endpoints unfunctional

Severity: Informational

Context: LienToken.sol#L56-L65

Description: LienToken's constructor does not set ASTARIA_ROUTER. That means till an authorized entity calls file to set this parameter, the following functions would be broken/revert:

- buyoutLien
- _buyoutLien
- _payDebt
- getBuyout
- _getBuyout
- _isPublicVault
- setPayee, partially broken
- _paymentAH
- payDebtViaClearingHouse

Recommendation: Document the current design's decision as to not set the ASTARIA_ROUTER in the constructor.

Astaria: Working as intended

Spearbit: Acknowledged.

5.6.51 Document the approval process for a user's CollateralToken before calling commitToLiens

Severity: Informational

Context:

- AstariaRouter.sol#L680-L683
- VaultImplementation.sol#L232

Description: In the _executeCommitment's return statement:

```
IVaultImplementation(c.lienRequest.strategy.vault).commitToLien(
  c,
  address(this)
);
```

address(this) is the AstariaRouter. The call here to commitToLien enters into _validateCommitment with AstariaRouter as the receiver and so for it to no revert, the holder would have needed to set the approval for the router previously/beforehand:

```
CT.isApprovedForAll(holder, receiver)  // needs to be true
```

Recommendation: Document and comment for the users that for them to commit to liens they would need to approve all `CollateralToken`'s tokens for `AstariaRouter` so that `AstariaRouter` can act as an operator/spender.

5.6.52 `isValidRefinance`'s return statement can be reformatted

Severity: Informational

Context: AstariaRouter.sol#L606-L611

Description: Currently, it is a bit hard to read the return statement of `isValidRefinance`.

Recommendation: We suggest to reformat the statement similiar to the following for better readability:

```
(
  newLien.details.rate < maxNewRate &&
  block.timestamp + newLien.details.duration >= stack[position].point.end
) || (
  newLien.details.rate <= stack[position].lien.details.rate &&
  block.timestamp + newLien.details.duration - stack[position].point.end >= s.minDurationIncrease
);
```

5.6.53 Withdraw Reserves should always be transferred before Commit to Lien

Severity: Informational

Context: PublicVault.sol#L387-398

Description: When a new lien is requested, the `_beforeCommitToLien()` function is called. If the epoch is over, this calls `processEpoch()`. Otherwise, it calls `transferWithdrawReserve()`.

```
function _beforeCommitToLien(
  IAstariaRouter.Commitment calldata params,
  address receiver
) internal virtual override(VaultImplementation) {
  VaultData storage s = _loadStorageSlot();

  if (timeToEpochEnd() == uint256(0)) {
    processEpoch();
  } else if (s.withdrawReserve > uint256(0)) {
    transferWithdrawReserve();
  }
}
```

However, the `processEpoch()` function will fail if the withdraw reserves haven't been transferred. In this case, it would require the user to manually call `transferWithdrawReserve()` to fix things, and then request their lien again.

Instead, the protocol should transfer the reserves whenever it is needed, and only then call `processEpoch()`.

Recommendation:

```
  function _beforeCommitToLien(
    IAstariaRouter.Commitment calldata params,
    address receiver
  ) internal virtual override(VaultImplementation) {
    VaultData storage s = _loadStorageSlot();

-   if (timeToEpochEnd() == uint256(0)) {
-     processEpoch();
-   } else if (s.withdrawReserve > uint256(0)) {
-     transferWithdrawReserve();
+   if (s.withdrawReserve > uint256(0)) {
+     transferWithdrawReserve();
+   }
+   if (timeToEpochEnd() == uint256(0)) {
+     processEpoch();
   }
  }
```

Astaria: Fixed in PR 189.

Spearbit: Verified.

5.6.54 Remove owner() variable from withdraw proxies

Severity: Informational

Context: PublicVault.sol#L182-192

Description: When a `withdrawProxy` is deployed, it is created with certain immutable arguments. Two of these values are `owner()` and `vault()`, and they will always be equal. They seem to be used interchangeably on the withdraw proxy itself, so should be consolidated into one variable.

Recommendation: Remove owner from the immutable arguments passed to withdrawProxy, and change all instances of `owner()` used to `vault()`.

Astaria: Fixed in PR 190.

Spearbit: Verified.

5.6.55 Unnecessary checks in _validateCommitment

Severity: Informational

Context: VaultImplementation.sol#L220-234

Description: In `_validateCommitment()`, we check to confirm that either the sender of the message is adequately qualified to be making the decision to take a lien against the collateral (ie they are the holder, the operator, etc).

However, the way this is checked is somewhat roundabout and can be substantially simplified. For example, we check `require(operator == receiver);` in a block that is only triggered if we've already validated that `receiver != operator`.

Recommendation: To fix these unnecessary checks, I'd recommend refactoring as follows:

```
    address holder = CT.ownerOf(collateralId);
    address operator = CT.getApproved(collateralId);

    if (
      msg.sender != holder &&
      receiver != holder &&
      receiver != operator &&
-     !ROUTER().isValidVault(receiver)
+     !ROUTER().isValidVault(receiver) &&
+     !CT.isApprovedForAll(holder, receiver)
    ) {
-      if (operator != address(0)) {
-        require(operator == receiver);
-     } else {
-        require(CT.isApprovedForAll(holder, receiver));
+    revert ReceiverIsNotAllowed();
      }
    }
```

However, there are larger issues with this validation, and fixing these issues will resolve this. See PR 75

5.6.56 Comment or remove unused function parameters

Severity: Informational

Context:

- LienToken.sol#LL573
- CollectionValidator.sol#L51
- CollateralToken.sol#L627
- VaultImplementation.sol#L107-L110
- PublicVault.sol#L389
- PublicVault.sol#L537
- LienToken.sol#L195
- LienToken.sol#L294
- LienToken.sol#L410
- LienToken.sol#L726
- VaultImplementation.sol#L341

Description: Highlighted functions above take arguments which are never used. If the function has to have a particular signature, comment that argument name, otherwise remove that argument completely.

Additional instances noted in Context above.

- LienToken.sol#L726 : `LienStorage storage s` input parameter is not used in `_getRemainingInterest`. It can be removed and this function can be `pure`.
- VaultImplementation.sol#L341 : `incoming` is not used `buyoutLien`, was this variable meant to be used?

Recommendation: Remove `collateralId` from `_paymentAH()` arguments.

If commenting out a parameter but keeping its type to conform to a particular signature you can use the following form:

```
function f(
   A a,
   B b,
   ...
   C /* c */,
   ...
   X x
) ...
```

5.6.57 Zero address check can never fail

Severity: Informational

Context: CollectionValidator.sol#L63, UNI_V3Validator.sol#L93, UniqueValidator.sol#L64

Description: The `details.borrower != address(0)` check will never be `false` in the current system as AstariaRouter.sol#L352-L354 will revert when `ownerOf` is `address(0)`.

Recommendation: If this check is intended to assist off chain calls to `validateAndParse`, consider reverting whenever `details.borrower != address(0)`.

5.6.58 UX differs between `Router.commitToLiens` and `VaultImplementation.commitToLien`

Severity: Informational

Context: VaultImplementation.sol#L277

Description: The `Router` function creates the Collateralized Token while the `VaultImplementation` requires the collateral owner to `ERC721.safeTransferFrom` to the CollateralToken contract prior to calling.

Recommendation: UX Implications only. Confirm this is desired behavior and document.

5.6.59 Document what vaults are listed by Astaria

Severity: Informational

Context: AstariaRouter.sol#L443

Description: Anyone can call `newPublicVault` with `epochLength` in the correct range to create a public vault.

Recommendation: It would be best to document the process of listing and selecting public vaults, as the protocol currently allows anyone to register a public vault.

Astaria: This is correct, anyone can deploy a public vault but the UI will only show vaults by whitelisted strategists.

Spearbit: Acknowledged.

5.6.60 Simplify nested `if/else` blocks in `for` loops

Severity: Informational

Context:

- AstariaRouter.sol#L196
- AstariaRouter.sol#L268
- CollateralToken.sol#L191
- LienToken.sol#L77

Description: There are quite a few instances that nested `if/else` blocks are used in `for` loops and that is the only block in the `for` loop.

```
for ( ... ) {
  if (<CONDITION>) { ... }
  if else (<CONDITION>) { ... }
  ...
  if else (<CONDITION>) { ... }
  else { revert CustomError(); }
}
```

Recommendation: For better readability, it might be best to transform these blocks into single `if` blocks and use the `continue` keyword instead.

```
for ( ... ) {
  if( <CONDITION> ) {
     ...
     continue;
  }

  if( <CONDITION> ) {
     ...
     continue;
  }

  ...

  if( <CONDITION> ) {
     ...
     continue;
  }

  revert CustomError();
}
```

5.6.61 Document the role `guardian` plays in the protocol

Severity: Informational

Context: `AstariaRouter.sol`

Description: The role of `guardian` is not documented.

Recommendation: Document that the `guardian` can `file` or update the `implementations`, `COLLATERAL_TOKEN`, `LienToken` and `TRANSFER_PROXY`. If there are other actions or roles that the `guardian` can take or is supposed to have, they would also need to be documented.

5.6.62 `strategistFee...` have not been used can be removed from the codebase.

Severity: Informational

Context:

- AstariaRouter.sol#L219-L220
- IAstariaRouter.sol#L79-L80

Description: `strategistFeeNumerator` and `strategistFeeDenominator` are not used except in `getStrategist-Fee` (which itself also has not been referred to by other contracts).

It looks like these have been replaced by the vault fee which gets set by public vault owners when they create the vault.

Recommendation: Remove these 2 storage parameters if not planning to incorporate them in the future.

5.6.63 `redeemFutureEpoch` can be called directly from a public vault to avoid using the endpoint from `AstariaRouter`

Severity: Informational

Context:

- AstariaRouter.sol#L110
- PublicVault.sol#L128

Description: One can call the `redeemFutureEpoch` endpoint of the `vault` directly to avoid the extra gas of juggling assets and multiple contract calls when using the endpoint from `AstariaRouter`.

Recommendation: Document the decision of having the endpoint for `AstariaRouter` and/or why the same endpoint on the public vault has no restriction and can be called by any actor.

5.6.64 Remove unused imports

Severity: Informational

Context:

- AstariaRouter.sol#L30
- AstariaRouter.sol#L37
- AstariaRouter.sol#L39
- AstariaRouter.sol#L40
- LienToken.sol#L24
- PublicVault.sol#L13
- PublicVault.sol#L22
- PublicVault.sol#L23
- PublicVault.sol#L35)
- Vault.sol#L13
- Vault.sol#L15
- Vault.sol#L16
- Vault.sol#L18
- Vault.sol#L21
- Vault.sol#L24-L27
- Vault.sol#L29
- WithdrawProxy.sol#L13
- WithdrawProxy.sol#L21
- ERC20-Cloned.sol#L4
- ERC20-Cloned.sol#L5

Description: If an imported file is not used, it can be removed.

- LienToken.sol#L24 : since `Base64` is only imported in this file, if not used it can be removed from the codebase.

Recommendation: Remove unused imports.

5.6.65 Reduce nesting by reverting early

Severity: Informational

Context: CollateralToken.sol#L645-L669

Description: Code following this pattern:

```
if (<CONDITION>) {
  <BODY>
} else {
  revert();
}
```

can be simplified to remove nesting using custom errors:

```
if (!<CONDITION>) {
  revert();
}

<BODY>
```

or if using `require` statements, it can be transformed into:

```
require(<CONDITION>)

<BODY>
```

Recommendation: Check for invalid conditions and revert early. The suggested patterns would make the codebase more readable.

5.6.66 `assembly` can read `constant` global variables

Severity: Informational

Context: WithdrawProxy.sol#L190-L192, Pausable.sol#LL39, ERC20-Cloned.sol#L26

Description: Yul cannot read global variables, but that is not true for a constant variable as its value is embedded in the bytecode. For instance, highlighted code above have the following pattern:

```
bytes32 slot = WITHDRAW_PROXY_SLOT;
assembly {
  s.slot := slot
}
```

Here, `WITHDRAW_PROXY_SLOT` is a constant which can be used directly in assembly code.

Recommendation: Directly use constant variables in in assembly code.

5.6.67 Revert with error messages

Severity: Informational

Context: astaria-core

Description: There are many instances of `require` and `revert` statements being used without an accompanying error message. Error messages are useful for unit tests to ensure that a call reverted due the intended reason, and helps in identifying the root cause.

Recommendation: Add an error message with each revert. Since we recommend converting `require` statements to `revert` in "Mixed use of `require` and `revert`" issue, you can use custom errors.

5.6.68 Mixed use of `require` and `revert`

Severity: Informational

Context: astaria-core, astaria-gpl, CollectionValidator.sol#L64-L67, UniqueValidator.sol#L65-L68

Description: Astaria codebase uses a mix of `require` and `revert` statements. We suggest only following one of these ways to do conditional revert for standardization.

Recommendation: Convert all `require` statements to `revert`. Be careful with inverting the conditions while doing so.

5.6.69 `tokenURI` should revert on non-existing tokens

Severity: Informational

Context: LienToken.sol#LL294

Description: As per ERC721 standard, `tokenURI()` needs to revert if `tokenId` doesn't exist. The current code returns empty string for all inputs.

Recommendation: Revert if `_exists(tokenId) == address(0)`.

Astaria: Fixed in commit d9ebf8.

Spearbit: Verified.

5.6.70 Inheriting the same contract twice

Severity: Informational

Context: Vault.sol#L34, PublicVault.sol#L48-L50

Description: `VaultImplementation` inherits from `AstariaVaultBase` (reference). Hence, there is no need to inherit `AstariaVaultBase` in `Vault` and `PublicVault` contract as they both inherit `VaultImplementation` already.

Recommendation: Remove the explicit inheritance of `AstariaVaultBase` in `Vault.sol` and `PublicVault.sol`.

Astaria: Fixed in commit 282350.

Spearbit: Verified.

5.6.71 No need to re-cast variables

Severity: Informational

Context: VaultImplementation.sol#L219, LienToken.sol#L627, VaultImplementation.sol#L315, VaultImplementation.sol#L329

Description: Code above highlights redundant type castings.

```
ERC721 CT = ERC721(address(COLLATERAL_TOKEN()));
...
address(msg.sender)
```

These type castings are casting variables to the same type.

Recommendation: Remove type casting.

```
- ERC721 CT = ERC721(address(COLLATERAL_TOKEN()));
+ ERC721 CT = COLLATERAL_TOKEN();
...
-address(msg.sender)
+msg.sender
```

5.6.72 Comments do not match implementation

Severity: Informational

Context: AstariaVaultBase.sol#L9-L40, ILienToken.sol#L63, LienToken.sol#L758

Description:

- Scenario 1 & 2: Comments note where each parameter ends in a packed byte array, or parameter width in bytes. The comments are outdated.
- Scenario 3:

The unless is not implemented.

Recommendation:

- Scenario 1:

AstariaVaultBase.sol#L9-L40 update to correct locations:

```
abstract contract AstariaVaultBase is Clone, IAstariaVaultBase {
  function name() public view virtual returns (string memory);

  function symbol() public view virtual returns (string memory);

  function ROUTER() public pure returns (IAstariaRouter) {
    return IAstariaRouter(_getArgAddress(0)); //ends at 20
  }

  function IMPL_TYPE() public pure returns (uint8) {
    return _getArgUint8(20); //ends at 21
  }

  function owner() public pure returns (address) {
-    return _getArgAddress(21); //ends at 44
+    return _getArgAddress(21); //ends at 41
  }

  function asset() public pure virtual returns (address) {
-    return _getArgAddress(41); //ends at 64
+    return _getArgAddress(41); //ends at 61
```

```
  }

  function START() public pure returns (uint256) {
-    return _getArgUint256(61);
+    return _getArgUint256(61); //ends at 93
  }

  function EPOCH_LENGTH() public pure returns (uint256) {
-    return _getArgUint256(93); //ends at 116
+    return _getArgUint256(93); //ends at 125
  }

  function VAULT_FEE() public pure returns (uint256) {
-    return _getArgUint256(125);
+    return _getArgUint256(125); //ends at 157
  }

  function COLLATERAL_TOKEN() public view returns (ICollateralToken) {
    return ROUTER().COLLATERAL_TOKEN();
  }
}
```

- Scenario 2:

ILienToken.sol#L63 `Details` has 5 `unint256` values: `32 * 5`

- Scenario 3:

Remove comment or update implementation.

5.6.73 Function can be made internal

Severity: Informational

Context: LienToken.sol#L649

Description: `calculateSlope` is only used internally and can be edited from `public` to `internal`.

5.6.74 Incomplete Natspec

Severity: Informational

Context:

- LienToken.sol#L616
- LienToken.sol#L738-L750
- CollateralToken.sol#L616-L628
- VaultImplementation.sol#L153-L165
- VaultImplementation.sol#L298-L310
- AstariaRouter.sol#L75-L77
- AstariaRouter.sol#L44-L47

Description:

- LienToken.sol#L616 `s`, `@return` missing
- LienToken.sol#L738-L750 `s`, `position`, `@return` missing
- CollateralToken.sol#L616-L628 `tokenId_` missing

- VaultImplementation.sol#L153-L165 The second `*` on `/**` is missing causing the compiler to ignore the Natspec. The Natspec appears to document an old function interface. Params do not match with the function inputs.
- VaultImplementation.sol#L298-L310 missing `stack` and return vaule
- AstariaRouter.sol#L75-L77 `@param` NatSpec is missing for `_WITHDRAW_IMPL`, `_BEACON_PROXY_IMPL` and `_CLEARING_HOUSE_IMPL`
- AstariaRouter.sol#L44-L47 : Leave a comment that `AstariaRouter` also acts as an `IBeacon` for different cloned contracts.

Recommendation: Add to Natspec comments.

5.6.75 Cannot have multiple liens with same parameters

Severity: Informational

Context: LienToken.sol#L396

Description: Lien Ids are computed by hashing the `Lien` struct itself. This means that no two liens can have the same parameters (e.g. same amount, rate, duration, etc.).

Recommendation: No changes. Document constraint.

Astaria: Working as intended.

Spearbit: Acknowledged.

5.6.76 Redundant `unchecked` can be removed

Severity: Informational

Context: LienToken.sol#L264, LienToken.sol#L347, LienToken.sol#L395, WithdrawProxy.sol#LL282, PublicVault.sol#L286

Description: There are no arithmetic operations in these `unchecked` blocks. For clarity, it can be removed.

Recommendation: Removed `unchecked`.

5.6.77 Argument name reuse with different meaning across contracts

Severity: Informational

Context: AstariaRouter.sol#L476

Description: In `VaultImplementation` `receiver` is the borrower, while here and in `ILienToken.LienActionEncumber` `receiver` is the lender (the `receiver` of the LienToken)

Recommendation: Consider verbose naming, such as `lienTokenReceiver`, to reduce opportunity for confusion.

5.6.78 Licensing conflict on inherited dependencies

Severity: Informational

Context: WithdrawProxy.sol#L18, and others

Description: The version of Solmate contracts depended in tne `gpl` repository on are AGPL Licensed, making the `gpl` repository adopt the same license. This license is incompatible with the currently UNLICENSED Astaria related contracts.

Recommendation:

1. Consider the later versions of Solmate which have updated licensing.
2. Consider applying AGPL license to Astaria.