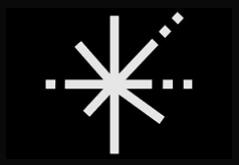
certora

Security Assessment & Formal Verification Report



Astaria starport

January 2024

Prepared for
Astaria

certora astaria

Table of content

Project Summary

Project Scope

Repo Name	Repository	Commits	Compiler version	Platform
starport	https://github.com/AstariaXYZ/starport/	67e3177 67b3182 Be0b40b 5da8b31	Solidity 0.8.17	EVM

Project Overview

This document describes the specification and verification of the **Starport Lending Kernel protocol and AstariaV1 modules** using the Certora Prover and manual code review findings. The work was undertaken from **16 November 2023** to **25 January 2024**.

The following contract list is included in our scope:

```
starport/src/Starport.sol
starport/src/Custodian.sol
starport/src/BNPLHelper.sol

starport/src/originators/StrategistOriginator.sol
starport/src/originators/Originator.sol

starport/src/enforcers/BorrowerEnforcerBNPL.sol
starport/src/enforcers/BorrowerEnforcer.sol
starport/src/enforcers/LenderEnforcer.sol
starport/src/enforcers/CaveatEnforcer.sol

starport/src/lib/StarportLib.sol
starport/src/lib/RefStarportLib.sol
starport/src/lib/PausableNonReentrant.sol
starport/src/lib/Validation.sol

starport/src/pricing/SimpleInterestPricing.sol
```

`starport/src/pricing/BasePricing.sol`
`starport/src/pricing/Pricing.sol`

`starport/src/settlement/FixedTermDutchAuctionSettlement.sol`
`starport/src/settlement/DutchAuctionSettlement.sol`
`starport/src/settlement/Settlement.sol`

`starport/src/status/FixedTermStatus.sol`
`starport/src/status/Status.sol`

The Certora Prover demonstrated the implementation of the Solidity contracts above is correct with respect to the formal rules written by the Certora team. In addition, the team performed a manual audit of all the Solidity contracts. During the verification process and the manual audit, the Certora Prover discovered bugs in the Solidity contracts code, as listed below.

Findings Summary

The table below summarizes the findings of the review, including type and severity details.

Severity	Discovered	Acknowledged	Code Fixed
Critical	0	0	0
High	8	1	7
Medium	6	1	5
Low	15	8	7
Informational	18	5	13
Total	**47**	**15**	**32**

Detailed Findings

High Severity Concerns

H-1. `Starport._feeRake()`: wrong index used for `feeItem`

Severity: High
Probability: High
Description:
If some tokens aren't ERC20 in `_feeRake`, there will be some empty `feeItems`, and the truncation would cut some real fees as `totalFeeItems < i`: Starport.sol#L633
This line (https://github.com/AstariaXYZ/starport/blob/b835af20d7dd20d1634c58270b11f5dfa300431b/src/Starport.sol#L607) should be `SpentItem memory feeItem = feeItems[totalFeeItems];`.
With a cleverly crafted `loan`, it's possible to avoid paying fees to the protocol.
Indeed, consider an array of debts with 3 ERC721 tokens and 1 ERC20 token. We would have `feeItems == [0, 0, 0, feeAmount]` and `totalFeeItems == 1`. Then, at line 694, the final result would be `feeItems == [0]`.

Recommendation:

```
Unset
File: Starport.sol
645:     function _feeRake(SpentItem[] memory debt)
...
649:     {
650:         feeItems = new SpentItem[](debt.length);
...
652:         uint256 totalFeeItems;
653:         for (uint256 i = 0; i < debt.length;) {
654:             uint256 amount;
655:             SpentItem memory debtItem = debt[i];
656:             if (debtItem.itemType == ItemType.ERC20) {
...
```

```
- 658:                 SpentItem memory feeItem = feeItems[i];
+ 658:                 SpentItem memory feeItem =
feeItems[totalFeeItems];
..
674:                 if (amount > 0) {
...
679:                     ++totalFeeItems;  //@audit only gets
incremented when there's a fee
680:                 }
681:             }
...
688:             unchecked {
689:                 ++i;  //@audit gets incremented for each `debt` in
the array
690:             }
691:         }
692:
693:         assembly ("memory-safe") {
694:             mstore(feeItems, totalFeeItems)
695:         }
```

Astaria's response: Fixed in commit [3189cea](). We accept the finding and have implemented the recommended changes. Additionally added a new test case `testDefaultFeeRake2`.

H-2. `Starport.incrementCounter()`: DOS due to shifting in the wrong direction

Severity: High
Probability: High
Description:
`Starport.incrementCounter()` intends to use a quasi-random number, just like Seaport does:

- CounterManager.sol#L31-L55

```
Unset
    function _incrementCounter() internal returns (uint256 newCounter)
{
        // Ensure that the reentrancy guard is not currently set.
        _assertNonReentrant();

        // Utilize assembly to access counters storage mapping
directly. Skip
        // overflow check as counter cannot be incremented that far.
        assembly {
            // Use second half of previous block hash as a quasi-random
number.
            let quasiRandomNumber := shr(Counter_blockhash_shift,
blockhash(sub(number(), 1)))
...
            // Derive new counter value using random number and
original value.
            newCounter := add(quasiRandomNumber, sload(storagePointer))
...
```

Here, `Counter_blockhash_shift == 0x80`, where `0x80 == 128`, and the code is shifting right the `block.number - 1`'s `blockhash` by `128` thereby making it a quasi random value evaluating to at most `type(uint128).max`. This means that there would need around `1e38` additions for an overflow to ever occur if this is being cast to `uint256`.

However, the non-assembly implementation from Starport is shifting left instead of right:

- Starport.sol#L302-L306

```
Unset
File: Starport.sol
    function incrementCaveatNonce() external {
        uint256 newNonce = caveatNonces[msg.sender] +
uint256(blockhash(block.number - 1) << 0x80);
        caveatNonces[msg.sender] = newNonce;
        emit CaveatNonceIncremented(msg.sender, newNonce);
    }
```

Shifting left means that the final value here would still be in the realm of `uint256` (with half the bits being zeros). This means that `newNonce` can be very close to `type(uint256).max`, easily putting `incrementCaveatNonce()` in a DOS situation.

As a reminder:

- Shifting right by n is like dividing by 2 ** n
- Shifting left by n is like multiplying by 2 ** n

Coded Proof of Concept 1

The POC below simulates 2 calls to `incrementCaveatNonce()`:

- *1 call at `block.number == 18784547`*
- *1 call at `block.number == 18784577`*

The following test can be run with `forge test --mt test_incrementCaveatNonce --fork-url {YOUR_ALCHEMY_ETHEREUM_MAINNET_RPC} --fork-block-number 18784577 -vvvv`:

```
Unset
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.17;

import "forge-std/Test.sol";
import "forge-std/console.sol";
```

```
contract BlockTest is Test {
    function setUp() public {}

    function test_incrementCaveatNonce() external {
        uint firstBlockNumber = 18784547;
        uint firstNonce = uint256(blockhash(firstBlockNumber - 1) <<
0x80);
        console.logBytes32(blockhash(firstBlockNumber - 1)); //
0x34ee8faee749ca86ce3b2c5119a22a2aa3621ade1f0c12d0720451c20178fc5f
        console.logBytes32(blockhash(firstBlockNumber - 1) << 0x80); //
0xa3621ade1f0c12d0720451c20178fc5f00000000000000000000000000000000
        console.log(
            "LOG: ~ file: BlockTests.sol:38 ~ test_incrementCaveatNonce
~ firstNonce:",
            firstNonce
        ); //
73900330765203527419966388593481834913399252111296159578743434103646956027904 (7.3e76)

        uint currentBlock = block.number; // 18784577
        bytes32 currentRightPart = blockhash(currentBlock - 1);
        console.logBytes32(currentRightPart); //
0x640a156fb6da4f9ad1b5e18077bcb035dfd908c6d2ce9281f9f76917806fb13f
        console.log(
            "LOG: ~ file: BlockTests.sol:42 ~ test_incrementCaveatNonce
~ currentRightPart:",
            uint256(currentRightPart)
        ); //
45249101277458430338161465564748733311993977424873570720032161665183850148159

        // Below will revert with "Arithmetic over/underflow"
        vm.expectRevert();
```

```
        uint newNonce = firstNonce + uint256(currentRightPart) << 0x80;
    }
}
```

As a reminder, the `--fork-block-number 18784577` part of the command is important as `blockhash` returns `0` for all block numbers below the latest 256th one.

Coded Proof of Concept 2

Apply the following diff to `TestStarport.sol` and run with `forge test --mt testIncrementCaveatNonce` to see this test reverting with `FAIL. Reason: Arithmetic over/underflow`:

Unset

```
File: TestStarport.sol
320:     function testIncrementCaveatNonce() public {
321:         vm.roll(5);
322:         uint256 newNonce = SP.caveatNonces(address(this)) +
uint256(blockhash(block.number - 1) << 0x80);
323:         vm.expectEmit();
324:         emit CaveatNonceIncremented(address(this), newNonce);
325:         SP.incrementCaveatNonce();
+ 326:         vm.roll(50);
+ 327:         SP.incrementCaveatNonce(); // [FAIL. Reason: Arithmetic
over/underflow]
328:     }
```

Recommendation:

Seaport's ReferenceCounterManager contract can be looked at as an example as it's more readable than the assembly version:

- ReferenceCounterManager.sol#L34-L49

```
Unset
    function _incrementCounter() internal returns (uint256 newCounter)
{
        // Use second half of the previous block hash as a quasi-random
number.
        uint256 quasiRandomNumber = uint256(blockhash(block.number -
1)) >> 128;

        // Retrieve the original counter value.
        uint256 originalCounter = _counters[msg.sender];

        // Increment current counter for the supplied offerer.
        newCounter = quasiRandomNumber + originalCounter;

        // Update the counter with the new value.
        _counters[msg.sender] = newCounter;

        // Emit an event containing the new counter.
        emit CounterIncremented(newCounter, msg.sender);
    }
```

The mitigation in the current codebase would therefore be:

```
Unset
File: Starport.sol
    function incrementCaveatNonce() external {
-       uint256 newNonce = caveatNonces[msg.sender] +
uint256(blockhash(block.number - 1) << 0x80);
+       uint256 newNonce = caveatNonces[msg.sender] +
(uint256(blockhash(block.number - 1) >> 0x80));
        caveatNonces[msg.sender] = newNonce;
        emit CaveatNonceIncremented(msg.sender, newNonce);
    }
```

Notice that putting the `quasiRandomNumber` (shift part) of the operation between parenthesis is important here as, unlike the expected behavior from divisions and multiplications, the shift operation here would apply on the whole addition instead of only the blockhash. See https://docs.soliditylang.org/en/latest/cheatsheet.html#order-of-precedence-of-operators.

Astaria's response: Fixed in commit 7f1d2ca. We accept the finding and have modified the recommended changes

```
- uint256 newNonce = caveatNonces[msg.sender] + uint256(blockhash(block.number - 1) << 0x80);
+ uint256 newNonce = caveatNonces[msg.sender] + 1 + (uint256(blockhash(block.number - 1) >> 0x80));
```

Incrementer added to ensure nonce changes in instances where `` `uint256(blockhash(block.number - 1) >> 0x80) == uint256(0)` ``

H-3. Funds will be locked inside `StrategistOriginator`

Severity: High
Probability: High
Description:
In `StrategistOriginator.originate()`, for the loan, `originator` is hardcoded to `address(0)`. It means that `StrategistOriginator` will be set as the loan's originator in `Starport._issueLoan()`.

If there are `carryConsiderations`, which target the `loan.originator`: the assets will be stuck inside `StrategistOriginator`, given that:

- ERC20 tokens aren't withdrawable
- ERC721 tokens are transferred with `transferFrom()` (not `safeTransferFrom()`), so they're received but not withdrawable
- ERC1155 tokens transfers will revert due to the use of `safeTransferFrom()` and the lack of `onERC1155Received()` implementation on the `StrategistOriginator`

Recommendation:
Consider adding access-controlled receivers and withdraw functions to the `StrategistOriginator` (the assets mustn't be stealable)
Additionally, consider adding `onERC1155Received()` on the abstract `Originator` itself.

Astaria's response: Fixed in commit 67b3182. We accept the recommendations and added withdraw methods and receivers to the `StrategistOriginator`.

H-4. Wrong calculation for excess of funds

Severity: High
Probability: High
Description and Recommendation:

- DutchAuctionSettlement.sol#L142-L144

Unset

```
        if (carry > 0 && loan.debt[0].amount + interest - carry <
settlementPrice) {
            consideration = new ReceivedItem[](2);
-            uint256 excess = settlementPrice - loan.debt[0].amount +
interest - carry;
+            uint256 excess = settlementPrice - (loan.debt[0].amount +
interest - carry);
```

Astaria's response: Fixed in commit 3189cea. We accept the finding and have applied the recommended changes to `DutchAuctionSettlement.sol`. As an informational note, modules implemented in the starport repository are for demonstration purposes only.

H-5. `USDT` won't be accepted as a collateral in `Custodian` anymore after the first user

Severity: Medium
Probability: High
Description:
When Seaport calls `generateOrder()`, on `Actions.Repayment` there's a systematic call to `_setOfferApprovalsWithSeaport(loan)`, which will give the maximum approval for the given tokens to `Seaport`:

- Custodian.sol#L353-L362

Unset
```
    function _enableAssetWithSeaport(SpentItem memory offer) internal
{
...
        } else if (offer.itemType == ItemType.ERC20) {
            ERC20(offer.token).approve(seaport, type(uint256).max);
        }
    }
```

However, it happens that USDT is a token that reverts when called with a different `approve` value than `0` when the current allowance isn't `0`. This is why OpenZeppelin implemented the `forceApprove()` function:

- SafeERC20.sol#L76-L83

Unset
```
    /**
     * @dev Set the calling contract's allowance toward `spender` to
`value`. If `token` returns no value,
     * non-reverting calls are assumed to be successful. Meant to be
used with tokens that require the approval
     * to be set to zero before setting it to a non-zero value, such as
USDT.
     */
```

```
    function forceApprove(IERC20 token, address spender, uint256
value) internal {
        bytes memory approvalCall = abi.encodeCall(token.approve,
(spender, value));

        if (!_callOptionalReturnBool(token, approvalCall)) {
            _callOptionalReturn(token, abi.encodeCall(token.approve,
(spender, 0)));
            _callOptionalReturn(token, approvalCall);
        }
    }
```

Given that, here, there isn't a call to `approve(seaport, 0)` before `approve(seaport, type(uint256).max)`: any calls to `generateOrder()` using `USDT` as collateral, will revert after the first call ever.

Recommendation

Consider using Solady's equivalent: `safeApproveWithRetry`:

- SafeTransferLib.sol#L321-L325

Unset

```
    /// @dev Sets `amount` of ERC20 `token` for `to` to manage on
behalf of the current contract.
    /// If the initial attempt to approve fails, attempts to reset the
approved amount to zero,
    /// then retries the approval again (some tokens, e.g. USDT,
requires this).
    /// Reverts upon failure.
    function safeApproveWithRetry(address token, address to, uint256
amount) internal {
```

Astaria's response: Fixed in commit 41af20b. We agree with the findings and accept the recommendation. We have updated implementation to use Solady `SafeTransferLib.safeApproveWithRetry`

H-6. In `DutchAuctionSettlement`, the `carryRate`'s decimals is assumed to always be 18

Severity: High
Probability: High
Description:
`mulWad` is used at DutchAuctionSettlement.sol#L137, however the `interest` is calculated by using `pricingDetails.decimals` right above at DutchAuctionSettlement.sol#L134.

Given the mismatch in unit and how the carry consideration is calculated elsewhere (BasePricing.sol#L92, StarportLib.sol#L95), then most likely the correct formula should be:

Unset
```
-        uint256 carry = interest.mulWad(pricingDetails.carryRate);
+        uint256 carry = interest * pricingDetails.carryRate / 10 **
pricingDetails.decimals;
```

Astaria's response: Fixed in commit 7f1d2ca. Accept the finding and have applied the recommended fix. As an informational note, modules implemented in the starport repository are for demonstration and testing purposes only.

H-7. `interest` will always be 0 for low-decimal tokens

Severity: High
Probability: High
Description:
Given StarportLib.sol#L94 :

```
Unset
    function calculateSimpleInterest(uint256 delta_t, uint256 amount,
uint256 rate, uint256 decimals)
        public
        pure
        returns (uint256)
    {
        rate /= 365 days;
        return ((delta_t * rate) * amount) / 10 ** decimals;
    }
```

The rate can be rounded down to 0 given that `365 days == 31_536_000`.
As an example, with USDC (6 decimals) and a 100% rate (`rate == 1e6`), this `rate` will always be 0, giving no interest.

Recommendation:

```
Unset
    function calculateSimpleInterest(uint256 delta_t, uint256 amount,
uint256 rate, uint256 decimals)
        public
        pure
        returns (uint256)
    {
-       rate /= 365 days;
-       return ((delta_t * rate) * amount) / 10 ** decimals;
+       return ((delta_t * rate) * amount) / 10 ** decimals / 365
days;
```

```
    }
```

Astaria's response: Fixed in commit 7f1d2ca. We accept the findings and have implemented the a modification of the recommended fix. Additionally we will note that this is a module implementation used to demonstrate Starport and is not intended to be deployed.

```
- rate /= 365 days;
- return ((delta_t * rate) * amount) / 10 ** decimals;
+ return ((delta_t * rate) * amount) / 10 ** decimals / 365 days;
```

H-8. Global concerns about modules pricing settlement and status

Severity: High
Probability: Medium
Description:

Inside `struct Loan`, there's the `struct Terms` which contains 3 addresses:

```
Unset
File: Starport.sol
117:
118:     struct Terms {
119:         address status; // the address of the status module
...
121:         address pricing; // the address of the pricing module
...
123:         address settlement; // the address of the handler module
..
125:     }
```

Arbitrary addresses are one of the most dangerous types to be arbitrary. Here, these can be abused to either lock funds or steal funds.

Given that users will be interacting with the protocol through the frontend, the concept of "Trusted Modules" is held outside of the protocol. This can put users at risk in case of a frontend attack (e.g. XSS adding an address to the list of trusted modules).

Recommendation:
While the users are told not to interact with untrusted modules, it is important to have several sources of truth regarding the list of trusted addresses for the `status`, `pricing` and `settlement` modules, so as to not select a malicious one in case of attack on the frontend.
Alternatively, consider adding an allowlist (or officialList) on `Starport`

Astaria's response: Acknowledged. We will establish documentation outlining the trusted modules and their deployed addresses.

Medium Severity Concerns

M-1. Lack of EIP-712 compliance: using `keccak256()` directly on an array or struct variable

Severity: Low
Probability: High
Description:
Directly using the actual variable instead of encoding the array values goes against the EIP-712 specification https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md#definition-of-encodedata.
Note: OpenSea's Seaport's example with offerHashes and considerationHashes can be used as a reference to understand how array of structs should be encoded.

Affected code:

- src/Starport.sol

```
Unset
File: Starport.sol
396:         CaveatEnforcer.Caveat[] calldata caveats
...
411:                         keccak256(abi.encode(caveats))
```

The adopted methodology for hashing an array of structs deviates from the prescribed guidelines set forth in EIP-712.

Astaria's response: Fixed in commit 41af20b. We accept the recommendation. The implementation has been adjusted to conform with the EIP-712 signing spec.

M-2. Missing right parenthesis on `INTENT_ORIGINATION_TYPEHASH`

Severity: Low
Probability: High
Description:
By adding the right parenthesis, the typehash would go from `0x3b79d2ff6939199614b0e56e719f097dc6eafc66adf2e5992da19e9e20413b1b` to `0x93d5b3eb7e7c73e817f1f0b6a9b409fa1b84da976c364c97b62acaf9c35047bb`:

- Starport.sol#L103

```
Unset
File: Starport.sol
102:     bytes32 public constant INTENT_ORIGINATION_TYPEHASH =
keccak256(
- 103:         "Origination(address account,uint256 accountNonce,bool
singleUse,bytes32 salt,uint256 deadline,bytes32 caveatHash"
+ 103:         "Origination(address account,uint256 accountNonce,bool
singleUse,bytes32 salt,uint256 deadline,bytes32 caveatHash)"
104:     );
```

Astaria's response: Fixed in commit 41af20b. We accept the finding, and have made the recommended change to conform with EIP-712 signing spec.

M-3. Error-prone string casting for `tokenURI`

Severity: Low
Probability: High
Description:
While not implemented yet,`tokenURI` is using an error-prone `string` casting:

- Custodian.sol#L125

```
Unset
File: Custodian.sol
121:     function tokenURI(uint256 loanId) public view override
returns (string memory) {
122:         if (!_exists(loanId)) {
123:             revert InvalidLoan();
124:         }
125:         return string("");
126:     }
```

In a previous commit, the `tokenURI` was computed this way:

```
Unset
    function tokenURI(uint256 tokenId) public view override returns
(string memory) {
        return
string(abi.encodePacked("https://astaria.xyz/loans?id=", tokenId));
    }
```

This wouldn't work as `abi.encodePacked` doesn't differentiate between bytes data and string data. And, here, `tokenId` isn't a string.

By trying this exact function into Remix, as an example with `42`as an input, the following string would be returned: `https://astaria.xyz/loans?id=*`.

Indeed, to use a `uint256` type as a string, the following library from Solady should be used: https://github.com/Vectorized/solady/blob/main/src/utils/LibString.sol.

To fix the `tokenURI`, the following syntax could be used:

```
Unset
+ import
"https://github.com/Vectorized/solady/blob/main/src/utils/LibString.s
ol";

    function tokenURI(uint256 tokenId) public view override returns
(string memory) {
-       return
string(abi.encodePacked("https://astaria.xyz/loans?id=", tokenId));
+       return
string(abi.encodePacked("https://astaria.xyz/loans?id=",
LibString.toString(tokenId)));
    }
```

However, the best mitigation would be to use `string.concat` instead of combining the casting to `string` and use of `abi.encodePacked` for type-safety:

```
Unset
+ import
"https://github.com/Vectorized/solady/blob/main/src/utils/LibString.s
ol";

    function tokenURI(uint256 tokenId) public view override returns
(string memory) {
-       return
string(abi.encodePacked("https://astaria.xyz/loans?id=", tokenId));
+       return string.concat("https://astaria.xyz/loans?id=",
LibString.toString(tokenId));
    }
```

Indeed, this latest fix wouldn't even allow the project to compile with a non-string parameter.

This finding is showcased as a Medium Severity one given the medium impact (`tokenURI` never working) and the high probability of making a mistake given:

- The absence of `import {LibString} from "solady/src/utils/LibString.sol";` in `Custodian.sol`

- The use of string casting
- The use of `abi.encodePacked` for concatenating strings in a previous commit
- The following tests under `TestCustodian.sol` that can be improved

```
Unset
File: TestCustodian.sol
105:     function testTokenURI() public {
106:
assertEq(custodian.tokenURI(uint256(keccak256(abi.encode(activeLoan))
)), "");
107:     }
108:
109:     function testTokenURIInvalidLoan() public {
110:
vm.expectRevert(abi.encodeWithSelector(Custodian.InvalidLoan.selector
));
111:         custodian.tokenURI(uint256(0));
112:     }
```

Proof of concept to try on Remix

```
Unset
// SPDX-License-Identifier: GPL-3.0
pragma solidity ^0.8.17;
import
"https://github.com/Vectorized/solady/blob/main/src/utils/LibString.s
ol";

contract TestStringConcat {
    function tokenURIBroken(uint256 tokenId)
        public
        pure
        returns (string memory)
    {
```

```
        return
            string(abi.encodePacked("https://astaria.xyz/loans?id=",
tokenId));
    }

    function tokenURIFixed(uint256 tokenId)
        public
        pure
        returns (string memory)
    {
        return
            string(
                abi.encodePacked(
                    "https://astaria.xyz/loans?id=",
                    LibString.toString(tokenId)
                )
            );
    }

    function tokenURIOptimized(uint256 tokenId)
        public
        pure
        returns (string memory)
    {
        // string.concat resulted in a compile error with `tokenId`
        return
            string.concat(
                "https://astaria.xyz/loans?id=",
                LibString.toString(tokenId)
            );
    }
```

Astaria's response: Fixed in commit 748326b. We accept the finding and have applied the recommended fix.

M-4. `defaultFeeRake` assumes 18 decimals

Severity: High
Probability: Low
Description:
Given the following in `Starport._feeRake()`:

- Starport.sol#L603-L614

```
Unset
File: Starport.sol
            if (debtItem.itemType == ItemType.ERC20) {
                Fee memory feeOverride = feeOverrides[debtItem.token];
                SpentItem memory feeItem = feeItems[i];
                feeItem.identifier = 0;
                amount = debtItem.amount.mulDiv(
                    !feeOverride.enabled ? defaultFeeRake :
feeOverride.amount, 10 ** ERC20(debtItem.token).decimals()
                );

                if (amount > 0) {
                    feeItem.amount = amount;
                    feeItem.token = debtItem.token;
                    feeItem.itemType = debtItem.itemType;
```

We can see that the unit used for `amount` is `[ERC20.decimals] * [WAD] / [ERC20.decimals] == [WAD]` However, when the `feeItem` is constructed, we have `feeItem.token = debtItem.token` and `feeItem.amount = amount`, which wouldn't be the right amount. Let's imagine `[ERC20.decimals] = 6` like with USDC: we'd get an amount above `1e17` (10% `defaultFeeRake` for USDC), which amounts to above hundreds of billions of USDC for fees.

Most likely, but not always, the call would revert here:

- Starport.sol#L625

```
Unset
            paymentToBorrower[i] = SpentItem({
                token: debtItem.token,
                itemType: debtItem.itemType,
                identifier: debtItem.identifier,
                amount: debtItem.amount - amount
            });
```

As for tokens with a lot of decimals, they'll make the fee being a dust amount.

In conclusion, the `defaultFeeRake` shouldn't be used for tokens that aren't 18 decimals.

Consider adding a check that `feeOverride.enabled == true` if `ERC20(debtItem.token).decimals() != 18`.

Astaria's response: Fixed in commit f0fc582. We agree with the finding and have reworked the calculation to calculate using basis points standardized across all decimal points.

M-5. Unsafe use of `transfer()`/`transferFrom()` with `IERC20`

Severity: Medium
Probability: Medium
Description:
Some tokens do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens.
For example **Tether (USDT)**'s `transfer()` and `transferFrom()` functions on L1 do not return booleans as the specification requires, and instead have no return value. When these sorts of tokens are cast to `ERC20`, their function signatures match but they're not complying to the interfaces they're cast to, so when a call is made, it reverts (see this link for a test case to use on Remix).
Use OpenZeppelin's `SafeERC20`'s `safeTransfer()`/`safeTransferFrom()` instead.

Affected code:

- src/Custodian.sol

```
Unset
File: Custodian.sol
387:             ERC20(offer.token).transfer(authorized, offer.amount);
```

Astaria's response: Fixed in commit 41af20b. We accept the finding and have made the recommended changes regarding using Solady `SafeTransferLib.safeTransfer()`.

M-6. Fee-On-Transfer tokens are not explicitly mentioned as unsupported

Severity: Medium
Probability: Medium

Description:
This is missing in the whitepaper at the time of the audit. The collateral amount being transferred to the Custodian won't be the amount held by the custodian. Hence, when repaying or on settlement, a higher amount than what is held will be transferred out from the Custodian.

Recommendation:
Add "Fee-On-Transfer tokens are not supported" in the whitepaper, alongside Rebasing tokens

Astaria's response: Acknowledged. We accept this finding and will update the Starport whitepaper to reflect the lack of support for fee-on-transfer tokens.

Low Severity Concerns

L-1. Immutable `_DOMAIN_SEPARATOR`

Severity: Low
Probability: Low
Description:
`_DOMAIN_SEPARATOR` is immutable or only defined in the `constructor`. As noted in https://eips.ethereum.org/EIPS/eip-2612#security-considerations this may lead to replay attacks in case of a future chain split.
Remediation: Consider using the implementation from OpenZeppelin, which recalculates the domain separator if the current `block.chainid` is not the cached chain ID.
Past occurrences of this issue:

- Reality Cards Contest
- Swivel Contest
- Malt Finance Contest

See Starport.sol#L397-L422:

Unset

```
File: Starport.sol
    function hashCaveatWithSaltAndNonce(
        address account,
        bool singleUse,
        bytes32 salt,
        uint256 deadline,
        CaveatEnforcer.Caveat[] calldata caveats
    ) public view virtual returns (bytes32) {
        return keccak256(
            abi.encodePacked(
                bytes1(0x19),
                bytes1(0x01),
                _DOMAIN_SEPARATOR,
                keccak256(
```

```
                abi.encode(
                    INTENT_ORIGINATION_TYPEHASH,
                    account,
                    caveatNonces[account],
                    singleUse,
                    salt,
                    deadline,
                    keccak256(abi.encode(caveats))
                )
            )
        )
    );
```

Astaria's response: Fixed in commit 7f1d2ca. We accept the finding and have implemented a `CACHED_DOMAIN_SEPARATOR`.

L-2. The Custodian contract shouldn't be an authorized collateral recipient on settlement

Severity: Low
Probability: Low
Description:
If, by mistake/misunderstanding of the protocol, users set the `authorized == loan.custodian` inside their `settlement`, the funds would be locked:

- Custodian.sol#L249-L250

```
Unset
    } else if (authorized == loan.terms.settlement || authorized ==
loan.issuer) {
        _moveCollateralToAuthorized(loan.collateral, authorized);
```

Consider disallowing it:

```
Unset
File: Custodian.sol
- 246:             if (authorized == address(0) || fulfiller ==
authorized) {
+ 246:             if (authorized == address(0) || authorized ==
address(this) || fulfiller == authorized) {
247:                 offer = loan.collateral;
248:                 _setOfferApprovalsWithSeaport(loan);
```

Astaria's response: Fixed in commit 7f1d2ca. We accept the finding, and have committed a change resolving this issue.

L-3. As `AmountDeriver._locateCurrentAmount` can underflow, there should exist a check that `block.timestamp >= start`

Severity: Low
Probability: Low
Description:

Given this note: AmountDeriver.sol#L25-L29, it would be great to check that `block.timestamp` is above `start` at the following places calling `_locateCurrentAmount` to avoid an underflow:

- DutchAuctionSettlement.sol#L124
- AstariaV1Settlement.sol#L65
- AstariaV1Settlement.sol#L158

The constructed loans inside the protocol shouldn't be affected by this. However, a third party interacting with the protocol or building on the protocol could receive some invalid data depending on their input.

Recommendation:
Check that `block.timestamp >= start`.

Astaria's response: Fixed in commit e390d2e. We accept the finding and have made the recommended fix.

L-4. A non-default Custodian could omit the calls to `postSettlement` or `postRepayment`, opening the path to a frontrunning attack

Severity: Low
Probability: Low
Description:

`postSettlement` and `postRepayment` functions are used in the `Starport.refinance()`, and `Custodian.generateOrder()`.
As non-default Custodians are authorized, the calls to `postSettlement` and `postRepayment` may be omitted, which would open the path to a frontrunning attack on such Custodians.

If we look at Astaria `v1-core`, the functions execute `BaseRecall.withdraw()`:

```
Unset
File: AstariaV1Settlement.sol
102:     function postSettlement(Starport.Loan calldata loan, address)
external virtual override returns (bytes4) {
103:         (address recaller,) =
BaseRecall(loan.terms.status).recalls(loan.getId());
104:         _executeWithdraw(loan, recaller);
105:         return Settlement.postSettlement.selector;
106:     }
...
109:     function postRepayment(Starport.Loan calldata loan, address
fulfiller) external virtual override returns (bytes4) {
110:         _executeWithdraw(loan, fulfiller);
111:
112:         return Settlement.postRepayment.selector;
113:     }
```

This must indeed be done here as, on refinance, settlement or repayment: the loan is set to inactive. From the moment this is done, `BaseRecall.withdraw()` can be called with the `loan`'s data, and a `receiver`: BaseRecall.sol#L148

Failing to call withdraw in the same transaction where the loan is set to inactive would let anyone claim the recaller's staked funds

Recommendation:
Make sure this is clearly explained in the integration guidelines.

Astaria's response: Acknowledged. We accept the finding and will ensure it is clearly explained in the integration guidelines.

L-5. The protocol should round up on incoming funds and round up on outgoing funds

Severity: Low
Probability: Low
Description:
In `_feeRake`, `feeItem.amount` is calculated using a `mulDiv`, which rounds down. Then, `PaymentToBorrower.amount = debtItem.amount - feeItem.amount`, which is akin to a round up in favor of the borrower.

It's best practice to round down on outgoing funds to users and tp round up on incoming funds to the protocol.

Consider using `mulDivUp` instead of `mulDiv` when calculating `feeItem.amount`: Starport.sol#L669

Astaria's response: Fixed in commit 67b3182. We accept the finding and have implemented the recommended fix.

L-6. `Starport`: `originate()` and `refinance()` are frontrunnable when all caveats are provided

Severity: Low
Probability: Low
Description:

When all caveats are provided, the calls to `originate()` and `refinance()` are frontrunnable.

On `originate()`: if `loan.originator == address(0)`, the first caller would become the `loan.originator` in `_issueLoan`.

On `refinance()`: while there's no impact in the `Starport` repo, in `v1-core` it becomes a race to redeem the staked recalled funds on `AstariaV1Status`.

Remediation: While this behavior is intended, it could surprise some users. Consider highlighting those behavior in the documentation.

Astaria's response: Acknowledged. We accept this finding and will annotate the behavior in the documentation.

L-7. If there's a `carryRate`, `ERC721` tokens will be locked

Severity: Low
Probability: Low
Description:

In `getPaymentConsideration()`, if it happens that `carryRate` is set to non-zero, there may be 2 `SpentItem` with the same ERC721 identifier, which means that the first transfer will work but the second one will fail: BasePricing.sol#L88-L100. This means that Repayment, Refinance and Settlement operations will always fail, effectively locking the ERC721 token

Recommendation:
There should never be ERC721 tokens combined with a concept of `carryRate != 0` or even `rate != 0`

Astaria's response: Acknowledged. We accept this finding, however this is by design as carry cannot be applied to an ERC-721. We will document this for implementers.

L-8. `decimals()` is not a part of the ERC-20 standard

Severity: Low
Probability: Low
Description:
The `decimals()` function is not a part of the ERC-20 standard, and was added later as an optional extension. As such, some valid ERC20 tokens do not support this interface, so their use would make the `originate()` function revert at the following line:

- Starport.sol#L610

```
Unset
    amount = debtItem.amount.mulDiv(
        !feeOverride.enabled ? defaultFeeRake : feeOverride.amount, 10
** ERC20(debtItem.token).decimals()
    );
```

Astaria's response: Fixed in commit a653644. We have reworked our fee calculations to be independent of decimals of the base token.

L-9. Solidity version 0.8.20 may not work on other chains due to `PUSH0`

Severity: Low
Probability: Low
Description:
The usage of floating `pragma solidity ^0.8.17` (not recommended) makes it possible for the project to be compiled with 0.8.20.

The compiler for Solidity 0.8.20 switches the default target EVM version to Shanghai, which includes the new `PUSH0` op code. This op code may not yet be implemented on all L2s, so deployment on these chains will fail. To work around this issue, use an earlier EVM version.

Astaria's response: Acknowledged. We accept the finding, and will note in our documentation regarding compilation and deployments.

L-10. Owner can renounce while system is paused

Severity: Low
Probability: Low
Description:
The contract owner or single user with a role is not prevented from renouncing the role/ownership while the contract is paused, which would cause any user assets stored in the protocol, to be locked indefinitely.

Affected code:

- src/lib/PausableNonReentrant.sol

```
Unset
# File: src/lib/PausableNonReentrant.sol

PausableNonReentrant.sol:71:    function pause() external onlyOwner {

PausableNonReentrant.sol:87:    function unpause() external onlyOwner
{
```

Astaria's response: Acknowledged. We accept the finding, and will note this condition in our documentation.

L-11. `defaultFeeRake` and `overrideValue` should be bounded

Severity: Low
Probability: Low
Description:
To increase trust in the protocol, `defaultFeeRake_` should be upper bounded to a reasonable value, like 30%:

- Starport.sol#L322-L326

```
Unset
File: Starport.sol
    function setFeeData(address feeTo_, uint88 defaultFeeRake_)
external onlyOwner {
        feeTo = feeTo_;
+         require(defaultFeeRake <= 3e17, "Fees are too high");
        defaultFeeRake = defaultFeeRake_;
        emit FeeDataUpdated(feeTo_, defaultFeeRake_);
    }
```

- Starport.sol#L334-L337

```
Unset
File: Starport.sol
    function setFeeOverride(address token, uint88 overrideValue, bool
enabled) external onlyOwner {
+         require(overrideValue <= 3e17, "Fees are too high");
        feeOverrides[token] = Fee({enabled: enabled, amount:
overrideValue});
        emit FeeOverrideUpdated(token, overrideValue, enabled);
    }
```

Given that those functions are behind the `onlyOwner` modifier, the extra gas cost shouldn't be a concern.

Astaria's response: Fixed in commit a653644. We accept the finding and have made the recommended changes.

L-12. DutchAuctionSettlement.validate(): window can be 0

Severity: Low
Probability: Low
Description:

There should be a check that `window` is not 0 in `validate()`: DutchAuctionSettlement.sol#L68

Otherwise, this would result in a divide by zero in `_locateCurrentAmount()`'s assembly: DutchAuctionSettlement.sol#L131

As this would return a zero instead of reverting (dividing by zero in assembly results in zero), consider adding a check to `validate()`.

Astaria's response: Fixed in commit 748326b. We accept the finding and have made the recommended changes.

L-13. Stargate is unknown

Severity: Low
Probability: Low
Description:

The Stargate contract isn't deployed onchain yet. Still, it is declared as immutable and the interface it uses isn't inherited from an official source, therefore there could be a mismatch in function signatures:

```
Unset
File: Starport.sol
44: interface Stargate {
45:     function getOwner(address) external returns (address);
46: }
...
96:     Stargate public immutable SG;
```

Astaria's response: Acknowledged. We accept the finding and will set the Stargate immutable address to `address(0)` on deployment and note the finding in our documentation.

L-14. DutchAuctionSettlement assumes a debt array of length 1

Severity: Low
Probability: Low
Description:

DutchAuctionSettlement.sol should be moved to v1-core since it assumes that debt array is always of length 1 (see DutchAuctionSettlement.sol#L139)

Astaria's response: Acknowledged. To mitigate, we will move `DutchAuctionSettlement.sol` into v1-core.

L-15. DutchAuctionSettlement assumes startPrice = endPrice = 1 for an ERC721 token

Severity: Low
Probability: Low
Description:
DutchAuctionSettlement.sol#L127-L133: for an ERC721 token we need to have `startPrice = endPrice = 1` but this isn't enforced or validated.

Astaria's response: Acknowledged. This is a valid concern regarding `DutchAuctionSettlement.sol`. We plan to move `DutchAuctionSettlement.sol` to v1-core due to the nature of it's specificity and inflexibility to support ERC-721, and ERC-1155s as debt tokens.

Informational Concerns

I-1. Extra warnings will need to be given to users with funds approved to Starport and the use of `singleUse == false`

If, for "convenience", a user approves `type(uint256).max` of their ERC20 token to Starport, and sign a caveat with `singleUse == false`, then the whole balance from the user could be used as a loan, potentially against their consent.

Astaria's response: Acknowledged. We acknowledge this and it is a core design element of the `singleUse` parameter. Selection of this parameter will be explicit within the Astaria front end.

I-2. For user-friendliness, consider returning the final `loan` in `Starport.originate()` and `Starport.refinance()`

Given that `loan.start` and `loan.originator` can be changed during `Starport.originate()` and `Starport.refinance()`: the output `loan` won't be equal to the input one.

As the final `loan` is the one used to compute the `loanId`, consider returning it for ease of further interaction with the protocol.

Astaria's response: Acknowledged. We did not do this because the gas costs were prohibitive ~3000 gas units to return the `loan` struct.

I-3. There are still mentions of the `LoanManager` contract

```
Unset
src/Custodian.sol:
  403:      * @dev settle the loan with the LoanManager
  416:      * @dev settle the loan with the LoanManager
  429:      * @dev settle the loan with the LoanManager

src/Starport.sol:
  450:      * @dev Settle the loan with the LoanManager
```

Astaria's response: Fixed in commit f0fc582. We accept this finding and have applied a fix.

I-4. Renaming suggestions

```
Unset
File: Starport.sol
- 125:         SpentItem[] collateral; // array of collateral
+ 125:         SpentItem[] collaterals; // array of collateral
- 126:         SpentItem[] debt; // array of debt
+ 126:         SpentItem[] debts; // array of debt
- 156:     mapping(uint256 => uint256) public loanState;
+ 156:     mapping(uint256 => uint256) public loansStates;
```

```
Unset
File: AstariaV1Lib.sol
- 93:     function getBaseRecallRecallMax(bytes memory statusData)
internal pure returns (uint256 recallMax) {
+ 93:     function getBaseRecallMax(bytes memory statusData) internal
pure returns (uint256 recallMax) {
```

Astaria's response: Fixed in commit [be0b40b]

I-5. Refactoring suggestion: Use `loan.getId()` in `Custodian`'s mint functions

In `Custodian`, several uses of `loan.getId()` can be seen. However, the following places reimplement the functions:

- Custodian.sol#L162-L163

```
Unset
    function mint(Starport.Loan calldata loan) external {
        bytes memory encodedLoan = abi.encode(loan);
        uint256 loanId = uint256(keccak256(encodedLoan));
```

- Custodian.sol#L176-L177

```
Unset
    function mintWithApprovalSet(Starport.Loan calldata loan, address
approvedTo) external {
        bytes memory encodedLoan = abi.encode(loan);
        uint256 loanId = uint256(keccak256(encodedLoan));
```

Consider just using `uint256 loanId = loan.getId();`

Astaria's response: Fixed in commit 7f1d2ca.

I-6. Refactoring suggestion: make a private function for repeated code

Consider creating a private function as to not duplicate code:

- Custodian.sol#L162-L166

```
Unset
File: Custodian.sol
162:         bytes memory encodedLoan = abi.encode(loan);
163:         uint256 loanId = uint256(keccak256(encodedLoan));
164:         if (loan.custodian != address(this) || !SP.active(loanId))
{
165:             revert InvalidLoan();
166:         }
```

- Custodian.sol#L176-L180

```
Unset
File: Custodian.sol
176:         bytes memory encodedLoan = abi.encode(loan);
177:         uint256 loanId = uint256(keccak256(encodedLoan));
178:         if (loan.custodian != address(this) || !SP.active(loanId))
{
179:             revert InvalidLoan();
180:         }
```

Astaria's response: Fixed in commit 7f1d2ca.

I-7. Renaming suggestion: parameter `address borrower` on `StarportLib.validateSalt()` should be `address validator`

`StarportLib.validateSalt()` isn't only called on the `borrower`. Sometimes, it's on the `issuer` or the `lender`. Consider renaming the following:

- StarportLib.sol#L108

```
Unset
File: StarportLib.sol
106:     function validateSalt(
107:         mapping(address => mapping(bytes32 => bool)) storage
usedSalts,
- 108:         address borrower,
+ 108:         address validator,
109:         bytes32 salt
110:     ) internal {
```

Astaria's response: Fixed in commit 7f1d2ca.

I-8. Delete unused errors, or use them

The following errors are not used: consider using them where appropriate or deleting them:

```
Unset
File: BNPLHelper.sol
79:     error DoNotSendETH();
```

```
Unset
File: Starport.sol
57:     error AdditionalTransferError();
58:     error CannotTransferLoans();
...
67:     error InvalidRefinance();
```

```
Unset
File: LenderEnforcer.sol
35:     error LenderOnlyEnforcer();
```

```
Unset
File: StrategistOriginator.sol
50:     error InvalidCustodian();
```

Astaria's response: Fixed in commit f0fc582.

I-9. `maximumSpent` isn't used on `Custodian.generateOrder` and can be removed

- Custodian.sol#L222

```
Unset
File: Custodian.sol
207:     /**
208:      * @dev Generates the order for this contract offerer
209:      * @param fulfiller The address of the contract fulfiller
- 210:      * @param maximumSpent The maximum amount of items to be
spent by the order
211:      * @param context The context of the order
212:      * @return offer The items spent by the order
213:      * @return consideration The items received by the order
214:      */
215:     function generateOrder(
216:         address fulfiller,
217:         SpentItem[] calldata,
- 218:         SpentItem[] calldata maximumSpent,
+ 218:         SpentItem[] calldata,
```

Astaria's response: Fixed in commit 748326b. We accept the finding and have applied the recommended fix.

I-10. References to the old naming `Loan Manager` or `LM` instead of `Starport`

```
Unset
starport/src/BNPLHelper.sol:
  98:         address lm;

starport/src/Custodian.sol:
  475:      * @dev  Hook to call before the loan is settled with the LM
  481:      * @dev  Hook to call after the loan is settled with the LM

starport/src/Starport.sol:
  658:      * @dev Issues a LM token if needed, only owner can call

starport/src/originators/StrategistOriginator.sol:
  163:             start: uint256(0), // Set in the loan manager
  164:             originator: address(0), // Set in the loan manager
```

Astaria's response: Fixed in commit f0fc582.

I-11. `Starport.originate()` shouldn't be `payable`

While using the `payable` keyword makes the function cheaper by 24 gas, it puts at risk users' funds that could make a mistake (albeit at a very low probability).

Even if it means paying 24 more gas, it's recommended to remove the `payable` keyword

Astaria's response: Acknowledged. We accept the finding and have opted not to make a fix for gas savings of a nonpayable function.

I-12. Consider renaming `open` to `opened` to match `closed`

- Starport.sol#L91
- Starport.sol#L474-L476

Astaria's response: Acknowledged. We accept the finding, but will not make any changes.

I-13. Consider adding the `name` field to `EIP712Domain`

- Starport.sol#L99-L100
- StrategistOriginator.sol#L75

Astaria's response: Fixed in commit 7f1d2ca.

I-14. Missing pragma in `PausableNonReentrant.sol`

This file is missing the pragma directive: PausableNonReentrant.sol#L1-L29

Astaria's response: Fixed in commit 41af20b.

I-15. Constants should be in CONSTANT_CASE

For `constant` variable names, each word should use all capital letters, with underscores separating each word (CONSTANT_CASE)

Affected code:

- src/BNPLHelper.sol

```
Unset
85:     address private constant vault =
address(0xBA12222222228d8Ba445958a75a0704d566BF2C8);
```

- src/Custodian.sol

```
Unset
75:    address public immutable seaport;
```

Astaria's response: Fixed in commit 41af20b.

I-16. Default Visibility for constants

Some constants are using the default visibility. For readability, consider explicitly declaring them as `internal`.

Affected code:

- src/originators/StrategistOriginator.sol

```
Unset
# File: src/originators/StrategistOriginator.sol

StrategistOriginator.sol:75:    bytes32 constant EIP_DOMAIN =
keccak256("EIP712Domain(string version,uint256 chainId,address
verifyingContract)");

StrategistOriginator.sol:77:    bytes32 constant VERSION =
keccak256("0");
```

Astaria's response: Fixed in commit 41af20b. We accept the finding, and have made the constants `public`.

I-17. Consider using named mappings

Consider moving to solidity version 0.8.18 or later, and using named mappings to make it easier to understand the purpose of each mapping

Affected code:

- src/Starport.sol

```
Unset
# File: src/Starport.sol

Starport.sol:152:    mapping(address => Fee) public feeOverrides;
```

```
Starport.sol:153:     mapping(address => mapping(address =>
ApprovalType)) public approvals;

Starport.sol:154:     mapping(address => mapping(bytes32 => bool))
public invalidSalts;

Starport.sol:155:     mapping(address => uint256) public caveatNonces;

Starport.sol:156:     mapping(uint256 => uint256) public loanState;
```

- src/lib/RefStarportLib.sol

```
Unset
# File: src/lib/RefStarportLib.sol
RefStarportLib.sol:58:         mapping(address => mapping(bytes32 =>
bool)) storage usedSalts,
```

- src/lib/StarportLib.sol

```
Unset
# File: src/lib/StarportLib.sol
StarportLib.sol:107:         mapping(address => mapping(bytes32 =>
bool)) storage usedSalts,
```

- src/originators/StrategistOriginator.sol

```
Unset
# File: src/originators/StrategistOriginator.sol
StrategistOriginator.sol:84:     mapping(bytes32 => bool) public
usedHashes;
```

Astaria's response: Acknowledged. We accept the finding, but will not make any changes.

I–18. `StarportLib.transferSpentItemsSelf()` shouldn't take a `from` parameter

In the fixed code in StarportLib, function transferSpentItemsSelf transfers ERC20 from itself, however it also allows a from as input argument for ERC721 and ERC1155. The StrategistOriginator calls the function with from == address(this), so it's fine, but it would be less error prone and more consistent to remove the from parameter in transferSpentItemsSelf().

Astaria's response: Fixed.

Gas Optimizations Recommendations

G-1. Consider checking the `allowance` before calling `ERC20.approve()`

In Custodian, if `offer.itemType == ItemType.ERC20`, there's always a call to `approve` with `type(uint256).max`: Custodian.sol#L369-L371.
However, this is a `call()`, which is a lot more expensive than a `staticcall()` (`staticcall()` is generally executed when you call a function marked as view or pure on an external contract).
Consider checking if the `allowance()` isn't already `type(uint256).max` before calling `approve()`, so as to save gas for all users after the first one who used a certain type of ERC20 token.

Astaria's response: Fixed in commit a6aaccb.

G-2. Use the lighter version of `ERC721.safeMint()`

Given that the `loanId` already holds the information from the `loan`, using the `encodedLoan` data in the `bytes` field for `safeMint()` seems redundant: Custodian.sol#L167

Consider using the version of `safeMint()` without the bytes data:
https://github.com/Vectorized/solady/blob/68fe9b5829467515cae89079fa7aea7bcdbf838a/src/tokens/ERC721.sol#L479

Astaria's response: Fixed in commit 7f1d2ca.

G-3. Redundant operations can be deleted

Given that named returns already have a default value, it's not necessary to assign that same value at the end of the function. Hence, the following can be deleted:

- BasePricing.sol#L80
- SimpleInterestPricing.sol#L89
- DutchAuctionSettlement.sol#L126
- StarportLib: `new bytes(0)` can be replaced with `""` (#L253, #L284, #L361, #L385)

Astaria's response: Fixed in commit d760601. We accept the finding and have removed `BaseRecallPricing.sol` since all the methods were overridden by `AstariaV1Pricing.sol`.

G-4. `BNPLHelper.activeUserDataHash` can be deleted

- BNPLHelper.sol#L91

Given that `activeUserDataHash` is set in the same transaction in which it's deleted, the checks around it are redundant with the fact that there are just 2 functions (`makeFlashLoan`, called by the user, and `receiveFlashLoan`, called by the balancer vault) and no state changes.

Astaria's response: Fixed in commit a6aaccb.

G-5. Unchecking arithmetics operations that can't underflow/overflow

Solidity version 0.8+ comes with implicit overflow and underflow checks on unsigned integers. When an overflow or an underflow isn't possible (as an example, when a comparison is made before the arithmetic operation), some gas can be saved by using an `unchecked` block: https://docs.soliditylang.org/en/v0.8.10/control-structures.html#checked-or-unchecked-arithmetic

Consider wrapping with an `unchecked` block where it's certain that there cannot be an underflow

25 gas saved per instance

Affected code:

- src/Starport.sol

```
Unset
# File: src/Starport.sol

Starport.sol:323:         uint256 newNonce = caveatNonces[msg.sender] +
1 + uint256(blockhash(block.number - 1) >> 0x80);
```

- src/originators/StrategistOriginator.sol

```
Unset
# File: src/originators/StrategistOriginator.sol

StrategistOriginator.sol:150:         _counter +=
uint256(blockhash(block.number - 1) >> 0x80);
```

- src/settlement/DutchAuctionSettlement.sol

```
Unset
# File: src/settlement/DutchAuctionSettlement.sol

+ DutchAuctionSettlement.sol:144:             uint256 excess =
settlementPrice - (loan.debt[0].amount + interest - carry);
```

Astaria's response: Fixed in commit a6aaccb.

G-6. Cache array length outside of loop

If not cached, the solidity compiler will always read the length of the array during each iteration. That is, if it is a storage array, this is an extra sload operation (100 additional extra gas for each iteration except for the first) and if it is a memory array, this is an extra mload operation (3 additional gas for each iteration except for the first).

Affected code:

- src/BNPLHelper.sol

```
Unset
# File: src/BNPLHelper.sol

BNPLHelper.sol:133:         for (uint256 i = 0; i < tokens.length;) {

BNPLHelper.sol:144:         for (uint256 i = 0; i < tokens.length;) {
```

- src/Custodian.sol

```
Unset
# File: src/Custodian.sol

Custodian.sol:380:         for (uint256 i = 0; i <
loan.collateral.length; i++) {

Custodian.sol:407:         for (uint256 i = 0; i < offer.length; i++) {
```

- src/Starport.sol

```
Unset
# File: src/Starport.sol

Starport.sol:344:         for (uint256 i = 0; i <
defaultFeeRakeByDecimals_.length;) {
```

```
Starport.sol:389:            for (; i < considerationPayment.length;)
{

Starport.sol:402:            for (; i < considerationPayment.length;)
{

Starport.sol:432:        for (; i < caveats.length;) {

Starport.sol:532:        for (; i < additionalTransfers.length;) {

Starport.sol:556:        for (; i < additionalTransfers.length;) {

Starport.sol:600:        for (uint256 i = 0; i <
signedCaveats.caveats.length;) {

Starport.sol:653:        for (uint256 i = 0; i < debt.length;) {
```

- src/enforcers/BorrowerEnforcer.sol

Unset

```
# File: src/enforcers/BorrowerEnforcer.sol

BorrowerEnforcer.sol:85:            for (; i <
additionalTransfers.length;) {
```

- src/enforcers/LenderEnforcer.sol

Unset

```
# File: src/enforcers/LenderEnforcer.sol

LenderEnforcer.sol:78:            for (; i <
additionalTransfers.length;) {
```

- src/lib/StarportLib.sol

```
Unset
# File: src/lib/StarportLib.sol

StarportLib.sol:150:         for (; i < payment.length;) {

StarportLib.sol:172:             for (; i < carry.length;) {

StarportLib.sol:210:         for (uint256 i = 0; i <
consideration.length;) {

StarportLib.sol:236:         for (; i < transfers.length;) {

StarportLib.sol:267:         for (; i < transfers.length;) {

StarportLib.sol:299:             for (; i < transfers.length;) {
```

- src/originators/StrategistOriginator.sol

```
Unset
# File: src/originators/StrategistOriginator.sol

StrategistOriginator.sol:226:         for (uint256 i = 0; i <
request.debt.length;) {
```

- src/pricing/BasePricing.sol

```
Unset
# File: src/pricing/BasePricing.sol

BasePricing.sol:85:         for (; i < loan.debt.length;) {
```

Astaria's response: Acknowledged. We accept the finding, but will not make any changes as we are optimizing the protocol for array sizes of 1, with support for larger array sizes at higher gas profiles.

G-7. `++i` costs less gas compared to `i++` or `i += 1`

Pre-increments and pre-decrements are cheaper.

For a `uint256 i` variable, the following is true with the Optimizer enabled at 10k:

Increment:

- `i += 1` is the most expensive form
- `i++` costs 6 gas less than `i += 1`
- `++i` costs 5 gas less than `i++` (11 gas less than `i += 1`)

In the pre-increment case, the compiler has to create a temporary variable (when used) for returning `1` instead of `2`.

Consider using pre-increments where they are relevant (meaning: not where post-increments/decrements logic are relevant).

Saves 5 gas per instance

Affected code:

- src/Custodian.sol

```
Unset
# File: src/Custodian.sol

Custodian.sol:380:         for (uint256 i = 0; i <
loan.collateral.length; i++) {

Custodian.sol:407:         for (uint256 i = 0; i < offer.length; i++) {
```

- src/originators/StrategistOriginator.sol

```
Unset
# File: src/originators/StrategistOriginator.sol

StrategistOriginator.sol:242:                 i++;
```

G-8. Increments/decrements can be unchecked

In Solidity 0.8+, there's a default overflow check on unsigned integers. It's possible to uncheck this in for-loops and save some gas at each iteration, but at the cost of some code readability, as this uncheck cannot be made inline.

ethereum/solidity#10695

The change would be:

```
Unset
- for (uint256 i; i < numIterations; i++) {
+ for (uint256 i; i < numIterations;) {
  // ...
+   unchecked { ++i; }
}
```

These save around **25 gas saved** per instance.

The same can be applied with decrements (which should use `break` when `i == 0`).

The risk of overflow is non-existent for `uint256`.

Affected code:

- src/Custodian.sol

```
Unset
# File: src/Custodian.sol

Custodian.sol:380:         for (uint256 i = 0; i <
loan.collateral.length; i++) {

Custodian.sol:407:         for (uint256 i = 0; i < offer.length; i++) {
```

Astaria's response: Fixed in commit a6aaccb.

G-9. Unused `StrategistOriginator.strategistFee`

StrategistOriginator has a strategistFee
This can be retrieved via getStrategistData(), but it isn't used/checked anywhere else in the code. Consider deleting it.
Astaria's response: Fixed in commit 748326b.

G-10. The `_moveCollateralToAuthorized()` path is less expensive

In Custodian.generateOrder(), after the call to getSettlementConsideration(), if the DutchAuction has failed and the collateral should be sent back to the loan.issuer, then the path taken when the order is filled by the loan.issuer will be the Seaport one (fulfiller == authorized and authorized == loan.issuer), whereas the _moveCollateralToAuthorized() path is quite likely less expensive but is only reachable when the order is filled by a third party.

Astaria's response: Acknowledged. Your description is correct. It might be more gas efficient to use `_moveCollateralToAuthorized` but would be best not to make the change this close to code freeze

G-11. Cache `_counter` in `StrategistOriginator.incrementCounter()`

To save some gas: In StrategistOriginator, the function incrementCounter(), could use a temporary variable (for the state variable _counter), similar to incrementCaveatNonce() in Starport.sol

Astaria's response: Fixed in commit 748326b. We accept the finding and have a applied a modified fix.

G-12. Cache `Status.isActive` in `Custodian.generateOrder()`

To save some gas: In Custodian, the function generateOrder(), could use a temporary variable for Status(loan.terms.status).isActive(loan, close.extraData)

Astaria's response: Fixed in commit 748326b. We accept the finding and have applied the recommended fix.

Formal Verification

Assumptions and Simplifications Made During Verification

General Assumptions

A. Any loop was unrolled to two iterations at most.

Token transfers summarization

When transfer of any of the `ERC20/ERC721/ERC1155` tokens occurred, instead of using a contract implementation, we used a ghost mapping to monitor and store the relevant transfers. We checked manually the approval mechanism implemented by `_enableAssetWithSeaport()` and during verification assumed that it works correctly.

Code refactoring and explicit summarizations of internal parts of the code

Some functions had as input `bytes calldata` that represented encoded structs. We had to refactor the code and present internal functions that were having as input the explicit structs instead. This way we could prove in a faster and more efficient manner any properties related to the parameters passed within those structs. In addition, in `Starport.sol`, the `compute()` function was introduced to wrap the `mulDiv` computation in the _feeRake() function, thus reducing the Prover's run time.

Replacing explicit keccak computations with equivalent getId() call and summarizing them

in `Custodian.sol`, the `_validateAndMint()` function used explicit `keccak` computations to calculate the `loanId` of the input loan. Instead of the explicit computation, we used the call to the `getId()` function already implementing the exact logic within `StarportLib.sol`. This modification allowed us later to summarize the computation to explicit value, and reduce the Prover's run time without impacting the correctness of the properties we proved.

Solady implementations replaced by OpenZeppelin's implementations

The mostly-assembly implementation of the imported Solady contracts used in the protocol caused some issues (like longer running time) to the Prover. We used instead the OpenZeppelin implementations of the `ERC721` and `PausableNonReentrant` contracts. Later, we individually proved properties on the original Solady contract `PausableNonReentrant` which can be seen in the list of properties below.

Formal Verification Properties

Notations

✅Indicates the rule is formally verified.
❌Indicates the rule is violated.

Since the protocol consists of different contracts, we will present the relative properties for each of the main contracts in separate sections.

The following files were formally verified, and the properties are listed below per library/contract:

A. Custodian.sol
B. Starport.sol
C. PausableNonReentrant.sol

Custodian.sol

Assumptions

- We verified the contract functions against an arbitrary storage state.

Properties

1. ✅ Only the seaport contract address can call the ratifyOrder() and generateOrder() methods
2. ✅ One cannot mint a custody token for a closed loan
3. ✅ One cannot mint a custody token if the custodian of the loan is not the same as the custodian contract
4. ✅ One cannot mint and approve a loan if msg.sender is not the loan's borrower
5. ✅ One cannot call mint() twice using the same Starport.Loan loan
6. ✅ After minting a ERC721 token for a loan, the owner of the token must be loan.borrower
7. ✅ Custodian cannot settle a loan when loan.custodian field differs from custodian.
8. ✅ getBorrower returns the owner of the token or the loan.borrower
9. ✅ If previewOrder() reverted then generateOrder() would also revert
10. ✅ generateOrder() can only settle an open loanId, and the requested action can only be Actions.Repayment of active loan or Actions.Settlement of inactive loan
11. ✅ Only the party returned by Custodian._getBorrower(loan) (equal to ownerOf(loanId) or loan.issuer) is able to repay a loan or the fulfiller is approved, and the collateral will be sent to the fulfiller of the order
12. ✅ Only authorized party is able to settle a loan if the authorized address is not the loan.terms.settlement or loan.issuer.

Starport.sol

Assumptions

- We verified the contract functions against an arbitrary storage state.

Properties

1. ✅ setOrignateApproval sets the correct values
2. ✅ setFeeData sets the correct values
3. ✅ setFeeOverride sets the correct values
4. ✅ Issuing a loan results in a open loan
5. ✅ Settling a loan results in a closed loan and can only be performed by the custodian of the loan
6. ✅ Only the message sender can change the caveat nonce
7. ✅ Only the message sender can change the approvals
8. ✅ Settling an closed loan is not possible
9. ✅ Settling a loan can only close the input loan, and doesn't change the status of any other loan
10. ✅ Only the owner can call pause and unpause
11. ✅ One cannot originate a loan without providing a collateral
12. ✅ Calling originate() can create only one new active loan
13. ✅ Only the protocol owner can change fee settings (feeTo, defaultFeeRake, feeOverrides)
14. ✅ When originating a loan, if there are fees, the debt items will be split between the borrower and the fee collector, and their sum must be equal to the initial debt (no assets are lost or created due to fees)
15. ✅ After calling originate, the sum of the balances of the loan’s custodian, issuer, borrower, and the feeRecipient, remains the same
16. ✅ A closed loan cannot be refinanced
17. ✅ An already open loan cannot be originated
18. ✅ Invalidating salt indeed invalidates the various caveats as intended
19. ✅ One cannot originate a loan with unauthorized additional transfers

PausableNonReentrant.sol

Assumptions

- We verified the contract functions against an arbitrary storage state.

Properties

1. ✅ Only owner can call the pause(), unpause(), renounceOwnership(), transferOwnership() methods
2. ✅ The only method that can pause a contract is pause()
3. ✅ The only method that can unpause a contract is unpause()
4. ✅ Only owner can change the owner of the contract
5. ✅ Once the renounceOwnership() is executed correctly, no method can set a new owner
6. ✅ One cannot call transferOwnership() with the zero address (even the owner cannot)
7. ✅ Only owner can call completeOwnershipHandover()
8. ✅ To complete successfully ownership handover, the pending owner must have requested ownership handover in less than 48 hours prior to the handover
9. ✅ No one (that is not owner or the pending owner) cannot interfere with the handover
10. ✅ Integrity of ownershipHandoverExpiresAt(), i.e., it returns correctly the handover expiration, which is block.timestamp + 48 hours from the request ownership handover
11. ✅ cancelOwnershipHandover() affects only the msg.sender

Disclaimer

The Certora Prover takes a contract and a specification as input and formally proves that the contract satisfies the specification in all scenarios. Notably, the guarantees of the Certora Prover are scoped to the provided specification and the Certora Prover does not check any cases not covered by the specification.

Even though we hope this information is helpful, we provide no warranty of any kind, explicit or implied. The contents of this report should not be construed as a complete guarantee that the contract is secure in all dimensions. In no event shall Certora or any of its employees be liable for any claim, damages, or other liability, whether in an action of contract, tort, or otherwise, arising from, out of, or in connection with the results reported here.

About Certora

Certora is a Web3 security company that provides industry-leading formal verification tools and smart contract audits. Certora's flagship security product, Certora Prover, is a unique SaaS product that automatically locates even the most rare & hard-to-find bugs on your smart contracts or mathematically proves their absence. The Certora Prover plugs into your standard deployment pipeline. It is helpful for smart contract developers and security researchers during auditing and bug bounties.

Certora also provides services such as auditing, formal verification projects, and incident response.