PALADIN
BLOCKCHAIN SECURITY

Smart Contract Security Assessment

Final Report

For ApeSwap (Jungle Fund)

05 April 2022

paladinsec.co

info@paladinsec.co

Table of Contents

Table of Contents 2

Disclaimer 4

1 Overview 5

1.1 Summary 5

1.2 Contracts Assessed 5

1.3 Findings Summary 6

1.3.1 Global Issues 7

1.3.2 BillHelper 7

1.3.3 BillNft 8

1.3.4 CustomBillFactory 8

1.3.5 CustomTreasury 9

1.3.6 FactoryStorage 9

1.3.7 Policy 9

1.3.8 SubsidyRouter 10

1.3.9 CustomBill 11

2 Findings 12

2.1 Global Issues 12

2.1.1 Issues & Recommendations 13

2.2 BillHelper 17

2.2.1 Issues & Recommendations 18

2.3 BillNft 20

2.3.1 Privileged Functions 20

2.3.2 Issues & Recommendations 21

2.4 CustomBillFactory 28

2.4.1 Privileged Functions 28

2.4.2 Issues & Recommendations 29

2.5 CustomTreasury 32

2.5.1 Privileged Functions 32

2.5.2 Issues & Recommendations 33

2.6 FactoryStorage 37

2.6.1 Privileged Functions 37

2.6.2 Issues & Recommendations 38

2.7 Policy 40
2.7.1 Privileged Functions 40
2.7.2 Issues & Recommendations 41
2.8 SubsidyRouter 43
2.8.1 Privileged Functions 43
2.8.2 Issues & Recommendations 44
2.9 CustomBill 46
2.9.1 Privileged Functions 47
2.9.2 Issues & Recommendations 48

Disclaimer

Paladin Blockchain Security ("Paladin") has conducted an independent audit to verify the integrity of and highlight any vulnerabilities or errors, intentional or unintentional, that may be present in the codes that were provided for the scope of this audit. This audit report does not constitute agreement, acceptance or advocation for the Project that was audited, and users relying on this audit report should not consider this as having any merit for financial advice in any shape, form or nature. The contracts audited do not account for any economic developments that may be pursued by the Project in question, and that the veracity of the findings thus presented in this report relate solely to the proficiency, competence, aptitude and discretion of our independent auditors, who make no guarantees nor assurance that the contracts are completely free of exploits, bugs, vulnerabilities or deprecation of technologies. Further, this audit report shall not be disclosed nor transmitted to any persons or parties on any objective, goal or justification without due written assent, acquiescence or approval by Paladin.

All information provided in this report does not constitute financial or investment advice, nor should it be used to signal that any persons reading this report should invest their funds without sufficient individual due diligence regardless of the findings presented in this report. Information is provided 'as is', and Paladin is under no covenant to the completeness, accuracy or solidity of the contracts audited. In no event will Paladin or its partners, employees, agents or parties related to the provision of this audit report be liable to any parties for, or lack thereof, decisions and/or actions with regards to the information provided in this audit report.

Cryptocurrencies and any technologies by extension directly or indirectly related to cryptocurrencies are highly volatile and speculative by nature. All reasonable due diligence and safeguards may yet be insufficient, and users should exercise considerable caution when participating in any shape or form in this nascent industry.

The audit report has made all reasonable attempts to provide clear and articulate recommendations to the Project team with respect to the rectification, amendment and/or revision of any highlighted issues, vulnerabilities or exploits within the contracts provided. It is the sole responsibility of the Project team to sufficiently test and perform checks, ensuring that the contracts are functioning as intended, specifically that the functions therein contained within said contracts have the desired intended effects, functionalities and outcomes of the Project team.

Paladin retains full rights over all intellectual property (including expertise and new attack or exploit vectors) discovered during the audit process. Paladin is therefore allowed and expected to re-use this knowledge in subsequent audits and to inform existing projects that may have similar vulnerabilities. Paladin may, at its discretion, claim bug bounties from third-parties while doing so.

1 Overview

This report has been prepared for ApeSwap's Jungle Fund contracts on the BNB Smart Chain. Paladin provides a user-centred examination of the smart contracts to look for vulnerabilities, logic errors or other issues from both an internal and external perspective.

1.1 Summary

Project Name	ApeSwap (Jungle Fund)
URL	https://apeswap.finance/
Platform	BNB Smart Chain
Language	Solidity

1.2 Contracts Assessed

Name	Contract	Live Code Match
`BillHelper`	`Not used`	UNUSED
`BillNft`	`0xb0278e43dbd744327fe0d5d0aba4a77cbfc7fad8`	✓ MATCH
`CustomBillFactory`	`0x01ed9098ba2ea916bf7d6528b9dadd35c2072337`	✓ MATCH
`CustomTreasury`	`0x2e0765fEDD4bCe99bF95E90d22d7d397042d175e`	✓ MATCH
`FactoryStorage`	`0x2013a058d339b95e69ec73eaaaad990649d43ab7`	✓ MATCH
`Policy`	`Dependency`	✓ MATCH
`SubsidyRouter`	`0xd589aa30a456e78f3d3ffb4eff270c941e5a7a8f`	✓ MATCH
`CustomBill`	`0x8b57Fc5BE65118188D50d42fcD5614e447F7FAbE`	✓ MATCH

1.3 Findings Summary

Severity	Found	Resolved	Partially Resolved	Acknowledged (no change made)
High	2	1	-	1
Medium	3	3	-	-
Low	6	6	-	-
Informational	44	42	-	2
Total	**55**	**52**	**-**	**3**

Classification of Issues

Severity	Description
High	Exploits, vulnerabilities or errors that will certainly or probabilistically lead towards loss of funds, control, or impairment of the contract and its functions. Issues under this classification are recommended to be fixed with utmost urgency.
Medium	Bugs or issues that may be subject to exploit, though their impact is somewhat limited. Issues under this classification are recommended to be fixed as soon as possible.
Low	Effects are minimal in isolation and do not pose a significant danger to the project or its users. Issues under this classification are recommended to be fixed nonetheless.
Informational	Consistency, syntax or style best practices. Generally pose a negligible level of risk, if any.

1.3.1 Global Issues

ID	Severity	Summary	Status
01	HIGH	Governance risk	ACKNOWLEDGED
02	INFO	Inside the entire code base there is a mix of `principleToken` and `principalToken`	RESOLVED
03	INFO	SafeMath is no longer required within Solidity ^0.8	ACKNOWLEDGED
04	INFO	Inconsistent licensing	RESOLVED

1.3.2 BillHelper

ID	Severity	Summary	Status
05	INFO	`batchRedeem` does not return the total payout	ACKNOWLEDGED
06	INFO	`batchRedeem` uses 2 arrays but does not check that they have the same length	RESOLVED
07	INFO	Typographical errors	RESOLVED
08	INFO	Gas optimizations	RESOLVED

1.3.3 BillNft

ID	Severity	Summary	Status
09	MEDIUM	baseURI cannot be set due to uriUnlocked initializing to false in the upgradeable proxy	RESOLVED
10	INFO	The contract contains functions that can be made external	RESOLVED
11	INFO	Unchanged and inadequately overridden function	RESOLVED
12	INFO	The overridden functions should not be marked virtual if the contract is not expected to be inherited	RESOLVED
13	INFO	AccessControlEnumerableUpgradeable is not initialized	RESOLVED
14	INFO	Unused modifier throughout the contracts	RESOLVED
15	INFO	Typographical errors	RESOLVED
16	INFO	Lack of event	RESOLVED
17	INFO	Gas optimization: Re-using mint uses more gas than necessary	RESOLVED

1.3.4 CustomBillFactory

ID	Severity	Summary	Status
18	LOW	`createBill` sets the treasury parameter to `_customTreasury`	RESOLVED
19	INFO	Lack of events	RESOLVED
20	INFO	Unused imports	RESOLVED
21	INFO	Typographical errors	RESOLVED

1.3.5 CustomTreasury

ID	Severity	Summary	Status
22	INFO	Unused function throughout the contracts	RESOLVED
23	INFO	CustomTreasury is a proxy but does not use upgradeable OpenZeppelin dependencies	RESOLVED
24	INFO	`valueOfToken` can be made external	RESOLVED
25	INFO	Typographical errors and inconsistencies	RESOLVED

1.3.6 FactoryStorage

ID	Severity	Summary	Status
26	INFO	Lack of getter for `_tierCeilings` and `_fees`	RESOLVED
27	INFO	Fee recipient treasury is not saved within `BillDetails`	ACKNOWLEDGED
28	INFO	Lack of events	RESOLVED

1.3.7 Policy

ID	Severity	Summary	Status
29	HIGH	After policy has been renounced, it can be reclaimed by the pending policy through `pullPolicy`	RESOLVED
30	INFO	`_newPolicy` needs a getter	RESOLVED
31	INFO	Lack of events	RESOLVED

1.3.8 SubsidyRouter

ID	Severity	Summary	Status
32	INFO	Lack of events	RESOLVED
33	INFO	Typographical errors	RESOLVED
34	INFO	Gas Optimization: Cache values to save gas	RESOLVED

1.3.9 CustomBill

ID	Severity	Summary	Status
35	MEDIUM	DoS: A malicious party can block a specific address from redeeming for a certain period of time	RESOLVED
36	MEDIUM	The UI function `getBillIdsInRange` does not function	RESOLVED
37	LOW	Phishing risk: Deposits can be sent to another address	RESOLVED
38	LOW	CustomBill is a proxy but does not use upgradeable OpenZeppelin dependencies	RESOLVED
39	LOW	Adjustment target is never reached	RESOLVED
40	LOW	Lack of mint safeguards	RESOLVED
41	LOW	Private variables	RESOLVED
42	INFO	`trueBillPrice` is slightly inaccurate	RESOLVED
43	INFO	Unused imports	RESOLVED
44	INFO	Contract does not work with a zero vestingTerm	RESOLVED
45	INFO	Lack of validation	RESOLVED
46	INFO	Typographical error	RESOLVED
47	INFO	Gas optimization: Certain variables can be cached	RESOLVED
48	INFO	Usage of wildcards	RESOLVED
49	INFO	Indexing of events	RESOLVED
50	INFO	Lack of events	RESOLVED
51	INFO	Initializing terms prematurely adds unnecessary state	RESOLVED
52	INFO	BillPriceChanged event is emitted prematurely	RESOLVED
53	INFO	`batchRedeem` does not return the total payout	RESOLVED
54	INFO	The usage of FixedPoint throughout the contract does not contribute to the integrity of the contract and can therefore be removed	RESOLVED
55	INFO	`decayDebt` is not path independent	RESOLVED

2 Findings

2.1 Global Issues

The issues in this section are applicable to the entire protocol.

2.1.1 Issues & Recommendations

Issue #01	Governance risk
Severity	HIGH SEVERITY
Description	Any address that is added to the `policy` (the governance) is allowed to to make crucial modifications to the protocol, including but not limited to upgrading all the important aspects of the protocol since the contracts are upgradeable proxies. Some of the methods of abusing governance risk are mentioned below. As there are no safeguards, governance can use withdraw on the CustomTreasury contract with any amount of any token and send them to an address. The governance can drain any token and thus can drain the `payoutToken` (the main token) of the Treasury and then potentially dump this. Also, governance can add any address to `billContract` and that address will then be able to drain the `payoutToken` of the contract using `deposit` or `sendPayoutToken` with some malicious inputs. Governance can set the factory to a malicious address and will then be able to add a bill using `pushBill` with malicious input and contract to steal users' funds. Governance can adjust the CustomBill bond emission parameters to emit bonds to themselves at a too low price, to then potentially dump the vested tokens.

Recommendation	Consider designing a strong governance structure where it is unlikely and ideally impossible for the governance to abuse these privileges. Consider also keeping a bill registry within the bill factory instead of adding addresses manually. Or, at least, adding the following code to the `toggleBillContract` function to prevent the Governance from adding a non-bill contract: `require(factory.isBill(msg.sender), "Not a bill contract")` Finally, it should be noted that the client, ApeSwap, is a very reputable actor within the DeFi space. They've been around for a significant time and have shown adequate behavior and professionalism over that period. The main risk for this issue is therefore that governance is centralized within a single private key (or single entities have the ability to execute the aforementioned actions). This could cause theft or loss in case this private key was stolen, the single entities go rogue or similar situations where the organization overall remains honest but a single party is compromised. Generally speaking, a good governance structure ensures that not a single party can execute any of these critical functionalities. It is therefore ideal to lock all critical functionality behind a reputable multisig.
Resolution	ACKNOWLEDGED

Issue #02	Inside the entire code base there is a mix of `principleToken` and `principalToken`
Severity	INFORMATIONAL
Description	CustomBillContract mentions `pincipalToken` while the other contracts mentions `principleToken`. This is inconsistent and only one name should be used in the entire code base.
Recommendation	Consider sticking to a single name for this variable.
Resolution	RESOLVED

Issue #03	SafeMath is no longer required within Solidity ^0.8
Severity	INFORMATIONAL
Description	The most recent version of Solidity, version 0.8, has integrated the SafeMath overflow checks. By default, things like overflow, underflow, and division by zero will now cause a reversion even if traditional math operators are used. SafeMath is thus no longer required. Keeping SafeMath in will not only make the code slightly less legible, it will also make it more expensive gas-wise.
Recommendation	Consider removing the use of SafeMath.
Resolution	ACKNOWLEDGED

Issue #04	Inconsistent licensing
Severity	INFORMATIONAL
Description	Certain contracts are in-line licensed under the MIT license while others are licensed under "Unlicense".
Recommendation	Consider being consistent and always licensing under MIT. This issue will be resolved as well if the client has an argument for the different licenses.
Resolution	RESOLVED The codebase is now licensed partially under MIT and partially under GPL-3.0.

2.2 BillHelper

BillHelper is an extremely simple utility contract that allows for the redemption of the vested amount of multiple bills at once in a single transaction. It contains a single function, `batchRedeem`.

2.2.1 Issues & Recommendations

Issue #05	`batchRedeem` does not return the total payout
Severity	INFORMATIONAL
Description	`redeem` returns the payout of bill while `batchRedeem` does not return the sum of the payout of each bill. This is inconsistent.
Recommendation	`batchRedeem` should return the sum of each bill's payout.
Resolution	ACKNOWLEDGED

Issue #06	`batchRedeem` uses 2 arrays but does not check that they have the same length
Severity	INFORMATIONAL
Description	Users can provide the list of bill ids they wish to redeem on specific CustomBill addresses. However, presently there is no explicit requirement that these two arrays must be of equal length. If wrong lengths are provided, the code will potentially revert ambiguously without an explicit user error. Furthermore, if the bill addresses array is longer, the function will not revert but certain addresses will simply be ignored.
Recommendation	Add a `require` statement to check that the 2 arrays have the same length. `require(_billIds.length == _billAddresses.length, "Unequal parameter lengths");`
Resolution	RESOLVED

Issue #07	Typographical errors
Severity	INFORMATIONAL
Location	`ICustomBill(_billAddresses[i]).redeem(_billIds[i]);`
Description	`_billAddresses` is unnecessarily cast to `ICustomBill` as the `batchRedeem` function could have directly taken the `ICustomBill[]` as a parameter.
Recommendation	Consider rewriting the function to the following signature to avoid the unnecessary cast. `function batchReedem(uint256[] calldata _billIds, ICustomBill[] calldata _bills) external`
Resolution	RESOLVED

Issue #08	Gas optimizations
Severity	INFORMATIONAL
Location	`for (uint i = 0; i < _billIds.length; i++) {`
Description	When looping as done in the snippet above, the length of `_billIds` is read from storage each time. This is highly expensive in gas usage. Consider caching `_billIds.length` to save some gas.

Recommendation	Consider rewriting this section of code as follows:
	`uint256 billsLength = _billId.length;`
	`for (uint i = 0; i < billsLength; i++) {`

Resolution	RESOLVED

2.3 BillNft

The BillNft contract is a classic ERC-721 contract from OpenZeppelin. It is used by the CustomBills (the Bond contracts) as the token that represents a vesting claim. This means that whoever owns a BillNft can at some point claim vested tokens from the related bond.

Bill NFTs can be minted by addresses with the `MINTER_ROLE`. These addresses can be set by admins with the `FACTORY_ROLE` or `DEFAULT_ADMIN_ROLE`.

2.3.1 Privileged Functions

The following functions can be called by the `feeToSetter`:

- `setBaseURI`
- `lockURI`
- `addMinter`
- `mint`
- `mintMany`
- `grantRole`
- `revokeRole`
- `renounceRole`

2.3.2 Issues & Recommendations

Issue #09	**baseURI cannot be set due to uriUnlocked initializing to false in the upgradeable proxy**
Severity	MEDIUM SEVERITY
Location	Line 30 `bool public uriUnlocked = true;`
Description	Within upgradeable proxies, any initial state is set in the implementation, but not in the proxy layer of the contract. This means that all though `uriUnlocked` is set to `true` initially, this will only be the case for the implementation contract and not the actual proxy contract which will be used. Due to this shortcoming, the contract will launch with `uriUnlocked` set to false causing `setBaseURI` to never be callabe. We expect most deployment tools to throw errors when you try to deploy this contract, as they will indicate that this variable value will be lost within the proxy state. We've noticed that the test cases by ApeSwap do not actually deploy this contract as an upgradeable proxy.
Recommendation	Consider adding the following section of code to the initializer: `uriUnlocked = true;` Alternatively, the variable can be renamed to `uriLocked` and start off as `false`. Finally, it might not be necessary to make this NFT an upgradeable proxy, as the code is simple.
Resolution	RESOLVED The `uriUnlocked` parameter has been inverted to `uriLocked` which defaults to `false`.

Issue #10	The contract contains functions that can be made external
Severity	INFORMATIONAL
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword: - `setBaseURI` - `lockURI` - `addMinter` - `mintMany` Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases. If changes to `mint` are applied using an internal function, then `mint` can be made external too.
Recommendation	Consider marking the above functions as `external`. The string parameter in `setBaseURI` can then also be changed to `calldata`.
Resolution	RESOLVED

Issue #11	Unchanged and inadequately overridden function
Severity	INFORMATIONAL

Description	`function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal virtual override(ERC721EnumerableUpgradeable) {`
	`  super._beforeTokenTransfer(from, to, tokenId);`
	`}`
	
	The function `_beforeTokenTransfer` is overridden, which implies that it will be modified, but no such modification is done hence this override can be removed.
	
	The `supportsInterface` function should indicate `true` if either of the two interfaces is queried.
	
	`function supportsInterface(bytes4 interfaceId) public view virtual override(AccessControlEnumerableUpgradeable, ERC721EnumerableUpgradeable) returns (bool) {`
	`  return super.supportsInterface(interfaceId);`
	`}`

Recommendation	Consider removing the `_beforeTokenTransfer` function. Consider returning a union for `supportsInterface`: `return AccessControlEnumerableUpgradeable.supportsInterface(interfaceId) || ERC721EnumerableUpgradeable.supportsInterface(interfaceId);`
Resolution	RESOLVED `_beforeTokenTransfer` has been removed and `supportsInterface` now returns `true` for both interfaces.

Issue #12	**The overridden functions should not be marked `virtual` if the contract is not expected to be inherited**
Severity	INFORMATIONAL
Description	In solidity, the virtual keyword allows an inheriting contract to override its behavior. The function that overrides that base function should be marked as override. As BillNft is not supposed to be inherited, the functions below should not be marked `virtual`. - `_baseURI()` - `mint` - `_beforeTokenTransfer` - `supportsInterface`
Recommendation	Consider removing the `virtual` keyword if the contract is not expected to be inherited.
Resolution	RESOLVED

Issue #13	**AccessControlEnumerableUpgradeable is not initialized**
Severity	INFORMATIONAL
Description	The contract includes AccessControlEnumerableUpgradeable but doesn't initialize it. This is not a severe issue as all dependecies that needs to be initialized are initialized by ERC721EnumerableUpgradeable, but this is considered a bad practice to not initialize every dependency. ERC721EnumerableUpgradeable is furthermore not initialized itself, only ERC721Upgradeable is.
Recommendation	Consider initializing AccessControlEnumerable by adding `__AccessControlEnumerable_init()` inside the constructor. Consider also initializing ERC721EnumerableUpgradeable.
Resolution	RESOLVED

Issue #14	Unused modifier throughout the contracts
Severity	INFORMATIONAL

Location	Line 37~
	`modifier onlyMinter() {`
	`    require(hasRole(MINTER_ROLE, _msgSender()), "BillNft: Only minter role");`
	`    _;`
	`}`

Description	The contract includes an unused modifier. This will unnecessarily increase the contract source code size, and it can also make third-party reviewing more cumbersome.
Recommendation	Consider removing the aforementioned modifier or use it here: `require(hasRole(MINTER_ROLE, _msgSender()), "Must have minter role to mint");`
Resolution	RESOLVED The modifier is now used within the `mint` functions. `DEFAULT_ADMIN_ROLE` can now no longer mint.

Issue #15	Typographical errors
Severity	INFORMATIONAL

Location	Line 17~
	`* The account that deploys the contract will be granted the minter`
	`* role, as well as the default admin role, which will let it grant minter`
	`* roles to other accounts.`

Description	The statement above is outdated, as the roles are now provided as parameters within the initializer.
Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

Issue #16	Lack of event
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. - `setBaseURI` - `lockURI` The initializer could also emit a `BaseURISet` event as it also sets the `BaseURI`.
Recommendation	Consider adding events for the above functions.
Resolution	RESOLVED

Issue #17	Gas optimization: Re-using `mint` uses more gas than necessary
Severity	INFORMATIONAL
Location	Line 97 `mint(to);`
Description	The `mintMany` function presently re-runs all minting logic for every NFT minted — this unecessarily wastes more gas as the `MINTER_ROLE` only needs to be checked once.

Recommendation	Consider checking the requirement only once using an internal function for minting new tokens that you would also use in the `mint` function.
	
	`function mint(address to) external virtual returns (uint256 newTokenId) {`
	`    require(hasRole(MINTER_ROLE, _msgSender()), "Must have minter role to mint");`
	`    _mintInternal(to);`
	`}`
	
	`function mintMany(uint256 amount, address to) external {`
	`    require(hasRole(MINTER_ROLE, _msgSender()), "Must have minter role to mint");`
	`    for(uint i = 0; i < amount; i++){`
	`        _mintInternal(to);`
	`    }`
	`}`
	
	`function _mintInternal(address to) internal {`
	`    newTokenId = _tokenIdTracker.current();`
	`    _tokenIdTracker.increment();`
	`    _mint(to, newTokenId);`
	`}`

Resolution	RESOLVED The recommended internalization has been implemented.

2.4 CustomBillFactory

The CustomBillFactory allows the Policy addresses to create Bill and Treasury contracts using OpenZeppelin Clones, which are non-upgradeable proxy addresses. Policy addresses can set the implementation of Bill and Treasury contracts at any time.

The CustomBillFactory stores all created Bill instance addresses into a separate contract called the FactoryStorage.

2.4.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `createBillAndTreasury`
- `createBill`
- `setBillNft`
- `setBillImplementation`
- `setTreasuryImplementation`
- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.4.2 Issues & Recommendations

Issue #18	`createBill` sets the treasury parameter to `_customTreasury`
Severity	LOW SEVERITY
Description	When creating a Bill using `createBill`, the treasury parameter is wrongly set to `_customTreasury` when it should be `treasury`. This will cause the fees to be sent back to the `customTreasury` and not the actual treasury, and cause inconsistency between `createBill` and `createBillAndTreasury`.
Recommendation	Consider setting the treasury parameter to `treasury`.
Resolution	RESOLVED

Issue #19	Lack of events
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. - `createBillAndTreasury` - `createBill` - `setBillNft` - `setBillImplementation` - `setTreasuryImplementation` The initializer could also emit a `BaseURISet` event as it also sets the `BaseURI`.
Recommendation	Consider adding an event to the above functions.
Resolution	RESOLVED

Issue #20	Unused imports
Severity	INFORMATIONAL

Location	Lines 4-5
	`import "@openzeppelin/contracts/utils/math/SafeMath.sol";`
	`import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";`
	
	Lines 11-12
	`import "./libraries/FullMath.sol";`
	`import "./libraries/FixedPoint.sol";`

Description	Files that are imported in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length and bytecode size unnecessarily.
Recommendation	Consider removing the above imports to keep the contract short and simple.
Resolution	RESOLVED

Issue #21	Typographical errors
Severity	INFORMATIONAL
Description	Line 52~ /** @notice deploys custom treasury and custom bill contracts and returns address of both @param _payoutToken address @param _principleToken address @param _initialOwner address @return _treasury address @return _bill address */ Line 100~ /** @notice deploys custom treasury and custom bill contracts and returns address of both @param _payoutToken address @param _principleToken address @param _customTreasury address @return _treasury address @return _bill address */ These 2 comments lack 4 parameters: `_payoutAddress`, `_tierCeilings`, `_fees`, `_feeInPayout`. Consider adding the parameters to the comments to keep the code consistent. Line 101 @notice deploys custom treasury and custom bill contracts and returns address of both The comment mentions creating a custom treasury while this function only creates a custom bill contract. Consider changing the comment to be accurate with the behavior of the function.
Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

2.5 CustomTreasury

The CustomTreasury contract is a simple treasury contract to store ERC-20 tokens for the DAO.

It contains a `deposit` function callable by the bill contracts (the bonds). `deposit` transfers `_amountPrincipleToken` into this contract and transfers out `_amountPayoutToken` to the bill. `sendPayoutToken` does the same thing without transferring in any `principleToken`.

`withdraw` allows a policy (the governance) to withdraw any amount of any token to an address.

It should be noted that this CustomTreasury actually forwards any ERC-20 tokens deposited by CustomBills `payoutAddress`. It might therefore be the case that these specific contracts do not hold much value. If they do, there is high governance risk within this contract. Especially if the policy wallet and the proxy admin are deployed as an upgradeable proxies.

2.5.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `deposit [ Bill ]`
- `sendPayoutTokens [ Bill ]`
- `withdraw`
- `toggleBillContract`
- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.5.2 Issues & Recommendations

Issue #22	Unused function throughout the contracts
Severity	INFORMATIONAL

Location	Line 66~
	`function sendPayoutTokens(uint _amountPayoutToken) external {`
	`    require(billContract[msg.sender], "msg.sender is not a bill contract");`
	`    IERC20(payoutToken).safeTransfer(msg.sender, _amountPayoutToken);`
	`}`

Description	The function sendPayoutToken is expected to be used by a bill contract, but this function is never used inside CustomBill nor in the entire codebase. Functions like this unnecessarily increase the contract source code size and can also make third-party reviewing more cumbersome.
Recommendation	Consider removing the unused function.
Resolution	RESOLVED

Issue #23	CustomTreasury is a proxy but does not use upgradeable OpenZeppelin dependencies
Severity	INFORMATIONAL

Location	Line 4~
	`import "@openzeppelin/contracts/utils/math/SafeMath.sol";`
	`import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";`
	
	`import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";`

Description	Given that this contract is a proxy, it should use the upgradeable OpenZeppelin dependencies as these have `init` functions.
Recommendation	Consider consistently using upgradeable dependencies with Proxy contracts. In fact, as SafeMath should not be used, it should simply be removed completely in this instance.
Resolution	RESOLVED SafeMath usage has been replaced with normal math that is still overflow-safe due to the version of the contract (0.8.9).

Issue #24	`valueOfToken` can be made external
Severity	INFORMATIONAL
Location	Line 79 `function valueOfToken(address _principleTokenAddress, uint256 _amount) public view returns (uint256 value_)`
Description	Functions that are not used within the contract but only externally can be marked as such with the external keyword. Apart from being a best practice when the function is not used within the contract, this can lead to a lower gas usage in certain cases.
Recommendation	Consider marking the function as external.
Resolution	RESOLVED

Issue #25	Typographical errors and inconsistencies
Severity	INFORMATIONAL

Description	`event BillContractToggled(address billContract, bool approved);`
	`event Withdraw(address token, address destination, uint256 amount);`
	
	`billContract`, `token` and `destination` could be indexed within these events.
	
	Line 34
	`require(_initialOwner != address(0), "CustomTreasury: Payout token cannot address zero");`
	
	The error message should mention initial owner and not payout token. Consider changing the returned error message.
	
	Line 35
	`_policy = _initialOwner`
	
	The initial `policy` (owner) assignment in the initializer lacks a PolicyTransfer event. This makes it inconsistent with the constructor of a non-upgradable Policy deployment.
	
	Line 53
	`require(billContract[msg.sender], "msg.sender is not a bond contract");`
	
	To be consistent with line 67, the error message should mention `bill` and not `bond`.

Line 54~

```
IERC20(_principleTokenAddress).safeTransferFrom(
    msg.sender,
    payoutAddress,
    _amountPrincipleToken
);
```

`_principleTokenAddress` is unnecessarily cast to IERC20 as the `deposit` function could have directly taken the IERC20 as a parameter. Consider rewriting the function to the following signature to avoid the unnecessary cast.

Line 48~

```
function deposit(
    IERC20 _principleTokenAddress,
    uint256 _amountPrincipleToken,
    uint256 _amountPayoutToken
) external
```

For the same reasons, the `_payoutToken` parameter can be directly taken as IERC20Metadata. Consider rewriting the initialize to the following signature to avoid the unnecessary cast.

Line 31

```
function initialize(IERC20Metadata _payoutToken, address
_initialOwner, address _payoutAddress) public initializer
```

Line 94

```
*  @param _token uint
```

The `_token` parameter is in fact an address and not an `uint`. Consider changing the `natspec` comment.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED

2.6 FactoryStorage

FactoryStorage stores the different parameters of bills created by the factory. This bill information is stored outside of the CustomBillFactory to allow the factory implementation to change while keeping the bills previously created by the factory.

2.6.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `setFactoryAddress`
- `pushBill`
- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.6.2 Issues & Recommendations

Issue #26	Lack of getter for `_tierCeilings` and `_fees`
Severity	INFORMATIONAL
Description	The `BillDetails` struct contains two arrays: these arrays cannot be fetched by other contracts or the frontend as solidity drops them from the `BillDetails` function response. If these parameters should be accessed (which we believe should be the case), separate getter functions must be added.
Recommendation	Consider adding a getter function for `_tierCeilings` and `_fees`. This getter could optionally be paginated but we do understand that these arrays will be so short that they likely do not need pagination.
Resolution	RESOLVED The view function `billFees` has been introduced. It can theoretically run out of gas, but due to the fees array being so short, we expect this to never happen.

Issue #27	Fee recipient treasury is not saved within `BillDetails`
Severity	INFORMATIONAL
Description	Only the CustomTreasury address for the bill is saved while the treasury address that receives the fees is not saved as a parameter within `BillDetails`.
Recommendation	Consider also adding and saving a `feeRecipient` (or more ambiguously `treasury`) address to the `BillDetails`.
Resolution	ACKNOWLEDGED

Issue #28	Lack of events
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. - `setFactoryAddress`
Recommendation	Consider adding an event to the above function.
Resolution	RESOLVED

2.7 Policy

The Policy contract is an owner dependency, it allows an user to be the owner of a contract and execute privileged functions for said contract. Here the owner is called the policy of a contract. The contract furthermore differs from a traditional OpenZeppelin Ownable implementation by improving over it with push and pull mechanics. Such mechanics require the new owner to claim ownership before it is actually transferred, preventing accidental transfers to non-existent addresses from occurring.

2.7.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.7.2 Issues & Recommendations

Issue #29	**After `policy` has been renounced, it can be reclaimed by the pending policy through `pullPolicy`**
Severity	HIGH SEVERITY
Description	After policy has been renounced, the address newPolicy_ can still use pullPolicy to regain the privileges that have been renounced. This issue is marked as High Severity because users cannot know if `newPolicy` is assigned to an address. This could allow the previous policy to regain its privileges and use them against users, while they think privileges has been renounced.
Recommendation	Consider implementing the following code to renounce `policy`: `function renouncePolicy() public virtual override onlyPolicy { emit PolicyTransferred(_policy, address(0)); _policy = address(0); _newPolicy = address(0); }` Personally we would go one step further and create an internal function `_transferPolicy(address to)` that sets the new policy to `to` and unsets `_newPolicy`. This avoids writing redundant code and can furthermore be used in the initializers to automatically emit an event.
Resolution	RESOLVED The first recommendation has been introduced. The `_newPolicy` variable is now set to `address(0)`.

Issue #30	`_newPolicy` needs a getter
Severity	INFORMATIONAL
Description	As `_newPolicy` is internal, user are not able to see it.

Recommendation	Consider either adding the following code to the contract or making the value public:
	`function newPolicy() external override returns (address) {`
	`  return _newPolicy;`
	`}`

Resolution	RESOLVED

Issue #31	Lack of events
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. - `pushPolicy`
Recommendation	Consider adding an event to the above function.
Resolution	RESOLVED

2.8 SubsidyRouter

2.8.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `addSubsidyController`
- `removeSubsidyController`
- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.8.2 Issues & Recommendations

Issue #32	Lack of events
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. - `addSubsidyController` - `removeSubsidyController`
Recommendation	Consider adding an event to the above functions.
Resolution	RESOLVED

Issue #33	Typographical errors
Severity	INFORMATIONAL

Description	Line 19
	`function getSubsidyInfo() external returns (uint256)`
	
	The function `GetSubsidyInfo` is not a getter as it calls the `paySubsidy` that changes the storage. Consider renaming it to `paySubsidy`.
	
	Line 32
	`function addSubsidyController(address _bill, address _subsidyController) external onlyPolicy`
	
	The `addSubsidyController` is in fact setting the `subsidyController`, not adding it. Consider renaming the function to `setSubsidyController`.

Recommendation	Consider fixing the typographical errors.
Resolution	RESOLVED The `getSubsidyInfo` function has been renamed to `paySubsidy` and `addSubsidyController` to `setSubsidyController`.

Issue #34	Gas Optimization: Cache values to save gas
Severity	INFORMATIONAL

Location	Line 19~
	`function getSubsidyInfo() external returns (uint256) {`
	`require(`
	`billForController[msg.sender] != address(0),`
	`"Address not mapped"`
	`);`
	`return`
	`IBill(billForController[msg.sender]).paySubsidy();`
	`}`

Description	The code contains sections of code that re-use storage variables. Accessing the same storage multiple amounts of times unecessarily wastes gas.
Recommendation	Consider caching `billForController[msg.sender]` to save some gas.
Resolution	RESOLVED

2.9 CustomBill

The CustomBill is a contract inspired and based on the Ohm BondDepository contract.

It allows users to buy the `payoutToken` by providing some `principalToken`. However, contrary to a normal purchase, `payoutToken` is then vested linearly over a configurable vesting period (eg. 5 days). The price at which users can "buy" these `payoutTokens` can be configured and automatically adjusts with two separate mechanisms.

Firstly, there's the `totalDebt` mechanism. Whenever a bond is bought, the `totalDebt` increases which causes the `debtRatio` (`totalDebt/ payoutTokens.totalSupply()`) to increase. This directly causes the price to increase. In other words, whenever bonds are purchased, the price increases. `totalDebt` will then slowly decrease automatically in a non-linear fashion (eg. a percentage of `totalDebt` decays every so often).

Secondly, there's an adjustment mechanism. The governance can independently configure an adjustment `direction`, `rate`, `buffer` time and `target`. Every time the buffer time passes, the price will be adjusted in the configured `direction` at a rate of `rate`, until `target` is reached. This adjustment is done indirectly through the "Control Variable", which is the proxy variable controllable by the governance to influence the price.

The governance should be extremely careful with managing the `control variable` —a wrongly set control variable could cause an undesirable situation where `payoutToken` is given at a significant discount.

The DAO address can change the treasury which receives the bond fees.

The contract contains logic referred to as the "payout subsidy". This is an accounting variable which accumulates the total payout given over time. Whenever `paySubsidy` is called by a subsidy controller, it resets. We've confirmed

with the client that they do not plan to use this mechanism and therefore the controller itself was not included within the scope of this audit. For this specific audit, this variable therefore purely has accounting purposes similar to `totalPayoutGiven`.

The governance can set a minimum price, however, as soon as the actual price reaches above this, the minimum is removed. This minimum is therefore only useful for avoiding mistakes during initial configuration.

2.9.1 Privileged Functions

The following functions can be called by the owner of the contract:

- `initializeBill`
- `setBillTerms`
- `setAdjustment`
- `changeTreasury`
- `paySubsidy`
- `renouncePolicy`
- `pushPolicy`
- `pullPolicy`

2.9.2 Issues & Recommendations

Issue #35	DoS: A malicious party can block a specific address from redeeming for a certain period of time
Severity	MEDIUM SEVERITY
Description	When a user purchases a bond, they need to wait a period of time until the `payoutTokens` are vested. Over this period, they can already start linearly claiming their `payoutTokens` with the redeem function. However, presently, anyone can redeem tokens for anyone. As these tokens are still sent to the rightful owner, this does not initially seem like a large issue. However, when we dig further into the options one has to potentially annoy other users using this redemption, we find a method which allows an exploiter to prevent another user from redeeming throughout their vest period. We demonstrate this attack by example: If the vesting period is 5 days according to the comments, an exploiter would need to call `redeem` every 43.2 (`5 * 86_400 / 10_000 = 43.2`) seconds on their target to lock their vest in until nearly the end of the vesting period. This is possible because the `percentVestedFor` function significantly rounds down and at intervals under 43.2 seconds, it rounds to zero. When a malicious party then calls `redeem`, the redemption amount is also zero while the bond is re-locked.
Recommendation	Consider not allowing anyone other than the `msg.sender` to call `redeem`. If the intention was to allow another contract to call `redeem` for users (eg. the utility BatchRedeem contract), consider using a whitelist or approval mechanism.
Resolution	RESOLVED The contract now contains an approval mechanism where users can enable a user to redeem for them through the `toggleRedeemer` function.

Issue #36	The UI function `getBillIdsInRange` does not function
Severity	MEDIUM SEVERITY

Location	Line 584~
	`function getBillIdsInRange(address user, uint start, uint end)`
	`public`
	`view`
	`returns (uint[] memory)`
	`{`
	`uint[] memory result = new uint[](end - start);`
	`for (uint i = start; i < end; i++) {`
	`uint tokenId = billNft.tokenOfOwnerByIndex(user, i);`
	`if (billIssuedIds.contains(tokenId))`
	`result[i] = tokenId;`
	`}`
	`return result;`
	`}`

Description	The `getBillIdsInRange` function presently does not work. The specific shortcoming is that it tries to set `result[i]` with `i` potentially starting at `start` instead of zero. Secondly, the `tokenId` zero exists, which means there is a semantical overlap between the connotation of 0. This can either be the 0 `tokenId` or the fact that the `billIssuedIds` did not contain this specific `tokenId`. There is no way for the frontend to distinguish these two different connotations.
Recommendation	Consider fixing the index as follows: `result[i - start] = tokenid;` For the zero overlap, we recommend simply starting the NFT ids at id 1 instead of zero. This completely avoids the overlap issue and therefore does not require any further changes within CustomBill.
Resolution	RESOLVED The NFT token now starts from id zero and the recommended fix has been introduced to index at `i - start`.

Issue #37	Phishing risk: Deposits can be sent to another address
Severity	LOW SEVERITY

Location	Line 267~
	`function deposit(`
	`  uint256 _amount,`
	`  uint256 _maxPrice,`
	`  address _depositor`
	`) external returns (uint256)`

Description	The deposit function allows a user to set a `_depositor` parameter that references to the recipient of the Bill. If the frontend is ever hacked, this `_depositor` parameter could be overridden to the hacker, without most people noticing it.
Recommendation	Consider keeping a whitelist of `msg.sender` which can set a `_depositor` that is different to themselves. Alternatively one can only allow this flow if `Address.isContract(msg.sender)` as a contract cannot be phished. This last recommendation might be even more desirable. `require(msg.sender == _depositor || Address.isContract(msg.sender), "Users cannot deposit to another address");`
Resolution	RESOLVED The recommended fix has been introduced.

Issue #38	CustomBill is a proxy but does not use upgradeable OpenZeppelin dependencies
Severity	LOW SEVERITY
Description	The CustomBill is a non-upgradeable proxy contract as it is deployed using the Clones library. This allows for the CustomBill code to be stored in a separate implementation contract from the CustomBillFactory and also reduces the deployment cost of CustomBills. However, given that this contract is a proxy, it should still use the upgradeable OpenZeppelin dependencies as these have `init` functions.
Recommendation	Consider consistently using upgradeable dependencies with Clones contracts.
Resolution	RESOLVED

Issue #39	Adjustment `target` is never reached
Severity	LOW SEVERITY

Location	Line 391~
	`uint timestampCanAdjust = adjustment.lastBlockTimestamp.add(adjustment.buffer);`
	`if(adjustment.rate != 0 && block.timestamp >= timestampCanAdjust) {`
	`uint initial = terms.controlVariable;`
	`if (adjustment.add) {`
	`terms.controlVariable = terms.controlVariable.add( adjustment.rate );`
	`if (terms.controlVariable >= adjustment.target) {`
	`adjustment.rate = 0;`
	`}`
	`} else {`
	`terms.controlVariable = terms.controlVariable.sub(adjustment.rate);`
	`if (terms.controlVariable <= adjustment.target) {`
	`adjustment.rate = 0;`
	`}`
	`}`
	`adjustment.lastBlockTimestamp = block.timestamp;`
	`emit ControlVariableAdjustment(initial, terms.controlVariable, adjustment.rate, adjustment.add);`
	`}`

Description	The code contains an `adjust` function which allows the adjustment of the control variable with a fixed increment or decrement after a fixed period. It also contains a `target` after which the adjustment stops once it is reached. However, due to the code implementation, the `target` might be slightly missed, as the adjustment will only stop after it is passed due to the increments being rather large. Additionally, if the target would be set close to zero, the subtraction might cause this to revert.

Recommendation Consider setting the `terms.controlVariable` to the target once the target has been reached.

It should be noted that this adjustment method is also slightly wasteful in gas as it often re-reads `terms.controlVariable` from storage. If gas-usage is a concern, consider caching some of these variables.

A possible implementation of this recommendation is:

```
function adjust() internal {
      uint timestampCanAdjust =
adjustment.lastBlockTimestamp.add(adjustment.buffer);
      if(adjustment.rate != 0 && block.timestamp >=
timestampCanAdjust) {
          uint initial = terms.controlVariable;
          uint bcv = terms.controlVariable;
          if ( adjustment.add ) {
              bcv = bcv.add( adjustment.rate );
              if ( bcv >= adjustment.target ) {
                  bcv = adjustment.target;
                  adjustment.rate = 0;
              }
          } else {
              bcv = bcv > adjustment.rate ? bcv
-adjustment.rate : 0;
              if ( bcv <= adjustment.target ) {
                  bcv = adjustment.target;
                  adjustment.rate = 0;
              }
          }
          adjustment.lastBlockTimestamp = block.timestamp;
          terms.controlVariable = bcv;
          emit ControlVariableAdjustment( initial, bcv,
adjustment.rate, adjustment.add );
      }
  }
}
```

Resolution RESOLVED

The recommended fix has been introduced.

Issue #40	Lack of mint safeguards
Severity	LOW SEVERITY
Description	One of the main risks within Ohm bonds in general is mis-configurations. Arguably the code is not well-structured to prevent configurational errors and having either good on-chain safeguards and/or off-chain tools that calculate the correct parameterizations is an absolute must. Another safeguard is to always have a staging environment (copy of the production environment) on the mainnet where any bond configurations are first deployed to. As a last resort, we do recommend having a hard payout cap per bond to avoid severe over-payout in the worst case scenario. The treasury could also be provided a more limited number of `payoutTokens` to further safeguard this.
Recommendation	Consider adding a requirement that the total payout given never exceeds an increasable maximum. This requirement should only be done after `totalPayoutGiven` is incremented within `deposit`. `require(totalPayoutGiven <= maxTotalPayout, "Max total payout exceeded");`
Resolution	RESOLVED The recommended fix has been introduced.

Issue #41	Private variables
Severity	LOW SEVERITY
Description	Important variables that third-parties might want to inspect should be marked as public so that these third-parties can easily inspect them through the explorer, web3 and derivative contracts. - `feeTiers` - `feeInPayout`
Recommendation	Consider making the variables public.
Resolution	RESOLVED

Severity	INFORMATIONAL
Description	`trueBillPrice` is slightly incorrect. This function returns the price if it were to account for the fee which is sent to the governance. We demonstrate the logical flaw in the `trueBillPrice` calculation with a simple example: Let's say there's 5% VAT (in a country where this is calculated on the gross price). When you purchase a beer of $1 including VAT, the beer price excluding VAT would be $0.95. Using the `trueBillPrice` calculation, the real beer price would be $0.95 * (100% + 5%) which is $0.9975.
Recommendation	Consider whether this small difference is acceptable. If not, consider updating the arithmetic.
Resolution	RESOLVED

Issue #43	**Unused imports**
Severity	INFORMATIONAL
Location	Line 10 `import "./libraries/FullMath.sol";`
Description	Files that are imported in a contract but not used within said contract could confuse third-party auditors. They also increase the contract length and potentially bytecode size unnecessarily.
Recommendation	Consider removing the aforementioned import to keep the contract short and simple.
Resolution	RESOLVED

Issue #44	Contract does not work with a zero `vestingTerm`
Severity	INFORMATIONAL
Description	The `debtDecay` function reverts due to a division by zero if `terms.vestingTerm` is set to zero. Additionally, the `percentVestedFor` function will always return a zero vested percentage if the remaining vesting duration is zero (eg. with a zero vestingTerm). This should more accurately return 10,000 (100%) as at this point the bonds instantly vests. The contract would therefore become unusable if the `vestingTerm` is zero.
Recommendation	Consider making the requirement of a non-zero vesting term explicit when the term is set.
Resolution	RESOLVED A non-zero case has been introduced, however, the contract does not revert in this scenario. Instead, it instantly decays the whole debt. We recommend the client to be careful in this scenario and keep the mint limit safeguard tight.

Issue #45	Lack of validation
Severity	INFORMATIONAL
Description	The `initializeBill` function has no validation of the parameters that are used for initialization of the bill terms when the contract is first deployed. We are also unsure why there should be an `initialDebt` on this function. Additionally, the `initialize` function lacks validation on certain parameters: - The fee ceilings should be checked to be in the ascending order - The fees assigned in the `FeeTiers` must be smaller or equal to 1e6 (1 million). If not, the fee logic within the contract will cause an underflow revert when is used on `trueBillPrice` or `payoutFor`. ❗ The `setAdjustment` function on the other hand becomes completely locked out if `controlVariable` ever reaches zero, which is strange behavior to have defined so implicitly.
Recommendation	Consider adding proper validation for the above functions and remove the `initialDebt` parameter if there is no need for an initial debt. Check `setBillTerms` for the desired initialization checks.
Resolution	RESOLVED

Issue #46	Typographical error
Severity	INFORMATIONAL
Location	Line 127 `require(address(_config[5]) != address(0), "billNft cannot be zero");`
Description	The array `_config` is already set to `address`, casting twice is unnecessary.
Recommendation	Consider removing the unnecessary cast.
Resolution	RESOLVED

Issue #47	Gas optimization: Certain variables can be cached
Severity	INFORMATIONAL
Location	Line 131 `for (uint256 i; i < _tierCeilings.length; i++)`
Description	The `_tierCeilings.length` can be cached within a separate variable. Within the current implementation, this length variable is fetched from storage on each iteration, wasting a lot of gas.
Recommendation	Consider caching `_tierCeilings.length` to save some gas.
Resolution	RESOLVED

Issue #48	Usage of wildcards
Severity	INFORMATIONAL
Location	Line 16 `using FixedPoint for *;`
Description	Usage of wildcards is considered bad practice as the library is made only for `uint256`. Using a wildcard indicates that you can use the functions of that library with any types.
Recommendation	Consider using FixedPoint only for `uint256`.
Resolution	RESOLVED

Issue #49	Indexing of events
Severity	INFORMATIONAL
Description	It is considered best practice to add indexing for events. This allows to filter those events on indexed parameters making the process of finding some specific event more convenient.

Recommendation	Consider using the following implementations of events:
	`event TreasuryChanged(address indexed newTreasury);`
	`event BillCreated(uint256 deposit, uint256 payout, uint256 expires, uint256 indexed billId);`
	`event BillRedeemed(uint256 indexed billId, address indexed recipient, uint256 payout, uint256 remaining);`
	`event BillPriceChanged(uint256 internalPrice, uint256 indexed debtRatio);`
	`event ControlVariableAdjustment( uint256 initialBCV, uint256 newBCV, uint256 adjustment, bool addition );`
	`event SetAdjustment( bool addition, uint256 increment, uint256 target, uint256 buffer );`

Resolution	RESOLVED

Issue #50	Lack of events
Severity	INFORMATIONAL
Description	Functions that affect the status of sensitive variables should emit events as notifications. - `initializeBill` - `setBillTerms` - `paySubsidy`
Recommendation	Consider adding an event to the above functions.
Resolution	RESOLVED

Issue #51	Initializing terms prematurely adds unnecessary state
Severity	INFORMATIONAL

Location	Line 139~
	`terms = Terms ({`
	`  controlVariable: 1,`
	`  vestingTerm: 1,`
	`  minimumPrice: 1,`
	`  maxPayout: 1,`
	`  maxDebt: 1`
	`});`

Description	The initializer sets all term variables to 1 — this is absolutely unecessary especially because a low `controlVariable` makes the bond extremely cheap.
Recommendation	Consider setting `maxPayout` and `maxDebt` to zero to further indicate to third-parties that this term will not actually be usable.
Resolution	RESOLVED The premature initialization has been removed.

Issue #52	`BillPriceChanged` event is emitted prematurely
Severity	INFORMATIONAL

Location	Line 331~
	`emit BillPriceChanged(_billPrice(), debtRatio());`
	
	`adjust();`

Description	The `BillPriceChanged` event is emitted before the final price adjustment initialized by `adjust()` is incorporated. This causes the `BillPriceChanged` event to be emitted with a bill price which is actually likely not the price after the deposit happened.
Recommendation	Consider emitting the event after `adjust()` if desired.
Resolution	RESOLVED The two lines of code have been inverted.

Issue #53	`batchRedeem` does not return the total payout
Severity	INFORMATIONAL
Description	`redeem` returns the payout while `batchRedeem` does not return the sum of the payout of each nft.
Recommendation	This is inconsistent: `batchRedeem` should return the sum of each nfts payout.
Resolution	RESOLVED `batchRedeem` now returns the sum of payouts.

Issue #54	**The usage of FixedPoint throughout the contract does not contribute to the integrity of the contract and can therefore be removed**
Severity	INFORMATIONAL
Description	The usage of FixedPoint does not contribute to the integrity or safety of this codebase, arguably it even makes the codebase more difficult to comprehend and more prone to issues.
Recommendation	Consider simply using multiply before divide patterns throughout the codebase.
Resolution	RESOLVED

Issue #55	**`decayDebt` is not path independent**
Severity	INFORMATIONAL
Description	Decaying debt every second for 1 day long does not bring the same outcome as decaying it once after 24 hours.
Recommendation	Consider whether this is an issue. If so, `decayDebt` can either decay a fixed amount of debt linear to time and not the current debt value, alternatively a minimum waiting period between individual decays could be considered.
Resolution	RESOLVED


PALADIN
BLOCKCHAIN SECURITY