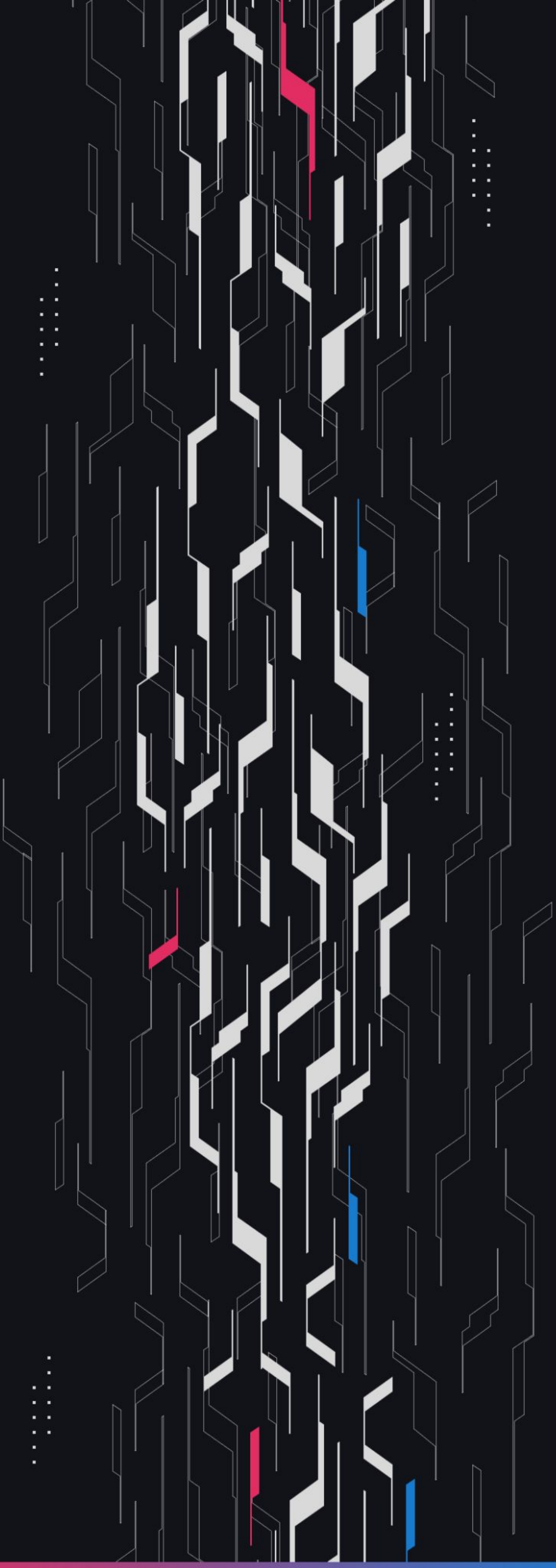
GUARDIAN

Buttonwood

Buttonwood v1

Security Assessment

October 4th, 2025



Summary

Audit Firm Guardian

Prepared By Owen Thurm, Osman Ozdemir, Mark Jonathas, 0xDjango

Client Firm Buttonwood

Final Report Date October 4, 2025

Audit Summary

Buttonwood engaged Guardian to review the security of their v1 Protocol. From the 21st of July to the 7th of August, a team of 4 auditors reviewed the source code in scope. All findings have been recorded in the following report.

Confidence Ranking

Given the number of non-critical issues detected and code changes following the main review, Guardian assigns a Confidence Ranking of 3 to the protocol. Guardian advises the protocol to address issues thoroughly and consider a targeted follow-up audit depending on code changes. For detailed understanding of the Guardian Confidence Ranking, please see the rubric on the following page.

🔗 Blockchain network: **HyperEVM**

✅ Verify the authenticity of this report on Guardian's GitHub: https://github.com/guardianaudits

📊 Code coverage & PoC test suite: https://github.com/GuardianOrg/cash-buttonwood-cash-fuzz

Guardian Confidence Ranking

Confidence Ranking	Definition and Recommendation	Risk Profile
5: Very High Confidence	Codebase is mature, clean, and secure. No High or Critical vulnerabilities were found. Follows modern best practices with high test coverage and thoughtful design. **Recommendation:** Code is highly secure at time of audit. Low risk of latent critical issues.	0 High/Critical findings and few Low/Medium severity findings.
4: High Confidence	Code is clean, well-structured, and adheres to best practices. Only Low or Medium-severity issues were discovered. Design patterns are sound, and test coverage is reasonable. Small changes, such as modifying rounding logic, may introduce new vulnerabilities and should be carefully reviewed. **Recommendation:** Suitable for deployment after remediations; consider periodic review with changes.	0 High/Critical findings. Varied Low/Medium severity findings.
3: Moderate Confidence	Medium-severity and occasional High-severity issues found. Code is functional, but there are concerning areas (e.g., weak modularity, risky patterns). No critical design flaws, though some patterns could lead to issues in edge cases. **Recommendation:** Address issues thoroughly and consider a targeted follow-up audit depending on code changes.	1 High finding and ≥ 3 Medium. Varied Low severity findings.
2: Low Confidence	Code shows frequent emergence of Critical/High vulnerabilities (~2/week). Audit revealed recurring anti-patterns, weak test coverage, or unclear logic. These characteristics suggest a high likelihood of latent issues. **Recommendation:** Post-audit development and a second audit cycle are strongly advised.	2-4 High/Critical findings per engagement week.
1: Very Low Confidence	Code has systemic issues. Multiple High/Critical findings (≥5/week), poor security posture, and design flaws that introduce compounding risks. Safety cannot be assured. **Recommendation:** Halt deployment and seek a comprehensive re-audit after substantial refactoring.	≥5 High/Critical findings and overall systemic flaws.

Table of Contents

Project Overview

Project Summary

Project Name	Buttonwood
Language	Solidity
Codebase	https://github.com/GuardianOrg/cash-buttonwood-cash-team1
Commit(s)	Initial commit: 083bc61f665cd70e71f4e5bae384b88c693e1744 Final commit: f45282bb781f3dd620fe788c9216a4efcccbb414

Audit Summary

Delivery Date	October 4, 2025
Audit Methodology	Static Analysis, Manual Review, Test Suite, Contract Fuzzing

Vulnerability Summary

Vulnerability Level	Total	Pending	Declined	Acknowledged	Partially Resolved	Resolved
● Critical	3	0	0	0	0	3
● High	8	0	0	0	0	8
● Medium	20	0	0	10	0	10
● Low	66	0	0	22	0	44
● Info	2	0	0	0	0	2

Audit Scope & Methodology

Scope and details:

```
contract,source,total,comment
./src/UsdxQueue.sol,36,75,25
./src/USDX.sol,56,115,45
./src/SubConsol.sol,65,153,67
./src/RebasingERC20.sol,56,107,42
./src/PythPriceOracle.sol,43,108,55
./src/PythInterestRateOracle.sol,48,131,71
./src/OriginationPoolScheduler.sol,228,489,195
./src/OriginationPool.sol,121,231,79
./src/OrderPool.sol,108,203,73
./src/MultiTokenVault.sol,121,253,92
./src/MortgageQueue.sol,111,216,76
./src/MortgageNFT.sol,60,133,58
./src/LoanManager.sol,216,434,149
./src/LenderQueue.sol,81,172,70
./src/GeneralManager.sol,383,745,273
./src/ForfeitedAssetsQueue.sol,37,77,26
./src/ForfeitedAssetsPool.sol,74,157,58
./src/ConversionQueue.sol,155,323,125
./src/Consol.sol,62,124,47
./src/libraries/SharesMath.sol,28,61,29
./src/libraries/Roles.sol,4,11,6
./src/libraries/MortgageMath.sol,276,543,232
./src/libraries/Constants.sol,12,55,42
./src/types/WithdrawalRequest.sol,8,20,11
./src/types/TokenScalars.sol,5,12,6
./src/types/OriginationPoolConfig.sol,14,30,13
./src/types/OPoolConfigId.sol,8,22,11
./src/types/MortgagePosition.sol,25,52,27
./src/types/MortgageNode.sol,8,20,11
./src/types/enums/OriginationPoolPhase.sol,6,14,7
./src/types/enums/MortgageStatus.sol,6,14,7
./src/types/orders/PurchaseOrder.sol,13,29,14
./src/types/orders/OriginationParameters.sol,11,24,11
./src/types/orders/OrderRequests.sol,20,46,23
./src/types/orders/OrderAmounts.sol,6,14,7
./src/types/orders/MortgageParams.sol,13,28,14
source count: {total: 5241, source: 2524, comment: 2097, single: 545, block: 1552, mixed: 3, empty: 623,
todo: 0, blockEmpty: 0, commentToSourceRatio: 0.830824088748019}
```

Audit Scope & Methodology

Vulnerability Classifications

Severity	Impact: *High*	Impact: *Medium*	Impact: *Low*
Likelihood: *High*	● Critical	● High	● Medium
Likelihood: *Medium*	● High	● Medium	● Low
Likelihood: *Low*	● Medium	● Low	● Low

Impact

High	Significant loss of assets in the protocol, significant harm to a group of users, or a core functionality of the protocol is disrupted.
Medium	A small amount of funds can be lost or ancillary functionality of the protocol is affected. The user or protocol may experience reduced or delayed receipt of intended funds.
Low	Can lead to any unexpected behavior with some of the protocol's functionalities that is notable but does not meet the criteria for a higher severity.

Likelihood

High	The attack is possible with reasonable assumptions that mimic on-chain conditions, and the cost of the attack is relatively low compared to the amount gained or the disruption to the protocol.
Medium	An attack vector that is only possible in uncommon cases or requires a large amount of capital to exercise relative to the amount gained or the disruption to the protocol.
Low	Unlikely to ever occur in production.

Audit Scope & Methodology

Methodology

Guardian is the ultimate standard for Smart Contract security. An engagement with Guardian entails the following:

- Two competing teams of Guardian security researchers performing an independent review.
- A dedicated fuzzing engineer to construct a comprehensive stateful fuzzing suite for the project.
- An engagement lead security researcher coordinating the 2 teams, performing their own analysis, relaying findings to the client, and orchestrating the testing/verification efforts.

The auditing process pays special attention to the following considerations:

- Testing the smart contracts against both common and uncommon attack vectors.
- Assessing the codebase to ensure compliance with current best practices and industry standards.
- Ensuring contract logic meets the specifications and intentions of the client.
- Cross-referencing contract structure and implementation against similar smart contracts produced by industry leaders.
- Thorough line-by-line manual review of the entire codebase by industry experts. Comprehensive written tests as a part of a code coverage testing suite.
- Contract fuzzing for increased attack resilience.

Findings & Resolutions

ID	Title	Category	Severity	Status
C-01	Expired Orders Do Not Refund Tokens	Logical Error	● Critical	Resolved
C-02	Missing Access Control In Callback	Access Control	● Critical	Resolved
C-03	All USDX Withdrawals Are DoS'ed	DoS	● Critical	Resolved
H-01	Compounding Orders Cannot Be Processed	Logical Error	● High	Resolved
H-02	Mortgage Queue DoS Due To Unused Hint	DoS	● High	Resolved
M-01	MultiTokenVault Inflation Attack	Gaming	● Medium	Acknowledged
M-02	enqueueMortgage Cannot Accept Gas Fees	Unexpected Behavior	● Medium	Resolved
M-03	Incorrect Async Usage	Logical Error	● Medium	Resolved
M-04	ExpandBalanceSheet Allows Lower Rate	Gaming	● Medium	Resolved
M-05	Expand Balance Sheet Has No Restrictions	Validation	● Medium	Resolved
M-06	Large Penalty Payments Arbitraged	Gaming	● Medium	Acknowledged
M-07	Conversion Queue Perturbed	Validation	● Medium	Resolved
M-08	Borrowers Avoid Interest Through Console	Gaming	● Medium	Acknowledged

Findings & Resolutions

ID	Title	Category	Severity	Status
M-09	Consol Price Jumps After Queue Actions	Frontrunning	● Medium	Acknowledged
M-10	Re-enqueued Mortgages Trap Gas Fees	Logical Error	● Medium	Resolved
M-11	Incorrect MortgageId Update Logic	Logical Error	● Medium	Resolved
M-12	Forclosures Used To Avoid Penalty Payments	Gaming	● Medium	Acknowledged
M-13	First Epoch Of All OriginationPools Can Be DoSed	DoS	● Medium	Resolved
M-14	Collateral Value Not Considered For Foreclosures	Logical Error	● Medium	Acknowledged
M-15	USDX Allows Slippage Free Swaps	MEV	● Medium	Acknowledged
L-01	ForfeitedAssetPool Value Changes Drastically	Logical Error	● Low	Acknowledged
L-02	Some Lenders Don't Receive Yield	Unexpected Behavior	● Low	Acknowledged
L-03	Forfeited Assets Queue Zero Value DoS	DoS	● Low	Resolved
L-04	Cancelled Mortgages Retain The MortgageId	Logical Error	● Low	Resolved
L-05	Withdrawal Queue Traps Yield	Logical Error	● Low	Acknowledged
L-06	Interest Rate Used From Order Creation	Validation	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-07	Forfeited Asset Pool Value Inflation	Logical Error	● Low	Acknowledged
L-08	Multiple OPool Origination Discrepancy	Unexpected Behavior	● Low	Resolved
L-09	Collateral Caps Not Validated On Execution	Validation	● Low	Resolved
L-10	Expansion Requests Can Unexpectedly Fail	Unexpected Behavior	● Low	Resolved
L-11	Blacklisted Addresses May Halt Queues	DoS	● Low	Acknowledged
L-12	Zero Payments Allowed	Validation	● Low	Resolved
L-13	Refinance Allowed For Completed Mortgages	Validation	● Low	Resolved
L-14	Conversion Queue May Get Stuck	Warning	● Low	Resolved
L-15	Converted Positions Can Avoid Penalties	Unexpected Behavior	● Low	Acknowledged
L-16	Creation Request DoS	DoS	● Low	Resolved
L-17	Yield Must Be Withdrawable From The Strategy	Warning	● Low	Acknowledged
L-18	Asset Removal Issues	Warning	● Low	Acknowledged
L-19	Missing SubConsol Validation	Validation	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-20	Lacking Configs Validation	Validation	● Low	Resolved
L-21	Zero Amount Actions Allowed	Validation	● Low	Resolved
L-22	Lacking Refund Mechanism	Unexpected Behavior	● Low	Resolved
L-23	Lacking SafeCast Usage	Unexpected Behavior	● Low	Resolved
L-24	Convert Rounds In Favor Of The User	Rounding	● Low	Resolved
L-25	Lacking Expiration Validation	Validation	● Low	Resolved
L-26	Ineffective Mortgage Request Fills	Unexpected Behavior	● Low	Resolved
L-27	Actions Can Only Occur During Market Hours	Warning	● Low	Resolved
L-28	Term And Penalty Payments Can Be Grieved	DoS	● Low	Resolved
L-29	Check Balance In removeAsset	Unexpected Behavior	● Low	Resolved
L-30	USDX Withdrawals Can Be DoSed	DoS	● Low	Resolved
L-31	ForfeitedAssetsQueue Can Be DoSed	DoS	● Low	Acknowledged
L-32	Incorrect Natspec Comment	Informational	● Low	Resolved

Findings & Resolutions

ID	Title	Category	Severity	Status
L-33	Inactive Mortgages In ConversionQueue	Unexpected Behavior	● Low	Resolved
L-34	Old forfeitedAssetsPool Remains Supported	Logical Error	● Low	Resolved
L-35	Duplicate Deletion Of Order	Error	● Low	Resolved
L-36	Early Closures Still Pay Full Interest	Informational	● Low	Acknowledged
L-37	Race Condition For ForfeitedAssetsPool Tokens	Logical Error	● Low	Acknowledged
L-38	Penalty Calculation Rounds In Favor Of Borrower	Rounding	● Low	Resolved
L-39	Initial oPoolAdmin Does Not Have Admin Role	Configuration	● Low	Resolved
L-40	Penalty Rate Updates Affect Existing Mortgages	Best Practices	● Low	Acknowledged
L-41	Theft Of Funds From USDX With scalarDenominator	Rounding	● Low	Resolved
L-42	CEI Pattern Not Followed	Best Practices	● Low	Resolved
I-01	calculateNewAvergageInterestRate Naming	Typo	● Info	Resolved
I-02	Typos	Informational	● Info	Resolved

C-01 | Expired Orders Do Not Refund Tokens

Category	Severity	Location	Status
Logical Error	● Critical	OrderPool.sol: 163-176	Resolved

Description PoC

When a user requests a mortgage, either the collateral token or USDX is transferred from the user to the order pool, depending on if the request is compounding or not.

If the order expires before it is processed, the order is simply cancelled and the user's tokens are not returned. The order is deleted.

```
function _processOrder(uint256 index, uint256 hintPrevId) internal returns (uint256
collectedGasFee) {
// Fetch the order from the orders mapping
PurchaseOrder memory order = _orders[index];
// If the order has expired, cancel it. Otherwise, process it.
if (order.expiration < block.timestamp) {
// Cancel the mortgage request
IGeneralManager(generalManager).burnMortgageNFT(order.mortgageParams.tokenId);
// Delete the order
delete _orders[index];
// Emit the PurchaseOrderExpired event
emit PurchaseOrderExpired(index);
```

Recommendation

Return the user's collateral or USDX depending on the type of order.

Resolution

Buttonwood Team: The issue was resolved in PR#14.

C-02 | Missing Access Control In Callback

Category	Severity	Location	Status
Access Control	● Critical	GeneralManager.sol: 689	Resolved

Description PoC

The originationPoolDeployCallback function lacks access control. A malicious actor can invoke it directly with a collateralAmount of 0 and an excessively large amountBorrowed.

This results in the minting of amountBorrowed Consol tokens to the malicious user.

Recommendation

Implement access control for the originationPoolDeployCallback function to prevent unauthorized usage.

Resolution

Buttonwood Team: Resolved.

C-03 | All USDX Withdrawals Are DoS'ed

Category	Severity	Location	Status
DoS	● Critical	UsdxQueue.sol: 44C1-50C8	Resolved

Description PoC

Cancelled withdrawal requests are not removed from the queue; instead, the request.amount and request.share values are set to 0 to allow these requests to be skipped.

However, the UsdxQueue.processWithdrawalRequests function does not skip these cancelled requests and attempts to withdraw zero amounts from the Consol, which reverts with an AmountTooSmall error.

Since withdrawal processing is linear, subsequent requests cannot be processed until the previous ones are completed. As a result, a single cancellation causes all subsequent withdrawal requests to be effectively DoSed.

Note that the exact same issue occurs in ForfeitedAssetsQueue.processWithdrawalRequests function as well.

Recommendation

Skip cancelled requests. Do not withdraw from Consol if request.amount is zero.

Resolution

Buttonwood Team: The issue was resolved in PR#15.

H-01 | Compounding Orders Cannot Be Processed

Category	Severity	Location	Status
Logical Error	● High	GeneralManager.sol: 689	Resolved

Description PoC

Users create mortgage requests in the GeneralManager contract, and these requests are processed by the fulfiller using the processOrders function in the OrderPool.

During this process, the GeneralManager.originate function is called, which subsequently calls OriginationPool.deploy, and finally invokes the originationPoolDeployCallback function in GeneralManager.

The _enqueueMortgage function is called within this callback when originationParameters.conversionQueue is non-zero.

However, the _enqueueMortgage function requires the mortgageGasFee to be sent as msg.value, but neither the originate nor the originationPoolDeployCallback functions are payable, making it impossible to send this fee as msg.value.

As a result, processOrders will always fail when a mortgage request has a conversionQueue with a non-zero mortgageGasFee.

Since compounding orders must include a conversionQueue, this issue will have the greatest impact on them. However, it is not limited to compounding orders, as non-compounding orders can also have a conversionQueue set.

Recommendation

Ensure that the required mortgageGasFee is passed as msg.value during the processOrders flow. However, since the fulfiller will invoke this function, the required amount should be transferred from the user to the fulfiller at the time of order creation.

Another important consideration is that the fulfiller can process multiple orders at once and directly using msg.value within the processOrders loop may introduce new issues. Care should be taken when sending value, and only the required amount for each order should be sent.

Resolution

Buttonwood Team: The issue was resolved in PR#13.

H-02 | Mortgage Queue DoS Due To Unused Hint

Category	Severity	Location	Status
DoS	● High	MortgageQueue.sol	Resolved

Description

In the _insertMortgage function of the MortgageQueue contract the hintPrevId value is only validated and does not functionally serve to reduce the iterations necessary to lodge a mortgage in the right spot in the queue.

When the triggerPrice of the head is less than the triggerPrice of the newly inserted mortgage the hintPrevId always begins from the head. Therefore the insertion must always traverse the full set of nodes to insert a new mortgage.

In many cases the queue will either see an amount of natural usage or an intentional malicious amount of usage such that subsequent mortgages require more iterations than the block gas limit to insert.

This would prevent the origination process for compounding mortgages from being fulfilled as well as other processes in the Buttonwood protocol that rely on this underlying queue logic.

Recommendation

Do not reset the hintPrevId to the head node if the hintPrevId has already proven to have a trigger price less than or equal to the triggerPrice of the new mortgage.

Resolution

Buttonwood Team: The issue was resolved in PR#32.

M-01 | MultiTokenVault Inflation Attack

Category	Severity	Location	Status
Gaming	● Medium	MultiTokenVault.sol	Acknowledged

Description

In the MultiTokenVault the inflation attack is defended against by using a decimalsOffset to preserve a larger amount of shares relative to the number of assets deposited.

However this mechanism can be circumvented because there is a public forfeit function which can be used to burn shares of the vault and counteract the decimalsOffset, allowing the total initial shares to be manipulated back down to 1 wei by an attacker.

As a result the typical inflation attack can be carried out, resulting in the victim receiving 0 or significantly reduced shares of the vault.

Recommendation

Firstly, make the forfeit function only callable by the LoanManager contract.

Even then the inflation attack may still be possible by transferring Consol tokens directly to the LoanManager before it invokes forfeit with it's total balance.

Consider either requiring that some initial shares are minted to the dead address, or that forfeit cannot be used to forfeit shares below a certain reasonable threshold of totalShares.

Resolution

Buttonwood Team: Acknowledged.

M-02 | enqueueMortgage Cannot Accept Gas Fees

Category	Severity	Location	Status
Unexpected Behavior	● Medium	GeneralManager.sol	Resolved

Description

In the GeneralManager contract the enqueueMortgage function is not marked as payable and therefore cannot accept the necessary Ether to call the conversionQueue with the expected value for gas fees.

As a result the enqueueMortgage function cannot be used to enqueue a mortgage as expected.

Recommendation

Make the external enqueueMortgage function on the GeneralManager payable so that it can accept the necessary gas fee ether to send to the conversionQueue.

Resolution

Buttonwood Team: The issue was resolved in PR#6.

M-03 | Incorrect Async Usage

Category	Severity	Location	Status
Logical Error	● Medium	LoanManager.sol: 318	Resolved

Description

In the `LoanManager` `redeemMortgage` function if the async value is provided as true the non-async path will be taken and if the `async` value is false the asynchronous path will be taken.

Recommendation

Flip the case such that the `async` path is taken if the `async` value is true.

Resolution

Buttonwood Team: The issue was resolved in PR#7.

M-04 | ExpandBalanceSheet Allows Lower Rate

Category	Severity	Location	Status
Gaming	● Medium	Global	Resolved

Description

The expandBalanceSheet action does not perform any validation to ensure that the expansion totalPeriods matches the totalPeriods of the existing mortgage.

As a result, users may create a minimum size expansion directly after initiating a 3-year mortgage at a lower rate, as the three-year treasury rate will typically be lower than the fiver year treasury rate, and expand their existing mortgage to be extended to 5 years while maintaining this lower rate.

This way those who are given the expansion role are able to game the protocol and obtain a 5-year length mortgage at the rate of a 3-year length mortgage. Even without paying any kind of refinance fee.

Recommendation

Validate that an expansion can only be made with a totalPeriods value that matches the existing mortgage.

Resolution

Buttonwood Team: The issue was resolved in PR#25.

M-05 | Expand Balance Sheet Has No Restrictions

Category	Severity	Location	Status
Validation	● Medium	GeneralManager.sol	Resolved

Description

The `expandBalanceSheet` function in the `GeneralManager` performs no validations that the caller is the owner of the balance sheet being expanded, this allows approved callers of the `expandBalanceSheet` function to extend the length of any user’s mortgage at will while expanding their mortgage balance by only the minimum amount.

This results in users holding a mortgage for longer than they expect and requires them to pay more interest than initially promised.

Recommendation

Perform validation that the caller of the `expandBalanceSheet` function is the owner of the mortgage being expanded.

Resolution

Buttonwood Team: The issue was resolved in PR#26.

M-06 | Large Penalty Payments Arbitraged

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description

The penalty payments made through the loanManager penaltyPay function and the interest paid through the periodPay function both cause a stepwise increase in the value of consol tokens which can be taken advantage of, especially for large payments.

An opportunistic actor can observe that a large penalty/interest is about to be paid, or have a large outstanding penalty/interest payment they themselves have to make, and frontrun this action to deposit a large amount of USDX into the consol.

With a non-trivial share of the total supply of consol shares, the opportunistic actor can realize a significant portion of the penalty/interest payment made and effectively vampire attack this from other Consol depositors who held their shares while the penalty/interest was actually accrued over time.

The opportunistic actor may not be able to instantly withdraw their Consol to realize the immediate stepwise gain, however they profit this amount on paper immediately compared to the other Consol shareholders who held for the entire time previous and now had their payment diluted.

Furthermore, there may be cases where a large conversion is available through the conversion or forfeited assets queue which does allow the opportunistic actor to enter and exit consol in a very short period while gaining yield which should have accrued to other Consol holders.

Recommendation

There are many potential solutions to resolve this stepwise increase in share value issue. A fee for outside deposits into Consol could be added to disincentivize this behavior.

The yield paid to the Consol contract could be spread out over time to avoid single stepwise jumps in share value which create such opportunities. Or a deposit delay could be implemented such that there is no guarantee that the actor is able to frontrun the interest/penalty payment value increase.

Resolution

Buttonwood Team: Acknowledged.

M-07 | Conversion Queue Perturbed

Category	Severity	Location	Status
Validation	● Medium	GeneralManager.sol	Resolved

Description

In the `expandBalanceSheet` function there is no validation that requires the caller to provide a non-zero `conversionQueue` address if that mortgage is currently enqueued, even though the mortgage must be re-enqueued into the `ConversionQueue` at the new trigger price.

If this mortgage was previously enqueued in the `ConversionQueue` and is not re-enqueued in the `ConversionQueue` then within the `processWithdrawalRequests` function the mortgage will be executed at the original trigger price when the mortgage was originally enqueued.

This trigger price is not accurate to the trigger price of the new mortgage and in some cases may even reflect a loss for the mortgage's average buy price, if there have not been frequent withdrawals processed, which can result in reverts which block the queue from progressing for an extended period of time.

Recommendation

If the mortgage being expanded in the `expandBalanceSheet` function is already enqueued in the `ConversionQueue`, require that the conversion queue address provided is non-zero and a valid conversion queue.

Resolution

Buttonwood Team: The issue was resolved in PR#50.

M-08 | Borrowers Avoid Interest Through Consol

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description

When a large percentage of Consol holders have pending withdrawal requests in the various queues a large mortgage holder can recoup a large portion of their interest payments by simply depositing USDX into Consol and holding consol while they make their payments.

The borrower can then process the queue entries as they are available so that each entry returns their share of the yield to the borrower's Consol holdings.

In the case where a large mortgage exists and a significant portion of Consol holders are deposited in a queue with entries that can be processed or can be processed soon the mortgage owner can simply pay off their entire mortgage and then recoup the interest in a short period.

Recommendation

To disincentivize this behavior a fee could be levied directly to deposits made to Consol through USDX and outside of the Buttonwood system.

Otherwise, there can be caps on the percentage of Consol that is allowed to be deposited into queues at any given time to limit the feasibility of this scenario.

Resolution

Buttonwood Team: Acknowledged.

M-09 | Consol Price Jumps After Queue Actions

Category	Severity	Location	Status
Frontrunning	● Medium	Global	Acknowledged

Description

Throughout the codebase there are several invocations to `burnExcessShares` throughout the various withdrawal queues. This function call creates a stepwise jump that re-distributes the yield from the holder who is withdrawing their tokens from the queue to the rest of the Consol holders.

This can cause a significant step-wise jump in the Consol token price, especially for large Consol withdrawals which have spent a non-trivial amount of time in the queue. This is likely given the rate of interest payments made and length of mortgages in the Buttonwood cash protocol.

This stepwise jump can be arbitraged by other users who obtain consol right before this stepwise jump and lock in a guaranteed profit that otherwise they would have had to wait to accumulate honestly.

Recommendation

Instead of retroactively burning the excess shares of the users who deposit into the withdrawal queue, consider removing them from the Consol token pool by withdrawing their assets from the Consol vault immediately upon their request to withdraw in the queue.

If the request is cancelled, then simply deposit these assets back into the Consol token at the updated share price.

Resolution

Buttonwood Team: Acknowledged.

M-10 | Re-enqueued Mortgages Trap Gas Fees

Category	Severity	Location	Status
Logical Error	● Medium	ConversionQueue.sol	Resolved

Description

When dequeuing and re-enqueuing a mortgage a new gas fee is required to be covered by the user, however the gas fee provided for the original queue position is not recovered.

As a result, this gas fee amount is left trapped in the ConversionQueue contract.

Recommendation

Refund the gas fee collected by the old, queued mortgage and validate that the current gas fee has been provided.

Resolution

Buttonwood Team: The issue was resolved in PR#37.

M-11 | Incorrect MortgageId Update Logic

Category	Severity	Location	Status
Logical Error	● Medium	MortgageNFT.sol	Resolved

Description

The updateMortgageId function is flawed because it does not remove the association of the previous mortgageId with the tokenId. Therefore, the tokenId is still associated with the old mortgageId in the getTokenId mapping.

As a result, one tokenId could be used to occupy many common mortgageId's, or these leftover getTokenId mapping values can cause confusion and issues for consumers of this mapping in normal use.

Recommendation

Be sure to remove the getTokenId entry for the previous mortgageId in the updateMortgageId function.

Resolution

Buttonwood Team: Resolved.

M-12 | Forclosures Used To Avoid Penalty Payments

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description

When foreclosing a position into the forfeited assets queue the amount of forfeited asset tokens minted corresponds to the outstanding debt of the position without interest. It is one issue that this amount does not include interest.

However, this amount also does not include any penalty payments that the mortgage holder had outstanding. This makes sense from the third party Consol holder's perspective.

However, from the perspective of the mortgage holder, if they have accrued unpaid penalty payments up to this point then it makes sense for them to instead obtain Consol, enter the forfeited assets queue and allow their mortgage to forclose to redeem their underlying collateral without the penalty payments.

Recommendation

To protect against this potential gaming, consider setting aside the unpaid penalties from the collateral seized on foreclosure and distributing it to all Consol holders similar to how would be done in the `penaltyPay` function.

Resolution

Buttonwood Team: Acknowledged.

M-13 | First Epoch Of All OriginationPools Can Be DoSed

Category	Severity	Location	Status
DoS	● Medium	OriginationPoolScheduler.sol: 436C1-445C74	Resolved

Description PoC

The deployOriginationPool function does not have access control, and pools are deployed based on predetermined configs added by the admin.

However, the function does not check if the config for the user-provided oPoolConfigId exists, allowing pools to be deployed with an empty config.

A malicious actor can front-run the admin just before a new config is added, fetch the oPoolConfigId, and deploy a pool before the config is stored.

The resulting pool is unusable, as the Consol and USDX addresses are set to 0. Since OriginationPool deployments can only occur once per epoch, the initial epoch is effectively DoS'ed by this action, and the correct pool with a legitimate config can only be deployed in the next epoch.

Recommendation

If the function is intended to be external, add a check to ensure that config.consol = address(0) and verify that a valid config exists for the provided oPoolConfigId.

Additionally, consider restricting the deployOriginationPool function to admin access only.

Resolution

Buttonwood Team: The issue was resolved in PR#9.

M-14 | Collateral Value Not Considered For Foreclosures

Category	Severity	Location	Status
Logical Error	● Medium	Global	Acknowledged

Description

Foreclosures only occur if the mortgage owner misses more than `MAXIMUM_MISSED_PAYMENTS` payments. There is no check on the backing collateral value of the mortgage.

When the market declines and the collateral value drop significantly, there is no incentive to continue making monthly payments to keep the collateral and redeem it later.

Recommendation

Consider defining a safe ratio between the backing collateral value and the debt and allow mortgages to be foreclosed if the collateral value falls below that ratio, even if there are no missed payments. Additionally, allow borrowers to add more collateral to secure their loans.

Resolution

Buttonwood Team: Acknowledged.

M-15 | USDX Allows Slippage Free Swaps

Category	Severity	Location	Status
MEV	● Medium	USDX.sol	Acknowledged

Description

The USDX contract is a `MultiTokenVault` with no overrides on the deposit or withdrawal functions, and the conversion functions simply adjust the token amount by the relevant decimals. Therefore, asset A can be deposited and asset B can be withdrawn at exactly the same `exchangeRate`.

This allows for a slippage free swap on the supported tokens, e.g. USDC and USDT. If one token depegs by even a small amount USDX would be immediately arbitraged to hold the depegged token at the cost of all USDX holders.

Recommendation

This could be addressed in many ways, including but not limited to adding an exchange rate mechanism for the tokens being deposited and withdrawn from USDX, adding a withdrawal fee that applies when withdrawals are made shortly after deposits, or requiring a cooldown period for withdrawals.

Otherwise acknowledge and be aware of the risk of arbitrage in the case where one token backing USDX depegs, or a user simply wishes to abuse a slippage free swap.

Resolution

Buttonwood Team: Acknowledged.

L-01 | ForfeitedAssetPool Value Changes Drastically

Category	Severity	Location	Status
Logical Error	● Low	ForfeitedAssetsPool.sol: 109-127	Acknowledged

Description

When a mortgage is foreclosed, the entire remaining collateral is deposited into the forfeited assets pool, and an `amountOutstanding` amount of `ForfeitedAssetPool` tokens is minted. Neither the mint amount nor the added collateral value is relevant to the current state of the forfeited assets pool.

As a result, each foreclosure can significantly change the value of a single `ForfeitedAssetsPool` token. Users who want to withdraw from the `ForfeitedAssetsPool` burn their Consol tokens and enter the queue, expecting to get underlying assets from the pool based on the current value.

However, there is no guarantee that the same amount of `ForfeitedAssetsPool` token will be worth a similar value when this withdrawal request is processed, and users may end up withdrawing their Consol tokens at a much lower value than expected.

Recommendation

Consider allowing users to provide a minimum value when requesting a withdrawal from the `ForfeitedAssetsPool`, or clearly document this behavior, since users who enter the `ForfeitedAssetsQueue` should be aware of it.

Resolution

Buttonwood Team: Acknowledged.

L-02 | Some Lenders Don’t Receive Yield

Category	Severity	Location	Status
Unexpected Behavior	● Low	UsdxQueue.sol	Acknowledged

Description

Due to the structure of the USDX queue, some lenders who deposited into the origination pool and received Consol tokens will not be able to withdraw their principal or yield for a significant period of time.

Because a queue was chosen for this process, the lenders who submit their withdrawal requests in the USDX queue first will be fully redeemed, with interest, entirely first.

And the ones who queue after them will have to wait until the final mortgages are fully repaid to redeem their principle and realize any interest.

As a result, some lenders will be able to withdraw within hours to days, while others will have to wait months to withdraw any amount of their Consol backing amount.

Recommendation

Consider restructuring the Consol redemption such that all Consol holders can withdraw a portion of their backing tokens as principal and interest is paid down over time, where this portion increases for all Consol holders evenly over time as payments are made.

Resolution

Buttonwood Team: Acknowledged.

L-03 | Forfeited Assets Queue Zero Value DoS

Category	Severity	Location	Status
DoS	● Low	ForfeitedAssetsPool.sol	Resolved

Description

The forfeited assets queue burn function iterates through all of the supported tokens and transfers them out to the burner relative to their holdings of the forfeited assets shares.

However, if any of the `redeemedAssets` tokens reverts on zero value transfers and the corresponding contract balance is empty, due to no foreclosures occurring for that token, then a zero-value transfer revert will occur.

This will result in a stuck forfeited assets queue until a foreclosure occurs for that asset or the admin sends some tokens to the contract to correct it.

Recommendation

Only perform the `safeTransfer` call if the amount to transfer is non-zero.

Resolution

Buttonwood Team: The issue was resolved in PR#34.

L-04 | Cancelled Mortgages Retain The MortgageId

Category	Severity	Location	Status
Logical Error	● Low	MortgageNFT.sol	Resolved

Description

In the burn function of the MortgageNFT contract there is no clearing of the getMortgageId or getTokenId mappings for the relevant tokenId. Therefore, the tokenId continues to occupy the entries in these mappings and act as if it still existed.

Recommendation

Remove the entries associated with the tokenId that is being burned in the burn function.

Resolution

Buttonwood Team: The issue was resolved in PR#33.

L-05 | Withdrawal Queue Traps Yield

Category	Severity	Location	Status
Logical Error	● Low	Global	Acknowledged

Description

For the last holder of the Consol token the yield generated is trapped in the withdrawal queue contract and cannot be recovered by the protocol.

Consider the following series of actions using a withdrawal request and subsequent cancellation as an example:
• 100 total shares 100 total tokens
• Request withdrawal of 100 shares worth 100 tokens
• Withdrawal time: 100 shares worth 110 tokens, price per share = 1.1
• 100 total shares 110 total tokens
• SharesMinted = 100 * (100 - 100) / (110 - 100) > (100-100 totalShares is 0) > 100
• Shares - shares minted = 0
• No Adjustment for totalSupply or the sender's balance
• Transfer(100)
• Convert to shares(100) > shares = 100 * 100 / 110 = 90.909 shares transferred
• 100 - 90.909 = 9.091 shares representing 10 tokens remain trapped in the withdrawal queue contract.

If the final Consol divestment is large this could result in a significant amount of funds trapped in the lending queue.

Recommendation

Consider adding a way to rescue any yield that could be trapped in the lending queue that would occur when the last Consol holder divests or cancels from the queue.

Resolution

Buttonwood Team: Acknowledged.

L-06 | Interest Rate Used From Order Creation

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol	Resolved

Description

In the `_prepareOrder` function the interest rate for an order is fetched from the `PythInterestRateOracle`.

However, the interest rate recorded from the oracle at the time of request creation may be significantly different than the current rate in effect when the mortgage is actually originated.

Since there is no validation that the expiry is at most a certain amount in the future it's possible that some mortgage requests could stick around in the `OrderPool` and use stale interest rates when originated.

Recommendation

Consider validating that the expiry is not too far in the future upon creation requests.

Resolution

Buttonwood Team: The issue was resolved in PR#35.

L-07 | Forfeited Asset Pool Value Inflation

Category	Severity	Location	Status
Logical Error	● Low	Global	Acknowledged

Description

When assets are forfeited to the forfeited assets pool during foreclosure the amount of outstanding debt is minted as forfeited asset shares, however the entire mortgage collateral is deposited into the pool.

In cases where the mortgage position has already had payments made these changes the exchange rate of forfeited asset shares and allows the Consol holder at the front of the forfeited assets queue to withdraw more value than their consol should be worth.

For the most explicative example, consider a mortgage where all but 1 wei of the termBalance has been paid off, and the collateral is 10 BTC. The amount flash swapped from the Consol contract is simply 1 wei, and therefore the amount of new forfeited assets shares minted is just 1 wei.

However, 10 BTC goes into the forfeited assets pool, and all of a sudden, the first Consol holder in the queue can redeem 1 wei of Consol for 10 BTC, while the queued redemptions behind them do not get the liquidity they should have to exit from this foreclosure.

Recommendation

Either mint and transfer the current dollar amount of all collateral forfeited to the Consol contract, thus increasing the balance of all Consol holders and allowing them to exit fairly through the forfeited assets queue or consider converting between the amount of forfeited assets shares and underlying dollar value of collaterals in the forfeited assets pool so that Consol holders can be redeemed at a 1 USD value per Consol like the other exit queues.

Resolution

Buttonwood Team: Acknowledged.

L-08 | Multiple OPool Origination Discrepency

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Resolved

Description

The whitepaper suggests that multiple OriginationPool's can be used to originate a loan for an order, however the order fulfillment flow in the OrderPool contract only allows for one OriginationPool that has been specified by the user to be used with the mortgage origination.

This may limit the size of individual mortgages that can be created with a single request, and deviates from the whitepaper.

Recommendation

Confirm if this is the expected behavior or not, and if the whitepaper is simply outdated.

Resolution

Buttonwood Team: The issue was resolved in PR#46.

L-09 | Collateral Caps Not Validated On Execution

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol	Resolved

Description

The minimum and maximum caps for the chosen collateral token are validated upon a mortgage request but not upon the execution of that mortgage request.

Therefore, if these caps are updated in between the request and execution the resulting order execution may violate the most up to date caps.

Recommendation

Consider validating the minimum and maximum caps for an order during the execution of that order in addition to the request creation.

Resolution

Buttonwood Team: The issue was resolved in PR#36.

L-10 | Expansion Requests Can Unexpectedly Fail

Category	Severity	Location	Status
Unexpected Behavior	● Low	ConversionQueue.sol	Resolved

Description

In the origination flow, if the order being executed is an expansion order the reenqueue boolean is always marked as true when a conversion queue is provided.

However, if the mortgage being expanded is not already in the conversion queue, then this enqueue request will fail with the TokenIdNotInQueue error.

This leads to unexpectedly un-executable expansion orders which must be re-requested without a conversion queue and enqueued in a separate transaction.

Recommendation

Consider only re-enqueuing a mortgage if that mortgage is already present in the ConversionQueue.

Resolution

Buttonwood Team: The issue was resolved in PR#50.

L-11 | Blacklisted Addresses May Halt Queues

Category	Severity	Location	Status
DoS	● Low	Global	Acknowledged

Description

The conversion queue transfers collateral tokens directly to the withdrawer’s address. As a result, if the withdrawer is blacklisted for the underlying collateral token, they can halt the withdrawal queue due to a revert which occurs when attempting to transfer to their blacklisted address.

A similar action can occur that prevents the Forfeited assets queue from functioning.

Recommendation

If the transfer fails in any queue, consider incrementing a mapping value whereby the withdrawer can claim the collateral token in a separate transaction. Otherwise, be sure to only list collateral tokens which do not have a blacklist mechanism.

Resolution

Buttonwood Team: Acknowledged.

L-12 | Zero Payments Allowed

Category	Severity	Location	Status
Validation	● Low	LoanManager.sol	Resolved

Description

In the `periodPay` and `penaltyPay` functions there is no validation that prevents a 0-amount payment from occurring.

This could lead to unexpected issues since a zero payment is not an expected operation. Furthermore, a minimum amount validation could be performed to add additional safety checks.

Recommendation

Validate that the amount being paid in the `periodPay` and `penaltyPay` functions is non-zero and ideally introduce a minimum amount validation for these payments.

Resolution

Buttonwood Team: The issue was resolved in PR#30.

L-13 | Refinance Allowed For Completed Mortgages

Category	Severity	Location	Status
Validation	● Low	LoanManager.sol	Resolved

Description

The `refinanceMortgage` function performs no validation that prevents a user from refinancing a mortgage that is nearly entirely paid off or entirely paid off already.

This allows the user to create a mortgage with a term balance of 0 which can lead to unexpected edge cases in the protocol.

Recommendation

Validate that the mortgage being refinanced has above a minimum threshold of debt before allowing the refinance to occur.

Resolution

Buttonwood Team: The issue was resolved in PR#31.

L-14 | Conversion Queue May Get Stuck

Category	Severity	Location	Status
Warning	● Low	ConversionQueue.sol	Resolved

Description

In the processWithdrawalRequests function for the conversionQueue, the numberOfRequests is only decremented when a withdrawal request is fully fulfilled. Additionally, there is no maximum limit on the size of a withdrawal request.

Therefore, for large withdrawal requests it is possible that the queue processing can be halted due to the invocation of the processWithdrawalRequests function with even a 0 numberOfRequests value.

This is possible if there are many mortgages near the minimum size which can be matched against a single large withdrawal request, requiring that many loop iterations be completed before the processWithdrawalRequests function can finish execution, which could use more than the block gas limit to fully process.

Recommendation

To resolve this issue, consider implementing a maximum size for the individual withdrawal request amount, such that it is a multiple of the minimum mortgage size which can be processed in a single block.

Otherwise, adapt the usage of the numberOfRequests value in the processWithdrawalRequests function such that it decrements upon every loop iteration, even when the withdrawal request is only partially processed and a mortgage is fully processed.

Resolution

Buttonwood Team: The issue was resolved in PR#47.

L-15 | Converted Positions Can Avoid Penalties

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Acknowledged

Description

When a position is fully converted without making any period payments the termBalance becomes 0. As a result, the position no longer accrues late payments in the _applyPendingMissedPayments function and cannot be forclosed on as a result.

For any position which had accrued penalty payments and then gets entirely converted they will not have any time constraint on paying these penalties as foreclosure cannot occur.

Recommendation

The user still has to pay off these penalties to redeem their position, and this serves as the only incentive to do so. Be aware of this case where a mortgage cannot be foreclosed on and penalty payments can be put off.

Resolution

Buttonwood Team: Acknowledged.

L-16 | Creation Request DoS

Category	Severity	Location	Status
DoS	● Low	GeneralManager.sol	Resolved

Description

The requestMortgageCreation function accepts a creationRequest object with a mortgageId NFT id attribute to mint the corresponding mortgage NFT id to the caller.

However, if this mortgageId is already taken then the mint invocation and thus the higher level requestMortgageCreation function invocation reverts.

As a result, it may be possible for a malicious actor to frontrun and DoS a user's calls to the requestMortgageCreation function by taking their specified mortgageId first.

Recommendation

Be aware of this and consider refactoring the mortgageId logic to be keyed based on the owner of the mortgage.

Resolution

Buttonwood Team: The issue was resolved in PR#17.

L-17 | Yield Must Be Withdrawable From The Strategy

Category	Severity	Location	Status
Warning	● Low	SubConsol.sol	Acknowledged

Description

In the SubConsol contract there is no way to withdraw yield from the underlying yieldStrategy beyond the collateral deposited into it.

This may be unexpected depending on the intended use-case for the yieldStrategy and the target yieldStrategy implementations.

Recommendation

Be sure there is a way to withdraw the yield from the yieldStrategy so that this amount is not lost.

Resolution

Buttonwood Team: Acknowledged.

L-18 | Asset Removal Issues

Category	Severity	Location	Status
Warning	● Low	Global	Acknowledged

Description

In the case where an asset is being updated as no longer supported and has a non-zero balance in either the USDX or Consol contract a stepwise decrease in the vault shares value will occur.

In the case of Consol this creates a DoS for the queues which rely on the share price increasing monotonically due to the `burnExcessShares` function.

Recommendation

Be aware of this risk and be sure to never delist an asset that has a nonzero balance in the Consol contract. It may be best to not explicitly limit this at a contract level to allow the admin to make important updates in emergency situations.

Furthermore, consider updating the `burnExcessShares` function so that it can successfully execute and early return if the share value has decreased since the withdrawal request was made.

Resolution

Buttonwood Team: Acknowledged.

L-19 | Missing SubConsol Validation

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol	Resolved

Description

In the `requestMortgageCreation` function there is no validation which prevents a user from providing a `SubConsol` address that is invalid either because it is not supported as a `SubConsol` address or it does not match the collateral token the user is using.

This leads to orders being created which are non executable due to reverts in the `LoanManager.createMortage` function.

Recommendation

Add validation in the `requestMortgageCreation` to prevent users from creating orders with a `SubConsol` address that is not a registered `SubConsol` contract and does not support the collateral they are providing.

Resolution

Buttonwood Team: The issue was resolved in PR#23.

L-20 | Lacking Configs Validation

Category	Severity	Location	Status
Validation	● Low	OriginationPoolScheduler.sol	Resolved

Description

In the addConfig function there is no validation that prevents a config with a zero consol address from being added, such a config would be able to be added again, creating a duplicate in the _oPoolConfigIds list and not allowing this config to be removed.

Recommendation

Consider validating that the consol on the provided config is not the zero address.

Resolution

Buttonwood Team: The issue was resolved in PR#22.

L-21 | Zero Amount Actions Allowed

Category	Severity	Location	Status
Validation	● Low	Global	Resolved

Description

Across the codebase in the `ForfeitedAssetsPool`, `Consol`, `USDX`, and `SubConsol` contracts 0 value deposits and withdrawals or burns are allowed which may lead to unexpected edge cases or event emissions.

Recommendation

Validate that the amount being deposited or withdrawn or burned is non-zero to avoid unnecessary paths and unexpected edge cases.

Resolution

Buttonwood Team: The issue was resolved in PR#43.

L-22 | Lacking Refund Mechanism

Category	Severity	Location	Status
Unexpected Behavior	● Low	GeneralManager.sol	Resolved

Description

In the GeneralManager requestMortgageCreation function among others, there is no ether value refund in the event that the value sent is greater than the order pool's gas fee.

In the event that the default admin re-assigns the gas fee value for the OrderPool before a user's transaction is recorded, this may result in the user sending more gas than is necessary and not receiving an appropriate refund back.

Recommendation

Consider sending a refund of any excess ether to the caller at the end of the transaction.

Resolution

Buttonwood Team: The issue was resolved in PR#8.

L-23 | Lacking SafeCast Usage

Category	Severity	Location	Status
SafeCast	● Low	Global	Resolved

Description

In the periodsSinceTermOrigination function the periods amount is casted to a uint8 without checking if the result fits within a uint8 object. This will result in a perturbation of the resulting periods value when the time passed exceeds 256 periods.

In the periodsPaid function the resulting value is casted to a uint8 without validating that the result fits within this storage.

In the interestRate function of the pyth interest rate oracle the rate result is casted to a uint16 without checking if it fits within this storage.

In the price function the price result is casted to a uint64 value without checking if it fits within this storage.

Recommendation

Use safeCast for all of these casting results as well as all other casting that occurs in the codebase to avoid any unexpected overflows.

Resolution

Buttonwood Team: The issue was resolved in PR#29.

L-24 | Convert Rounds In Favor Of The User

Category	Severity	Location	Status
Rounding	● Low	MortgageMath.sol	Resolved

Description

In the convert function, when a partial conversion is occurring the remaining termPaid is calculated using Math.Rounding.Ceil, however the termPaid amount should be rounded down to round against the user and in favor of the protocol as a best practice.

Recommendation

Round the termPaid by the user down instead of up to favor the safety of the protocol.

Resolution

Buttonwood Team: The issue was resolved in PR#27.

L-25 | Lacking Expiration Validation

Category	Severity	Location	Status
Validation	● Low	OrderPool.sol	Resolved

Description

The `OrderPool` `sendOrder` function allows orders to be stored that have an expiration that has already passed.

Recommendation

Do not allow these orders to be stored and instead revert.

Resolution

Buttonwood Team: The issue was resolved in PR#28.

L-26 | Ineffective Mortgage Request Fills

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Resolved

Description

The fulfiller of a mortgage origination receives no spread on the collateral price for fulfilling the order.

Furthermore, the price used is a static price from when the order was originated, therefore if price increases after the pyth price used at the time of the request initiation then it is unlikely that the order will be fulfilled, and this is simply a waste of the gas down payment made by the user.

Recommendation

Consider improving the fulfillment incentives so that mortgage requests are more likely to be fulfilled.

Resolution

Buttonwood Team: The issue was resolved in PR#54.

L-27 | Actions Can Only Occur During Market Hours

Category	Severity	Location	Status
Warning	● Low	Global	Resolved

Description

Given that the Pyth feed for the US treasury rate is only updated during relevant market hours it is not possible to query the rate with a recent published result during off-market hours.

As a result, actions like loan request creation and refinancing are unavailable during off-hours.

Recommendation

Be aware of this behavior and be sure to surface it to users in a palatable way.

Resolution

Buttonwood Team: The issue was resolved in PR#45.

L-28 | Term And Penalty Payments Can Be Grieved

Category	Severity	Location	Status
DoS	● Low	MortgageMath.sol: 281-283	Resolved

Description

In the case where a borrower attempts to pay their full penalty or term value, a malicious user can cause the borrower's payment transaction to revert by frontrunning and paying the loan down by 1 wei.

The reason for the revert is because the borrower is not able to overpay their penalties or outstanding principal. The single-wei donation will cause the borrower's full payment to overpay by 1 wei, leading to their payment transaction reverting.

This is possible for all penalty payments and only period payments when the mortgage has a payment plan.

Recommendation

The simple solution is to lock down payments only to the mortgage owner. Otherwise, allow the borrower to overpay and simply refund the excess balance at the end of the transaction.

Resolution

Buttonwood Team: The issue was resolved in PR#17.

L-29 | Check Balance In removeAsset

Category	Severity	Location	Status
Unexpected Behavior	● Low	Global	Resolved

Description

The removeAsset function in the ForfeitedAssetsPool contract removes the asset from the list without checking the contract balance.

If this asset has already been forfeited before, that balance remains locked in the contract unless the asset is added back to the list, since a removed asset cannot be redeemed during a burn.

Recommendation

Consider checking the contract balance in removeAsset and either reverting if the asset has already been forfeited or clearing the balance. However, note that clearing the balance might be unfair to users who are awaiting withdrawal.

Resolution

Buttonwood Team: The issue was resolved in PR#39.

L-30 | USDX Withdrawals Can Be DoSed

Category	Severity	Location	Status
DoS	● Low	MultiTokenVault.sol: 165-214	Resolved

Description

The USDX.sol contract allows users to deposit and withdraw various USD-valued tokens. Since the deposit and withdrawal flow does not impose fees, users are able to swap between tokens freely.

This poses a problem because when users specify their withdrawal token, a malicious actor can deposit another token and withdraw all of the user's desired token's balance. All withdrawals can be temporarily grieved through this method.

Recommendation

Note this possibility in the documentation.

Resolution

Buttonwood Team: The issue was resolved in PR#49.

L-31 | ForfeitedAssetsQueue Can Be DoSed

Category	Severity	Location	Status
DoS	● Low	ForfeitedAssetsQueue.sol: 51	Acknowledged

Description

The ForfeitedAssetsQueue handles each withdrawal request sequentially and must fill have enough liability in the ForfeitedAssetsPool to be able to fulfill the request, otherwise it reverts.

Therefore, a large Consol holder can queue a withdrawal through the ForfeitedAssetsQueue and DoS the queue until there are enough forfeited assets to satisfy the large withdrawal.

The user could cancel their request resulting in simply stalling the distribution of forfeited assets and a delay in allowing users to exit their Consol positions.

This is unlikely to occur in practice because it would take a large financial commitment by the malicious user and provide little to no benefit.

Recommendation

Document the possibility of such a scenario.

Resolution

Buttonwood Team: Acknowledged.

L-32 | Incorrect Natspec Comment

Category	Severity	Location	Status
Informational	● Low	MortgageQueue.sol: 200	Resolved

Description

The _findFirstTriggered function has the following Natspec comment:

"@return tokenId the tokenId of the first MortgagePosition in the Conversion Queue that has a trigger price greater than or equal to the input trigger price."

However, the function returns the position with a trigger price less than or equal to the input trigger price.

Recommendation

Change the comment.

Resolution

Buttonwood Team: The issue was resolved in PR#38.

L-33 | Inactive Mortgages In ConversionQueue

Category	Severity	Location	Status
Unexpected Behavior	● Low	ConversionQueue.sol	Resolved

Description

Redeemed or foreclosed mortgages remain in the `ConversionQueue` until the `dequeueMortgage` function is called.

If such a mortgage becomes the head of the queue, the `processWithdrawalRequests` function will revert until that mortgage is removed.

Recommendation

Consider either dequeuing a mortgage when it becomes inactive or skipping it in the while loop of the `processWithdrawalRequests` function.

Resolution

Buttonwood Team: The issue was resolved in PR#48.

L-34 | Old forfeitedAssetsPool Remains Supported

Category	Severity	Location	Status
Logical Error	● Low	Consol.sol: 61-70	Resolved

Description

When updating the forfeitedAssetsPool in Consol.sol, the previous value of forfeitedAssetsPool remains in the supportedTokens enumerable set.

The cap value for the previous pool is deleted, therefore it would be wise to remove the pool token from the list of supported tokens to avoid its use in the flashSwap() function.

Recommendation

Consider removing the previous forfeitedAssetsPool from the enumerable set with supportedTokens.remove().

Resolution

Buttonwood Team: The issue was resolved in PR#18.

L-35 | Duplicate Deletion Of Order

Category	Severity	Location	Status
Error	● Low	OrderPool.sol: 168-176	Resolved

Description

When an order is expired, the order record is deleted twice. Firstly, in the if clause:

```
if (order.expiration < block.timestamp) {
// Cancel the mortgage request
IGeneralManager(generalManager).burnMortgageNFT(order.mortgageParams.tokenId);
// Delete the order
delete _orders[index];
// Emit the PurchaseOrderExpired event
emit PurchaseOrderExpired(index);
```

And finally at the end of function execution:

```
collectedGasFee = order.gasFee;
// Delete the order
delete _orders[index];
```

Recommendation

Remove the deletion in the if clause as the order will eventually be deleted at the end of function execution.

Resolution

Buttonwood Team: The issue was resolved in PR#19.

L-36 | Early Closures Still Pay Full Interest

Category	Severity	Location	Status
Informational	● Low	Global	Acknowledged

Description

Borrowers may close their mortgages early if they choose. However, closing early does not recalculate the interest or total amount payable, and borrowers are still required to pay interest for the entire mortgage term.

Recommendation

Document this behavior and ensure users are aware of it.

Resolution

Buttonwood Team: Acknowledged.

L-37 | Race Condition For ForfeitedAssetsPool Tokens

Category	Severity	Location	Status
Logical Error	● Low	Consol.sol, MultiTokenVault.sol	Acknowledged

Description

Consol is a MultiTokenVault that supports USDX and ForfeitedAssetsPool tokens. The total supply of MultiTokenVaults is calculated based on the balances of the supported tokens, and MultiTokenVault assumes that all supported tokens share the same unit of account (UOA).

However, the UOA of the supported tokens in Consol can differ significantly. For example, a single share of ForfeitedAssetsPool can easily be worth 1.50 due to seized collaterals, while USDX is 1,00.

As a result, Consol supporting both of these tokens contradicts the intended behavior of a MultiTokenVault.

Since seized assets can be purchased at 1:1 by burning Consol, a race condition can occur when a single share of ForfeitedAssetsPool is worth more than $1. Users will attempt to withdraw from the ForfeitedAssetsPool first, followed by withdrawals from USDX.

Recommendation

Consider documenting this if it is the intended behavior. Otherwise, ensure that all supported assets in Consol share the same UOA.

This may require calculating the mint amount of ForfeitedAssetsPool tokens at the time of foreclosure based on the value of the seized collateral and the value of Consol at that point.

Resolution

Buttonwood Team: Acknowledged.

L-38 | Penalty Calculation Rounds In Favor Of Borrower

Category	Severity	Location	Status
Rounding	● Low	MortgageMath.sol: 337-341	Resolved

Description

When calculating the number of penalties that the borrower owes, the penalty calculation rounds down in favor of the late borrower.

Likely this only results in a negligible advantage for the late borrower; however, it is best practice to always round in favor of the protocol.

Recommendation

Use `Math.Rounding.Ceil` to round in favor of the protocol.

Resolution

Buttonwood Team: The issue was resolved in PR#40.

L-39 | Initial oPoolAdmin Does Not Have Admin Role

Category	Severity	Location	Status
Configuration	● Low	OriginationPoolScheduler.sol: 114	Resolved

Description

oPoolAdmin in the OriginationPoolScheduler must have the DEFAULT_ADMIN_ROLE. However, this role is not granted to oPoolAdmin during the initialization process. Instead, it is granted to msg.sender, which may not be the oPoolAdmin.

Recommendation

Grant the admin role to oPoolAdmin during initialization.

Resolution

Buttonwood Team: The issue was resolved in PR#21.

L-40 | Penalty Rate Updates Affect Existing Mortgages

Category	Severity	Location	Status
Best Practices	● Low	LoanManager.sol: 75	Acknowledged

Description

If the admin updates the penalty rate value stored in `GeneralManager.sol`, the currently late mortgages could experience increased penalty fees.

These mortgage owners enter into the agreement with a set rate and can experience fluctuations depending on the global penalty rate setting.

Recommendation

Consider writing the current global `penaltyRate` for the mortgage at the time of origination request and use that value in subsequent penalty calculations.

Resolution

Buttonwood Team: Acknowledged.

L-41 | Theft Of Funds From USDX With scalarDenominator

Category	Severity	Location	Status
Rounding	● Low	MultiTokenVault.sol: 199	Resolved

Description

There is a possible, but unlikely theft vector within `USDX.sol`. The USDX contract allows users to deposit and withdraw various USD-valued tokens.

Because each token may have different decimals, there is a scaling mechanism that multiplies the token amount by a numerator and divides by a denominator.

The potential theft vector occurs when the denominator is greater than 1, for example 10e12. Imagine USDX is set up with 6-decimal precision and 18-decimal USD tokens must be scaled down to 6 decimals. When calculating the mint amount of 1e18 token, the resulting amount would be 1e6.

When the user attempted to withdraw 1e18 tokens, the same conversion would take place and require that they burn 1e6 pool tokens.

The theft can occur if the user specifies that they wish to withdraw (1e18 + 1e11) tokens. With precision loss, the amount of pool tokens to burn will still be 1e6.

However, the user will receive 1e11 tokens more than they deposited. They can perform this call many times to drain funds from the pool.

Recommendation

Ensure that the denominator is always 1, as is the case in the deployment script.

Resolution

Buttonwood Team: The issue was resolved in PR#41.

L-42 | CEI Pattern Not Followed

Category	Severity	Location	Status
Best Practices	● Low	LenderQueue.sol: 178-179	Resolved

Description

When cancelling a withdraw request, the consol token is transferred to the user, however, the amount is not updated to 0 until after the external call.

If Consol were to include a hook for the receiver of the token, the withdrawal queue could be drained. It is unlikely that such would be the case.

Recommendation

Move the storage writes above the external call to the consol token.

Resolution

Buttonwood Team: The issue was resolved in PR#20.

I-01 | calculateNewAvergageInterestRate Naming

Category	Severity	Location	Status
Typo	● Info	MortgageMath.sol	Resolved

Description

The `calculateNewAvergageInterestRate` function misspells average.

Recommendation

Correct the spelling of average in the `calculateNewAvergageInterestRate` function name.

Resolution

Buttonwood Team: The issue was resolved in PR#16.

I-02 | Typos

Category	Severity	Location	Status
Informational	● Info	LoanManager.sol: 417	Resolved

Description

• Corresponding ->
https://github.com/GuardianOrg/cash-buttonwood-cash-team2/blob/bc9e1c53bc1f549af634ad100a5c0b6a3f6f54a4/src/LoanManager.sol#L417

• Surplus ->
https://github.com/GuardianOrg/cash-buttonwood-cash-team2/blob/bc9e1c53bc1f549af634ad100a5c0b6a3f6f54a4/src/LoanManager.sol#L270

• Additional ->
https://github.com/GuardianOrg/cash-buttonwood-cash-team2/blob/bc9e1c53bc1f549af634ad100a5c0b6a3f6f54a4/src/interfaces/IGeneralManager/IGeneralManager.sol#L226

Recommendation

Consider fixing these typos.

Resolution

Buttonwood Team: The issue was resolved in PR#42.

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
H-01	Some Requests Cannot Be Cancelled	Logical Error	● High	Resolved
H-02	amountForfeited Credits Mortgage Holders	Logical Error	● High	Resolved
H-03	Gas Fees Trapped In enqueueMortgage	Logical Error	● High	Resolved
H-04	Missed Payments Perturbed	Logical Error	● High	Resolved
H-05	Rounding Prevents Action Cancellations	Rounding	● High	Resolved
H-06	Multiple Origination Pools Broken	Logical Error	● High	Resolved
M-01	Mortgage Fee Lost On Expiration	Warning	● Medium	Resolved
M-02	Risk Free Time Difference Arbitrage	Gaming	● Medium	Acknowledged
M-03	Collateral Consumed At Stale Trigger Price	Logical Error	● Medium	Acknowledged
M-04	Converted No Payment Plan Mortgages	Logical Error	● Medium	Resolved
M-05	No Payment Plan Mortgages Arbitraged	Gaming	● Medium	Acknowledged
L-01	Empty Requests Can Be Cancelled	Validation	● Low	Resolved
L-02	safeTransferFrom Should Come First	Best Practices	● Low	Acknowledged

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
L-03	More Than 18 Decimals Incompatible	Validation	● Low	Acknowledged
L-04	Lacking CEI In Burn	Best Practices	● Low	Resolved
L-05	Dangerous Withdrawal Request Amounts	Validation	● Low	Acknowledged
L-06	Lacking Pausability Guards	Validation	● Low	Resolved
L-07	Missing Reentrancy Guard	Validation	● Low	Resolved
L-08	Blacklist Prevents Order Processing	Warning	● Low	Acknowledged
L-09	Lacking Processing CEI	Warning	● Low	Resolved
L-10	Lacking Length Validation	Validation	● Low	Resolved
L-11	Outdated Gas Fee Validation	Validation	● Low	Acknowledged
L-12	Typo	Typo	● Low	Resolved
L-13	Dangerous Conversion Queue Refund	Best Practices	● Low	Resolved
L-14	Lacking Individual Borrow Validation	Validation	● Low	Acknowledged
L-15	Duplicate Origination Pools	Validation	● Low	Resolved

Remediation Findings & Resolutions

ID	Title	Category	Severity	Status
L-16	Lacking Reentrancy Guards	Validation	● Low	Resolved
L-17	Unused subConsol Variable	Superfluous Code	● Low	Resolved
L-18	Incorrect NatSpec Comment	Informational	● Low	Resolved
L-19	Redundant Check In convert	Superfluous Code	● Low	Acknowledged
L-20	Redundant Values	Logical Error	● Low	Acknowledged
L-21	Forfeited Assets Queue Held Up	Warning	● Low	Acknowledged
L-22	Missing requestWithdrawal Refund	Warning	● Low	Acknowledged
L-23	Incorrect Event Emission	Logical Error	● Low	Resolved
L-24	Count Unexpectedly Incremented Twice	Unexpected Behavior	● Low	Resolved

H-01 | Some Requests Cannot Be Cancelled

Category	Severity	Location	Status
Logical Error	● High	LenderQueue.sol: 158	Resolved

Description

In the `cancelWithdrawal` function the validation on the index ensures that the index is less than the `withdrawalQueueLength`. However, the `withdrawalQueueLength` is variable and the indexes are now fixed.

Therefore, there can be a withdrawal at absolute index 10, while the withdrawal head is at 10 and the withdrawal queue length is 1. The index provided to the `cancelWithdrawal` function must be 10, which is greater than the `withdrawalQueueLength` of 1.

Consider the following example:
• The withdrawal queue is empty with the head being 0 and length being 0
• Bob creates withdrawal A, which is stored at index 0, the head is still 0 and length is now 1
• Alice creates withdrawal B, which is stored at index 1, the head is still 0 and length is now 2
• Bob’s withdrawal A is now processed; the head is at 1 and the length is now 1
• Alice attempts to cancel her withdrawal, however providing the index 1 fails the index validation and she cannot cancel

Furthermore, the current validation allows requests that are before the head to be cancelled, however these requests have already been processed.

Recommendation

Correct the index validation such that only indexes that satisfy `head < index < head + length` are allowed.

Resolution

Buttonwood Team: The issue was resolved in PR#57.

H-02 | amountForfeited Credits Mortgage Holders

Category	Severity	Location	Status
Logical Error	● High	MortgageMath.sol	Resolved

Description

The amountConverted variable has been changed to represent the amount of debt converted in past terms. And the current term converted value is now tracked in the termConverted variable. This subtly changes all formulas that currently use amountConverted, and if termConverted is not also tracked there this causes notable accounting issues.

The amountForfeited function uses amountConverted but does not account for the newly introduced termConverted correctly.
The original implementation of amountForfeited was as follows:
amountBorrowed - amountConverted - amountOutstanding
Where amountOutstanding = amountBorrowed - amountConverted - amountPrior - convertToPrincipal(termPaid)

And in this context amountConverted represents the entire amount converted across all terms for the mortgage.

This simplifies to:
amountBorrowed - amountConverted - amountOutstanding
= amountBorrowed - amountConverted - (amountBorrowed - amountConverted - amountPrior - convertToPrincipal(termPaid))
= amountPrior + convertToPrincipal(termPaid)

This is correct because this is the amount of debt that the user has already paid and is forfeiting during liquidation.

However, the new implementation of amountForfeited is as follows:
amountBorrowed - amountConverted - principalRemaining
Where principalRemaining = amountBorrowed - amountConverted - amountPrior - convertToPrincipal(termPaid + termConverted)

And in this context amountConverted represents the entire amount converted across only past terms for the mortgage.

This simplifies to:
amountBorrowed - amountConverted - principalRemaining
= amountBorrowed - amountConverted - (amountBorrowed - amountConverted - amountPrior - convertToPrincipal(termPaid + termConverted))
= amountPrior + convertToPrincipal(termPaid + termConverted)

This is incorrect because now the mortgage holder is credited with forfeiting the converted amount in the current term.

Recommendation

Update the forfeitedAssets function as follows:

```
function amountForfeited(MortgagePosition memory mortgagePosition) internal pure returns (uint256) {
if (mortgagePosition.status = MortgageStatus.FORECLOSED) {
return 0;
}
• return mortgagePosition.amountBorrowed - mortgagePosition.amountConverted - mortgagePosition.principalRemaining();
+ return mortgagePosition.amountPrior + mortgagePosition.convertPaymentToPrincipal(mortgagePosition.termPaid);
}
```

Resolution

Buttonwood Team: The issue was resolved in PR#72.

H-03 | Gas Fees Trapped In enqueueMortgage

Category	Severity	Location	Status
Logical Error	● High	GeneralManager.sol	Resolved

Description

The enqueueMortgage function uses the _calculateRequiredGasFee function to calculate the msg.value that is required to send to all of the conversion queues being added with the conversionQueueList parameter.

However, the _calculateRequiredGasFee function computes the required gas fees for all conversion queues that the tokenId currently has in their conversionQueues(tokenId) list, not just the ones that are being added with the call to enqueueMortgage.

Therefore, the requiredGasFee is larger than what is necessary to send to the new conversion queues, and this amount is lost to the mortgage owner.

Recommendation

Instead of using the _calculateRequiredGasFee function which computes the necessary gas fee for all of the conversion queues, old and new. Implement a bespoke for-loop to loop through all of the new conversion queues in the provided conversionQueueList parameter.

Resolution

Buttonwood Team: The issue was resolved in PR#73.

H-04 | Missed Payments Perturbed

Category	Severity	Location	Status
Logical Error	● High	MortgageMath.sol	Resolved

Description

In the applyPenalties function, for mortgages that do not have a payment plan, the additionalPaymentsMissed are assigned to:

periodsSinceOrigination - mortgagePosition.totalPeriods + 1 - mortgagePosition.paymentsMissed

However, in the periodPay and convert functions, the mortgagePosition.paymentsMissed value is assigned to the following for mortgages with and without payment plans:

mortgagePosition.paymentsMissed = _periodsPaid > periodsSinceOrigination 0 : periodsSinceOrigination - _periodsPaid;

There are several issues with this, the most glaring being that in conversion the paymentsMissed update does not account for the fact that mortgages without a payment plan cannot have missed payments before their mortgage is complete.

This is not an issue in the periodPay function since mortgages without payment plans must be totally paid off when invoking the periodPay function.

Secondly, if the periodsSinceOrigination is larger than the total mortgage periods then the paymentsMissed will always be assigned to a non-zero amount even if the total debt has been paid off.

Thirdly, both the periodPay function and the convert function do not account for the + 1 that is applied to the additionalPaymentsMissed in the applyPenalties function and therefore overwrite this additional missed payment period for mortgages that do not have a payment plan.

Recommendation

Firstly, in the convert function, be sure to account for the fact that mortgages without a payment plan cannot have missed payments before their term is up.

Secondly, account for the additional payment period that is applied to non-payment-plan mortgages in the periodPay and convert functions.

Resolution

Buttonwood Team: The issue was resolved in PR#74.

H-05 | Rounding Prevents Action Cancellations

Category	Severity	Location	Status
Rounding	● High	Global	Resolved

Description PoC

In the cancelWithdrawal function in some cases the number of shares being transferred for the request.amount will round to be one wei greater than the number of shares that the LenderQueue holds for that withdrawal request after the burnExcessShares invocation.

In cases where there is only one withdrawal request in the LenderQueue, the cancellation will revert due to ERC20InsufficientBalance, and in the case where there are multiple withdrawal requests, the cancellation will take 1 wei that should have been allocated to other withdrawal requests, ultimately causing a revert for the last withdrawal being cancelled or even processed.

Recommendation

The rounding of burnExcessShares and the subsequent transfer must be overhauled to avoid sending an additional wei. A simple solution may be to transfer request.amount - 1 to ensure this case does not arise.

Resolution

Buttonwood Team: Resolved.

H-06 | Multiple Origination Pools Broken

Category	Severity	Location	Status
Logical Error	● High	GeneralManager.sol	Resolved

Description

The latest logic in the origination flow sends the available consol token balance of the GeneralManager contract to the immediate Origination pool being processed.

```
IERC20($._consol).safeTransfer(_msgSender(), IConsol($._consol).balanceOf(address(this)));
```

This however leaves an insufficient amount of consol tokens to repay the following origination pools. This is masked by the following logic:

```
uint256 consolBalance = IConsol($._consol).balanceOf(address(this));
if (consolBalance < returnAmount) {
IConsol($._consol).deposit($._usdx, IConsol($._consol).convertUnderlying($._usdx,
returnAmount - consolBalance));
}
```

Which will take from the USDX balance of the contract to repay the following origination pools. This is not surfaced immediately as an error because the fulfiller now receives the balance of the contract instead of the purchaseAmount they should have received.

Recommendation

Instead of transferring the entire GeneralManager balance to the immediate origination pool being processed, transfer the specified returnAmount.

This way there still may be small rounding which affects the fulfiller, but it is not confused with an amount that covers a dearth for other origination pools.

Resolution

Buttonwood Team: The issue was resolved in PR#76.

M-01 | Mortgage Fee Lost On Expiration

Category	Severity	Location	Status
Warning	● Medium	OrderPool.sol	Resolved

Description

In the OrderPool contract the executor of the processOrders function has an adverse incentive to wait until orders are expired before processing them, this is because the order.mortgageGasFee is added to the collectedGasFee in _processOrder when an order has expired.

This means that when an order expires the user loses their mortgageGasFee even though a mortgage was not created, and the executor gets to keep it for themselves.

Recommendation

If an order expires, still collect the order.orderPoolGasFee, however the order.mortgageGasFee should be refunded to the user.

It should be refunded by incrementing a mapping value entry and allowing the user to claim this Ether in a separate transaction to avoid any DoS vectors or gas griefing vectors. Or by wrapping the Ether to wrapped Ether which does not allow for these attacks.

Resolution

Buttonwood Team: The issue was resolved in PR#69.

M-02 | Risk Free Time Difference Arbitrage

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description

When a mortgage creation request is created, the purchaseAmount and collateralAmount are defined based on the price of the asset using the pyth oracle in the block of the creation request.

The creation request is then executed up to 5 minutes later, using these prices from as old as 5 minutes ago.

This allows users to potentially arbitrage price movements which occur in those 5 minute timeframe, by deciding whether or not their order is executed.

For example:
• At time 100, User A makes their expand balance sheet request when Bitcoin is $100,000
• User A configures their creation request to enqueue into a conversion queue that they are already enqueued in
• User A sets their receive function to revert based on a boolean switch, which is currently set to true to prevent the order from being executed
• At time 110, Bitcoin rises to $110,000
• User A sees that Bitcoin price has moved in their favor and flips the boolean switch allowing the receive function to no longer revert and the refund that occurs on the re-enqueue action to proceed
• User A's action is now processed using an outdated price of $100,000 per Bitcoin for their borrow
• If price had not moved in User A's favor, then they would have just left the switch and allowed their order to be cancelled.

Recommendation

Be aware of this time difference arbitrage risk, the OrderPool execution fee may be enough to disincentivize this, furthermore the net interest rate paid on a mortgage will always overshadow what can be arbitraged through this method in the current expiration window.

Resolution

Buttonwood Team: Acknowledged.

M-03 | Collateral Consumed At Stale Trigger Price

Category	Severity	Location	Status
Logical Error	● Medium	ConversionQueue.sol: 117	Acknowledged

Description

When processing withdrawals, collateralToUse is computed by dividing the payment by the stored triggerPrice rather than the current oracle price. If the market price has risen above the trigger price, the system will consume more collateral than necessary for the same repayment.

While the borrower accepted the trigger price for conversion, this should not mean that the borrower converts their collateral at a rate lower than its market value. Additionally, the comment in the code states: 'Figure out how much collateral corresponds to the amountToUse at the current price.'

Recommendation

Consider using the current value of the collateral during conversions, as in the previous version.

However, if this change is an intentional design choice, clearly document this behavior for users, since they will convert their collateral at a lower rate unless the trigger price matches the current value. In that case, also update the comment accordingly.

Resolution

Buttonwood Team: Acknowledged.

M-04 | Converted No Payment Plan Mortgages

Category	Severity	Location	Status
Logical Error	● Medium	MortgageMath.sol	Resolved

Description

In the periodPay function validation is performed on mortgages with no payment plan to ensure that the amount is at least the mortgagePosition.termBalance.

However, for mortgages with no payment plan that have been partially converted, there isn't enough debt to pay off the entire termBalance, since the termConverted is reducing a portion of this debt.

Due to the refund logic this doesn't strictly prevent these mortgages from being paid off, however users must obtain and approve more than their actual debt amount of Consol to pay off their remaining debt in this case.

Recommendation

Update the validation to:

```
if (mortgagePosition.hasPaymentPlan amount < mortgagePosition.termRemaining()) {
revert CannotPartialPrepay(mortgagePosition);
}
```

Resolution

Buttonwood Team: The issue was resolved in PR#77.

M-05 | No Payment Plan Mortgages Arbitraged

Category	Severity	Location	Status
Gaming	● Medium	Global	Acknowledged

Description

Mortgages without payment plans require that their entire debt is paid off in a single `periodPay` invocation.

However, these mortgages in particular, create a large arbitrage opportunity when they pay back their debt. Where the forfeited Consol that pays the interest on the loan is forfeited, increasing the Consol price.

This may be abused by third parties by depositing into USDX, then obtaining Consol tokens right before the large payment and then immediately exiting one of the available queues, likely the USDX queue using their own deposited USDX and the mortgage holders USDX, as long as this queue is not too full.

Furthermore, this may serve as an avenue for the mortgage holder to avoid paying interest. If they are able to flash loan or make a short period loan of a large amount of Consol tokens, they themselves could absorb much of the Consol price increase and recoup their interest payment.

Recommendation

Be aware of this heightened risk of gaming, particularly for mortgages without a payment plan.

Resolution

Buttonwood Team: Acknowledged.

L-01 | Empty Requests Can Be Cancelled

Category	Severity	Location	Status
Validation	● Low	LenderQueue.sol: 156	Resolved

Description

In the `cancelWithdrawal` function there is no validation that prevents an empty, already cancelled, withdrawal from being processed again.

There is no net effect of processing such a withdrawal, however this execution path should be limited to avoid unexpected outcomes and the emission of a misleading `WithdrawalCancelled` event.

Recommendation

Validate that either of the request shares and amount must be non-zero to allow processing to occur.

Resolution

Buttonwood Team: The issue was resolved in PR#68.

L-02 | safeTransferFrom Should Come First

Category	Severity	Location	Status
Best Practices	● Low	MultiTokenVault.sol	Acknowledged

Description

In the MultiTokenVault deposit function the safeTransferFrom action comes after the minting of tokens to the user, therefore the user receives shares before officially paying for them.

To avoid any unexpected re-entrancies with arbitrary supported tokens, the safeTransferFrom should occur before the _mint action so there is no point where the protocol is in an invalid state at the time of an external call.

Recommendation

Perform the safeTransferFrom action before the mint action in the deposit function.

Resolution

Buttonwood Team: Acknowledged.

L-03 | More Than 18 Decimals Incompatible

Category	Severity	Location	Status
Validation	● Low	USDX.sol: 72	Acknowledged

Description

In the addSupportedToken function validation has been added to ensure that the scalarNumerator is greater than or equal to the scalarDenominator.

This disallows tokens with greater than 18 decimals from being used with the system, such as yamV2 which has 24 decimals.

Recommendation

Be aware of this incompatibility or consider removing the validation.

Resolution

Buttonwood Team: Acknowledged.

L-04 | Lacking CEI In Burn

Category	Severity	Location	Status
Best Practices	● Low	MultiTokenVault.sol	Resolved

Description

In the USDX burn function the USDX burn occurs for the user at the end of the function after all tokens have been redeemed to the user. However, to follow CEI the burn should occur first.

Recommendation

Invoke `_burn` after initial validations in the burn function.

Resolution

Buttonwood Team: The issue was resolved in PR#59.

L-05 | Dangerous Withdrawal Request Amounts

Category	Severity	Location	Status
Validation	● Low	LenderQueue.sol	Acknowledged

Description

In the requestWithdrawal function if the amount is a few wei the action may produce a withdrawalRequest that holds a non-zero amount value but a zero shares value.

This creates an unexpected case in the queues that attempt to process such withdrawals.

Furthermore, if the amount specified is zero, this allows users to create withdrawals with both zero-amount value and zero share value.

Recommendation

Both of these cases are disallowed when the minimumWithdrawalAmount is assigned to a non-trivial value.

Ensure that the minimumWithdrawalAmount is always assigned to a minimum reasonable amount to ensure no game-ability and rounding issues are present in all queues.

Resolution

Buttonwood Team: Acknowledged.

L-06 | Lacking Pausability Guards

Category	Severity	Location	Status
Validation	● Low	Global	Resolved

Description

The `ConversionQueue` has a `whenNotPaused` guard which applies to the `processWithdrawalRequests` function, however the `USDXQueue` and `ForfeitedAssetsQueue` do not implement such a validation.

Recommendation

Consider if the `USDXQueue` and `ForfeitedAssetsQueue` should use a `whenNotPaused` modifier for their `processWithdrawalRequests` functions.

Resolution

Buttonwood Team: The issue was resolved in PR#67.

L-07 | Missing Reentrancy Guard

Category	Severity	Location	Status
Validation	● Low	OrderPool.sol	Resolved

Description

In the OrderPool the processOrders function is the only function which has a nonReentrant modifier which provides limited protection against reentrancy into the processOrders function.

The sendOrder function still poses a risk of unexpected reentrancy and therefore should also use the nonReentrant modifier so that it cannot be unexpectedly entered into during a processOrder execution.

Recommendation

Add a nonReentrant guard to the sendOrder function.

Resolution

Buttonwood Team: The issue was resolved in PR#61.

L-08 | Blacklist Prevents Order Processing

Category	Severity	Location	Status
Warning	● Low	OrderPool.sol	Acknowledged

Description

In the case that an order is expired, the collected collateral and USDX are returned to the user and the executor gets to collect the associated gas fee.

However, if the user happens to be blacklisted for the collateral token, the `_processOrder` function will not be able to execute and process their order as it attempts to send tokens to a blacklisted address.

Recommendation

Be aware of this in the event that any supported collateral tokens implement a blacklist. Consider using a mapping to track balances that a user may claim in a separate transaction rather than pushing assets to the user.

Resolution

Buttonwood Team: Acknowledged.

L-09 | Lacking Processing CEI

Category	Severity	Location	Status
Warning	● Low	OrderPool.sol	Resolved

Description

In the _processOrder function the _orders mapping is cleared at the end of the function, however, to follow CEI it would be best if this mapping entry is cleared at the beginning of the function.

Recommendation

Clear the _orders mapping at the beginning of the _processOrder function directly after caching the order in memory.

Resolution

Buttonwood Team: The issue was resolved in PR#62.

L-10 | Lacking Length Validation

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol: 867	Resolved

Description

When creating a mortgage request the `collateralAmounts` length must match the `originationPools` length, however this is not validated in the `requestMortgageCreation` function.

Recommendation

Add validation so that the `collateralAmounts` length is required to match the `originationPools` length.

Resolution

Buttonwood Team: The issue was resolved in PR#66.

L-11 | Outdated Gas Fee Validation

Category	Severity	Location	Status
Validation	● Low	OrderPool.sol: 133	Acknowledged

Description

In the OrderPool contract sendOrder function the validation performed on the msg.value is still against the gasFee, however the gasFee only represents the order pool gas fee and does not include the mortgage gas fee that should be provided.

Recommendation

Update the msg.value validation in the sendOrder function so that it includes the _calculateMortgageGasFee result.

Resolution

Buttonwood Team: Acknowledged.

L-12 | Typo

Category	Severity	Location	Status
Typo	● Low	GeneralManager.sol	Resolved

Description

A comment in the `originationPoolDeployCallback` function states:

`// Send in the collateral to the LoanManager before creating the origination pool`

However, this should read:

`// Send in the collateral to the LoanManager before creating the mortgage`

Recommendation

Correct the comment.

Resolution

Buttonwood Team: The issue was resolved in PR#63.

L-13 | Dangerous Conversion Queue Refund

Category	Severity	Location	Status
Best Practices	● Low	ConversionQueue.sol	Resolved

Description

When enqueuing a mortgage in the ConversionQueue, if the mortgage was previously enqueued it is first removed from the queue to be re-enqueued. In this process the original gas fee is returned to the user, however this allows the user to execute arbitrary action in their receive function.

This may cause unexpected issues due to the user being able to DoS, gas grief the executor, or re-enter into the system unexpectedly.

Recommendation

Instead of sending Ether directly to the user, consider incrementing a mapping value where they can claim their owed Ether in a separate transaction. Otherwise wrap the Ether and send it to the user as to avoid triggering untrusted code.

Resolution

Buttonwood Team: The issue was resolved in PR#69.

L-14 | Lacking Individual Borrow Validation

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol	Acknowledged

Description

In the _validateBorrowCaps function there are some baseline validations that the borrowed amounts are within a set range. However, there is no validation that the individual borrows amounts in each originationParameters.borrowAmounts list entry are within an expected range.

Therefore, while the total may be inside the expected range, the amount requested from each origination pool may be unexpected.

Recommendation

Consider validating that each origination pool requested amount is above some minimum expected amount, or at least above 0.

Resolution

Buttonwood Team: Acknowledged.

L-15 | Duplicate Origination Pools

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol	Resolved

Description

Now multiple origination pools are allowed to be used for an origination of a mortgage. However, there is no validation to ensure that each entry of the `originationParameters.originationPools` list is unique upon creation of the request.

Recommendation

Consider validating that each entry is unique when a mortgage request is being created.

Resolution

Buttonwood Team: The issue was resolved in PR#71.

L-16 | Lacking Reentrancy Guards

Category	Severity	Location	Status
Validation	● Low	GeneralManager.sol	Resolved

Description

Throughout the `GeneralManager` there are important functions which could be re-entered into which are lacking reentrancy guards.

These functions include:

- `requestMortgageCreation`
- `requestBalanceSheetExpansion`
- `originate`
- `enqueueMortgage`
- `convert`

Recommendation

Add the `nonReentrant` modifier to these functions.

Resolution

Buttonwood Team: The issue was resolved in PR#65.

L-17 | Unused subConsol Variable

Category	Severity	Location	Status
Superfluous Code	● Low	ConversionQueue.sol: 44	Resolved

Description

In the ConversionQueue contract the subConsol storage variable is unused.

Recommendation

Consider removing it.

Resolution

Buttonwood Team: The issue was resolved in PR#64.

L-18 | Incorrect NatSpec Comment

Category	Severity	Location	Status
Informational	● Low	QueueProcessor.sol: 11	Resolved

Description

```
/**
* @title QueueProcessor
* @author SocksNFlops
* @notice The StaticInterestRateOracle contract is a contract that returns a static interest
rate for new Mortgages being originated. //@audit-issue L incorrect Natspec comment
*/
```

The @notice comment in QueueProcessor contract belongs to StaticInterestRateOracle.

Recommendation

Update the comment.

Resolution

Buttonwood Team: The issue was resolved in PR#57.

L-19 | Redundant Check In convert

Category	Severity	Location	Status
Superfluous Code	● Low	MortgageMath.sol: 673	Acknowledged

Description

The convert function in the MortgageMath library checks that the current price is greater than the conversionTriggerPrice.

```
if (mortgagePosition.conversionTriggerPrice() > currentPrice) {
revert ConversionTriggerPriceNotMet(mortgagePosition, currentPrice);
}
```

However, the same check is performed again at line 673, making it redundant.

Recommendation

Remove the redundant check.

Resolution

Buttonwood Team: Acknowledged.

L-20 | Redundant Values

Category	Severity	Location	Status
Logical Error	● Low	GeneralManager.sol: 88-89	Acknowledged

Description

When a mortgage is created with a conversion queue or enqueued later, the _conversionQueues and _mortgageEnqueued mappings are populated. However, these mappings are never cleared, even though the mortgage is eventually fully converted.

A fully converted mortgage is removed from its respective ConversionQueue during the _popMortgage call. However, it still remains in other ConversionQueues and must be popped from these queues during withdrawal request processing.

Additionally, it remains in the GeneralManager mappings as if it were still in the queue.

Recommendation

Consider updating or clearing the _conversionQueues and _mortgageEnqueued mappings in GeneralManager when a mortgage is fully converted.

However, care must be taken to ensure that this does not cause a DoS when the mortgage still needs to be popped from other ConversionQueues during loop iterations.

Resolution

Buttonwood Team: Acknowledged.

L-21 | Forfeited Assets Queue Held Up

Category	Severity	Location	Status
Warning	● Low	ForfeitedAssetsQueue.sol	Acknowledged

Description

The ForfeitedAssetsQueue can be held up by large withdrawal requests which cannot be fully processed because the Consol holdings have not acquired enough ForfeitedAssetsPool tokens to withdraw.

This can reasonably occur especially for foreclosures where the remaining debt of the position is small at the time of foreclosure, in which case it would require many such foreclosures to occur to allow the forfeited assets queue to fill a significant withdrawal in the queue.

Recommendation

In future iterations consider allowing partial withdrawals. Otherwise, be aware of this blocking behavior and communicate it with users.

Resolution

Buttonwood Team: Acknowledged.

L-22 | Missing requestWithdrawal Refund

Category	Severity	Location	Status
Warning	● Low	LenderQueue.sol	Acknowledged

Description

The requestWithdrawal function does not implement a refund mechanism of too much ether is sent, this does not match other functions in the GeneralManager that do perform refunds for native ETH.

Recommendation

Consider introducing a refund for the requestWithdrawal function in future iterations.

Resolution

Buttonwood Team: Acknowledged.

L-23 | Incorrect Event Emission

Category	Severity	Location	Status
Logical Error	● Low	USDX.sol: 171	Resolved

Description

In the `burn` function the Withdraw event for the final withdrawal is intended to have the remaining `amount - totalBurned`, however this case is entered for all withdrawals that come before the final withdrawal, because the main if case is `i = supportedTokens.length() - 1`, meaning that the last withdrawal goes into the first if branch.

Recommendation

Update the if case to use the condition `i = supportedTokens.length() - 1`.

Resolution

Buttonwood Team: The issue was resolved in PR#77.

L-24 | Count Unexpectedly Incremented Twice

Category	Severity	Location	Status
Unexpected Behavior	● Low	ConversionQueue.sol	Resolved

Description

In the ConversionQueue, the count variable which is meant to track iterations of the processing loop can unexpectedly be incremented twice in a single loop iteration when the amountToUse matches both the request.amount and the mortgagePosition.principalRemaining().

This may be unexpected for the caller of the processWithdrawalRequests and may result in less withdrawal requests being processed than expected.

Recommendation

Be aware of this edge case and inform callers of the processWithdrawalRequests function.

Resolution

Buttonwood Team: The issue was resolved in PR#77.

Disclaimer

About Guardian

Founded in 2022 by DeFi experts, Guardian is a leading audit firm in the DeFi smart contract space. With every audit report, Guardian upholds best-in-class security while achieving our mission to relentlessly secure DeFi.

To learn more, visit https://guardianaudits.com

To view our audit portfolio, visit https://github.com/guardianaudits

To book an audit, message https://t.me/guardianaudits