



Security Assessment

Hourglass Protocol - HFXB

Audit Summary Report

Nov 22, 2023

Table of contents

Introduction to Sublime Group	4
Technical Security Assessment and Advisory Services	4
Smart Contract Auditing	4
Decentralized Finance (DeFi)	5
Quantitative Trading	5
Market Making	5
Audit Test and Reporting Disclaimer	6
Audit Summary	7
Findings Summary	10
List of Issues Found	12
ISSUE-1 finalizeDeposit wrong calculation of conversion rate	12
ISSUE-2 Cancel withdrawal functionality issues	13
ISSUE-3 Deposits roll not working	15
ISSUE-4 settleDeposits and cancelDeposit logic error	16
ISSUE-5 Mistake in settleWithdrawals bufferFee calculation	17
ISSUE-6 shareConversionRateAt buffer fee calculation	19
ISSUE-7 HFXBWithdrawalQueue - lacks init owner setting	20
ISSUE-8 trancheToConversionRate bad validation	21
ISSUE-9 finalizeWithdrawal wrong accounting of buffer fee	22
ISSUE-10 Mistake in NotEligibleForRedemption check	23
ISSUE-11 updateConversionRate wrong condition check	24
ISSUE-12 finalize withdrawal for another user	25
ISSUE-13 Deposits pause not effective after unpause	26
ISSUE-14 Unpausing of deposits can cause a wrong state	27
ISSUE-15 Possibility to mint 0 shares	28
ISSUE-16 finalizeWithdrawal no min amount validation	29
ISSUE-17 sfraxToFinalize should be > 0 in settle withdrawals	30
ISSUE-18 Following checks, effects & interactions	31
ISSUE-19 finalizeWithdrawals reentrancy vectors	32
ISSUE-20 Protection against reentrancy vectors	33
ISSUE-21 AccessControlUpgradeable bypass	34
Test coverage	35

Static analysis	36
Slither analysis	37
Integration with Other Protocols	38
Enhancing Interoperability:	38
Risk Assessment:	38
Ongoing Risk Management:	38

Introduction to Sublime Group

Sublime Group is a leading organization revolutionizing the financial landscape through its expertise in Decentralized Finance (DeFi), quantitative trading, market making, and technical security assessment. With a commitment to innovation and trust, Sublime Group offers cutting-edge solutions and advisory services in the rapidly evolving digital asset ecosystem.

Technical Security Assessment and Advisory Services

Sublime Group excels in providing cutting-edge technical security assessment and advisory services. Leveraging advanced tools and methodologies, we conduct comprehensive security assessments, penetration testing, and vulnerability analysis to fortify systems and infrastructure. Our team of highly skilled cybersecurity experts possesses extensive expertise in identifying and mitigating potential risks and ensuring compliance with regulatory frameworks. By partnering with Sublime Group, clients benefit from our industry-leading security solutions, enabling them to safeguard their digital assets and maintain a robust security posture that surpasses competitors.

Smart Contract Auditing

At Sublime Group, we pride ourselves on delivering meticulous smart contract audits that go beyond industry standards. Our experienced auditors combine their deep understanding of blockchain technology with an arsenal of cutting-edge tools and advanced methodologies. In addition to manual code analysis, our auditors utilize static code analysis tools to perform automated checks, machine learning techniques to identify complex vulnerabilities, and comprehensive fuzz testing to detect potential security loopholes. By employing these advanced methods, we thoroughly review the reliability, functionality, and security aspects of smart contract code. Our auditors provide actionable recommendations to address identified vulnerabilities, ensuring the robustness and integrity of our clients' blockchain-based applications. Sublime Group's smart contract auditing process, powered by state-of-the-art LLM and fuzzing tools, puts Sublime ahead of the industry standard. By choosing Sublime Group, clients benefit from our unrivaled expertise and innovative approach, guaranteeing the utmost security for their smart contracts and staying ahead of the curve in the ever-evolving landscape of software security.

Decentralized Finance (DeFi)

Sublime Group leverages decentralized protocols and smart contracts to provide seamless access to decentralized lending, borrowing, yield farming, and decentralized exchanges. Our solutions bridge traditional finance with the blockchain ecosystem, empowering users in the DeFi space.

Quantitative Trading

Sublime Group's skilled quantitative traders optimize trading strategies using advanced algorithms and data analysis techniques. Our expertise in market dynamics, liquidity, and risk management enables efficient execution and enhanced trading performance across digital asset markets.

Market Making

Sublime Group ensures market liquidity, reduces spreads, and minimizes price volatility through our market-making services. Our proprietary algorithms and risk management systems contribute to fair and efficient price discovery, benefiting institutional and retail investors.

Audit Test and Reporting Disclaimer

Sublime Group conducted activities for this project according to the statement of work. However, it is important to note that security assessments have time constraints and rely on client-provided information. Thus, the findings in this report may not encompass all security issues or flaws in the system or codebase.

Sublime Group employs automated testing techniques and manual security reviews to assess software controls. However, automated tools have limitations, such as not capturing all edge cases or incomplete analysis within time limits. These limitations are subject to project time and resource constraints.

Clients should understand that while Sublime Group's test coverage is comprehensive within the project's scope, it may not uncover all potential vulnerabilities or flaws. The audit scope does not cover code provided by third party libraries or protocols that Hourglass Protocol integrates with. However, we have made due diligence when conducting our test to check for any known vulnerabilities and correct usage of said libraries. Ongoing security assessments and proactive measures are advised to maintain system integrity. Sublime Group remains dedicated to assisting clients in enhancing their security and providing expert guidance throughout the development process.

Audit Summary

Hourglass Protocol engaged Sublime Group to review the security of its contracts. Our dedicated team has invested 3 person-week of effort into conducting a security review of the client-provided source code. We have conducted a thorough audit of the smart contracts code provided by Hourglass Protocol. After completing the initial audit Hourglass approached Sublime with a request to review minor updates to the smart contract which took an additional 4 person-days of work and was included in this updated report.

Hourglass Protocol approached Sublime Group at an early stage of development of HFXB smart contracts. This was an unusual situation that gave us the ability to spot potential problems and suggest best design practices at an early development stage. Concurrently, the audit posed more of a challenge than usual, given the extensive modifications the codebase underwent throughout the process. The audit commenced with revision **3bcf5c363865547039e08e325a5d68bff1635a64** on the main branch, which served as the starting point for our evaluation.

Throughout the audit process, our team has diligently reviewed the HFXB smart contracts, carefully scrutinizing the codebase for potential security vulnerabilities, logical flaws, and adherence to best practices. Our objective was to assess the robustness and integrity of the contracts, identifying any potential issues that could compromise the security and functionality of the system.

As part of the audit work, we engaged in a collaborative effort with Hourglass Protocol, ensuring a smooth and effective communication channel between our team and theirs. This allowed for the timely resolution of any issues or concerns that arose during the audit. Hourglass Protocol actively participated in addressing the identified issues, promptly implementing fixes and enhancements based on our recommendations.

The audit process involved a comprehensive analysis of the codebase, including but not limited to the smart contracts and associated libraries or dependencies. Our experienced auditors meticulously reviewed the code, examining key aspects such as contract architecture, data handling, access controls, input validation, and adherence to industry best practices.

During the audit, we uncovered multiple issues, which were promptly communicated to the Hourglass Protocol team. Due to the early stage at which the audit commenced the number of issues found is greater than in a typical report. Some of the items relate to functionality that was either removed or reworked in subsequent commits, however we have chosen to keep them in the report for the sake of transparency and completeness (refer to issues list for details). We have provided detailed reports outlining the identified vulnerabilities and flaws, along with recommended remediation measures. Hourglass Protocol demonstrated a strong commitment to security and swiftly addressed the identified issues by implementing the recommended fixes. This collaborative approach between our team and Hourglass Protocol ensured a robust and secure smart contract implementation.

It is important to note that the audit conducted on HFXB contracts was focused solely on the code provided by Hourglass Protocol. While we diligently assessed the code for potential vulnerabilities, logical flaws, and adherence to best practices, it is essential to acknowledge that unforeseen risks or vulnerabilities may exist beyond the scope of the audit.

Hourglass Protocol's proactive engagement during the audit process, including addressing the identified issues, reflects their dedication to ensuring the security and reliability of their smart contracts. By actively participating in the audit and promptly resolving any identified issues, Hourglass Protocol showcases a commitment to delivering a secure and trustworthy platform for their users.

The contracts were updated by Hourglass Protocol multiple times during the audit. The list of all revisions used during the audit (in chronological order) can be found below:

1. `3bcf5c363865547039e08e325a5d68bff1635a64` - initial revision
2. `87af826a46e43fdda9e94ee3f946de23854bd467` - iter 1
3. `1849263939e1af8dc6b3403a0dcd7a999f6625a3` - iter 2
4. `6cc96de36cad98260bab8e6158029e783073f7fd` - iter 3
5. `86e86ad81db622791bd88005a87ffea33d8a789c` - candidate 1
6. `700a241455f07c7472fbde93eed55bd40c9660e5` - candidate 2
7. `E1c8140bf6ffefaa1d86bce5b90d135ece0cf482` - candidate 3
8. `E9156ed43a7f3f3db6b3308edaecc1772f1eb816` - update iter 1
9. `5242160f9008b2a48ffe9b9b944186dae8fbb744` - update iter 2
10. `F1d0324f85b7a8685b87524b7df7a8490db57ed6` - update iter 3

Below is a list of smart contracts included in the audit scope along with their respective sha256 checksums at revision **F1d0324f85b7a8685b87524b7df7a8490db57ed6**:

Filename	SHA-256
HFXB.sol	345b62aacdd02e4d9991aa860a3869e1e0534c1148acc605b29edd0cefc4b50d
HFXBWithdrawalQueue.sol	4cceba09cd8b8be1c0d28f49af3c83a678b057ad04f15f915ee625d959e15388
TwoStepOwnable.sol	592a1a798946ec89c01c38949ed419aa13e4c914da6f7810d9dc90a9916097d1

Findings Summary

Our primary focus during the audit was to identify potential vulnerabilities in the Hourglass Protocol HFXB smart contract, specifically related to business logic, arithmetic operations, and integration with other protocols and tokens. Our objective was to ensure that the protocol remains secure, preventing any unauthorized access or fund theft. Additionally, we have provided Hourglass Protocol with guidance on clean code practices and industry best practices wherever applicable.

Throughout the audit process, we diligently analyzed the smart contracts codebase, resulting in the identification of several issues with varying severity levels. We provided Hourglass Protocol with comprehensive guidance and recommendations on security enhancements, best practices, and maintaining clean code standards. Collaboratively, we worked with Hourglass Protocol to address and rectify all the issues discovered during the audit.

Due to the early stage at which the audit commenced, we have identified a large number of issues and recommendations. Some of the items relate to functionality that was either removed or reworked in subsequent commits, however, we have chosen to keep them in the report for the sake of transparency and completeness (refer to issues list for details).

Thanks to the commitment from Hourglass Team the smart contracts have undergone multiple changes and fixes. After multiple iterations we are happy to say that all the identified issues have been fixed and all the recommendations were addressed by Hourglass Team. Their dedication to securing the protocol is commendable.

The test coverage of the protocol was determined to be good. A detailed section in the report outlines the test coverage of HFXB.

Below is a summary of all the issues found during the audit divided into respective severity categories ranging from critical to recommendation.

disclaimer : please note that the audit was commenced at an early development stage and due to this fact the issues found are related to multiple code iterations, including functionalities that were either removed or simplified in the process. We choose to include all the items for the sake of transparency and completeness.

Severity	Issues found	Remaining after Audit
● Critical	4	0
● High	6	0
● Medium	4	0
● Low	2	0
● Recommendation	5	0

List of Issues Found

ISSUE-1 | finalizeDeposit wrong calculation of conversion rate

Severity	Revision found	Status
critical	700a241455f07c7472fbde93eed55bd40c9660e5	Resolved

Description

The sharesToMint calculation in the finalize deposit function contains a critical mistake. The amount of deposited sFRAX is multiplied instead of being divided by the conversion rate factor.

This mistake can lead to users being able to deposit funds and after the deposit is settled immediately request a withdrawal to gain profit. At the same time it's worth noting that due to the nature of the implementation the risk is limited because processing of deposits and withdrawals requires an action to be taken by the protocol and only funds related to current deposits and withdrawals are stored in the contract.

```
uint256 sharesToMint = (request[msg.sender][tranche] * trancheShareConversionRates[tranche]) / PRECISION;
```

Recommendation

Settle withdrawals, finalize deposit and finalize withdrawal should all be adjusted to use trancheShareConversionRates in a consistent manner.

ISSUE-2 | Cancel withdrawal functionality issues

Severity	Revision found	Status
● Critical	3bcf5c363865547039e08e325a5d68bff1635a64	Resolved

Description

We have discovered that the cancel withdrawal function introduces a lot of complexity into the contract logic and opens up some attack vectors that can endanger the contract funds.

Due to the fact that an incrementing **requestIndex** (used for identifying the withdrawal request) is not adjusted upon a cancellation of a withdrawal request a dangerous attack vector is created with the following steps:

1. 3 withdrawal requests (A, B and C) are made
2. A and B requests are canceled, state variables denoting pending withdrawals are updated.
3. Now the request index of request C is higher than expected as it was not adjusted
4. This means that **NotEligibleForRedemption** and **WithdrawalFinalized** calculations are wrong for this request
5. User finalizes the request to receive sFRAX
6. User can still wrongly cancel the request to receive back the shares, protocol is at a loss.

At the same time if the withdrawal request is canceled after the tranche counter is advanced (by settling the deposits) the `sharesWithdrawnPerTranche` variable will be updated for the wrong tranche.

Recommendation

Due to the implementation of the withdrawal & deposits tranche logic, allowing the users to cancel withdrawal requests introduces very complex relationships that need to be handled.

During a discussion with the Hourglass team it was uncovered that the implementation of the withdrawal queue aims to mimic `erc1155` functionality.

Together with the Hourglass Team it was decided that removing the cancel withdrawal function would be the best approach to mitigate the issues and simplify the contract logic. At the same time properly implementing the erc1155 standard will greatly increase the functionality of the HFXB contract as well as simplify the implementation and mitigate any potential issues.

ISSUE-3 | Deposits roll not working

Severity	Revision found	Status
critical	f1d0324f85b7a8685b87524b7df7a8490db57ed6	Resolved

Description

In the **managePausedDeposit()** function in case of rolling the deposit, the old deposit info is removed but the **addressToTrancheToSFraxDepositRequest** is not actually set for the new tranche. This means that the deposit is effectively canceled without sending the funds back to the user or rolling into a new tranche.

```
addressToTrancheToSFraxDepositRequest[msg.sender][depositTranche] = 0;
pausedTrancheAmounts[depositTranche] -= amount;
if (rollDeposit) {
    if (depositsPaused) revert DepositsPaused();
    totalDepositedSFraxPending += amount;
    emit DepositRolled(msg.sender, depositTranche, currentTranche, amount);
}
```

Recommendation

This needs to be fixed and the **addressToTrancheToSFraxDepositRequest** needs to be properly set with the deposit amount for the new tranche.

ISSUE-4 | settleDeposits and cancelDeposit logic error

Severity	Revision found	Status
● Critical	3bcf5c363865547039e08e325a5d68bff1635a64	Resolved

Description

In the settleDeposits function there is a check `if (sfraxDeposited != 0) {` This is a mistake and sfraxDeposited should be changed to the sfraxPendingDeposit variable. SfraxDeposited is not set at this line and the function will revert which is not intended.

At the same time the `cancelDeposit` function is not properly adjusting the `sfraxPendingDeposit` variable. This will cause a mismatch when settleDeposits is called

Recommendation

The check in settleDeposits should be updated. The cancel deposit function should adjust the sfraxPendingDeposit variable.

ISSUE-5 | Mistake in settleWithdrawals bufferFee calculation

Severity	Revision found	Status
● high	700a241455f07c7472fbde93eed55bd40c9660e5	Resolved

Description

When settleWithdrawals is called the BufferFee is only taken into account in case of partial tranche, but it's always subtracted from the amount when finalizing a withdrawal. This means that over time more sFRAX will be pulled into the contract than necessary to finalize all the withdrawals.

At the same time in case of partial tranche the shares calculation subtracts the buffer fee denominated in sFRAX from a variable denominated in shares. Depending on the conversion rate this can lead to either too much or too little sFRAX being pulled into the contract to settle the withdrawals.

Additionally the buffer fee should increase the amount of shares that can be settled for the same amount of sFRAX instead of decreasing it - it should be added instead of subtracted.

Furthermore the calculation of the precise amount of shares to settle per amount of sFRAX supplied might produce a different result due to round-off issues. In case of settle withdrawals the buffer fee and conversion rate calculations are done per tranche so the rounding error will be of a different magnitude than when actually finalizing the withdrawals one by one.

```
if (partialTranche) {  
    // if partial, calculate how much sfrax is available for this tranche  
    sfraxForTranche = sfraxToFinalize - sfraxDisbursed;  
  
    // calculate the shares to finalize for this tranche, deducting the buffer fee  
    uint256 trancheShares = ((sfraxForTranche * PRECISION) / conversionRate) - ((sfraxForTranche *  
bufferFee) / DENOMINATOR);  
}
```

Recommendation

We recommend removing the buffer fee adjustment from the settleWithdrawals calculation. Properly taking into account the fee and all the precision errors would be a costly and complicated calculation. Instead of this we recommend to create a new counter of all the accrued fees during withdrawal finalization and a new admin function allowing to transfer cumulated fees back to the fund.

ISSUE-6 | shareConversionRateAt buffer fee calculation

Severity	Revision found	Status
● High	6cc96de36cad98260bab8e6158029e783073f7fd	Resolved

Description

The `calculateShareConversionRateAt` function used during withdrawal calculation mixes a buffer fee denominated in shares with `amountPayable` denominated in sFRAX.

```
amountPayable = (  
    (userSharesFinalized * trancheShareConversionRates[tranche]) / 1e18  
    ) - ((userSharesFinalized * bufferFee) / DENOMINATOR);  
(...)
```

Recommendation

We recommend fixing this calculation and converting the buffer fee from shares to sFRAX.

ISSUE-7 | HFXBWithdrawalQueue - lacks init owner setting

Severity	Revision found	Status
● High	1849263939e1af8dc6b3403a0dcd7a999f6625a3	Resolved

Description

HFXBWithdrawalQueue - lacks init owner setting for TwoStepOwnable in the constructor.

Recommendation

Add missing initialization to the contract constructor.

ISSUE-8 | trancheToConversionRate bad validation

Severity	Revision found	Status
● High	e9156ed43a7f3f3db6b3308edaecc1772f1eb816	Resolved

Description

calculateSFraxForSharesAtTranche() function contains a validation enforcing that **trancheToConversionRate[tranche] > DENOMINATOR**, whereas other places including the setter of this state variable only check that it's greater than zero. Due to other changes in the code this check is no longer valid. Furthermore in an extreme case it can break withdrawal finalization for a given tranche if it was set to a value lower than the denominator. As there is no way to reset the conversion rate after it is set the withdrawals will be permanently blocked and fixing this state will require an update of the smart contract.

Recommendation

We recommend adjusting the validation to be consistent with all the other functions and check only if **trancheToConversionRate[tranche] > 0**.

ISSUE-9 | finalizeWithdrawal wrong accounting of buffer fee

Severity	Revision found	Status
● High	e9156ed43a7f3f3db6b3308edaecc1772f1eb816	Resolved

Description

The **bufferFee** is not subtracted from the amount transferred out to the end user in **finalizeWithdrawal**. At the same time **_afterWithdraw** internal function is called with an sfrax amount that includes the **bufferFee** twice.

Recommendation

The amount transferred out to the user should be decreased by the **bufferFee**. At the same time **afterWithdraw** internal function should be called with an sfrax amount equal to what was actually transferred to the user plus the buffer fee to properly update the state variables such as the amount of sfrax claimable and already claimed.

ISSUE-10 | Mistake in NotEligibleForRedemption check

Severity	Revision found	Status
● High	3bcf5c363865547039e08e325a5d68bff1635a64	Resolved

Description

There is a mistake in the NotEligibleForRedemption check. It is supposed to check if the request was already settled and is ready to be finalized however the check has a mistake.

```
if (requestIndex + shares < totalSharesRedeemed + redeemableShares) {  
    revert NotEligibleForRedemption();  
}
```

Recommendation

The condition should be changed to:

```
if (requestIndex + shares > totalSharesRedeemed + redeemableShares) {  
    revert NotEligibleForRedemption();  
}
```

ISSUE-11 | updateConversionRate wrong condition check

Severity	Revision found	Status
● medium	86e86ad81db622791bd88005a87ffea33d8a789c	Resolved

Description

The updateConversionRate function checks that the current tranche conversion rate is not set as one of the preconditions. This condition could never be satisfied and thus the function will always revert.

```
function updateConversionRate(uint256 conversionRate) external onlyOwner {
    // cannot have pending deposits when using this method
    if (sfraxPendingDeposit > 0) {
        revert MustSettleDeposits();
    }

    // conversion rate should be in wei, not bps (full precision)
    if (conversionRate <= DENOMINATOR) {
        revert InvalidConversionRate();
    }

    uint256 trancheBeingSettled = queueTranche;

    if (
        trancheReadyForProcessing
        ||
        trancheShareConversionRates[trancheBeingSettled] == 0
    ) {
        revert PriorConversionRateNotSet();
    }
    (...)
}
```

Recommendation

After a discussion with the team we have confirmed that the intended use of this function is to update the conversion rate and store it in the array for checkpointing at the same time incrementing the tranche counter.

The check for prior conversion rate not being set is not necessary and can be safely removed to achieve the intended purpose of the function.

ISSUE-12 | finalize withdrawal for another user

Severity	Revision found	Status
● medium	6cc96de36cad98260bab8e6158029e783073f7fd	Resolved

Description

The contract allowed us to call the settle withdrawals function and supply a user address parameter. This in turn opened the possibility for anyone to finalize a withdrawal request of another user. While the funds were transferred to the correct account we questioned this behavior. There does not seem to be a clear benefit of allowing the user to be != msg.sender. At the same time further modifications to the contract can put the funds at risk if developers are not careful and aware of this relationship.

During the discussion of this item with the Hourglass Team it was further uncovered that this could have further implications in case the erc1155 withdrawal representation is used in a secondary market e.g. as a collateral. Anyone being able to finalize the withdrawal belonging to the lending market could put its funds at risk.

```
/// @notice Once the Operator has set the redemption rate for the tranche, the user can finalize their withdrawal  
  
/// @param user The user address that has an existing pending withdrawal request  
  
/// @param requestIndex The tranche the user is requesting to withdraw from  
  
function finalizeWithdraw(address user, uint256 requestIndex) external withdrawalsNotPaused {  
    (...)  
}
```

Recommendation

After a discussion with the Hourglass Team it was decided that this parameter will be removed in favor of using msg.sender.

ISSUE-13 | Deposits pause not effective after unpause

Severity	Revision found	Status
● medium	5242160f9008b2a48ffe9b9b944186dae8fbb744	Resolved

Description

DepositsPausedAtTranche state variable is reset when unpause. This means that after this happens all previous deposits can be finalized at the old **ConversionRate** before they are settled by **settleDeposits**. This leads to an unexpected state as deposits are designed to be finalized after **settleDeposits**. At the same time there is no way to re-pause previous deposits after unpause and they can be instantly finalized at the historical rate.

Recommendation

Upon discussing this issue with Hourglass Team we have agreed that the best solution would be to disallow users from finalizing deposits after they are paused. Instead users should only be able to cancel paused deposits or roll them to a new tranche when unpause (with a new **ConversionRate**).

ISSUE-14 | Unpausing of deposits can cause a wrong state

Severity	Revision found	Status
● medium	f1d0324f85b7a8685b87524b7df7a8490db57ed6	Resolved


Description

When deposits are paused and unpaused before the tranche is incremented a User can make a new deposit to the same previously-paused tranche. In such cases both pending deposits and paused deposits can be >0 for that tranche. This state is not properly handled by the contract functions. It can lead to many unforeseen problems such as the User not being able to finalize his new deposit due to the DepositTranchePaused error. At the same time calling pause twice (or calling pause, unpause & pause again) before tranche is advanced will break accounting of deposits because it overrides **pausedTrancheAmounts**.

Recommendation

After a discussion the Hourglass team decided that the best approach would be to increment the tranche by forcing a call to stepTranche whenever the deposits are paused. Thanks to this after unpausing any new deposits will be made to the next tranche.

ISSUE-15 | Possibility to mint 0 shares

Severity	Revision found	Status
 low	6cc96de36cad98260bab8e6158029e783073f7fd	Resolved

Description


Currently there is no protection against minting 0 shares or withdrawing 0 sFRAX depending on the set exchange rate. It's possible to deposit not enough sFRAX to get 0 shares or to withdraw not enough shares to get 0 sFRAX back. This is caused by a division by the conversion rate being rounded down. If e.g. the deposit amount is lower than the conversion rate the amount of minted shares will be 0.

This is not trivial to solve due to the fact that the conversion rate is not known at the time of making a deposit or withdrawal request.

Recommendation

After a discussion with the Hourglass Team it was decided that the best solution will be to supply a minimum withdrawal and deposit amount check. A default amount of 1e18 was chosen for this value and can be further changed by the contract admin if necessary.

ISSUE-16 | finalizeWithdrawal no min amount validation

Severity	Revision found	Status
 low	5242160f9008b2a48ffe9b9b944186dae8fbb744	Resolved

Description

inalizeWithdrawal allows for as little as 1 wei of shares to be finalized, can lead to 0 sfrax transferred to the user if **ConversionRate** > 1. At the same time this can lead to buffer fee being calculated as 0. Although the buffer fee can be calculated as 0 for small amounts it is not a critical issue because due to gas fees and rounding errors it is not a viable strategy to avoid paying the fees.

Recommendation

We recommend validating the min amount to finalize.

Resolution

During the resolution of this issue another problem was uncovered. In case the user chooses to finalize less than 100% of his withdrawal in some cases a certain amount lower than the min amount can be left unfinalized. This means that after introducing the validation this small amount will be permanently blocked from finalization.

Hourglass decided to solve this issue by introducing an additional function allowing the end user to finalize 100% of the remaining shares.

ISSUE-17 | sfraxToFinalize should be > 0 in settle withdrawals

Severity	Revision found	Status
● recommendation	86e86ad81db622791bd88005a87ffea3 3d8a789c	Resolved

Description

Currently settle withdrawals can be called with 0 sFRAX supplied. While it doesn't pose any threat to protocol security it does not have any effect and thus the transaction would be a waste of gas.

Recommendation

Add an error check into the function to make sure it cannot be called with 0 sFRAX supplied.

ISSUE-18 | Following checks, effects & interactions

Severity	Revision found	Status
● Recommendation	1849263939e1af8dc6b3403a0dcd7a999f6625a3	Resolved

Description

Transfer can be made the last call in the `prepareTrancheForSettlement` and `settleWithdrawals` function in order to follow the CEI pattern. Although the function is only callable by a privileged User and the account used in transfer is also a privileged user, it's a good practice to follow this pattern whenever possible. Further changes to the contract or dependent contracts and libraries can open up new and unforeseen attack vectors.

Recommendation

We recommend changing the order of execution in order to follow the checks, effects & interactions pattern.

ISSUE-19 | finalizeWithdrawals reentrancy vectors

Severity	Revision found	Status
● Recommendation	6cc96de36cad98260bab8e6158029e783073f7fd	Resolved

Description

In the current implementation of `finalizeWithdrawals` there are no state changing or external contract calls before `burnBatchQueuePositions` is called. However you could consider increasing security and reduce the possibility of future updates introducing new attack vectors by refactoring the code to make `burnBatchQueuePositions` the first call in the function or applying non-reentrant locks.

Recommendation

We recommend adding the additional mitigations for reentrancy attacks into this function.

Hourglass team has acknowledged this and in further iterations and after multiple refactors this function was deprecated and removed from the contract due to its high complexity and low usefulness.

ISSUE-20 | Protection against reentrancy vectors

Severity	Revision found	Status
● Recommendation	6cc96de36cad98260bab8e6158029e783073f7fd	Resolved

Description

After the analysis of the HFXB contract code we have come to the conclusion that because of: following checks effects and interactions pattern, interacting only with trusted contracts and having strict access control checks, the contract is currently protected against reentrancy attacks.

Despite this we feel that introduction of new features to the contract can easily break this state and introduce new attack vectors. Because of this we have approached Hourglass team with the recommendation of applying reentrancy locks on the user facing external state modifying functions.

Recommendation

We recommend applying reentrancy locks on external user facing functions.

The Hourglass team has acknowledged this and applied the additional protections.

ISSUE-21 | AccessControlUpgradeable bypass

Severity	Revision found	Status
● Recommendation	1849263939e1af8dc6b3403a0dcd7a999f6625a3	Resolved

Description

The contract used the functionality from AccessControlUpgradeable for role based access control, however this functionality was extended and partially bypassed for the custom handling of the OPERATOR role. This concern was already acknowledged by the Hourglass Team.

```
function grantRole(bytes32 role, address account) public override {
    if (role == OPERATOR) {
        revert UseGrantOperatorRole();
    }
    // now send down to AccessControl
    AccessControlUpgradeable.grantRole(role, account);
}

function grantOperatorRole(address newOperator) external onlyRole(DEFAULT_ADMIN_ROLE) {
    if (!hasRole(getRoleAdmin(OPERATOR), msg.sender)) {
        revert CallerNotRoleAdmin();
    }

    if (IERC20(sfrax).allowance(newOperator, address(this)) != type(uint256).max) {
        revert ApprovalNotActive();
    }
    // set the pending operator
    pendingOperator = newOperator;
}
```

Recommendation

After a discussion with the Hourglass Team it was decided that the role based access control is not necessary for this contract. The team has decided to remove it and replace it with a 2 step ownable pattern.

Test coverage

The test coverage of the smart contracts is satisfactory and covers the majority of implemented functionality.

File	% Lines	% Statements	% Branches	% Funcs
src/HFXB.sol	82.52% (170/206)	84.00% (189/225)	60.00% (42/70)	89.66% (26/29)
src/HFXBWithdrawalQueue.sol	89.58% (43/48)	89.83% (53/59)	83.33% (5/6)	70.59% (12/17)
src/Utils/TwoStepOwnable.sol	85.00% (17/20)	86.96% (20/23)	62.50% (5/8)	100.00% (7/7)

Static analysis

During our audit of the HFXB smart contracts, we conducted a thorough static code analysis using a set of our proprietary rules. This analysis aimed to identify any known bugs or recurring attack vectors. The results revealed that all issues detected were false positives, indicating the absence of actual vulnerabilities or bugs. Our automated examination applied specific rules designed to detect common coding mistakes, security vulnerabilities, and attack patterns. We carefully verified each reported issue and determined that they did not pose real security risks. This static analysis provides additional assurance of the codebase's thorough examination and reinforces confidence in the security and reliability of the Hourglass Protocol smart contracts. For more detailed information, please refer to the comprehensive report provided. In conclusion, the static code analysis confirmed the absence of genuine vulnerabilities or bugs, further enhancing the security assessment's credibility.

```
Scanning 24 files with 44 solidity rules. 24/24 tasks 0:00:00

[
  | Results |
]

Findings:

/hfxb/src/HFXBWithdrawalQueue.sol
solidity.deflation-token

138| uint256 burned = _totalAmountBurned[id];
   | -----
139| if ((supply - burned) < amount) {
   | -----
140| revert BurnAmountExceedsSupply();
   | -----
143| _totalAmountBurned[id] = burned + amount;

[
  | Scan Summary |
]

Some files were skipped or only partially analyzed.
Scan was limited to files tracked by git.

Ran 44 rules on 24 files: 7 findings.
```

Slither analysis

Although it is not a key offering of Sublime and its usefulness is limited, we have conducted static analysis of Hourglass Protocol smart contracts using the industry standard Slither tool. Despite its limitations, it can be useful in identifying areas otherwise missed by the auditors.

We have analyzed slithers report and concluded that all of the reported issues are either false positives or only informational. Nevertheless we have provided the team with the analysis extract with selected items that can be optionally taken into consideration

For the purpose of this report we choose to only provide the summary of slither findings below:

```
Total number of contracts in source files: 7
Number of contracts in dependencies: 20
Source lines of code (SLOC) in source files: 582
Source lines of code (SLOC) in dependencies: 1994
Number of assembly lines: 0
Number of optimization issues: 0
Number of informational issues: 78
Number of low issues: 11
Number of medium issues: 6
Number of high issues: 6
ERCs: ERC20, ERC165
```

Name	# functions	ERCs	ERC20 info	Complex code	Features
HFXB	99	ERC20	No Minting Approve Race Cond.	Yes	Receive ETH Delegatecall Tokens interaction
HFXBWithdrawalQueue	88	ERC165		No	Upgradeable Receive ETH Delegatecall Upgradeable
IFXB	3			No	
IQueue	20	ERC165		No	Upgradeable
ISFRAX	1			No	

```
INFO:Slither:/media/rincewind/Files/projects/hfxb/ analyzed (27 contracts)
```

Integration with Other Protocols

In the process of auditing the HFXB contracts, we have identified that the project directly or indirectly interfaces with, or is based upon, a list of existing protocols: prominently the Frax Protocol and their sFRAX offering. The integration of multiple protocols within a project brings both opportunities and potential risks. This summary highlights the significance integrating with other protocols has for protocol security and emphasizes the limitations of the audit scope.

Enhancing Interoperability:

Integrating with well-established protocols enables the project to leverage existing functionalities and tap into a broader ecosystem. This interoperability provides numerous benefits, including enhanced liquidity, access to additional services, and the ability to leverage established communities. By integrating with these protocols, the project positions itself to deliver a more comprehensive and seamless experience for its users. At the same time, each integration brings another layer of complexity into the project and makes it more difficult for the team to ensure project security.

Risk Assessment:

During our audit, we diligently analyzed the Hourglass Protocol contracts for known issues or common mistakes that may arise when interacting with the aforementioned protocols. Our objective was to identify any potential vulnerabilities or risks within the contracts that could impact the integration. However, it is important to note that our audit scope was specifically limited to the Hourglass Protocol contracts and did not encompass a comprehensive verification of the integrated protocols themselves.

Ongoing Risk Management:

To mitigate potential risks associated with protocol integrations, it is imperative for the project to maintain a proactive approach to risk management. This includes staying informed about updates, upgrades, and potential vulnerabilities of the integrated protocols. By actively monitoring the security landscape and engaging in ongoing risk assessments, the project can promptly address any emerging threats or vulnerabilities, safeguarding the interests of its users and the overall integrity of the system.

The integration of the project with other protocols expands its capabilities and potential. While we have analyzed the Hourglass Protocol contracts for issues related to these integrated protocols, it is important to reiterate that our audit scope did not encompass the verification of the integrated protocols themselves. Therefore, it is crucial for project stakeholders to undertake independent audits of the integrated protocols to ensure their security and reliability. By adopting a proactive approach to risk management and ongoing assessments, the project can effectively navigate the complexities of integration and deliver a secure and robust experience for its users.