



Security Assessment

Hourglass Protocol - Hourglass Index

Audit Summary Report

Jan 08, 2024

Table of contents

Introduction to Sublime Group	3
Technical Security Assessment and Advisory Services	3
Smart Contract Auditing	3
Decentralized Finance (DeFi)	4
Quantitative Trading	4
Market Making	4
Audit Test and Reporting Disclaimer	5
Audit Summary	6
Findings Summary	8
List of Issues Found	10
ISSUE-1 HourglassIndexFactory wrong mapping key	10
ISSUE-2 Proxy implementation can be partially initialized	11
ISSUE-3 Finalize withdrawal functionality issues	12
ISSUE-4 Potential issues with opening up to new assets	13
ISSUE-5 IndexCreated event parameters bad naming	14
ISSUE-6 Base contracts Upgradeability	15
Test coverage	16
Static analysis	17

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 generalized version of the HFXB smart contracts. Our dedicated team has invested 3.5 person-days 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. Sublime Group has previously conducted an audit of the original HFXB implementation and as such this audit was focused on the updates to the existing contracts.

The audit commenced with revision **85b4e5155e0859ad824d9d8ba82da689de61405b** on the main branch of the repository, which served as the starting point for our evaluation.

Throughout the audit process, our team has diligently reviewed the Hourglass Index 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. 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 Hourglass Index 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. **85b4e5155e0859ad824d9d8ba82da689de61405b**
2. **86c2fb4a43fd7e8e1be4e5698910464abe927f72**
3. **642e40bef670becc024af772088e30811394a99b**

Below is a list of smart contracts included in the audit scope along with their respective sha256 checksums at revision **642e40bef670becc024af772088e30811394a99b**:

Filename	SHA-256
HourglassIndex.sol	ecb12e7902d59d05125acd7c20d8257fb39ab4372221b52b912243d22471fc4e
HourglassIndexWithdrawalQueueue.sol	6d35b2c14ae86b8ebb035da9cba12b396f09ad36217b7d7aef7ff11c5a3fc859
TwoStepOwnable.sol	d5e9c97c50e8e43dbd1a4468370a7ad87d5e0253469747b0ee7ddc428900630
HourglassIndexFactory.sol	72e5c53b8eb295e69ffe9a38a87d753ce22970f4912228a126e76c32b1dfade1

Findings Summary

Our primary focus during the audit was to identify potential vulnerabilities in the Hourglass Protocol HourglassIndex and HourglassIndexFactory smart contract, specifically related to business logic, arithmetic operations, proxy pattern implementation, 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 have provided Hourglass Protocol with comprehensive guidance and recommendations on security enhancements, best practices, and maintaining clean code standards. Collaboratively, we have worked with Hourglass Protocol to address and rectify all the issues discovered during the audit.

During the audit we have focused on verifying the beacon proxy pattern introduced by the Hourglass team in the Hourglass Index project and general adherence to best practices when it comes to proxy implementation. We have also analyzed the security implications of generalizing the HFXB smart contracts and opening the protocol to other underlying assets.

We are happy to say that all the issues identified during the audit 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 HourglassIndex smart contracts.

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

Severity	Issues found	Remaining after Audit
● Critical	1	0
● High	0	0
● Medium	3	0
● Low	1	0
● Recommendation	1	0

List of Issues Found

ISSUE-1 | HourglassIndexFactory wrong mapping key

Severity	Revision found	Status
● Critical	85b4e5155e0859ad824d9d8ba82da689de61405b	Resolved

Description

In the **HourglassIndexFactory** **deployIndex** function there was a mistake in the key used to store a new **IndexDeployment** in the indexes mapping.

```
indexIndex = indexes[_versionIndex].length;  
// store the deployments by version index mapping key at the end of the indexes array  
indexes[indexIndex].push(  
  IndexDeployment({indexProxy: indexDeployment, withdrawalQueueProxy:  
    queueDeployment})  
);
```

Instead of using **_versionIndex** as the key, **indexIndex** was used. This is a mistake as **indexIndex** represents the position of the new deployment in the **indexes[_versionIndex]** array.

The final position of the new deployment in the contract storage should in fact be **indexes[_versionIndex][indexIndex]**.

Recommendation

The **indexIndex** should be replaced with **_versionIndex** in line 3 of the above code snippet.

ISSUE-2 | Proxy implementation can be partially initialized

Severity	Revision found	Status
● Medium	85b4e5155e0859ad824d9d8ba82da689de61405b	Resolved

Description

There is a potential issue with the **setDeployment** function of the **HourglassIndex** and **HourglassIndexWithdrawalQueue** contract. In its current state it allows for anyone to call this function directly on the implementation contract and initialize the implementation storage with arbitrary values..

In the current state of the implementation this should not pose any direct problems, as other preconditions required to exploit this flaw are not present inside of the smart contracts. However, allowing anyone to do any type of initialization on the implementation contract of a proxy pattern is not a good practice and should be avoided.

Recommendation

We have advised the Hourglass team to consider blocking a call to this function inside of the implementations constructor similar to how it is done with `_disableInitializers`.

At the same time we advise the team to make sure that the contracts in question are created solely by the **HourglassIndexFactory**. Since **setDeployment** functions are not permissioned they have to be called atomically when creating a new contract instance

ISSUE-3 | Finalize withdrawal functionality issues

Severity	Revision found	Status
● Medium	85b4e5155e0859ad824d9d8ba82da689de61405b	Resolved

Description

The generalized HourglassIndex.sol introduced a mistake compared to the original HFXB.sol. In the **finalizeWithdrawal** function the **totalWithdrawnAssetsClaimable** amount is compared against the **amountDepositAssetIgnoringFee**. However in case of the batch version (**_finalizeWithdrawals**) it's compared against **depositAssetWithdrawn** + **bufferFeesIncurred**.

```
// calculate the shares of _depositAsset the user will receive
(uint256 amountDepositAssetsPayable, uint256 bufferFeeIncurred) =
    calculateDepositAssetForSharesAtTranche(shares, requestTranche);

// sanity check since contract may hold _depositAsset not allocated for redemptions
if (amountDepositAssetsPayable > totalWithdrawnAssetsClaimable) {
    revert InsufficientBalanceForWithdrawal();
}
```

In other words: the amount of claimable asset is compared to the amount being withdrawn. However in one case this comparison takes into account the fee paid by the user, but ignores it in the other. The code suggests that inclusion of the fee in this comparison was the intended version, since the **totalWithdrawnAssetsClaimable** contains both the amount requested by the user and the incurred fee.

Recommendation

This looks like a simple oversight when refactoring the HFXB.sol contract. We recommend revisiting the two functions to confirm the intention behind this code fragment and making sure that both use the same logic.

ISSUE-4 | Potential issues with opening up to new assets

Severity	Revision found	Status
● Medium	85b4e5155e0859ad824d9d8ba82da689de61405b	Resolved

Description

Generalizing the contract and opening up to new underlying tokens can have some unexpected consequences.


One such problem could be if the underlying will be a tax on transfer token. In such case the deposit logic can become broken as currently it relies on the assumption that the actual transferred amount is equal to the amount provided in the transfer or transferFrom function parameter.

Recommendation

Two things come to mind as a solution to this issue. One is to enforce a strict screening of newly added tokens to make sure they do not have any non-standard logic inside of their transfer function. This is a good practice even when ignoring the issue above.

Another possible improvement would be to enforce a check inside of the smart contract to validate that the actual transfer amount is equal to what was expected.

ISSUE-5 | IndexCreated event parameters bad naming

Severity	Revision found	Status
 low	85b4e5155e0859ad824d9d8ba82da689de61405b	Resolved

Description

It looks like the parameters of the IndexCreated event are badly named in contrast to the values supplied when emitting the event.

The property names that are most likely wrong are **indexIndex** - should actually be versionIndex and **indexIndexIndex** - should actually be indexIndex.

Recommendation

We recommend double checking the event parameter names and making sure they are aligned with the values supplied when emitting said events.

ISSUE-6 | Base contracts Upgradeability

Severity	Revision found	Status
● Recommendation	85b4e5155e0859ad824d9d8ba82da689de61405b	Resolved

Description

The order of **TwoStepOwnable** in the inheritance chain of **HourglassIndex** and **HourglassIndexWithdrawalQueue** means that if there is any need to upgrade the contract's functionality it could be difficult or impossible to do so due to storage conflicts.

Recommendation

We recommend analyzing the possibility of such implementation upgrades and potentially adding a storage gap similar to OpenZeppelin Upgradable contracts in **TwoStepOwnable** contract to ensure that it can be upgraded if really needed. A good practice would also be to make **TwoStepOwnable** last in the inheritance chain after the OpenZeppelin base contracts.

Test coverage

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

```
Running tests...
```

File	% Lines	% Statements	% Branches	% Funcs
HourglassIndexFactory.sol	50.00% (18/36)	51.11% (23/45)	25.00% (2/8)	50.00% (3/6)
HourglassIndex.sol	87.94% (175/199)	88.58% (194/219)	69.12% (47/68)	92.59% (25/27)
HourglassIndexWithdrawalQueue.sol	91.49% (43/47)	90.00% (54/60)	87.50% (7/8)	81.25% (13/16)
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 Hourglass Index 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.

```

Scan Status
Scanning 9 files (only git-tracked) with 44 Code rules:

CODE RULES
Scanning 9 files with 44 solidity rules.

9 Code Findings

/src/factory/queues/HourglassIndexWithdrawalQueue.sol
solidity.deflation-token

124| // amount to burn
125| -----
128| // total supply burned for this token id
129| -----
129| uint256 burned = _totalAmountBurned[id];
130| -----
130| // minted - burned should be less than amount
131| (...)

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

Ran 44 rules on 9 files: 9 findings.
```