Security Assessment

Hourglass Protocol - Locking TBT

Audit Summary Report

March 20, 2024

Table of contents

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 Locking TBT smart contracts. Our dedicated team has invested 4 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.

The audit commenced with revision **99212106f16bee8d11359c132448f01bcfd19a7f** on the **permissioned-creation** branch of the repository, which served as the starting point for our evaluation.

Throughout the audit process, our team has diligently reviewed the Hourglass TBT 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 have uncovered several 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 TBT 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. `99212106f16bee8d11359c132448f01bcfd19a7f`
2. `B58838466f7ce643996e42564c0d29eeaae6bdef`
3. `17b9a0196a5b78d292579afc5dbba63463a2b5f9`

Below is a list of smart contracts included in the audit scope along with their respective sha256 checksums at revision `17b9a0196a5b78d292579afc5dbba63463a2b5f9`:

Filename	SHA-256
HourglassERC20TBT.sol	3479f8f9e94319b59915a2cebbe7d3c18338ac7ca66a809d7aa1cdf632ec5422
HourglassTBTFactory.sol	459dea1776ae7b9ccc5a155d1606baecd5df23600035938cf35c97c28b06dda7
HourglassLockDepositor.sol	26aad11e88b1c7e9660ac3ebedd251032418d3313ad2f82ffd76540494eef9ea
IHourglassDepositor.sol	7c5c0f41a6a89e6aee52b4ab4dcbcadf4c34a4357cee47300f1c199e6e45f8f9

IHourglassERC20TBT.sol	5dc353471592927e82d07f10001633c1b6adc3f8575242e84393a5432b116509
IHourglassLockingTBTFactory.sol	83d6403bff77e8f037eea091f60f552612eb32ea58900b01064edf0564dca769

Findings Summary

Our primary focus during the audit was to identify potential vulnerabilities in the HourglassERC20TBT, HourglassLockDepositor and HourglassLockingTBTFactory smart contracts, 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 implementation in HourglassLockingTBTFactory and HourglassLockDepositor contracts, verifying that the process deploying new versions of the contracts and new Maturities instances is secure. We have also investigated the security and general adherence to best practices of the HourglassERC20TBT token implementation.

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	1	0
Low	1	0
Recommendation	5	1

List of Issues Found

ISSUE-1 | transferFromWithPermit bad signature validation

Severity	Revision found	Status
Critical	99212106f16bee8d11359c132448f01bcfd19a7f	Resolved

Description

In the **HourglassERC20TBT transferFromWithPermit** function there is a mistake in the way the signature of the transfer permit is validated**.** The intention of the function is to allow the user to validate the permit signature and do a transfer in a single atomic transaction (bypassing setting the approval in a separate permit call).

```
function transferFromWithPermit(
   address from,
   address spender,
   address to,
   uint256 value,
   uint256 deadline,
   bytes calldata signature
) external {
   _permit(from, spender, value, deadline, signature);
   _transfer(from, to, value);
}
```

The main issue with this implementation is that the spender is not validated to be the msg.sender, at the same time the “to” address is not part of the signature. This allows for a potential attacker to frontrun an existing transaction and use a valid signature to send funds to any arbitrary address.

Recommendation

The function should explicitly validate that msg.sender == spender or as an alternative remove the spender parameter and force it to be always equal to msg.sender.

ISSUE-2 | Usage of SafeERC20

Severity	Revision found	Status
Medium	b58838466f7ce643996e42564c0d29eeaae6bdef	Resolved

Description

Some tokens might not revert upon failed transfer and instead only return a boolean. Consider using OZ SafeERC20 for transfers inside of HourglassLockDepositor. This can be especially problematic by breaking the deposit function.

Recommendation

As most token implementations revert under such conditions this fix is not necessary if the team can guarantee no interactions with such tokens. At the same time special care should be made to screen the tokens the protocol interacts with

As another solution - SafeERC20 can be used or additional validation of the transferred amount can be enforced

Resolution

The hourglass team has decided to implement the SafeERC20 for transfers inside of their smart contracts.

ISSUE-3 | LockingTBTFactory - recoverTokens permission

Severity	Revision found	Status
Low	99212106f16bee8d11359c132448f01bcfd19a7f	Resolved

Description

Together with the Hourglass team we have uncovered that there is a possibility of granting the **DEPLOYER** permission to a 3rd party in the future.

Because of this the function **recoverTokens** inside of the **HourglassLockingTBTFactory**, permissioned for the **DEPLOYER** role, should be changed to require a higher privilege role such as **REGISTRY_MANAGER**.

Recommendation

Consider changing the permission required for the **recoverTokens** function to a higher privilege role such as the **REGISTRY_MANAGER**.

ISSUE-4 | Receipt implementation addresses validation

Severity	Revision found	Status
Recommendation	99212106f16bee8d11359c132448f01bcfd19a7f	Resolved by introducing the whitelist

Description

When creating a new Maturity inside of **HourglassTBTFactory createMaturity** function, the receipt implementation addresses come from the calldata. This means that they are just cloned and are not versioned or validated in any way. So in case a 3rd party is granted the deployer permission a high degree of trust must exist for that 3rd party.

The cloned token contracts are also marked as **isSystemAddress** which right now grants them only the permission to clone contacts. Consider if this **isSystemAddress** permission will be used in more cases in the future.

This could be a problem in case a malicious actor is granted the **DEPLOYER** role or in case such a private key is compromised.

Recommendation

Consider Introducing a separate whitelist of receipt token implementation contracts and make that whitelist editable only by the manager or other high-privilege role. Additionally consider introducing other means of security of admin keys like multisig to reduce centralisation and risk of key becoming compromised..

ISSUE-5 | Initialization sanity check

Severity	Revision found	Status
Recommendation	b58838466f7ce643996e42564c0d29eeaae6bdef	Resolved by adding the empty data validation.

Description

Consider double checking that **HourglassLockingTBTFactory createMaturity** did indeed initialize the depositor contract. This is very unlikely but it is possible to create **depositorData** which will succeed newDepositor.call() but not initialize the contract. This is extremely unlikely to happen by mistake and would have to most likely be done on purpose by the **DEPOSITOR.** Because of this we leave this just as a recommendation. Further changes to the Depositor contract such as adding a fallback function might allow for empty data to also not revert the version creation transaction as it only validates the call success.

Recommendation

Consider adding this sanity check. As an alternative consider at least validating that data used to initialize the depositor contract is not empty as this is the most likely scenario that might happen by accident.

ISSUE-6 | setDepositCap sanity check

Severity	Revision found	Status
Recommendation	b58838466f7ce643996e42564c0d29eeaae6bdef	Resolved

Description

Consider validating that arrays passed as the parameters of **setDepositCap** function have the same length as a sanity check. This could help detect a configuration mistake that would potentially lead to wrong Caps being set for wrong depositors.

Recommendation

Consider adding this sanity check.

ISSUE-7 | Additional zero address checks

Severity	Revision found	Status
Recommendation	b58838466f7ce643996e42564c0d29eeaae6bdef	Resolved

Description

Consider adding an explicit zero address check for the **upgradeVersion** function of **HourglassLockingTBTFactory**. Supplying a zero address will revert downstream, however other functions of the contract already have this explicit check and it might provide a more user friendly error message. Consider adding a zero address check also for the **grantReceiptImplementation** function.

At the same time to avoid potential zero address transfers consider adding a zero address check to **rewardsDistributor recoverToken** function.

Recommendation

Consider adding the extra checks.

ISSUE-8 | Dependencies revisions

Severity	Revision found	Status
Recommendation	b58838466f7ce643996e42564c0d29eeaae6bdef	Acknowledged by the Team

Description

Some of the dependencies of the projects are set to a revision not corresponding to an actual release version of the imported library. This does not seem to introduce any errors into the codebase at this time as there is no significant difference between the revisions, however we recommend using the finalized revisions of each library.

Recommendation

Consider setting the git submodules to a revision corresponding to an actual release version of the dependency.

Security advisory

During the the audit we have created a list of items that might affect the robustness and ongoing security of the system and should be brought to the attention of the Hourglass Team:

Please note that EIP-1271 signed transfer could be susceptible to DOS attacks in certain conditions. If any contract or other part of the system triggers such transfers (e.g. in a batch transaction) a malicious impl of **signer.isValidSignature** by the signing contract could be used to gas grief or simply revert the tx. This does not affect the protocol in its current version but should be taken into account in future development.

The Hourglass team should also take note of the ERC-3009 Security Considerations regarding the usage of **transferWithAuthorization** and **receiveWithAuthorization.** Please note that **receiveWithAuthorization** should be used instead of **transferWithAuthorization** when calling from other smart contracts. It is possible for an attacker watching the transaction pool to extract the transfer authorization and front-run the transferWithAuthorization call to execute the transfer without invoking the wrapper function. This could potentially result in unprocessed, locked up deposits. Currently neither of the functions is used inside of the smart contracts

With regard to ERC20 permits usage. Previous exploits existed when a protocol mistakenly used ERC20 permits on a token without permit implementation but with a fallback function. The calling contract called the permit function which did not revert because of the fallback function despite the empty signature and proceeded with execution assuming the signature was verified properly. The calling contract should make sure that the token being used is the correct one and that it does implement the permit function properly. As with the other items this is just an advisory as currently the permit function is not used in such a way throughout the protocol.

The protocol owners should also be advised to analyze for potential reentrancy, read only reentrancy and cross function reentrancy attack vectors in case of extending the current Depositor contract implementation. In case of adding interactions with external protocols current checks effects and interactions pattern usage could prove no longer adequate to protect against such threats and additional non-reentrant locks should be applied.

Additional caution should also be applied when adding support for new tokens. New tokens should be verified for non-standard behavior such as on-transfer callbacks, fee on transfer, rebasing or reflection. The protocol should be additionally analyzed for the impact of supporting such token types.

And finally, the EIP-3009 proposal used in the **HourglassERC20TBT** token contract is marked as stagnant in https://eips.ethereum.org/EIPS/eip-3009. Despite this fact, a number of implementations already exist and it should not impact the security of the system, however we think that it is important to know the current status of the EIP.

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       |
|-----------------------------------------------|----------------|----------------|---------------|---------------|
| src/HourglassTBTFactory.sol                   | 95.83% (46/48) | 88.89% (64/72)| 70.83% (17/24)| 92.31% (12/13)|
| src/depositors/HourglassLockDepositor.sol     | 100.00% (49/49)| 100.00% (57/57)| 90.00% (9/10)| 100.00% (12/12)|
| src/recipts/HourglassERC20TBT.sol             | 100.00% (33/33)| 97.56% (40/41)| 95.00% (19/20)| 100.00% (14/14)|
```

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. Our automated examination applied specific rules designed to detect common coding mistakes, security vulnerabilities, and attack patterns. The analysis did not find any vulnerabilities in the codebase giving an additional assurance of the codebase's thorough examination and reinforcing confidence in the security and reliability of the Hourglass Protocol smart contracts.