CERTIK

Astra - audit

Security Assessment

CertiK Assessed on Jan 3rd, 2025

CertiK Assessed on Jan 3rd, 2025

Astra - audit

The security assessment was prepared by CertiK, the leader in Web3.0 security.

Executive Summary

TYPES	ECOSYSTEM	METHODS
Bridge	EVM Compatible	Formal Verification, Manual Review, Static Analysis

LANGUAGE	TIMELINE	KEY COMPONENTS
Solidity	Delivered on 01/03/2025	N/A

CODEBASE

Astra Github Repo

View All in Codebase Page

COMMITS

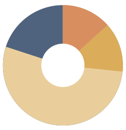
- 550ee8a559cc50d1c642bfd84d4280d7d64fe1bd
- d3c911e076323978dd6dbbfcd2e3c58538b26c95
- b34e2769b4546b3c6c98f000e125afde3eee0ffb

View All in Codebase Page

Highlighted Centralization Risks

Contract upgradeability

Vulnerability Summary



Total Findings	Resolved	Mitigated	Partially Resolved	Acknowledged	Declined
15	8	0	0	7	0

0	Critical		Critical risks are those that impact the safe functioning of a platform and must be addressed before launch. Users should not invest in any project with outstanding critical risks.
2	Major	2 Acknowledged	Major risks can include centralization issues and logical errors. Under specific circumstances, these major risks can lead to loss of funds and/or control of the project.
2	Medium	1 Resolved, 1 Acknowledged	Medium risks may not pose a direct risk to users' funds, but they can affect the overall functioning of a platform.
8	Minor	6 Resolved, 2 Acknowledged	Minor risks can be any of the above, but on a smaller scale. They generally do not compromise the overall integrity of the project, but they may be less efficient than other solutions.
3	Informational	1 Resolved, 2 Acknowledged	Informational errors are often recommendations to improve the style of the code or certain operations to fall within industry best practices. They usually do not affect the overall functioning of the code.

TABLE OF CONTENTS | ASTRA - AUDIT

CODEBASE | ASTRA - AUDIT

Repository

Astra Github Repo

Commit

- 550ee8a559cc50d1c642bfd84d4280d7d64fe1bd
- d3c911e076323978dd6dbbfcd2e3c58538b26c95
- b34e2769b4546b3c6c98f000e125afde3eee0ffb
- e4f898fa8a5bc72daf50dacebf67e3f43c960069

AUDIT SCOPE | ASTRA - AUDIT

1 file audited • 1 file without findings

ID	Repo	File	SHA256 Checksum
HTL	Infi-network/astraContract	HashedTimeLock.sol	72ba759ecd3654fd07cd49712c40c0d355956edfa460c04f34a041242f4d67a8

APPROACH & METHODS | ASTRA - AUDIT

This report has been prepared for Astra to discover issues and vulnerabilities in the source code of the Astra - audit project as well as any contract dependencies that were not part of an officially recognized library. A comprehensive examination has been performed, utilizing Formal Verification, Manual Review, and Static Analysis techniques.

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.

The security assessment resulted in findings that ranged from critical to informational. We recommend addressing these findings to ensure a high level of security standards and industry practices. We suggest recommendations that could better serve the project from the security perspective:

- Testing the smart contracts against both common and uncommon attack vectors;
- Enhance general coding practices for better structures of source codes;
- Add enough unit tests to cover the possible use cases;
- Provide more comments per each function for readability, especially contracts that are verified in public;
- Provide more transparency on privileged activities once the protocol is live.

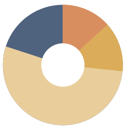
REVIEW NOTES | ASTRA - AUDIT

Out-of-Scope Components

The LSP off-chain program, responsible for authorizing asset transfers from the Lightning Network to the EVM chain, is not included in the current audit scope. The audit team assumes that its code has been securely implemented.

The off-chain program responsible for purchasing preimages (used for withdrawing funds on EVM) by paying Lightning Invoices is not within the current audit scope. The audit team assumes that its code has been implemented securely.

FINDINGS | ASTRA - AUDIT



15	0	2	2	8	3
Total Findings	Critical	Major	Medium	Minor	Informational

This report has been prepared to discover issues and vulnerabilities for Astra - audit. Through this audit, we have uncovered 15 issues ranging from different severity levels. Utilizing the techniques of Formal Verification, Manual Review & Static Analysis to complement rigorous manual code reviews, we discovered the following findings:

ID	Title	Category	Severity	Status
GLOBAL-01	**Centralization Risks In HashedTimeLock.Sol**	**Centralization**	**Major**	**Acknowledged**
HTL-01	**Centralized Control Of Contract Upgrade**	**Centralization**	**Major**	**Acknowledged**
HTL-02	`initialize()` Is Unprotected	Logical Issue	Medium	Acknowledged
HTL-03	Potential Unauthorized Token Transfers In `deposit` Function	Volatile Code	Medium	Resolved
HTC-01	Unclaimed Rewards From Aave During Asset Transfer In `deposit` And `withdraw`	Volatile Code	Minor	Resolved
HTL-05	Incompatibility With Deflationary Tokens	Volatile Code	Minor	Acknowledged
HTL-06	Usage Of `transfer()` For Sending Native Tokens	Coding Style	Minor	Resolved
HTL-07	Third-Party Dependency Usage	Design Issue	Minor	Resolved
HTL-08	Potential Cross-Chain Replay Attack	Logical Issue	Minor	Resolved
HTL-09	Lack Of Maximum Length Validation For LSP Name In `registerLsp()` Allows Invalid Long LSP Name	Volatile Code	Minor	Resolved

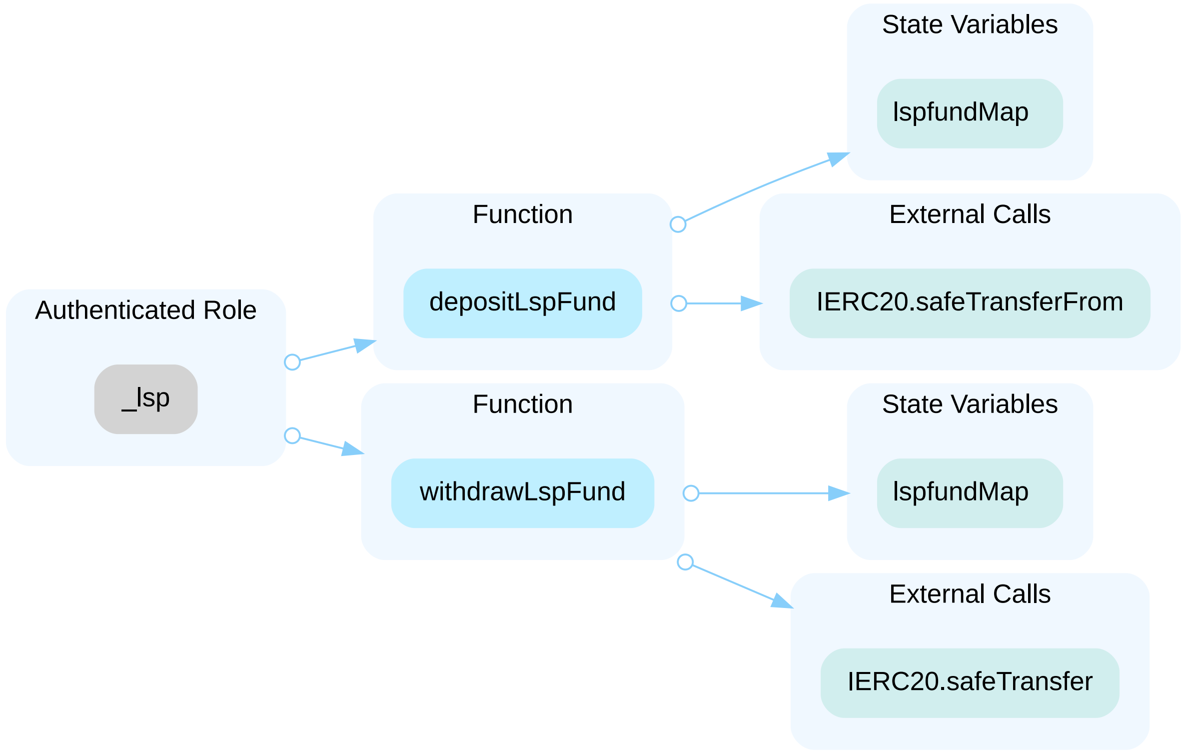
ID	Title	Category	Severity	Status
HTL-12	Missing TimeLock Validation In `withdraw()` Allows Asset Withdrawal After The Lock Time Expires	Logical Issue	Minor	Acknowledged
HTL-13	Missing Fee Refund In `deposit()` Function Locks ETH Fee Leftover In Contract	Logical Issue	Minor	Resolved
HTL-10	Unlocked Compiler Version	Language Version	Informational	Acknowledged
HTL-11	Unused Variable	Design Issue	Informational	Resolved
HTL-14	Inconsistency Between Code And Comment	Logical Issue	Informational	Acknowledged

GLOBAL-01 | CENTRALIZATION RISKS IN HASHEDTIMELOCK.SOL

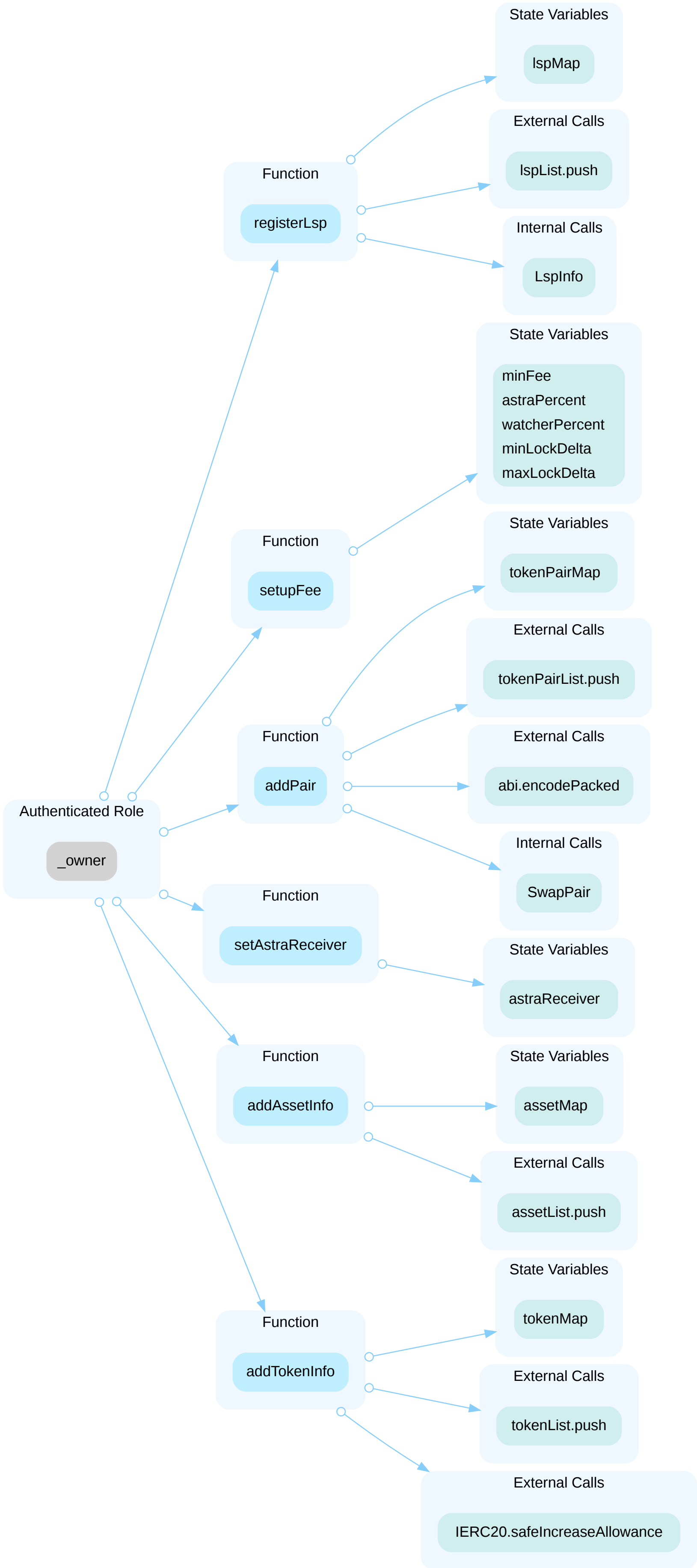
Category	Severity	Location	Status
Centralization	**Major**		**Acknowledged**

Description

In the contract `HashedTimeLock`, the role `_lsp` has authority over the functions shown in the diagram below. Any compromise to the `_lsp` account may allow the hacker to take advantage of this authority and deposit LSP fund, withdraw specified token amount from LSP fund.



In the contract `HashedTimeLock`, the role `_owner` has authority over the functions shown in the diagram below. Any compromise to the `_owner` account may allow the hacker to take advantage of this authority and register a new LSP with validations, set up fee parameters with constraints, add a new token-asset pair, set the astra receiver address, and add asset and token information to the mapping and list.


State Variables
lspMap
External Calls
lspList.push
Function
registerLsp
Internal Calls
LspInfo
State Variables
minFee
astraPercent
watcherPercent
minLockDelta
maxLockDelta
Function
setupFee
State Variables
tokenPairMap
External Calls
tokenPairList.push
Function
addPair
External Calls
abi.encodePacked
Authenticated Role
_owner
Internal Calls
SwapPair
Function
setAstraReceiver
State Variables
astraReceiver
Function
addAssetInfo
State Variables
assetMap
External Calls
assetList.push
State Variables
tokenMap
Function
addTokenInfo
External Calls
tokenList.push
External Calls
IERC20.safeIncreaseAllowance

Recommendation

The risk describes the current project design and potentially makes iterations to improve in the security operation and level of decentralization, which in most cases cannot be resolved entirely at the present stage. We advise the client to carefully manage the privileged account's private key to avoid any potential risks of being hacked. In general, we strongly recommend centralized privileges or roles in the protocol be improved via a decentralized mechanism or smart-contract-based accounts with enhanced security practices, e.g., multisignature wallets. Indicatively, here are some feasible suggestions that would also mitigate the potential risk at a different level in terms of short-term, long-term and permanent:

Short Term:

Timelock and Multi sign (⅔, ⅗) combination *mitigate* by delaying the sensitive operation and avoiding a single point of key management failure.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to the private key compromised;
  AND
- A medium/blog link for sharing the timelock contract and multi-signers addresses information with the public audience.

Long Term:

Timelock and DAO, the combination, *mitigate* by applying decentralization and transparency.

- Time-lock with reasonable latency, e.g., 48 hours, for awareness on privileged operations;
  AND
- Introduction of a DAO/governance/voting module to increase transparency and user involvement.
  AND
- A medium/blog link for sharing the timelock contract, multi-signers addresses, and DAO information with the public audience.

Permanent:

Renouncing the ownership or removing the function can be considered *fully resolved*.

- Renounce the ownership and never claim back the privileged roles.
  OR
- Remove the risky functionality.

Alleviation

[Astra Team, 12/25/2024]: we will implement a TimeLockController and multi-sig mechanism. Additionally, the addresses for the TimeLockController and multi-sig will be displayed in Astra's GitBook.

[CertiK, 12/25/2024]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

HTL-01 | CENTRALIZED CONTROL OF CONTRACT UPGRADE

Category	Severity	Location	Status
Centralization	**Major**	**HashedTimeLock.sol (commit:550ee8): 27**	**Acknowledged**

Description

In the contract `HashedTimeLock`, the role `admin` has the authority to update the implementation contract behind the contract.

Any compromise to the `admin` account may allow a hacker to take advantage of this authority and change the implementation contract which is pointed by proxy and therefore execute potential malicious functionality in the implementation contract.

Recommendation

We recommend that the team make efforts to restrict access to the admin of the proxy contract. A strategy of combining a time-lock and a multi-signature (⅔, ⅗) wallet can be used to prevent a single point of failure due to a private key compromise. In addition, the team should be transparent and notify the community in advance whenever they plan to migrate to a new implementation contract.

Here are some feasible short-term and long-term suggestions that would mitigate the potential risk to a different level and suggestions that would permanently fully resolve the risk.

Short Term:

A combination of a time-lock and a multi signature (⅔, ⅗) wallet mitigate the risk by delaying the sensitive operation and avoiding a single point of key management failure.

- A time-lock with reasonable latency, such as 48 hours, for awareness of privileged operations;
  AND
- Assignment of privileged roles to multi-signature wallets to prevent a single point of failure due to a private key compromised;
  AND
- A medium/blog link for sharing the time-lock contract and multi-signers addresses information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.

- Provide a link to the **medium/blog** with all of the above information included.

Long Term:

A combination of a time-lock on the contract upgrade operation and a DAO for controlling the upgrade operation mitigate the contract upgrade risk by applying transparency and decentralization.

- A time-lock with reasonable latency, such as 48 hours, for community awareness of privileged operations;
  AND
- Introduction of a DAO, governance, or voting module to increase decentralization, transparency, and user involvement;
  AND
- A medium/blog link for sharing the time-lock contract, multi-signers addresses, and DAO information with the community.

For remediation and mitigated status, please provide the following information:

- Provide the deployed time-lock address.
- Provide the **gnosis** address with **ALL** the multi-signer addresses for the verification process.
- Provide a link to the **medium/blog** with all of the above information included.

Permanent:

Renouncing ownership of the `admin` account or removing the upgrade functionality can *fully* resolve the risk.

- Renounce the ownership and never claim back the privileged role;
  OR
- Remove the risky functionality.

Note: we recommend the project team consider the long-term solution or the permanent solution. The project team shall make a decision based on the current state of their project, timeline, and project resources.

Alleviation

[Astra Team, 12/25/2024]: we will implement a TimeLockController and multi-sig mechanism. Additionally, the addresses for the TimeLockController and multi-sig will be displayed in Astra's GitBook.

[CertiK, 12/25/2024]: It is suggested to implement the aforementioned methods to avoid centralized failure. Also, CertiK strongly encourages the project team to periodically revisit the private key security management of all addresses related to centralized roles.

HTL-02 | `initialize()` IS UNPROTECTED

Category	Severity	Location	Status
Logical Issue	● Medium	HashedTimeLock.sol (commit:550ee8): 27	● Acknowledged

Description

The `HashedTimeLock` logic contract does not protect the initializer. An attacker can front-run the `initialize` call and assume ownership of the logic contract. Once in control, the attacker can perform privileged operations, misleading users into believing that they are interacting with the legitimate owner of the upgradeable contract.

Recommendation

We recommend adding

```
/// @custom:oz-upgrades-unsafe-allow constructor
constructor() initializer {}
```

The addition will prevent the function `initialize()` from being called directly in the implementation contract, but the proxy will still be able to `initialize()` its storage variables.

Alleviation

[Astra Team, 12/25/2024]: Manual verification will be performed during the contract deployment phase to avoid the mentioned issues.

HTL-03 | POTENTIAL UNAUTHORIZED TOKEN TRANSFERS IN `deposit` FUNCTION

Category	Severity	Location	Status
Volatile Code	● Medium	HashedTimeLock.sol (commit:550ee8): 345	● Resolved

Description

The `deposit()` function in the contract has a significant logic flaw. While it allows users to deposit assets, the function fails to properly validate the parameters when the `_toLightning` flag is set to `true`. In this case, other critical parameters, such as the `_from` and `_to`, can be arbitrarily defined. As a result, an attacker can manipulate the parameters to transfer tokens from another user's balance to their own, leading to potential unauthorized token transfers and loss of funds.

Recommendation

To mitigate this issue, enforce strict validation of all parameters, particularly when `_toLightning` is enabled. Additional access control measures should be implemented to prevent unauthorized users from exploiting this function.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: d3c911e076323978dd6dbbfcd2e3c58538b26c95.

HTC-01 | UNCLAIMED REWARDS FROM AAVE DURING ASSET TRANSFER IN `deposit` AND `withdraw`

Category	Severity	Location	Status
Volatile Code	Minor	HashedTimeLock.sol (commit:b34e276): 559	Resolved

Description

In the `deposit()` function, user assets are deducted and deposited into Aave if the variable `tokenMap[_token].stake` is true. When the `withdraw()` function is called, the same amount of assets is withdrawn from Aave. However, any rewards that accumulate while the assets are in Aave remain unclaimed, as there is no mechanism to withdraw these rewards. This could result in the users not receiving their entitled rewards from Aave.

Recommendation

Introduce a mechanism to claim Aave rewards before performing the withdrawal in the withdraw function.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: e4f898fa8a5bc72daf50dacebf67e3f43c960069.

HTL-05 | INCOMPATIBILITY WITH DEFLATIONARY TOKENS

Category	Severity	Location	Status
Volatile Code	Minor	HashedTimeLock.sol (commit:550ee8): 513	Acknowledged

Description

The project design may not be compatible with non-standard ERC20 tokens, such as deflationary tokens or rebase tokens.

The functions use `transferFrom()` / `transfer()` to move funds from the sender to the recipient but fail to verify if the received token amount matches the transferred amount. This could pose an issue with fee-on-transfer tokens, where the post-transfer balance might be less than anticipated, leading to balance inconsistencies. There might be subsequent checks for a second transfer, but an attacker might exploit leftover funds (such as those accidentally sent by another user) to gain unjustified credit.

Scenario

When transferring deflationary ERC20 tokens, the input amount may not equal the received amount due to the charged transaction fee. For example, if a user sends 100 deflationary tokens (with a 10% transaction fee), only 90 tokens actually arrive to the contract. However, a failure to discount such fees may allow the same user to withdraw 100 tokens from the contract, which causes the contract to lose 10 tokens in such a transaction.

Recommendation

We advise the client to regulate the set of tokens supported and add necessary mitigation mechanisms to keep track of accurate balances if there is a need to support non-standard ERC20 tokens.

Alleviation

[Astra Team, 12/25/2024]: During actual operation, the functionality of adding token support will involve filtering and selection.

HTL-06 | USAGE OF `transfer()` FOR SENDING NATIVE TOKENS

Category	Severity	Location	Status
Coding Style	Minor	HashedTimeLock.sol (commit:550ee8): 385, 390, 393, 439, 442	Resolved

Description

After EIP-1884 was included in the Istanbul hard fork, it is not recommended to use `.transfer()` or `.send()` for transferring native tokens as these functions have a hard-coded value for gas costs making them obsolete as they are forwarding a fixed amount of gas, specifically `2300`. This can cause issues in case the linked statements are meant to be able to transfer funds to other contracts instead of EOAs.

Recommendation

We recommend using the `sendValue()` function of `Address` contract from OpenZeppelin. See https://github.com/OpenZeppelin/openzeppelin-contracts/blob/v4.4.2/contracts/utils/Address.sol

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: d3c911e076323978dd6dbbfcd2e3c58538b26c95.

HTL-07 | THIRD-PARTY DEPENDENCY USAGE

Category	Severity	Location	Status
Design Issue	Minor	HashedTimeLock.sol (commit:550ee8): 38	Resolved

Description

The contract is serving as the underlying entity to interact with one or more third party protocols. The scope of the audit treats third party entities as black boxes and assumes their functional correctness. However, in the real world, third parties can be compromised and this may lead to lost or stolen assets. In addition, upgrades of third parties can possibly create severe impacts, such as increasing fees of third parties, migrating to new LP pools, etc.

- The contract `HashedTimeLock` interacts with third party contract with `AavePool` interface via `aavePool` .

Recommendation

The auditors understood that the business logic requires interaction with third parties. item_output is recommended for the team to constantly monitor the statuses of third parties to mitigate the side effects when unexpected activities are observed.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: e4f898fa8a5bc72daf50dacebf67e3f43c960069 by removing the protocol.

HTL-08 | POTENTIAL CROSS-CHAIN REPLAY ATTACK

Category	Severity	Location	Status
Logical Issue	Minor	HashedTimeLock.sol (commit:550ee8): 347	Resolved

Description

Within `verifySignature()` function, the signed messages are not properly verified with the current chain ID, thus allowing attackers to perform replay attacks across chains. Hardcoded or cached chain ID values are also vulnerable since a hard fork may occur and change the chain ID in the future.

Recommendation

Recommend verifying the data against the current chain ID by using `block.chainid` or `chainid()` within the same transaction.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: d3c911e076323978dd6dbbfcd2e3c58538b26c95.

HTL-09 | LACK OF MAXIMUM LENGTH VALIDATION FOR LSP NAME IN `registerLsp()` ALLOWS INVALID LONG LSP NAME

Category	Severity	Location	Status
Volatile Code	Minor	HashedTimeLock.sol (commit:550ee8): 480~489	Resolved

Description

The `registerLsp()` function is designed to validate the length of the LSP name and URL. However, there is an issue in the current code: at lines 480-482, the logic used to validate the LSP name length is incorrectly applied to validate the `_url`. As a result, the LSP name length can exceed the expected limit.

Additionally, the error message at L488 does not accurately reflect the incorrect length of the passed-in `_url`.

Recommendation

Recommend refactoring the code to validate the length of the LSP name argument value within `registerLsp()` function.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: d3c911e076323978dd6dbbfcd2e3c58538b26c95.

HTL-12 | MISSING TIMELOCK VALIDATION IN `withdraw()` ALLOWS ASSET WITHDRAWAL AFTER THE LOCK TIME EXPIRES

Category	Severity	Location	Status
Logical Issue	● Minor	HashedTimeLock.sol (commit:550ee8): 368	● Acknowledged

Description

The `withdraw()` function allows users to withdraw swapped assets on the EVM chain before the expiration time. The current issue is that the `withdraw()` function lacks logic to validate the expiration time. As a result, even after the lock time expires, users can still withdraw assets if the locked asset has not been refunded.

Recommendation

The audit team would like to ask Astra team to confirm if the current code logic aligns with the original design.

Alleviation

[Astra Team, 12/25/2024]: Issue acknowledged. If the preimage is known, it means the invoice in the Lightning Network has been paid. Even though the time has expired, performing a withdrawal is still reasonable.

HTL-13 | MISSING FEE REFUND IN `deposit()` FUNCTION LOCKS ETH FEE LEFTOVER IN CONTRACT

Category	Severity	Location	Status
Logical Issue	Minor	HashedTimeLock.sol (commit:550ee8): 91~93, 326	Resolved

Description

Users will pay a fee for locking their assets, which is used for cross-chain swapping, by calling the `deposit()` function. The current code lacks logic to refund any excess fee payment if the fee exceeds the required amount.

The `_fee` represents the total amount of ETH to be transferred to the `watcher/astra/lsp/user` accounts. In this case, if `msg.value` exceeds `_fee`, the `deposit()` function will still execute successfully, but the excess ETH (`msg.value - _fee`) will remain locked in the contract because, except for ETH fee distribution, there is no code logic implemented for ETH withdrawal from the contract.

Recommendation

Recommend refactoring the code to ensure that any leftover fee payment is refunded to the user.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: d3c911e076323978dd6dbbfcd2e3c58538b26c95.

HTL-10 | UNLOCKED COMPILER VERSION

Category	Severity	Location	Status
Language Version	Informational	HashedTimeLock.sol (commit:550ee8): 2	Acknowledged

Description

The contract has unlocked compiler version. An unlocked compiler version in the source code of the contract permits the user to compile it at or above a particular version. This, in turn, leads to differences in the generated bytecode between compilations due to different compiler versions. This can lead to an ambiguity when debugging as compiler specific bugs may occur in the codebase that would be hard to identify over a span of multiple compiler versions rather than a specific one.

Recommendation

We advise that the compiler version is instead locked at the lowest version possible that the contract can be compiled at. For example, for version `v0.6.2` the contract should contain the following line:

```
pragma solidity 0.6.2;
```

Alleviation

[Astra Team, 12/25/2024]: The team acknowledged the finding.

HTL-11 | UNUSED VARIABLE

Category	Severity	Location	Status
Design Issue	Informational	HashedTimeLock.sol (commit:550ee8): 43	Resolved

Description

The variable `minFee` is declared and assigned but never used in the `HashedTimeLock` contract.

Recommendation

Consider utilizing the variable, or remove unused variables.

Alleviation

[Astra Team, 12/25/2024]: The team heeded the advice and resolved the issue in commit: d3c911e076323978dd6dbbfcd2e3c58538b26c95.

HTL-14 | INCONSISTENCY BETWEEN CODE AND COMMENT

Category	Severity	Location	Status
Logical Issue	Informational	HashedTimeLock.sol (commit:550ee8): 19, 345	Acknowledged

Description

Based on the contract comments, the `HashedTimeLock` contract is described as supporting cross-chain swaps between EVM chains and the Lightning Network.

However, the logic of the `deposit()` function indicates that it also supports non-Lightning Network chains.

Recommendation

Recommend refactoring the code to ensure the comments are consistent with the code logic.

Alleviation

[Astra Team, 12/25/2024]: The team acknowledged the finding.

APPENDIX | ASTRA - AUDIT

Finding Categories

Categories	Description
Coding Style	Coding Style findings may not affect code behavior, but indicate areas where coding practices can be improved to make the code more understandable and maintainable.
Language Version	Language Version findings indicate that the code uses certain compiler versions or language features with known security issues.
Volatile Code	Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases and may result in vulnerabilities.
Logical Issue	Logical Issue findings indicate general implementation issues related to the program logic.
Centralization	Centralization findings detail the design choices of designating privileged roles or other centralized controls over the code.
Design Issue	Design Issue findings indicate general issues at the design level beyond program logic that are not covered by other finding categories.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.

DISCLAIMER | CERTIK

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Services Agreement, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Agreement. This report provided in connection with the Services set forth in the Agreement shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Agreement. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without CertiK’s prior written consent in each instance.

This report is not, nor should be considered, an “endorsement” or “disapproval” of any particular project or team. This report is not, nor should be considered, an indication of the economics or value of any “product” or “asset” created by any team or project that contracts CertiK to perform a security assessment. This report does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors, business, business model or legal compliance.

This report should not be used in any way to make decisions around investment or involvement with any particular project. This report in no way provides investment advice, nor should be leveraged as investment advice of any sort. This report represents an extensive assessing process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. CertiK’s position is that each company and individual are responsible for their own due diligence and continuous security. CertiK’s goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies, and in no way claims any guarantee of security or functionality of the technology we agree to analyze.

The assessment services provided by CertiK is subject to dependencies and under continuing development. You agree that your access and/or use, including but not limited to any services, reports, and materials, will be at your sole risk on an as-is, where-is, and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.

ALL SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF ARE PROVIDED “AS IS” AND “AS AVAILABLE” AND WITH ALL FAULTS AND DEFECTS WITHOUT WARRANTY OF ANY KIND. TO THE MAXIMUM EXTENT PERMITTED UNDER APPLICABLE LAW, CERTIK HEREBY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS, IMPLIED, STATUTORY, OR OTHERWISE WITH RESPECT TO THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS. WITHOUT LIMITING THE FOREGOING, CERTIK SPECIFICALLY DISCLAIMS ALL IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, TITLE AND NON-INFRINGEMENT, AND ALL WARRANTIES ARISING FROM COURSE OF DEALING, USAGE, OR TRADE PRACTICE. WITHOUT LIMITING THE FOREGOING, CERTIK MAKES NO WARRANTY OF ANY KIND THAT THE SERVICES, THE LABELS, THE ASSESSMENT REPORT, WORK PRODUCT, OR OTHER MATERIALS, OR ANY PRODUCTS OR RESULTS OF THE USE THEREOF, WILL MEET CUSTOMER’S OR ANY OTHER PERSON’S REQUIREMENTS, ACHIEVE ANY INTENDED RESULT, BE COMPATIBLE OR WORK WITH ANY SOFTWARE, SYSTEM, OR OTHER SERVICES, OR BE SECURE, ACCURATE, COMPLETE, FREE OF HARMFUL CODE, OR ERROR-FREE. WITHOUT LIMITATION TO THE FOREGOING, CERTIK PROVIDES NO WARRANTY OR

UNDERTAKING, AND MAKES NO REPRESENTATION OF ANY KIND THAT THE SERVICE WILL MEET CUSTOMER'S REQUIREMENTS, ACHIEVE ANY INTENDED RESULTS, BE COMPATIBLE OR WORK WITH ANY OTHER SOFTWARE, APPLICATIONS, SYSTEMS OR SERVICES, OPERATE WITHOUT INTERRUPTION, MEET ANY PERFORMANCE OR RELIABILITY STANDARDS OR BE ERROR FREE OR THAT ANY ERRORS OR DEFECTS CAN OR WILL BE CORRECTED.

WITHOUT LIMITING THE FOREGOING, NEITHER CERTIK NOR ANY OF CERTIK'S AGENTS MAKES ANY REPRESENTATION OR WARRANTY OF ANY KIND, EXPRESS OR IMPLIED AS TO THE ACCURACY, RELIABILITY, OR CURRENCY OF ANY INFORMATION OR CONTENT PROVIDED THROUGH THE SERVICE. CERTIK WILL ASSUME NO LIABILITY OR RESPONSIBILITY FOR (I) ANY ERRORS, MISTAKES, OR INACCURACIES OF CONTENT AND MATERIALS OR FOR ANY LOSS OR DAMAGE OF ANY KIND INCURRED AS A RESULT OF THE USE OF ANY CONTENT, OR (II) ANY PERSONAL INJURY OR PROPERTY DAMAGE, OF ANY NATURE WHATSOEVER, RESULTING FROM CUSTOMER'S ACCESS TO OR USE OF THE SERVICES, ASSESSMENT REPORT, OR OTHER MATERIALS.

ALL THIRD-PARTY MATERIALS ARE PROVIDED "AS IS" AND ANY REPRESENTATION OR WARRANTY OF OR CONCERNING ANY THIRD-PARTY MATERIALS IS STRICTLY BETWEEN CUSTOMER AND THE THIRD-PARTY OWNER OR DISTRIBUTOR OF THE THIRD-PARTY MATERIALS.

THE SERVICES, ASSESSMENT REPORT, AND ANY OTHER MATERIALS HEREUNDER ARE SOLELY PROVIDED TO CUSTOMER AND MAY NOT BE RELIED ON BY ANY OTHER PERSON OR FOR ANY PURPOSE NOT SPECIFICALLY IDENTIFIED IN THIS AGREEMENT, NOR MAY COPIES BE DELIVERED TO, ANY OTHER PERSON WITHOUT CERTIK'S PRIOR WRITTEN CONSENT IN EACH INSTANCE.

NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH SERVICES, ASSESSMENT REPORT, AND ANY ACCOMPANYING MATERIALS.

THE REPRESENTATIONS AND WARRANTIES OF CERTIK CONTAINED IN THIS AGREEMENT ARE SOLELY FOR THE BENEFIT OF CUSTOMER. ACCORDINGLY, NO THIRD PARTY OR ANYONE ACTING ON BEHALF OF ANY THEREOF, SHALL BE A THIRD PARTY OR OTHER BENEFICIARY OF SUCH REPRESENTATIONS AND WARRANTIES AND NO SUCH THIRD PARTY SHALL HAVE ANY RIGHTS OF CONTRIBUTION AGAINST CERTIK WITH RESPECT TO SUCH REPRESENTATIONS OR WARRANTIES OR ANY MATTER SUBJECT TO OR RESULTING IN INDEMNIFICATION UNDER THIS AGREEMENT OR OTHERWISE.

FOR AVOIDANCE OF DOUBT, THE SERVICES, INCLUDING ANY ASSOCIATED ASSESSMENT REPORTS OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.

Elevating Your Entire Web3 Journey

Founded in 2017 by leading academics in the field of Computer Science from both Yale and Columbia University, CertiK is a leading blockchain security company that serves to verify the security and correctness of smart contracts and blockchain-based protocols. Through the utilization of our world-class technical expertise, alongside our proprietary, innovative tech, we're able to support the success of our clients with best-in-class security, all whilst realizing our overarching vision; provable trust for all throughout all facets of blockchain.