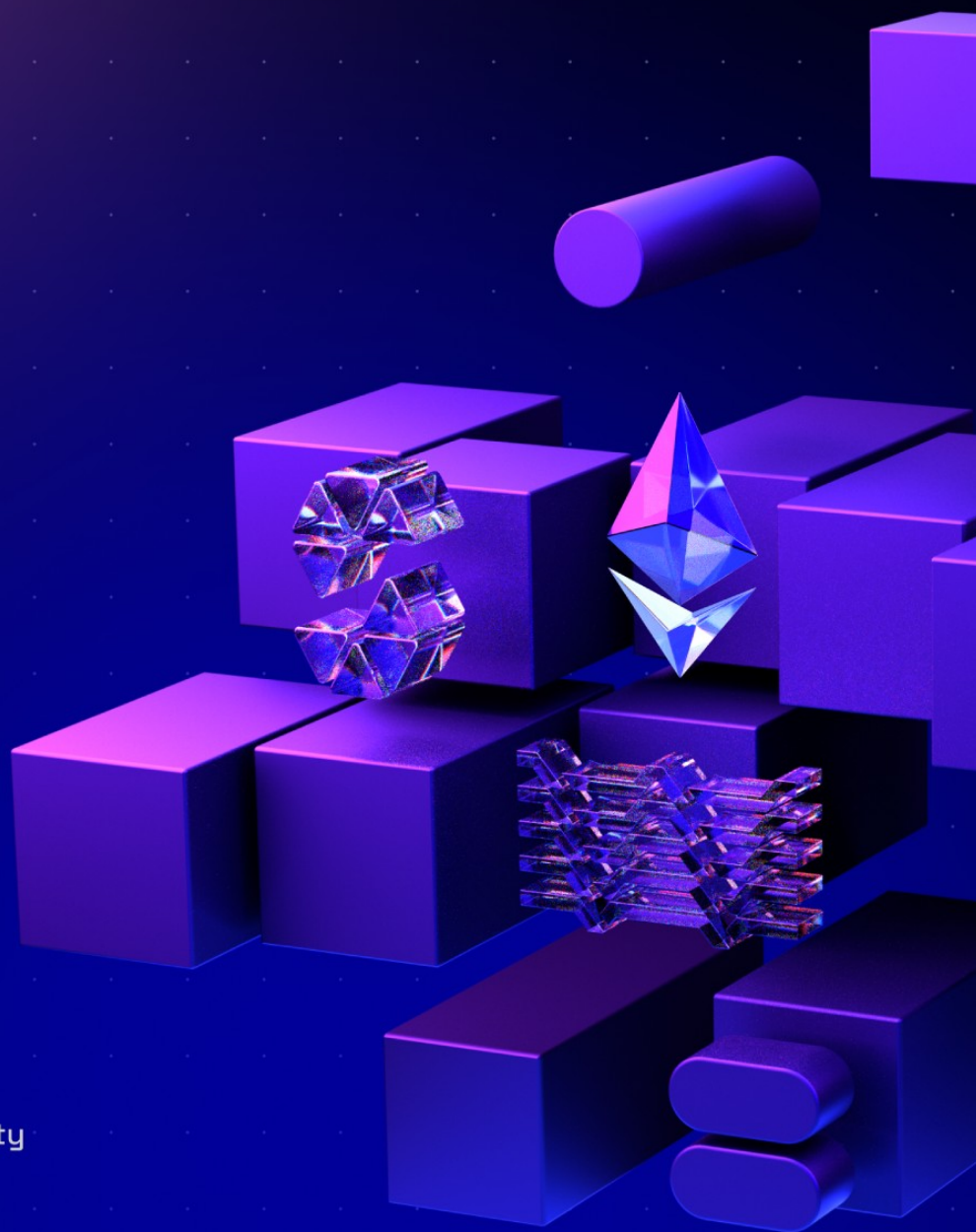
ackee
blockchain security

Hyperlend

Protocol

24.3.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	11.02.2025
1.0	Final Report	23.02.2025
2.0-draft	Draft Report	27.02.2025
3.0-draft	Draft Report	13.03.2025
3.0	Final Report	24.03.2025

2. Overview

This document presents our findings in reviewed contracts.

2.1. Ackee Blockchain Security

Ackee Blockchain Security is an in-house team of security researchers performing security audits focusing on manual code reviews with extensive fuzz testing for Ethereum and Solana. Ackee is trusted by top-tier organizations in web3, securing protocols including Lido, Safe, and Axelar.

We develop open-source security and developer tooling Wake for Ethereum and Trident for Solana, supported by grants from Coinbase and the Solana Foundation. Wake and Trident help auditors in the manual review process to discover hardly recognizable edge-case vulnerabilities.

Our team teaches about blockchain security at the Czech Technical University in Prague, led by our co-founder and CEO, Josef Gattermayer, Ph.D. As the official educational partners of the Solana Foundation, we run the School of Solana and the Solana Auditors Bootcamp.

Ackee's mission is to build a stronger blockchain community by sharing our knowledge.

Ackee Blockchain a.s.
Rohanske nabrezi 717/4
186 00 Prague, Czech Republic
https://ackee.xyz
hello@ackee.xyz

2.2. Audit Methodology

1. **Verification of technical specification**
   The audit scope is confirmed with the client, and auditors are onboarded to the project. Provided documentation is reviewed and compared to the audited system.

2. **Tool-based analysis**
   A deep check with Solidity static analysis tool Wake in companion with Solidity (Wake) extension is performed, flagging potential vulnerabilities for further analysis early in the process.

3. **Manual code review**
   Auditors manually check the code line by line, identifying vulnerabilities and code quality issues. The main focus is on recognizing potential edge cases and project-specific risks.

4. **Local deployment and hacking**
   Contracts are deployed in a local Wake environment, where targeted attempts to exploit vulnerabilities are made. The contracts' resilience against various attack vectors is evaluated.

5. **Unit and fuzz testing**
   Unit tests are run to verify expected system behavior. Additional unit or fuzz tests may be written using Wake framework if any coverage gaps are identified. The goal is to verify the system's stability under real-world conditions and ensure robustness against both expected and unexpected inputs.

2.3. Finding Classification

A *Severity* rating of each finding is determined as a synthesis of two sub-ratings: *Impact* and *Likelihood*. It ranges from *Informational* to *Critical*.

If we have found a scenario in which an issue is exploitable, it will be assigned an impact rating of *High*, *Medium*, or *Low*, based on the direness of the consequences it has on the system. If we haven't found a way, or the issue is only exploitable given a change in *configuration* (system settings or parameters, such as deployment scripts, compiler configurations, using multi-signature wallets for owners, etc.) or given a change in the codebase, then it will be assigned an impact rating of *Warning* or *Info*.

Low to *High* impact issues also have a *Likelihood*, which measures the probability of exploitability during runtime.

The full definitions are as follows:

Severity

		Likelihood			
		High	**Medium**	**Low**	**N/A**
Impact	**High**	Critical	High	Medium	-
	Medium	High	Medium	Low	-
	Low	Medium	Low	Low	-
	Warning	-	-	-	Warning
	Info	-	-	-	Info

Table 1. Severity of findings

Impact

- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue cannot be exploited given the current code and/or *configuration*, but could be a security vulnerability if these were to change slightly. If we haven't found a way to exploit the issue given the time constraints, it might be marked as a "Warning" or higher, based on our best estimate of whether it is currently exploitable.
- **Info** - The issue is on the borderline between code quality and security. Examples include insufficient logging for critical operations. Another example is that the issue would be security-related if code or *configuration* was to change.

Likelihood

- **High** - The issue is exploitable by virtually anyone under virtually any circumstance.
- **Medium** - Exploiting the issue currently requires non-trivial preconditions.
- **Low** - Exploiting the issue requires strict preconditions.

2.4. Review Team

The following table lists all contributors to this report. For authors of the specific revision, see the “Revision team” section in the respective “Report revision” chapter.

Member’s Name	Position
Michal Převrátil	Lead Auditor
Martin Veselý	Auditor
Naoki Yoshida	Auditor
Jan Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

2.5. Disclaimer

We’ve put our best effort to find all vulnerabilities in the system, however our findings shouldn’t be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

3. Executive Summary

Hyperlend is a lending protocol deployed on the Hyperliquid chain. The protocol implements risk-segmented lending pools designed for different use cases. The protocol's infrastructure includes cross-chain deposit endpoints for protocol pools, looping contracts that enable position management through flashloans, and helper contracts for asset listing functionality.

Revision 1.0

Hyperlend engaged Ackee Blockchain Security to perform a security review of the Hyperlend protocol with a total time donation of 46 engineering days in a period between January 10 and February 7, 2025, with Michal Převrátil as the lead auditor.

The audit was performed on the following repositories and commits:

- hyperlend-core commit `425624`[1];
- hyperlend-isolated commit `37c678`[2];
- looping-contracts (private repository) commit `0fdde7`[3];
- core-config-engine commit `0339f1`[4];
- cross-chain-lending-deposits (private repository) commit `43b101`[5].

The scope of the review included:

- differential changes in the hyperlend-core repository compared to the forked Aave v3.0.2 codebase;
- differential changes in the hyperlend-isolated repository compared to the forked Fraxlend V3 codebase;
- all Solidity contracts in the looping-contracts repository, excluding interfaces;

- all Solidity contracts in the core-config-engine repository, excluding interfaces and mocks;
- all Solidity contracts in the cross-chain-lending-deposits repository, excluding interfaces and mocks.

No additional code was reviewed outside of the scope of the review.

Initially, the cross-chain-lending-deposits repository was delivered for the review with commit `2576ca`[6] but as the code was not in a compilable state and was missing essential functionality, the commit was changed during the review period.

The protocol is intended to be deployed on the Hyperliquid chain. As the chain is still under development and the documentation is insufficient (missing HyperEVM technical specification, addresses of deployed contracts, etc.), the review process was only supported by the information provided by the client and it was not possible to test the contracts under the real conditions.

We began our review with a deep dive into the logic of the contracts. We supported our review with static analysis tools, including Wake, and manually guided fuzzing checking basic functionality of the code in-scope.

During the review, we paid special attention to:

- ensuring tokens cannot be stolen or unintentionally locked in the contracts;
- detecting possible reentrancies in the code;
- checking integration with third-party contracts is correct and secure;
- looking for common issues such as data validation.

Our review resulted in 44 findings, ranging from Info to Critical severity. The most severe finding C1 poses a critical risk of all collateral tokens being stolen

from the isolated pools of the protocol. The finding was reported despite being out-of-scope for the review, as the core issue was in incorrect usage of a new Chainlink-like price provider in the context of the original Fraxlend V3 codebase. The issue was undetectable by performing only differential review without the context of the original codebase.

Findings C1, M1, M2, M7, M10, L2 were discovered through manually-guided fuzzing using the Wake testing framework. Findings M5, M10, M11, M13, and I10 were discovered through Wake static analysis.

The overall code quality is below average with many findings being easy to catch even by static analysis tools (M10, M11) and a peer review. The major part of the protocol, the core pool, and isolated pools, are forks of the Aave v3.0.2 and Fraxlend V3 codebases, respectively. Forking of such codebases must be performed very carefully with a lot of attention to detail, especially given the fact that the original authors are often not willing to share all security-relevant details[7] and even intentionally retain high-severity issues in their codebases[8][9] to discourage forking. Codebases of such protocols do not always reflect the code used on-chain[8][9] (see Appendix C). The Aave v3.0.2 repository is marked as deprecated and no longer maintained[10].

The security guarantee of this review is especially limited by that only differential changes were reviewed for the forked codebases. This is advised to be considered insufficient given the aforementioned reasons and a new critical (C1) finding being found even after a third-party security assessment performed before this review (with the critical finding being in-scope for the assessment).

The communication with the client was unreliable. Although explicitly stated by the client that native tokens are not expected to be bridged through `BridgeInitiator` cross-chain gateways, the support was added right after the communication regarding the missing support. Because of that, the finding

was reported as M2.

The review revision concluded with 12.5 engineering days unused, providing sufficient time for the fix review needed given the number and severity of the issues discovered.

Due to the poor code quality, high number of discovered issues including high and critical severity issues, and especially because the forked codebases were out-of-scope for the review even though they cannot be considered safe[8][9] and the C1 critical finding was discovered out-of-scope, Ackee Blockchain Security recommends Hyperlend before the deployment to:

- fix the reported issues and perform a review of the fixed codebase; and
- perform a full review of the forked codebases (Aave v3.0.2 and Fraxlend V3).

Additionally, Ackee Blockchain Security recommends Hyperlend to:

- perform deposits larger than 10,000 raw tokens when listing a new asset;
- consider using `Ownable2Step` instead of `Ownable` for better security of changing contract owners;
- use static analysis tools (like Wake) to detect common issues;
- extend the project testing suite to cover all code from Hyperlend and involve integration testing.

See Report Revision 1.0 for the system overview and trust model.

Revision 2.0

Hyperlend engaged Ackee Blockchain Security to review fixes of the findings from the previous revision in a period between February 17 and February 24, 2025, with Michal Převrátil as the lead auditor.

The fix review was performed on the following repositories and commits:

- hyperlend-core commit `625161`[11];
- hyperlend-isolated commit `0b90ce`[12];
- looping-contracts (private repository) commit `cb6fac`[13];
- core-config-engine commit `4ff785`[14];
- cross-chain-lending-deposits (private repository) commit `38dc8a`[15].

Only the fixes for the findings from the previous revision were reviewed.

30 findings were fixed and the remaining 14 findings were acknowledged by the client. Hyperlend has decided to acknowledge the M13 to avoid unnecessary modifications to the original Aave codebase. While this is a valid approach, it requires reliability of the HyperEVM oracles responsible for the price data.

The overall code quality has improved.

Additionally, Hyperlend is working with Aave DAO to become a friendly fork and use the latest version v3.3 of the Aave protocol[16]. This will improve the security of the protocol as the official forks receive all security fixes and have security incidents reported. Once updated, Ackee Blockchain Security recommends validating the new Aave codebase integration with Hyperlend's codebase.

Revision 3.0

Hyperlend engaged Ackee Blockchain Security to perform a security review of the new Hyperlend Core (an Aave v3.2 friendly fork) in a period between March 12 and March 18, 2025, with Michal Převrátil as the lead auditor.

The audit was performed on the hyperlend-core-new repository, commit `0c2b14`[17]. The scope included all changes made in the `src` directory compared to the original Aave v3.2 codebase.

We began our review by assessing the changes made to the original Aave v3.2 codebase and performing a manual review of the modifications. We used static analysis tools to ensure no issues were introduced.

Our review concluded with no new findings identified. The protocol now uses a more recent Aave codebase with minimal modifications as a friendly fork, enabling it to benefit from any security-related updates and announcements from the Aave team. This addresses our previous remarks from Executive Summary Revision 1.0. There is nothing blocking the protocol deployment.

Ackee Blockchain Security recommends Hyperlend to:

- keep informed about the latest fixes made to the Aave and Fraxlend codebases; and
- maintain best security practices when listing new tokens, ensuring quality of price oracles, and monitoring health of the protocol pools.

See Report Revision 3.0 for the changes made to the Aave v3.2 codebase.

[1] full commit hash: `4256249f94b8762a5ce41caa2283c0d989efc593`

[2] full commit hash: `37c678450f37a923517e7a203f8353599e143b7e`

[3] full commit hash: `0fdde7b1033263b7c7579c9ec505dfb635197f3c`

[4] full commit hash: `0339f192bb98b55c856d1cc6f76a04bd6fe687ee`

[5] full commit hash: `43b101ae5f63ea873cb29199f6ac96ece2207576`

[6] full commit hash: `2576caa13409d49bd886965d3c4b3fe61e40397e`

[7] https://x.com/lemiscate/status/1762076544250322974

[8] https://governance.aave.com/t/aave-v2-v3-security-incident-04-11-2023/15335

[9] https://github.com/aave/aave-v3-core/blob/782f51917056a53a2c228701058a6c3fb233684a/contracts/protocol/tokenization/StableDebtToken.sol#L124

[10] https://github.com/aave/aave-v3-core

[11] full commit hash: `6251612c12c1d413348771ba096e437a3d6c69e2`

[12] full commit hash: `0b90ce3a992623d4a49c97d888d0188825530e7a`

[13] full commit hash: `cb6fac72d8469393a0a01b49f09e0511955ea9c1`

[14] full commit hash: `4ff7856850cf02854b8fe8df94f487f659007b40`

[15] full commit hash: `38dc8ae0a3bb9134cf21c83912bb587d22c3e435`

[16] https://governance.aave.com/t/temp-check-recognize-hyperlend-as-a-friendly-fork/20969

[17] full commit hash: `0c2b14f62d95282f2a93a568cc04f6dd2dc89fc0`

4. Findings Summary

The following section summarizes findings we identified during our review. Unless overridden for purposes of readability, each finding contains:

- *Description*
- *Exploit scenario* (if severity is low or higher)
- *Recommendation*
- *Fix* (if applicable).

Summary of findings:

Critical	High	Medium	Low	Warning	Info	Total
1	1	13	8	11	10	44

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
C1: No revert on stale Chainlink price	Critical	1.0	Fixed
H1: Possible locked tokens	High	1.0	Fixed
M1: Incorrect proposal ID emitted	Medium	1.0	Fixed
M2: Missing support for bridging native tokens	Medium	1.0	Fixed
M3: Arbitrary token transfer through unrestricted refund function	Medium	1.0	Fixed

Finding title	Severity	Reported	Status
M4: Incorrect token balance check leading to failed position closures	Medium	1.0	Fixed
M5: Divide before multiply in `openPosition` function	Medium	1.0	Fixed
M6: Missing `payable` modifier	Medium	1.0	Fixed
M7: `minAmountOut` calculation too restrictive	Medium	1.0	Fixed
M8: Inconsistent token symbol formatting	Medium	1.0	Fixed
M9: Missing token validation in bridge initiation	Medium	1.0	Fixed
M10: `SafeERC20` not used	Medium	1.0	Fixed
M11: Native `transfer` revert out-of-gas	Medium	1.0	Fixed
M12: `WalletBalanceProvider` native tokens lockup	Medium	1.0	Fixed
M13: Missing Chainlink price feed validation	Medium	1.0	Acknowledged
L1: Missing swap deadline protection	Low	1.0	Fixed
L2: Try/catch may still revert	Low	1.0	Partially fixed
L3: Unsatisfiable condition on closing positions with flashloans	Low	1.0	Fixed
L4: Incorrect error messages	Low	1.0	Fixed

Finding title	Severity	Reported	Status
L5: Missing receive function for native token handling	Low	1.0	Acknowledged
L6: Missing queued transaction verification in `cancelTransaction`	Low	1.0	Acknowledged
L7: Same transaction can be queued multiple times	Low	1.0	Acknowledged
L8: Native token recovery can be bypassed	Low	1.0	Fixed
W1: Missing check to catch underflow error	Warning	1.0	Fixed
W2: Double listing proposal ID	Warning	1.0	Acknowledged
W3: Case insensitive import	Warning	1.0	Fixed
W4: Hardhat console imports	Warning	1.0	Fixed
W5: Unused state variables in `StrategyManager`	Warning	1.0	Acknowledged
W6: Missing zero address validation	Warning	1.0	Acknowledged
W7: Lack of events	Warning	1.0	Acknowledged
W8: Missing proposal existence validation	Warning	1.0	Acknowledged
W9: Potential negative exponent in `CHAINLINK_NORMALIZATION` calculation	Warning	1.0	Fixed

Finding title	Severity	Reported	Status
W10: Incorrect token balance used for debt value calculation	Warning	1.0	Fixed
W11: Incorrect interface usage for debt token	Warning	1.0	Acknowledged
I1: Missing underscore in internal function name	Info	1.0	Fixed
I2: Incorrect variable naming due to typo	Info	1.0	Fixed
I3: Unused `Ownable` inheritance	Info	1.0	Acknowledged
I4: Inconsistent visibility for `_reversePath` function	Info	1.0	Fixed
I5: `getUserAccountData` function reverts when token price is zero	Info	1.0	Acknowledged
I6: `getUserPairs` returns an array with empty positions	Info	1.0	Acknowledged
I7: Unused `swapPath` parameter in `SwapParams` struct	Info	1.0	Acknowledged
I8: Unused function with potential data truncation risk	Info	1.0	Fixed
I9: Incorrect documentation	Info	1.0	Fixed
I10: Variables can be immutable	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Martin Veselý	Auditor
Naoki Yoshida	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

Hyperlend is a lending protocol that enables users to borrow and lend tokens through either a core pool with shared risk or through isolated pools with independent risk. The protocol forks Aave v3.0.2 and Fraxlend V3.

The protocol will be deployed on the Hyperliquid chain. Deposit gateways using Stargate will be deployed on other chains, enabling users to perform cross-chain deposits into Hyperlend's pools.

The solution includes a helper `Looping` contract that enables users to open and close leveraged positions through flashloans. Token listing occurs through helper contracts to ensure initial liquidity seeding of new pools.

Trust Model

Users must trust Hyperlend not to lock funds in the protocol or manipulate token prices.

Stargate gateways must be trusted to correctly relay messages between chains during cross-chain deposits.

Fuzzing

Manually-guided fuzzing tested the basic functionalities of the protocol. The fuzzing did not involve differential comparison of token accounting between the on-chain and fuzzed states, as most accounting logic was out of scope for this audit.

The fuzzing focused on the following aspects:

- basic usage of the core pool;
- listing of new tokens to the core pool using the `ListingConfigEngine` and `ListingsConfigEngineFactory` contracts;
- basic usage of isolated pools;
- listing of new tokens to isolated pools using the `ListingsConfigEngine` contract;
- cross-chain deposits through the `BridgeInitiator` and `BridgeReceiver` contracts; and
- opening and closing leveraged positions through the `Looping` contract.

The list of all implemented execution flows and invariants is available in Appendix B.

Findings

The following section presents the list of findings discovered in this revision. For the complete list of all findings, Go back to Findings Summary

C1: No revert on stale Chainlink price

Critical severity issue

Impact:	High	Likelihood:	High
Target:	HyperlendPairCore.sol	Type:	Logic error

Description

The `HyperlendPairCore._updateExchangeRate` function is responsible for updating the borrow asset to collateral asset exchange rate. The function fetches the latest price using a Chainlink-like interface through the `OracleChainlink` contract.

Listing 1. Excerpt from HyperlendPairCore._updateExchangeRate

```
484 bool _oneOracleBad;
485 (_oneOracleBad, _lowExchangeRate, _highExchangeRate) = IDualOracle(
486     _exchangeRateInfo.oracle
487 ).getPrices();
488
489 // If one oracle is bad data, emit an event for off-chain monitoring
490 if (_oneOracleBad) emit WarnOracleData(_exchangeRateInfo.oracle);
```

The contract does not revert the execution when the oracle returns a bad price.

A price is considered bad when it is stale or negative in either the multiply or divide price aggregators.

Listing 2. Excerpt from OracleChainlink._getChainlinkPrice

```
79 if (CHAINLINK_MULTIPLY_ADDRESS != address(0)) {
80     (, int256 _answer, , uint256 _updatedAt, ) = AggregatorV3Interface(
81         CHAINLINK_MULTIPLY_ADDRESS
82     ).latestRoundData();
83
84     // If data is stale or negative, set bad data to true and return
```

```
85      if (_answer <= 0 || (block.timestamp - _updatedAt > maxOracleDelay)) {
86          _isBadData = true;
87          return (_isBadData, _price);
88      }
89      _price = _price * uint256(_answer);
90  }
91
92  if (CHAINLINK_DIVIDE_ADDRESS != address(0)) {
93      (, int256 _answer, , uint256 _updatedAt, ) = AggregatorV3Interface(
94          CHAINLINK_DIVIDE_ADDRESS
95      ).latestRoundData();
96
97      // If data is stale or negative, set bad data to true and return
98      if (_answer <= 0 || (block.timestamp - _updatedAt > maxOracleDelay)) {
99          _isBadData = true;
100          return (_isBadData, _price);
101       }
102       _price = _price / uint256(_answer);
103  }
```

When a bad price is returned, the value `1e36` or greater is used in the `HyperlendPairCore` contract as the exchange rate. This behavior poses a critical risk to the protocol.

The issue was discovered through manually-guided fuzzing using the Wake testing framework. See Appendix B for more information on the fuzzing campaign performed during the audit.

Exploit scenario

One of the price aggregators in the UNI/USDT pair becomes outdated. The pair logic uses `1e36` as the exchange rate, making any borrowing inefficient or impossible.

All opened borrowing positions become insolvent. The positions can be liquidated by repaying the user's debt and receiving the user's collateral in return. Due to the extremely high exchange rate, the liquidation is significantly profitable for any attacker. The attacker effectively steals the entire user's collateral for a negligible amount of borrow tokens.

Note that the original forked code of the `HyperlendPairCore` contract is not affected by this issue. The original code uses a dual Uniswap/Chainlink oracle that does not suffer from the described problem.

Recommendation

Revert the execution when the oracle returns a bad price.

Fix 2.0

The issue was fixed by adding revert calls to all execution branches that represent an invalid price in the `OracleChainLink` contract.

Listing 3. Excerpt from OracleChainlink._getChainlinkPrice

```
86 if (_answer <= 0 || (block.timestamp - _updatedAt > maxOracleDelay)) {
87     revert("invalid oracle price");
88 }
```

Listing 4. Excerpt from OracleChainlink._getChainlinkPrice

```
98 if (_answer <= 0 || (block.timestamp - _updatedAt > maxOracleDelay)) {
99     revert("invalid oracle price");
100 }
```

Go back to Findings Summary

H1: Possible locked tokens

High severity issue

Impact:	High	Likelihood:	Medium
Target:	ListingsConfigEngineFactory.sol	Type:	Logic error

Description

The `ListingConfigEngine` contract represents a listing proposal for a token with all necessary logic to list it in the Hyperlend's core pool. The contract requires the listed token to be atomically supplied to the pool upon listing to prevent rounding issues caused by the token's supply being too low.

Listing 5. Excerpt from ListingConfigEngine

```
241 function _seedPool(address token, address pool, uint256 amount, address
    seedAmountsHolder) internal {
242     require(amount >= 10000, 'seed amount too low');
243
244     IERC20Detailed(token).transferFrom(msg.sender, address(this), amount);
245     IERC20Detailed(token).approve(pool, amount);
246     IPool(pool).supply(token, amount, seedAmountsHolder, 0);
247 }
```

The initial supply is transferred from `msg.sender`, which is the `ListingsConfigEngineFactory` contract, as all proposals are executed from there.

Listing 6. Excerpt from ListingsConfigEngineFactory

```
59 function executeProposal(uint256 _id) external onlyOwner() {
60     ListingConfigEngine(proposalConfigEngines[_id]).executeProposal();
61     emit ProposalExecuted(_id);
62 }
```

For the execution to succeed, tokens must be prepared in advance in the factory contract. However:

- tokens may become permanently locked if the proposal is never agreed upon, as there is no recovery mechanism; and
- the `ListingConfigEngine` contract does not use `SafeERC20` or its alternative (see M10), which may cause proposal execution to fail with non-standard ERC-20 tokens.

Exploit scenario

A proposal to list a new ABC token is created, and tokens are transferred to the factory contract. A subsequent transaction attempts to execute the proposal.

However, due to a non-standard token interface, the execution fails when ABI decoding data from the `transferFrom` call inside `ListingConfigEngine`. As a result, the tokens become permanently locked in the factory contract.

Recommendation

Implement a recovery mechanism for ERC-20 tokens in the `ListingConfigEngineFactory` contract.

Fix 2.0

The issue was fixed by sending all tokens bound with a proposal to the sender upon the proposal cancellation.

Listing 7. Excerpt from ListingsConfigEngineFactory

```
69 function cancelProposal(uint256 _id) external onlyOwner() {
70     ListingConfigEngine(proposalConfigEngines[_id]).cancelProposal();
71     emit ProposalCanceled(_id);
72
73     //refund seed token balance
```

```
74     IERC20 underlyingToken =
   IERC20(ListingConfigEngine(proposalConfigEngines[_id]).getAssetConfig().under
   lyingAsset);
75     underlyingToken.safeTransfer(msg.sender,
   underlyingToken.balanceOf(address(this)));
76 }
```

Note that the solution is overly restrictive as it requires tokens to be already present in the factory contract to be able to cancel a proposal. Without the tokens temporarily being present in the factory, the proposal cannot be cancelled.

Go back to Findings Summary

M1: Incorrect proposal ID emitted

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	ListingsConfigEngineFactory.sol	Type:	Logic error

Description

The `ListingsConfigEngineFactory` contract enables the creation of listing proposals for new assets in the protocol core pool. The `createProposal` function emits a `ProposalCreated` event with a proposal ID parameter.

Listing 8. Excerpt from ListingsConfigEngineFactory.createProposal

```
52 proposalConfigEngines[lastProposalId] = address(listingConfigEngine);
53 lastProposalId++;
54
55 emit ProposalCreated(lastProposalId);
```

The `lastProposalId` variable is incremented before the `ProposalCreated` event is emitted, resulting in an incorrect proposal ID being emitted in the event.

The issue was discovered through manually-guided fuzzing using the Wake testing framework. See Appendix B for more information on the fuzzing campaign performed during the audit.

Exploit scenario

The incorrect proposal ID in the emitted event prevents off-chain systems from correctly matching proposals with their corresponding IDs.

Recommendation

Move the `ProposalCreated` event emission before incrementing the

`lastProposalId` variable.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

M2: Missing support for bridging native tokens

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	BridgeInitiator.sol	Type:	Logic error

Description

The `BridgeInitiator.sol` contract facilitates token bridging from the source chain to Hyperliquid for deposits into Hyperlend's pools. The `initiate` function initiates the bridging process by transferring tokens to the `BridgeInitiator` contract and increasing the allowance to the Stargate pool:

Listing 9. Excerpt from BridgeInitiator.initiate

```
78 IERC20(_sourceToken).safeTransferFrom(msg.sender, address(this), _amount);
79 IERC20(_sourceToken).safeIncreaseAllowance(_stargatePool, _amount);
```

The contract implementation does not support bridging native tokens, as it exclusively uses ERC-20 token functions.

The issue was discovered through manually-guided fuzzing using the Wake testing framework. See Appendix B for more information on the fuzzing campaign performed during the audit.

Exploit scenario

The contract fails to process native token transfers due to its reliance on ERC-20 token functions, making it impossible to bridge native tokens from source chains.

Recommendation

Only perform the ERC-20 transfer and increase the allowance if the

`_sourceToken` address is non-zero.

Fix 2.0

Fixed by calling the ERC-20 functions only if the `_sourceToken` address is non-zero and sending the correct native token value to the Stargate pool.

Go back to Findings Summary

M3: Arbitrary token transfer through unrestricted refund function

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	Looping.sol	Type:	Logic error

Description

The `_refund` function in the `Looping` contract is responsible for transferring all tokens before repaying a flashloan.

Listing 10. Excerpt from Looping

```
241 function _refund(address debtAsset, address yieldAsset, uint256 amount,
    uint256 premium, address user) internal {
242     uint256 debtAssetBalance = IERC20(debtAsset).balanceOf(address(this));
243     uint256 yieldAssetBalance = IERC20(yieldAsset).balanceOf(address(this));
244
245     if (debtAssetBalance > amount + premium){
246         IERC20(debtAsset).transfer(user, debtAssetBalance - (amount +
    premium));
247     }
248     if (yieldAssetBalance > 0){
249         IERC20(yieldAsset).transfer(user, yieldAssetBalance);
250     }
251 }
```

The contract also implements a token recovery mechanism through the `rescueTokens` function.

Listing 11. Excerpt from Looping

```
287 function rescueTokens(address _token, uint256 _amount) external onlyOwner(){
288     if (_token == address(0)){
289         (bool success, ) = payable(msg.sender).call{value: _amount}("");
290         require(success, "transfer failed");
291     } else {
```

```
292         IERC20(_token).transfer(msg.sender, _amount);
293     }
294 }
```

However, any user can call the `openPosition` and `closePosition` functions that trigger the `_refund` function, allowing them to withdraw any leftover tokens from the contract.

Exploit scenario

Alice sends ERC-20 tokens to the `Looping` contract by mistake. Bob calls the `openPosition` or `closePosition` function with any parameters. The `_refund` function transfers all tokens of the same type to Bob. When the owner attempts to recover the tokens using `rescueTokens`, the tokens have already been transferred to Bob.

Recommendation

Implement the following changes:

- calculate the exact refund amount that should be returned to the user; and
- restrict the `_refund` function to only transfer the calculated amount.

Fix 2.0

The issue was fixed by refunding any leftover tokens in the contract to the owner before performing any logic in the `openPosition` and `closePosition` functions.

Go back to Findings Summary

M4: Incorrect token balance check leading to failed position closures

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	Looping.sol	Type:	Logic error

Description

The following code snippet shows the logic for calculating token amounts to be repaid and withdrawn when closing a full position using a flashloan.

Listing 12. Excerpt from Looping._executeClosePosition

```
186 IERC20 debtDebtToken = IERC20(IPool(
    msg.sender).getReserveData(yieldAsset).variableDebtTokenAddress);
187
188 //close full position if repaymentAmount == maxUint256
189 if (repaymentAmount == type(uint256).max){
190     repaymentAmount = debtDebtToken.balanceOf(user);
191     withdrawAmount = hYieldToken.balanceOf(user);
192 }
```

The calculation uses the `yieldToken` variable debt instead of the `debtAsset` variable debt, which is incorrect.

Exploit scenario

Alice attempts to close a full position using the `Looping` contract. However, the transaction repays a different amount based in the `yieldToken` variable debt instead of the `debtAsset` variable debt, potentially leading to unintended position modifications.

Recommendation

Modify the code to retrieve the variable debt token using the `debtAsset`

parameter:

```
IERC20 debtDebtToken = IERC20(IPool(
msg.sender).getReserveData(debtAsset).variableDebtTokenAddress);
```

Also note that the `debtDebtToken` fetching logic can be moved into the if condition as a gas optimization.

Fix 2.0

The issue was fixed by using the `debtAsset` parameter to retrieve the variable debt token. The statement was moved into the if condition as a gas optimization.

Go back to Findings Summary

M5: Divide before multiply in `openPosition` function

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	Looping.sol	Type:	Arithmetics

Description

Listing 13. Excerpt from Looping.openPosition

```
70 uint256 minAmountOut = (_flashloanAmount * 1e8 / _minAmountOut) *
   _initialAmount / 1e8;
```

The `openPosition` function performs division before multiplication in its calculations. This order of operations causes precision loss and results in inaccurate values. As a consequence, swaps may execute with lower slippage protection than intended.

Exploit scenario

Alice attempts to swap from a yield token to a debt token. Due to the precision loss in calculations, the `minAmountOut` value becomes zero, effectively removing all slippage protection from the swap.

This issue is particularly severe when the swap involves tokens with different decimal places (for example, when `_minAmountOut` uses 18 decimals and `_flashloanAmount` uses 6 decimals).

Recommendation

Rearrange the calculation to perform multiplication before division to preserve precision.

Fix 2.0

Fixed as a consequence of the M7 finding fix.

Go back to Findings Summary

M6: Missing `payable` modifier

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	StrategyManager.sol	Type:	Logic error

Description

The `StrategyManager` contract implements functions for executing external calls, including multicalls, that may send native tokens.

Listing 14. Excerpt from StrategyManager

```
32 function executeCall(address target, uint256 value, bytes memory data, bool
   allowRevert) public onlyOwner() returns (bytes memory) {
33     (bool success, bytes memory returnData) = target.call{value:
   value}(data);
```

The `executeCall` and `executeMultiCall` functions in the `StrategyManager` contract lack the `payable` modifier, preventing them from receiving native tokens through regular transactions.

Exploit scenario

Due to the missing `payable` modifier, native tokens can only be transferred to the `StrategyManager` contract through the `selfdestruct` mechanism. This approach is gas inefficient and provides a poor user experience.

Recommendation

Add the `payable` modifier to the `executeCall` and `executeMultiCall` functions.

Also implement the `receive` function to allow the contract to receive native tokens.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

M7: `minAmountOut` calculation too restrictive

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	Looping.sol	Type:	Logic error

Description

The `Looping.openPosition` function opens a leveraged position using a flashloan. Users can initiate the position with either:

- a debt token (the token to be borrowed); or
- a yield token (the token to be supplied).

When starting with a yield token, the function performs an initial swap to convert it into the debt token.

The minimum amount of tokens required for the initial swap is calculated as follows:

Listing 15. Excerpt from Looping.openPosition

```
70 uint256 minAmountOut = (_flashloanAmount * 1e8 / _minAmountOut) *
   _initialAmount / 1e8;
```

This calculation assumes that `_flashloanAmount / _minAmountOut` represents the price of the debt token in terms of the yield token. However, this assumption is incorrect because:

- the calculation does not properly account for slippage since the first swap occurs in the opposite direction; and
- the first swap affects the pool price, which is not considered in the calculation.

The issue was discovered through manually-guided fuzzing using the Wake testing framework. See Appendix B for more information on the fuzzing campaign performed during the audit.

Exploit scenario

Alice attempts to open a position starting with yield tokens. The transaction fails because:

- the initial swap changes the pool price; and
- the slippage is calculated in the wrong direction, making the `minAmountOut` check too restrictive.

Recommendation

Add a separate input parameter to specify the minimum amount of tokens required for the initial swap, instead of calculating `minAmountOut` within the `openPosition` function.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

M8: Inconsistent token symbol formatting

Medium severity issue

Impact:	Low	Likelihood:	High
Target:	ListingsConfigEngineFactory.sol	Type:	Code quality

Description

Listing 16. Excerpt from ListingConfigEngine._initReserve

```
200 variableDebtTokenSymbol: string(abi.encodePacked("hVariableDebt",
    debtTokenPrefix, symbol)),
201 stableDebtTokenName: string(abi.encodePacked("HyperLend ", debtTokenPrefix,
    " Stable Debt ", symbol)),
202 stableDebtTokenSymbol: string(abi.encodePacked("hStableDebt", symbolPrefix,
    symbol)),
```

The token symbol naming convention is inconsistent in the `ListingConfigEngine._initReserve` function. The `variableDebtTokenSymbol` uses `debtTokenPrefix` instead of `symbolPrefix`. The `debtTokenPrefix` is designed for token names and may contain multiple words, while token symbols should be concise.

Exploit scenario

The proposal tried to create a listing with a `debtTokenPrefix` intended for the token name and a `symbolPrefix` for the token symbol. However, the proposal execution unintentionally created a debtToken with a symbol containing `debtTokenPrefix`, confusing users.

Due to the incorrect use of `debtTokenPrefix`, the token symbols are not concise and cause confusion.

Recommendation

Replace `debtTokenPrefix` with `symbolPrefix` when setting the `variableDebtTokenSymbol` value.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

M9: Missing token validation in bridge initiation

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	BridgeInitiator.sol	Type:	Data validation

Description

The `initiate` function in the `BridgeInitiator` contract lacks validation between the `_sourceToken` parameter and the token associated with the Stargate pool. Specifically, the contract does not verify that `_sourceToken` matches the address returned by the Stargate pool's `token()` function during ERC-20 transfers.

The `token()` function in the Stargate pool contract returns the address of the authorized ERC-20 token implementation for that pool.

Exploit scenario

Alice initiates a bridge transaction with the following conditions:

- uses WETH as the `_sourceToken`;
- selects a Stargate pool configured for native tokens;
- provides approval for WETH; and
- sends native tokens (also needed to pay for the bridge transaction).

The transaction executes successfully, resulting in:

- WETH tokens becoming locked in the `BridgeInitiator` contract; and
- Alice receiving native tokens on the destination chain.

Bob, a malicious actor, can exploit this vulnerability by:

- specifying any unauthorized ERC-20 token as `_sourceToken`; and
- using a Stargate pool configured for the WETH token.

Recommendation

Add validation to ensure `_sourceToken` matches the address returned by `IStargate(_stargatePool).token()` when processing cross-chain deposits.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

M10: `SafeERC20` not used

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	ReservesSetupHelper.sol, ListingsConfigEngine.sol, HyperlendPairCore.sol, UniswapV3Swapper.sol, Looping.sol, ListingConfigEngine.sol, ListingsConfigEngineFactory.sol	Type:	Denial of service

Description

The project interacts with ERC-20 tokens in multiple places without using the `SafeERC20` library or its equivalents.

See Appendix B for the full list of all occurrences.

Note that `SafeERC20` is not used in `core/contracts/periphery/WrappedTokenGatewayV3.sol`. However, the contract only interacts with WETH and Aave-like ATokens, which are expected to be fully ERC-20 compliant.

Exploit scenario

Alice attempts to interact with a non-compliant ERC-20 token through one of the affected contracts. Due to `SafeERC20` not being used, the transaction reverts, resulting in a denial of service.

Recommendation

Use `SafeERC20` or its equivalents in all the aforementioned files.

Fix 2.0

The issue was fixed by using the `SafeERC20` library in all the affected files.

Go back to Findings Summary

M11: Native `transfer` revert out-of-gas

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	BridgeInitiator.sol, BridgeReceiver.sol	Type:	Denial of service

Description

The `BridgeInitiator` contract allows users to initiate token bridge transfers. Native tokens are required to pay for the cross-chain transfer, with unused native tokens being returned to the user.

Listing 17. Excerpt from `BridgeInitiator.initiate`

```
89 payable(msg.sender).transfer(address(this).balance);
```

Both the `BridgeInitiator` and `BridgeReceiver` contracts implement native token recovery functionality.

Listing 18. Excerpt from `BridgeInitiator.recover`

```
146 payable(msg.sender).transfer(amount);
```

Listing 19. Excerpt from `BridgeReceiver.recover`

```
153 payable(msg.sender).transfer(amount);
```

The `transfer` function forwards only 2300 gas units to the recipient. If the recipient is a contract, this limited gas amount may cause the transaction to revert.

See Appendix B for the full list of affected contracts.

Exploit scenario

Alice, a user with tokens in a smart contract wallet, attempts to receive native tokens from the bridge. The transaction reverts because the `transfer` function's gas limit of 2300 units is insufficient for the smart contract wallet to process the incoming transfer. As a result:

- native tokens remain locked in the bridge contracts; and
- the bridge initiation functionality becomes unusable for users with smart contract wallets.

Recommendation

Use low-level calls instead of `transfer` to send native tokens.

Fix 2.0

The issue was fixed by using low-level calls instead of `transfer` in all the affected contracts.

Go back to Findings Summary

M12: `WalletBalanceProvider` native tokens lockup

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	WalletBalanceProvider.sol	Type:	Configuration

Description

Listing 20. Excerpt from WalletBalanceProvider

```
24 receive() external payable {
25     //only contracts can send ETH to the core
26     require(msg.sender.isContract(), '22');
27 }
```

The `WalletBalanceProvider` contract implements a `receive()` function that enables it to receive native tokens from contract addresses. However, the contract lacks functionality to send native tokens, resulting in permanent native token lockup.

Exploit scenario

Alice interacts with the `WalletBalanceProvider` contract through a smart contract. During the interaction, Alice accidentally sends native tokens to the contract. The native tokens become permanently locked in the contract due to the absence of a withdrawal mechanism.

Recommendation

Remove the `receive()` function to prevent the contract from accepting native token transfers (except for `selfdestruct`).

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

M13: Missing Chainlink price feed validation

Medium severity issue

Impact:	Medium	Likelihood:	Medium
Target:	ListingConfigEngine.sol, Oracle.sol, UiPoolDataProviderV3.sol	Type:	Data validation

Description

The Aave `Oracle` contract retrieves asset prices using a Chainlink-compatible interface.

Listing 21. Excerpt from Oracle.getAssetPrice

```
108 int256 price = source.latestAnswer();
```

The implementation has two significant issues:

- it does not validate the freshness of the price data; and
- it uses the deprecated `latestAnswer` function.

See Appendix B for a complete list of affected code locations.

Exploit scenario

Alice can exploit stale price data when the Chainlink oracle has not been updated, potentially manipulating protocol operations that depend on accurate asset prices.

Recommendation

Use the `latestRoundData` function instead of the deprecated `latestAnswer` function and validate the staleness of the price data in the `Oracle` contract.

Acknowledgment 2.0

The client has acknowledged the issue, deciding not to modify the code present in the latest Aave codebase.

Go back to Findings Summary

L1: Missing swap deadline protection

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	BridgeReceiver.sol, Looping.sol	Type:	N/A

Description

The `BridgeReceiver` contract is responsible for receiving cross-chain Stargate messages with tokens and depositing them to the requested pool. Optionally, tokens may be first needed to be converted to the pool's token. For this purpose, the `BridgeReceiver` contract calls a `Swapper` contract.

Listing 22. Excerpt from BridgeReceiver.lzCompose

```
91 try ISwapper(_swapParams.swapper).swap(
92     _swapParams.oftOnDestination,
93     address(tokenToSupply),
94     amountToSupply,
95     _swapParams.minAmountOut,
96     address(this),
97     owner(),
98     block.timestamp,
99     _swapParams.extraData
100 ) returns (uint256 _amountOut) {
```

The swap call uses `block.timestamp` as the swap deadline. This offers no protection against postponing the transaction execution by miners and relayers.

The same issue exists in the `Looping` contract, where the `_swap` function uses `block.timestamp` as the swap deadline.

Listing 23. Excerpt from Looping._swap

```
223 ISwapper(swapper).swapExactTokensForTokensSupportingFeeOnTransferTokens(
224     amountToSwap,
225     minAmountOut,
226     path,
227     address(this),
228     referralAddress,
229     block.timestamp
230 );
```

Exploit scenario

Hyperliquid's miners or Stargate's relayers may take advantage of the missing deadline protection by postponing the transaction execution/relaying for their own benefit.

Recommendation

To mitigate this issue:

- add a deadline parameter to the cross-chain transfer payload and use it as the swap deadline in the `BridgeReceiver` contract; and
- add a deadline parameter to the relevant functions in the `Looping` contract.

Fix 2.0

Both occurrences of the issue have been fixed by adding a deadline parameter to the relevant functions and using it as the swap deadline.

Go back to Findings Summary

L2: Try/catch may still revert

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	BridgeReceiver.sol	Type:	Logic error

Description

The `BridgeReceiver` contract implements cross-chain message processing with token transfers. To prevent tokens from being locked on the destination chain, the contract uses try/catch blocks for external calls.

Listing 24. Excerpt from BridgeReceiver.lzCompose

```
91 try ISwapper(_swapParams.swapper).swap(
92     _swapParams.oftOnDestination,
93     address(tokenToSupply),
94     amountToSupply,
95     _swapParams.minAmountOut,
96     address(this),
97     owner(),
98     block.timestamp,
99     _swapParams.extraData
100 ) returns (uint256 _amountOut) {
```

Listing 25. Excerpt from BridgeReceiver.lzCompose

```
112 try IIsolatedPool(_swapParams.lendingPool).deposit(amountToSupply, _user) {
113     emit Supplied(_user, address(tokenToSupply), amountToSupply);
114 } catch {
```

Listing 26. Excerpt from BridgeReceiver.lzCompose

```
119 try ICorePool(_swapParams.lendingPool).supply(address(tokenToSupply),
    amountToSupply, _user, 0) {
120     emit Supplied(_user, address(tokenToSupply), amountToSupply);
121 } catch {
```

However, the try/catch blocks do not prevent execution revertions in the following cases:

- the target of the external call is not a contract (its `code.length` is 0); and
- the external call returns data that cannot be ABI-decoded to the expected return type.

The issue was discovered through manually-guided fuzzing using the Wake testing framework. See Appendix B for more information on the fuzzing campaign performed during the audit.

Exploit scenario

Alice accidentally registers an empty account (EOA) as a swapper contract in the `BridgeReceiver` contract. When this empty account is used as a swapper, the execution reverts due to the code size check implicitly added by the Solidity compiler.

As a result, tokens sent through the `BridgeReceiver` contract become locked on the destination chain.

Recommendation

Use low-level external calls instead of try/catch blocks and perform the fallback operation if the external call fails or returned data cannot be ABI-decoded.

Partial solution 2.0

The issue was partially fixed by adding a check for non-zero contract code size when whitelisting swapper and lending pool contracts.

Listing 27. Excerpt from BridgeReceiver

```
128 function setSwapperWhitelist(address swapper, bool status) external
    onlyOwner {
```

```
129     if (status) require(swapper.code.length != 0, "invalid swapper code
    length");
130     swappers[swapper] = status;
131 }
```

Listing 28. Excerpt from BridgeReceiver

```
136 function setPoolWhitelist(address pool, bool status) external onlyOwner {
137     if (status) require(pool.code.length != 0, "invalid pool code length");
138     lendingPools[pool] = status;
139 }
```

The scenario with failing ABI decoding was acknowledged by the client because the whitelisted contracts are trusted and expected to be valid.

Go back to Findings Summary

L3: Unsatisfiable condition on closing positions with flashloans

Low severity issue

Impact:	Low	Likelihood:	Medium
Target:	Looping.sol	Type:	Logic error

Description

The `Looping` contract contains an unsatisfiable condition in the flash loan position closure logic.

Listing 29. Excerpt from Looping._executeClosePosition

```
189 if (repaymentAmount == type(uint256).max){
190     repaymentAmount = debtDebtToken.balanceOf(user);
191     withdrawAmount = hYieldToken.balanceOf(user);
192 }
```

The condition compares `repaymentAmount` with `uint256.max`. Since `repaymentAmount` equals `flashloanAmount` during position closure, and it is impossible to execute a flash loan of `uint256.max` tokens, this condition can never evaluate to true.

Exploit scenario

Alice attempts to close a full position using the `Looping` contract. However, the body of the condition is never reached, as it is impossible to take a flashloan with `uint256.max` amount.

Recommendation

Replace the `repaymentAmount` variable with `withdrawAmount` in the condition check.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

L4: Incorrect error messages

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	ListingConfigEngine.sol, Timelock.sol	Type:	Code quality

Description

The `ListingConfigEngine` contract contains an error message in the `_beforeProposal` function that does not match the actual check being performed. The error message states `missing assetListingAdmin privilegies` while the code checks for risk admin privileges.

Listing 30. Excerpt from ListingConfigEngine._beforeProposal

```
142 require(aclManager.isRiskAdmin(address(this)), "missing assetListingAdmin
    privilegies");
```

The `Timelock` contract contains an error message in the constructor that references an incorrect function name. The error message states `Timelock::setDelay` when it should reference `Timelock::constructor`.

Listing 31. Excerpt from Timelock.constructor

```
45 require(
46     delay_ <= MAXIMUM_DELAY,
47     'Timelock::setDelay: Delay must not exceed maximum delay.'
48 );
```

Exploit scenario

Alice attempts to deploy the `Timelock` contract with an invalid delay parameter. When the transaction reverts, Alice sees an error message referencing the `setDelay` function instead of the constructor, leading to

confusion about where the error occurred.

Recommendation

In the `ListingConfigEngine` contract, update the error message to reference the risk admin privilege check.

In the `Timelock` contract, update the constructor's error message to reference the correct function name.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

L5: Missing receive function for native token handling

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	Timelock.sol	Type:	Logic error

Description

Listing 32. Excerpt from Timelock

```
158 (bool success, bytes memory returnData) = target.call{value:
    value}(callData);
```

The `Timelock` contract accepts native tokens during the `executeTransaction` function execution; however, it lacks functionality to receive native tokens through direct transfers or refunds from external contracts.

Exploit scenario

Alice queues a transaction in the `Timelock` contract and executes it. The transaction reverts when it attempts to receive native tokens (e.g., through `timelock.call{value: 1 ether}("")`) because the contract lacks the required `receive` function.

Recommendation

Implement a `receive` function in the `Timelock` contract to enable native token transfers.

Acknowledgment 2.0

The client acknowledged this issue, as the `Timelock` contract will not be deployed in the production environment. Instead, the `TimelockController`

contract from the OpenZeppelin library will be used.

Go back to Findings Summary

L6: Missing queued transaction verification in `cancelTransaction`

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	Timelock.sol	Type:	Access control

Description

Listing 33. Excerpt from Timelock.cancelTransaction

```
118 bytes32 txHash = keccak256(abi.encode(target, value, signature, data, eta));
119 queuedTransactions[txHash] = false;
120
121 emit CancelTransaction(txHash, target, value, signature, data, eta);
```

The `cancelTransaction` function in the `Timelock` contract does not verify whether a transaction exists in the queue before canceling it. The function executes successfully and emits a `CancelTransaction` event even when attempting to cancel non-existent transactions.

Exploit scenario

Alice performs the following actions:

- queues a transaction using the `queueTransaction` function;
- attempts to cancel the transaction using `cancelTransaction` but provides incorrect parameters;
- receives successful cancellation confirmation and a `CancelTransaction` event is emitted;
- the originally queued transaction remains in the queue; and
- the queued transaction becomes executable after the timelock period expires.

Note: Only the owner can execute queued transactions.

Recommendation

Add a verification check for the transaction's existence in the `queuedTransactions` mapping before setting its status to false.

Acknowledgment 2.0

The client acknowledged this issue, as the `Timelock` contract will not be deployed in the production environment. Instead, the `TimelockController` contract from the OpenZeppelin library will be used.

Go back to Findings Summary

L7: Same transaction can be queued multiple times

Low severity issue

Impact:	Low	Likelihood:	Low
Target:	Timelock.sol	Type:	Logic error

Description

Listing 34. Excerpt from Timelock.queueTransaction

```
89 function queueTransaction(
90     address target,
91     uint value,
92     string memory signature,
93     bytes memory data,
94     uint eta
95 ) public returns (bytes32) {
96     require(msg.sender == admin, 'Timelock::queueTransaction: Call must come
   from admin.');
97     require(
98         eta >= getBlockTimestamp() + delay,
99         'Timelock::queueTransaction: Estimated execution block must satisfy
   delay.'
100     );
101
102     bytes32 txHash = keccak256(abi.encode(target, value, signature, data,
   eta));
103     queuedTransactions[txHash] = true;
```

The `Timelock` contract's `queueTransaction` function lacks protection against duplicate transaction queueing. The transaction hash is calculated using the input parameters (target, value, signature, data, eta). The function executes successfully even when identical parameters are provided multiple times. However, the `executeTransaction` function can only be executed once with a specific set of parameters.

Exploit scenario

Alice performs the following actions:

- queues a transaction with parameters (target, value, signature, data, eta);
- queues the exact same transaction again with identical parameters;
- both transactions are successfully queued; and
- when attempting to execute these transactions, only the first one succeeds while the second one fails.

Recommendation

Implement the following changes:

- add a `nonce` parameter when calculating the transaction hash in the `queueTransaction` function
- emit the event with the updated parameters
- increment the `nonce` in the `queueTransaction` function

Acknowledgment 2.0

The client acknowledged this issue, as the `Timelock` contract will not be deployed in the production environment. Instead, the `TimelockController` contract from the OpenZeppelin library will be used.

Go back to Findings Summary

L8: Native token recovery can be bypassed

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	BridgeInitiator.sol	Type:	Logic error

Description

Listing 35. Excerpt from BridgeInitiator.initiate

```
87 //refund any unused payment
88 if (address(this).balance > 0){
89     payable(msg.sender).transfer(address(this).balance);
90 }
```

The `recover()` function in the BridgeInitiator contract allows the owner to recover both ERC-20 tokens and native tokens. However, the native token recovery functionality is effectively disabled because any user can call the `initiate` function and receive a refund of the contract's entire native token balance, including any trapped native tokens meant to be recovered by the owner.

Additionally, the `_stargatePool` parameter lacks validation, allowing an attacker to specify malicious IStargate contracts to receive native tokens through the `sendToken` function.

Listing 36. Excerpt from BridgeInitiator.initiate

```
85 IStargate(_stargatePool).sendToken{ value: valueToSend }(sendParam,
   messagingFee, msg.sender);
```

Exploit scenario

Alice sends native tokens to the contract by mistake. When Alice contacts the contract owner to recover the tokens using the `recover` function, Bob

notices the contract's balance. Bob executes a transaction with the `initiate` function using arbitrary parameters before the contract owner can recover the tokens, allowing Bob to obtain the entire contract balance.

Recommendation

Implement the following changes:

- calculate the refund amount based on `msg.value` and transaction-specific parameters; and
- implement an address whitelist to validate the `_stargatePool` parameter.

Fix 2.0

The issue was fixed by refunding only the difference between `msg.value` and the value sent to the Stargate pool.

Listing 37. Excerpt from BridgeInitiator

```
103 if (address(this).balance > msg.value - valueToSend){
104     (bool success,) = payable(msg.sender).call{value: msg.value -
    valueToSend}("");
105     require(success, "refund failed");
106 }
```

Additionally, Stargate pool addresses are now whitelisted by the contract owner.

Go back to Findings Summary

W1: Missing check to catch underflow error

Impact:	Warning	Likelihood:	N/A
Target:	Looping.sol	Type:	Overflow/Underflow

Description

The calculation `uint256 repaymentAmount = _flashloanAmount - _initialAmount` in the `Looping.openPosition` function lacks an explicit validation check to ensure that `_flashloanAmount` is greater than or equal to `_initialAmount`. Although Solidity `^0.8.0` provides built-in overflow/underflow protection, implementing a custom check with a descriptive error message would improve code clarity and maintainability. The current implementation relies on the default arithmetic underflow revert, which does not provide clear information about the failure cause.

Listing 38. Excerpt from Looping.openPosition

```
78 uint256 repaymentAmount = _flashloanAmount - _initialAmount;
```

Recommendation

Add an explicit validation check with a descriptive error message before the subtraction operation.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

W2: Double listing proposal ID

Impact:	Warning	Likelihood:	N/A
Target:	ListingsConfigEngineFactory.sol, ListingConfigEngine.sol	Type:	Code quality

Description

The `ListingsConfigEngineFactory` contract allows creating `ListingConfigEngine` contract instances. Each instance is identified by a proposal ID:

Listing 39. Excerpt from ListingsConfigEngineFactory.createProposal

```
52 proposalConfigEngines[lastProposalId] = address(listingConfigEngine);
```

The `ListingConfigEngine` contract stores a `Proposal` struct that defines another proposal ID which may have a different value:

Listing 40. Excerpt from ListingConfigEngine

```
25 struct Proposal {
26     uint256 proposalId;             // proposal ID
27     MarketConfig marketConfig;      // info about the market we want to add
   the asset to
28     AssetConfig assetConfig;        // info about the asset we want to add
29 }
```

Recommendation

Consider removing the second, unused, proposal ID from the `Proposal` struct to avoid confusion.

Also note that the `ACLConfigEngine` and `CapsConfigEngine` contracts also define proposal IDs that are not used in the logic.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

W3: Case insensitive import

Impact:	Warning	Likelihood:	N/A
Target:	UiDataProviderIsolated.sol	Type:	Code quality

Description

The `UiDataProviderIsolated.sol` file imports the `OracleChainlink.sol` file. This may cause issues on case-sensitive file systems as the file is named `OracleChainLink.sol`.

Listing 41. Excerpt from UiDataProviderIsolated

```
7 import { OracleChainlink } from '../oracles/OracleChainlink.sol';
```

Recommendation

Update the import statement to match the exact file name:

- Rename the imported file to `OracleChainlink.sol`; or
- Change the import statement to use `OracleChainLink.sol`.

Fix 2.0

Fixed by renaming the file to `OracleChainlink.sol`.

Go back to Findings Summary

W4: Hardhat console imports

Impact:	Warning	Likelihood:	N/A
Target:	CapsConfigEngine.sol, UniswapV3Swapper.sol	Type:	Code quality

Description

The following contracts contain unused imports of the `hardhat/console.sol` library:

Listing 42. Excerpt from CapsConfigEngine

```
15 import "hardhat/console.sol";
```

Listing 43. Excerpt from UniswapV3Swapper

```
7 import "hardhat/console.sol";
```

These imports should be removed as they are not used in the code and could potentially be deployed to production.

Recommendation

Remove the unused `hardhat/console.sol` imports.

Fix 2.0

Fixed by removing the unused imports.

Go back to Findings Summary

W5: Unused state variables in `StrategyManager`

Impact:	Warning	Likelihood:	N/A
Target:	StrategyManager.sol	Type:	Code quality

Description

The `StrategyManager` contract contains state variables that are not utilized in any of the contract's logic:

Listing 44. Excerpt from StrategyManager

```
10 address public pool;
11 address public yieldAsset;
12 address public debtAsset;
```

While the contract implements functionality for performing arbitrary external calls:

Listing 45. Excerpt from StrategyManager

```
32 function executeCall(address target, uint256 value, bytes memory data, bool
   allowRevert) public onlyOwner() returns (bytes memory) {
33     (bool success, bytes memory returnData) = target.call{value:
   value}(data);
34     if (!allowRevert) require(success, 'execution reverted');
35     return returnData;
36 }
37
38 function executeMultiCall(Call[] memory calls) external onlyOwner() {
39     for (uint256 i  = 0; i < calls.length; i++){
40         executeCall(calls[i].target, calls[i].value, calls[i].data,
   calls[i].allowRevert);
41     }
42 }
```

The defined state variables remain unused throughout the implementation.

Recommendation

Consider removing the state variables or adjusting the contract logic to use them.

Acknowledgment 2.0

The client acknowledged this finding. The state variables are used by the UI although they are not used in the contract logic.

A comment was added to the code explaining the purpose of these variables:

Listing 46. Excerpt from StrategyManager

```
10 /// @notice variables are used on the UI to identify which StrategyManager is
   used for certain pairs of assets
```

Go back to Findings Summary

W6: Missing zero address validation

Impact:	Warning	Likelihood:	N/A
Target:	*/.sol	Type:	Data validation

Description

Multiple contracts in the codebase lack zero address validation for critical address parameters in constructors and setter functions. The absence of zero address validation could result in permanently broken contract functionality if zero addresses are inadvertently set. The following code snippets demonstrate instances where zero address validation is missing:

Listing 47. Excerpt from Timelock

```
43 constructor(address admin_, uint delay_) {
44     require(delay_ >= MINIMUM_DELAY, 'Timelock::constructor: Delay must
   exceed minimum delay.');
45     require(
46         delay_ <= MAXIMUM_DELAY,
47         'Timelock::setDelay: Delay must not exceed maximum delay.'
48     );
49
50     admin = admin_;
51     delay = delay_;
52 }
```

This is a user address in the destination chain.

Listing 48. Excerpt from BridgeInitiator.initiate

```
69 address _user,
```

Listing 49. Excerpt from BridgeReceiver

```
40 constructor(address[] memory _endpoints, address[] memory _stargatePools)
   Ownable(msg.sender) {
41     for (uint256 i = 0; i < _endpoints.length; i++){
```

```
42          endpoints[_endpoints[i]] = true;
43      }
44
45      for (uint256 i = 0; i < _stargatePools.length; i++){
46          stargatePools[_stargatePools[i]] = true;
47      }
48  }
```

Listing 50. Excerpt from Looping

```
28  constructor(address[] memory _pools, address[] memory _swappers, address
    _owner) Ownable(_owner) {
29      for (uint256 i = 0; i < _pools.length; i++){
30          pools[_pools[i]] = true;
31      }
32      for (uint256 i = 0; i < _swappers.length; i++){
33          swappers[_swappers[i]] = true;
34      }
35
36      referralAddress = _owner;
37  }
```

Listing 51. Excerpt from OracleChainlink.constructor

```
39      address _timelockAddress,
40      string memory _name
41  ) Timelock2Step() {
42      _setTimelock({_newTimelock: _timelockAddress});
```

Recommendation

Implement zero address validation checks in all constructors and setter functions where address parameters are used.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

W7: Lack of events

Impact:	Warning	Likelihood:	N/A
Target:	Looping.sol	Type:	Code quality

Description

The `Looping` contract lacks event definitions and emissions for tracking important state changes and operations. Events are essential for:

- monitoring contract activities;
- facilitating off-chain tracking;
- enabling debugging capabilities; and
- providing transparency for users and developers.

Recommendation

Define and emit events for all significant state changes and important operations in the `Looping` contract.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

W8: Missing proposal existence validation

Impact:	Warning	Likelihood:	N/A
Target:	ListingsConfigEngineFactory.sol	Type:	Data validation

Description

Listing 52. Excerpt from ListingsConfigEngineFactory

```
59 function executeProposal(uint256 _id) external onlyOwner() {
60     ListingConfigEngine(proposalConfigEngines[_id]).executeProposal();
61     emit ProposalExecuted(_id);
62 }
```

The `executeProposal` function lacks input validation for proposal existence. When called with a non-existent proposal ID, the transaction reverts without providing a descriptive error message, making it difficult for users to diagnose the failure reason.

Recommendation

Implement input validation to verify the proposal's existence and return a descriptive error message when the proposal ID is invalid.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

W9: Potential negative exponent in `CHAINLINK_NORMALIZATION` calculation

Impact:	Warning	Likelihood:	N/A
Target:	OracleChainLink.sol	Type:	Arithmetics

Description

Listing 53. Excerpt from OracleChainlink.constructor

```
54 CHAINLINK_NORMALIZATION =
55     10 **
56         (18 +
57             _multiplyDecimals -
58             _divideDecimals +
59             IERC20Metadata(_baseToken).decimals() -
60             IERC20Metadata(_quoteToken).decimals());
```

The `CHAINLINK_NORMALIZATION` calculation in the `OracleChainlink` contract may result in a negative exponent, which would cause the transaction to revert.

This could happen when the sum of `_divideDecimals` and `_quoteToken.decimals()` is greater than the sum of `_multiplyDecimals` and `_baseToken.decimals()` plus 18.

Recommendation

Implement input validation to ensure the exponent remains non-negative before performing the calculation.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

W10: Incorrect token balance used for debt value calculation

Impact:	Warning	Likelihood:	N/A
Target:	UiDataProvider.sol	Type:	Arithmetics

Description

Listing 54. Excerpt from UiDataProvider.getStrategy

```
119 debtValueUsd = vars.aYieldToken.scaledBalanceOf(_manager) * vars.debtPrice;
120 yieldValueUsd = vars.variableDebtToken.scaledBalanceOf(_manager) *
    vars.yieldPrice;
```

The code incorrectly uses the scaled balance of the `vars.aYieldToken` instead of the `vars.variableDebtToken` to calculate the debt value, and vice versa. This results in incorrect leverage calculations.

Recommendation

Use the correct token balance sources for calculating debt and yield values to ensure accurate leverage calculations.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

W11: Incorrect interface usage for debt token

Impact:	Warning	Likelihood:	N/A
Target:	OracleChainLink.sol	Type:	Code quality

Description

Listing 55. Excerpt from UiDataProvider.getStrategy

```
114 vars.variableDebtToken = IAToken(vars.debtReserve.variableDebtTokenAddress);
```

The variable debt token is incorrectly cast to `IAToken` interface. The function signatures is identical, but using the wrong interface type could lead to confusion.

Recommendation

Cast the variable debt token to the correct interface type.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

I1: Missing underscore in internal function name

Impact:	Info	Likelihood:	N/A
Target:	BridgeInitiator.sol	Type:	Code quality

Description

The `addressToBytes32` function in the `BridgeInitiator` contract is declared as `internal`. According to Solidity naming conventions, internal functions should have an underscore prefix to distinguish them from public functions. This convention improves code readability and helps prevent accidental external calls to internal functions.

Listing 56. Excerpt from BridgeInitiator

```
153 function addressToBytes32(address _addr) internal pure returns (bytes32) {
154     return bytes32(uint256(uint160(_addr)));
155 }
```

Recommendation

Rename the function to `_addressToBytes32` to follow the Solidity naming convention.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

I2: Incorrect variable naming due to typo

Impact:	Info	Likelihood:	N/A
Target:	BridgeInitiator.sol	Type:	Code quality

Description

A typographical error exists in the variable name `stargarePool` in the `BridgeInitiator` contract. While this issue does not impact the protocol's functionality, it affects code readability and maintainability.

Listing 57. Excerpt from BridgeInitiator.InitiateBridge

```
47 address stargarePool,
```

The variable `stargarePool` should be renamed to `stargatePool`.

Recommendation

Rename the variable `stargarePool` to `stargatePool` to correct the typographical error.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

I3: Unused `Ownable` inheritance

Impact:	Info	Likelihood:	N/A
Target:	StrategyManagerFactory.sol	Type:	Code quality

Description

The `StrategyManagerFactory` contract inherits from the `Ownable` contract but does not utilize any of its functionality. This inheritance introduces unnecessary code complexity.

Recommendation

Remove the `Ownable` inheritance from the `StrategyManagerFactory` contract.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

I4: Inconsistent visibility for `_reversePath` function

Impact:	Info	Likelihood:	N/A
Target:	Looping.sol	Type:	Function visibility

Description

Listing 58. Excerpt from Looping

```
254 function _reversePath(address[] memory _array) public pure returns(address[]
    memory) {
```

The `_reversePath` function in the `Looping` contract starts with an underscore, which by convention indicates an internal or private function. However, the function is declared as `public`, which violates this naming convention.

Recommendation

Change the visibility modifier of the `_reversePath` function from `public` to `internal`.

Fix 2.0

Fixed by following the recommendation.

Go back to Findings Summary

I5: `getUserAccountData` function reverts when token price is zero

Impact:	Info	Likelihood:	N/A
Target:	Pool.sol	Type:	N/A

Description

The `getUserAccountData` function in the `Pool` contract will revert if a user has deposited tokens whose price has fallen to zero. This core view function is essential for retrieving user account information.

Recommendation

Implement strict token listing criteria and carefully evaluate price oracle reliability before adding new tokens to the protocol.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

I6: `getUserPairs` returns an array with empty positions

Impact:	Info	Likelihood:	N/A
Target:	UiDataProviderIsolated.sol	Type:	Logic error

Description

Listing 59. Excerpt from UiDataProviderIsolated.getUserPairs

```
152 if (userAssetShares > 0 || userBorrowShares > 0 || userCollateralBalance >
    0){
153     (,,,uint64 ratePerSec,) = pair.currentRateInfo();
154     (uint128 borrowAmount,) = pair.totalBorrow();
155     (uint128 assetAmount,) = pair.totalAsset();
156
157     IERC20Metadata asset = IERC20Metadata(address(pair.asset()));
158     IERC20Metadata collateral = IERC20Metadata(
    address(pair.collateralContract()));
159
160     userPositions[i] = UserPosition({
161         pair: pairs[i],
```

The `getUserPairs` function returns an array of `UserPosition` structs for all `HyperlendPairs`. When a user has no position in a particular pair, the corresponding array slot contains a `UserPosition` struct with zero values. This implementation results in an array containing both valid positions and empty data structures.

Recommendation

Modify the function to return only pairs where the user has an active position by:

- implementing a counter to track the number of valid positions;
- inserting data only when a position exists; and

- resizing the final array to match the number of actual positions.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

I7: Unused `swapPath` parameter in `SwapParams` struct

Impact:	Info	Likelihood:	N/A
Target:	BridgeInitiator.sol	Type:	Unused code

Description

Listing 60. Excerpt from BridgeInitiator.SwapParams

```
30 address[] swapPath;
```

The `SwapParams` struct in the `BridgeInitiator` contract contains an unused `swapPath` parameter.

Recommendation

Remove the `swapPath` parameter from the `SwapParams` struct.

Acknowledgment 2.0

The client acknowledged this finding.

Go back to Findings Summary

I8: Unused function with potential data truncation risk

Impact:	Info	Likelihood:	N/A
Target:	BridgeInitiator.sol	Type:	Code quality

Description

Listing 61. Excerpt from BridgeInitiator

```
158 function prepareComposeMsg(address _user, SwapParams memory _swapParams)
    external pure returns (bytes32) {
159     return bytes32(abi.encode(_user, _swapParams));
160 }
```

The `prepareComposeMsg` function in the `BridgeInitiator` contract forcibly converts encoded data into `bytes32`, which leads to data truncation.

The function is not used in the current implementation of the contract.

Recommendation

Either:

- modify the function to return `bytes memory` instead of `bytes32` to prevent data truncation; or
- remove the unused function entirely from the contract.

Fix 2.0

The finding was fixed by removing the function.

Go back to Findings Summary

I9: Incorrect documentation

Impact:	Info	Likelihood:	N/A
Target:	BridgeInitiator.sol, BridgeReceiver.sol	Type:	Function visibility

Description

The following documentation issues were identified:

- the endpoints in `BridgeReceiver.sol` are incorrectly documented as Stargate instead of Layer Zero:

Listing 62. Excerpt from BridgeReceiver

```
29 /// @notice whitelisted stargate endpoints
```

Listing 63. Excerpt from BridgeReceiver

```
38 /// @param _endpoints array of stargate endpoints to whitelist
```

- the parameters in `BridgeInitiator.sol` are incorrectly documented as chain IDs instead of endpoint IDs:

Listing 64. Excerpt from BridgeInitiator

```
63 /// @param _destinationEndpointId id of the target chain
```

Listing 65. Excerpt from BridgeInitiator

```
97 /// @param _dstEid id of the target chain
```

- the following spelling errors exist in `BridgeReceiver.sol`:
  - ”emmited” should be ”emitted”

Listing 66. Excerpt from BridgeReceiver

```
34 /// @notice emmited when funds are successfully supplied to the lending pool
```

- ”amount to sent” should be ”amount to send”

Listing 67. Excerpt from BridgeReceiver

```
150 /// @param amount amount to sent
```

Recommendation

Update the documentation to:

- correct the endpoint references from Stargate to Layer Zero;
- fix the parameter descriptions from chain IDs to endpoint IDs; and
- correct the identified spelling errors.

Fix 2.0

All documentation issues were fixed.

Go back to Findings Summary

I10: Variables can be immutable

Impact:	Info	Likelihood:	N/A
Target:	ListingConfigEngine.sol	Type:	Code quality

Description

The codebase contains multiple state variables that can be declared as `immutable`. These variables are assigned only once during contract deployment and never modified afterward.

See Appendix B for the complete list of affected variables.

Recommendation

Declare all listed variables as `immutable` to:

- reduce gas costs for reading these variables;
- make the immutability of these values explicit in the code; and
- prevent accidental modifications.

Fix 2.0

All identified variables were declared as `immutable`.

Go back to Findings Summary

Report Revision 2.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Jan Převrátil	Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

Overview

Since there were no comprehensive changes in this revision, the complete overview is listed in the Executive Summary section Revision 2.0.

Report Revision 3.0

Revision Team

Member's Name	Position
Michal Převrátil	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The core pool implementation was replaced with a friendly fork of Aave v3.2. The following changes were made to the forked codebase:

- a new `Executor` contract inheriting from `Ownable` was added to delegatecall into Aave configuration payloads;
- `DefaultMarketInput` contract's configuration was updated to reflect the Hyperliquid chain's addresses and parameters;
- rebranding of `AToken` and `VariableDebtToken` names and symbols was performed in the `ListingEngine` contract.

Appendix A: How to cite

Please cite this document as:

Ackee Blockchain Security, Hyperlend: Protocol, 24.3.2025.

Appendix B: Wake Findings

This section lists the outputs from the Wake framework used for testing and static analysis during the audit.

B.1. Fuzzing

The following table lists all implemented execution flows in the Wake fuzzing framework.

ID	Flow	Added
F1	Listing new token to the core pool	1.0
F2	Supplying tokens to the core pool	1.0
F3	Withdrawing tokens from the core pool	1.0
F4	Borrowing tokens from the core pool	1.0
F5	Repaying tokens to the core pool	1.0
F6	Repaying with ATokens to the core pool	1.0
F7	Liquidating positions in the core pool	1.0
F8	Opening a position in the core pool through flashloan in `Looping`	1.0
F9	Closing a position in the core pool through flashloan in `Looping`	1.0
F10	Deploying a new isolated token pair pool	1.0
F11	Depositing tokens into an isolated pool	1.0
F12	Adding collateral to an isolated pool	1.0
F13	Removing collateral from an isolated pool	1.0
F14	Borrowing tokens from an isolated pool	1.0
F15	Repaying tokens to an isolated pool	1.0

ID	Flow	Added
F16	Queuing new transactions into `Timelock`	1.0
F17	Setting pending admin proposal in `Timelock`	1.0
F18	Accepting admin proposal in `Timelock`	1.0
F19	Setting delay of transactions in `Timelock`	1.0
F20	Cancelling transactions in `Timelock`	1.0
F21	Executing transactions in `Timelock`	1.0

Table 4. Wake fuzzing flows

The following table lists the invariants checked after each flow.

ID	Invariant	Added	Status
IV1	Total sums of all borrowed and supplied assets in the base currency match expected values for the core pool	1.0	Success
IV2	Collateral balance matches expected value in isolated pools	1.0	Success
IV3	Token balances match expected values when interacting with isolated pools	1.0	**Fail** (C1)
IV4	Current admin matches expected address in `Timelock`	1.0	Success
IV5	Delay is set correctly in `Timelock`	1.0	Success
IV6	Transaction emit events with expected values	1.0	**Fail** (M1)
IV7	Tested functions do not revert except where explicitly expected	1.0	**Fail** (M2, M7, M10, L2)

Table 5. Wake fuzzing invariants

B.2. Detectors

This section contains vulnerability and code quality detections from the Wake tool.

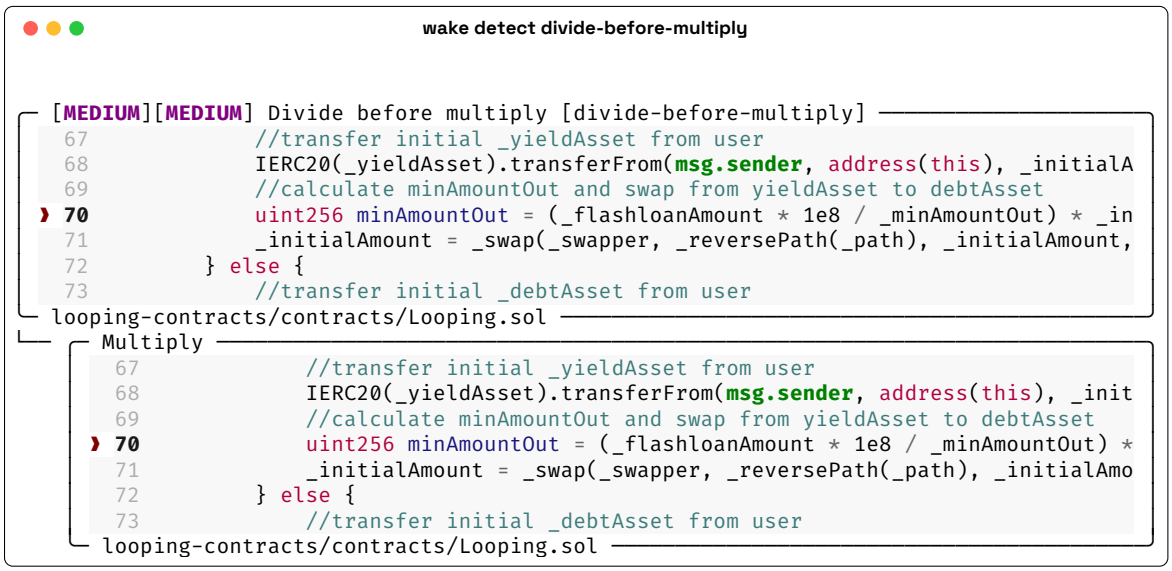


Figure 1. Divide before multiply

Figure 2. Unsafe `transfer` usage

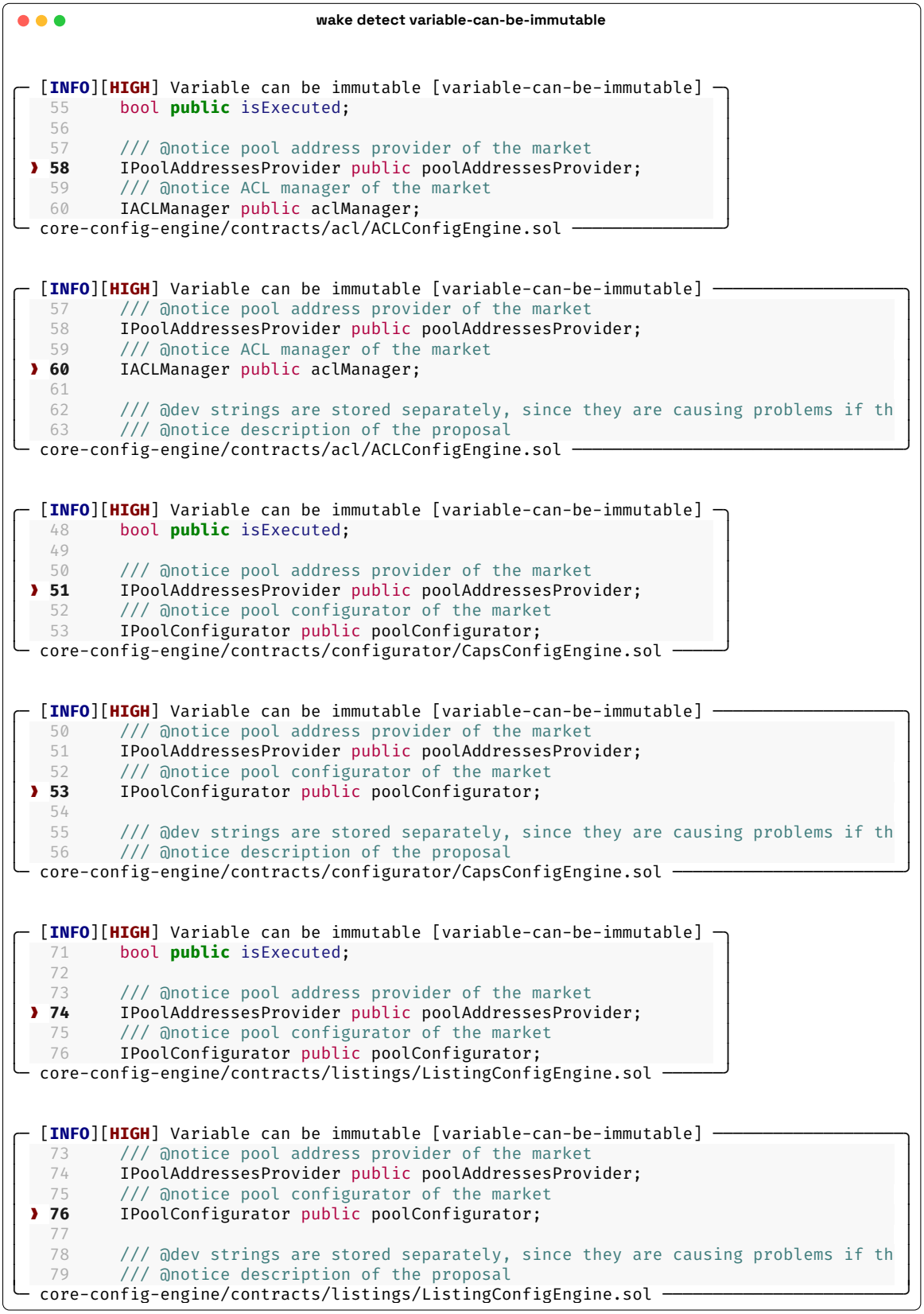


Figure 3. Variables that can be declared as immutable

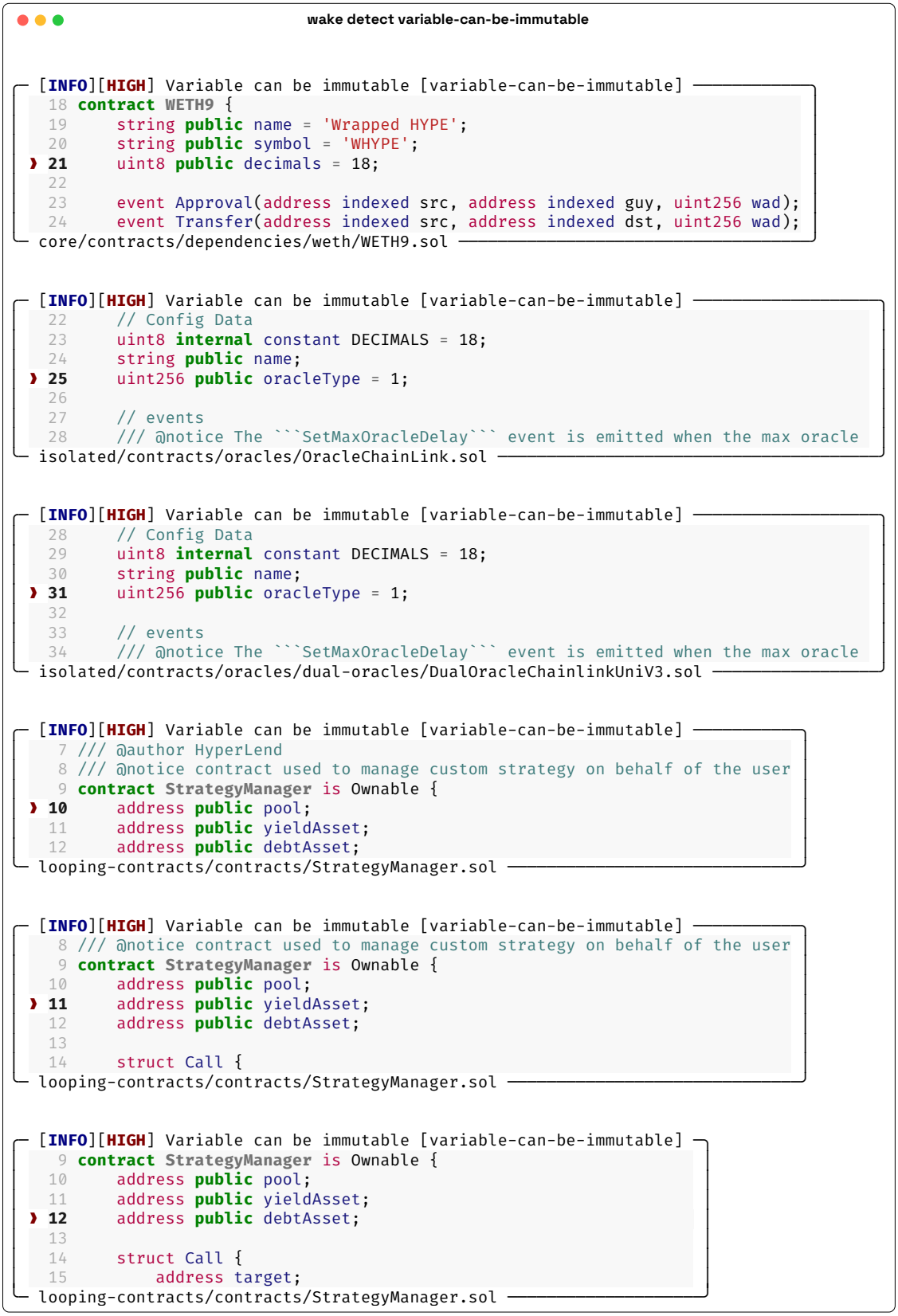


Figure 4. Variables that can be declared as immutable

Figure 5. Usage of deprecated Chainlink API

Figure 6. Unsafe ERC20 calls

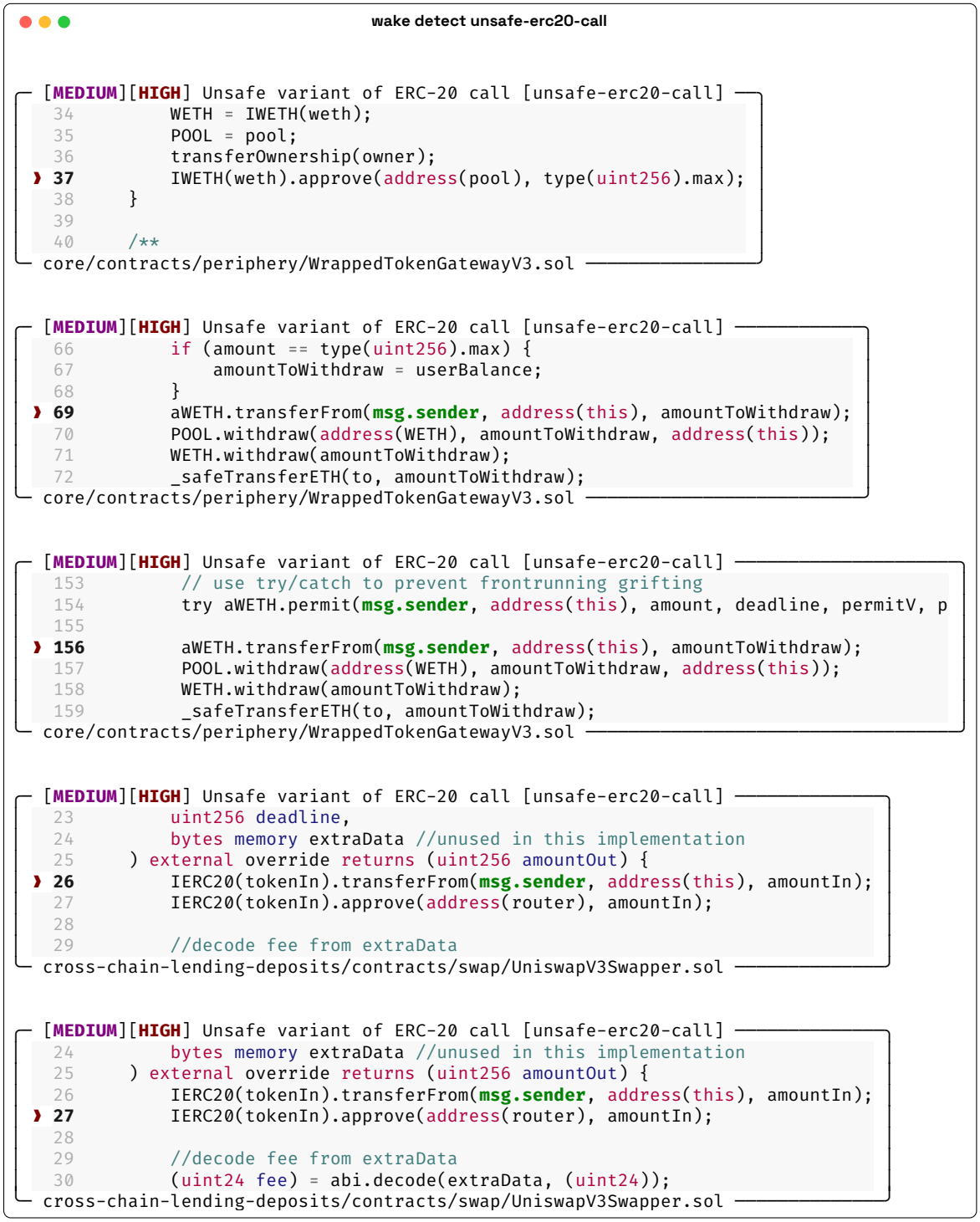


Figure 7. Unsafe ERC20 calls

Figure 8. Unsafe ERC20 calls

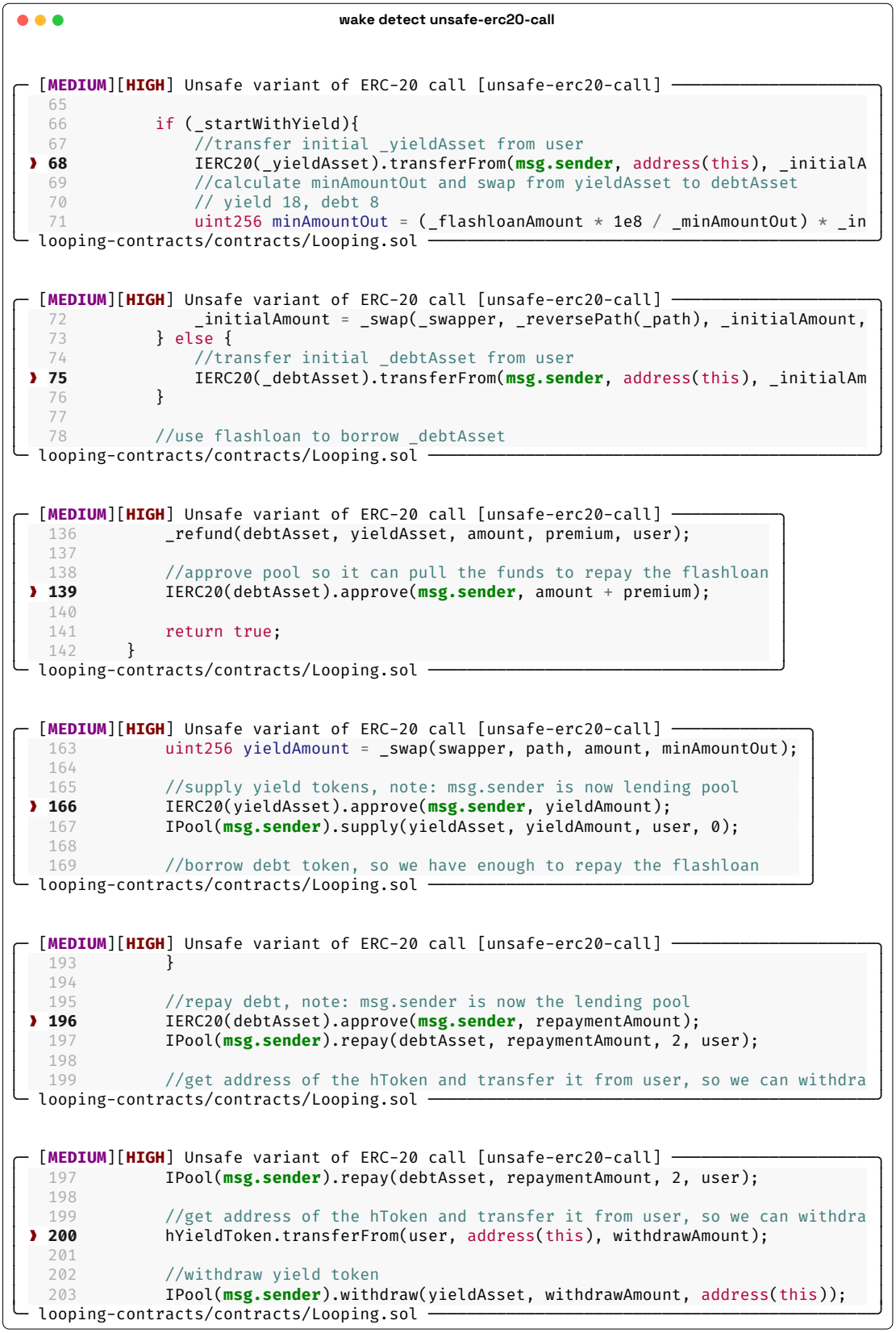


Figure 9. Unsafe ERC20 calls

Figure 10. Unsafe ERC20 calls

Appendix C: Aave Diff with Mainnet

This appendix summarizes the differences between the following repositories:

1. the HyperLend's forked Aave v3.0.2 including the HyperLend's changes:
    - from repository hyperlendx/hyperlend-core;
    - on commit `4256249f94b8762a5ce41caa2283c0d989efc593`; and
2. the mainnet Aave v3.2.0. based on the latest protocol patch:
    - from repository bgd-labs/aave-v3-origin;
    - on commit `181fcd32eb02e83f79902f010153b29136fa00cb`.

C.1. Comparison Methodology

The comparison focuses only on the `contracts/` folder (specifically `src/contracts/` in the Aave v3.2.0 repository) of both repositories.

All contracts were formatted using prettier-plugin-solidity to minimize whitespace changes and other formatting-related changes.

Listing 68. Command `npx` to format the contracts

```
npx prettier --write --plugin=prettier-plugin-solidity 'contracts/**/*.sol'
```

After formatting both versions, the HyperLend's version of the Aave `contracts/` folder was locally committed and replaced with the Aave v3.2.0 `src/contracts/` folder. This replacement was also committed, and the final diff was generated using `git diff`.

The created diff was manually checked while classifying changes into categories and making summarized notes.

All results were documented and summarized in the following sections.

C.2. Results Summary

File diff created according to Comparison Methodology contained 17 137 LOC. Manual review of the resulting diff yielded the following results:

- 35 files contain *major changes* in logic or interface;
- 59 files are new (added by Aave);
- 10 files were removed (8 by Aave, 2 were added by HyperLend); and
- 67 files were renamed, contain only *minor changes* or no changes at all.

More than 94% of all files (163 from 171 reviewed) were different and more than 60% of all files (104 from 171 reviewed) were changed substantially. Notes summarizing the changes in each file consist of 24 pages and are presented in the List of Changes.

Symbol	Definition	Count
[_]	No changes	8
[-]	Only *minor changes*	45
[!]	*Major changes*	28
[M]	File was moved only	2
[M-]	Only *minor changes*, file was moved	12
[M!]	*Major changes*, file was moved	7
[N]	New file	59
[R]	File was removed	10
	Total	**171**

Table 6. Categorisation of File Changes

minor changes := changes in license, pragma, function arg. names, etc.

major changes := changes in logic, interface, constants, etc.

C.3. List of Changes

Each file in the following list is categorized by Table 6.

- dependencies/chainlink/AggregatorInterface.sol `[_]`
- dependencies/gnosis/contracts/GPv2SafeERC20.sol `[-]`
- dependencies/openzeppelin/ReentrancyGuard.sol `[N]` (new in Aave)
- dependencies/openzeppelin/contracts/
  - AccessControl.sol `[-]`
  - Address.sol `[-]`
  - Context.sol `[-]`
  - ERC165.sol `[-]`
  - ERC20.sol `[-]`
  - IAccessControl.sol `[-]`
  - IERC165.sol `[-]`
  - IERC20.sol `[-]`
  - IERC20Detailed.sol `[-]`
  - Ownable.sol `[-]`
  - SafeCast.sol `[-]`
  - SafeERC20.sol `[-]`
  - SafeMath.sol `[-]`
  - Strings.sol `[-]`
- dependencies/openzeppelin/upgradeability/
  - Proxy.sol `[!]`
    - `receive` function added

    - AdminUpgradeabilityProxy.sol `[-]`
    - BaseAdminUpgradeabilityProxy.sol `[-]`
    - BaseUpgradeabilityProxy.sol `[-]`
    - Initializable.sol `[-]`
    - InitializableAdminUpgradeabilityProxy.sol `[-]`
    - InitializableUpgradeabilityProxy.sol `[-]`
    - UpgradeabilityProxy.sol `[-]`
- dependencies/weth/WETH9.sol `[!]` (renamed by Hyperlend)
    - renamed `Wrapped HYPE` ⇒ `Wrapped Ether`
    - renamed `WHYPE` ⇒ `WETH`
- deployments/ReservesSetupHelper.sol `[R]` (removed by Aave)
- deployments/SeedAmountsHolder.sol `[R]` (new in Hyperlend)
- extensions/paraswap-adapters/
    - AaveParaSwapFeeClaimer.sol `[N]` (new in Aave)
    - BaseParaSwapAdapter.sol `[N]` (new in Aave)
    - BaseParaSwapBuyAdapter.sol `[N]` (new in Aave)
    - BaseParaSwapSellAdapter.sol `[N]` (new in Aave)
    - ParaSwapLiquiditySwapAdapter.sol `[N]` (new in Aave)
    - ParaSwapRepayAdapter.sol `[N]` (new in Aave)
    - ParaSwapWithdrawSwapAdapter.sol `[N]` (new in Aave)
    - interfaces/IFeeClaimer.sol `[N]` (new in Aave)
    - interfaces/IParaSwapAugustus.sol `[N]` (new in Aave)
    - interfaces/IParaSwapAugustusRegistry.sol `[N]` (new in Aave)
- extensions/static-a-token/

    - ERC20AaveLMUpgradeable.sol `[N]` (new in Aave)
    - ERC4626StataTokenUpgradeable.sol `[N]` (new in Aave)
    - StataTokenFactory.sol `[N]` (new in Aave)
    - StataTokenV2.sol `[N]` (new in Aave)
    - interfaces/IAToken.sol `[N]` (new in Aave)
    - interfaces/IERC20AaveLM.sol `[N]` (new in Aave)
    - interfaces/IERC4626StataToken.sol `[N]` (new in Aave)
    - interfaces/IStataTokenFactory.sol `[N]` (new in Aave)
    - interfaces/IStataTokenV2.sol `[N]` (new in Aave)
- extensions/v3-config-engine/
    - AaveV3ConfigEngine.sol `[N]` (new in Aave)
    - AaveV3Payload.sol `[N]` (new in Aave)
    - EngineFlags.sol `[N]` (new in Aave)
    - IAaveV3ConfigEngine.sol `[N]` (new in Aave)
    - libraries/BorrowEngine.sol `[N]` (new in Aave)
    - libraries/CapsEngine.sol `[N]` (new in Aave)
    - libraries/CollateralEngine.sol `[N]` (new in Aave)
    - libraries/EModeEngine.sol `[N]` (new in Aave)
    - libraries/ListingEngine.sol `[N]` (new in Aave)
    - libraries/PriceFeedEngine.sol `[N]` (new in Aave)
    - libraries/RateEngine.sol `[N]` (new in Aave)
- helpers/AaveProtocolDataProvider.sol ⇐ misc/ProtocolDataProvider.sol `[M!]` (changed by Aave)
    - all stable debt related parameters deprecated in `v3.2.0`

        - `stableBorrowRateEnabled = false;`
        - `currentStableDebt = principalStableDebt = stableBorrowRate = stableRateLastUpdated = 0;`
        - `reserve.stableDebtTokenAddress = address(0);`
    - removed parameter in `configuration.getFlags();`
    - removed `getReserveEModeCategory`
    - function `getReserveData` return 0 on places where previously were values:
        - `IERC20Detailed(reserve.stableDebtTokenAddress).totalSupply()`
        - `reserve.currentStableBorrowRate`
        - `IStableDebtToken(reserve.stableDebtTokenAddress).getAverageStableRate()`
    - added functions:
        - `getVirtualUnderlyingBalance`
        - `getIsVirtualAccActive`
- helpers/L2Encoder.sol ⇐ misc/L2Encoder.sol `[M!]` (changed by Aave)
    - changes in `encodeWithdrawParams`, `encodeRepayParams`, `encodeRepayWithATokensParams`
        - `interestRateMode` … 2 for Variable, 1 is deprecated (changed on v3.2.0)
    - removed `encodeRebalanceStableBorrowRate`, `encodeSwapBorrowRateMode`
- helpers/UiIncentiveDataProviderV3.sol `[N]` (new in Aave)
- helpers/UiPoolDataProviderV3.sol ⇐ periphery/UiPoolDataProviderV3.sol `[M!]` (changed by Aave)
    - public ⇒ external for `getReservesList` and `getReservesData`

    - changes in `getReservesData`
        - removed try/catch blocks
        - removed `reserveData.reserveFactor`, `reserveData.stableBorrowRateEnabled`
        - removed `reserveData.eMode*` assignments
    - added function `getEModes`
- helpers/WalletBalanceProvider.sol ⇐ periphery/WalletBalanceProvider.sol `[M!]` (changed by Aave)
    - removed parameter in `configuration.getFlags();`
- helpers/WrappedTokenGatewayV3.sol `[M!]` (changed by Aave)
    - function `repayETH` – removed parameter `rateMode`
    - function `borrowETH` – removed parameter `interestRateMode`
    - removed try/catch block around `aWETH.permit` call
- helpers/interfaces/IEACAggregatorProxy.sol ⇐ periphery/interfaces/IEACAggregatorProxy.sol `[M-]` (moved by Aave)
- helpers/interfaces/IERC20DetailedBytes.sol ⇐ periphery/interfaces/IERC20DetailedBytes.sol `[M-]` (moved by Aave)
- helpers/interfaces/IUiIncentiveDataProviderV3.sol `[N]` (new in Aave)
- helpers/interfaces/IUiPoolDataProviderV3.sol ⇐ periphery/interfaces/IUiPoolDataProviderV3.sol `[M!]` (moved by Aave)
    - removed struct `InterestRates`
    - from struct `AggregatedReserveData`
        - removed:
            - `stableBorrowRateEnabled`
            - `stableBorrowRate`

        - `stableDebtTokenAddress`
        - `totalPrincipalStableDebt`
        - `averageStableRate`
        - `stableDebtLastUpdateTimestamp`
        - `stableRateSlope1`
        - `stableRateSlope2`
        - `baseStableBorrowRate`
        - `eMode*`
      - added (in v3.1)
        - `virtualAccActive`
        - `virtualUnderlyingBalance`
    - from struct `UserReserveData` removed:
      - `stableBorrowRate`
      - `principalStableDebt`
      - `stableBorrowLastUpdateTimestamp`
    - added struct `Emode`
    - added function `getEModes`
- helpers/interfaces/IWETH.sol ⇐ misc/interfaces/IWETH.sol `[M-]` (moved by Aave)
- helpers/interfaces/IWrappedTokenGatewayV3.sol ⇐ interfaces/IWrappedTokenGatewayV3.sol `[M!]` (moved by Aave)
    - function `repayETH` – removed parameter `rateMode`
    - function `borrowETH` – removed parameter `interestRateMode`
- instances/ATokenInstance.sol `[N]` (new in Aave)

- instances/L2PoolInstance.sol `[N]` (new in Aave)
- instances/PoolConfiguratorInstance.sol `[N]` (new in Aave)
- instances/PoolInstance.sol `[N]` (new in Aave)
- instances/VariableDebtTokenInstance.sol `[N]` (new in Aave)
- interfaces/IACLManager.sol `[-]`
- interfaces/IAToken.sol `[-]`
- interfaces/IAaveIncentivesController.sol `[-]`
- interfaces/IAaveOracle.sol ⇐ interfaces/IOracle.sol `[M-]` (moved by Hyperlend)
- interfaces/ICreditDelegationToken.sol `[-]`
- interfaces/IDefaultInterestRateStrategy.sol `[R]` (removed by Aave)
- interfaces/IDefaultInterestRateStrategyV2.sol `[N]` (new in Aave)
- interfaces/IDelegationToken.sol `[-]`
- interfaces/IERC20WithPermit.sol `[-]`
- interfaces/IInitializableAToken.sol `[-]`
- interfaces/IInitializableDebtToken.sol `[-]`
- interfaces/IL2Pool.sol `[!]` (changed by Aave)
  - removed function `rebalanceStableBorrowRate`
  - removed function `swapBorrowRateMode`
- interfaces/IPool.sol `[!]` (changed by Aave)
  - event `Borrow` – `interestRateMode` – 2 for Variable, 1 is deprecated (in v3.2.0)
  - event `FlashLoan` – `interestRateMode` – 0 for regular flashloan, 1 is deprecated (in v3.2.0), 2 for Variable

- event `ReserveDataUpdated` – `stableBorrowRate` deprecated (in v3.2.0)
- removed event `SwapBorrowRateMode`
- removed event `RebalanceStableBorrowRate`
- function `flashLoan` arguments:
  - `interestRateMode` – 0 for regular flashloan, 1 is deprecated (in v3.2.0), 2 for Variable
  - `onBehalfOf` – only for using 2 on `modes`
- function `initReserve` – removed argument `stableDebtAddress`
- function `dropReserve` – does not reset eMode flags
- function `getEModeCategoryData` – deprecated
- function `configureEModeCategory` – can alter an existing collateral configuration
- added functions
  - `getReserveDataExtended`
  - `getVirtualUnderlyingBalance`
  - `configureEModeCategoryCollateralBitmap`
  - `configureEModeCategoryBorrowableBitmap`
  - `getLiquidationGracePeriod`
  - `getFlashLoanLogic`
  - `getBorrowLogic`
  - `getBridgeLogic`
  - `getEModeLogic`
  - `getLiquidationLogic`
  - `getPoolLogic`

        - `getSupplyLogic`
- interfaces/IPoolAddressesProvider.sol `[-]`
- interfaces/IPoolAddressesProviderRegistry.sol `[-]`
- interfaces/IPoolConfigurator.sol `[!]` (changed by Aave)
    - function `ReserveInitialized` – argument `stableDebtToken` – deprecated (in v3.2.0)
    - function `setReserveInterestRateStrategyAddress` – added argument `rateData`
    - function `setEModeCategory` – removed argument `oracle`
    - added events
        - `PendingLtvChanged`
        - `LiquidationGracePeriodChanged`
        - `LiquidationGracePeriodDisabled`
        - `AssetCollateralInEModeChanged`
        - `AssetBorrowableInEModeChanged`
        - `ReserveInterestRateDataChanged`
    - removed events
        - `ReserveStableRateBorrowing`
        - `EModeAssetCategoryChanged`
        - `StableDebtTokenUpgraded`
    - deprecated param `oracle` in event `EModeCategoryAdded`
    - added functions
        - `setReservePause` with `gracePeriod` argument
        - `disableLiquidationGracePeriod`

        - `setReserveInterestRateData`
        - `setPoolPause` with `gracePeriod` argument
        - `setAssetBorrowableInEMode`
        - `getPendingLtv`
        - `getConfiguratorLogic`
        - `MAX_GRACE_PERIOD`
    - removed functions
        - `setReserveStableRateBorrowing`
        - `setAssetEModeCategory`
- interfaces/IPoolDataProvider.sol `[!]` (changed by Aave)
    - removed function `getReserveEModeCategory`
    - function `getReserveTokensAddresses` return value `stableDebtTokenAddress` deprecated (in v3.2.0)
    - added functions
        - `getIsVirtualAccActive`
        - `getVirtualUnderlyingBalance`
- interfaces/IPriceOracle.sol `[-]`
- interfaces/IPriceOracleGetter.sol `[-]`
- interfaces/IPriceOracleSentinel.sol `[-]`
- interfaces/IReserveInterestRateStrategy.sol `[!]` (changed by Aave)
    - added function `setInterestRateParams`
    - function `calculateInterestRates` – removed return value `variableBorrowRate`
- interfaces/IScaledBalanceToken.sol `[-]`

- interfaces/ISequencerOracle.sol `[-]`
- interfaces/IStableDebtToken.sol `[R]` (removed by Aave)
- interfaces/IVariableDebtToken.sol `[-]`
- misc/AaveOracle.sol ⇐ misc/Oracle.sol `[M-]` (moved by Hyperlend)
- misc/DefaultReserveInterestRateStrategyV2.sol `[N]` (new in Aave)
- misc/DataTypesHelper.sol `[R]` (new in Hyperlend)
- misc/PriceOracleSentinel.sol ⇐ protocol/configuration/PriceOracleSentinel.sol `[M]` (moved by Aave)
  - public ⇒ external for:
    - `isBorrowAllowed`
    - `isLiquidationAllowed`
    - `setSequencerOracle`
    - `setGracePeriod`
    - `getSequencerOracle`
    - `getGracePeriod`
- misc/ZeroReserveInterestRateStrategy.sol `[R]` (removed by Aave)
- misc/aave-upgradeability/BaseImmutableAdminUpgradeabilityProxy.sol ⇐ protocol/libraries/aave-upgradeability/BaseImmutableAdminUpgradeabilityProxy.sol `[M-]` (moved by Aave)
- misc/aave-upgradeability/InitializableImmutableAdminUpgradeabilityProxy.sol ⇐ protocol/libraries/aave-upgradeability/InitializableImmutableAdminUpgradeabilityProxy.sol `[M-]` (moved by Aave)

- misc/aave-upgradeability/VersionedInitializable.sol ⇐ protocol/libraries/aave-upgradeability/VersionedInitializable.sol `[M-]` (moved by Aave)
- misc/flashloan/base/FlashLoanReceiverBase.sol ⇐ flashloan/base/FlashLoanReceiverBase.sol `[M-]` (moved by Aave)
- misc/flashloan/base/FlashLoanSimpleReceiverBase.sol ⇐ flashloan/base/FlashLoanSimpleReceiverBase.sol `[M-]` (moved by Aave)
- misc/flashloan/interfaces/IFlashLoanReceiver.sol ⇐ flashloan/interfaces/IFlashLoanReceiver.sol `[M-]` (moved by Aave)
- misc/flashloan/interfaces/IFlashLoanSimpleReceiver.sol ⇐ flashloan/interfaces/IFlashLoanSimpleReceiver.sol `[M-]` (moved by Aave)
- protocol/configuration/ACLManager.sol `[_]`
- protocol/configuration/PoolAddressesProvider.sol `[-]`
- protocol/configuration/PoolAddressesProviderRegistry.sol `[_]`
- protocol/libraries/configuration/EModeConfiguration.sol `[N]` (new in Aave)
- protocol/libraries/configuration/ReserveConfiguration.sol `[!]` (changed by Aave)
  - removed constants:
    - `STABLE_BORROWING_MASK`
    - `EMODE_CATEGORY_MASK`
    - `STABLE_BORROWING_ENABLED_START_BIT_POSITION`
    - `EMODE_CATEGORY_START_BIT_POSITION`
    - `MAX_VALID_EMODE_CATEGORY`
  - added constants:
    - `VIRTUAL_ACC_ACTIVE_MASK`

        - `VIRTUAL_ACC_START_BIT_POSITION`
    - removed functions:
        - `setStableRateBorrowingEnabled`
        - `getStableRateBorrowingEnabled`
        - `setEModeCategory`
        - `getEModeCategory`
    - added functions:
        - `setVirtualAccActive`
        - `getIsVirtualAccActive`
    - function `ReserveConfigurationMap` – removed return value flag `stableRateBorrowing`
    - function `getParams` – removed argument `eMode` category
    - function `getReserveConfigurationData` – removed argument `userEModeCategory`
- protocol/libraries/configuration/UserConfiguration.sol `[-]`
- protocol/libraries/helpers/Errors.sol `[_]`
    - removed constants:
        - `STABLE_BORROWING_NOT_ENABLED`
        - `AMOUNT_BIGGER_THAN_MAX_LOAN_SIZE_STABLE`
        - `NO_OUTSTANDING_STABLE_DEBT`
        - `STABLE_DEBT_NOT_ZERO`
        - `INVALID_OPTIMAL_STABLE_TO_TOTAL_DEBT_RATIO`
        - `STABLE_BORROWING_ENABLED`
    - added constants:

        - `INVALID_MAX_RATE`
        - `WITHDRAW_TO_ATOKEN`
        - `SUPPLY_TO_ATOKEN`
        - `SLOPE_2_MUST_BE_GTE_SLOPE_1`
        - `CALLER_NOT_RISK_OR_POOL_OR_EMERGENCY_ADMIN`
        - `LIQUIDATION_GRACE_SENTINEL_CHECK_FAILED`
        - `INVALID_GRACE_PERIOD`
        - `INVALID_FREEZE_STATE`
        - `NOT_BORROWABLE_IN_EMODE`
- protocol/libraries/helpers/Helpers.sol `[R]` (removed by Aave)
- protocol/libraries/logic/BorrowLogic.sol `[!]` (changed by Aave)
    - removed events `RebalanceStableBorrowRate` and `SwapBorrowRateMode`
    - added event `ReserveUsedAsCollateralDisabled`
    - function `executeBorrow` – changed implementation
    - function `executeRepay` – changed implementation
    - removed functions `executeRebalanceStableBorrowRate` and `executeSwapBorrowRateMode`
- protocol/libraries/logic/BridgeLogic.sol `[!]` (changed by Aave)
    - changes calls of other contracts due to changes in `ReserveLogic` and `ValidationLogic`
- protocol/libraries/logic/CalldataLogic.sol `[!]` (changed by Aave)
    - `interestRateMode` … 2 for Variable, 1 is deprecated (changed on v3.2.0) on:
        - `decodeRepayWithPermitParams`

        - `decodeRepayParams`
        - `decodeBorrowParams`
    - removed functions:
        - `decodeSwapBorrowRateModeParams`
        - `decodeRebalanceStableBorrowRateParams`
- protocol/libraries/logic/ConfiguratorLogic.sol `[!]` (changed by Aave)
    - removed event `StableDebtTokenUpgraded`
    - public ⇒ external for:
        - `executeInitReserve` – changed implementation
        - `executeUpdateAToken`
        - `executeUpdateVariableDebtToken`
- protocol/libraries/logic/EModeLogic.sol `[!]` (changed by Aave)
    - function `executeSetUserEMode` – changed implementation
    - removed function `getEModeConfiguration` and `isInEModeCategory`
- protocol/libraries/logic/FlashLoanLogic.sol `[!]` (changed by Aave)
    - struct `FlashLoanLocalVars` – removed `i`
    - function `executeFlashLoan` – changed implementation
    - function `executeFlashLoanSimple` – changed implementation
- protocol/libraries/logic/GenericLogic.sol `[!]` (changed by Aave)
    - struct `CalculateUserAccountDataVars` – removed `eModeAssetPrice` and `eModeAssetCategory`
    - function `calculateUserAccountData` – changed implementation
    - function `calculateAvailableBorrows` – added `=` to comparison
    - function `_getUserDebtInBaseCurrency` – changed implementation

- protocol/libraries/logic/IsolationModeLogic.sol `[_]`
- protocol/libraries/logic/LiquidationLogic.sol `[!]` (changed by Aave)
  - struct `LiquidationCallLocalVars` – removed `userVariableDebt`, `collateralPriceSource` and `debtPriceSource`
  - function `executeLiquidationCall` – changed implementation
  - function `_burnCollateralATokens` – changed implementation
  - function `_calculateDebt` – changed implementation
  - removed function `_getConfigurationData`
- protocol/libraries/logic/PoolLogic.sol `[!]` (changed by Aave)
  - removed `params.stableDebtAddress` from `init` call
  - added function `executeSetLiquidationGracePeriod`
- protocol/libraries/logic/ReserveLogic.sol `[!]` (changed by Aave)
  - function `init` – removed `stableDebtTokenAddress`
  - removed structs `UpdateInterestRatesLocalVars` and `AccrueToTreasuryLocalVars`
  - function `updateInterestRates` renamed to `updateInterestRatesAndVirtualBalance` and changed implementation
  - function `_accrueToTreasury` – changed implementation
  - function `cache` – changed implementation
- protocol/libraries/logic/SupplyLogic.sol `[!]` (changed by Aave)
  - function `executeSupply` – changed implementation
  - function `executeWithdraw` – changed implementation
  - function `executeFinalizeTransfer` – changed implementation
- protocol/libraries/logic/ValidationLogic.sol `[!]` (changed by Aave)

    - function `validateSupply` – added parameter `onBehalfOf`
    - struct `ValidateSupplyLocalVars` – removed `eModePriceSource` and `stableRateBorrowingEnabled`
    - function `validateBorrow` – changed implementation
    - function `validateRepay`
        - removed arguments `stableDebt` and `variableDebt`
        - added argument `debt`
        - changed implementation
    - removed functions `validateSwapRateMode` and `validateRebalanceStableBorrowRate`
    - function `validateFlashloanSimple` – added parameter `amount` and extended validation
    - function `validateLiquidationCall` – added parameter `debtReserve` and extended validation
    - function `validateDropReserve` – removed require with `stableDebtTokenAddress`
    - function `validateSetUserEMode` – removed arguments `reservesData` and `reservesList`, changed implementation
- protocol/libraries/math/MathUtils.sol `[_]`
- protocol/libraries/math/PercentageMath.sol `[_]`
- protocol/libraries/math/WadRayMath.sol `[_]`
- protocol/libraries/types/ConfiguratorInputTypes.sol `[!]` (changed by Aave)
    - struct `InitReserveInput`
        - removed `stableDebtTokenAddress`

        - removed `underlyingAssetDecimals`
        - removed `stableDebtTokenName`
        - removed `stableDebtTokenSymbol`
        - added `useVirtualBalance`
        - added `interestRateData`
- protocol/libraries/types/DataTypes.sol `[!]` (changed by Aave)
    - struct `ReserveData` renamed to `ReserveDataLegacy`
    - struct `EModeCategory` renamed to `EModeCategoryLegacy`
    - added structs
        - `ReserveData`
        - `EModeCategory`
        - `CollateralConfig`
        - `EModeCategoryBaseConfiguration`
    - struct `ExecuteBorrowParams` – removed `maxStableRateBorrowSizePercent`
    - struct `FlashloanParams` – removed `maxStableRateBorrowSizePercent`
    - struct `ValidateBorrowParams` – removed `maxStableLoanPercent`
    - struct `ValidateBorrowParams` – removed `maxStableLoanPercent`
    - struct `ReserveCache` – removed:
        - `currPrincipalStableDebt`
        - `currAvgStableBorrowRate`
        - `currTotalStableDebt`
        - `nextAvgStableBorrowRate`
        - `nextTotalStableDebt`
        - `stableDebtTokenAddress`

        - `stableDebtLastUpdateTimestamp`
    - struct `CalculateInterestRatesParams`
        - removed `totalStableDebt`
        - removed `totalVariableDebt`
        - removed `averageStableBorrowRate`
        - removed `aToken`
        - added `totalDebt`
        - added `usingVirtualBalance`
        - added `virtualUnderlyingBalance`
- protocol/pool/DefaultReserveInterestRateStrategy.sol `[R]` (removed by Aave)
- protocol/pool/L2Pool.sol `[!]` (changed by Aave)
    - contract `L2Pool` made abstract
    - removed functions `swapBorrowRateMode` and `rebalanceStableBorrowRate`
- protocol/pool/Pool.sol `[!]` (changed by Aave)
    - contract `Pool` made abstract
    - function `initialize` made virtual
    - function `getReserveData` – renamed to `getReserveDataExtended`
    - function `initReserve` – removed argument `stableDebtAddress`
    - function `configureEModeCategory` – changed `category` type
    - function `getEModeCategoryData` – changed implementation
    - added functions:
        - `getReserveData`
        - `getVirtualUnderlyingBalance`

        - `configureEModeCategoryCollateralBitmap`
        - `configureEModeCategoryBorrowableBitmap`
        - `getEModeCategoryCollateralConfig`
        - `getEModeCategoryLabel`
        - `getEModeCategoryCollateralBitmap`
        - `getEModeCategoryBorrowableBitmap`
        - `getFlashLoanLogic`
        - `getBorrowLogic`
        - `getBridgeLogic`
        - `getEModeLogic`
        - `getLiquidationLogic`
        - `getPoolLogic`
        - `getSupplyLogic`
    - removed functions:
        - `getRevision`
        - `swapBorrowRateMode`
        - `rebalanceStableBorrowRate`
        - `MAX_STABLE_RATE_BORROW_SIZE_PERCENT`
- protocol/pool/PoolConfigurator.sol `[!]` (changed by Aave)
    - contract `PoolConfigurator` made abstract
    - function `initialize` made virtual
    - added mapping `_pendingLtv`
    - added constant `MAX_GRACE_PERIOD`
    - added modifier `onlyRiskOrPoolOrEmergencyAdmins`

- added functions:
  - `setReservePause`
  - `disableLiquidationGracePeriod`
  - `setReserveInterestRateData`
  - `getPendingLtv`
  - `getConfiguratorLogic`
  - `_updateInterestRateStrategy`
  - `_onlyRiskOrPoolOrEmergencyAdmins`
- removed modifier `onlyEmergencyAdmin`
- removed functions:
  - `updateStableDebtToken`
  - `updateStableDebtToken`
  - `setReserveStableRateBorrowing`
  - `_onlyEmergencyAdmin`
- changed implementation of functions:
  - `setReserveBorrowing`
  - `configureReserveAsCollateral`
  - `setReserveFreeze`
  - `setReservePause`
  - `setDebtCeiling`
  - `setEModeCategory`
  - `setAssetBorrowableInEMode`
  - `setReservePause`
  - `_checkNoSuppliers`

    - function `setAssetEModeCategory`
        - removed argument `allowed`
        - changed implementation
    - function `setAssetEModeCategory` renamed to `setAssetCollateralInEMode`
        - added argument `allowed`
        - changed implementation
    - function `setReserveInterestRateStrategyAddress`
        - added argument `rateData`
        - changed implementation
    - function `setPoolPause`
        - added argument `gracePeriod`
        - changed implementation
- protocol/pool/PoolStorage.sol `[!]` (changed by Aave)
    - renamed `_maxStableRateBorrowSizePercent` to `__DEPRECATED_maxStableRateBorrowSizePercent`
- protocol/tokenization/AToken.sol `[!]` (changed by Aave)
    - renamed `HTOKEN_IMPL` ⇒ `ATOKEN_IMPL [M]` (renamed by Hyperlend)
    - contract `AToken` made abstract
    - removed function `getRevision`
    - removed implementation of `initialize`
- protocol/tokenization/DelegationAwareAToken.sol `[R]` (removed by Aave)
- protocol/tokenization/StableDebtToken.sol `[R]` (removed by Aave)
- protocol/tokenization/VariableDebtToken.sol `[!]` (changed by Aave)
    - contract `VariableDebtToken` made abstract

    - removed constants `DEBT_TOKEN_REVISION`
    - removed implementation of `initialize`
- protocol/tokenization/base/DebtTokenBase.sol `[-]`
- protocol/tokenization/base/EIP712Base.sol `[-]`
- protocol/tokenization/base/IncentivizedERC20.sol `[-]`
- protocol/tokenization/base/MintableIncentivizedERC20.sol `[-]`
- protocol/tokenization/base/ScaledBalanceTokenBase.sol `[-]`
- rewards/
    - EmissionManager.sol `[N]` (new in Aave)
    - RewardsController.sol `[N]` (new in Aave)
    - RewardsDistributor.sol `[N]` (new in Aave)
    - interfaces/IEmissionManager.sol `[N]` (new in Aave)
    - interfaces/IPullRewardsTransferStrategy.sol `[N]` (new in Aave)
    - interfaces/IRewardsController.sol `[N]` (new in Aave)
    - interfaces/IRewardsDistributor.sol `[N]` (new in Aave)
    - interfaces/IStakedToken.sol `[N]` (new in Aave)
    - interfaces/IStakedTokenTransferStrategy.sol `[N]` (new in Aave)
    - interfaces/ITransferStrategyBase.sol `[N]` (new in Aave)
    - libraries/RewardsDataTypes.sol `[N]` (new in Aave)
    - transfer-strategies/PullRewardsTransferStrategy.sol `[N]` (new in Aave)
    - transfer-strategies/StakedTokenTransferStrategy.sol `[N]` (new in Aave)
    - transfer-strategies/TransferStrategyBase.sol `[N]` (new in Aave)
- treasury/
    - Collector.sol `[N]` (new in Aave)

- ICollector.sol [N] (new in Aave)
- IRevenueSplitter.sol [N] (new in Aave)
- RevenueSplitter.sol [N] (new in Aave)

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz