SHERLOCK

Sherlock Security Review For Velodrome Superchain

Collaborative audit prepared for: **Velodrome Superchain**
Lead Security Expert(s): **xiaoming90 unforgiven AkshaySrivastav GiuseppeDeLaZara**
Date Audited: **October 11th - October 25th**

Introduction

Velodrome Finance is a next-generation AMM that combines the best of Curve, Convex and Uniswap, designed to serve as the liquidity hub for the Superchain.
Velodrome NFTs vote on token emissions and receive incentives and fees generated by the protocol.

Scope

Velodrome is a DEX on Optimism. You can read more about the protocol and the mechanics at:

https://velodrome.finance/docs

The Superchain release of the Velodrome is designed to support the expansion of the OP Stack network of chains with the goal of servicing these native chains with

- constant product liquidity pools
- concentrated liquidity pools / Slipstream
- (universal) router
- gauges for specific pools
- token registry used for keeping track of the tokens allowed to be used as incentives
- Hyperlane powered bridges / interop for
- the XERC20 VELO token (emissions)
- the root → leaf gauge messaging

Codebase

Core Interoperability + v2 Pools + Gauges

The Superchain repository includes new contracts and changes for the previously audited implementations.

https://github.com/velodrome-finance/superchain-contracts

Everything in the above repository, except for the modified, but previously audited contracts (see per folder diffs):

- src/pools

  ```
  $ git diff --stat dbb3cbcd28bbb3b5e693e47a31496ffed8e73adb...main src/pools/
  ```

- src/gauges

  ```
  $ git diff --stat bf1793476eea9464b378fcc3bc0190f58bff76fe...main src/gauges
  ```

- src/rewards

  ```
  $ git diff -R --stat main...29d79b2f src/rewards/
  ```

- src/Router.sol

  ```
  $ git diff 5094f4f197592985f35375c4ab7f0eb5916834c7...main src/Router.sol
  ```

- src/libraries/rateLimits and src/xerc20/MintLimits.sol

  ```
  $ git diff -R main...daa0703 src/libraries/rateLimits/
  src/xerc20/MintLimits.sol
  ```

Slipstream / Concentrated Liquidity Pools + Gauges

There are a few commits that we've added since the first audit. We'd like to include these changes in the audit along with the additions to support the Superchain release.

https://github.com/velodrome-finance/superchain-slipstream/compare/55b677900751f89566fb19d72d091cc101a9d28b...main

Clean changelog for the above.

```
$ git diff -R --stat main...0684cf9 ':!contracts/test' ':!contracts/core/test'
↪  ':!contracts/periphery/test' ':!contracts/periphery/LpMigrator.sol'
```

Findings

Each issue has an assigned severity:

- Medium issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- High issues are directly exploitable security vulnerabilities that need to be fixed.
- Low/Info issues are non-exploitable, informational findings that do not pose a security risk or impact the system's integrity. These issues are typically cosmetic or related to compliance requirements, and are not considered a priority for remediation.

Issues found

Low/Info	Medium	High
7	7	0

Issues not fixed or acknowledged

Low/Info	Medium	High
0	0	0

Issue M-1: Changing the bridging modules can cause loss of in-flight cross-chain messages

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/13

Summary

Changing the `module` state parameter of `RootMessageBridge` and `LeafMessageBridge` contracts can cause the delivery of in-flight messages to always revert.

Vulnerability Detail

The `RootMessageBridge` and `LeafMessageBridge` contracts have the ability to change the `module` parameter via `setModule` functions.

RootMessageBridge: https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/bridge/CrossChainRegistry.sol#L45-L50

LeafMessageBridge: https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/bridge/LeafMessageBridge.sol#L25-L29

During the module change, in case there exists a message which got transmitted from the root chain but haven't arrived on leaf chain then that in-flight message cannot be delivered on leaf chain. Delivery of that message will revert because all leaf chain operations validate that they are being called by latest leaf module.
https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/gauges/LeafGauge.sol#L192

Hence delivery of that in-flight message will revert on leaf chain.

Scenario:

1. A cross chain operation gets initiated via `RootMessageBridge.sendMessage`.
2. Owner of `RootMessageBridge` and `LeafMessageBridge` contracts changes the modules on both root and leaf chains.
3. The message from step 1 arrives to leaf chain but gets reverted.

This scenario can occur in real life when Velodrome decides to upgrade their currently active bridge modules.

Impact

Atomicity of cross-chain messages will get broken when a module upgrade is in progress. The messages will get executed on root chain but will fail on leaf chain.

Recommendation

It should be made sure that there are no in-flight messages during a module change. There are cases when messages cannot be executed instantly on leaf chains (like messages waiting due to leaf chain's XERC20 buffer limits), those cases should also be taken care of.

One way of resolving this could be to have a message pause mechanism on `RootMessageBridge` so that the message bridging can be paused earlier to a module change upgrade.

Discussion

simplyoptimistic

Status: Won't fix

Comments: The upgrade process will be structured in a way that minimizes disruption to users. This can be done by tactically choosing time periods where user actions are minimal (e.g. away from epoch flip, where the majority of voting and reward claim transactions are submitted). Upgrades can also be done such that the root is upgraded first, allowing time for in-flight transactions to process before setting the module on the leaf chain.

It is also possible to deregister the chain temporarily, which would allow in-flight transactions to process while blocking any new transactions.

Issue M-2: Hardcoded gas for bridge transactions

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/9

Summary

Code sets static hardcoded gas for all the bridge messages but the issue is that commands GET_INCENTIVES and GET_FEES consumes dynamic amount of gas based on unclaimed epochs and total epochs so leading to stuck messages, i.e. needing manual intervention to deliver the message.

Vulnerability Detail

Commands GET_INCENTIVES and GET_FEES are used to collect fee and incentives from IncentiveVotingReward and FeesVotingReward in the leaf chains for users. The amount of gas required for those transactions in the leaf chains depends on the unclaimed epochs of the user and total snapshot counts as code loops through all the unclaimed epochs and also perform a binary search inside all the snapshots. Also user can specify multiple tokens (up to 5) in those commands so the gas consumption may be differ based on number of tokens user specifies. In the current configurations, code uses 440K and 220K gas for those commands:

```
if (_command == Commands.GET_INCENTIVES) return 440_000;
if (_command == Commands.GET_FEES) return 220_000;
```

These hardcoded gas amounts won't be enough to perform transactions on the leaf chain if users didn't claimed their rewards for some times.

Impact

Users need to pay for additional gas in a separate transaction in order to deliver their message to the destination chain.

Recommendation

Allow the user to pay for additional gas.

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/d76ce39a6e44cf20c9adf353e099d5a1a960bdad

Comments: The fix will involve benchmarking get_incentives / get_fees for a 2 month period for 5 tokens. This should be sufficient for most use cases. It should be noted that the vast majority of users claim rewards on a weekly basis.

Furthermore, as a fallback option, there is the option to pay for extra gas via hyperlane, or to reduce the batch size of the rewards.

0xunforgiven

Fix confirmed

Issue M-3: XERC20 does not work on Superchain due to outdated Superchain's interfaces

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/6

Summary

The Superchain's interfaces and events used within the `XERC20` are outdated. As a result, existing XVELO will not work on Superchain in its current form.

Vulnerability Detail

The Superchain's interfaces and events used within the in-scope `XERC20` are outdated.

In-scope XERC20	Superchain's Spec
__crosschainMint(address _to, uint256 _amount)	crosschainMint(address _account, uint256 _amount)
__crosschainBurn(address _from, uint256 _amount)	crosschainBurn(address _account, uint256 _amount)
emit CrosschainMinted(address indexed _to, uint256 _amount);	event CrosschainMint(address indexed _to, uint256 _amount)
emit CrosschainBurnt(address indexed _from, uint256 _amount);	event CrosschainBurn(address indexed _from, uint256 _amount)

https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/xerc20/XERC20.sol#L173

```
File: XERC20.sol
172:     /// @inheritdoc ICrosschainERC20
```

```
173:     function __crosschainMint(address _to, uint256 _amount) external
 ↪  onlySuperchainERC20Bridge {
174:         _depleteBuffer(msg.sender, _amount);
175:         _mint(_to, _amount);
176:
177:         emit CrosschainMinted(_to, _amount);
178:     }
179:
180:     /// @inheritdoc ICrosschainERC20
181:     function __crosschainBurn(address _from, uint256 _amount) external
 ↪  onlySuperchainERC20Bridge {
182:         _spendAllowance(_from, msg.sender, _amount);
183:         _replenishBuffer(msg.sender, _amount);
184:         _burn(_from, _amount);
185:
186:         emit CrosschainBurnt(_from, _amount);
187:     }
```

Impact

The existing XVELO will not work on Superchain in its current form.

Recommendation

Consider updating the XERC20's interfaces and events to the latest Superchain Specification.

Per Optimism's documentation as of 24 October 2024:

> Interop is currently in active development and not yet ready for production use. The information provided here may change. Check back regularly for the most up-to-date information.

Thus, it is recommended to verify the finality of the SuperchainERC20 Token Standard again before XVELO deployment to ensure that there are no further changes expected from the Optimism team since XVELO is immutable once deployed.

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/8eb7fd41912a5fb026ea248f655c470502c532b5

Comment: Fix updates the interface. We have been informed that the interface is now frozen and will no longer be changed.

xiaoming9090

Fix confirmed

Issue M-4: Upgradability of Token Bridge leading to integration issue

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/5

Summary

The `TokenBridge` address will change when migrating from Hyperlane to Superchain. As a result, external integrators might not be able to bridge XVELO.

Vulnerability Detail

It is announced that Velodrome will migrate from Hyperlane to Superchain in the future. However, the `TokenBridge` is immutable and not upgradable. As a result, to facilitate the migration from Hyperlane to Superchain, a new `TokenBridge` contract with the new Superchain's logic has to be deployed, which will inevitably lead to a new bridge address.

As a result, external protocols that integrate with the `TokenBridge` and hardcode the bridge address OR the external protocol itself is immutable will encounter issues when the bridge address changes.

Impact

External integrators might not be able to bridge XVELO if the `TokenBridge` address changes due to migration to the Superchain bridge and the older Hyperplane's token bridge has been decommissioned.

Recommendation

Consider any of the following mitigation actions:

1. Update the `TokenBridge` to be upgradable so that the bridge logic can be updated without changing the bridge address
2. Consider documenting that the token bridge address will change when migrating from Hyperlane to Superchain in the future so that the integrators are aware of this.

Discussion

simplyoptimistic

Status: Won't fix

Comment: We will add additional documentation stating that the token bridge may be upgraded in the future. We will maintain the same interface, but will note that the bridge address may change.

Issue M-5: Token bridges process messages from unsupported chains

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/4

Summary

Messages sent from unsupported chains will be processed by the Token Bridge, which might lead to unauthorized minting of XVELO if they are malicious.

Vulnerability Detail

The Token Bridge was found not to restrict messages from only supported chains. As a result, it might lead to the following issues:

- When the admin removes a supported chain from the protocol via the `deregisterChain` function, the Token Bridge will still continue to process messages sent from the unsupported chains.
- Although It is unlikely that the attacker will be able to deploy a `TokenBridge` with the same address on another chain because they do not have Velodrome's deployer's private key, however, since Hyperlane supports 60+ chains, there might be a possibility that one of them has some bugs that attackers can exploit to obtain the same address and perform authorized minting of XVELO. Thus, the Token Bridge should only process messages from chains that are approved by the protocol team and are considered to be secured.

https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/bridge/TokenBridge.sol#L72

```
File: TokenBridge.sol
71:     /// @inheritdoc IHLHandler
72:     function handle(uint32 _origin, bytes32 _sender, bytes calldata _message)
 ↪  external payable {
73:         if (msg.sender != mailbox) revert NotMailbox();
74:         if (_sender != TypeCasts.addressToBytes32(address(this))) revert
 ↪  NotBridge();
75:
76:         (address recipient, uint256 amount) = _message.recipientAndAmount();
77:
78:         IXERC20(xerc20).mint({_user: recipient, _amount: amount});
```

Impact

Messages sent from unsupported chains will be processed by the Token Bridge, which might lead to unauthorized minting of XVELO if they are malicious.

Recommendation

Consider only processing messages from supported chains.

```
function handle(uint32 _origin, bytes32 _sender, bytes calldata _message) external
 ↪  payable {
    if (msg.sender != mailbox) revert NotMailbox();
    if (_sender != TypeCasts.addressToBytes32(address(this))) revert NotBridge();
+   if (!_chainids.contains({value: _origin})) revert UnsupportedChain();
    (address recipient, uint256 amount) = _message.recipientAndAmount();

    IXERC20(xerc20).mint({_user: recipient, _amount: amount});
```

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/28ab0f665e15a21f139877cb969061084c7f12cf

xiaoming9090

Fix confirmed

Issue M-6: Token bridging with Hyperlane doesn't refund the user and might require paying additional gas to be delivered

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/2

Summary

Token bridging through Hyperlane doesn't allow the user to specify the gas limit for message delivery on the destination chain leading to stuck messages, i.e. needing manual intervention to deliver the message.

There is also no refund mechanism for sending `msg.value` that is higher than the Hyperlane fee as default hooks for Hyperlane don't refund.

Vulnerability Detail

TokenBridge.sendToken(...) function calls `Mailbox.dispatch(...)` function with default parameters, i.e. doesn't allow specifying additional gas for message delivery. The default gasLimit is 50k and this might not be sufficient to execute the `handle` function on the destination chain and bridge the tokens. The only option the user has is paying for additional gas through InterchainGasPaymaster.payForGas(...) function call. This is not ideal as user needs to monitor the delivery of his message and in case it can't be delivered send another transaction to pay for the additional gas.

Aside from this, Hyperlane's default hook doesn't allow any gas refunds. If the `msg.value` is higher than needed it's not going to be refunded.

Impact

Users sending `msg.value` higher than needed are not getting refunded and they might need to pay for additional gas in a separate transaction in order to deliver their message to the destination chain.

Recommendation

In order to solve the issues surfaced above:

- Allow the user to pay for additional gas by using the dispatch function overload that allows specifying StandardHookMetadata.
- Quote the dispatch fee using `Mailbox.quoteDispatch(...)`and refund the excess value to the user.

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/e019c513aed632f2097ae2158c1ab84e87c1b5c7

Comment: The fix applied involves specifying a higher gas limit for token bridging, and refunding any excess gas over the quoted dispatch fee.

windhustler

Fix looks good.

Issue M-7: Hyperlane fee is always under-quoted in `RootMessageBridge` contract resulting in permanent DoS

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/1

Summary

Hyperlane fee that is quoted and extracted from the sender while calling `RootMessageBridge.sendMessage(...)` doesn't take into account the additional `gasLimit` specified in the `RootHLMessageModule` contract. This results in the permanent DoS of the functionality as it's not possible to pay for the required fee.

Vulnerability Detail

RootMessageBridge.sendMessage(...) function determines the Hyperlane fee without considering the enforced `gasLimit` specified in the RootHLMessageModule for each action. As a result fee passed and extracted from the sender will always correspond to the message delivery with the default `gasLimit`(50k) while the actual message fee should be paid for much higher gas limits.

See the following PoC:

```
function testQuoteDispatch() public {
        IMailbox mailbox = IMailbox(0xd4C1905BB1D26BC93DAC913e13CaCC278CdCC80D);
        vm.createSelectFork("https://mainnet.optimism.io", 126570298);
        uint256 fee = mailbox.quoteDispatch(43114,
↪  TypeCasts.addressToBytes32(address(this)), "");
        console.log("fee: ", fee);

        bytes memory _metadata =  StandardHookMetadata.formatMetadata({
            _msgValue: 0,
            _gasLimit: 2_000_000,
            _refundAddress: address(mailbox),
            _customMetadata: ""
        });

        fee = mailbox.quoteDispatch(43114,
↪  TypeCasts.addressToBytes32(address(this)), "", _metadata);
        console.log("fee: ", fee);
    }
```

`Mailbox.dispatch(...)` function call with additional `gasLimit` specified in the `defaultHookMetadata` will always require a higher fee.

Impact

As the fee is not enough to cover the cost for `Mailbox.dispatch(...)` this will result in the permanent DoS of `RootMessageBridge.sendMessage(...)` function.

Recommendation

Fee extracted should always be quoted with the same parameters that are passed to the actual `dispatch` function call. Mailbox.quoteDispatch() function that allows passing `defaultHookMetadata` should be used for quoting.

Discussion

simplyoptimistic

Status: Fixed

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/ade3f91854ace8f96201f347fddf407684b1bde0

windhustler

FIx looks good.

Issue L-1: Lack of protection against unexpected high relay fees charged by modules

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/14

Summary

`RootMessageBridge` contract lacks any mechanism to protect users from getting charged an unexpectedly high cross-chain message relay fees.

Vulnerability Detail

The `RootMessageBridge.sendMessage` takes a relay fee from `tx.origin` account in `WETH` tokens for every cross-chain operation. The fee amount is determined by the returned value of `module.quote` function call.

https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/root/bridge/RootMessageBridge.sol#L81-L86

It can be observed that any fee amount returned by the module gets pulled from the `tx.origin` account in WETH tokens. The txn originator has no control over the amount of relay fees that will be charged to it.

Considering the possibility of multiple module<->chain support in `RootMessageBridge`, any bug/exploit in module can have significant impact on `tx.origin`'s WETH token balance.

Sophisticated users can overcome this by only approving a max-acceptable WETH amount to `RootMessageBridge`. But this requires separate limited approval transaction for every cross-chain operation. It is likely that most users (EOAs/Contracts) won't be aware of this behaviour and will simply choose to go with one time WETH approval, making them susceptible to unexpected fees.

Impact

Unexpectedly high fees amount can be charged from users who perform cross-chain operations via `RootMessageBridge`.

Recommendation

Ideally there should be a `maxFee` parameter which is chosen by `tx.origin` and is passed with all cross-chain calls, by which `tx.origin` can control the max relay fee that can be charged to it for every individual cross-chain call.

Discussion

simplyoptimistic

Status: Won't fix

Comments: We will include additional documentation on this issue for integrators. As for users, we will highlight this issue, but it must be noted that there is already an existing multi-step flow for performing most actions on Velodrome, so users will be familiar with the process and not burdened by one additional step.

Issue L-2: RootHLMessageModule does not include the dispatch cost of `sendingNonce` bytes

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/12

Summary

In case of `DEPOSIT` & `WITHDRAW` operations `RootHLMessageModule` passes more bytes data to `Mailbox.dispatch` than what was passed to `Mailbox.quoteDispatch` function, leading to inconsistency in quoted and actual cross-chain message relay cost.

Vulnerability Detail

For `DEPOSIT` & `WITHDRAW` operations the `RootHLMessageModule.quote` function does not include the `sendingNonce` bytes while calculating dispatch cost. But the `RootHLMessageModule.sendMessage` function appends `sendingNonce` bytes to the original `_message` data before calling the `Mailbox.dispatch`.

https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/root/bridge/hyperlane/RootHLMessageModule.sol#L38-L44 https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/root/bridge/hyperlane/RootHLMessageModule.sol#L54-L65

There are blockchains that charge `gas` based upon the length of transaction input bytes. Greater the transaction input greater is the gas cost. Considering that, the `RootHLMessageModule` seems to be under-quoting the deposit/withdraw operation relay cost.

Impact

This implementation creates inconsistency between quotation data and actual relay data. In extreme scenarios this may lead to relayers not relaying cross chain calls due to underpaid fees. Users will need to pay additional fee or relay the message themselves to resolve the issue.

Recommendation

Ideally `RootHLMessageModule` contract should pass exactly the same data for quotation as they pass in the actual message dispatch.

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/598b85ee79da3835b60a4047fc8edb0294b0abee

akshaysrivastav

Fix confirmed

Issue L-3: On root chain XERC20Lockbox can be created for any ERC20 token by anyone

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/11

Summary

Anyone can call `XERC20Factory.deployXERC20WithLockbox` function with any `_erc20` token address as input which leads to an incorrect lockbox deployment.

Vulnerability Detail

On root chain an `XERC20Lockbox` contract is intended to be deployed by `XERC20Factory`. The factory's `deployXERC20WithLockbox` function can only be executed once.

https://github.com/sherlock-audit/2024-10-velodrome/blob/3fd863a33b963e3bdf76a1c871e6a6ecd7c5403d/superchain-contracts-private/src/xerc20/XERC20Factory.sol#L88-L102

This `deployXERC20WithLockbox` function takes the `_erc20` param as input but never validates the input value. As the function is publicly accessible anyone can call this function with any random/malicious token address value as soon as the `XERC20Factory` gets deployed.

Impact

This results in an `XERC20Lockbox` contract deployment with an incorrect `ERC20` token address. This causes minor inconvenience to protocol team as they will need to redeploy the `XERC20Factory` on root chain.

Recommendation

The ERC20 token address (Velo) can be stored as an `immutable` variable in `XERC20Factory`. Then `deployXERC20WithLockbox` function will not need any input parameters.

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/de41ed1ec27f5c8b8330c394b096d036ba224090

akshaysrivastav

Fix confirmed

Issue L-4: Adjusting the buffer cap may result in temporary DoS

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/10

Summary

Max cap most be equal to all the chains all the time otherwise some bridge transactions will fail. So if admins changes the max cap by calling `setBufferCap()` for one of the chains then some on the fly transactions may be blocked.

Vulnerability Detail

XERC20 has max cap for bridged token amounts that doesn't allow sending or receiving tokens more than the max cap. The max cap should be equal for all the chains all the time, otherwise users can bridge tokens in the chain with bigger max cap to the chain with lower max cap and that transaction will revert in destination chain. Admins can change the max cap for a chain's bridge by calling `setBufferCap()`. The issue is that if admins tries to reduce to max cap for all the chains because the process can't be done at the same time for all the chains so there would be moments where max cap isn't equal for all the chains and if a user send a tokens from a chain that has bigger max cap to chain that has lower max cap then that transaction would fail and tokens would stuck in the bridge.

Impact

Bridged token may stuck if max cap get lowered.

Recommendation

Admins should be careful about reducing the max cap in big steps to reduce the probability of the issue. To make sure issue would never happen, allow receiving tokens even if it's bigger than max cap.

Discussion

simplyoptimistic

Status: Won't fix.

Comments: Won't fix, but noted.

Issue L-5: XVELO delivery might be delayed

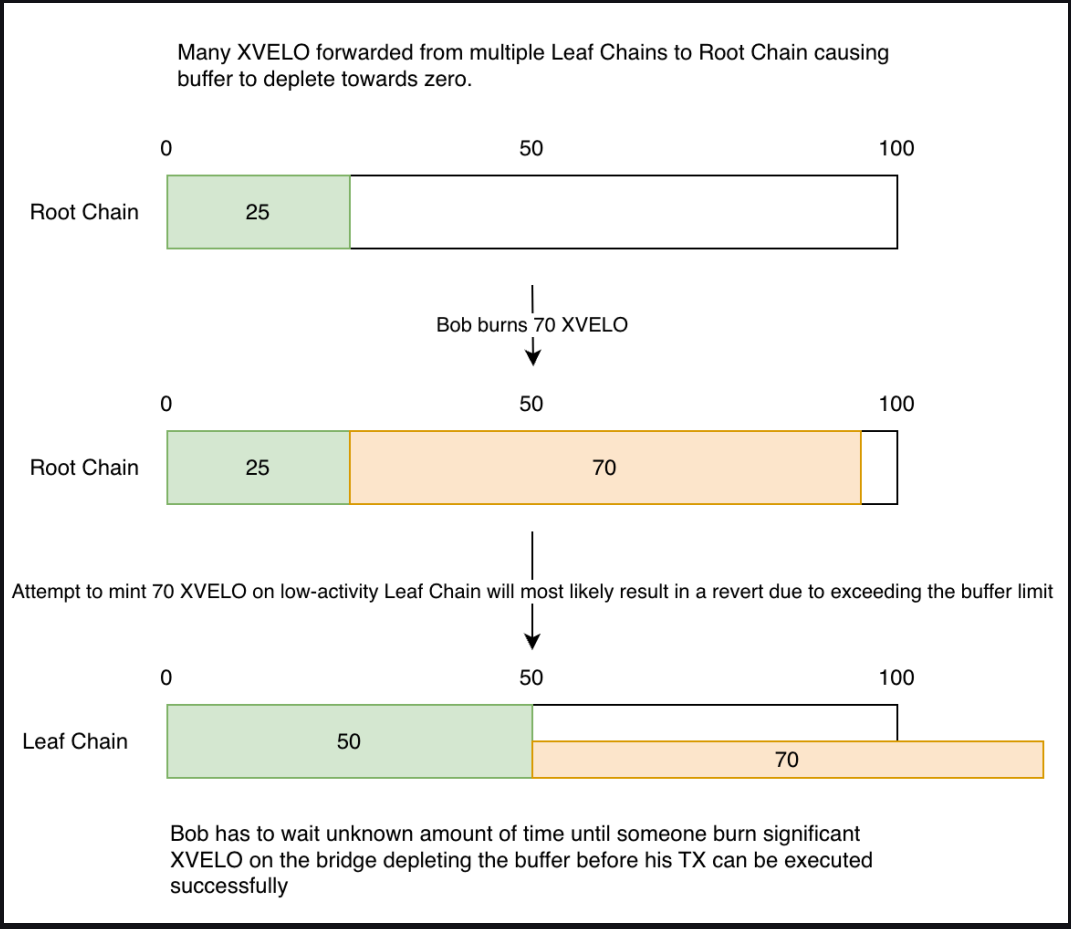
Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/8

Summary

Users who mint more than 50% of the buffer limit might have their XVELO delivery delayed on the destination chain.

Vulnerability Detail

The buffer limit will be at the midpoint most of the time (especially in chains with little activity). Thus, if we attempt to mint more than 50% of the buffer limit, it will revert, as shown in the following scenario:



Note
Hyperlane uses exponential backoff to re-deliver reverted transactions.

In addition, it is possible that Bob might not have a chance for his transaction to succeed even if someone burns significant XVELO on the leaf chain because while

waiting for exponential backoff delay, the buffer will ”recover” back to the midpoint.

Nevertheless, this is not a permanent DOS, as the protocol team can readjust the buffer/limit on the bridge to get the transaction unstuck.

Impact

Users who mint more than 50% of the buffer limit might have their XVELO delivery delayed on the destination chain.

Recommendation

Understood from the protocol team that they will monitor the bridging traffic and adjust the limits accordingly. Thus, it is recommended to take into consideration the above observation when configuring the limits to minimize the possibility of such an event from happening.

Discussion

simplyoptimistic

Status: Won't fix.

Comments: When new chains are added to the deployment, bridging traffic will be monitored to ensure that bridging limits are sufficient for the bridging activity. We can also add ui elements to inform users the size of the impact of their mints / burns on the buffer.

Issue L-6: Compatibility between existing and new `VotingRewards`

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/7

Summary

The new `VotingReward` or cross-chain pools require the veNFT's owner to explicitly execute the `RootVotingRewardsFactory.setRecipient` function if the recipient is a smart contract. This might lead to compatibility issues if the veNFT's owner is a smart contract or external protocol without the ability to execute this function.

Vulnerability Detail

Assuming an external protocol or smart contract owns a number of veNFTs. Currently, the protocol can call `Voter.claimFees` and `Voter.claimBribes` functions without issue because it does not check if the veNFT's owner (`IVotingEscrow(ve).ownerOf(_tokenId)`) is a smart contract or not. Thus, the recipient can be a smart contract.

However, in the new `RootFeesVotingReward.getReward` and `RootIncentiveVotingReward.getReward` functions, it will revert if the recipient is a smart contract (`_recipient.code.length > 0`). This leads to some compatibility issues with the new and existing `VotingReward` because one of them allows transfer to the smart contract, but the other does not.

The smart contract/protocol is supposed to execute the `RootVotingRewardsFactory.setRecipient` function to whitelist its address. In this case, the code will not check whether the recipient is a smart contract or not.

Impact

However, some existing smart contracts/protocols might not have the ability to upgrade their codebase to incorporate the new `RootVotingRewardsFactory.setRecipient` function. As a result, they might encounter the following issues:

- If they hardcoded their logic to vote for both existing pools and new cross-chain pools at the same time, they might encounter a revert, which would prevent reward tokens from being claimed.
- They might be forced to work only with older/existing `VotingRewards` or pools on Optimism Mainnet

Recommendation

After discussing it with the protocol team, it was understood that the new controls intend to prevent reward tokens from being forwarded to an address not controlled by the veNFT owner on the Leaf Chain, which could lead to a loss of funds.

Thus, it is recommended that additional documentation be included so that the integrators are aware of the potential impact of the new controls.

Discussion

simplyoptimistic

Status: Won't fix.

Comments: We will add additional documentation around the requirements for contracts to be able to support the voting of rewards x-chain. In most cases, both voting and reward fetching is likely done by passing in arguments, which means this issue can be mitigated.

Issue L-7: No support for smart contract wallets

Source: https://github.com/sherlock-audit/2024-10-velodrome/issues/3

Summary

There is no support for smart contract wallets while bridging tokens and paying for cross-chain fees. This leads to bad ux for smart contract wallets and stuck funds on destination chains if the token bridging is invoked by a smart contract wallet.

Vulnerability Detail

While bridging tokens through `TokenBridge` contract
the receiver on the destination chain is always the msg.sender. This works fine for EOAs, but won't work for smart contract wallets where it is not guaranteed that the wallet controls the same address on the destination chain.

Similarly, while paying for cross-chain fees WETH is being transferred from the tx.origin so other signers can't pay for the tx fee and only owner of the smart wallet must execute their transaction.

Impact

No support for smart contract wallets can lead to stuck funds on destination chains.

Recommendation

Add an option for the sender to specify the recipient of bridged funds on the destination chain.

Discussion

simplyoptimistic

Status: Will fix

Commit: https://github.com/velodrome-finance/superchain-contracts/commit/0a8f523aa92c3310e3046d6dcd8d46d5c3d9752e

Comment: Added support for the sender to specify the recipient on the token bridge. Smart contract wallets are not supported by the message bridge in the conventional way and documentation outlining additional steps required for integration will be added.

windhustler

Fix looks good.

Disclaimers

Sherlock does not provide guarantees nor warranties relating to the security of the project.

Usage of all smart contract software is at the respective users' sole risk and is the users' responsibility.