shieldify

Dinero

SECURITY REVIEW

Date: 2 December 2024

CONTENTS

1. About Shieldify

Positioned as the first hybrid Web3 Security company, Shieldify shakes things up with a unique subscription-based auditing model that entitles the customer to unlimited audits within its duration, as well as top-notch service quality thanks to a disruptive 6-layered security approach. The company works with very well-established researchers in the space and has secured multiple millions in TVL across protocols, also can audit codebases written in Solidity, Rust, Go, Vyper, Move and Cairo.

Learn more about us at shieldify.org.

2. Disclaimer

This security review does not guarantee bulletproof protection against a hack or exploit. Smart contracts are a novel technological feat with many known and unknown risks. The protocol, which this report is intended for, indemnifies Shieldify Security against any responsibility for any misbehavior, bugs, or exploits affecting the audited code during any part of the project's life cycle. It is also pivotal to acknowledge that modifications made to the audited code, including fixes for the issues described in this report, may introduce new problems and necessitate additional auditing.

3. About Dinero

Dinero is a suite of products that scale yield for protocols and users. It includes: (i) an ETH liquid staking token (pxETH) which benefits from ETH staking yield from Dinero's validators; (ii) a decentralized, collateral-backed stablecoin (pxUSD) as a medium of exchange on Ethereum; and (iii) a public and permissionless RPC for users.

pxETH is an Ethereum liquid staking solution that forms the foundation of the Dinero protocol. It uses a two-token system: pxETH and apxETH. This design provides users with a choice between liquidity and DeFi yields or boosted ETH staking yield. When depositing ETH, users can choose to hold pxETH or deposit it into an auto-compounding rewards vault for apxETH (i.e. staking their ETH):

- `pxETH` is for those prioritizing liquidity over staking yield. Holding pxETH means holding an ETH-pegged asset that can be used across DeFi for providing liquidity, participating in lending protocols, and more. Dinero's treasury and DINERO incentives will support and expand these opportunities for pxETH holders.
- `apxETH` is for users focused on maximizing staking yields. After minting pxETH, users can deposit it into Dinero's auto-compounding rewards vault to enjoy boosted staking yields without managing their own validators. Since some users will hold pxETH, each apxETH benefits from the staking rewards of more than one staked ETH, amplifying yields for apxETH holders.

A holistic overview of the protocol's core workflows can be explored here:

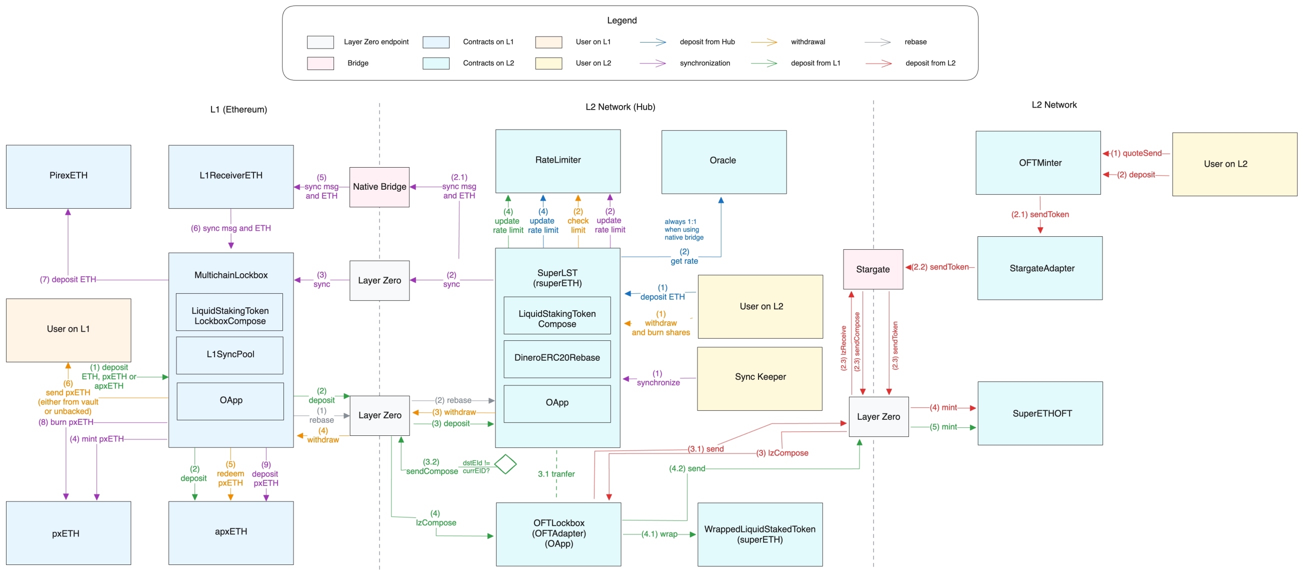

Figure 1: Dinero's Architecture

To learn more about Dinero, read the litepaper here.

4. Risk Classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

4.1 Impact

- **High** – results in a significant risk for the protocol's overall well-being. Affects all or most users
- **Medium** – results in a non-critical risk for the protocol affects all or only a subset of users, but is still unacceptable
- **Low** – losses will be limited but bearable – and covers vectors similar to griefing attacks that can be easily repaired

4.2 Likelihood

- **High** – almost certain to happen and highly lucrative for execution by malicious actors
- **Medium** – still relatively likely, although only conditionally possible
- **Low** – requires a unique set of circumstances and poses non-lucrative cost-of-execution to rewards ratio for the actor

5. Security Review Summary

The security review lasted 9 days with a total of 288 hours dedicated by 4 researchers from the Shieldify team.

Overall, the code is well-written. The audit report identified two Medium-severity issues and two Low-severity vulnerabilities, primarily related to the liquid staking and bridging functionality.

The Dinero team has been very responsive to the Shieldify research team's inquiries, demonstrating a strong commitment to security by extending the audit timeframe by an additional two days. Their proactive security approach is evident, as they have conducted multiple security reviews with various companies, showcasing their thoroughness and dedication to securing their platform.

5.1 Protocol Summary

Project Name	**Dinero**
Repository	dinero-pirex-eth
Type of Project	DeFi, Liquidity Staking
Audit Timeline	9 days
Review Commit Hash	6479303e66250eea77559021abfd61664a1dfd57
Fixes Review Commit Hash	4ef9af2c2a78519ad49dfd58e49b00f95b6d6437

5.2 Scope

The following smart contracts were in the scope of the security review:

File	**nSLOC**
src/layer2/L1ZKReceiverETH.sol	34
src/layer2/LiquidStakingToken.sol	466
src/layer2/LiquidStakingTokenCompose.sol	46
src/layer2/LiquidStakingTokenLockbox.sol	409
src/layer2/LiquidStakingTokenLockboxCompose.sol	474
src/layer2/MultichainLockbox.sol	134
src/layer2/interfaces/ICrossDomainMessengerMorph.sol	3
src/layer2/interfaces/IL2BaseToken.sol	3
src/layer2/libraries/Errors.sol	21
src/layer2/libraries/MsgCodec.sol	59
src/layer2/old/L2SyncPool.sol	201

src/layer2/old/LiquidStakingToken.sol	433
src/layer2/old/LiquidStakingTokenLockbox.sol	404
src/layer2/old/LiquidStakingTokenNonNative.sol	34
src/layer2/tokens/OrbitLST.sol	20
src/layer2/tokens/SuperLST.sol	21
src/layer2/tokens/ZKSyncLST.sol	20
src/layer2/utils/Oracle.sol	33
Total	**2815**

6. Findings Summary

The following number of issues have been identified, sorted by their severity:

- **Medium** issues: `2`
- **Low** issues: `2`
- **Info** issues: `1`

ID	Title	Severity	Status
[M-01]	L2 Oracle Update Can Be Sandwiched to Gain Extra Tokens	Medium	Acknowledged
[M-02]	`LiquidStakingToken._minGasLimit()` Is Set to Zero	Medium	Fixed
[L-01]	Slow Bridging from L2-L1 Using `Arbitrum/Optimism` Bridge May Fail the Dispute process in Favour of an Attacker	Low	Acknowledged
[L-02]	`LiquidStakingToken.withdraw()` Should Not Have a `whenNotPaused` Modifier	Low	Acknowledged
[I-01]	Increase Sanity Checks for `deposit()` Functions in `MultichainLockbox.sol`	Informational	Fixed

7. Findings

[M-01] L2 Oracle Update Can Be Sandwiched to Gain Extra Tokens

Severity

Medium Risk

Description

When a user calls `deposit()` in `L2SyncPool`, he deposits in the asset (tokenIn) and get some shares. This asset will be in the contract until `sync()` is called to bridge the assets to L1 and be deposited into the `apxETH` vault.

The shares the user gets is calculate by calling `convertToShares()`, and the ratio is dependent on the oracle rate which is updated manually by the keeper.

```
    uint256 _totalShares = getTotalShares();
>   uint256 shares = _totalShares == 0 ? _amount : convertToShares(_amount
    );
    uint256 depositFee = $.syncDepositFee;

     if (depositFee > 0) {
         uint256 feeShares = shares.mulDivDown(depositFee, Constants.
            FEE_DENOMINATOR);

        _mintShares($.treasury, feeShares);
        $.unsyncedShares += feeShares;

        shares -= feeShares;
    }

>   _mintShares(_to, shares);
```

There is also a `withdraw()` function in `LiquidStakingToken.sol`. This function burns the shares that the user has, call `previewWithdraw()` to get the amount of assets that the shares is worth which will be redeemable for the user on L1. The share:asset ratio is also dependent on the same oracle pricing.

```
function withdraw(
    address _receiver,
    address _refundAddress,
    uint256 _amount,
    bytes calldata _options
) external payable virtual nonReentrant whenNotPaused {
    if (_receiver == address(0)) revert Errors.ZeroAddress();
    if (_amount == 0) revert Errors.ZeroAmount();

    L2TokenStorage storage $ = _getLiquidStakingTokenStorage();
```

```
>   uint256 shares = previewWithdraw(_amount);

    IRateLimiter(getRateLimiter()).updateRateLimit(
        address(this),
        address(this),
        0,
        shares
    );

>   _burnShares(msg.sender, shares);

    bytes memory payload = abi.encode(
        Constants.MESSAGE_TYPE_WITHDRAW,
>       _amount,
        _receiver
    );
    // code
```

A malicious user can sandwich the oracle update by frontrunning the oracle and calling `deposit()` to get shares, wait for the oracle to update to a better conversion rate, and immediately backrunning the oracle by calling `withdraw()` to convert the shares back to assets.

For example (ignoring fees):

- Let's say asset:share is 1:1, Alice deposits 100 pxETH assets in L2 and gets 100 shares
- The oracle updates such that asset:share is now 1.1:1 (share is worth more now)
- Alice immediately calls withdraw and now her 100 shares is worth 110 pxETH, which is redeemable in L1
- Every time the oracle updates the prices (shares prices increases relative to asset), Alice can sandwich the oracle to gain a profit

Impact

Every time there is a favorable change in conversion rate, anyone can exploit the rate by sandwiching the oracle.

Recommendation

One potential mitigation is for the oracle to be constantly updated so the updated rate is not too exploitable for gains.

Team Response

Acknowledged. We are currently updating the Oracle as close to each harvest event on the mainnet as possible. However, the potential gains from these transactions are unlikely to justify the associated fees, especially given the L2 deposit limits and the possibility of an L2 deposit fee.

[M-02] `LiquidStakingToken._minGasLimit()` Is Set to Zero

Severity

Medium Risk

Description

When sending slow messages using `_sendSlowSyncMessage()`, `_minGasLimit()` is used:

```
function _sendSlowSyncMessage(
    address,
    uint256 _value,
    uint256,
    bytes memory _data
) internal override {
    bytes memory message = abi.encodeCall(
        IL1Receiver.onMessageReceived,
        _data
    );

    ICrossDomainMessenger(getMessenger()).sendMessage{value: _value}(
        getReceiver(),
        message,
>       _minGasLimit()
    );
}
```

The return value of `_minGasLimit()` is hardcoded to zero in all the contracts, like `LiquidStakingToken.sol` , and not overridden.

```
    /**
     * @dev Internal function to get the minimum gas limit
>    * This function should be overridden to set a minimum gas limit to
   forward during the execution of the message
     * by the L1 receiver contract. This is mostly needed if the
        underlying contract have some try/catch mechanism
     * as this could be abused by gas-griefing attacks.
     * @return minGasLimit Minimum gas limit
     */
    function _minGasLimit() internal view virtual returns (uint32) {
>       return 0;
    }
```

From the L2StandardBridge OP contract, a common number for `minGasLimit` is 200,000 - 250,000 when transferring ETH tokens or ERC20 tokens (bridgeETHTo() is called).

The `minGasLimit()` is set to zero only when `withdraw()` is called.

Impact

The bridged transaction may revert.

Recommendation

For best practice to prevent any bridge reverts from L2 to L1, set the `_minGasLimit()` to at least `200,000`.

Team Response

Fixed, set the `minGasLimit` to `200,000`.

[L-01] Slow Bridging from L2-L1 Using `Arbitrum/Optimism` Bridge May Fail the Dispute process in Favour of an Attacker

Severity

Low Risk

Description

The protocol uses a fast/slow sync method. This means that on L2, when the user calls `deposit()` and sends some asset tokens, they immediately get shares. A fast message is sent to the L1 side, informing the L1 side that assets are coming, which will increase the `pendingDeposit` variable. `_lzReceive()` also calls `_anticipatedDeposit()` which mints `pxETH` for the contract.

```
function _anticipatedDeposit(
    bytes32 guid,
    address,
    uint256,
    uint256 amountOut
) internal virtual returns (uint256) {
// if the message was already processed, return 0
// this should only happen if the slow sync msg arrives first than the
   fast sync msg
    if (_getL1SyncPoolStorage().processedMessages[guid]) return 0;

    L1SyncPoolStorage storage $ = _getL1SyncPoolStorage();

    _getL1SyncPoolStorage().processedMessages[guid] = true;

    $.totalUnbackedTokens += amountOut;

    IDineroERC20 pxETH = IDineroERC20(address(getTokenOut()));

>   pxETH.mint(address(this), amountOut);

    return amountOut;
}
```

At the same time, a slow message is sent to the L1 side. This message will carry the asset with it and bridge the asset to the L1 side.

The slow message uses Arbitrum `IArbitrumMessenger(getMessenger()).sendTxToL1` or OP `ICrossDomainMessenger(getMessenger()).sendMessage{value: _value}`, which takes ~7 days.

In the OP docs, it mentions:

The point here is that you don't want to be making decisions about Layer 2 transaction results from inside a smart contract on Layer 1 until this challenge period has elapsed.

If the bridge gets hacked or the fault dispute resolves in favour of an attacker, ETH will not be deposited in the L1 contract, which will affect the accounting in `L1SyncPool._finalizeDeposit()`. Specifically, `totalUnbackedTokens()` will not be correct since `onMessageReceived()` may not be called.

```
$.totalUnbackedTokens > amountOut
    ? $.totalUnbackedTokens -= amountOut
    : $.totalUnbackedTokens = 0;
```

Impact

Variables will not be updated accordingly.

Recommendation

7 days is a long time for confirmation. It is recommend to use the LZ bridge instead to bridge tokens. If the protocol intends on using the ARB/OP bridge, ensure that there is a rollback mechanism on L1 that the owner can do in case of a bridge attack.

Team Response

Acknowledged.

[L-02] `LiquidStakingToken.withdraw()` Should Not Have a `whenNotPaused` Modifier

Severity

Low Risk

Description

A user calls `LiquidStakingToken.withdraw()` to burn their shares for assets and redeem their assets in L1. Usually in staking protocols, the `whenNotPaused` modifier is not used for withdrawing tokens due to centralization risk. In the event that the user wants to offload their tokens and withdraw from the protocol, they should be allowed to do so.

```
function withdraw(
    address _receiver,
    address _refundAddress,
    uint256 _amount,
    bytes calldata _options
>   ) external payable virtual nonReentrant whenNotPaused {
    if (_receiver == address(0)) revert Errors.ZeroAddress();

    // code
}
```

Impact

Centralization risk.

Recommendation

It is recommend to remove the `whenNotPaused` modifier from the `withdraw()` function.

Team Response

Acknowledged.

[I-01] Increase Sanity Checks for `deposit()` Functions in `MultichainLockbox.sol`

Severity

Informational

Description

In `MultichainLockbox.sol`, the functions `depositEth()`, `depositPxEth()` and `depositApxEth()` checks whether the `_receiver()` parameter is address zero but not the `_refundAddress()`.

```
function depositEth(
    uint32 _dstEid,
>   address _receiver,
>   address _refundAddress,
    uint256 _amount,
    bool _shouldWrap,
    bytes calldata _options
) external payable override nonReentrant whenNotPaused {
    if (_shouldWrap) revert Errors.MultichainDepositsCannotBeWrapped();
>   if (_receiver == address(0)) revert Errors.ZeroAddress();
```

If the refund address is not set, any excess `msg.value` will not be refunded when `_lzSend()` is called:

```
function _lzSend(
    uint32 _dstEid,
    bytes memory _message,
    bytes memory _options,
    MessagingFee memory _fee,
    address _refundAddress
) internal virtual returns (MessagingReceipt memory receipt) {
// @dev Push corresponding fees to the endpoint, any excess is sent back
   to the _refundAddress from the endpoint.
    uint256 messageValue = _payNative(_fee.nativeFee);
    if (_fee.lzTokenFee > 0) _payLzToken(_fee.lzTokenFee);
```

Recommendation

It is recommend to check for zero address in the `deposit()` function:

```
function depositEth(
    uint32 _dstEid,
    address _receiver,
    address _refundAddress,
    uint256 _amount,
    bool _shouldWrap,
    bytes calldata _options
) external payable override nonReentrant whenNotPaused {
    if (_shouldWrap) revert Errors.MultichainDepositsCannotBeWrapped();
    if (_receiver == address(0)) revert Errors.ZeroAddress();
+   if (_refundAddress == address(0)) revert Errors.ZeroAddress();
```

Team Response

Fixed.

our shielding · Your smart contracts, our shielding · Your smart c

shieldify

Thank you!