External SHIELD

DefiLens

Smart Contract Audit Report

Mar 15th, 2024

externalshield.com

Table of Content

Executive Summary

Overview

PROJECT DETAIL	
Project Name	DefiLens
Language	Solidity
Repository	https://github.com/DefiLens/defilens-contracts
nSLOC	140

Vulnerability Summary

Vulnerability Level	Total	Acknowledged	Fixed	Declined
High	**1**	**0**	**0**	**1**
Medium	**4**	**1**	**3**	**0**
Low	**6**	**1**	**5**	**0**
Informational	**8**	**0**	**8**	**0**

Audit Scope

- **Smart Contracts**
  - ChainPing.sol

Vulnerability Severity Classification

- **High** - vulnerabilities where assets face direct threats of theft, loss, or compromise. Exploitation of these vulnerabilities can result in significant harm to the system or assets.

- **Medium** - vulnerabilities imply that assets are not at direct risk, but the function of the protocol or its availability could be impacted with a hypothetical attack path with stated assumptions, but external requirements.

- **Low** - It involve non-critical issues that do not directly impact the security or functionality of the system.

- **Informational** - Informational vulnerabilities focus on aspects such as code style, syntax, providing recommendations for improvement in non-critical areas to enhance code readability, maintainability, or compliance with best practices. These vulnerabilities do not pose any direct risk to the system or assets.

Methodology

Our audit methodology combines industry best practices, security standards, and specialized tools to ensure a comprehensive evaluation of smart contracts. Key aspects of our methodology include:

- **Code Review:** Analyzing the smart contract code line by line to identify vulnerabilities and potential risks.

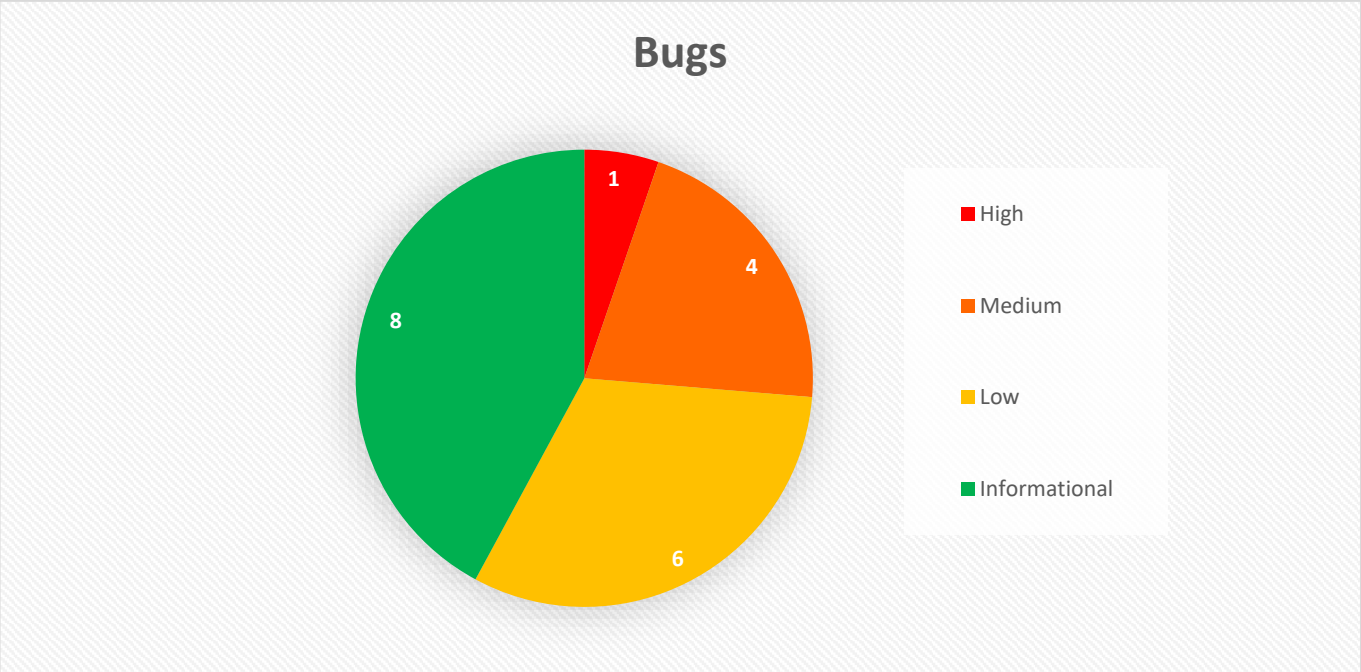
- **Security Testing:** Conducting various tests, including static analysis, dynamic analysis, and manual review, to uncover security weaknesses

- **Functional Testing:** Verifying the smart contract's functionality against intended specifications and business logic.

- **Risk Assessment:** Evaluating risks associated with the smart contract's design, implementation, and deployment.

By following this structured approach and methodology, ExternalShield delivers comprehensive smart contract audits that provide valuable insights and recommendations to enhance security and functionality.

Findings



ID	Name	Severity	Status
H-1	Lack of Slippage Control in Token Swapping	High	Decline
M-1	call() should be used instead of transfer() on an address payable	Medium	Fixed
M-2	Unsafe use of transfer()/transferFrom() with IERC20	Medium	Fixed
M-3	Centralization Risk for trusted owners	Medium	Acknowledged
M-4	approve()/safeApprove() may revert if the current approval is not zero	Medium	Fixed

ID	Name	Severity	Status
L-1	Return values of transfer()/transferFrom() not checked	Low	Fixed
L-2	Some tokens may revert when zero value transfers are made	Low	Fixed
L-3	setOwner() function missing checks for address(0) when assigning values to address state variables	Low	Acknowledged
L-4	Deprecated approve() function	Low	Fixed
L-5	External call recipient may consume all transaction gas	Low	Fixed
L-6	Lack of Two-Step Ownership Transfer in setOwner Function	Low	Fixed
NC-1	Event missing indexed field	Informational	Fixed
NC-2	Change uint to uint256	Informational	Fixed
NC-3	Missing Event for critical parameters change	Informational	Fixed
NC-4	override function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings	Informational	Fixed

ID	Name	Severity	Status
NC-5	public functions not called by the contract should be declared external instead	Informational	Fixed
NC-6	Gas Optimization - State variables should be cached in stack variables rather than re-reading them from storage	Gas	Fixed
NC-7	Gas Optimization - Use calldata instead of memory for function arguments that do not get mutated	Gas	Fixed
NC-8	Gas Optimization - Use Custom Errors instead of Revert Strings to save Gas	Gas	Fixed

High Severity Issues

H-1 : Lack of Slippage Control in Token Swapping

Description:

The ChainPing contract does not provide a way for users to control slippage during token swaps. Without the ability to control slippage, users may experience higher than expected losses during token swaps, especially in volatile market conditions. This could discourage users from using the protocol and negatively impact the protocol's liquidity and user base.

Code Location:

```
200:  function swap(
        address _tokenIn,
        address _tokenOut,
        address _user,
        // uint24 _poolFee,
        uint256 _amount,
        bytes memory txData
      ) internal returns(uint256 amountOut) {
          IERC20(_tokenIn).approve(address(swapRouter), _amount);
          (bool success, ) = address(swapRouter).call(txData);
          if (!success) IERC20(_tokenIn).transfer(_user, _amount);
          emit Swap(_tokenIn, _tokenOut, _amount, amountOut);
      }
```

Recommendation:

Modify the swap function and related functions to include an amountOutMinimum parameter. Pass this parameter to the swap router when making a swap. This will allow users to set a limit on the slippage for their swaps.

Status:

Decline: Because txData consists of amountOutMinimum But more possibility if not success then no event emit.

Medium Severity Issues

M-1 : call() should be used instead of transfer() on an address payable

Description:

In the rescueFunds() function of the ChainPing contract, transfer() is used to send Ether from the contract to the owner. This is a potential security risk as transfer() only forwards 2300 gas, which might not be enough if the receiving contract is complex and requires more gas to execute.

Code Location:

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

Recommendation:

Use call() instead of transfer(). call() forwards all remaining gas and does not limit the gas stipend.

```
(bool success, ) = payable(msg.sender).call{value: address(this).balance}("");
require(success, "Transfer failed");
```

Status:

Fixed

M-2 : Unsafe use of transfer()/transferFrom() with IERC20

Description:

Some tokens do not implement the ERC20 standard properly but are still accepted by most code that accepts ERC20 tokens. For example Tether (USDT)'s transfer() and transferFrom() functions on L1 do not return booleans as the specification requires, and instead have no return value. When these sorts of tokens are cast to IERC20, their function signatures do not match and therefore the calls made, revert (see link for a test case)

Code Location:

```
94:     IERC20(token).transfer(msg.sender, IERC20(token).balanceOf(address(this)));
```

```
144:    IERC20(_token).transfer(_user, amountLD);
```

```
167:    if (anyDust > 0) IERC20(_token).transfer(_user, anyDust);
```

```
171:     IERC20(_extraOrShareToken).transfer(_user, afterTokens-beforeTokens);
```

```
210:    if (!success) IERC20(_tokenIn).transfer(_user, _amount);
```

Recommendation:

Use OpenZeppelin's SafeERC20's safeTransfer() / safeTransferFrom() instead

Status:

Fixed

M-3 : Centralization Risk for trusted owners

Description:

The smart contract ChainPing in the file contracts/ChainPing.sol has a potential centralization risk due to the over-reliance on trusted owners. Several functions in the contract are only executable by the owner of the contract, which could lead to a single point of failure if the owner's private key is compromised or lost.

These functions control critical aspects of the contract, such as setting the owner, setting the router, rescuing funds, and approving tokens. If the owner's private key is compromised, an attacker could potentially take over the contract, change the router to a malicious one, or drain the contract's funds.

Code Location:

```
74:     function setBlockTimeStamp(uint256 _blockTimeStamp) external onlyOwner {
```

```
78:     function setOwner(address _owner) external onlyOwner {
```

```
82:     function setRouter(address _stargateRouter) external onlyOwner {
```

```
90:     function rescueFunds(address token) public onlyOwner {
```

```
98:     function approve(address token, uint256 amount) public onlyOwner {
```

Recommendation:

Implement a decentralized governance mechanism, such as a multisig wallet or a DAO, to replace the single owner. This would distribute the power to make critical

changes to the contract among multiple trusted parties, reducing the risk of a single point of failure.

Status:

Acknowledged

M-4 : setOwner() function missing checks for address(0) when assigning values to address state variables

Description:

The setOwner() function allows the contract owner to change the owner of the contract. However, there is no check to prevent the owner from being set to the zero address (address(0)). This could lead to a loss of control over the contract if the owner is accidentally or intentionally set to address(0).

Code Location:

```
79:     owner = _owner;
```

Recommendation:

Add a check in the setOwner function to reject address(0) as a valid owner address.

```
function setOwner(address _owner) external onlyOwner {
     require(_owner != address(0), "Owner address cannot be 0");
     owner = _owner;
 }
```

Status:

Fixed

Low Severity Issues

L-1 : Return values of transfer()/transferFrom() not checked

Description:

The contract does not check the return values of transfer()/transferFrom() calls. According to the ERC20 standard, these functions return a boolean value indicating whether the operation was successful. However, not all ERC20 token implementations revert on failure, some return false instead. By not checking the return value, operations that should have been marked as failed may potentially go through without actually making a payment.

Code Location:

```
94:     IERC20(token).transfer(msg.sender, IERC20(token).balanceOf(address(this)));
```

```
144:    IERC20(_token).transfer(_user, amountLD);
```

```
167:    if (anyDust > 0) IERC20(_token).transfer(_user, anyDust);
```

```
171:    IERC20(_extraOrShareToken).transfer(_user, afterTokens-beforeTokens);
```

```
210:    if (!success) IERC20(_tokenIn).transfer(_user, _amount);
```

Recommendation:

Consider checking the return value of transfer()/transferFrom() calls.

```
  bool success =
IERC20(token).transfer(msg.sender,IERC20(token).balanceOf(address(this)));
  require(success, "Transfer failed");
```

Status:

Fixed

L-2 : approve()/safeApprove() may revert if the current approval is not zero

Description:

- Some tokens (like the USDT) do not work when changing the allowance from an existing non-zero allowance value (it will revert if the current approval is not zero to protect against front-running changes of approvals). These tokens must first be approved for zero and then the actual allowance can be approved.

- Furthermore, OZ's implementation of safeApprove would throw an error if an approve is attempted from a non-zero value (`"SafeERC20: approve from non-zero to non-zero allowance"`)

Code Location:

```
147:    IERC20(_token).approve(_to, amountLD);
```

```
208:    IERC20(_tokenIn).approve(address(swapRouter), _amount);
```

Recommendation:

Set the allowance to zero immediately before each of the existing allowance calls

Status:

Fixed

L-3 : Some tokens may revert when zero value transfers are made

Description:

checkout: https://github.com/d-xo/weird-erc20#revert-on-zero-value-transfers.
In spite of the fact that EIP-20 states that zero-valued transfers must be accepted, some tokens, such as LEND will revert if this is attempted, which may cause transactions that involve other tokens (such as batch operations) to fully revert.

Code Location:

```
94:     IERC20(token).transfer(msg.sender, IERC20(token).balanceOf(address(this)));
```

```
144:    IERC20(_token).transfer(_user, amountLD);
```

```
167:    if (anyDust > 0) IERC20(_token).transfer(_user, anyDust);
```

```
171:    IERC20(_extraOrShareToken).transfer(_user, afterTokens-beforeTokens);
```

```
210:    if (!success) IERC20(_tokenIn).transfer(_user, _amount);
```

Recommendation:

Consider skipping the transfer if the amount is zero, which will also save gas.

Status:

Acknowledged

L-4 : Deprecated approve() function

Description:

Due to the inheritance of ERC20's approve function, there's a vulnerability to the ERC20 approve and double spend front running attack. Briefly, an authorized spender could spend both allowances by front running an allowance-changing transaction.

Code Location:

```
99:     IERC20(token).approve(msg.sender, 0);
100:    IERC20(token).approve(msg.sender, amount);
```

```
147:    IERC20(_token).approve(_to, amountLD);
```

```
161:    IERC20(_token).approve(_to, 0);
```

```
208:    IERC20(_tokenIn).approve(address(swapRouter), _amount);
```

Recommendation:

Consider implementing OpenZeppelin's .safeApprove() function to help mitigate this.

Status:

Fixed

L-5 : External call recipient may consume all transaction gas

Description:

There is no limit specified on the amount of gas used, so the recipient can use up all of the transaction's gas, causing it to revert.

Code Location:

```
153:    (success, ) = _to.call{value: nativeFee}(txData);
```

```
155:    (success, ) = _to.call(txData);
```

```
209:    (bool success, ) = address(swapRouter).call(txData);
```

Recommendation:

Use addr.call{gas: <amount>}("") or this library instead.

Status:

Fixed

L-6 : Lack of Two-Step Ownership Transfer in setOwner Function

Description:

The setOwner function allows for direct transfer of contract ownership. This can potentially lead to a loss of contract control if the new owner's address is inputted incorrectly.

A two-step ownership transfer process is expected to mitigate the risk of accidental loss of contract control. The process should be as follows:

1. The current owner calls a function to propose a new owner.
2. The proposed new owner has to call a function to accept the ownership.

Code Location:

```
78:   function setOwner(address _owner) external onlyOwner {
        owner = _owner;
      }
```

Recommendation:

Implement two step ownership,

```
address public proposedOwner;

function proposeNewOwner(address _proposedOwner) external onlyOwner {
    proposedOwner = _proposedOwner;
}

function acceptOwnership() external {
    require(msg.sender == proposedOwner, "Not the proposed owner");
    owner = proposedOwner;
    proposedOwner = address(0);
}
```

Status:

Fixed

Informational Issues

NC-1 : Event missing indexed field

Description:

Index event fields make the field more quickly accessible to off-chain tools that parse events. This is especially useful when it comes to filtering based on an address. However, note that each index field costs extra gas during emission, so it's not necessarily best to index the maximum allowed per event (three fields). Where applicable, each `event` should use three `indexed` fields if there are three or more fields, and gas usage is not particularly of concern for the events in question.

Code Location:

```
54:     event ReceivedOnDestination(address token, uint256 amountLD, bool success);
```

Recommendation:

mark at least one field as indexed.

Status:

Fixed

NC-2 : Change uint to uint256

Description:

Throughout the code base, some variables are declared as uint. To favor explicitness, consider changing all instances of `uint` to uint256

Code Location:

```
24:     uint _nonce,
```

```
26:     uint amountLD,
```

```
112:     uint _nonce,
```

```
114:     uint amountLD,
```

```
129: function withoutSwap(address _token, uint amountLD, bytes memory _payload)
internal {
```

```
180:     uint amountLD,
```

Recommendation:

change all instances of uint to uint256

Status:

Fixed

NC-3 : Missing Event for critical parameters change

Description:

Events help non-contract tools to track changes, and events prevent users from being surprised by changes.

Code Location:

```
74:   function setBlockTimeStamp(uint256 _blockTimeStamp) external onlyOwner {
        blockTimeStamp = _blockTimeStamp;
```

```
78:   function setOwner(address _owner) external onlyOwner {
        owner = _owner;
```

```
82:   function setRouter(address _stargateRouter) external onlyOwner {
        stargateRouter = _stargateRouter;
```

```
86:   function setSwapRouter(ISwapRouter _swapRouter) public onlyOwner{
        swapRouter = _swapRouter;
```

Recommendation:

Emit events for critical parameters change

Status:

Fixed

NC-4 : override function arguments that are unused should have the variable name removed or commented out to avoid compiler warnings

Description:

In the sgReceive() function, there are unused arguments that are causing compiler warnings. These arguments are _chainId, _srcAddress, and _nonce.

Code Location:

```
110:    uint16 _chainId,
111:    bytes memory _srcAddress,
112:    uint _nonce,
```

Recommendation:

remove the unused arguments.

Status:

Fixed

NC-5 : public functions not called by the contract should be declared external instead

Description:

Several functions in the contract are declared as public (e.g. balanceOf(), setSwapRouter(), rescueFunds() and approve()) when they should be declared as external. In Solidity, public functions can be called both internally and externally, while external functions can only be called from outside the contract. If a function is not called by the contract itself, it should be declared as external to save gas and improve contract efficiency.

Code Location:

```
70:     function balanceOf(address token) public view returns(uint256) {
```

```
86:     function setSwapRouter(ISwapRouter _swapRouter) public onlyOwner{
```

```
90:     function rescueFunds(address token) public onlyOwner {
```

```
98:     function approve(address token, uint256 amount) public onlyOwner {
```

Recommendation:

Change the visibility specifier of these functions from public to external.

Status:

Fixed

NC-6 : Gas Optimization - State variables should be cached in stack variables rather than re-reading them from storage

Description:

The instances below point to the second+ access of a state variable within a function. Caching of a state variable replaces each Gwarmaccess (100 gas) with a much cheaper stack read. Other less obvious fixes/optimizations include having local memory caches of state variable structs, or having local caches of state variable contracts/addresses. It saves 100 gas per instance.

Code Location:

```
209:    (bool success, ) = address(swapRouter).call(txData);
```

Recommendation:

Cache the swapRouter state variable in a stack variable at the beginning of the function. Here's a code snippet demonstrating this:

```
function swap(
    address _tokenIn,
    address _tokenOut,
    address _user,
    uint256 _amount,
    bytes memory txData
) internal returns(uint256 amountOut) {
    ISwapRouter _swapRouter = swapRouter;
    IERC20(_tokenIn).approve(address(_swapRouter), _amount);
    (bool success, ) = address(_swapRouter).call(txData);
    if (!success) IERC20(_tokenIn).transfer(_user, _amount);
    emit Swap(_tokenIn, _tokenOut, _amount, amountOut);
}
```

Status:

Fixed

NC-7 : Gas Optimization - Use calldata instead of memory for function arguments that do not get mutated

Description:

It saves 60 gas per instance. When a function with a memory array is called externally, the abi.decode() step has to use a for-loop to copy each index of the calldata to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. 60 * <mem_array>.length). Using calldata directly bypasses this loop.

If the array is passed to an internal function which passes the array to another internal function where the array is modified and therefore memory is used in the external call, it's still more gas-efficient to use calldata when the `external` function uses modifiers, since the modifiers may prevent the internal functions from being called. Structs have the same overhead as an array of length one.

Code Location:

```
23:     bytes memory _srcAddress,
```

```
27:     bytes memory _payload
```

```
43:     ISwapRouter.ExactInputSingleParams memory params
```

```
111:     bytes memory _srcAddress,
```

```
115:     bytes memory _payload
```

Recommendation:

Use calldata instead of memory.

Status:

Fixed

NC-8 : Gas Optimization - Use Custom Errors instead of Revert Strings to save Gas

Description:

Custom errors are available from solidity version 0.8.4. Custom errors save **~50 gas** each time they're hit by avoiding having to allocate and store the revert string. Not defining the strings also save deployment gas
Additionally, custom errors can be used inside and outside of contracts (including interfaces and libraries).

Source:
> *Starting from Solidity v0.8.4, there is a convenient and gas-efficient way to explain to users why an operation failed through the use of custom errors. Until now, you could already use strings to give more information about failures (e.g., revert("Insufficient funds.");), but they are rather expensive, especially when it comes to deploy cost, and it is difficult to use dynamic information in them.*

Consider replacing ****all revert strings**** with custom errors in the solution, and particularly those that have multiple occurrences:

Code Location:

```
57:     require(owner == msg.sender, "Invalid Owner");
```

Recommendation:

Use custom errors.

Status:

Fixed

Disclaimer

This disclaimer is intended to outline the limitations and responsibilities associated with the audit report provided by ExternalShield for the smart contract in question.

- The audit report is based on the information and documentation provided to ExternalShield at the time of the audit. Any changes or updates made to the smart contract after the audit may impact its security and functionality.

- While every effort has been made to conduct a thorough audit, it is important to note that no audit can provide absolute assurance of the smart contract's security or functionality.

- The audit report may not cover vulnerabilities or risks arising from third-party dependencies or interactions with external systems beyond the scope of the audit.

- ExternalShield shall not be held liable for any damages, losses, or liabilities arising from the use of the audit report or its recommendations.

Thank you for entrusting us with the audit of your smart contract.