# Competitive Security Assessment

zklink Nova

Mar 8th, 2024

Secure3

secure3.io

Summary

This report is prepared for the project to identify vulnerabilities and issues in the smart contract source code. A group of NDA covered experienced security experts have participated in the Secure3's Audit Contest to find vulnerabilities and optimizations. Secure3 team has participated in the contest process as well to provide extra auditing coverage and scrutiny of the finding submissions.

The comprehensive examination and auditing scope includes:

- Cross checking contract implementation against functionalities described in the documents and white paper disclosed by the project owner.

- Contract Privilege Role Review to provide more clarity on smart contract roles and privilege.

- Using static analysis tools to analyze smart contracts against common known vulnerabilities patterns.

- Verify the code base is compliant with the most up-to-date industry standards and security best practices.

- Comprehensive line-by-line manual code review of the entire codebase by industry experts.

The security assessment resulted in findings that are categorized in four severity levels: Critical, Medium, Low, Informational. For each of the findings, the report has included recommendations of fix or mitigation for security and best practices.

Overview

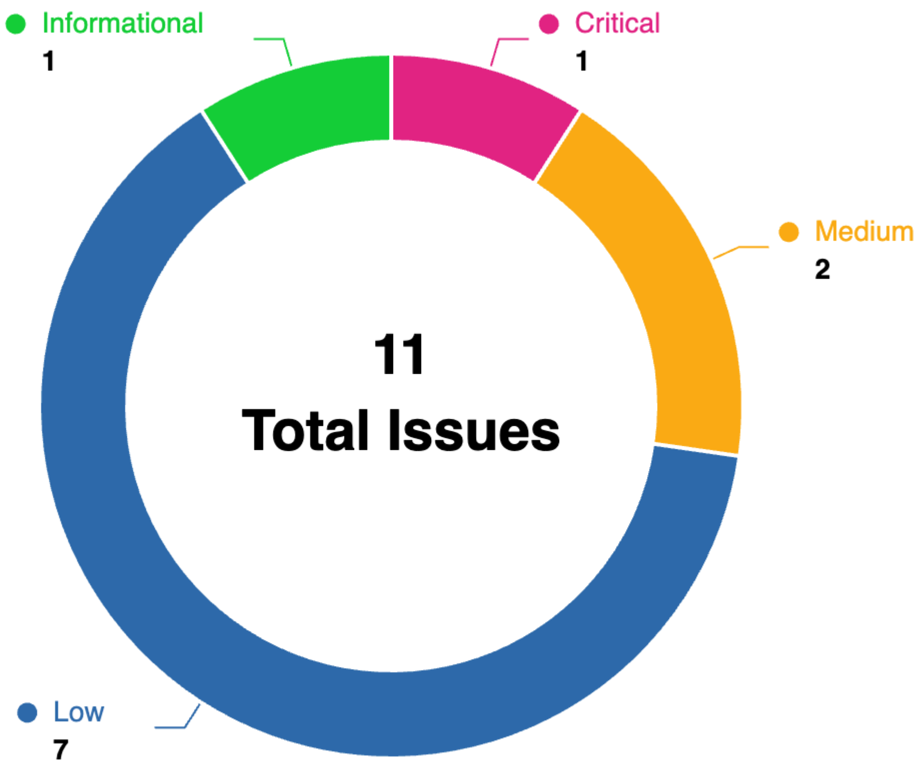
Project Name	zklink Nova
Language	Solidity
Codebase	• https://github.com/zkLinkProtocol/zklink-evm-contracts • audit version - 8a1385601543f44ede92782c14a94d817c767aaf • final version - 933e578a5df998af72109a57e8b9501f672cdf9f • https://github.com/zkLinkProtocol/era-contracts • audit version - 3f9875a1ca028b182152237a54017b98f67a1e6a • final version - e487cbb89dd62223d70ece7a2331f55860016a1c
Audit Methodology	• Audit Contest • Business Logic and Code Review • Privileged Roles Review • Static Analysis

Audit Scope

File	SHA256 Hash
./contracts/ZkLink.sol	5385099d50b926f9afe4c305daa9824be56c8c2dad56b2d68ed95fe4ab0f29f3
./contracts/Arbitrator.sol	9cd232873078d3a9a9ba21e541b249b6645d823eb2776d1849719f20c04b49f1
./contracts/gateway/zksync/ZkSyncL1Gateway.sol	c92bc97e74007db377a96255466b3ab42949f890aa051630aed4d58a582c355e
./contracts/gateway/arbitrum/ArbitrumL1Gateway.sol	ef2b3cfb688cc39cc2d66bec91b0f9b5422d60e63783c85e2fb06a94c2079efa
./contracts/gateway/ethereum/EthereumGateway.sol	b16534899b9e329f0fdfb95cfe91909eb92495d001d2233f67d3243a14ba4c60
./contracts/gateway/scroll/ScrollL1Gateway.sol	e6f90db8b15bd5232eecfaf4c6e8f8bce2e34f360f4c1b60757028ab5ce17229
./contracts/gateway/zkpolygon/ZkPolygonL1Gateway.sol	f581315fcbf24bf91551e58cdca31743971c13a264cb54681c6a9b3ed499e41b
./contracts/gateway/zkpolygon/ZkPolygonL2Gateway.sol	bca757528039d78b5bf4b8680a605467e1ce17b9386f859a114d506f671d113e
./contracts/gateway/zksync/ZkSyncL2Gateway.sol	555dba48e03fa0374ea76aac69f5742c0f6c27702e6328d8818be2eea3c8b05c
./contracts/gateway/optimism/OptimismL1Gateway.sol	40f7515b251616283b4bef8278cb88fe873f5e34f487a0a1b9d16d076a55c36b
./contracts/gateway/scroll/ScrollL2Gateway.sol	51eea3404b56208415a1b5da9e784b92a25f21d57011951743ddb7cf460af1f4
./contracts/gateway/linea/LineaL2Gateway.sol	5762bd0358150f04bbbeac7832f1823d6ce4e3d58b0dc71f3f7bb6349b099cb8
./contracts/gateway/linea/LineaL1Gateway.sol	849195a397fbe2ffbf221342aad265caf1a1946a958dcbf875fdecb095c0c896
./contracts/gateway/arbitrum/ArbitrumL2Gateway.sol	7cb6641022cdb69e3d5a9d4b85419e291f9680f944b68bf3ffc72fb71ed8259c
./contracts/gateway/optimism/OptimismL2Gateway.sol	5aa01e01877f21638e1aa6afd1a9ebe8945662ef8e77f280b92ca31bdd60e798
./contracts/gateway/BaseGateway.sol	b31ec4a7a8f439ec314e9e24acd1e6683fda8b6adfc318c61396a14e67120b67
./contracts/gateway/linea/LineaGateway.sol	ca782ab2da9e2ed2d5e89d049d70805e252faa5b8df21437385e72cf2917e7bf
./contracts/gateway/optimism/OptimismGateway.sol	1a0c7e649ea9b3825dde278b44a1057d903c99370b760638428b0856e56a5b3e
./contracts/gateway/scroll/ScrollGateway.sol	7255b904b35a0a6391822f1f8a9e5c9faf1ed4d99fec4803db6ec325dd55773d

./contracts/gateway/L1BaseGateway.sol	d806c7ba7ec73eb7b7c709a7a33051044f7fe5207e893379498cac178a763a3c
./contracts/gateway/L2BaseGateway.sol	562dcf7d64d131f9ed2b7d1c4b9cc99c0126e02cc08245e7d503341fff96aa73
contracts/zksync/l1-contracts/zksync/interfaces/IMailbox.sol	92fbc1bc92c48cc90a96acf213152887e68acb92cab174384a82620a5f4a5866
contracts/zksync/l1-contracts/zksync/Storage.sol	db47f8ffcdc9996a02b42f8f241bd42dc68bf61dc9a4ea8de8876ba332f69426
contracts/zksync/l1-contracts/zksync/interfaces/IGetters.sol	55acb0ffc2e3835ec53df90776d29e675bb14a046ea7acad70b81f998b9d3c67
contracts/zksync/l1-contracts/zksync/interfaces/IAdmin.sol	ad731b468c19da62e10f2efc469e2bb30db6177a1751bb00973624612812267b

Code Assessment Findings



ID	Name	Category	Severity	Client Response	Contributor
ZKL-1	Vulnerability in `withdrawForwardFee` Function Allows Validators to Overdraw Fees	Logical	Critical	Fixed	BradMoonUESTC
ZKL-2	Lack of EIP-712 compliance: using `keccak256()` directly on an array or struct variable	Logical	Medium	Fixed	rajatbeladiya
ZKL-3	Discrepancy in Message Length Validation for L2 Withdrawal Message Parsing	Logical	Medium	Fixed	BradMoonUESTC
ZKL-4	unused return value	Code Style	Low	Fixed	BradMoonUESTC
ZKL-5	Use a 2-step ownership transfer pattern	Logical	Low	Acknowledged	rajatbeladiya, danielt
ZKL-6	Use `string.concat()` or `bytes.concat()` instead of `abi.encodePacked`	Logical	Low	Acknowledged	rajatbeladiya

ZKL-7	Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later	Logical	Low	Fixed	rajatbeladiya
ZKL-8	Miss 0 amount check	Logical	Low	Acknowledged	NoodleDonn212
ZKL-9	Centralization of Control	Privilege Related	Low	Acknowledged	NoodleDonn212
ZKL-10	Add _disableInitializers() to the constructor	Logical	Low	Fixed	NoodleDonn212, comcat, danielt
ZKL-11	Use calldata instead of memory for function arguments that do not get mutated	Gas Optimization	Informational	Fixed	rajatbeladiya

ZKL-1:Vulnerability in `withdrawForwardFee` Function Allows Validators to Overdraw Fees

Category	Severity	Client Response	Contributor
Logical	Critical	Fixed	BradMoonUESTC

Code Reference

- code/contracts/ZkLink.sol#L408-L419

```
408: function withdrawForwardFee(uint256 _amount) external nonReentrant onlyValidator {
409:         require(_amount > 0, "Invalid amount");
410:         uint256 newWithdrawnFee = totalValidatorForwardFeeWithdrawn + _amount;
411:         require(totalValidatorForwardFee >= newWithdrawnFee, "Withdraw exceed");
412:
413:         // Update withdrawn fee
414:         totalValidatorForwardFeeWithdrawn = newWithdrawnFee;
415:         // solhint-disable-next-line avoid-low-level-calls
416:         (bool success, ) = msg.sender.call{value: _amount}("");
417:         require(success, "Withdraw failed");
418:         emit WithdrawForwardFee(_amount);
419:     }
```

Description

BradMoonUESTC: The `withdrawForwardFee` function in the smart contract contains a critical vulnerability that allows validators to withdraw more than their allocated share of forwarding fees, potentially leading to unfair distributions and loss of funds. This function lacks a crucial check to ensure that each validator can only withdraw their rightful share based on the total amount of fees collected and their contribution.

The vulnerability arises from the function's failure to track and limit individual validators' withdrawals according to their proportionate share. The function calculates the new total withdrawn fee by simply adding the requested withdrawal amount (`_amount`) to `totalValidatorForwardFeeWithdrawn`, without considering the requesting validator's entitled share. The only check performed is against the total collected forwarding fees (`totalValidatorForwardFee`), ensuring that the new total withdrawn does not exceed this amount. However, this does not prevent individual validators from withdrawing more than their share.

Here's the problematic part of the code:

```
function withdrawForwardFee(uint256 _amount) external nonReentrant onlyValidator {
 require(_amount > 0, "Invalid amount");
 uint256 newWithdrawnFee = totalValidatorForwardFeeWithdrawn + _amount;
 require(totalValidatorForwardFee >= newWithdrawnFee, "Withdraw exceed");

 totalValidatorForwardFeeWithdrawn = newWithdrawnFee;
 (bool success, ) = msg.sender.call{value: _amount}("");
 require(success, "Withdraw failed");
 emit WithdrawForwardFee(_amount);
}
```

Recommendation

BradMoonUESTC: To mitigate this vulnerability and ensure fair fee distribution among validators, we recommend introducing a mechanism to track each validator's share of the total fees and their withdrawn amount. This can be achieved by maintaining a mapping of validators to their respective shares and amounts withdrawn. The `` `withdrawForwardFee` `` function should then be updated to check the requesting validator's remaining share before proceeding with the withdrawal.
Here's a suggested code snippet to implement the recommended changes:

```
// Mapping to track each validator's share and withdrawn amount
mapping(address => uint256) private validatorShares;
mapping(address => uint256) private validatorWithdrawn;

function setValidatorShare(address validator, uint256 share) external {
    // This function can be used to set the validator's share, ideally called by the contract owner or through a consensus mechanism
    validatorShares[validator] = share;
}

function withdrawForwardFee(uint256 _amount) external nonReentrant onlyValidator {
    require(_amount > 0, "Invalid amount");
    require(validatorShares[msg.sender] > 0, "Validator has no share");

    uint256 availableToWithdraw = validatorShares[msg.sender] - validatorWithdrawn[msg.sender];
    require(_amount <= availableToWithdraw, "Withdrawal amount exceeds available share");

    validatorWithdrawn[msg.sender] += _amount;
    (bool success, ) = msg.sender.call{value: _amount}("");
    require(success, "Withdraw failed");

    emit WithdrawForwardFee(msg.sender, _amount);
}
```

In this revised approach, `` `setValidatorShare` `` allows for the allocation of shares to each validator. The `` `withdrawForwardFee` `` function now checks if the requested amount does not exceed the validator's remaining share (their total share minus what they've already withdrawn). This ensures each validator can only withdraw funds up to their entitled share, effectively mitigating the vulnerability and promoting fairness.

Client Response

BradMoonUESTC: Fixed,Introduce a fee allocator, which can be a contract. We will implement different proportions of fee sharing for different validators within this allocator contract.

- The fix commitment

ZKL-2:Lack of EIP-712 compliance: using `keccak256()` directly on an array or struct variable

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	rajatbeladiya

Code Reference

- code/contracts/ZkLink.sol#L274

```
274: canonicalTxHash = keccak256(abi.encode(request));
```

Description

rajatbeladiya: Directly using the actual variable instead of encoding the array values goes against the EIP-712 specification https://github.com/ethereum/EIPs/blob/master/EIPS/eip-712.md#definition-of-encodedata.
Note: OpenSea's Seaport's example with offerHashes and considerationHashes can be used as a reference to understand how array of structs should be encoded.

Recommendation

rajatbeladiya: change it to

```
function hashForwardL2Request(ForwardL2Request memory _request) internal pure returns (bytes32) {
    return keccak256(
        abi.encode(
            FORWARD_REQUEST_TYPEHASH,
            _request.remoteGateway,
            _request.isContractCall,
            _request.sender,
            _request.txId,
            _request.contractL2,
            _request.l2Value,
            keccak256(_request.l2CallData),
            _request.l2GasLimit,
            _request.l2GasPricePerPubdata,
            keccak256(abi.encodePacked(_request.factoryDeps)),
            _request.refundRecipient
        )
    );
}
```

Client Response

rajatbeladiya: Fixed,The fix commit of zklink-evm-contracts
The fix commit of era-contracts

ZKL-3:Discrepancy in Message Length Validation for L2 Withdrawal Message Parsing

Category	Severity	Client Response	Contributor
Logical	Medium	Fixed	BradMoonUESTC

Code Reference

- code/contracts/ZkLink.sol#L504-L527

```
504: /// @dev Decode the withdraw message that came from L2
505:     function _parseL2WithdrawalMessage(
506:         bytes memory _message
507:     ) internal pure returns (address l1Gateway, uint256 amount, address l1Receiver) {
508:         // We check that the message is long enough to read the data.
509:         // Please note that there are two versions of the message:
510:         // 1. The message that is sent by `withdraw(address _l1Receiver)`
511:         // It should be equal to the length of the bytes4 function signature + address l1Receiver + uint256 amount = 4 + 20 + 32 = 56 (bytes).
512:         // 2. The message that is sent by `withdrawWithMessage(address _l1Receiver, bytes calldata _additionalData)`
513:         // It should be equal to the length of the following:
514:         // bytes4 function signature + address l1Receiver + uint256 amount + address l2Sender + bytes _additionalData =
515:         // = 4 + 20 + 32 + 32 + _additionalData.length >= 68 (bytes).
516:
517:         // So the data is expected to be at least 56 bytes long.
518:         require(_message.length == 108, "pm");
519:
520:         (uint32 functionSignature, uint256 offset) = UnsafeBytes.readUint32(_message, 0);
521:         require(bytes4(functionSignature) == this.finalizeEthWithdrawal.selector, "is");
522:
523:         (l1Gateway, offset) = UnsafeBytes.readAddress(_message, offset);
524:         (amount, offset) = UnsafeBytes.readUint256(_message, offset);
525:         // The additional data is l1 receiver address
526:         (l1Receiver, offset) = UnsafeBytes.readAddress(_message, offset + 32);
527:     }
```

Description

BradMoonUESTC: Upon reviewing the smart contract function `_parseL2WithdrawalMessage`, a discrepancy has been identified between the comments/documentation and the actual implementation regarding the expected length of the L2 withdrawal message. According to the comments, the message length is expected to be at least 56 bytes to accommodate the basic withdrawal information or more to include additional data. However, the implemented requirement strictly enforces the message length to be exactly 108 bytes (`require(_message.length == 108, "pm");`). This rigid validation does not align with the explained variable lengths and could potentially lead to issues with message parsing, especially in scenarios where additional data is present or the basic data structure changes.

Recommendation

BradMoonUESTC: To resolve this discrepancy and ensure flexibility in handling L2 withdrawal messages of variable lengths, it is recommended to adjust the length validation logic to check for a minimum length of 56 bytes, thus allowing for messages with additional data beyond the basic withdrawal information. This approach accommodates variable-sized additional data while ensuring that the necessary information for withdrawal is present.

The following code snippet provides an updated version of the length check, replacing the strict equality check with a more flexible condition that ensures the message contains at least the essential data:

```
// Ensure the message is at least 56 bytes to contain the necessary withdrawal information
require(_message.length >= 108, "Invalid message length; must be at least 56 bytes.");

// Additional logic to handle variable-sized _additionalData, if applicable
```

Additionally, if the protocol requires handling messages of specific lengths beyond the basic requirements (e.g., exactly 108 bytes for certain operations), it is advised to document the rationale clearly in the comments and implement conditional logic to handle such cases explicitly. This approach ensures clarity for developers and auditors and maintains flexibility for handling messages of varying lengths.

Client Response

BradMoonUESTC: Fixed,The length of the message must be 108 bits.
The fix commit

ZKL-4:unused return value

Category	Severity	Client Response	Contributor
Code Style	Low	Fixed	BradMoonUESTC

Code Reference

- code/contracts/gateway/arbitrum/ArbitrumL1Gateway.sol#L44-L54

```
44: INBOX.createRetryableTicket{value: msg.value}(
45:             remoteGateway,
46:             _value,
47:             maxSubmissionCost,
48:             // solhint-disable-next-line avoid-tx-origin
49:             tx.origin,
50:             remoteGateway,
51:             gasLimit,
52:             maxFeePerGas,
53:             data
54:         );
```

- code/contracts/gateway/arbitrum/ArbitrumL2Gateway.sol#L33

```
33: ARB_SYS.sendTxToL1{value: _value}(remoteGateway, message);
```

- code/contracts/gateway/zksync/ZkSyncL1Gateway.sol#L39-L48

```
39: MESSAGE_SERVICE.requestL2Transaction{value: msg.value}(
40:             remoteGateway,
41:             _value,
42:             executeData,
43:             _l2GasLimit,
44:             _l2GasPerPubdataByteLimit,
45:             new bytes[](0),
46:             // solhint-disable-next-line avoid-tx-origin
47:             tx.origin
48:         );
```

- code/contracts/gateway/zksync/ZkSyncL2Gateway.sol#L45

```
45: L2_MESSENGER.sendToL1(message);
```

Description

BradMoonUESTC: Either the return value of an external call is not stored in a local or state variable, or the return value is declared but never used in the function body.

Recommendation

BradMoonUESTC: Ensure the return value of external function calls is used. Remove or comment out the unused return function parameters.

Client Response

BradMoonUESTC: Fixed,The fix commit

ZKL-5:Use a 2-step ownership transfer pattern

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	rajatbeladiya, danielt

Code Reference

- code/contracts/Arbitrator.sol#L17

```
17: contract Arbitrator is IArbitrator, OwnableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable {
```

- code/contracts/gateway/BaseGateway.sol#L9

```
9: abstract contract BaseGateway is IGateway, OwnableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable {
```

- code/contracts/ZkLink.sol#L5
- code/contracts/ZkLink.sol#L32

```
5: import {OwnableUpgradeable} from "@openzeppelin/contracts-upgradeable/access/OwnableUpgradeable.sol";
```

```
32: OwnableUpgradeable,
```

Description

rajatbeladiya: Recommend considering implementing a two step process where the owner or admin nominates an account and the nominated account needs to call an `` `acceptOwnership()` `` function for the transfer of ownership to fully succeed. This ensures the nominated EOA account is a valid and active account. Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two step procedure on the critical functions.
danielt: In contract `` `ZkLink` ``, it is possible that the `` `owner` `` role transfers ownership to the wrong address by mistake, resulting in authorization loss from the team.

Recommendation

rajatbeladiya: Use Openzeppelin's `` `Ownable2StepUpgradeable.sol` ``
danielt: Consider using the `` `Ownable2StepUpgradeable` `` contract for a two-step ownership transfer.

Client Response

rajatbeladiya: Acknowledged,All owners will be transferred to the Governor contract after the protocol deployment is completed. We will carefully to avoid transferring to a wrong address.
danielt: Acknowledged,All owners will be transferred to the Governor contract after the protocol deployment is completed. We will carefully to avoid transferring to a wrong address.

ZKL-6:Use `string.concat()` or `bytes.concat()` instead of `abi.encodePacked`

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	rajatbeladiya

Code Reference

- code/contracts/ZkLink.sol#L283
- code/contracts/ZkLink.sol#L488

```
283: syncStatus.hash = keccak256(abi.encodePacked(syncStatus.hash, canonicalTxHash));
```

```
488: abi.encodePacked(_log.l2ShardId, _log.isService, _log.txNumberInBatch, _log.sender, _log.key, _
log.value)
```

Description

rajatbeladiya: Solidity version 0.8.4 introduces `` `bytes.concat()` `` (vs `` `abi.encodePacked(<bytes>,<bytes>)` ``) Solidity version 0.8.12 introduces `` `string.concat()` `` (vs `` `abi.encodePacked(<str>,<str>), which catches concatenation errors (in the event of a `bytes` data mixed in the concatenation)` ``)

Recommendation

rajatbeladiya: use `` `string.concat()` `` or `` `bytes.concat()` `` instead of `` `abi.encodePacked` ``

Client Response

rajatbeladiya: Acknowledged,According to the solidity documentation, the parameter types of bytes.concat are limited to bytes or bytes1,..,bytes32

```
If you want to use string parameters or other types that are not implicitly convertible to bytes,
you need to convert them to bytes or bytes1/…/bytes32 first.
```

We don't see any obvious benefit to replacing it.

ZKL-7:Upgradeable contract is missing a `__gap[50]` storage variable to allow for new storage variables in later

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	rajatbeladiya

Code Reference

- code/contracts/Arbitrator.sol#L6

```
6: import {UUPSUpgradeable} from "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
```

- code/contracts/ZkLink.sol#L6

```
6: import {UUPSUpgradeable} from "@openzeppelin/contracts-upgradeable/proxy/utils/UUPSUpgradeable.sol";
```

Description

rajatbeladiya: `__gap[50]` storage variable is used to allow for new storage variables later. See upgradeable#storage_gaps for a description of this storage variable. While some contracts may not currently be sub-classed, adding the variable now protects against forgetting to add it in the future.

Recommendation

rajatbeladiya: add `__gap[50]` storage variable to allow for new storage variables in upgradeable contracts

Client Response

rajatbeladiya: Fixed,The fix commit

ZKL-8:Miss 0 amount check

Category	Severity	Client Response	Contributor
Logical	Low	Acknowledged	NoodleDonn212

Code Reference

- code/contracts/ZkLink.sol#L46
- code/contracts/ZkLink.sol#L172
- code/contracts/ZkLink.sol#L236
- code/contracts/ZkLink.sol#361

```
46: uint256 public txGasPrice;
```

```
172: function setTxGasPrice(uint256 _newTxGasPrice) external onlyOwner {
```

```
236: uint256 l2GasPrice = _deriveL2GasPrice(txGasPrice, _l2GasPerPubdataByteLimit);
```

```
361: function syncL2Requests(uint256 _newTotalSyncedPriorityTxs) external payable onlyValidator {
```

Description

NoodleDonn212: ##### Summary
User can pass 0 as the _value parameter in the sendMessage method and claimMessageCallback method. The contract does not have any restrictions on the _value being greater than 0. The _value represents the amount of Ether (in wei) that will be sent along with the message. If _value is 0, it simply means no Ether is being bridged with the message.
Vulnerability Details
User can pass 0 as the _value parameter in the sendMessage method of every gateway. The contract does not have any restrictions on the _value being greater than 0. The _value represents the amount of Ether (in wei) that will be sent along with the message. If _value is 0, it simply means no Ether is being bridged with the message.
A user can also pass 0 as the _value parameter in the claimMessageCallback method of every gateway. However, since this method is marked as payable and includes a require statement checking that msg.value equals _value, the transaction must also carry a value of 0 ETH for it to succeed. If the transaction's msg.value does not match the _value parameter, the transaction will revert.
Impact
not adhering to CEI or check effects interaction pattern , like validating every input before changing state can cause incorrect calculations in the forwarding of eth amount when calling syncL2Requests() method of zkLink.sol.
NoodleDonn212: Summary:
The owner can set the txGasPrice to zero. This action would result in the l2GasPrice also being calculated as zero within the requestL2Transaction() method, allowing transactions to be processed without any cost.
Vulnerability Details
The setTxGasPrice() function allows the contract owner to update the txGasPrice state variable. There are no current safeguards to prevent the txGasPrice from being set to zero. The requestL2Transaction() method relies on txGasPrice to calculate the cost of executing transactions on Layer 2, which means setting txGasPrice to zero would effectively eliminate transaction fees.
Impact
The implications of this vulnerability are significant:

Validators would lose their incentives to process transactions, potentially halting network operations.
The network could be susceptible to spam and denial-of-service attacks due to the absence of transaction fees.
The economic model of the network would be compromised, affecting the sustainability and security of the system.
The quality of service for legitimate users could degrade due to an influx of low-value transactions.

Recommendation

NoodleDonn212: use proper require statements to validate that the _amount is greater than zero. In both the sendMessage() and claimMessageCallback() methods of every gateway.
NoodleDonn212: Recommendations
Implement a minimum threshold for txGasPrice to ensure that transaction fees cannot be eliminated entirely. This threshold should be carefully considered to balance validator compensation with user transaction costs.
Add validation logic to the setTxGasPrice() function to reject attempts to set txGasPrice to zero, ensuring that there is always some cost associated with Layer 2 transactions.
// Check that the new transaction gas price is greater than zero
require(_newTxGasPrice > 0, "Tx gas price must be greater than zero");

Client Response

NoodleDonn212: Acknowledged,Currently txGasPrice is allowed to be 0 for the following reasons:

```
Reduce users’ recharge costs (disguised subsidies)
Even if txGasPrice is 0, the value returned by _deriveL2GasPrice will not be 0. In the mainnet env
ironment, it is 0.25Gwei(payed for tx executed on zkLink), which can prevent DDOS attacks to a cer
tain extent (they also need to pay for tx fee)
In the future we may use a mechanism similar to EIP1559 to adjust txGasPrice on the secondary chai
n
```

ZKL-9:Centralization of Control

Category	Severity	Client Response	Contributor
Privilege Related	Low	Acknowledged	NoodleDonn212

Code Reference

- code/contracts/gateway/arbitrum/ArbitrumL2Gateway.sol#L36

```
36: function claimMessageCallback(uint256 _value, bytes memory _callData) external payable onlyRemoteGateway {
```

- code/contracts/gateway/BaseGateway.sol#11
- code/contracts/gateway/BaseGateway.sol#40

```
11: address internal remoteGateway;
```

```
40: function setRemoteGateway(address _remoteGateway) external onlyOwner {
```

- code/contracts/ZkLink.sol#L40
- code/contracts/ZkLink.sol#L159
- code/contracts/ZkLink.sol#L178
- code/contracts/ZkLink.sol#L362
- code/contracts/ZkLink.sol#L408

```
40: IL2Gateway public gateway;
```

```
159: function setGateway(IL2Gateway _gateway) external onlyOwner {
```

```
178: function setValidator(address _validator, bool _active) external onlyGateway {
```

```
362: // Check newTotalSyncedPriorityTxs
```

```
408: function withdrawForwardFee(uint256 _amount) external nonReentrant onlyValidator {
```

Description

NoodleDonn212: Summary:
The BaseGateway contract allows the owner to set the remoteGateway address, which is critical for the security of the ArbitrumL1Gateway. If the owner sets themselves or an externally owned account (EOA) they control as the remoteGateway, they could potentially bypass the intended security checks
Impact
If exploited, the vulnerability could allow the contract owner to bypass security mechanisms, enabling them to claim messages and execute arbitrary logic on L1 that should only be allowed by a legitimate L2 gateway. This could lead to loss of funds or unauthorized actions on the L1 chain.
NoodleDonn212: Summary:

Owner can set themselves as the gateway and a validator, thereby centralizing control and enabling the owner to unilaterally call withdrawForwardFee and syncL2Requests.

Vulnerability Details

The zkLink system utilizes a gateway state variable intended for communication with L1. The onlyGateway modifier ensures that only the address stored in gateway can perform certain actions, such as setting validators. The owner has the authority to set the gateway address, and if the owner sets a gateway or EOA under their control, they can then designate themselves as a validator. As a validator, the owner could potentially withdraw accumulated fees without checks and balances, which is facilitated by the withdrawForwardFee function.

POC:

```
contract ZkLinkTest is Test {
address owner;
address fakeGateway;
address validator;
ZkLink zkLink;
```

```
 function setUp() public {
   zkLink = new ZkLink();
   owner = address(this); // The test contract itself acts as the owner for simplicity
   fakeGateway = address(0xBEEF); // An address we control simulating a malicious gateway
   validator = address(0xFEED); // An address we control simulating a malicious validator

 }

  function testOwnerSetsMaliciousGatewayAndValidator() public {
   // Simulate the owner setting a malicious gateway
   vm.prank(owner);
   zkLink.setGateway(fakeGateway);

   // Simulate the malicious gateway setting itself as a validator
   vm.prank(fakeGateway);
   zkLink.setValidator(validator);

   // Assert that the validator was set correctly
   bool isValidatorSet = zkLink.validators(validator);
   assertTrue(isValidatorSet, "Validator should be set");

  }

    // Simulate the validator withdrawing fees
   uint256 withdrawAmount = 1 ether; // Assume there's enough balance
   vm.prank(validator);
   zkLink.withdrawForwardFee(withdrawAmount);

   // Assert that the fees were withdrawn
 withdrawnFees = zkLink.totalValidatorForwardFeeWithdrawn(); assertEq(withdrawnFees, withdrawAmoun
t, "Fees should be withdrawn by the validator");

// Check the balance of the validator to ensure it received the funds
uint256 validatorBalance = address(validator).balance;
assertEq(validatorBalance, withdrawAmount, "Validator should receive the withdrawn fees");
```

}

In this PoC:

1. We simulate the owner setting a malicious gateway by calling **`setGateway`** with a fake gateway address we control.
2. We then simulate the malicious gateway setting itself as a validator by calling **`setValidator`**.
3. We assert that the validator was set correctly by checking the **`validators`** mapping.
4. We simulate the validator withdrawing fees by calling **`withdrawForwardFee`**.
5. We assert that the fees were withdrawn by checking **`totalValidatorForwardFeeWithdrawn`**.
6. We check the balance of the validator to ensure it received the funds.

This test demonstrates the vulnerability where the owner can set themselves as both the gateway and a validator, and then withdraw fees, indicating a serious security issue that needs to be addressed.
Impact
If exploited, this vulnerability could lead to:
Misappropriation of funds through withdrawForwardFee. Loss of trust in the zkLink system due to centralization concerns. Potential for censorship or preferential treatment in transaction processing and damage to the system's reputation and possibly to the value of associated tokens.

Recommendation

NoodleDonn212: Recommendations To address the potential vulnerability of centralization and unauthorized fee withdrawal, it is recommended to implement additional security checks within the onlyGateway modifier to align with best practices for decentralized systems:
Multi-Party Governance: Ensure that the process of setting the gateway involves a multi-party governance mechanism, such as a multi-signature wallet or a DAO, to prevent any single entity from having unilateral control.
NoodleDonn212: Recommendations:
To address the potential vulnerability of centralization and unauthorized fee withdrawal, it is recommended to implement additional security checks within the onlyGateway modifier to align with best practices for decentralized systems:
Enhanced Gateway Verification: Introduce a check to validate that the gateway address is an authorized and secure bridge contract, similar to the ArbitrumL1Gateway's approach of verifying the msg.sender against the bridge contract's address.
EXAMPLE.
https://github.com/Secure3Audit/code_zklink_L3/blob/main/code/contracts/gateway/arbitrum/ArbitrumL1Gateway.sol
IBridge bridge = INBOX.bridge();
require(msg.sender == address(bridge), "Not bridge");
https://github.com/Secure3Audit/code_zklink_L3/blob/main/code/contracts/Arbitrator.sol#99
Ex:
require(_gateway == primaryChainGateway || secondaryChainGateways[_gateway], "Invalid gateway");
Multi-Party Governance: Ensure that the process of setting the gateway involves a multi-party governance mechanism, such as a multi-signature wallet or a DAO, to prevent any single entity from having unilateral control.
Transparent Process: Create a transparent and auditable process for gateway and validator changes, allowing stakeholders to monitor and verify the legitimacy of these actions.
Cross-Chain Authentication: Adopt a cross-chain authentication mechanism that verifies the original sender of a message, ensuring that only messages from an authorized L2 gateway can invoke the onlyGateway protected functions.

Client Response

NoodleDonn212: Acknowledged,All remoteGateway settings will be set immediately after the protocol deployment is completed, and are only allowed to be set once

ZKL-10:Add _disableInitializers() to the constructor

Category	Severity	Client Response	Contributor
Logical	Low	Fixed	NoodleDonn212, comcat, danielt

Code Reference

- code/contracts/Arbitrator.sol#L17

```
17: contract Arbitrator is IArbitrator, OwnableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable {
```

- code/contracts/gateway/BaseGateway.sol#L9

```
9: abstract contract BaseGateway is IGateway, OwnableUpgradeable, UUPSUpgradeable, ReentrancyGuardUpgradeable {
```

- code/contracts/ZkLink.sol#L27
- code/contracts/ZkLink.sol#L111

```
27: contract ZkLink is
```

```
111: function initialize() external initializer {
```

Description

NoodleDonn212: Summary
The BaseGateway contract is intended to be part of an upgradeable system but lacks a constructor that calls _disableInitializers. This omission could potentially allow re-initialization of the contract, leading to vulnerabilities.
Vulnerability Details
The BaseGateway contract uses OpenZeppelin's upgradeable contract pattern but does not include a constructor that invokes _disableInitializers. Initializers in upgradeable contracts are used to replace constructors and must be protected to prevent being called more than once. The absence of _disableInitializers means that if the contract or any derived contract does not correctly implement a constructor to disable initializers, the contract's state could be reset or manipulated through re-initialization.
Impact
If exploited, an attacker could re-initialize the contract, potentially resetting the owner or other critical state variables. This could result in unauthorized control over the contract, loss of funds, or disruption of service.
comcat: ZkLink is implemented as a UUPS contract, but the initialization function has been left exposed. As is commonly understood, the UUPS contract upgrade pattern depends on the implementing contract for upgrades. If the ZkLink contract is deployed and a UUPS proxy is initialized to point to the ZkLink implementation, but fails to invoke the initialize function within the implementation contract, this means that anyone could call the ZkLink initialization function and gain administrative access to the ZkLink implementation contract, potentially causing severe damage.
danielt: The contracts `ZkLink`, and `Arbitrator` are intended to be used as implementation contracts. However, they are still able to be initialized, due to lacking a call to `_disableInitializers` function from the constructor. It is recommended not to leave an implementation contract uninitialized, to prevent other attack vectors. See the

document of OpenZepplin for the _disableInitializers function:
Locks the contract, preventing any future reinitialization. This cannot be part of an initializer call. Calling this in the constructor of a contract will prevent that contract from being initialized or reinitialized to any version. It is recommended to use this to lock implementation contracts that are designed to be called through proxies.

Recommendation

NoodleDonn212: Implement a constructor in the BaseGateway contract or ensure that derived contracts have a constructor that calls _disableInitializers. This should be done to prevent any possibility of re-initialization. Additionally, thorough testing and auditing should be conducted on the final contract implementation to ensure that initializers cannot be called more than once.
comcat: Add a constructor and ensure that the constructor performs the initialization tasks
danielt: Consider using **`` `disableInitializers` ``** to lock an implementation contract.

Client Response

NoodleDonn212: Fixed,We add disableInitializers in ZkLink, the commit.
We believe that if it is not necessary to initialize some immutable variables in the constructor, it is best not to add a constructor, so there is no need to use disableInitializers
comcat: Fixed,We add disableInitializers in ZkLink, the commit.
We believe that if it is not necessary to initialize some immutable variables in the constructor, it is best not to add a constructor, so there is no need to use disableInitializers
danielt: Fixed,We add disableInitializers in ZkLink, the commit.
We believe that if it is not necessary to initialize some immutable variables in the constructor, it is best not to add a constructor, so there is no need to use disableInitializers

ZKL-11:Use calldata instead of memory for function arguments that do not get mutated

Category	Severity	Client Response	Contributor
Gas Optimization	Informational	Fixed	rajatbeladiya

Code Reference

- code/contracts/Arbitrator.sol#L76
- code/contracts/Arbitrator.sol#L119

```
76: bytes memory _adapterParams
```

```
119: function receiveMessage(uint256 _value, bytes memory _callData) external payable {
```

- code/contracts/gateway/arbitrum/ArbitrumL1Gateway.sol#L36
- code/contracts/gateway/arbitrum/ArbitrumL1Gateway.sol#L37
- code/contracts/gateway/arbitrum/ArbitrumL1Gateway.sol#L57

```
36: bytes memory _callData,
```

```
37: bytes memory _adapterParams
```

```
57: function claimMessageCallback(uint256 _value, bytes memory _callData) external payable onlyRemot
eGateway {
```

- code/contracts/ZkLink.sol#L322

```
322: L2Message memory _message,
```

Description

rajatbeladiya: 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.

Recommendation

rajatbeladiya: Use calldata instead of memory for function arguments that do not get mutated
Change it to,

```
function setSecondaryChainGateway(
        IL1Gateway _gateway,
        bool _active,
        bytes calldata _adapterParams
    )
```

instead of

```
function setSecondaryChainGateway(
        IL1Gateway _gateway,
        bool _active,
        bytes memory _adapterParams
    )
```

Client Response

rajatbeladiya: Fixed,The fix commit

Disclaimer

This report is subject to the terms and conditions (including without limitation, description of services, confidentiality, disclaimer and limitation of liability) set forth in the Invoices, or the scope of services, and terms and conditions provided to you (“Customer” or the “Company”) in connection with the Invoice. This report provided in connection with the services set forth in the Invoices shall be used by the Company only to the extent permitted under the terms and conditions set forth in the Invoice. This report may not be transmitted, disclosed, referred to or relied upon by any person for any purposes, nor may copies be delivered to any other person other than the Company, without Secure3’s prior written consent in each instance.

This report is not an “endorsement” or “disapproval” of any particular project or team. This report is not an indication of the economics or value of any “product” or “asset” created by any team or project that contracts Secure3 to perform a security assessment. This report does not provide any warranty or guarantee of free of bug of codes analyzed, nor do they provide any indication of the technologies, business model or legal compliancy.

This report should not be used in any way to make decisions around investment or involvement with any particular project. Instead, it represents an extensive assessing process intending to help our customers increase the quality of their code and high-level consistency of implementation and business model, while reducing the risk presented by cryptographic tokens and blockchain technology.

Secure3’s position on the final decisions over blockchain technologies and corresponding associated transactions is that each company and individual are responsible for their own due diligence and continuous security.

The assessment services provided by Secure3 is subject to dependencies and under continuing development. The assessment reports could include false positives, false negatives, and other unpredictable results. The services may access, and depend upon, multiple layers of third-parties.