HashEx

BLOCKCHAIN SECURITY

Asterizm

smart contracts
preliminary audit report
for internal use only

March 2023

hashex.org

contact@hashex.org

Contents

1. Disclaimer

This is a limited report on our findings based on our analysis, in accordance with good industry practice at the date of this report, in relation to cybersecurity vulnerabilities and issues in the framework and algorithms based on smart contracts, the details of which are set out in this report. In order to get a full view of our analysis, it is crucial for you to read the full report. While we have done our best in conducting our analysis and producing this report, it is important to note that you should not rely on this report and cannot claim against us on the basis of what it says or doesn’t say, or how we produced it, and it is important for you to conduct your own independent investigations before making any decisions. We go into more detail on this in the disclaimer below – please make sure to read it in full.

By reading this report or any part of it, you agree to the terms of this disclaimer. If you do not agree to the terms, then please immediately cease reading this report, and delete and destroy any and all copies of this report downloaded and/or printed by you. This report is provided for information purposes only and on a non-reliance basis and does not constitute investment advice. No one shall have any right to rely on the report or its contents, and HashEx and its affiliates (including holding companies, shareholders, subsidiaries, employees, directors, officers, and other representatives) (HashEx) owe no duty of care towards you or any other person, nor does HashEx make any warranty or representation to any person on the accuracy or completeness of the report. The report is provided "as is", without any conditions, warranties, or other terms of any kind except as set out in this disclaimer, and HashEx hereby excludes all representations, warranties, conditions, and other terms (including, without limitation, the warranties implied by law of satisfactory quality, fitness for purpose and the use of reasonable care and skill) which, but for this clause, might have effect in relation to the report. Except and only to the extent that it is prohibited by law, HashEx hereby excludes all liability and responsibility, and neither you nor any other person shall have any claim against HashEx, for any amount or kind of loss or damage that may result to you or any other person (including without limitation, any direct, indirect, special, punitive, consequential or pure economic loss or damages, or any loss of income, profits, goodwill, data, contracts, use of money, or business interruption, and whether in delict, tort (including without limitation negligence), contract, breach of statutory duty, misrepresentation (whether innocent or negligent) or otherwise under any claim of any nature whatsoever in any jurisdiction) in any way arising from or connected with this report and the use, inability to use or the results of the use of this report, and any reliance on this report. The analysis of the security is purely based on the smart contracts alone. No applications or operations were reviewed for security. No product code has been reviewed. HashEx owns all copyright rights to the text, images, photographs, and other content provided in the following document. When using or sharing partly or in full, third parties must provide a direct link to the original document mentioning the author (hashex.org).

2. Overview

HashEx was commissioned by the Asterizm team to perform an audit of their smart contract. The audit was conducted between 13/03/2023 and 18/03/2023.

The purpose of this audit was to achieve the following:

- Identify potential security issues with smart contracts
- Formally check the logic behind given smart contracts.

Information in this report should be used for understanding the risk exposure of smart contracts, and as a guide to improving the security posture of smart contracts by remediating the issues that were identified.

The code is available at @Asterizm-Layer/asterizm-contracts GitHub repository and was audited after the commit a8d4c2.

Update. A recheck was done after the commit 206378.

Update 2. A second recheck was done after the commit 663972c.

Update 3. A third recheck was done after the commit a6d1e5e.

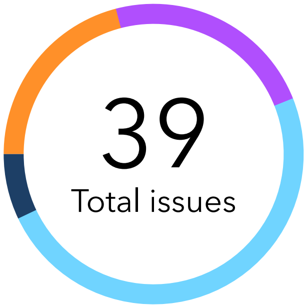
2.1 Summary

Project name	Asterizm
URL	https://asterizm.io/
Platform	Ethereum
Language	Solidity

2.2 Contracts

Name	Address
AsterizmClient	
AsterizmInitializer	
AsterizmNonce	
AsterizmTranslator	
Claimer	
GasSender	
MultichainToken	

3. Found issues



●	High	8 (21%)
●	Medium	9 (23%)
●	Low	19 (49%)
●	Info	3 (7%)

C1. AsterizmClient

ID	Severity	Title	Status
C1-01	● High	txId not incremented on sending outbound message	✓ Resolved
C1-02	● High	Lack of documentation on implementation risks	✓ Acknowledged
C1-03	● Medium	Unused function	✓ Resolved
C1-04	● Medium	Unclear purpose of encryption mechanics	✓ Acknowledged
C1-05	● Medium	Lack of checks for transferHash and txId	✓ Resolved
C1-06	● Medium	Risky authorization mechanics	✓ Resolved
C1-07	● Low	Side effect variable update	✓ Resolved
C1-08	● Low	Not used onlyAdmin access modifier	✓ Resolved
C1-09	● Low	Meaningless onlyTrustedTransfer check	✓ Acknowledged
C1-10	● Low	Unnecessary access modificator	✓ Resolved

C1-11	● Low	Lack of events	✓ Resolved
C1-12	● Low	Gas optimization	✓ Resolved
C1-13	● Info	Possibility to turn off cross-chain validation	? Open

C2. AsterizmInitializer

ID	Severity	Title	Status
C2-01	● High	No duplicate transaction id check	✓ Resolved
C2-02	● High	Cross client messages are allowed	✓ Resolved
C2-03	● Medium	Discrepancy in nonce calculation	✓ Resolved
C2-04	● Medium	Different nonce incrementation logic for inbound and outbound messages	✓ Resolved
C2-05	● Medium	The receive success flag is set even if the client failed to execute the message	✓ Resolved
C2-06	● Low	Lack of reentrancy checks	✓ Resolved
C2-07	● Info	Typos	✓ Resolved

C3. AsterizmNonce

ID	Severity	Title	Status
C3-01	● Low	Lack of events	✓ Resolved

C4. AsterizmTranslator

ID	Severity	Title	Status
C4-01	High	Dangerous authorisation mechanics	Resolved
C4-02	High	High centralization	Resolved
C4-03	Medium	Duplicate nonce calculation	Resolved
C4-04	Low	Possibility of removing local chain id from the list of chains	Resolved
C4-05	Low	Ability to change local chain id	Resolved
C4-06	Low	Lack of events	Resolved

C5. Claimer

ID	Severity	Title	Status
C5-01	Low	Gas optimization	Partially fixed

C6. GasSender

ID	Severity	Title	Status
C6-01	Low	Token transfer results are not checked	Resolved
C6-02	Low	Gas optimizations	Acknowledged
C6-03	Low	Discrepancy in outbound and inbound message format	Acknowledged
C6-04	Low	Lack of reentrancy checks	Resolved

C7. MultichainToken

ID	Severity	Title	Status
C7-01	● High	No check for the reuse of cross-chain transfer	✓ Resolved
C7-02	● High	Open mint for the owner	✓ Resolved
C7-03	● Medium	isEncoded parameter is always false	✓ Resolved
C7-04	● Low	No revert message	✓ Resolved
C7-05	● Low	Gas optimization	✓ Resolved
C7-06	● Low	Lack of events	✓ Partially fixed
C7-07	● Info	Unused parameters	✓ Resolved

4. Contracts

C1. AsterizmClient

Overview

An abstract contract with internal methods for sending and receiving messages. Should be inherited by the client's contract.

Issues

C1-01 txId not incremented on sending outbound message ● High ✓ Resolved

The function `initAsterizmTransfer()` does not increment txId. Calling this function several times will result in messages will be sent with the same `txId`.

Recommendation

Increment the `txId` when sending the outbound message with the `initAsterizmTransfer()` function.

C1-02 Lack of documentation on implementation risks ● High ✓ Acknowledged

The contract is supposed to be inherited by the client's implementation. These implementations may involve security risks such as unauthorized calls. For example, a client implements receive function which allows calling an external contract with arbitrary data if the contract gets the corresponding cross-chain message. In such a case an attacker would be able to call any other contract functions with privilege restrictions for the client's implementation contact. An example of such a call may be a transfer of approved tokens to the client's implementation contract from a user to an attacker's address.

Recommendation

Add NatSpec documentation for the contract and its function describing possible security complications in implementation details.

C1-03 Unused function

● Medium ✓ Resolved

The contract doesn't use the function `_validTransferHash()` anywhere. It should be used in functions `_initAsterizmTransferInternal()`, `asterizmIzReceive()`, and `asterizmClReceive()` to check the incoming `transferHash` parameter.

C1-04 Unclear purpose of encryption mechanics

● Medium ✓ Acknowledged

The documentation provided in the repository says that to send an encrypted message a client needs to call `_generateSendingEvent()` passing a destination address and a payload to be executed on the destination chain. Then the client application will pick up this event, encode the payload and execute `_sendMessage()` function to propagate the encoded message to the destination chain. The first step reveals all information about the payload. An attacker can listen for the same event as the client application and he will know the value of the encoded payload passed to the `_sendMessage()` function.

It should be also noted that no decryption mechanisms are implemented in the smart contract on receiving messages in the destination chain.

Recommendation

Change the encryption mechanism so that unencrypted messages won't appear on the public chain before decryption.

C1-05 Lack of checks for transferHash and txId

● Medium ✓ Resolved

Parameters `txId` and `transferHash` for an Asterizm transfer can be any value, even non-valid. The contract BaseAsterizmClient should set these values by itself using the global variable txId and the function `_buildTransferHash()`.

Recommendation

Add missing checks.

C1-06 Risky authorization mechanics ● Medium ✓ Resolved

The contract has a onlyTrustedSrcAddress() modifier which works as a whitelist for addresses in `ClAsterizmReceiveRequestDto` message.

```
    modifier onlyTrustedSrcAddress(uint64 _chainId, address _address) {
        if (trustedAddressCount > 0) {
            require(trustedSrcAddresses[_chainId] == _address, "BaseAsterizmClient: wrong
source address");
        }
        _;
    }
```

It only checks if the list of trusted addresses has one element or more. If the owner accidentally deletes the last address in the whitelist, the modifier will allow all addresses.

Recommendation

If no addresses added to the whitelist the modifier should fail.

C1-07 Side effect variable update ● Low ✓ Resolved

`setExecutedOutboundTransfer()` modifier sets `successReceive` to `true` as a side effect. Side effects may lead to unexpected bugs when changing the contract's code.

C1-08 Not used onlyAdmin access modifier ● Low ✓ Resolved

The modifier `onlyAdmin()` is not used anywhere in the contract. There is no documentation on how it should be used in the inherited contracts as well.

C1-09 Meaningless onlyTrustedTransfer check ● Low ✓ Acknowledged

The function `_asterizmReceiveEncoded()` called when an inbound message is received and checks that the Initializer has processed this message. But the function can only be called by the Initializer.

```
    function asterizmIzReceive(ClAsterizmReceiveRequestDto calldata _dto) external
onlyInitializer nonReentrant {
        _asterizmReceiveEncoded(_dto);
    }

    function _asterizmReceiveEncoded(ClAsterizmReceiveRequestDto calldata _dto) private
        onlyOwnerOrInitializer
        onlyTrustedSrcAddress(_dto.srcChainId, _dto.srcAddress)
        onlyTrustedTransfer(_dto.transferHash)
        onlyNonExecuted(_dto.transferHash)
        setReceiveTransfer(_dto.transferHash)
    {
        emit EncodedPayloadReceivedEvent(_dto.srcChainId, _dto.srcAddress, _dto.nonce,
_dto.txId, _dto.transferHash, _dto.payload);
    }
```

The onlyTrustedTransfer() modifier checks, if the hash is valid via calling the Initializer.

```
    modifier onlyTrustedTransfer(bytes32 _transferHash) {
        require(initializerLib.validIncomeTransferHash(_transferHash),
"BaseAsterizmClient: transfer hash is invalid");
        _;
    }
```

Recommendation

Make all necessary checks in the AsterizmInitializer contract.

C1-10 Unnecessary access modificator

● Low ✓ Resolved

Private functions `_setLocalChainId()`, `_setInitializer()`, `_setUseForceOrder()`, `_setDisableHashValidation()` have `onlyOwner` access modificators. Having a modificator in a private function is considered a bad practice which may lead to unexpected bugs when modifying the code of the smart contract later.

C1-11 Lack of events

● Low ✓ Resolved

The function `setIsEncoded()` , `setForceOrdered()`, `addAdmin()`, `removeAdmin()`, `setInitializer()`, `asterismReceive()` don't emit events, which complicates the tracking of important changes off-chain.

C1-12 Gas optimization

● Low ✓ Resolved

There is no need for the `initializer` global variable. The global variable `initializerLib` can replace it

C1-13 Possibility to turn off cross-chain validation

● Info ? Open

The contract has the possibility to be configured with turned-off validation for messages.

```
    constructor(IInitializerSender _initializerLib, bool _useForceOrder, bool
_disableHashValidation) {
        _setInitializer(_initializerLib);
        _setLocalChainId(initializerLib.getLocalChainId());
        _setUseForceOrder(_useForceOrder);
        _setDisableHashValidation(_disableHashValidation);
    }
```

With a turned-off hash validation, the contracts do not enforce that the same message is executed in the destination chain as in the source chain, so the owner of the client contract has the possibility to execute arbitrary messages.

C2. AsterizmInitializer

Overview

Gets messages from the client's contract and passes them to the AsterizmTranslator contract. Handles nonces of the cross-chain messages.

Issues

C2-01 No duplicate transaction id check

● High ✓ Resolved

The functions `receivePayload()` and `receiveEncodedPayload()` don't check if the transaction id passed to them was already used. This may lead to possible duplication of the transactions in the destination chain if a relayer accidentally sends a transaction twice.

Recommendation

Check if the transaction id has already been processed.

C2-02 Cross client messages are allowed

● High ✓ Resolved

The AstermizmInitializer contract can be used with several client implementations of the AsterismClient.

```
    modifier onlyClient() {
        if (!availableForAllClients) {
            require(clients[msg.sender].exists, "AsterizmInitializer: only client");
        }
        _;
    }
```

If any of these clients have privileged access provided for the AsterizmInitializer contract, other clients can form transactions to be executed on it with privileged rights from the AsterizmInitializer contract.

The vulnerability can be accessed through the function `send()`. A client can send any payload to any destination address in any chain id.

```
    function send(uint64 _dstChainId, address _destination, uint _transactionId, bool
_isEncoded, bool _forceOrdered, bytes calldata _payload) external payable onlyClient {
        _isEncoded ? require(isEncSendAvailable, "AsterizmInitializer: encode transfer is
unavailable") : require(isDecSendAvailable, "AsterizmInitializer: decode transfer is
unavailable");
        uint nonce = outboundNonce.increaseNonce(_dstChainId,
abi.encodePacked(msg.sender));
        translatorLibrary.send{value: msg.value}(msg.sender, nonce, _dstChainId,
_destination, _transactionId, _isEncoded, _forceOrdered,
clients[msg.sender].shouldCheckFee, _payload);
    }
```

For example, the AstermizmInitializer contract has two clients: one that forwards any message. Another client is a cross-chain token that has a `mint()` function accessible only by the AsterizmIntializer contract. An attacker calling the first client may pass the token address for the `destination` address and execute its `mint()` function passing malicious parameters.

Recommendation

Remove the possibility of having multiple clients for one AsterizmInitializer contract or document the proper usage of the multiple clients explicitly describing all potential risks.

C2-03 Discrepancy in nonce calculation

● Medium ✓ Resolved

In the function `send()` the nonce is calculated for the sender's address value.

```
    function send(uint64 _dstChainId, address _destination, uint _transactionId, bool
_isEncoded, bool _forceOrdered, bytes calldata _payload) external payable onlyClient {
        ...
        uint nonce = outboundNonce.increaseNonce(_dstChainId,
abi.encodePacked(msg.sender));
        ...
    }
    function receivePayload(uint64 _srcChainId, address _application, address _dstAddress,
uint _nonce, uint _gasLimit, uint _transactionId, bool _forceOrdered, bytes calldata
_payload) external onlyTranslator {
        bytes memory _srcPath = abi.encodePacked(_application, _dstAddress);
        if (_forceOrdered) {
            require(inboundNonce.increaseNonceWithValidation(_srcChainId, _nonce,
_srcPath) == _nonce, "AsterizmInitializer: wrong nonce");
        }
        ...
    }
```

But in the `receivePayload()` and `receiveEncodedPayload()` it's calculated for the destination and application address.

Recommendation

Unify nonce calculation in the contract.

C2-04 Different nonce incrementation logic for inbound and outbound messages

● Medium ✓ Resolved

The function `send()` increases outbound nonce every time it is called.

```
    function send(uint64 _dstChainId, address _destination, uint _transactionId, bool
_isEncoded, bool _forceOrdered, bytes calldata _payload) external payable onlyClient {
        ...
        uint nonce = outboundNonce.increaseNonce(_dstChainId,
abi.encodePacked(msg.sender));
        ...
    }
```

But the functions `receivePayload()` and `receiveEncodedPayload()` increase inbound nonce only when the parameter `_isEncoded = true` is passed to the function.

In case some application changes its parameter `forceOrdered` from false to true that it passes to the AsterizmInitializer contract through `send()` function, functions `receivePayload()` and `receiveEncodedPayload()` will revert because the contracts ��`outboundNonce` and `inboundNonce` will have different nonce parameters and the call of the function `increaseNonceWithValidation()` will always revert. To fix this, the application will need to set the parameter `forceOrdered` back to the false value

Recommendation

Unify nonce incrementation logic or if this is the intended behavior handled off-chain document it explicitly.

C2-05 The receive success flag is set even if the client failed to execute the message ● Medium ✓ Resolved

The function `receivePayload()` sets the flag `sendedTransfers[_dto.transferHash].successOutgoing` to true even if the transaction execution fails in the client.

```
    function receivePayload(IzReceivePayloadRequestDto calldata _dto) external
onlyTranslator {
        require(!blockAddresses[_dto.dstAddress], "AsterizmInitializer: target address is
blocked");
        if (_dto.forceOrder) {
            require(
                inboundNonce.increaseNonceWithValidation(_dto.srcChainId,
abi.encodePacked(_dto.srcAddress, _dto.dstAddress), _dto.nonce) == _dto.nonce,
                "AsterizmInitializer: wrong nonce"
            );
        }

        require(_dto.dstAddress != address(this) && _dto.dstAddress != msg.sender,
"AsterizmInitializer: wrong destination address");

        ClAsterizmReceiveRequestDto memory dto = _buildClAsterizmReceiveRequestDto(
            _dto.srcChainId, _dto.srcAddress, _dto.dstChainId,
            _dto.dstAddress, _dto.nonce, _dto.txId, _dto.transferHash, _dto.payload
        );

        sendedTransfers[_dto.transferHash].successIncome = true;
        try IClientReceiverContract(_dto.dstAddress).asterizmIzReceive{gas: _dto.gasLimit}
(dto) {
        } catch Error(string memory _err) {
            emit PayloadErrorEvent(_dto.srcChainId, _dto.srcAddress, _dto.dstChainId,
_dto.dstAddress, _dto.nonce, _dto.transferHash, _dto.payload, abi.encode(_err));
        } catch (bytes memory reason) {
            emit PayloadErrorEvent(_dto.srcChainId, _dto.srcAddress, _dto.dstChainId,
_dto.dstAddress, _dto.nonce, _dto.transferHash, _dto.payload, reason);
        }

        sendedTransfers[_dto.transferHash].successOutgoing = true;
        emit SentPayloadEvent(_dto.transferHash);
    }
```

Recommendation

Update the flag in the try section of the function.

C2-06 Lack of reentrancy checks

● Low ✓ Resolved

The function retryPayload() emits an event after calling the external contract and therefore is susceptible to reentrancy.

```
    function retryPayload(uint64 _srcChainId, address _application, address _dstAddress,
uint _gasLimit, uint _transactionId, bool _isEncoded, bytes calldata _payload) external {
        ...
        if (_isEncoded) {
            IClientContract(dstAddress).asterismReceiveEncoded{gas: _gasLimit}
(_srcChainId, _application, nonce, _transactionId, _payload);
        } else {
            IClientContract(dstAddress).asterismReceive{gas: _gasLimit}(_srcChainId,
_application, nonce, _transactionId, _payload);
        }

        emit PayloadCleared(_srcChainId, _srcPath, nonce, _transactionId, dstAddress);
    }
```

Recommendation

We recommend adding `nonReentrant` modifier from the OpenZepplin's ReentrancyGuard library.

Update

The function `retryPayload()` was deleted in the update, but in the update the function `receivePayload()` does not follow checks effects interactions pattern and has no reentrancy guard protection.

C2-07 Typos

● Info ✓ Resolved

Typos reduce the code's readability. There is a typo in the function name: `validOutgoingTarnsferHash();`

There is typo in a mapping name: `sendedTransfers`.

C3. AsterizmNonce

Overview

A helper contract to store and manipulate nonces for specified keys.

Issues

C3-01 Lack of events

● Low ✓ Resolved

The function `setManipulator()`, `setOwner()`, `forceSetNonce()` don’t emit events, which complicates the tracking of important changes off-chain.

C4. AsterizmTranslator

Overview

Receives messages from the AsterizmInitializer contract and emits an event to be processed by a relayer to be executed on the destination chain. Has privileged functions for the relayer to execute inbound messages.

Issues

C4-01 Dangerous authorisation mechanics ● High ✓ Resolved

Functions `translateMessage()` and `translateEncodedMessage()` are open to everyone if the global variable `isLock` is false. In this case, anyone can pass any message to client's application.

```
    modifier onlyRelayer() {
        if (isLock) {
            require(relayer == msg.sender, "Translator: only relayer");
        }
        _;
    }
```

The owner of the contract can change the variable `isLock` which may impose significant risks for the client's contracts.

Recommendation

Remove the lock functionality or make the `isLock` variable immutable.

C4-02 High centralization ● High ✓ Resolved

All inbound messages to be executed on the destination chain can be executed by a relayer contract.

```
    function translateMessage(uint _gasLimit, bytes calldata _payload) external
onlyRelayer {
        baseTranslateMessage(_gasLimit, _payload, false);
    }

    function translateEncodedMessage(uint _gasLimit, bytes calldata _payload) external
onlyRelayer {
        baseTranslateMessage(_gasLimit, _payload, true);
    }
```

If the private key of the relayer contract is compromised, the attacker can pass any data to clients in the destination chain without any transactions sent in the source chain.

Update

After the update to execute a cross-chain message, the message needs to be approved by a relayer and a client contract.

C4-03 Duplicate nonce calculation ● Medium ✓ Resolved

The function `send()` called by the AsterizmInitializer increases outbound nonce:

```
    function send(address _application, uint _nonce, uint64 _dstChainId, address
_dstAddress, uint _transactionId, bool _isEncoded, bool _forceOrdered, bool
_shouldCheckFee, bytes calldata _payload) external payable onlyEndpoint {
        ...
        uint nonce = outboundNonce.increaseNonce(_dstChainId,
abi.encodePacked(_dstAddress));
        ...
    }
```

Another outbound nonce is already increased in the AsterizmInitializer contract:

```
    function send(uint64 _dstChainId, address _destination, uint _transactionId, bool
_isEncoded, bool _forceOrdered, bytes calldata _payload) external payable onlyClient {
         ...
         uint nonce = outboundNonce.increaseNonce(_dstChainId,
abi.encodePacked(msg.sender));
```

```
            ...
    }
```

With a lack of documentation, it's hard to say whether it's a desired behavior.

Recommendation

Document the usage of the nonces explicitly.

C4-04 Possibility of removing local chain id from the list of chains

● Low ✓ Resolved

In the function `removeChainById()`, there is no check that the `localChainId` global variable isn’t equal to the `_chainId` function argument.

```
    function removeChainById(uint64 _chainId) external onlyOwner {
        delete chainsMap[_chainId];
    }
```

Recommendation

Add a missing check to avoid accidentally removing localChainId of the list of chains.

C4-05 Ability to change local chain id

● Low ✓ Resolved

The function `setLocalChainId()` should be deleted, and the variable `localChainId` should be set in the constructor.

C4-06 Lack of events

● Low ✓ Resolved

The functions `setLocalChainId()`, `removeChainById()`, `addChains()`, `addChain()`, `setIsLock()`, `setEndpoint()` don’t emit events, which complicates the tracking of important changes off-chain.

C5. Claimer

Overview

A helper contract to call multichain token's `crossChainTransfer()` function in batch with different parameters.

Issues

C5-01 Gas optimization ● Low ✓ Partially fixed

1. Global variables `multichainToken` and `multichainTokenAddress` can be immutable.
2. No need for the global variable `multichainTokenAddress`.
3. In the function `claim()` global variable `multichainToken` is read multiple times.

C6. GasSender

Overview

Implementation of the AsterizmClient which sends stable coins to another chain to be converted to a native currency.

Issues

C6-01 Token transfer results are not checked ● Low ✓ Resolved

The boolean value which is returned by the ERC20 token's `transfer()` function is not checked. We recommend using SafeERC20 library to handle all token transfers.

C6-02 Gas optimizations

● Low ✓ Acknowledged

Import of OpenZeppelin's SafeMath library is unnecessary. Solidity version 0.8 is used which includes automatic overflow checks.

C6-03 Discrepancy in outbound and inbound message format

● Low ✓ Acknowledged

Payload format for the inbound and outbound messages is different. The format for outbound message payload is `address[]`, `uint[]`, `address`, `uint8`. The inbound payload format is `address`, `uint`, `address`, `uint`, `uint`

C6-04 Lack of reentrancy checks

● Low ✓ Resolved

The function `asterismReceive()` doesn't implement the check effects interactions pattern and makes a call to an external address which opens the possibility of reentrancy. We recommend adding a mutex with OpenZeppelin's ReentrancyGuard library.

C7. MultichainToken

Overview

ERC20 token. Implementation of AsterizmClient for cross-chain token transfers.

Issues

C7-01 No check for the reuse of cross-chain transfer

● High ✓ Resolved

The encoded cross-chain transfer is done in two steps. At first, a user stores data for transfer in the contract with the `crossChainEncodedTransferFirstStep()` function.

```
    function crossChainEncodedTransferFirstStep(uint64 destChain, address from, address
to, uint amount, address target) public {
        uint transferId = ++transactionId;
        require(!transfers[transferId].exists, "MultichainToken: transaction already
exists");
        transfers[transferId].exists = true;
        transfers[transferId].destChain = destChain;
        transfers[transferId].from = from;
        transfers[transferId].to = to;
        transfers[transferId].amount = amount;
        transfers[transferId].target = target;
        emit CrossChainTransferReceived(transferId, destChain, from, to, amount,
transferId, target);
    }
```

Then the owner of the contract calls the function `crossChainEncodedTransferSecondStep()` to burn tokens from the user and emit an event for the cross-chain transfer. This function sets the `transfers[transferId].exists = false` but never checks if `transfers[transferId].exists` is true before the execution. This allows multiple execution of the same transfer.

Recommendation

Check if `transfers[transferId].exists` is true before the execution of the function.

C7-02 Open mint for the owner

● High ✓ Resolved

The owner of the contract has a possibility to mint any number of tokens. The function asterismReceive() which effectively mints tokens to a specified address can be called by the owner or initializer contracts.

```
    modifier onlyVerifiedSender {
        require(msg.sender == owner || msg.sender == initializer, "MultichainToken: only
verified sender");
        _;
    }

    function asterismReceive(uint64 _srcChainId, address _srcAddress, uint _nonce, uint
_transactionId, bytes calldata _payload) public onlyVerifiedSender {
        (address dstAddress, uint amount) = abi.decode(_payload, (address, uint));
        _creditTo(dstAddress, amount);
    }

    function _creditTo(address _toAddress, uint _amount) internal virtual returns(uint) {
        _mint(_toAddress, _amount);
        return _amount;
    }
```

C7-03 isEncoded parameter is always false ● Medium ✓ Resolved

The contract has functions to send encoded messages, but the parameter `isEncoded` is never set to true.

Recommendation

Update the code to send the encoded messages with the `isEncoded` parameter set to true.

C7-04 No revert message ● Low ✓ Resolved

In the function `setOwner()`, there is no revert message in the require statement

C7-05 Gas optimization ● Low ✓ Resolved

1. Global variables `initializerLib` and `initializer` can be immutable
2. The contract doesn't use the global variable `transfersCounter` anywhere
3. The global variable `initializer` can be deleted and replaced with the `initializerLib` variable
4. In the function `crossChainEncodedTransferSecondStep()`, the contract can delete the structure `transfers[id]` at the end

C7-06 Lack of events

● Low ✓ Partially fixed

The functions `setPool()` , `setOwner()`, `crossChainTransfer()`, `asterismReceive()` don't emit events, which complicates the tracking of important changes off-chain.

C7-07 Unused parameters

● Info ✓ Resolved

The contract doesn't use the information a user sent to the `crossChainEncodedTransferFirstStep()` function (`to` and `amount`). Instead, it uses the data that the owner passes.

5. Conclusion

8 high, 9 medium, 19 low severity issues were found during the audit. 7 high, 8 medium, 14 low issues were resolved in the update.

The audited contracts are highly dependent on privileged accounts. Users using the project have to trust that the privileged accounts are properly secured.

We would like to emphasize the importance of thorough auditing of any contracts that inherit from the BaseAsterizmClient. While the library itself has been designed with security in mind, it is still possible to introduce vulnerabilities or misconfigurations when implementing inherited contracts. In particular, when setting up parameters in the inherited contracts, it is crucial to ensure that they are configured correctly and securely. Failing to do so could potentially expose the contract to various attack vectors and vulnerabilities.

Appendix A. Issues' severity classification

- **Critical.** Issues that may cause an unlimited loss of funds or entirely break the contract workflow. Malicious code (including malicious modification of libraries) is also treated as a critical severity issue. These issues must be fixed before deployments or fixed in already running projects as soon as possible.
- **High.** Issues that may lead to a limited loss of funds, break interaction with users, or other contracts under specific conditions. Also, issues in a smart contract, that allow a privileged account the ability to steal or block other users' funds.
- **Medium.** Issues that do not lead to a loss of funds directly, but break the contract logic. May lead to failures in contracts operation.
- **Low.** Issues that are of a non-optimal code character, for instance, gas optimization tips, unused variables, errors in messages.
- **Informational.** Issues that do not impact the contract operation. Usually, informational severity issues are related to code best practices, e.g. style guide.

Appendix B. List of examined issue types

- Business logic overview
- Functionality checks
- Following best practices
- Access control and authorization
- Reentrancy attacks
- Front-run attacks
- DoS with (unexpected) revert
- DoS with block gas limit
- Transaction-ordering dependence
- ERC/BEP and other standards violation
- Unchecked math
- Implicit visibility levels
- Excessive gas usage
- Timestamp dependence
- Forcibly sending ether to a contract
- Weak sources of randomness
- Shadowing state variables
- Usage of deprecated code

✉ contact@hashex.org

@hashex_manager

blog.hashex.org

linkedin

github

twitter

HashEx

BLOCKCHAIN SECURITY