decurity

Smart Contract Security Audit Report

Asterizm EVM Audit

1. Contents

2. General Information

This report contains information about the results of the security audit of the Asterizm (hereafter referred to as "Customer") smart contracts, conducted by Decurity in the period from 25/10/2024 to 11/11/2024.

2.1. Introduction

Tasks solved during the work are:

- Review the protocol design and the usage of 3rd party dependencies,
- Audit the contracts implementation,
- Develop the recommendations and suggestions to improve the security of the contracts.

2.2. Scope of Work

The audit scope included the contracts in the following repository: https://github.com/Asterizm-Protocol/asterizm-contracts-evm. Initial review was done for the commit 4408ff, with re-testing performed for on commit da839d.

The following contracts have been tested:

- contracts/AsterizmTranslatorV1.sol
- contracts/AsterizmTranslatorChainlink.sol
- contracts/libs/AsterizmHashLib.sol
- contracts/AsterizmInitializerV1.sol
- contracts/base/AsterizmSender.sol
- contracts/base/AsterizmEnv.sol
- contracts/base/AsterizmClient.sol
- contracts/base/AsterizmChainEnv.sol
- contracts/base/AsterizmClientUpgradeable.sol
- contracts/base/AsterizmSenderUpgradeable.sol

- contracts/base/AsterizmWithdrawalUpgradeable.sol
- contracts/base/AsterizmConfig.sol
- contracts/base/AsterizmRefund.sol
- contracts/base/AsterizmWithdrawal.sol
- contracts/base/AsterizmRefundUpgradeable.sol

2.3. Threat Model

The assessment presumes actions of an intruder who might have capabilities of any role (an external user, token owner, token service owner, a contract).

The main possible threat actors are:

- User,
- Protocol owner.

The table below contains sample attacks that malicious attackers might carry out.

Table. Theoretically possible attacks

Attack	Actor
Contract code or data hijacking *Deploying a malicious contract or submitting malicious data*	Contract owner Token owner
Financial fraud *A malicious manipulation of the business logic and balances, such as a re-entrancy attack or a flash loan attack*	Anyone
Attacks on implementation *Exploiting the weaknesses in the compiler or the runtime of the smart contracts*	Anyone

2.4. Weakness Scoring

An expert evaluation scores the findings in this report, an impact of each vulnerability is calculated based on its ease of exploitation (based on the industry practice and our experience) and severity (for the considered threats).

2.5. Disclaimer

Due to the intrinsic nature of the software and vulnerabilities and the changing threat landscape, it cannot be generally guaranteed that a certain security property of a program holds.

Therefore, this report is provided "as is" and is not a guarantee that the analyzed system does not contain any other security weaknesses or vulnerabilities. Furthermore, this report is not an endorsement of the Customer's project, nor is it an investment advice.

That being said, Decurity exercises best effort to perform their contractual obligations and follow the industry methodologies to discover as many weaknesses as possible and maximize the audit coverage using the limited resources.

3. Summary

As a result of this work, we have discovered a single high exploitable security issue and medium level issues. The other suggestions included fixing the low-risk issues and some best practices (see Security Process Improvement). These vulnerabilities have been addressed and thoroughly re-tested as part of our process.

3.1. Suggestions

The table below contains the discovered issues, their risk level, and their status as of December 13, 2024.

Table. Discovered weaknesses

Issue	Contract	Risk Level	Status
Chainlink CCIP message sender is not checked	contracts/AsterizmTranslatorChainlink.sol	High	Fixed
Refund request could be processed after successful execution of the transaction	contracts/base/AsterizmClient.sol	Medium	Acknowledged
Rejected refund requests blocks transaction	contracts/base/AsterizmRefund.sol contracts/base/AsterizmRefundUpgradeable.sol	Medium	Fixed
Relays centralizations risks	contracts/base/AsterizmClient.sol	Medium	Fixed
SafeERC20 should be used	contracts/base/AsterizmClient.sol contracts/base/AsterizmClientUpgradeable.sol contracts/AsterizmInitializerV1.sol contracts/AsterizmTranslatorChainlink.sol	Medium	Fixed
No storage gaps for upgradeable contracts	contracts/base/AsterizmChainEnv.sol contracts/base/AsterizmClientUpgradeable.sol contracts/base/AsterizmConfig.sol contracts/base/AsterizmRefundUpgradeable.sol contracts/base/AsterizmSenderUpgradeable.sol contracts/base/AsterizmWithdrawalUpgradeable.sol	Medium	Fixed
Outdated `OpenZeppelin` library		Low	Fixed
Missing disable initializer	contracts/AsterizmTranslatorV1.sol contracts/AsterizmInitializerV1.sol	Low	Fixed

Issue	Contract	Risk Level	Status
Ownable2Step should be used	contracts/AsterizmTranslatorV1.sol contracts/base/AsterizmConfig.sol contracts/base/AsterizmSender.sol contracts/base/AsterizmSenderUpgradeable.sol contracts/base/AsterizmWithdrawal.sol contracts/base/AsterizmWithdrawalUpgradeable.sol	Low	Acknowledged
`resendMessage` is callable in AsterizmTranslatorChainlink	contracts/AsterizmTranslatorChainlink.sol	Low	Fixed
Upgradeable version of Refund contract works differently	contracts/base/AsterizmRefundUpgradeable.sol	Low	Fixed
Follow the check effects interaction pattern	contracts/base/AsterizmClient.sol contracts/base/AsterizmClientUpgradeable.sol	Info	Fixed
Chainlink extraArgs is immutable	contracts/AsterizmTranslatorChainlink.sol	Info	Acknowledged
Unused imports	contracts/AsterizmTranslatorChainlink.sol contracts/AsterizmTranslatorV1.sol	Info	Fixed
Redundant functionality	contracts/AsterizmTranslatorV1.sol contracts/AsterizmTranslatorChainlink.sol	Info	Fixed

4. General Recommendations

This section contains general recommendations on how to improve overall security level.

The Findings section contains technical recommendations for each discovered issue.

4.1. Security Process Improvement

The following is a brief long-term action plan to mitigate further weaknesses and bring the product security to a higher level:

- Keep the whitepaper and documentation updated to make it consistent with the implementation and the intended use cases of the system,
- Perform regular audits for all the new contracts and updates,
- Ensure the secure off-chain storage and processing of the credentials (e.g. the privileged private keys),
- Launch a public bug bounty campaign for the contracts.

5. Findings

5.1. Chainlink CCIP message sender is not checked

Risk Level: **High**

Status: Fixed in the commit [a7dd95](a7dd95).

Contracts:

- contracts/AsterizmTranslatorChainlink.sol

Description:

Current implementation of the `_ccipReceive()` function in the `AsterizmTranslatorChainlink`:

```
//@audit snippet of code from `CCIPReceiver` contract
function ccipReceive(Client.Any2EVMMessage calldata message) external virtual
override onlyRouter {
    _ccipReceive(message);
  }

//@audit snippet of code from `AsterizmTranslatorChainlink ` contract

function _ccipReceive(Client.Any2EVMMessage memory _dto) internal override {
    _baseTransferMessage(_buildTrTransferMessageRequestDto(gasleft(),
_dto.data));
}

/// Base transfer message
/// @param _dto TrTransferMessageRequestDto  Method DTO
function _baseTransferMessage(TrTransferMessageRequestDto memory _dto) private
{
    (
        uint64 srcChainId, uint srcAddress, uint64 dstChainId,
        uint dstAddress, uint txId, , bytes32 transferHash
    ) = abi.decode(
        _dto.payload,
        (uint64, uint, uint64, uint, uint, bool, bytes32)
    );

    {
        require(dstChainId == localChainId, "TranslatorChainlink: wrong chain
id");
        require(dstAddress.toAddress().isContract(), "TranslatorChainlink:
```

```
destination address is non-contract");

        initializerLib.receivePayload(_buildIzReceivePayloadRequestDto(
            _buildBaseTransferDirectionDto(srcChainId, srcAddress,
localChainId, dstAddress),
            _dto.gasLimit, txId, transferHash
        ));
    }

    emit TransferSendEvent(srcChainId, srcAddress, dstAddress, transferHash);
}
```

The external `ccipReceive()` function includes an `onlyRouter` modifier that verifies the function was called by the Chainlink router.

However, the `message` received from the Chainlink router can be constructed by any sender. The sender address is stored in the `Any2EVMMessage` struct:

```
  struct Any2EVMMessage {
    bytes32 messageId; // MessageId corresponding to ccipSend on source.
    uint64 sourceChainSelector; // Source chain selector.
    bytes sender; // abi.decode(sender) if coming from an EVM chain.
    bytes data; // payload sent in original message.
    EVMTokenAmount[] destTokenAmounts; // Tokens and their amounts in their
destination chain representation.
  }
```

The current implementation does not verify the actual sender of the message. As demonstrated in Chainlink's best practice examples, verifying the sender on the source chain is crucial:

```
 /// handle a received message
  function _ccipReceive(
      Client.Any2EVMMessage memory any2EvmMessage
  )
      internal
      override
      onlyAllowlisted(
          any2EvmMessage.sourceChainSelector,
          abi.decode(any2EvmMessage.sender, (address))
      ) // Make sure source chain and sender are allowlisted
  { ...
  }
```

Lack of this check allow to successfully call the `_asterizmReceiveExternal()` functions on clients contract and bypass the checks that should be performed on the Asterizm relayer to verify the that sender and chain are trusted.

Remediation:

Consider implementing the check that initial sender of a message on the source chain is trusted and equals to the `AsterizmTranslatorChainlink` address on another chain.

```
onlyAllowlisted(
    any2EvmMessage.sourceChainSelector,
    abi.decode(any2EvmMessage.sender, (address))
)
```

References:

- https://docs.chain.link/ccip/tutorials/send-arbitrary-data

5.2. Refund request could be processed after successful execution of the transaction

Risk Level: Medium

Status: Acknowledged

Contracts:

- contracts/base/AsterizmClient.sol

Description:

The Client contract includes a method, `transferSendingResultNotification`, that allows a translator to notify the client when a transfer request has been successfully processed:

```
    function transferSendingResultNotification(bytes32 _transferHash, uint8
_statusCode) external onlyInitializer
onlyExecutedOutboundTransfer(_transferHash) {
        if (notifyTransferSendingResult) {
            emit TransferSendingResultNotification(_transferHash,
_statusCode);
        }
    }
```

This notification can be used to improve the system's robustness. For example, the contract could use this information to prevent refund requests if a successful status code is received.

Remediation:

To address this issue: 1. Refactor the contract to store the status code from the `transferSendingResultNotification` method. 2. Prevent refunds from being processed if a successful status code is recorded for the corresponding `_transferHash`.

5.3. Rejected refund requests blocks transaction

Risk Level: Medium

Status: Fixed in the commit 316958.

Contracts:

- contracts/base/AsterizmRefund.sol
- contracts/base/AsterizmRefundUpgradeable.sol

Description:

If a refund request is rejected, the user's transaction will not be processed in `_initAsterizmTransferPrivate`, and the funds will not be returned. This is because `onlyNotRefundedTransferOnSrcChain` modifier checks the existence of the refund, not its successful execution.

Remediation:

Consider allowing the transaction to be processed in this case. Modify the `onlyNotRefundedTransferOnSrcChain` modifier to:

```
    modifier onlyNotRefundedTransferOnSrcChain(bytes32 _transferHash) {
        require(!refundRequests[_transferHash].successProcessed, "AR: transfer
was refunded");
        _;
    }
```

5.4. Relays centralizations risks

Risk Level: Medium

Status: We will make logic for limiting the amount of commission sent for each network, after which the client off-chain module will not send a transfer if the commission amount exceeds this threshold.

Contracts:

- contracts/base/AsterizmClient.sol

Description:

If the relay is compromised, funds may be stolen from clients' contracts.

The client queries the `externalRelay` for the amount of tokens that should be as a fee:

```
contracts/base/AsterizmClient.sol:
        if (address(feeToken) != address(0)) { // Token fee logic
            uint feeAmountInToken =
initializerLib.getFeeAmountInTokens(externalRelay, initDto);
            if (feeAmountInToken > 0) {
                require(feeToken.balanceOf(address(this)) >=
feeAmountInToken, "AsterizmClient: fee token balance is not enough");
                feeToken.approve(address(initializerLib),
feeAmountInToken);
            }
        }
```

These tokens are then approved for `initializerLib`, transferred from the client, and approved for the relay address:

```
contracts/AsterizmInitializerV1.sol:
        if (_dto.feeToken != address(0)) { // Token fee logic
            IERC20 feeToken = IERC20(_dto.feeToken);
            uint feeTokenAmount = feeToken.allowance(msg.sender,
address(this));
            if (feeTokenAmount > 0) {
                feeToken.transferFrom(msg.sender, address(this),
feeTokenAmount);
                feeToken.approve(relayAddress, feeTokenAmount);
            }
        }
```

In the case of server malfunction, funds may be stolen from clients' contracts.

Additionally, a compromised relay can steal fees from an automated client's server by sending false status notifications (e.g., if the server tries to resend messages).

Remediation:

Consider refactoring the logic to mitigate described risks.

5.5. SafeERC20 should be used

Risk Level: Medium

Status: Fixed in the commit 316958.

Contracts:

- contracts/base/AsterizmClient.sol
- contracts/base/AsterizmClientUpgradeable.sol
- contracts/AsterizmInitializerV1.sol
- contracts/AsterizmTranslatorChainlink.sol

Description:

There are IERC20 functions that are called directly in the protocol:

```
contracts/base/AsterizmClient.sol:
  397:                    feeToken.approve(address(initializerLib),
feeAmountInToken);

contracts/base/AsterizmClientUpgradeable.sol:
  406:                    feeToken.approve(address(initializerLib),
feeAmountInToken);


contracts/AsterizmInitializerV1.sol:
  223:                    feeToken.transferFrom(msg.sender, address(this),
feeTokenAmount);
  224:                    feeToken.approve(relayAddress, feeTokenAmount);

contracts/AsterizmTranslatorChainlink.sol:
  363:                feeToken.transferFrom(address(initializerLib),
address(this), chainlinkFee);
  364:                feeToken.approve(address(baseRouter), chainlinkFee);
```

Since the IERC20 interface requires a boolean return value, attempting to `approve()` , and `transferFrom()` ERC20s with missing return values will revert. This means Asterizm may won't support several popular ERC20s as a feeToken, including USDT and BNB tokens on the Ethereum chain.

Remediation:

Consider using a safe version of the mentioned functions from SafeERC20 library such as `safeTransferFrom()` and `forceApprove()` that handles a case when functions may not return values.

References:

- https://github.com/d-xo/weird-erc20#missing-return-values

5.6. No storage gaps for upgradeable contracts

Risk Level: Medium

Status: Fixed in the commit 316958.

Contracts:

- contracts/base/AsterizmChainEnv.sol
- contracts/base/AsterizmClientUpgradeable.sol
- contracts/base/AsterizmConfig.sol
- contracts/base/AsterizmRefundUpgradeable.sol
- contracts/base/AsterizmSenderUpgradeable.sol
- contracts/base/AsterizmWithdrawalUpgradeable.sol

Description:

There are several contracts that are inherited in an upgradeable contracts and have storage variables:

- contracts/base/AsterizmChainEnv.sol
- contracts/base/AsterizmClientUpgradeable.sol
- contracts/base/AsterizmConfig.sol
- contracts/base/AsterizmRefundUpgradeable.sol
- contracts/base/AsterizmSenderUpgradeable.sol
- contracts/base/AsterizmWithdrawalUpgradeable.sol

Without gaps, adding new storage variables to any of these contracts can potentially overwrite the beginning of the storage layout of the child contract, causing critical misbehaviours in the system.

Remediation:

Consider using gaps at the end of the described contracts to prevent the possibility of storage overriding.

5.7. Outdated `OpenZeppelin` library

Risk Level: Low

Status: Fixed in the commit ae6d2a.

Description:

There is an outdated `OpenZeppelin` library in the dependencies (version 4.8.2). This library contains several known vulnerabilities that may have a security impact when new features are added to the project.

Remediation:

Consider updating the dependencies on time to avoid the risk of known vulnerabilities.

References:

- https://github.com/OpenZeppelin/openzeppelin-contracts/security

5.8. Missing disable initializer

Risk Level: Low

Status: Fixed in the commit da839d.

Contracts:

- contracts/AsterizmTranslatorV1.sol
- contracts/AsterizmInitializerV1.sol

Description:

The upgradeable contracts do not disable initializers for the implementation contracts as recommended by OpenZeppelin. This could potentially lead to the initialization of the implementation contract, which poses a security risk if misused.

Remediation:

Consider disabling the initializers in the implementation contracts to prevent them from being directly used or initialized. This can be done using OpenZeppelin's recommended approach:

```
// Example to disable initializers
constructor() {
    _disableInitializers();
}
```

References:

- https://docs.openzeppelin.com/upgrades-plugins/1.x/writing-upgradeable#initializing_the_implementation_contract

5.9. Ownable2Step should be used

Risk Level: Low

Status: Acknowledged

Contracts:

- contracts/AsterizmTranslatorV1.sol
- contracts/base/AsterizmConfig.sol
- contracts/base/AsterizmSender.sol
- contracts/base/AsterizmSenderUpgradeable.sol
- contracts/base/AsterizmWithdrawal.sol
- contracts/base/AsterizmWithdrawalUpgradeable.sol

Description:

The contracts is inherited from Ownable contract. The owner may accidentally specify a non-active address and lose access. `Ownable2Step.sol` is more secure due to a 2-stage ownership transfer.

```
contract TrufMigrator {
        ...

    constructor(address _signer) {
        signer = _signer;
        deployer = msg.sender;
    }

    function initialize(IERC20 _trufToken, TrufVesting _trufVesting,
VotingEscrowTruf _veTRUF) external {
        if (msg.sender != deployer) revert();
        ...
```

Remediation:

We recommend using the `Ownable2Step` contract from Open Zeppelin (Ownable2Step.sol) instead.

5.10. `resendMessage` is callable in AsterizmTranslatorChainlink

Risk Level: Low

Status: Fixed in the commit 316958.

Contracts:

- contracts/AsterizmTranslatorChainlink.sol

Description:

The AsterizmTranslatorChainlink has a function `resendMessage()` that is designed to emit a resend event and send some native tokens to a relayer. However, if a user attempts to resend a transaction using ChainlinkCCIP, the user will end up spending native tokens without any actual effect, resulting in a waste of funds.

Remediation:

Consider modifying the `resendMessage()` function to include a revert condition to prevent users from accidentally spending native tokens without effect.

5.11. Upgradeable version of Refund contract works differently

Risk Level: Low

Status: Fixed in the commit 316958.

Contracts:

- contracts/base/AsterizmRefundUpgradeable.sol

Description:

The `AsterizmRefund` contract includes a check in the `addRefundRequest()` function to insure that the caller of the function is the predefined `userAddress`:

```
require(msg.sender == refundTransfers[_transferHash].userAddress, "AR: wrong
sender address");
```

However, the `AsterizmRefundUpgradeable` contract, which is an upgradeable version of `AsterizmRefund` contract, lacks this check.

Remediation:

Consider modifying the logic of the `AsterizmRefundUpgradeable` contract to include this caller verification check so that both contracts function identically and provide consistent security measures.

5.12. Follow the check effects interaction pattern

Risk Level: Info

Status: Fixed in the commit 316958.

Contracts:

- contracts/base/AsterizmClient.sol
- contracts/base/AsterizmClientUpgradeable.sol

Description:

According to the check-effects-interaction pattern, it is better to change the state before making external calls to avoid potential reentrancy issues. In the `_asterizmReceiveInternal()` function, the successExecute state is updated only after the external call to `_asterizmReceive(_dto)`. This could lead to vulnerabilities if an unexpected reentrant call occurs during the external interaction.

```
function _asterizmReceiveInternal(ClAsterizmReceiveRequestDto memory _dto)
private
        onlyReceivedTransfer(_dto.transferHash)
        onlyTrustedAddress(_dto.srcChainId, _dto.srcAddress)
        onlyTrustedTransfer(_dto.transferHash)
        onlyNonExecuted(_dto.transferHash)
        onlyValidTransferHash(_dto)
        onlyNotRefundedTransferOnDstChain(_dto.transferHash)
    {
        _asterizmReceive(_dto);
        inboundTransfers[_dto.transferHash].successExecute = true; //@audit-
issue better to put this before fully executing message
    }
```

Remediation:

Consider updating the `successExecute` state variable before the external call to `_asterizmReceive(_dto)` to follow the check-effects-interaction pattern, minimizing the risk of reentrancy vulnerabilities:

```
inboundTransfers[_dto.transferHash].successExecute = true;
_asterizmReceive(_dto);
```

5.13. Chainlink extraArgs is immutable

Risk Level: Info

Status: Acknowledged

Contracts:

- contracts/AsterizmTranslatorChainlink.sol

Description:

The `extraArgs` parameter, as described in the Chainlink documentation, is designed to enable compatibility with future CCIP upgrades. To fully leverage this benefit, `extraArgs` should be mutable in production deployments. If `extraArgs` is immutable, the contract may not be adaptable to potential future changes in the Chainlink CCIP.

Remediation:

Consider making the `extraArgs` mutable.

References:

- https://docs.chain.link/ccip/best-practices#using-extraargs

5.14. Unused imports

Risk Level: Info

Status: Fixed in the commit 316958.

Contracts:

- contracts/AsterizmTranslatorChainlink.sol
- contracts/AsterizmTranslatorV1.sol

Description:

The SafeMath import is never used.

Remediation:

Consider removing the unused imports.

5.15. Redundant functionality

Risk Level: Info

Status: Fixed in the commit 316958.

Contracts:

- contracts/AsterizmTranslatorV1.sol
- contracts/AsterizmTranslatorChainlink.sol

Description:

The AsterizmTranslatorV1 and AsterizmTranslatorChainlink contracts each have a `withdraw()` function that is redundant because both contracts already have `withdrawCoins()` functions that implement the same functionality.

```
    function withdraw(address _target, uint _amount) external onlyOwner {
        require(address(this).balance >= _amount, "Translator: coins balance
not enough");
        (bool success, ) = _target.call{value: _amount}("");
        require(success, "Translator: transfer error");
        emit WithdrawEvent(_target, _amount);
    }
```

Remediation:

Consider removing the redundant function.

6. Appendix

6.1. About us

The Decurity team consists of experienced hackers who have been doing application security assessments and penetration testing for over a decade.

During the recent years, we've gained expertise in the blockchain field and have conducted numerous audits for both centralized and decentralized projects: exchanges, protocols, and blockchain nodes.

Our efforts have helped to protect hundreds of millions of dollars and make web3 a safer place.