Security Audit Report for Argus

Date: June 12, 2023

Version: 1.0

Contact: contact@blocksec.com

BlockSec
Audit Team

Digitally signed by
BlockSec Audit Team
Date: 2023.06.15
14:04:02 +08'00'

Contents

1 Introduction 1

1.1 About Target Contracts 1

1.2 Disclaimer 2

1.3 Procedure of Auditing 3

1.3.1 Software Security 3

1.3.2 DeFi Security 3

1.3.3 NFT Security 4

1.3.4 Additional Recommendation 4

1.4 Security Model 4

2 Findings 6

2.1 Software Security 6

2.1.1 Potential Denial of Service in LendingBaseACL 6

2.1.2 Potential Incorrect Event Emitted 7

2.2 DeFi Security 8

2.2.1 Lack of Slippage Control when Swapping 8

2.3 Additional Recommendation 9

2.3.1 Event Generation with Indexing 9

2.3.2 Redundant Check of Roles and Authorizers 9

2.3.3 Removal of Unused Token in tokenSet 11

2.3.4 Lack of Length Check between Parameters 12

2.3.5 Improper Implementation of Updating Whitelist 12

2.4 Notes 13

2.4.1 Validation of Hints before Usage 14

2.4.2 Ignored Reverts in _preExecCheck() and _postExecCheck() 14

2.4.3 Deployments of Contracts within Argus Protocol 14

Report Manifest

Item	Description
Client	Cobo Global
Target	Argus

Version History

Version	Date	Description
1.0	June 12, 2023	First Version

About BlockSec The BlockSec Team focuses on the security of the blockchain ecosystem, and collaborates with leading DeFi projects to secure their products. The team is founded by top-notch security researchers and experienced experts from both academia and industry. They have published multiple blockchain security papers in prestigious conferences, reported several zero-day attacks of DeFi applications, and released detailed analysis reports of high-impact security incidents. They can be reached at Email, Twitter and Medium.

Chapter 1 Introduction

1.1 About Target Contracts

Information	Description
Type	Smart Contract
Language	Solidity
Approach	Semi-automatic and manual verification

The repository that has been audited includes argus-8f820969.zip.

The auditing process is iterative. Specifically, we will audit the commits that fix the discovered issues. If there are new issues, we will continue this process. The MD5 values of the files before and after the audit are shown in the following. Our audit report is responsible for the only initial version (i.e., `Version 1`), as well as new codes (in the following versions) to fix issues in the audit report.

Version 1

File	md5
1inchV5Authorizer.sol	c60db59d1c7ffd369cbb55032eca8365
IStargateFactory.sol	6af97488380b5f013fec9928fe086022
stargateClaimAuthorizer.sol	358af007ddf86e37e7c984b4682d5103
stargateDepositAuthorizer.sol	127d4e5adb9a2af759e43e43706ed8e6
stargateWithdrawAuthorizer.sol	322644e376e1151cfa72419b97a09cf9
ArgusRootAuthorizer.sol	f4261a56f67bd3882e5069eaf516bee0
DEXBaseACL.sol	6359bd27344bbee191326d997bddbaa0
FarmingBase.sol	31b699ae020c1518e93a62e3a0fa527e
FuncAuthorizer.sol	471e704942f3ac310d8cdd8be900cf3c
LendingBaseACL.sol	d85fa43534262e3fa1fdaf6727167683
TransferAuthorizer.sol	38bfcabb0a0130afcc0d742566649df2
BaseAccount.sol	a3d88d2cc2dbee51a3985a67975921ed
BaseACL.sol	45b11af752c20907f72bdeb41e8880e2
BaseAuthorizer.sol	2285e38c37dee842d90e17da694725e8
BaseOwnable.sol	5f8e87db5069aecd25fcfd35737de6d7
BaseVersion.sol	0d5b6d21abb283ce8ba31ad480dbf77f
PostBaseACL.sol	1c7d99c2564f1311ffb7e128d0fe7aa1
ArgusAccountHelper.sol	d97ed6e61aabfb2ba60de4e459cc8f4c
ArgusAuthorizerHelper.sol	f05b18e0619df8a7677f238d7ae49de7
ArgusViewHelper.sol	36169b91c674df15e2d484708df1673f
FlatRoleManager.sol	cb544f658e9806d6fd41638d9ade81a0
CoboFactory.sol	c7b17a1f31fa0852619b1615f108c407
CoboSafeAccount.sol	b76563db8307839322fa2d18d9ce62a7
CoboSmartAccount.sol	7b71e1a0fe91213197c7b5145bf93062
Errors.sol	320b3f2089e99b47a24d87f4d0392ec2

Types.sol	6bd66a643f7a760fad0c185d7ca69f84

Version 2

File	md5
OneinchV5Authorizer.sol	725d65559bb501e72fd9f946c731c60b
IStargateFactory.sol	6af97488380b5f013fec9928fe086022
stargateClaimAuthorizer.sol	457d5bf017daa499ddb8d186aad6b513
stargateDepositAuthorizer.sol	36de0b049290b7e2797331d95d2c77c3
stargateWithdrawAuthorizer.sol	451002c2d11e545d727013f524a6649f
ArgusRootAuthorizer.sol	259447213e1f28536fe204e78156570a
DEXBaseACL.sol	2228fa86fb57fa5957252c8628ffcbb3
FarmingBase.sol	7f6304be1610633d5bd63d8f61452678
FuncAuthorizer.sol	43198ce3a32a5b9a2fd188016b44f96a
LendingBaseACL.sol	6c656dae42a71a58936295f65bfc7e41
TransferAuthorizer.sol	2f8fc8e46324947dbd54bde590e5fcac
BaseAccount.sol	6f185808b9280dc1efded773d350e361
BaseACL.sol	fd4e05f95f5def73d7b565adc9ff251a
BaseAuthorizer.sol	e5fdd58bb56c2ff3782cfe53c3345d0c
BaseOwnable.sol	4769a034762dc4d312d8998eabd4ef13
BaseVersion.sol	0d5b6d21abb283ce8ba31ad480dbf77f
PostBaseACL.sol	2fc23e514b4991519810ad94a95fbff2
ArgusAccountHelper.sol	0cede9a30e43b3947f60c0de311f8d22
ArgusAuthorizerHelper.sol	a397e41a536404e6207d47d48ff72c61
ArgusViewHelper.sol	fdfe4c6414f591d53bf1b2658f9cdc79
FlatRoleManager.sol	cb544f658e9806d6fd41638d9ade81a0
CoboFactory.sol	bda5bf37d2ed79f725a9a79d78556a1f
CoboSafeAccount.sol	a1f146b1693d42bf6132a3a4284fc181
CoboSmartAccount.sol	4e46cad581289f181d98611733bb33fc
Errors.sol	1f87f1eaa607f708c8bdf86c41026676
Types.sol	c78a062acd9eb711d901ffa87f670874

1.2 Disclaimer

This audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset.

This audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contracts.

The scope of this audit is limited to the code mentioned in Section 1.1. Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.

1.3 Procedure of Auditing

We perform the audit according to the following procedure.

- **Vulnerability Detection** We first scan smart contracts with automatic code analyzers, and then manually verify (reject or confirm) the issues reported by them.
- **Semantic Analysis** We study the business logic of smart contracts and conduct further investigation on the possible vulnerabilities using an automatic fuzzing tool (developed by our research team). We also manually analyze possible attack scenarios with independent auditors to cross-check the result.
- **Recommendation** We provide some useful advice to developers from the perspective of good programming practice, including gas optimization, code style, and etc.

We show the main concrete checkpoints in the following.

1.3.1 Software Security

- Reentrancy
- DoS
- Access control
- Data handling and data flow
- Exception handling
- Untrusted external call and control flow
- Initialization consistency
- Events operation
- Error-prone randomness
- Improper use of the proxy system

1.3.2 DeFi Security

- Semantic consistency
- Functionality consistency
- Access control
- Business logic
- Token operation
- Emergency mechanism
- Oracle security
- Whitelist and blacklist

- ∗ Economic impact
- ∗ Batch transfer

1.3.3 NFT Security

- ∗ Duplicated item
- ∗ Verification of the token receiver
- ∗ Off-chain metadata security

1.3.4 Additional Recommendation

- ∗ Gas optimization
- ∗ Code quality and style

Note *The previous checkpoints are the main ones. We may use more checkpoints during the auditing process according to the functionality of the project.*

1.4 Security Model

To evaluate the risk, we follow the standards or suggestions that are widely adopted by both industry and academy, including OWASP Risk Rating Methodology [1] and Common Weakness Enumeration [2]. The overall *severity* of the risk is determined by *likelihood* and *impact*. Specifically, likelihood is used to estimate how likely a particular vulnerability can be uncovered and exploited by an attacker, while impact is used to measure the consequences of a successful exploit.

In this report, both likelihood and impact are categorized into two ratings, i.e., *high* and *low* respectively, and their combinations are shown in Table 1.3.

Table 1.3: Vulnerability Severity Classification

Impact	*High*	High	Medium
	Low	Medium	Low
		High	*Low*
		Likelihood	

Accordingly, the severity measured in this report are classified into three categories: **High**, **Medium**, **Low**. For the sake of completeness, **Undetermined** is also used to cover circumstances when the risk cannot be well determined.

Furthermore, the status of a discovered item will fall into one of the following four categories:

- **Undetermined** No response yet.

[1] https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

[2] https://cwe.mitre.org/

- **Acknowledged** The item has been received by the client, but not confirmed yet.
- **Confirmed** The item has been recognized by the client, but not fixed yet.
- **Fixed** The item has been confirmed and fixed by the client.

Chapter 2 Findings

In total, we find **three** potential issues. Besides, we have **five** recommendations and **three** notes as follows:

- High Risk: 0
- Medium Risk: 1
- Low Risk: 2
- Recommendations: 5
- Notes: 3

ID	Severity	Description	Category	Status
1	Low	Potential Denial of Service in LendingBaseACL	Software Security	Confirmed
2	Low	Potential Incorrect Event Emitted	Software Security	Fixed
3	Medium	Lack of Slippage Control when Swapping	DeFi Security	Confirmed
4	-	Event Generation with Indexing	Recommendation	Fixed
5	-	Redundant Check of Roles and Authorizers	Recommendation	Confirmed
6	-	Removal of Unused Token in tokenSet	Recommendation	Fixed
7	-	Lack of Length Check between Parameters	Recommendation	Fixed
8	-	Improper Implementation of Updating Whitelist	Recommendation	Fixed
9	-	Validation of Hints before Usage	Note	Confirmed
10	-	Ignored Reverts in _preExecCheck() and _postExecCheck()	Note	Confirmed
11	-	Deployments of Contracts within Argus Protocol	Note	Confirmed

The details are provided in the following sections.

2.1 Software Security

2.1.1 Potential Denial of Service in LendingBaseACL

Severity Low

Status Confirmed

Introduced by `Version 1`

Description The contract `LendingBaseACL` is a base `authorizer` designed for checking the state of the smart contract wallet when interacting with lending protocols. The current implementation checks the update of the health factor after the lending transaction is executed. If the updated health factor is less than the minHealthFactor, then reverts.

However, the `minHealthFactor` has to be manually set by the privileged owner via the function `setMinHealthFactor()`. Since the default value of `minHealthFactor` is zero, if the owner does not set it in time, the wallet will suffer a denial of service issue.

Besides, once the `minHealthFactor` is set in the function `setMinHealthFactor()`, the check which ensures that the `minHealthFactor` is larger or equal than `1e18` will be redundant (line 22).

```
14  abstract contract LendingBaseACL is PostBaseACL {
    uint256 public minHealthFactor;

    constructor(address _owner, address _caller) PostBaseACL(_owner, _caller) {}

    // Set functions
    function setMinHealthFactor(uint256 _minHealthFactor) external onlyOwner {
        require(_minHealthFactor >= 1e18, "minHealthFactor cannot be lowerer than 1e18");
        minHealthFactor = _minHealthFactor;
    }

    function getHealthFactor() internal view virtual returns (uint256);

    function checkHealthFactor() internal view {
        require(minHealthFactor >= 1e18, "minHealthFactor not set");
        uint256 factor = getHealthFactor();
        require(factor > minHealthFactor, "Health factor too low");
    }
}
```

Listing 2.1: auth/LendingBaseACL.sol

Impact The user is not able to borrow assets from the lending protocol.

Suggestion Set the minHealthFactor in the constructor, and remove the redundant check in the function checkHealthFactor().

Feedback from the Project Users are expected to set minHealthFactor correctly before usage. Redundant check in checkHealthFactor() will be kept to reject the tx if minHealthFactor is not set.

2.1.2 Potential Incorrect Event Emitted

Severity Low

Status Fixed in Version 2

Introduced by Version 1

Description In the contract TransferAuthorizer, there is no check in the function addTokenReceivers() and removeTokenReceivers() to ensure the “token-receiver” pairs to be added or removed are already (or not) in the mapping. The event TokenReceiverAdded() and TokenReceiverRemoved() will always be emitted, which is incorrect.

```
    function addTokenReceivers(TokenReceiver[] calldata tokenReceivers) external onlyOwner {
        for (uint i = 0; i < tokenReceivers.length; i++) {
            tokenSet.add(tokenReceivers[i].token);
            tokenToReceivers[tokenReceivers[i].token].add(tokenReceivers[i].receiver);

            emit TokenReceiverAdded(tokenReceivers[i].token, tokenReceivers[i].receiver);
        }
    }

    function removeTokenReceivers(TokenReceiver[] calldata tokenReceivers) external onlyOwner {
        for (uint i = 0; i < tokenReceivers.length; i++) {
            tokenToReceivers[tokenReceivers[i].token].remove(tokenReceivers[i].receiver);

            emit TokenReceiverRemoved(tokenReceivers[i].token, tokenReceivers[i].receiver);
        }
    }
```

Listing 2.2: auth/TransferAuthorizer.sol

Impact Incorrect events may be emitted.

Suggestion If the “token-receiver” pair to be added or removed is already (or not) in the mapping, do not emit any events.

2.2 DeFi Security

2.2.1 Lack of Slippage Control when Swapping

Severity Medium

Status Confirmed

Introduced by `Version 1`

Description The contract `OneinchV5Authorizer` is an authorizer specifically designed for smart contracts interacting with the `1inch` protocol. It supports specific parameter checks before transaction execution (i.e., swap) to ensure the safety of assets in the wallet.

To prevent the potential sandwich attack during the swap, the parameter `minReturn` will be compared with the exact amount of output tokens in the `1inch` router. However, in the authorizer, there is no check on the `minReturn`.

```
function unoswap(address _srcToken, uint256, uint256, uint256[] calldata pools) external view {
    uint256 lastPool = pools[pools.length - 1];
    IPool lastPair = IPool(address(uint160(lastPool & _ADDRESS_MASK)));
    address srcToken = _getToken(_srcToken);
    bool isReversed = lastPool & _REVERSE_MASK == 0;
    address tokenOut = isReversed ? lastPair.token1() : lastPair.token0();
    swapInOutTokenCheck(srcToken, tokenOut);
}
```

Listing 2.3: acls/1inch/1inchV5Authorizer.sol

```
function uniswapV3Swap(uint256 amount, uint256 minReturn, uint256[] calldata pools) external
    view {
    uint256 lastPoolUint = pools[pools.length - 1];
    uint256 firstPoolUint = pools[0];
    IPool firstPool = IPool(address(uint160(firstPoolUint)));
    IPool lastPool = IPool(address(uint160(lastPoolUint)));
    bool zeroForOneFirstPool = firstPoolUint & _ONE_FOR_ZERO_MASK == 0;
    bool zeroForOneLastPool = lastPoolUint & _ONE_FOR_ZERO_MASK == 0;
    address srcToken = zeroForOneFirstPool ? firstPool.token0() : firstPool.token1();
    address dstToken = zeroForOneLastPool ? lastPool.token1() : firstPool.token0();
    swapInOutTokenCheck(srcToken, dstToken);
}
```

Listing 2.4: acls/1inch/1inchV5Authorizer.sol

Impact The user may suffer a loss due to the sandwich attack.

Suggestion Implement the corresponding check on the parameter minReturn is not zero.

Feedback from the Project Thanks. This issue is known to us. The delegate is expected to set the correct minReturn via the official 1inch frontend or API to avoid sandwich attacks. Additionally, since the access control here is executed on-chain, it is difficult to calculate the correct value of the minReturn variable in the contract. Simply checking minReturn > 0 makes little sense because it can still be set to small values like 1 or 2.

2.3 Additional Recommendation

2.3.1 Event Generation with Indexing

Status Fixed in Version 2

Introduced by Version 1

Description In the contract BaseOwnable, there are two different events (i.e., PendingOwnerSet() and NewOwnerSet()) whose admin addesses are not indexed. Since the admins may be frequently queried, the performance can be influenced without indexing.

```
abstract contract BaseOwnable is BaseVersion {
 address public owner;
 address public pendingOwner;
 bool private initialized = false;

 event PendingOwnerSet(address to);
 event NewOwnerSet(address owner);
```

Listing 2.5: base/BaseOwnable.sol

Suggestion Index the emitted address.

2.3.2 Redundant Check of Roles and Authorizers

Status Confirmed

Introduced by Version 1

Description In the internal function _postExecCheck(), there is a check to ensure the role of the caller (i.e., delegate) should contain the provided authorizer (line 229). However, the data is provided by the function _preExecCheck(), which has already completed the check.

```
161 function _preExecCheck(
162         TransactionData calldata transaction
163     ) internal override returns (AuthorizerReturnData memory authData) {
164         if (transaction.hint.length >= HINT_SIZE) {
165             return _preExecCheckWithHint(transaction);
166         }
167
168         authData.result = AuthResult.FAILED;
169         bytes32[] memory txRoles = _authenticate(transaction);
170         uint256 roleLength = txRoles.length;
```

```
        if (roleLength == 0) {
            authData.message = Errors.EMPTY_ROLE_SET;
            return authData;
        }

        bool isDelegateCall = transaction.flag.isDelegateCall();
        for (uint256 i = 0; i < roleLength; ++i) {
            bytes32 role = txRoles[i];
            EnumerableSet.AddressSet storage authSet = authorizerSet[isDelegateCall][role];

            uint256 length = authSet.length();

            // Run all pre checks and record auth results.
            for (uint256 j = 0; j < length; ++j) {
                address auth = authSet.at(j);
                AuthorizerReturnData memory preData = _safePreExecCheck(auth, transaction);

                if (preData.result == AuthResult.SUCCESS) {
                    authData.result = AuthResult.SUCCESS;

                    // Only save success results.
                    preCheckDataCache.push(PreCheckData(role, auth, preData));
                }
            }
        }

        if (authData.result == AuthResult.SUCCESS) {
            // Temporary data for post checker to collect hint.
            authData.data = abi.encode(preCheckDataCache);
        } else {
            authData.message = Errors.ALL_AUTH_FAILED;
        }

        delete preCheckDataCache; // gas refund.
    }
```

Listing 2.6: auth/ArgusRootAuthorizer.sol

```
function _postExecCheck(
        TransactionData calldata transaction,
        TransactionResult calldata callResult,
        AuthorizerReturnData calldata preData
    ) internal override returns (AuthorizerReturnData memory postData) {
        if (transaction.hint.length >= HINT_SIZE) {
            return _postExecCheckWithHint(transaction, callResult, preData);
        }

        // Get pre check results from preData.
        PreCheckData[] memory preResults = abi.decode(preData.data, (PreCheckData[]));
        uint256 length = preResults.length;

        // We should have reverted in preExecCheck. But safer is better.
        require(length > 0, Errors.INVALID_HINT_COLLECTED);

        bool isDelegateCall = transaction.flag.isDelegateCall();

        for (uint256 i = 0; i < length; ++i) {
            bytes32 role = preResults[i].role;
            address authAddress = preResults[i].authorizer;

            require(authorizerSet[isDelegateCall][role].contains(authAddress), Errors.
                INVALID_HINT_COLLECTED);

            // Run post check.
            AuthorizerReturnData memory preCheckData = preResults[i].authData;
            postData = _safePostExecCheck(authAddress, transaction, callResult, preCheckData);

            // If pre and post both succeeded, we pass.
            if (postData.result == AuthResult.SUCCESS) {
                // Collect hint of sub authorizer if needed.
                bytes memory subHint;
                if (IAuthorizer(authAddress).flag().supportHint()) {
                    subHint = _safeCollectHint(authAddress, preCheckData, postData);
                }
                postData.data = _packHint(role, authAddress, subHint);
                return postData;
            }
        }
        postData.result = AuthResult.FAILED;
        postData.message = Errors.ALL_AUTH_FAILED;
    }
```

Listing 2.7: auth/ArgusRootAuthorizer.sol

Suggestion Remove the redundant check.

Feedback from the Project The redundant check will be kept. A little more gas but safer to protect the case if only `postExecCheck()` is called.

2.3.3 Removal of Unused Token in tokenSet

Status Fixed in `Version 2`

Introduced by `Version 1`

Description The contract `TransferAuthorizer` is used to check whether the transfer of specific tokens to certain receivers is allowed. The privileged owner can add “token-receiver” pairs via the function `addTokenReceivers()`. The approved tokens will be recorded into the `tokenSet`. On the contrary, the function `removeTokenReceivers()` allows the privileged owner to remove “token-receiver” pairs in the `authorizer`. However, if a token has no corresponding receiver after removal, this function will not remove the token from the `tokenSet`.

```
    function removeTokenReceivers(TokenReceiver[] calldata tokenReceivers) external onlyOwner {
        for (uint i = 0; i < tokenReceivers.length; i++) {
            tokenToReceivers[tokenReceivers[i].token].remove(tokenReceivers[i].receiver);

            emit TokenReceiverRemoved(tokenReceivers[i].token, tokenReceivers[i].receiver);
        }
    }
```

Listing 2.8: auth/TransferAuthorizer.sol

Suggestion Remove the token from the `tokenSet` if the token has no corresponding receiver.

2.3.4 Lack of Length Check between Parameters

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `ArgusAuthorizerHelper`, there is no check in the functions `addFuncAuthorizerVariants()` and `removeFuncAuthorizerVariants()` to ensure the lengths of the parameter `_contracts` and `funcLists` are the same.

```
function addFuncAuthorizerVariants(
    address authorizerAddress,
    address[] calldata _contracts,
    string[][] calldata funcLists
) public {
    if (_contracts.length == 0) return;
    FuncAuthorizer authorizer = FuncAuthorizer(authorizerAddress);
    for (uint i = 0; i < _contracts.length; i++) {
        authorizer.addContractFuncs(_contracts[i], funcLists[i]);
    }
}

function removeFuncAuthorizerVariants(
    address authorizerAddress,
    address[] calldata _contracts,
    string[][] calldata funcLists
) external {
    if (_contracts.length == 0) return;
    FuncAuthorizer authorizer = FuncAuthorizer(authorizerAddress);
    for (uint i = 0; i < _contracts.length; i++) {
        authorizer.removeContractFuncs(_contracts[i], funcLists[i]);
    }
}
```

Listing 2.9: helper/ArgusAuthorizerHelper.sol

Suggestion Add the check to ensure these two arrays are in equal length.

2.3.5 Improper Implementation of Updating Whitelist

Status Fixed in `Version 2`

Introduced by `Version 1`

Description In the contract `DEXBaseACL`, the privileged owner can add or remove tokens that the wallet can swap in/out. However, the implementation is unreasonable. Specifically, in the function `setSwapInToken()`, the privileged owner has to provide the _tokenStatus to add (true) or remove (false) the token from the whitelist. In this case, if the token to be added is already in the whitelist, the transaction is a waste of gas. What's worse, there are no meaningful events emitted upon the corresponding operation.

Similar problems also exist in the function setSwapInTokens(), setSwapOutToken(), and setSwapOutTokens().

```
function setSwapInToken(address _token, bool _tokenStatus) external onlyOwner {
      // sell
      if (_tokenStatus) {
          swapInTokenWhitelist.add(_token);
      } else {
          swapInTokenWhitelist.remove(_token);
      }
  }

  function setSwapInTokens(SwapInToken[] calldata _swapInToken) external onlyOwner {
      for (uint256 i = 0; i < _swapInToken.length; i++) {
          if (_swapInToken[i].tokenStatus) {
              swapInTokenWhitelist.add(_swapInToken[i].token);
          } else {
              swapInTokenWhitelist.remove(_swapInToken[i].token);
          }
      }
  }

  function setSwapOutToken(address _token, bool _tokenStatus) external onlyOwner {
      // buy
      if (_tokenStatus) {
          swapOutTokenWhitelist.add(_token);
      } else {
          swapOutTokenWhitelist.remove(_token);
      }
  }

  function setSwapOutTokens(SwapOutToken[] calldata _swapOutToken) external onlyOwner {
      for (uint256 i = 0; i < _swapOutToken.length; i++) {
          if (_swapOutToken[i].tokenStatus) {
              swapOutTokenWhitelist.add(_swapOutToken[i].token);
          } else {
              swapOutTokenWhitelist.remove(_swapOutToken[i].token);
          }
      }
  }
```

Listing 2.10: auth/DEXBaseACL.sol

Suggestion Check the actual status of the token (whether on the whitelist) before adding or removing. Besides, add correct events accordingly.

2.4 Notes

2.4.1 Validation of Hints before Usage

Status Confirmed

Introduced by version 1

Description In the current implementation, the privileged user can invoke the functions execTransaction() and execTransactions() with valid TransactionData to initialize a transaction. In TransactionData, there is a hint field, which provides specific authorizers to the ArgusRootAuthorizer for further checks. It allows skipping other unrelated authorizers and directly identifies the needed authorizers to be interacted with.

Although the hint field will be checked in the functions _preExecCheck() and _postExecCheck(), it may still be used in the functions _preExecProcess() and _postExecProcess(). In this case, there must be sufficient validation in the function _preExecProcess() and _postExecProcess() if the hint field is used.

2.4.2 Ignored Reverts in _preExecCheck() and _postExecCheck()

Status Confirmed

Introduced by version 1

Description In the functions _preExecProcess() and _postExecProcess(), all reverts will be ignored as all checks are done in the functions _preExecCheck() and _postExecCheck(). There should be no security issues upon the reverts in the functions _preExecProcess() and _postExecProcess().

2.4.3 Deployments of Contracts within Argus Protocol

Status Confirmed

Introduced by version 2

Description Here is a list of all current contract depolyments within the Argus protocol.

Ethereum

Contract	Address
OneinchV5Authorizer.sol	0xfecE55912861a401738604c52998604ba45115a1
ArgusRootAuthorizer.sol	0x3a36AD7f6C916B780733CFC71C8538716Dc55084
FuncAuthorizer.sol	0x92DdB2B7D17FF42078AFFf98721F6d1E38083ED6
TransferAuthorizer.sol	0x2148c4F124029c3A18CFcC7A86A67A5Bf4D88658
ArgusAccountHelper.sol	0x095665121A79F340CB1Ff1883015A78a089D96CE
ArgusViewHelper.sol	0xF97BB9AF9FE6A68b324EdBcd0fE698E631F5113A
FlatRoleManager.sol	0x2F2FDDb984cdEC4318D8d87BC70821e9B9Ed8e7E
CoboFactory.sol	0xC0B00000e19D71fA50a9BB1fcaC2eC92fac9549C
CoboSafeAccount.sol	0xE7168444CF4c25800C2817BFDC6dcf17C261994d

Binance Smart Chain

Contract	Address
OneinchV5Authorizer.sol	0x44362a387f5243be4a0355c706200ad2ea9b3CB7
ArgusRootAuthorizer.sol	0xC7D1bD5106785051AAf841805171eae5C396bc5A
FuncAuthorizer.sol	0x6DDe0424ae9ADaf5d305e20720Be6B9BC3f5ae8a

TransferAuthorizer.sol	0x1dB643a720856b1406499e7046414D317A5a6d4b
ArgusAccountHelper.sol	0x68A8e92936A30A4A66Ce4705081Da2260Cc1c670
ArgusViewHelper.sol	0x2f5F6B42678704B5A738456D1320327cea95ae09
FlatRoleManager.sol	0x5311Cc807625F54Eb810a4a0bEa5B4d2533961F0
CoboFactory.sol	0xC0B00000e19D71fA50a9BB1fcaC2eC92fac9549C
CoboSafeAccount.sol	0x16119BF35b764e6AB83DEDA11719F5a5Bb0C4dfD
StargateClaimAuthorizer.sol	0xbd9bDfF5636709cA9ff6a1598896D50Ce9d3E4cC
StargateDepositAuthorizer.sol	0x96BF0122E8212A6A5296c981c7ef062EfE4F8E7f
StargateWithdrawAuthorizer.sol	0x9f0910e9c31cC0442A9188264630Ef9E0dC51969

Polygon

Contract	Address
OneinchV5Authorizer.sol	0xD566FD8BF501Cd585Ed153Db828dcf880c1fE3fd
ArgusRootAuthorizer.sol	0x3265892Ca064302A615289a49E0C2aA46e5FdF94
FuncAuthorizer.sol	0x6DDe0424ae9ADaf5d305e20720Be6B9BC3f5ae8a
TransferAuthorizer.sol	0x9dB7299bBDDDBd30ac35A84Ca178a7E737357892
ArgusAccountHelper.sol	0x9f0910e9c31cC0442A9188264630Ef9E0dC51969
ArgusViewHelper.sol	0x20D0b245f72018c0EC105eCEDd11400124b518DB
FlatRoleManager.sol	0x16119BF35b764e6AB83DEDA11719F5a5Bb0C4dfD
CoboFactory.sol	0xC0B00000e19D71fA50a9BB1fcaC2eC92fac9549C
CoboSafeAccount.sol	0x9e9b19394cD85d2620af2689B16B0a95F69176Dc
StargateClaimAuthorizer.sol	0x2d9899Be6d1e57E3ee61Ee20DFb246fF22a0fdff
StargateDepositAuthorizer.sol	0x294b34Ec45429afE5b2DdC700C850032d87a3766
StargateWithdrawAuthorizer.sol	0x376819712D23F3a3775C416a1Ad5E7a8a05487d4

Avalanche

Contract	Address
OneinchV5Authorizer.sol	0x7Ba3CC542b70f8F1D6282dae222235D42CFd34CD
ArgusRootAuthorizer.sol	0xda185d7e539d4866FEeFCBAD0b437eA590715905
FuncAuthorizer.sol	0x37c43Df81B967d9Ee54bCcc0202bC8962bF7c3c2
TransferAuthorizer.sol	0x929fEA220AeEb5e09508fc1581202FeD84DcFD56
ArgusAccountHelper.sol	0x51e6540A5E766EB864aa32F548433D892Fd6008a
ArgusViewHelper.sol	0xE016BdEEd6f31A3C509621104bFE103fa7476B12
FlatRoleManager.sol	0x55059108c6b7F4f6085f485863EFE3e34D493368
CoboFactory.sol	0xC0B00000e19D71fA50a9BB1fcaC2eC92fac9549C
CoboSafeAccount.sol	0x7677E361aEC4ee6e13D27806BC914Dd35c0Da0D8
StargateClaimAuthorizer.sol	0xb1314e31a606ecd8F30c29b9149388529445 3BA3
StargateDepositAuthorizer.sol	0x30c4a1a21A14281c8EB5AE75fd874359D01200ED
StargateWithdrawAuthorizer.sol	0x59E3C907abe047f731e570B56D671EdFE57d2277

Arbitrum

Contract	Address
OneinchV5Authorizer.sol	0xE7CA78dc87B54EF3e0Ed82cC77F449772C469414
ArgusRootAuthorizer.sol	0xF935E7150a61e17e26f5F7FA6a305903A605F52b
FuncAuthorizer.sol	0x929fEA220AeEb5e09508fc1581202FeD84DcFD56
TransferAuthorizer.sol	0x1552C84f6f09B6117dD95996d8220B37Ca6BDC4F
ArgusAccountHelper.sol	0xE016BdEEd6f31A3C509621104bFE103fa7476B12
ArgusViewHelper.sol	0x7677E361aEC4ee6e13D27806BC914Dd35c0Da0D8
FlatRoleManager.sol	0x37c43Df81B967d9Ee54bCcc0202bC8962bF7c3c2
CoboFactory.sol	0xC0B00000e19D71fA50a9BB1fcaC2eC92fac9549C
CoboSafeAccount.sol	0x55059108c6b7F4f6085f485863EFE3e34D493368
StargateClaimAuthorizer.sol	0x73a08503931Bd6763C4CD60013802025F1fCc3D2
StargateDepositAuthorizer.sol	0x58bF21e7a425c92C4Af55928FfA9b28a38f7d2cc
StargateWithdrawAuthorizer.sol	0xABA1D868D89F29b46499E84C73BdE47481Af8074

Optimism

Contract	Address
OneinchV5Authorizer.sol	0x07f2AD9A6299E89019793706Ae39A780b49CDdDc
ArgusRootAuthorizer.sol	0x8524e4fBA45Eb8e70e22A1B9d3974DEfEe86bB42
FuncAuthorizer.sol	0x54815296e3b5ed59Ec50be739Fa7CcA4E8de5eC0
TransferAuthorizer.sol	0x8C230beB7649b016e52E85CF50777d3253068d6a
ArgusAccountHelper.sol	0x881C1695A5D6B0F251BcBb7BaB08Ec589c171cBc
ArgusViewHelper.sol	0x41e5C89Bb2207AaF9ae07441f64b0b822aB8a6ac
FlatRoleManager.sol	0x3D89555e239209F6Aa708520302eD8B4eD859791
CoboFactory.sol	0xC0B00000e19D71fA50a9BB1fcaC2eC92fac9549C
CoboSafeAccount.sol	0x37E369301beddd49574d22A7fa034d596766004B
StargateClaimAuthorizer.sol	0xe6f35629e03E755CC2f977DaAC0E45663B66c8E2
StargateDepositAuthorizer.sol	0x73a08503931Bd6763C4CD60013802025F1fCc3D2
StargateWithdrawAuthorizer.sol	0x58bF21e7a425c92C4Af55928FfA9b28a38f7d2cc