



Cyberscope

Audit Report

ArbVault

February 2025

Network ARBITRUM

Address 0x09e9222E96E7B4AE2a407B98d48e330053351EEe

Audited by © cyberscope

Table of Contents

Table of Contents	1
Risk Classification	3
Review	4
Audit Updates	4
Source Files	4
Overview	6
L2 Token Bridge Contracts for Arbitrum	6
Token Contracts	6
aeERC20 Contract	6
StandardArbERC20 Contract	6
TransferAndCallToken Contract	6
Token Bridge Contracts (For L2)	7
L2GatewayToken Contract	7
L2ArbitrumGateway Contract	7
L2ERC20Gateway Contract	7
Proxy and Upgradeability System	8
ClonableBeaconProxy Contract	8
TokenGateway Contract	8
Cross-Chain Messaging Contracts	8
L2ArbitrumMessenger Contract	8
GatewayMessageHandler Library	8
BytesParser Library	8
Findings Breakdown	9
Diagnostics	10
AME - Address Manipulation Exploit	11
Description	11
Recommendation	12
UTPD - Unverified Third Party Dependencies	13
Description	13
Recommendation	13
BT - Burns Tokens	14
Description	14
Recommendation	14
CCR - Contract Centralization Risk	15
Description	15
Recommendation	16
IDI - Immutable Declaration Improvement	17
Description	17
Recommendation	17

MT - Mints Tokens	18
Description	18
Recommendation	18
TDI - Token Data Inconsistencies	19
Description	19
Recommendation	20
L04 - Conformance to Solidity Naming Conventions	21
Description	21
Recommendation	22
L09 - Dead Code Elimination	23
Description	23
Recommendation	23
L14 - Uninitialized Variables in Local Scope	24
Description	24
Recommendation	24
L17 - Usage of Solidity Assembly	25
Description	25
Recommendation	26
L18 - Multiple Pragma Directives	27
Description	27
Recommendation	27
L19 - Stable Compiler Version	28
Description	28
Recommendation	29
Functions Analysis	30
Inheritance Graph	35
Flow Graph	36
Summary	37
Disclaimer	38
About Cyberscope	39

Risk Classification

The criticality of findings in Cyberscope's smart contract audits is determined by evaluating multiple variables. The two primary variables are:

1. **Likelihood of Exploitation:** This considers how easily an attack can be executed, including the economic feasibility for an attacker.
2. **Impact of Exploitation:** This assesses the potential consequences of an attack, particularly in terms of the loss of funds or disruption to the contract's functionality.

Based on these variables, findings are categorized into the following severity levels:

1. **Critical:** Indicates a vulnerability that is both highly likely to be exploited and can result in significant fund loss or severe disruption. Immediate action is required to address these issues.
2. **Medium:** Refers to vulnerabilities that are either less likely to be exploited or would have a moderate impact if exploited. These issues should be addressed in due course to ensure overall contract security.
3. **Minor:** Involves vulnerabilities that are unlikely to be exploited and would have a minor impact. These findings should still be considered for resolution to maintain best practices in security.
4. **Informative:** Points out potential improvements or informational notes that do not pose an immediate risk. Addressing these can enhance the overall quality and robustness of the contract.

Severity	Likelihood / Impact of Exploitation
● Critical	Highly Likely / High Impact
● Medium	Less Likely / High Impact or Highly Likely/ Lower Impact
● Minor / Informative	Unlikely / Low to no Impact

Review

Proxy Explorer	https://arbiscan.io/address/0x09e9222E96E7B4AE2a407B98d48e330053351EEe
Implementation Explorer	https://arbiscan.io/address/0x1dcf7d03574fbc7c205f41f2e116ee094a652e93

Audit Updates

Initial Audit	20 Feb 2025
---------------	-------------

Source Files

Filename	SHA256
contracts/tokenbridge/libraries/aeERC20.sol	179a129d47a1fa4d4635137821c75c20b45505fee4c1e2e0c2059a70baaa6404
contracts/tokenbridge/libraries/TransferAndCallToken.sol	6b03cdac63dc91527b215304bf59380cbc f9ae006c70bbce4465c00931d0cd95
contracts/tokenbridge/libraries/L2GatewayToken.sol	04d6dedf712a2c9709d5ec87adaff66a6eb e20bfc2c11bd5702ad4eb663bb875
contracts/tokenbridge/libraries/ITransferAndCall.sol	a3f80ebe34ea7f580124723cc7133ba3ec9 063fb9866f3aeebf4b48f25ce9b5d
contracts/tokenbridge/libraries/ClonableBeaconProxy.sol	23d6865521f1ec049d299b9be1985235a0 9f686c92d514ea6a8f606fccb992bb
contracts/tokenbridge/libraries/BytesParser.sol	402ebe70cf8d51533b8ff5fc6cb2cc6a2c42 1acb6ccdc8db103692cfb525d6cb
contracts/tokenbridge/libraries/gateway/TokenGateway.sol	931927def02a67fa1ccaf55883532ad2081 17e7abf684319a1082d9fda3e41ac

contracts/tokenbridge/libraries/gateway/ITokenGateway.sol	cbd83670dbf120a54297f794555f1bcf2a312fa8ca448dde2e84db0708a74d4e
contracts/tokenbridge/libraries/gateway/GatewayMessageHandler.sol	dcbddad464dade4165ddc8158dcf6d1ea281d67cd8b588200c1ac8da88526c4b
contracts/tokenbridge/arbitrum/StandardArbERC20.sol	6b6e3feb5452b716200acb552174223522a68382cec3eeb333e6bab26827b2f2
contracts/tokenbridge/arbitrum/L2ArbitrumMessenger.sol	3c777825a596da24506cf0041e9f825d1ca78d3fceb2747bf10a69d788f54ff4
contracts/tokenbridge/arbitrum/IArbToken.sol	cecdc54699e6306369b18e6fa061600bd6d4762ff99b060ccb3b851a4b7f8801
contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol	693849802eb8559653191f0dc240b36ab5d9599694307ec8e7d34127e98c2d53
contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol	c5d2194660c7d22aa3c87fd3a5250d1dff93bef89cf2e521ee3b7edde73d0306
arbos-precompiles/arbos/builtin/ArbSys.sol	edbf1afb97fa8681795dc1d0f899a265329104f857dd8a12f74abafd5ccb4350
arb-bridge-eth/contracts/libraries/ProxyUtil.sol	9995e79253dc1b2663ca0b60720f1a3f117cfed4530bf5af627f0a91c01a7eda
arb-bridge-eth/contracts/libraries/ICloneable.sol	4eabca7df61d226b7a4607dd9b618805310143eb8ef07a6eaae3f491c5f9b4ed
arb-bridge-eth/contracts/libraries/Cloneable.sol	6b4481de7e15baef38b560b8da860e9b69f8dca1d0b9061fd7f818a8b4e0ceb2
arb-bridge-eth/contracts/libraries/BytesLib.sol	b1bc50bb3581e94704a2e2733ad6287059869c1297edacfa2fd11ab4320712e0
arb-bridge-eth/contracts/libraries/AddressAliasHelper.sol	f1a06fc52bc83eb08c517eacf5bd9ff6b397bd4a643d20a6090653626bf53571

Overview

L2 Token Bridge Contracts for Arbitrum

The following contracts implement a Layer 2 (L2) token bridge for Arbitrum, facilitating ERC20 token transfers between Ethereum (L1) and Arbitrum (L2). The system manages token deposits, withdrawals, and cross-chain messaging using proxy-based upgradability.

Token Contracts

aeERC20 Contract

The `aeERC20` contract is an Arbitrum-extended `ERC20` token that supports permit-based approvals and cross-chain interactions. It extends the ERC20 standard with additional functionalities tailored for L2 environments. It implements `ERC20PermitUpgradeable`, allowing gasless approvals. It also supports `transferAndCall` for contract interactions. Generally, it is designed to be deployed behind a proxy, requiring an `initialize` function instead of a `constructor`.

StandardArbERC20 Contract

The `StandardArbERC20` contract is automatically deployed by the `L2Gateway.sol` contract when bridging an ERC20 token. It ensures `ERC20` standard compliance while supporting deposit and withdrawal functions. For token minting on L2 upon deposit, it uses the `bridgeMint` function. It also has the `bridgeBurn` to burn tokens on L2 upon withdrawal to L1. Only authorized gateway contracts can execute minting and burning operations.

TransferAndCallToken Contract

The `TransferAndCallToken` contract extends `ERC20` with the `ERC677` standard, enabling smart contract notifications upon transfers. It implements the `transferAndCall` function to trigger contract actions upon token transfer. Additionally, it supports interactions with external contracts. The `isContract` function is used for verification.

Token Bridge Contracts (For L2)

L2GatewayToken Contract

The `L2GatewayToken` contract manages minting and burning of `ERC20` tokens bridged from L1 to L2. It serves as the token contract counterpart on L2. When a user deposits tokens on L1, the `bridgeMint` function is called to create new tokens on L2. When a user withdraws tokens to L1, the `bridgeBurn` function removes tokens from circulation. It uses an `onlyGateway` modifier to restrict execution to authorized bridge contracts.

L2ArbitrumGateway Contract

The `L2ArbitrumGateway` contract is responsible for processing deposits and withdrawals between L1 and L2. It handles cross-chain messaging for deposits and withdrawals. `calculateL2TokenAddress` is used to determine whether an L2 token contract exists. If an L2 contract is not deployed, it will deploy a new token contract automatically if `deployData` is included, else it will revert the deposit and initiate an L1 withdrawal refund if no `deployData` is provided.

L2ERC20Gateway Contract

The `L2ERC20Gateway` contract is a specialized gateway for `ERC20` tokens. It extends `L2ArbitrumGateway` and provides additional functionality for `ERC20` bridging. The `BeaconProxyFactory.createProxy` is used to deploy new L2 token contracts. It also handles withdrawals so that only expected L1-L2 token pairs interact.

Proxy and Upgradeability System

ClonableBeaconProxy Contract

The `ClonableBeaconProxy` contract allows upgradable deployments of L2 token contracts using the Beacon Proxy Pattern. All L2 `ERC20` tokens can be upgraded in a single transaction by modifying the beacon contract. Uses `BeaconProxyFactory.createProxy` to generate new `ERC20` instances. It also implements `getSalt` to derive unique addresses for cloned contracts.

TokenGateway Contract

The `TokenGateway` contract serves as a base class for gateway implementations. It defines common functions for L1-L2 token bridging. It uses `counterpartGateway` to reference the L1 gateway. Additionally there is a shared `_initialize` function for all gateways.

Cross-Chain Messaging Contracts

L2ArbitrumMessenger Contract

The `L2ArbitrumMessenger` contract facilitates message passing between L1 and L2. It uses the `ArbSys.sendTxToL1` to send messages from L2 to L1. It allows outbound and inbound communication between bridge contracts while ensuring correct ordering of transactions.

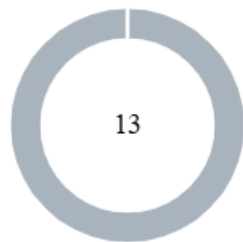
GatewayMessageHandler Library

The `GatewayMessageHandler` library is responsible for encoding and decoding cross-chain transaction data. Encodes deposit/withdrawal messages and decodes messages on L2 to process token transfers. It also supports router-to-gateway communication.

BytesParser Library

The `BytesParser` library provides utility functions for parsing encoded transaction data.

Findings Breakdown



● Critical	0
● Medium	0
● Minor / Informative	13

Severity	Unresolved	Acknowledged	Resolved	Other
● Critical	0	0	0	0
● Medium	0	0	0	0
● Minor / Informative	13	0	0	0

Diagnostics

● Critical ● Medium ● Minor / Informative

Severity	Code	Description	Status
●	AME	Address Manipulation Exploit	Unresolved
●	UTPD	Unverified Third Party Dependencies	Unresolved
●	BT	Burns Tokens	Unresolved
●	CCR	Contract Centralization Risk	Unresolved
●	IDI	Immutable Declaration Improvement	Unresolved
●	MT	Mints Tokens	Unresolved
●	TDI	Token Data Inconsistencies	Unresolved
●	L04	Conformance to Solidity Naming Conventions	Unresolved
●	L09	Dead Code Elimination	Unresolved
●	L14	Uninitialized Variables in Local Scope	Unresolved
●	L17	Usage of Solidity Assembly	Unresolved
●	L18	Multiple Pragma Directives	Unresolved
●	L19	Stable Compiler Version	Unresolved

AME - Address Manipulation Exploit

Criticality	Minor / Informative
Location	contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol#L166
Status	Unresolved

Description

The contract's design includes functions that accept external contract addresses as parameters without performing adequate validation or authenticity checks. This lack of verification introduces a significant security risk, as input addresses could be controlled by attackers and point to malicious contracts. Such vulnerabilities could enable attackers to exploit these functions, potentially leading to unauthorized actions or the execution of malicious code under the guise of legitimate operations.

Specifically, during the `outboundTransfer`, that anyone can call, a static call is used to `l2Token` to validate the `l1Address`. It is possible that `l1Address` may match the `_l1Token` that is added as an external parameter and perform the transfer. If the `_l1Token` has malicious code, funds may be stolen.

```
{
    address l2Token = calculateL2TokenAddress(_l1Token);
    require(l2Token.isContract(), "TOKEN_NOT_DEPLOYED");
    require(IArbToken(l2Token).l1Address() == _l1Token,
"NOT_EXPECTED_L1_TOKEN");

    _amount = outboundEscrowTransfer(l2Token, _from, _amount);
    id = triggerWithdrawal(_l1Token, _from, _to, _amount,
_extraData);
}
```

Recommendation

To mitigate this risk and enhance the contract's security posture, it is imperative to incorporate comprehensive validation mechanisms for any external contract addresses passed as parameters to functions. This could include checks against a whitelist of approved addresses, verification that the address implements a specific contract interface or other methods that confirm the legitimacy and integrity of the external contract. Implementing such validations helps prevent malicious exploits and ensures that only trusted contracts can interact with sensitive functions.

UTPD - Unverified Third Party Dependencies

Criticality	Minor / Informative
Location	contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol#L213,233
Status	Unresolved

Description

The contract uses an external contract or addresses that are contracts in other layers in order to determine the transaction's flow. These external parties are untrusted. As a result, they may produce security issues and harm the transactions.

```
if (isRouter(msg.sender)) {
    (_from, _extraData) =
GatewayMessageHandler.parseFromRouterToGateway(_data);
}
//...
function finalizeInboundTransfer(
    address _token,
    address _from,
    address _to,
    uint256 _amount,
    bytes calldata _data
) external payable override onlyCounterpartGateway {}
```

Recommendation

The contract should use a trusted external source. A trusted source could be either a commonly recognized or an audited contract. The pointing addresses should not be able to change after the initialization.

BT - Burns Tokens

Criticality	Minor / Informative
Location	contracts/tokenbridge/libraries/L2GatewayToken.sol#L76
Status	Unresolved

Description

`I2Gateway` has the authority to burn tokens from a specific address. They may take advantage of it by calling the `bridgeBurn` function. As a result, the targeted address will lose the corresponding tokens.

```
function bridgeBurn(address account, uint256 amount) external
virtual override onlyGateway {
    _burn(account, amount);
}
```

Recommendation

The team should carefully manage the way `I2Gateway` is handling the burning of tokens especially since the contracts are upgradable. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin and upgradable functions.

Possible solutions:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

CCR - Contract Centralization Risk

Criticality	Minor / Informative
Location	contracts/tokenbridge/arbitrum/StandardArbERC20.sol#L43 contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol#L56,63,87 contracts/tokenbridge/libraries/L2GatewayToken.sol#L66,76 contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol#L64,227
Status	Unresolved

Description

The contract's functionality and behavior are heavily dependent on external parameters or configurations. While external configuration can offer flexibility, it also poses several centralization risks that warrant attention. Centralization risks arising from the dependence on external configuration include Single Point of Control, Vulnerability to Attacks, Operational Delays, Trust Dependencies, and Decentralization Erosion.


```
function bridgeInit(address _l1Address, bytes memory _data)
public virtual {}
//...
BeaconProxyFactory(beaconProxyFactory).calculateExpectedAddress
(
    address(this),
    getUserSalt(11ERC20)
);
//...
BeaconProxyFactory(beaconProxyFactory).cloneableProxyHash();
//...
BeaconProxyFactory(beaconProxyFactory).createProxy(userSalt);
//...
function bridgeMint(address account, uint256 amount) external
virtual override onlyGateway {}
//...
function bridgeBurn(address account, uint256 amount) external
virtual override onlyGateway {}
//...
function postUpgradeInit() external {
    address proxyAdmin = ProxyUtil.getProxyAdmin();
    require(msg.sender == proxyAdmin, "NOT_FROM_ADMIN");
}
//...
function finalizeInboundTransfer(**args**) external payable
override onlyCounterpartGateway {}
//...
```

Recommendation

To address this finding and mitigate centralization risks, it is recommended to evaluate the feasibility of migrating critical configurations and functionality into the contract's codebase itself. This approach would reduce external dependencies and enhance the contract's self-sufficiency. It is essential to carefully weigh the trade-offs between external configuration flexibility and the risks associated with centralization.

IDI - Immutable Declaration Improvement

Criticality	Minor / Informative
Location	arb-bridge-eth/contracts/libraries/Cloneable.sol#L29
Status	Unresolved

Description

The contract declares state variables that their value is initialized once in the constructor and are not modified afterwards. The `immutable` is a special declaration for this kind of state variables that saves gas when it is defined.

```
isMasterCopy
```

Recommendation

By declaring a variable as immutable, the Solidity compiler is able to make certain optimizations. This can reduce the amount of storage and computation required by the contract, and make it more gas-efficient.

MT - Mints Tokens

Criticality	Minor / Informative
Location	contracts/tokenbridge/libraries/L2GatewayToken.sol#L66
Status	Unresolved

Description

`l2Gateway` address has the authority to mint tokens. They may take advantage of it by calling the `bridgeMint` function. As a result, token contacts generated may be highly inflated.

```
function bridgeMint(address account, uint256 amount) external
virtual override onlyGateway {
    _mint(account, amount);
}
```

Recommendation

The team should carefully manage the way `l2Gateway` is handling the minting of tokens especially since the contracts are upgradable. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin and upgradable functions.

Possible solutions:

- Introduce a time-locker mechanism with a reasonable delay.
- Introduce a multi-signature wallet so that many addresses will confirm the action.
- Introduce a governance model where users will vote about the actions.

TDI - Token Data Inconsistencies

Criticality	Minor / Informative
Location	contracts/tokenbridge/arbitrum/StandardArbERC20.sol#L81,87,93
Status	Unresolved

Description

The `bridgeInit` method handles the case of both successfully retrieving and failing to retrieve the data of the `ERC20` such as `name`, `symbol` and `decimals`. However in case of success for a token that does not follow the exact `ERC20` standards, the corresponding default functions will be used instead. This may create inconsistencies to the tokens between the two layers.

```
function bridgeInit(address _l1Address, bytes memory _data)
public virtual {
    (bytes memory name_, bytes memory symbol_, bytes memory
decimals_) = abi.decode(
        _data,
        (bytes, bytes, bytes)
    );
    (bool parseNameSuccess, string memory parsedName) =
BytesParser.toString(name_);
    (bool parseSymbolSuccess, string memory parsedSymbol) =
BytesParser.toString(symbol_);
    (bool parseDecimalSuccess, uint8 parsedDecimals) =
BytesParser.toUint8(decimals_);
    L2GatewayToken._initialize(
        parsedName,
        parsedSymbol,
        parsedDecimals,
        msg.sender,
        _l1Address
    );
    availableGetters = ERC20Getters({
        ignoreName: !parseNameSuccess,
        ignoreSymbol: !parseSymbolSuccess,
        ignoreDecimals: !parseDecimalSuccess
    });
}
```

Recommendation

The team should consider the possibility of not accepting non-standard implementation of tokens as by accepting them inconsistencies may be created.

L04 - Conformance to Solidity Naming Conventions

Criticality	Minor / Informative
Location	contracts/tokenbridge/libraries/TransferAndCallToken.sol#L21,22,23,36,37,38,44 contracts/tokenbridge/libraries/gateway/TokenGateway.sol#L46 contracts/tokenbridge/libraries/gateway/GatewayMessageHandler.sol#L33,52,63,72 contracts/tokenbridge/libraries/ClonableBeaconProxy.sol#L27 contracts/tokenbridge/libraries/aeERC20.sol#L27 contracts/tokenbridge/arbitrum/StandardArbERC20.sol#L43 contracts/tokenbridge/arbitrum/L2ArbitrumMessenger.sol#L31,32,33,34 contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol#L31,32,33,81,83 contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol#L79,81,99,100,101,102,103,118,119,120,121,134,135,136,139,175,176,177,178,179,194,195,196,208,209,210,228,229,230,231,232 arb-bridge-eth/contracts/libraries/BytesLib.sol#L15,26,37,48 arb-bridge-eth/contracts/libraries/AddressAliasHelper.sol#L22
Status	Unresolved

Description

The Solidity style guide is a set of guidelines for writing clean and consistent Solidity code. Adhering to a style guide can help improve the readability and maintainability of the Solidity code, making it easier for others to understand and work with.

The followings are a few key points from the Solidity style guide:

1. Use camelCase for function and variable names, with the first letter in lowercase (e.g., myVariable, updateCounter).
2. Use PascalCase for contract, struct, and enum names, with the first letter in uppercase (e.g., MyContract, UserStruct, ErrorEnum).
3. Use uppercase for constant variables and enums (e.g., MAX_VALUE, ERROR_CODE).
4. Use indentation to improve readability and structure.
5. Use spaces between operators and after commas.
6. Use comments to explain the purpose and behavior of the code.
7. Keep lines short (around 120 characters) to improve readability.

```
address _to
uint256 _value
bytes memory _data
address _addr
address _target
bytes calldata _data
address _from
address _beacon

...
```

Recommendation

By following the Solidity naming convention guidelines, the codebase increased the readability, maintainability, and makes it easier to work with.

Find more information on the Solidity documentation

<https://docs.soliditylang.org/en/stable/style-guide.html#naming-conventions>.

L09 - Dead Code Elimination

Criticality	Minor / Informative
Location	arb-bridge-eth/contracts/libraries/Cloneable.sol#L36
Status	Unresolved

Description

In Solidity, dead code is code that is written in the contract, but is never executed or reached during normal contract execution. Dead code can occur for a variety of reasons, such as:

- Conditional statements that are always false.
- Functions that are never called.
- Unreachable code (e.g., code that follows a return statement).

Dead code can make a contract more difficult to understand and maintain, and can also increase the size of the contract and the cost of deploying and interacting with it.

```
function safeSelfDestruct(address payable dest) internal {  
    require(!isMasterCopy, NOT_CLONE);  
    selfdestruct(dest);  
}
```

Recommendation

To avoid creating dead code, it's important to carefully consider the logic and flow of the contract and to remove any code that is not needed or that is never executed. This can help improve the clarity and efficiency of the contract.

L14 - Uninitialized Variables in Local Scope

Criticality	Minor / Informative
Location	contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol#L263
Status	Unresolved

Description

Using an uninitialized local variable can lead to unpredictable behavior and potentially cause errors in the contract. It's important to always initialize local variables with appropriate values before using them.

```
bool shouldWithdraw
```

Recommendation

By initializing local variables before using them, the contract ensures that the functions behave as expected and avoid potential issues.

L17 - Usage of Solidity Assembly

Criticality	Minor / Informative
Location	contracts/tokenbridge/libraries/TransferAndCallToken.sol#L46 contracts/tokenbridge/libraries/BytesParser.sol#L62 arb-bridge-eth/contracts/libraries/ProxyUtil.sol#L27 arb-bridge-eth/contracts/libraries/BytesLib.sol#L19,30,41,52
Status	Unresolved

Description

Using assembly can be useful for optimizing code, but it can also be error-prone. It's important to carefully test and debug assembly code to ensure that it is correct and does not contain any errors.

Some common types of errors that can occur when using assembly in Solidity include Syntax, Type, Out-of-bounds, Stack, and Revert.

```
assembly {
    length := extcodesize(_addr)
}
//...
assembly {
    res := inputTruncated
}
//...
assembly {
    admin := sload(slot)
}
//...
assembly {
    tempAddress := div(mload(add(add(_bytes, 0x20),
_start)), 0x100000000000000000000000000000000)
}
//...
assembly {
    tempUint := mload(add(add(_bytes, 0x1), _start))
}
//...
assembly {
    tempUint := mload(add(add(_bytes, 0x20), _start))
}
//...
assembly {
    tempBytes32 := mload(add(add(_bytes, 0x20), _start))
}
```

Recommendation

It is recommended to use assembly sparingly and only when necessary, as it can be difficult to read and understand compared to Solidity code.

L18 - Multiple Pragma Directives

Criticality	Minor / Informative
Location	contracts/tokenbridge/libraries/TransferAndCallToken.sol#L3 contracts/tokenbridge/libraries/L2GatewayToken.sol#L19 contracts/tokenbridge/libraries/gateway/TokenGateway.sol#L19 contracts/tokenbridge/libraries/gateway/GatewayMessageHandler.sol#L19 contracts/tokenbridge/libraries/ClonableBeaconProxy.sol#L3 contracts/tokenbridge/libraries/BytesParser.sol#L19 contracts/tokenbridge/libraries/aeERC20.sol#L19 contracts/tokenbridge/arbitrum/StandardArbERC20.sol#L19 contracts/tokenbridge/arbitrum/L2ArbitrumMessenger.sol#L19 contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol#L19 contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol#L19 arb-bridge-eth/contracts/libraries/ProxyUtil.sol#L19 arb-bridge-eth/contracts/libraries/Cloneable.sol#L19 arb-bridge-eth/contracts/libraries/BytesLib.sol#L11 arb-bridge-eth/contracts/libraries/AddressAliasHelper.sol#L19
Status	Unresolved

Description

If the contract includes multiple conflicting pragma directives, it may produce unexpected errors. To avoid this, it's important to include the correct pragma directive at the top of the contract and to ensure that it is the only pragma directive included in the contract.

```
pragma solidity ^0.6.11;  
pragma solidity >0.6.0 <0.8.0;
```

Recommendation

It is important to include only one pragma directive at the top of the contract and to ensure that it accurately reflects the version of Solidity that the contract is written in.

By including all required compiler options and flags in a single pragma directive, the potential conflicts could be avoided and ensure that the contract can be compiled correctly.

L19 - Stable Compiler Version

Criticality	Minor / Informative
Location	contracts/tokenbridge/arbitrum/gateway/L2ArbitrumGateway.sol#L19 contracts/tokenbridge/arbitrum/gateway/L2ERC20Gateway.sol#L19 contracts/tokenbridge/arbitrum/L2ArbitrumMessenger.sol#L19 contracts/tokenbridge/arbitrum/StandardArbERC20.sol#L19 contracts/tokenbridge/libraries/gateway/GatewayMessageHandler.sol#L19 contracts/tokenbridge/libraries/gateway/TokenGateway.sol#L19 contracts/tokenbridge/libraries/BytesParser.sol#L19 contracts/tokenbridge/libraries/aeERC20.sol#L19 contracts/tokenbridge/libraries/ClonableBeaconProxy.sol#L3 contracts/tokenbridge/libraries/L2GatewayToken.sol#L19 contracts/tokenbridge/libraries/TransferAndCallToken.sol#L3 arb-bridge-eth/contracts/libraries/ProxyUtil.sol#L19 arb-bridge-eth/contracts/libraries/Cloneable.sol#L19 arb-bridge-eth/contracts/libraries/BytesLib.sol#L11 arb-bridge-eth/contracts/libraries/AddressAliasHelper.sol#L19
Status	Unresolved

Description

The `^` symbol indicates that any version of Solidity that is compatible with the specified version (i.e., any version that is a higher minor or patch version) can be used to compile the contract. The version lock is a mechanism that allows the author to specify a minimum version of the Solidity compiler that must be used to compile the contract code. This is useful because it ensures that the contract will be compiled using a version of the compiler that is known to be compatible with the code.

```
pragma solidity ^0.6.11;  
pragma solidity >=0.6.0 <0.8.0;
```

Recommendation

The team is advised to lock the pragma to ensure the stability of the codebase. The locked pragma version ensures that the contract will not be deployed with an unexpected version. An unexpected version may produce vulnerabilities and undiscovered bugs. The compiler should be configured to the lowest version that provides all the required functionality for the codebase. As a result, the project will be compiled in a well-tested LTS (Long Term Support) environment.

Functions Analysis

Contract	Type	Bases		
	Function Name	Visibility	Mutability	Modifiers
aeERC20	Implementation	ERC20Permi tUpgradeabl e, TransferAnd CallToken		
		Public	✓	initializer
	_initialize	Internal	✓	initializer
TransferAndCal IToken	Implementation	ERC20Upgra deable, ITransferAnd Call		
	transferAndCall	Public	✓	-
	contractFallback	Private	✓	
	isContract	Private		
L2GatewayToke n	Implementation	aeERC20, IArbToken		
	_initialize	Internal	✓	
	bridgeMint	External	✓	onlyGateway
	bridgeBurn	External	✓	onlyGateway
ITransferAndCa ll	Interface	IERC20Upgr adeable		
	transferAndCall	External	✓	-

ITransferAndCallReceiver	Interface			
	onTokenTransfer	External	✓	-
ProxySetter	Interface			
	beacon	External		-
ClonableBeaconProxy	Implementation	BeaconProxy		
		Public	✓	BeaconProxy
BeaconProxyFactory	Implementation	ProxySetter		
	initialize	External	✓	-
	getSalt	Public		-
	createProxy	External	✓	-
	calculateExpectedAddress	Public		-
	calculateExpectedAddress	Public		-
BytesParser	Library			
	toUint8	Internal		
	toString	Internal		
TokenGateway	Implementation	ITokenGateway		
	_initialize	Internal	✓	
	isRouter	Internal		

	calculateL2TokenAddress	Public		-
ITokenGateway	Interface			
	outboundTransfer	External	Payable	-
	finalizeInboundTransfer	External	Payable	-
	calculateL2TokenAddress	External		-
	getOutboundCalldata	External		-
GatewayMessageHandler	Library			
	encodeToL2GatewayMsg	Internal		
	parseFromL1GatewayMsg	Internal		
	encodeFromL2GatewayMsg	Internal		
	parseToL1GatewayMsg	Internal		
	encodeFromRouterToGateway	Internal		
	parseFromRouterToGateway	Internal		
StandardArbERC20	Implementation	IArbToken, L2GatewayToken, Cloneable		
	bridgeInit	Public	✓	-
	decimals	Public		-
	name	Public		-
	symbol	Public		-
L2ArbitrumMessenger	Implementation			

	sendTxToL1	Internal	✓	
IArbToken	Interface			
	bridgeMint	External	✓	-
	bridgeBurn	External	✓	-
	l1Address	External		-
L2ERC20Gateway	Implementation	L2ArbitrumGateway		
	initialize	Public	✓	-
	calculateL2TokenAddress	Public		-
	cloneableProxyHash	Public		-
	getUserSalt	Public		-
	handleNoContract	Internal	✓	
L2ArbitrumGateway	Implementation	L2ArbitrumMessenger, TokenGateway		
	postUpgradeInit	External	✓	-
	_initialize	Internal	✓	
	createOutboundTx	Internal	✓	
	getOutboundCalldata	Public		-
	outboundTransfer	Public	Payable	-
	outboundTransfer	Public	Payable	-
	triggerWithdrawal	Internal	✓	
	outboundEscrowTransfer	Internal	✓	


	inboundEscrowTransfer	Internal	✓	
	finalizeInboundTransfer	External	Payable	onlyCounterpartGateway
	handleNoContract	Internal	✓	

Inheritance Graph



Flow Graph

The flow graph of the implementation can be found bellow:

 graph-original.png

Summary

ArbVault contract implements a bridge mechanism. This audit investigates security issues, business logic concerns and potential improvements. ArbVault is an interesting project that has a friendly and growing community. The Smart Contract analysis reported no compiler error or critical issues.

Disclaimer

The information provided in this report does not constitute investment, financial or trading advice and you should not treat any of the document's content as such. 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 Cyberscope's prior written consent. This report is not nor should be considered an "endorsement" or "disapproval" of any particular project or team. This report is not nor should be regarded as an indication of the economics or value of any "product" or "asset" created by any team or project that contracts Cyberscope to perform a security assessment. This document does not provide any warranty or guarantee regarding the absolute bug-free nature of the technology analyzed, nor do they provide any indication of the technologies proprietors' business, business model or legal compliance. This report should not be used in any way to make decisions around investment or involvement with any particular project. This report represents an extensive assessment process intending to help our customers increase the quality of their code while reducing the high level of risk presented by cryptographic tokens and blockchain technology.

Blockchain technology and cryptographic assets present a high level of ongoing risk. Cyberscope's position is that each company and individual are responsible for their own due diligence and continuous security. Cyberscope's goal is to help reduce the attack vectors and the high level of variance associated with utilizing new and consistently changing technologies and in no way claims any guarantee of security or functionality of the technology we agree to analyze. The assessment services provided by Cyberscope are subject to dependencies and are under continuing development. You agree that your access and/or use including but not limited to any services reports and materials will be at your sole risk on an as-is where-is and as-available basis. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. 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.

About Cyberscope

Cyberscope is a blockchain cybersecurity company that was founded with the vision to make web3.0 a safer place for investors and developers. Since its launch, it has worked with thousands of projects and is estimated to have secured tens of millions of investors' funds.

Cyberscope is one of the leading smart contract audit firms in the crypto space and has built a high-profile network of clients and partners.



The Cyberscope team

cyberscope.io