METATRUST

Security Assessment for

Main Cross

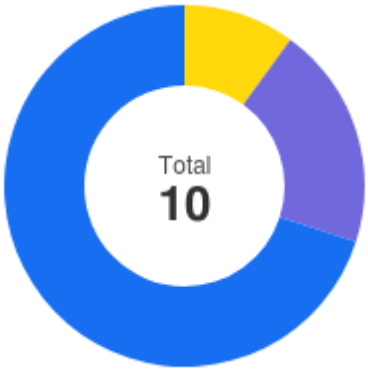
May 30, 2024

Executive Summary

Overview	
Project Name	Main Cross
Codebase URL	https://github.com/MainCross123/audit
Scan Engine	Security Analyzer
Scan Time	2024/05/30 10:44:02
Commit Id	87c306557963e6be29ff0fa86f0bab868649fcf8

Total	
Critical Issues	0
High risk Issues	0
Medium risk Issues	1
Low risk Issues	2
Informational Issues	7

Critical Issues	The issue can cause large economic losses, large-scale data disorder, loss of control of authority management, failure of key functions, or indirectly affect the correct operation of other smart contracts interacting with it.
High Risk Issues	The issue puts a large number of users' sensitive information at risk or is reasonably likely to lead to catastrophic impacts on clients' reputations or serious financial implications for clients and users.
Medium Risk Issues	The issue puts a subset of users' sensitive information at risk, would be detrimental to the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk Issues	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational Issue	The issue does not pose an immediate risk but is relevant to security best practices or Defence in Depth.



Critical Issues	0%	0
High risk Issues	0%	0
Medium risk Issues	10%	1
Low risk Issues	20%	2
Informational Issues	70%	7

Summary of Findings

MetaScan security assessment was performed on **May 30, 2024 10:44:02** on project **Main Cross** with the repository on branch **default branch**. The assessment was carried out by scanning the project's codebase using the scan engine **Security Analyzer**. There are in total **10** vulnerabilities / security risks discovered during the scanning session, among which **1** medium risk vulnerabilities, **2** low risk vulnerabilities, **7** informational issues.

ID	Description	Severity	Alleviation
MSA-001	Incorrect Usage of `transferFrom()` Function	Medium risk	Fixed
MSA-002	Centralized Risk	Low risk	Fixed
MSA-003	Potential Malicious Router Vulnerability During Deployment	Low risk	Fixed
MSA-004	Inline Modifier to Save Gas	Informational	Fixed
MSA-005	Missing Validation On Zero Address	Informational	Fixed
MSA-006	Lack or Improper Address Maliciousness Verification	Informational	Fixed
MSA-007	Use `calldata` instead of `memory` for read only arguments	Informational	Fixed
MSA-008	External Dependency	Informational	Fixed
MSA-009	Dangrous Max Approve to Router Address	Informational	Fixed
MSA-010	State Variables That Could Be Declared As Immutable	Informational	Fixed

Findings

Medium risk (1)

1. Incorrect Usage of `transferFrom()` Function

Medium risk | Security Analyzer

The ERC20 contract utilizes the `transferFrom()` function to handle incoming user payments. However, when the contract needs to transfer assets out, it's recommended to use the `transfer()` function. This is because some tokens may not support transfer from the user itself, potentially causing transactions to be blocked due to insufficient allowance even when the spender is the account owner.

Below is an example implementation of `ERC20.transferFrom()`:

```
function transferFrom(address from, address to, uint256 value) public virtual returns (bool) {
  address spender = _msgSender();
  _spendAllowance(from, spender, value);
  _transfer(from, to, value);
  return true;
}

function _spendAllowance(address owner, address spender, uint256 value) internal virtual {
  uint256 currentAllowance = allowance(owner, spender);
  if (currentAllowance != type(uint256).max) {
    if (currentAllowance < value) {
      revert ERC20InsufficientAllowance(spender, currentAllowance, value);
    }
    unchecked {
      _approve(owner, spender, currentAllowance - value, false);
    }
  }
}
```

File(s) Affected

AvaxInstantSwap #124-155

```
124   function swapFromUSDC(
125     address _tokenB,
126     uint256 _amountIn,
127     uint256 _minAmountOut,
128     address[] memory _path,
129     address _to,
130     bool _unwrappETH
131   ) public nonReentrant onlyExecutor {
132     if(_tokenB == usdc) {
133       IERC20(usdc).transferFrom(address(this), _to, _amountIn);
134       emit SwapFromUSDC(_to, _tokenB, _amountIn, _amountIn, block.timestamp);
135     } else {
136       checkAndApproveAll(usdc, address(lbRouter), _amountIn);
137
138       ILBRouter.Path memory pathQuote = getQuote(_path, _amountIn);
139       uint256 output = lbRouter
140         .swapExactTokensForTokensSupportingFeeOnTransferTokens(
141           _amountIn,
142           _minAmountOut, // Amount out min
143           pathQuote,
144           _unwrappETH ? address(this) : _to,
145           block.timestamp + 10 minutes
146       );
147
148       if (_unwrappETH) {
149         IWNATIVE(wethToken).withdraw(output);
150         payable(_to).transfer(output);
151       }
152
153       emit SwapFromUSDC(_to, _tokenB, _amountIn, output, block.timestamp);
154     }
155   }
```

Recommendation

We recommend using `transfer()/safeTransfer()` instead of `transferFrom()/safeTransferFrom()`.

Alleviation Fixed

Main Cross team fixed this in the 95d2871e2f0ef1cebad98cf0be1c6f07cdbe93 commit by using `transfer()`.

Low risk (2)

1. Centralized Risk

Low risk Security Analyzer

The contract has a centralized risk, which means that the contract is controlled by a single address. If the address is compromised, the contract will be compromised.

- `AvaxInstantSwap.sol`
  - `swapFromUSDC`: This function is only called by `executor` address, it can transfer USDC from this contract.
  - `setExecutor`: The owner can modify the `executor` address by this function.
  - `recoverStuckETH`: The owner can transfer native token from this contract.
  - `recoverStuckTokens`: The owner can transfer any ERC20 token from this contract.
- `OfficialInstantSwap`

- **`swapFromUSDC`**: This function is only called by **`executor`** address, it can swap token and send to specified address.
- **`directSendUSDC`**: This function is only called by **`executor`** address, it can transfer USDC token to specified address.
- **`setExecutor`**: The owner can modify the **`executor`** address by this function.
- **`recoverStuckETH`**: The owner can transfer native token from this contract.
- **`recoverStuckTokens`**: The owner can transfer any ERC20 token from this contract.

File(s) Affected

OfficialInstantSwap #155-173

```
155   function swapFromUSDC(
156     address _outputToken,
157     uint256 _amountIn,
158     uint256 _minAmountOut,
159     bool _useV2,
160     address[] memory _pathV2,
161     bytes memory _pathV3,
162     address to,
163     bool unwrapETH
164   ) public  nonReentrant onlyExecutor {
165     // USDC -> Token
166     uint256 outputAmount;
167     if(_useV2) {
168       outputAmount = v2Swap(_pathV2, _amountIn, _minAmountOut, to, unwrapETH);
169     } else {
170       outputAmount = v3Swap(USDC, _pathV3, _amountIn, _minAmountOut, to, unwrapETH);
171     }
172     emit SwapFromUSDC(to, _outputToken, _amountIn, outputAmount, block.timestamp);
173   }
```

OfficialInstantSwap #175-179

```
175   function directSendUSDC(address to, uint256 amount) external onlyExecutor {
176     require(to != address(0), "can not send address(0)");
177     IERC20(USDC).transfer(to, amount);
178     emit SwapFromUSDC(to, USDC, amount,amount, block.timestamp);
179   }
```

OfficialInstantSwap #234-237

```
234   function setExecutor(address _newExecutor) external onlyOwner {
235     emit ExecutorUpdated(executor, _newExecutor);
236     executor = _newExecutor;
237   }
```

OfficialInstantSwap #239-241

```
239   function recoverStuckETH(address payable _beneficiary) public onlyOwner {
240     _beneficiary.transfer(address(this).balance);
241   }
```

OfficialInstantSwap #243-246

```
243   function recoverStuckTokens(address _token) external onlyOwner {
244     uint256 amount = IERC20(_token).balanceOf(address(this));
245     IERC20(_token).safeTransfer(owner(), amount);
246   }
```

AvaxInstantSwap #178-181

```
178   function recoverStuckTokens(address _token) external onlyOwner {
179     uint256 amount = IERC20(_token).balanceOf(address(this));
180     IERC20(_token).safeTransfer(owner(), amount);
181   }
```

AvaxInstantSwap #124-155

```
124   function swapFromUSDC(
125     address _tokenB,
126     uint256 _amountIn,
127     uint256 _minAmountOut,
128     address[] memory _path,
129     address _to,
130     bool _unwrappETH
131   ) public nonReentrant onlyExecutor {
132     if(_tokenB == usdc) {
133       IERC20(usdc).transferFrom(address(this), _to, _amountIn);
134       emit SwapFromUSDC(_to, _tokenB, _amountIn, _amountIn, block.timestamp);
135     } else {
136       checkAndApproveAll(usdc, address(lbRouter), _amountIn);
137
138       ILBRouter.Path memory pathQuote = getQuote(_path, _amountIn);
139       uint256 output = lbRouter
140         .swapExactTokensForTokensSupportingFeeOnTransferTokens(
141           _amountIn,
142           _minAmountOut, // Amount out min
143           pathQuote,
144           _unwrappETH ? address(this) : _to,
145           block.timestamp + 10 minutes
146       );
147
148       if (_unwrappETH) {
149         IWNATIVE(wethToken).withdraw(output);
150         payable(_to).transfer(output);
151       }
152
153       emit SwapFromUSDC(_to, _tokenB, _amountIn, output, block.timestamp);
154     }
155   }
```

AvaxInstantSwap #157-160

```
157   function setExecutor(address _newExecutor) external onlyOwner {
158     emit ExecutorUpdated(executor, _newExecutor);
159     executor = _newExecutor;
160   }
```

AvaxInstantSwap #174-176

```
174   function recoverStuckETH(address payable _beneficiary) public onlyOwner {
175     _beneficiary.transfer(address(this).balance);
176   }
```

Recommendation

Consider implementing a decentralized governance mechanism or a multi-signature scheme that requires consensus among multiple parties before pausing or unpausing the contract. This can help mitigate the centralization risk associated with a single owner controlling critical contract functions. Alternatively, you can provide a clear justification for the centralization aspect and ensure that users are aware of the potential risks associated with a single point of control.

Alleviation Fixed

Main Cross team acknowledged this finding.

2. Potential Malicious Router Vulnerability During Deployment

Low risk Security Analyzer

The smart contract allows arbitrary setting of the router address at the deployment stage. This presents a security vulnerability, as a bad actor could deploy the contract with a malicious router address. Such a router could engage in harmful activities, such as redirecting funds or exploiting other vulnerabilities within the contract.

Example code demonstrating the vulnerability:

```
contract VulnerableContract {
    address public router;

  constructor(address _router) {
    router = _router;
  }

  // ... Additional contract code ...
}
```

This code snippet shows that the `_router` address is set during deployment without validation, which is a security concern.

File(s) Affected

AvaxInstantSwap #87-98

```
87    constructor(
88      address _lbRouter,
89      address _lbQuoter,
90      address _usdc,
91      address _executor
92    ) Ownable(msg.sender) {
93      lbRouter = ILBRouter(_lbRouter);
94      lbQuoter = IQuoter(_lbQuoter);
95      wethToken = address(lbRouter.getWNATIVE());
96      usdc = _usdc;
97      executor = _executor;
98    }
```

OfficialInstantSwap #141-153

```
141    constructor(
142      address _v3Router,
143      address _v2Router,
144      address _executor,
145      address _usdc,
146      address _weth
147    ) Ownable(msg.sender) {
148      v3Router = IV3SwapRouter(_v3Router);
149      v2Router = IUniswapV2Router02(_v2Router);
150      executor= _executor;
151      USDC = _usdc;
152      weth= _weth;
153    }
```

Recommendation

To mitigate this vulnerability, the following measures are recommended:

1. Restrict the router address to a list of pre-approved addresses during deployment.
2. Require a secure, multi-signature process for any post-deployment updates to the router address.
3. Establish a governance mechanism for changing the router address, ensuring transparency and security.

Alleviation Fixed

Main Cross team acknowledged this finding.

Informational (7)

1. Inline Modifier to Save Gas

Informational Security Analyzer

The modifier `onlyExecutor` that is only used once, change it to inline validator can save gas.

File(s) Affected

AvaxInstantSwap #82-85

```
82    modifier onlyExecutor() {
83      require(msg.sender == executor, "not executor");
84      _;
85    }
```

Recommendation

We recommend using inline modifier `onlyExecutor` to save gas.

Alleviation Fixed

Main Cross team acknowledged this finding.

2. Missing Validation On Zero Address

Informational Security Analyzer

Parameters of type address in your functions should be checked to ensure that they are not assigned the null address (address(0×0)). Failure to validate these parameters can lead to transaction reverts, wasted gas, the need for transaction resubmission, and may even require redeployment of contracts within the protocol in certain situations. Implement checks for address(0×0) to avoid these potential issues.

File(s) Affected

OfficialInstantSwap #155-173

```
155   function swapFromUSDC(
156     address _outputToken,
157     uint256 _amountIn,
158     uint256 _minAmountOut,
159     bool _useV2,
160     address[] memory _pathV2,
161     bytes memory _pathV3,
162     address to,
163     bool unwrapETH
164   ) public  nonReentrant onlyExecutor {
165     // USDC -> Token
166     uint256 outputAmount;
167     if(_useV2) {
168       outputAmount = v2Swap(_pathV2, _amountIn, _minAmountOut, to, unwrapETH);
169     } else {
170       outputAmount = v3Swap(USDC, _pathV3, _amountIn, _minAmountOut, to, unwrapETH);
171     }
172     emit SwapFromUSDC(to, _outputToken, _amountIn, outputAmount, block.timestamp);
173   }
```

AvaxInstantSwap #124-155

```
124   function swapFromUSDC(
125     address _tokenB,
126     uint256 _amountIn,
127     uint256 _minAmountOut,
128     address[] memory _path,
129     address _to,
130     bool _unwrappETH
131   ) public nonReentrant onlyExecutor {
132     if(_tokenB == usdc) {
133       IERC20(usdc).transferFrom(address(this), _to, _amountIn);
134       emit SwapFromUSDC(_to, _tokenB, _amountIn, _amountIn, block.timestamp);
135     } else {
136       checkAndApproveAll(usdc, address(lbRouter), _amountIn);
137
138       ILBRouter.Path memory pathQuote = getQuote(_path, _amountIn);
139       uint256 output = lbRouter
140         .swapExactTokensForTokensSupportingFeeOnTransferTokens(
141           _amountIn,
142           _minAmountOut, // Amount out min
143           pathQuote,
144           _unwrappETH ? address(this) : _to,
145           block.timestamp + 10 minutes
146       );
147
148       if (_unwrappETH) {
149         IWNATIVE(wethToken).withdraw(output);
150         payable(_to).transfer(output);
151       }
152
153       emit SwapFromUSDC(_to, _tokenB, _amountIn, output, block.timestamp);
154     }
155   }
```

Recommendation

We recommend validate the address are not assigned the null address (address(0×0)).

Alleviation Fixed

Main Cross team modified the code.

3. Lack or Improper Address Maliciousness Verification

Informational Security Analyzer

Lack or improper address maliciousness verification arises when the validation process fails to ensure the legitimacy and integrity of the address inputs.

Consequently, addresses that are untrusted or have been crafted with malicious intent can be processed by the contract. This could allow attackers to interact with the contract in unintended ways, possibly leading to the execution of unauthorized transactions or the disruption of contract logic.

File(s) Affected

OfficialInstantSwap #141-153

```
141   constructor(
142     address _v3Router,
143     address _v2Router,
144     address _executor,
145     address _usdc,
146     address _weth
147   ) Ownable(msg.sender) {
148     v3Router = IV3SwapRouter(_v3Router);
149     v2Router = IUniswapV2Router02(_v2Router);
150     executor= _executor;
151     USDC = _usdc;
152     weth= _weth;
153   }
```

OfficialInstantSwap #141-153

```
141   constructor(
142     address _v3Router,
143     address _v2Router,
144     address _executor,
145     address _usdc,
146     address _weth
147   ) Ownable(msg.sender) {
148     v3Router = IV3SwapRouter(_v3Router);
149     v2Router = IUniswapV2Router02(_v2Router);
150     executor= _executor;
151     USDC = _usdc;
152     weth= _weth;
153   }
```

AvaxInstantSwap #87-98

```
87    constructor(
88      address _lbRouter,
89      address _lbQuoter,
90      address _usdc,
91      address _executor
92    ) Ownable(msg.sender) {
93      lbRouter = ILBRouter(_lbRouter);
94      lbQuoter = IQuoter(_lbQuoter);
95      wethToken = address(lbRouter.getWNATIVE());
96      usdc = _usdc;
97      executor = _executor;
98    }
```

Recommendation

We recommend carefully setting the router address correctly in the constructor.

Alleviation Fixed

Main Cross team acknowledged this finding.

4. Use `calldata` instead of `memory` for read only arguments

Informational Security Analyzer

When a function with a `memory` array is called externally, the `abi.decode()` step has to use a for-loop to copy each index of the `calldata` to the `memory` index. Each iteration of this for-loop costs at least 60 gas (i.e. 60 * `<mem_array>.length`). Using calldata directly, obliviates the need for such a loop in the contract code and runtime execution. If the array is passed to an `internal` function which passes the array to another `internal` function where the array is modified and therefore `memory` is used in the `external` call, it's still more gas-efficient to use `calldata` when the external function uses modifiers, since the modifiers may prevent the `internal` functions from being called. `Structs` have the same overhead as an array of length one.

File(s) Affected

OfficialInstantSwap #188-212

```
188   function v2Swap(
189     address[] memory _path,
190     uint256 _amountIn,
191     uint256 _minAmountOut, // Slippage in base of 1000 meaning 10 is 1% and 1 is 0.1% where 1000 is 1
192     address to,
193     bool unwrapETH
194   ) internal returns (uint256) {
195     address tokenOut = _path[_path.length - 1];
196     checkAndApproveAll(_path[0], address(v2Router), _amountIn);
197     uint256 initial = IERC20(tokenOut).balanceOf(to);
198     v2Router.swapExactTokensForTokensSupportingFeeOnTransferTokens(
199         _amountIn,
200         _minAmountOut,
201         _path,
202         unwrapETH ? address(this) : to,
203         block.timestamp + 10 minutes
204     );
205     uint256 finalAmount = IERC20(tokenOut).balanceOf(to);
206     if (unwrapETH) { // Get ETH at the end
207         uint256 wethBalance = IERC20(weth).balanceOf(address(this));
208         IWETH(weth).withdraw(wethBalance);
209         payable(to).transfer(address(this).balance);
210     }
211     return finalAmount - initial;
212   }
```

OfficialInstantSwap #214-232

```
214   function v3Swap(
215     address _tokenIn,
216     bytes memory _path,
217     uint256 _amountIn,
218     uint256 _minAmountOut,
219     address to,
220     bool unwrapETH
221   ) internal returns (uint256 amountOutput) {
222     checkAndApproveAll(_tokenIn, address(v3Router), _amountIn);
223     IV3SwapRouter.ExactInputParams memory params = IV3SwapRouter.ExactInputParams(
224       _path, unwrapETH ? address(this) : to, _amountIn, _minAmountOut
225     );
226     amountOutput = v3Router.exactInput( params );
227     if (unwrapETH) { // Get ETH at the end
228         uint256 wethBalance = IERC20(weth).balanceOf(address(this));
229         IWETH(weth).withdraw(wethBalance);
230         payable(to).transfer(address(this).balance);
231     }
232   }
```

OfficialInstantSwap #155-173

```
155   function swapFromUSDC(
156     address _outputToken,
157     uint256 _amountIn,
158     uint256 _minAmountOut,
159     bool _useV2,
160     address[] memory _pathV2,
161     bytes memory _pathV3,
162     address to,
163     bool unwrapETH
164   ) public  nonReentrant onlyExecutor {
165     // USDC -> Token
166     uint256 outputAmount;
167     if(_useV2) {
168       outputAmount = v2Swap(_pathV2, _amountIn, _minAmountOut, to, unwrapETH);
169     } else {
170       outputAmount = v3Swap(USDC, _pathV3, _amountIn, _minAmountOut, to, unwrapETH);
171     }
172     emit SwapFromUSDC(to, _outputToken, _amountIn, outputAmount, block.timestamp);
173   }
```

AvaxInstantSwap #101-120

```
101   function getQuote(
102     address[] memory _path,
103     uint256 _amountIn
104   ) public view returns (ILBRouter.Path memory) {
105     // Use the quoter to find the best route for the swap
106     address[] memory path = new address[](_path.length);
107     IERC20[] memory pathToken = new IERC20[](_path.length);
108     for (uint256 i = 0; i < _path.length; i++) {
109       path[i] = _path[i];
110       pathToken[i] = IERC20(_path[i]);
111     }
112     IQuoter.Quote memory quote = lbQuoter.findBestPathFromAmountIn(
113       path,
114       uint128(_amountIn)
115     );
116     ILBRouter.Path memory myPath = ILBRouter.Path({
117       pairBinSteps: quote.binSteps,
118       versions: quote.versions,
119       tokenPath: pathToken
120     });
```

AvaxInstantSwap #124-155

```
124   function swapFromUSDC(
125     address _tokenB,
126     uint256 _amountIn,
127     uint256 _minAmountOut,
128     address[] memory _path,
129     address _to,
130     bool _unwrappETH
131   ) public nonReentrant onlyExecutor {
132     if(_tokenB == usdc) {
133       IERC20(usdc).transferFrom(address(this), _to, _amountIn);
134       emit SwapFromUSDC(_to, _tokenB, _amountIn, _amountIn, block.timestamp);
135     } else {
136       checkAndApproveAll(usdc, address(lbRouter), _amountIn);
137
138       ILBRouter.Path memory pathQuote = getQuote(_path, _amountIn);
139       uint256 output = lbRouter
140         .swapExactTokensForTokensSupportingFeeOnTransferTokens(
141           _amountIn,
142           _minAmountOut, // Amount out min
143           pathQuote,
144           _unwrappETH ? address(this) : _to,
145           block.timestamp + 10 minutes
146       );
147
148       if (_unwrappETH) {
149         IWNATIVE(wethToken).withdraw(output);
150         payable(_to).transfer(output);
151       }
152
153       emit SwapFromUSDC(_to, _tokenB, _amountIn, output, block.timestamp);
154     }
155   }
```

Recommendation

We recommend Use `calldata` instead of `memory` for such case.

Alleviation Fixed

Main Cross team modified the code.

5. External Dependency

Informational Security Analyzer

There are several functions rely on some external libraries and are not within the scope of this audit.

File(s) Affected

AvaxInstantSwap #101-122

```
101   function getQuote(
102     address[] memory _path,
103     uint256 _amountIn
104   ) public view returns (ILBRouter.Path memory) {
105     // Use the quoter to find the best route for the swap
106     address[] memory path = new address[](_path.length);
107     IERC20[] memory pathToken = new IERC20[](_path.length);
108     for (uint256 i = 0; i < _path.length; i++) {
109       path[i] = _path[i];
110       pathToken[i] = IERC20(_path[i]);
111     }
112     IQuoter.Quote memory quote = lbQuoter.findBestPathFromAmountIn(
113       path,
114       uint128(_amountIn)
115     );
116     ILBRouter.Path memory myPath = ILBRouter.Path({
117       pairBinSteps: quote.binSteps,
118       versions: quote.versions,
119       tokenPath: pathToken
120     });
121     return myPath;
122   }
```

AvaxInstantSwap #162-171

```
162   function checkAndApproveAll(
163     address _token,
164     address _target,
165     uint256 _amountToCheck
166   ) internal {
167     if (IERC20(_token).allowance(address(this), _target) < _amountToCheck) {
168       IERC20(_token).forceApprove(_target, 0);
169       IERC20(_token).forceApprove(_target, ~uint256(0));
170     }
171   }
```

OfficialInstantSwap #181-186

```
181   function checkAndApproveAll(address _token, address _target, uint256 _amountToCheck) internal {
182     if (IERC20(_token).allowance(address(this), _target) < _amountToCheck) {
183         IERC20(_token).forceApprove(_target, 0);
184         IERC20(_token).forceApprove(_target, ~uint256(0));
185     }
186   }
```

Recommendation

The team should fully understand their logic and keep an eye on possible version upgrades.

Alleviation Fixed

Main Cross team acknowledged this finding.

6. Dangrous Max Approve to Router Address

Informational Security Analyzer

Unlimited allowance is a dangerous act and the team should understand the implications of this approval and ensure that the destination address is correct and safe.

File(s) Affected

AvaxInstantSwap #162-171

```
162   function checkAndApproveAll(
163     address _token,
164     address _target,
165     uint256 _amountToCheck
166   ) internal {
167     if (IERC20(_token).allowance(address(this), _target) < _amountToCheck) {
168       IERC20(_token).forceApprove(_target, 0);
169       IERC20(_token).forceApprove(_target, ~uint256(0));
170     }
171   }
```

OfficialInstantSwap #181-186

```
181   function checkAndApproveAll(address _token, address _target, uint256 _amountToCheck) internal {
182     if (IERC20(_token).allowance(address(this), _target) < _amountToCheck) {
183         IERC20(_token).forceApprove(_target, 0);
184         IERC20(_token).forceApprove(_target, ~uint256(0));
185     }
186   }
```

Recommendation

We recommend approving a reasonable amount for the target address.

Alleviation Fixed

Main Cross team acknowledged this finding.

7. State Variables That Could Be Declared As Immutable

Informational Security Analyzer

In Solidity, the use of `constant` and `immutable` variables can lead to different gas cost implications. A `constant` variable has its expression copied and re-evaluated at every access point in the smart contract, which allows for local optimizations. On the other hand, an `immutable` variable is evaluated once during contract construction, and its value is then copied wherever it's accessed. However, it's important to note that for both `constant` and `immutable` variables, a fixed 32-byte storage is reserved, regardless of whether the actual value would require less space. This means that in certain cases, `constant` variables may incur lower gas costs than `immutable` ones. For more detailed information, refer to the Solidity documentation on constant and immutable state variables.

File(s) Affected

AvaxInstantSwap #67-68

```
67    address public wethToken;
68    address public usdc;
```

OfficialInstantSwap #121-122

```
121    address public USDC;
122    address public weth;
```

Recommendation

It is recommended to carefully evaluate the use of `constant` and `immutable` variables, considering their impact on gas costs. If a variable does not change after contract deployment, use `constant` to potentially save on gas.

For values that are set during contract construction and remain unchanged, `immutable` can be used; however, one must be aware of the 32-byte storage reservation which might not be gas-efficient for smaller data types.

Alleviation Fixed

Main Cross team acknowledged this finding.

Audit Scope

File	SHA256	File Path
AvaxInstantSwap	462fd506671fd11e20a340b35536e173	/AvaxInstantSwap
OfficialInstantSwap	2363328ba601cea8399dabe96cd7a46b	/OfficialInstantSwap

Disclaimer

This report is governed by the stipulations (including but not limited to service descriptions, confidentiality, disclaimers, and liability limitations) outlined in the Services Agreement, or as detailed in the scope of services and terms provided to you, the Customer or Company, within the context of the Agreement. The Company is permitted to use this report only as allowed under the terms of the Agreement. Without explicit written permission from MetaTrust, this report must not be shared, disclosed, referenced, or depended upon by any third parties, nor should copies be distributed to anyone other than the Company.

It is important to clarify that this report neither endorses nor disapproves any specific project or team. It should not be viewed as a reflection of the economic value or potential of any product or asset developed by teams or projects engaging MetaTrust for security evaluations. This report does not guarantee that the technology assessed is completely free of bugs, nor does it comment on the business practices, models, or legal compliance of the technology's creators.

This report is not intended to serve as investment advice or a tool for investment decisions related to any project. It represents a thorough assessment process aimed at enhancing code quality and mitigating risks inherent in cryptographic tokens and blockchain technology. Blockchain and cryptographic assets inherently carry ongoing risks. MetaTrust's role is to support companies and individuals in their security diligence and to reduce risks associated with the use of emerging and evolving technologies. However, MetaTrust does not guarantee the security or functionality of the technologies it evaluates.

MetaTrust's assessment services are contingent on various dependencies and are continuously evolving. Accessing or using these services, including reports and materials, is at your own risk, on an as-is and as-available basis. Cryptographic tokens are novel technologies with inherent technical risks and uncertainties. The assessment reports may contain inaccuracies, such as false positives or negatives, and unpredictable outcomes. The services may rely on multiple third-party layers.

All services, labels, assessment reports, work products, and other materials, or any results from their use, are provided "as is" and "as available," with all faults and defects, without any warranty. MetaTrust expressly disclaims all warranties, whether express, implied, statutory, or otherwise, including but not limited to warranties of merchantability, fitness for a particular purpose, title, non-infringement, and any warranties arising from course of dealing, usage, or trade practice. MetaTrust does not guarantee that the services, reports, or materials will meet specific requirements, be error-free, or be compatible with other software, systems, or services.

Neither MetaTrust nor its agents make any representations or warranties regarding the accuracy, reliability, or currency of any content provided through the services. MetaTrust is not liable for any content inaccuracies, personal injuries, property damages, or any loss resulting from the use of the services, reports, or materials.

Third-party materials are provided "as is," and any warranty concerning them is strictly between the Customer and the third-party owner or distributor. The services, reports, and materials are intended solely for the Customer and should not be relied upon by others or shared without MetaTrust's consent. No third party or representative thereof shall have any rights or claims against MetaTrust regarding these services, reports, or materials.

The provisions and warranties of MetaTrust in this agreement are exclusively for the Customer's benefit. No third party has any rights or claims against MetaTrust regarding these provisions or warranties. For clarity, the services, including any assessment reports or materials, should not be used as financial, tax, legal, regulatory, or other forms of advice.