
# Security Audit Report for Poly Contracts

**Date:** Oct 11, 2021

**Version:** 2.0

**Contact**: contact@blocksecteam.com

# Contents


**1 Introduction** **1**

1.1 About Target Contracts . . . 1
1.2 Disclaimer . . . 2
1.3 Procedure of Auditing . . . 2
1.3.1 Software Security . . . 2
1.3.2 DeFi Security . . . 3
1.3.3 NFT Security . . . 3
1.3.4 Additional Recommendation . . . 3
1.4 Security Model . . . 3

**2 Findings** **4**

2.1 Software Security . . . 4
2.1.1 The function `removeUnderlying` is not executed as expected . . . 4
2.1.2 The function `recoverEpochPk` is not executed successfully . . . 4
2.1.3 The events `UnlockEvent` and `LockEvent` may record wrong data . . . 5
2.2 Additional Recommendation . . . 7
2.2.1 Remove the repeated verification to save gas consumption . . . 7
2.2.2 Remove the redundant verification to save gas consumption . . . 8
2.2.3 Add the logic to update whitelist . . . 8
2.3 Others . . . 9

## Report Manifest

| Item | Description |
| --- | --- |
| Client | Poly Network |
| Target | Poly Contracts |

## Version History

| Version | Date | Description |
| --- | --- | --- |
| 1.0 | Sep 17, 2021 | First Release |
| 2.0 | Oct 11, 2021 | Second Release |

**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. The team won first place in the 2019 iDash competition (SGX Track). 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 smart contracts that are audited in this report include the following ones.

| Contract Name | Github URL |
|---|---|
| EthCrossChainData | https://github.com/polynetwork/eth-contracts/blob/master/contracts/core/cross_chain_manager/data/EthCrossChainData.sol |
| EthCrossChainManager | https://github.com/polynetwork/eth-contracts/blob/master/contracts/core/cross_chain_manager/logic/EthCrossChainManager.sol |
| UpgradableECCM | https://github.com/polynetwork/eth-contracts/blob/master/contracts/core/cross_chain_manager/upgrade/UpgradableECCM.sol |
| EthCrossChainManagerProxy | https://github.com/polynetwork/eth-contracts/blob/master/contracts/core/cross_chain_manager/upgrade/EthCrossChainManagerProxy.sol |
| ECCUtils | https://github.com/polynetwork/eth-contracts/blob/master/contracts/core/cross_chain_manager/libs/EthCrossChainUtils.sol |
| LockProxy | https://github.com/polynetwork/eth-contracts/blob/master/contracts/core/lock_proxy/LockProxy.sol |
| swapper | https://github.com/polynetwork/poly-swap/blob/dev/contracts/core/swapper/swapper_v3/ETH_swapper.sol |
| SwapProxy | https://github.com/polynetwork/poly-swap/blob/dev/contracts/core/lock_proxy/SwapProxy_v2.sol |

The commit hash values before the audit are shown in Table 1.1 on page 2.

We also checked the status of the fix based on the following commit hash value and pull request.

* SwapProxy_v3 [1]: 7b356a4f3b34991c41f6edcd170e506d0dc75716

* EthCrossChainUtils and EthCrossChainManager [2]

[1] https://github.com/polynetwork/poly-swap/blob/master/contracts/core/lock_proxy/SwapProxy_v3.sol

[2] https://github.com/polynetwork/eth-contracts/pull/22/files

Table 1.1: The commit hash of files before the audit

| Contract Name | Commit SHA |
|---|---|
| EthCrossChainData | 2b1cbe073e40a7bd26022d1cda9341b4780d07ee |
| EthCrossChainManager | 2b1cbe073e40a7bd26022d1cda9341b4780d07ee |
| UpgradableECCM | 2b1cbe073e40a7bd26022d1cda9341b4780d07ee |
| EthCrossChainManagerProxy | 2b1cbe073e40a7bd26022d1cda9341b4780d07ee |
| ECCUtils | 2b1cbe073e40a7bd26022d1cda9341b4780d07ee |
| LockProxy | 2b1cbe073e40a7bd26022d1cda9341b4780d07ee |
| swapper | 13003c356706e61ff662fb8ff2ed6af8eb0f7ec9 |
| SwapProxy | 13003c356706e61ff662fb8ff2ed6af8eb0f7ec9 |

## 1.2 Disclaimer

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. Besides, this report does not constitute personal investment advice or a personal recommendation.

## 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 [3] and Common Weakness Enumeration [4]. Accordingly, the severity measured in this report are classified into four categories: **High**, **Medium**, **Low** and **Undetermined**.

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

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

# Chapter 2 Findings

In total, we have identified **three potential issues** and three additional recommendations, as follows:

- Medium Risk: 1
- Low Risk: 3

| ID | Severity | Description | Category |
|---|---|---|---|
| 1 | Medium | The function `removeUnderlying` is not executed as expected | Software Security |
| 2 | Low | The function `recoverEpochPk` is not executed successfully | Software Security |
| 3 | Low | The events `UnlockEvent` and `LockEvent` may record wrong data | Software Security |
| 4 | - | Remove the repeated verification to save gas consumption | Recommendation |
| 5 | - | Remove the redundant verification to save gas consumption | Recommendation |
| 6 | - | Add the logic to update whitelist | Recommendation |

## 2.1 Software Security

### 2.1.1 The function `removeUnderlying` is not executed as expected

**Status** Confirmed and fixed.

**Description**

The `fromChainId` in the following code snippet should be replaced with `toChainId`. Otherwise, the function `removeUnderlying` can not work as expected.

```
224    address outAssetAddress = assetPoolMap[args.toPoolId][fromChainId][args.toAssetHash];
```

**Impact** The function `removeUnderlying` is not executed as expected.

**Suggestion** Replace `fromChainId` in line 224 of the contract SwapProxy with `toChainId`.

### 2.1.2 The function `recoverEpochPk` is not executed successfully

**Status** Confirmed and fixed.

**Description**

The function `recoverEpochPk` can be executed only when the contract ECCM's status (EthCrossChainManager) is paused, and it will call the function `putCurEpochConPubKeyBytes` of the contract ECCD (EthCrossChainData), as shown in the following code snippet(line 47).

```
44  function recoverEpochPk(bytes memory EpochPkBytes) whenPaused public {
45    require(unsetEpochPkBytes[EpochPkBytes],"Don't arbitrarily set");
46    unsetEpochPkBytes[EpochPkBytes] = false;
47    IEthCrossChainData(EthCrossChainDataAddress).putCurEpochConPubKeyBytes(EpochPkBytes);
48  }
```

However, the function `putCurEpochConPubKeyBytes` can be executed only when the contract ECCD's status is active, as shown in the following.

```
44  // Store Consensus book Keepers Public Key Bytes
45  function putCurEpochConPubKeyBytes(bytes memory curEpochPkBytes) public whenNotPaused onlyOwner
        returns (bool) {
46    ConKeepersPkBytes = curEpochPkBytes;
47    return true;
48  }
```

Note that, according to the function `pause` of ECCM, the status of ECCM and ECCD is always consistent, as shown in the following code snippet(line 21).

```
16  function pause() onlyOwner public returns (bool) {
17    if (!paused()) {
18      _pause();
19    }
20    IEthCrossChainData eccd = IEthCrossChainData(EthCrossChainDataAddress);
21    if (!eccd.paused()) {
22      require(eccd.pause(), "pause EthCrossChainData contract failed");
23    }
24    return true;
25  }
```

Therefore, the function `recoverEpochPk` will not be executed successfully.

**Impact** The function `recoverEpochPk` is useless, which wastes gas to store codes.

**Suggestion** Make the code logic consistent.

### 2.1.3 The events `UnlockEvent` and `LockEvent` may record wrong data

**Status** Confirmed and fixed.

**Description**

In the following functions `addUnderlying`, `outAssetAddress` equals to `ISwap(poolAddres- s).lp_token()` (line 172), and the first parameter of the event `LockEvent` should records `outAssetAddress`. However, the code in line 187 records `poolAddress`.

```
160   function addUnderlying(bytes memory argsBs, bytes memory fromContractAddr, uint64 fromChainId)
          onlyThis external returns (bool) {
161     SwapArgs memory args = _deserializeSwapArgs(argsBs);
162
163     require(fromContractAddr.length != 0, "from contract address cannot be empty");
164     require(Utils.equalStorage(swapperHashMap[fromChainId], fromContractAddr), "from swapper
            contract address error!");
165
166     address poolAddress = poolAddressMap[args.toPoolId];
167     require(poolAddress != address(0), "pool do not exsit");
168
169     address inAssetAddress = assetPoolMap[args.toPoolId][fromChainId][args.fromAssetHash];
170     require(inAssetAddress != address(0), "inAssetHash cannot be empty");
171
172     address outAssetAddress = ISwap(poolAddress).lp_token();
173
```

```
174         require(args.toAddress.length != 0, "toAddress cannot be empty");
175
176         bytes memory toAssetHash = assetHashMap[outAssetAddress][args.toChainId];
177         require(toAssetHash.length != 0, "empty illegal toAssetHash");
178
179
180         uint outAmount = _addInPool(poolAddress, inAssetAddress, args.amount, args.minOut);
181
182         require(_crossChain(args.toChainId, args.toAddress, toAssetHash, outAmount));
183
184
185         emit UnlockEvent(inAssetAddress, address(this), args.amount);
186         emit AddLiquidityEvent(args.toPoolId, inAssetAddress, args.amount, poolAddress, outAmount,
                args.toChainId, toAssetHash, args.toAddress);
187         emit LockEvent(poolAddress, address(this), args.toChainId, toAssetHash, args.toAddress,
                outAmount);
188
189         return true;
190     }
```

Similar with above, the code in line 234 of the following code snippet records `poolAddress` rather than `ISwap(poolAddress).lp_token()`.

```
212     function removeUnderlying(bytes memory argsBs, bytes memory fromContractAddr, uint64
            fromChainId) onlyThis external returns (bool) {
213         SwapArgs memory args = _deserializeSwapArgs(argsBs);
214
215         require(fromContractAddr.length != 0, "from contract address cannot be empty");
216         require(Utils.equalStorage(swapperHashMap[fromChainId], fromContractAddr), "from swapper
                contract address error!");
217
218         address poolAddress = poolAddressMap[args.toPoolId];
219         require(poolAddress != address(0), "pool do not exsit");
220         require(Utils.equalStorage(assetHashMap[ISwap(poolAddress).lp_token()][fromChainId], args.
                fromAssetHash), "from Asset do not match pool token address");
221
222         // address outAssetAddress = assetPoolMap[args.toPoolId][args.toChainId][args.toAssetHash];
223         // NOT fromChainId !!!!!!!!!!
224         address outAssetAddress = assetPoolMap[args.toPoolId][fromChainId][args.toAssetHash];
225         require(outAssetAddress != address(0), "target asset do not exsit");
226
227         require(args.toAddress.length != 0, "toAddress cannot be empty");
228
229         uint outAmount = _removeInPool(poolAddress, args.amount, outAssetAddress, args.minOut);
230
231         require(_crossChain(args.toChainId, args.toAddress, args.toAssetHash, outAmount));
232
233
234         emit UnlockEvent(poolAddress, address(this), args.amount);
235         emit RemoveLiquidityEvent(args.toPoolId, poolAddress, args.amount, outAssetAddress,
                outAmount, args.toChainId, args.toAssetHash, args.toAddress);
236         emit LockEvent(outAssetAddress, address(this), args.toChainId, args.toAssetHash, args.
                toAddress, outAmount);
```

```
237
238         return true;
239     }
```

**Impact** The events `LockEvent` and `UnlockEvent` may records wrong information.

**Suggestion** Replace `poolAddress` with `ISwap(poolAddress).lp_token()` in line 187 and line 234 of the contract SwapProxy.

## 2.2 Additional Recommendation

### 2.2.1 Remove the repeated verification to save gas consumption

**Status** Acknowledged.

**Description**

The internal functions `_pull` and `_checkoutFee` do the same validation: `require(msg.v- alue == amount, "insufficient ether");`, as shown in the following code snippet (line 271 and line 280).

```
268 // take input
269     function _pull(address fromAsset, uint amount) internal {
270         if (fromAsset == address(0)) {
271             require(msg.value == amount, "insufficient ether");
272         } else {
273             IERC20(fromAsset).safeTransferFrom(msg.sender, address(this), amount);
274         }
275     }
276
277     // take fee in the form of ether
278     function _checkoutFee(address fromAsset, uint amount, uint fee) internal view returns (uint) {
279         if (fromAsset == address(0)) {
280             require(msg.value == amount, "insufficient ether");
281             require(amount > fee, "amount less than fee");
282             return amount.sub(fee);
283         } else {
284             require(msg.value == fee, "insufficient ether");
285             return amount;
286         }
287     }
```

Furthermore, the two functions `_pull` and `_checkoutFee` are always executed consecutively, as shown in the following code snippets. Therefore, the validation will be executed repeatedly every time.

```
216     function remove_liquidity(address fromAssetHash, uint64 toPoolId, uint64 toChainId, bytes
            memory toAssetHash, bytes memory toAddress, uint amount, uint minOutAmount, uint fee, uint
             id) public payable nonReentrant whenNotPaused returns (bool) {
217         _pull(fromAssetHash, amount);
218
219         amount = _checkoutFee(fromAssetHash, amount, fee);
220
221         _push(fromAssetHash, amount);
222     ......
223 }
```

```
176    function add_liquidity(address fromAssetHash, uint64 toPoolId, uint64 toChainId, bytes memory
           toAddress, uint amount, uint minOutAmount, uint fee, uint id) public payable nonReentrant
           whenNotPaused returns (bool) {
177        _pull(fromAssetHash, amount);
178
179        amount = _checkoutFee(fromAssetHash, amount, fee);
180
181        _push(fromAssetHash, amount);
182        ......
183    }
```

```
135    function swap(address fromAssetHash, uint64 toPoolId, uint64 toChainId, bytes memory
           toAssetHash, bytes memory toAddress, uint amount, uint minOutAmount, uint fee, uint id)
           public payable nonReentrant whenNotPaused returns (bool) {
136        _pull(fromAssetHash, amount);
137
138        amount = _checkoutFee(fromAssetHash, amount, fee);
139
140        _push(fromAssetHash, amount);
141        ......
142    }
```

**Impact** Waste gas.

**Suggestion** Combine the three internal functions `_pull`, `_checkoutFee`, and `_push`, and remove the redundant validation.

### 2.2.2 Remove the redundant verification to save gas consumption

**Status** Acknowledged.

**Description**

The following code snippet comes from the function `remove_liquidity` of the contract swapper. The code in line 223 makes `fromAssetHash` impossible to be `address(0)`, and the code in line 225 requires `fromAddressHash` equal to `poolTokenMap[toPoolId]`. Therefore, the code in line 224 that requires `poolTokenMap[toPoolId]` not equal to `address(0)` is a redundant validation.

```
223    fromAssetHash = fromAssetHash==address(0) ? WETH : fromAssetHash ;
224    require(poolTokenMap[toPoolId] != address(0), "given pool do not exsit");
225    require(poolTokenMap[toPoolId] == fromAssetHash,"input token is not pool LP token");
```

**Impact** Waste gas.

**Suggestion** Remove the code in line 224.

### 2.2.3 Add the logic to update whitelist

**Status** Confirmed and fixed.

**Description**

The `constructor` of the contract ECCM initiate three whitelists: `fromContractWhiteList`, `toContractWhiteList`, and `methodWhiteList`. The three whitelists are extremely important, which can be leveraged to block illegal cross-chain transactions. However, there is no update mechanism for the three whitelists, which is

not flexible. Particularly, if Poly Network want to add new cross-chain services, they must **re-deploy** the ECCM contract.

```
24  constructor(
25        address _eccd,
26        uint64 _chainId,
27        address[] memory fromContractWhiteList,
28        address[] memory toContractWhiteList,
29        bytes[] memory methodWhiteList,
30        bytes memory curEpochPkBytes
31    ) UpgradableECCM(_eccd,_chainId) public {
32        for (uint i=0;i<fromContractWhiteList.length;i++) {
33            whiteListFromContract[fromContractWhiteList[i]] = true;
34        }
35        for (uint i=0;i<toContractWhiteList.length;i++) {
36            whiteListToContract[toContractWhiteList[i]] = true;
37        }
38        for (uint i=0;i<methodWhiteList.length;i++) {
39            whiteListMethod[methodWhiteList[i]] = true;
40        }
41        unsetEpochPkBytes[curEpochPkBytes] = true;
42    }
```

**Impact** No flexible mechanism to extend cross-chain services.

**Suggestion** Add the logic to update whitelists in the contract ECCM.

## 2.3 Others

We note that the cross-chain DEX service is relying on the *SwapProxy and the pool module*. These contracts are deployed on a permissioned blockchain called the Curve chain. This audit does not include the Curve chain itself and the pool module. We suggest the project owner can apply common security practice to protect these modules.