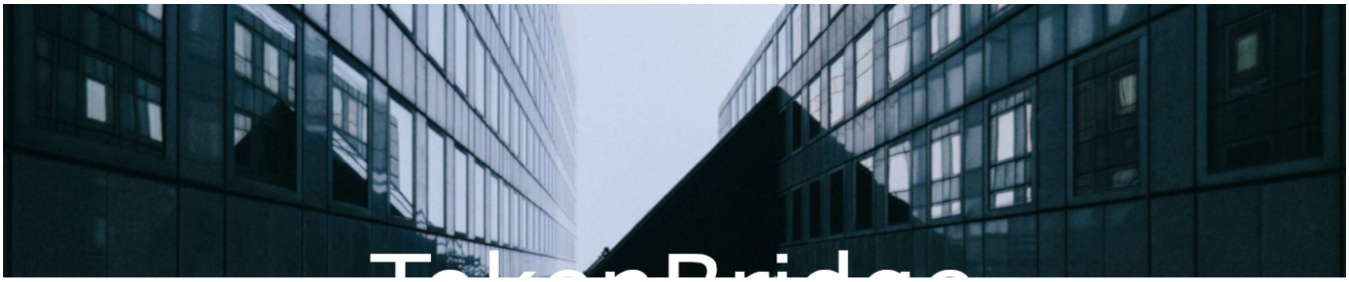
Published in SmartDec Cybersecurity Blog

Boris Nikashin Follow

Jul 31, 2019 · 14 min read · Listen

TokenBridge (by POA Network) Smart Contracts Security Analysis



SmartDec & POA Network

In this report, we consider the security of the TokenBridge project from POA Network. Our task is to find and describe security issues in the smart contracts of the platform.

Disclaimer

The audit does not give any warranties on the security of the code. One audit cannot be considered enough. We always recommend proceeding with several independent

In this report, we considered the security of POA Network smart contracts. We performed our audit according to the procedure described below.

The audit showed no critical issues. However, a number of medium and low severity issues were found. They do not endanger project security.

All of the issues were addressed, som[illegible]d in the latest version of the code.

General recommendations

The contracts code is of good code quality. Nevertheless, we recommend covering the code with tests and developing deploy scripts.

Procedure

In our audit, we consider the following crucial features of the smart contract code:

1. Whether the code is secure
2. Whether the code corresponds to the documentation (including whitepaper)
3. Whether the code meets best practices in efficient use of gas, code readability, etc.

We perform our audit according to the following procedure:

Automated analysis

- we scan project's smart contracts with our own Solidity static code analyzer SmartCheck
- we scan project's smart contracts with several publicly available automated Solidity analysis tools such as Remix, Slither, and Solhint
- we manually verify (reject or confirm) all the issues found by tools

Manual audit

- we manually analyze smart contracts for security vulnerabilities
- we check smart contracts logic and compare it with the one described in the documentation

Report

- we report all the issues found to the developer during the audit process
- we check the issues fixed by the developer
- we reflect all the gathered information in the report

Checked vulnerabilities

We have scanned TokenBridge smart contracts for commonly known and more specific vulnerabilities. Here are some of the commonly known vulnerabilities that we considered (the full list includes them but is not limited to them):

- Reentrancy
- Front Running
- DoS with (Unexpected) revert
- DoS with Block Gas Limit
- Gas Limit and Loops
- Locked money
- Integer overflow/underflow
- Unchecked external call
- ERC20 Standard violation
- Authentication with tx.origin
- Unsafe use of timestamp
- Using blockhash for randomness
- Balance equality
- Unsafe transfer of ether
- Fallback abuse


- Private modifier
- Compiler version not fixed
- Style guide violation
- Unsafe type deduction
- Implicit visibility level
- Using delete for arrays
- Byte array
- Incorrect use of assert/require
- Using deprecated constructions

Project overview

Project description

In our analysis we consider POA Network specification (”README.md and REWARD_MANAGEMENT.md” in the repo) and smart contracts’ code (version on commit bbb97a63c900f03a902d0e82358abac3b294e4d9).

The latest version of the code

After the initial audit, some fixes were applied and the code was updated to the latest version (commit b66a678648ea89b6441382c7a7adefb3b0b02667).

Project architecture

For the audit, we were provided with the truffle project. The project is an npm package and includes tests.

The files successfully compile with `truffle compile` command (with some warnings).

The files successfully pass all the tests with 100% coverage.

The total LOC of the audited Solidity sources is 2297.


issues lead to actual bugs or vulnerabilities, are described in the next section.

Manual analysis

The contracts were completely manually analyzed, their logic was checked and compared with the one described in the documentation. Besides, the results of the automated analysis were manually verified. All the confirmed issues are described below.

Critical issues

Critical issues seriously endanger smart contracts security. We highly recommend fixing them.

The audit showed no critical issues.

Medium severity issues

Medium issues can influence smart contracts operation in the current implementation. We highly recommend addressing them.

Code logic

Since ERC20 Standard `transfer()` function is used for placing tokens to the Foreign bridge in `erc20-to-erc20` and `erc20-to-native` modes, there is no convenient way to limit the number of tokens that a user can send. Thus, there could be a situation when a user sends too many tokens to the Foreign bridge. In this case, **OverdrawManagement** contract functionality is used. When validators try to validate the number of tokens that exceeds the limit, `setTxAboveLimits()` function is called. After that, the bridge owner has to call `fixAssetsAboveLimits()` function, which emits `UserRequestForSignature()` event. Then, the validators should sign the returning transaction and return the tokens back to the user on the Foreign side. But in this case, the validators will receive the fee.

In case of `erc20-to-native` mode, **FeeManagerErcToNative** fee manager, the fee will be charged from the Home bridge contract address. In other cases (in POSDAO environment), it will lead to the bridge imbalance: amount of coins/tokens on the


minted to validators. At the same time, the full requested amount of tokens will be unlocked when confirmations are passed to the foreign side.

> *This issue has been fixed in the pull request #218.*

Overpowered owner

The owner has the following powers:

- The owner has an ability to set crucial parameters at own will.

> *Comment from the developers:*
>
> *"It is assumed that there are three different roles: a) an account that could upgrade the contracts, b) an account that could manage parameters of the bridge contracts, c) an account that could manage parameters of the validators contracts.*
>
> *The first account is indeed has overpowered role since is able to modify the logic of the bridge contract at all. Others could introduce a limited impact. For each of three accounts it is assumed that they are a multisig wallets. And owners of the wallets are responsible for authorizing the bridge configuration actions."*

- Users heavily rely on the owner and validators. The owner manually adds validators and then ensures that validators are the same in both Home and Foreign networks.

> *Comment from the developers: "It is the responsibility of the token bridge owner to provide bridge operations. If a current set bridge validators sabotage the bridge operations, the token bridge owner could modify the set of validators (remove old ones and one at least one new) in order to unlock funds."*

- Staking contract can reallocate funds from any address to its own by calling `stake()` function. The owner is able to set any address as a staking contract by calling `setStakingContract()` function.

> *Comment from the developers: "The bridge contracts relies on the staking contract functionality. The staking contract is not the part of the token bridge code base."*


contract by calling `setBlockRewardContract()` function.

In the current implementation, the system depends heavily on the owner and validators of the contracts. In this case, there are scenarios that may lead to undesirable consequences for investors, e.g. if the owner's private keys become compromised. Thus, we recommend designing contracts in a trustless manner.

Low severity issues

Low severity issues can influence smart contracts operation in future versions of code. We recommend taking them into account.

Defines a return type but never explicitly returns a value

The following functions defines a return type, but never initialize the return values.

- **BasicHomeBridge.sol**, `onExecuteAffirmation()` function

> *This issue has been fixed in the pull request #203.*

- **HomeBridgeErcToErc.sol**, `_rewardableInitialize()` function

> *This issue has been fixed in the pull request #221.*

- **HomeBridgeErcToErc.sol**, `_initialize()` function

> *This issue has been fixed in the pull request #221.*

We recommend adding return statements or removing return types from function declaration.

Fallback function requires too much gas

Fallback function of the following contracts contract contains too much logic.

- **HomeBridgeErcToNative**
- **ClassicHomeBridgeNativeToErc**
- **HomeBridgeNativeToErc**

Upgrade code to Solidity 0.5.x

The code uses solidity compiler version `0.4.24`.

We recommend migrating code to a new version of compiler (`0.5.10`) since it contains several important changes. In order to update contracts to compiler version `0.5.10` the developers should follow Solidity documentation.

> *Comment from the developers: "There is an issue created for this. But it will not be addressed as part of the audit since requires additional resources for implementation and testing."*

Gas limit and loops

The following loops traverse through arrays of variable length:

1. **ERC677BridgeTokenRewardable.sol**, line 38:

```
for (uint256 i = 0; i < _receivers.length; i++)
```

> *Comment from the developers: "The only place where the `mintToken()` method is called is the following one. It is defined here that `receivers` array size is `stakers.length / DELEGATORS_ALIQUOTE`. Number of stakers cannot be greater than 3000. `DELEGATORS_ALIQUOTE` is 2. So, the maximum size of the `receivers` array is 1500."*

2. **RewardableValidators.sol**, line 24:

```
for (uint256 i = 0; i < _initialValidators.length; i++)
```

> *This issue has been fixed in the pull request #239.*

3. **ValidatorsFeeManager.sol**, line 41:

> *Comment from the developers: "There is no simple way to say what should be a limit since the method refers on the functionality implemented in a particular fee manager."*

Therefore, if there are too many items in these arrays, the execution of the corresponding functions will fail due to an out-of-gas exception.

In these cases, we recommend separating the calls into several transactions.

Extra gas consumption

Excessive gas consumption in a loop

`_receivers.length` variable (**ERC677BridgeTokenRewardable.sol** file, line 38) is read from the storage on every iteration of the corresponding loop. Reading from local memory requires significantly less gas compared to reading from the storage.

Thus, we recommend placing this variable into local memory in order to reduce gas consumption.

> *This issue has been fixed in the pull request #242.*

revert() vs require()

`revert()` is used in several places:

1. **BaseBridgeValidators.sol**, lines 46–48:

```
if (nextValidator == address(0) ) { revert(); }
```

2. **BaseBridgeValidators.sol**, lines 77–79:

```
if (next == F_ADDR || next == address(0) ) { revert(); }
```

3. **Message.sol**, line 109–111:


We recommend using `require(condition);` instead of `if (!condition) revert();` to improve code readability and transparency.

> *The issues have been fixed in the pull request #235.*

Assert violation

The execution of `random()` function from **BaseFeeManager** contract can fail with an exception if `_count` argument is equal to zero.

```
function random(uint256 _count) public view returns(uint256) {
    return uint256(blockhash(block.number.sub(1))) % _count;
}
```

As a result, all the provided gas will be spent.

We recommend checking that considered variable is not zero in order to reduce gas costs.

> *This issue has been fixed in the pull request #246.*

Redundant fallback function

The payment rejection fallback in **HomeBridgeErcToErc** contract is redundant. Before Solidity 0.4.0, payment rejection was done manually:

```
function () { revert(); }
```

Starting from Solidity 0.4.0, contracts without a fallback function automatically revert payments, therefore, the fallback function in this contract redundant.

> *This issue has been fixed in the pull request #224.*

Redundant code

The project has the following redundant code issues:

This issue has been fixed in the pull request #198.

- `claimTokens()` function from **ForeignBridgeErcToNative** contract has `onlyOwnerOfProxy()` modifier. However, this function calls `super()` function from **BasicBridge** contract, which also has the same modifier. Hence, this modifier is called twice. We recommend removing it from `claimTokens()` function.

This issue has been fixed in the pull request #198.

- In fallback function from **HomeBridgeErcToNative** contract, `totalBurntCoins()` function is called twice: at lines 24 and 34. We recommend avoiding multiple reads of storage variables in the same function in order to decrease execution costs.

This issue has been fixed in the pull request #203.

- `fireEventOnTokenTransfer()` function from **HomeBridgeErcToNative** is never used.

This issue has been fixed in the pull request #203.

- `messages()` function from **BasicHomeBridge** contract is redundant since it is called only from `message()` function and its functionality can be moved there.

This issue has been fixed in the pull request #203.

- `signatures()` function from **BasicHomeBridge** contract is redundant since it is called only from `signature()` function and its functionality can be moved there.

This issue has been fixed in the pull request #203.

- In **BasicHomeBridge** contract, the following functions have an empty body: `onExecuteAffirmation()`, `onSignaturesCollected()`, `affirmationWithinLimits()`, `onFailedAffirmation()`. We recommend not implementing these functions and using `;` instead in order to make these smart contracts abstract.

The issues have been fixed in the pull request #203

```
recipient := and(mload(add(message, 20)),
0xFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF)
```

`and` operation is redundant since this check is done automatically for variables with `address` type.

> *This issue has been fixed in the pull request #227.*

- **HomeBridgeNativeToErc** contract inherits from **RewardableHomeBridgeNativeToErc**. This means that if **HomeBridgeNativeToErc** contract is non-rewardable, it has lots of unused functions.

> *Comment from the developers: "It is done intentionally since will allow enable gathering fees from the bridge operation just by setting the Fee Manager contract without necessity to upgrade entire bridge contract."*

- `onSignatureCollected()` functions are identical in all bridges, except one. In **HomeBridgeNativeToErc** contract, this function has the following check at line 132:

```
if (fee != 0) {
```

If the case where `fee` is equal to zero is invalid, it is not clear why there are no such checks in other bridge contracts. We recommend clarifying the possible cases for the function in the documentation.

> *Comment from the developers: "There are two modes for the native-to-erc20 bridge to work with the Fee Manager: 1. The fee collected on each side of the bridge: fees from home-to-foreign transfers are collected on the foreign bridge contract, fees from foreign-to-home transfers are collected on the home side. 2. The fee collected on the home side only for both direction. For the mode 1, `calculateFee` invoked by `onSignatureCollected()` returns zero. So, fee will be zero as well. Such bridge mode so far is specific for the native-to-erc20*

- `_rewardableInitialize()` and `_initialize()` functions in **HomeBridgeErcToErc** contract always return `false`. We recommend removing return values in order to improve code readability.

> *This issue has been fixed in the pull request #221.*

- The following checks are redundant:

ERC677BridgeTokenRewardable, line 18:

```
_blockRewardContract != address(0)
```

ERC677BridgeTokenRewardable, line 23:

```
_stakingContract != address(0)
```

Later in these lines, it is checked whether addresses are contracts. The fact that the checked address is a contract implies it is not zero address since no contracts can be deployed at `address(0)`. Thus, these checks are redundant.

> *The issues have been fixed in the pull request #225.*

- The following checks at line 119, **HomeBridgeErcToNative.sol** are redundant:

```
_blockReward != address(0) && isContract(_blockReward)
```

Since there is an external call in this line, the function will revert if `_blockReward` is not a contract or if the called function is not implemented.

> *This issue has been fixed in the pull request #225.*

- `setInitialize()` and `setFixedAssets()` functions from **HomeBridgeErcToNative**

We highly recommend removing redundant code in order to improve code readability and transparency and decrease the cost of deployment and execution.

Code style

There are several code style issues in the project:

- Some getter functions have `get` prefix in their names while others do not have it. Moreover, `POSDAO` is a postfix in all the contracts except one — **POSDAOHomeBridgeErcToErc**. We recommend sticking to the same style when choosing names for functions or contracts.

> *This issue has been partially addressed in the pull request #226.*

- In BaseBridgeValidators contract, line 104, `"deployedAtBlock"` string should be used inside `abi.encodePacked()` function since it is suggested by Solidity documentation.

> *This issue has been fixed in the pull request #203.*

- `affirmationWithinLimits()` function is the same in all Home bridges. We recommend avoiding code duplication in order to improve code readability and reduce the chance of making a mistake when upgrading the code.

> *This issue has been fixed in the pull request #223.*

We recommend fixing these issues in order to improve code readability.

Code logic

There are places in the project that contain code logic issues:

- **FeeManagerNativeToErc** contract is used in both networks. However, `onSignatureFeeDistribution()` function is not used in Home network and `onAffirmationFeeDistribution()` function is not used in Foreign network.

We recommend splitting FeeManagerNativeToErc contract into two.

- In **ERC677BridgeToken** contract, the logic of `transfer()` function is modified, however, `transferFrom()` function remains unchanged.

We recommend changing the logic of `transferFrom()` function in order to make them more consistent.

> *This issue has been fixed in the pull request #220.*

Misleading comment

Line 89 in **Message.sol** contains the following comment:

```
// message is always 84 length
```

However, the next line has the following code:

```
string memory msgLength = "104";
```

We recommend fixing this comment in order to avoid confusion.

> *This issue has been fixed in the pull request #204.*

Missing input validation

There are functions where input values are not validated correctly:

- In `hasEnoughValidSignatures()` function, there is no check that `_vs`, `_rs`, and `_ss` arrays are of equal lengths.

> *Comment from the developers: "It is not necessary to check that arrays have the same length since it will be handled during attempt to access to the corresponding elements in the loop and the call will be reverted. It will save gas for the rational validators actions and still be safe enough from security point of view."*

- In **BaseFeeManager** contract, `setHomeFee()` and `setForeignFee()` functions should


We recommend implementing the mentioned checks.

> *This issue has been fixed in the pull request #209.*

Wrong import of OpenZeppelin library

In the current implementation, **OpenZeppelin** files are added to the repo. This violates **OpenZeppelin**'s MIT license, which requires the license and copyright notice to be included if its code is used. Moreover, it is more difficult and error-prone to update the code manually added to the repo.

We highly recommend using npm in order to guarantee that original **OpenZeppelin** contracts are used with no modifications. This also allows for any bug-fixes to be easily integrated into the codebase.

> *This issue has been fixed in the pull request #222.*

Lack of documentation

In the documentation, it is not described how the bridge should work in `POSDAO` environment.

We recommend amending the documentation.

> *This issue has been fixed in the pull request #202.*

Missing return value of ERC20 tokens

`claimTokens()` function from **ERC677BridgeToken** does not work with tokens that do not return `true` on `transfer()` function calls. However, some older ERC20 tokens do not provide any return value when functions such as `transferFrom()` are called.

We recommend using **SafeERC20** contract from **OpenZeppelin** library.

> *This issue has been fixed in the pull request #213.*

Private modifier


Contrary to a popular misconception, the private modifier does not make a variable invisible. Miners have access to all contracts' code and data. Developers must account for the lack of privacy in Ethereum.

> *This issue has been fixed in the pull request #198.*

Notes

Gas limit and loops

The loop at **BridgeValidators.sol**, line 22 traverses through an array of variable length:

```
for (uint256 i = 0; i < _initialValidators.length; i++)
```

`_initialValidators` array is passed as `initialize()` function parameter. Therefore, if there are too many items in `_initialValidators` array, the execution of `initialize()` function will fail due to an out-of-gas exception.

We recommend keeping this problem in mind since in the current implementation function call cannot be split into several calls.

> *This issue has been fixed in the pull request #239.*

Prefer external to public visibility level

Many functions in the code have `public` visibility when they could have `external` visibility. We recommend using the latter one since it indicates that the functions are not called internally.

> This article was created by SmartDec, a security team specialized in static code analysis, decompilation and secure development.
>
> Feel free to use SmartCheck, our smart contract security tool for Solidity and Vyper, and follow us on Medium, Telegram and Twitter. We are also available for smart

More from SmartDec Cybersecurity Blog

Follow

Security tutorials, tools, and ideas

Read more from SmartDec Cybersecurity Blog