peppersec

PUBLIC REPORT

POA Network Bridge

Smart Contract Security Audit

Clarity is a rare commodity. That is why for the convenience of both the client and the reader, we have introduced a system of marking vulnerabilities and security issues we discover during our security audits.

Let's start with an ideal case. If an identified security imperfection bears no impact on the security of our client, we mark it with the No issue label.

The fixed security issues get the ✓ Fixed label that informs those reading our public report that the flaws in question should no longer be worried about.

In case a client addresses an issue in another way (e.g., by updating the information in the technical papers and specification) we put a nice Addressed tag right in front of it.

If an issue is planned to be addressed in the future, it gets the Acknowledged tag, and a client clearly sees what is yet to be done.

Although the issues marker with ✓ Fixed and Acknowledged are no threat, we still list them to provide the most detailed and up-to-date information for the client and the reader.

TABLE OF CONTENTS

01. INTRODUCTION

Source code

Audit methodology

Auditors

02. SUMMARY

Discovered vulnerabilities

03. ERC677BRIDGE-TOKEN

Unnecessary functionality ✓ Fixed

Contract does not prevent accidental token transferring ✓ Fixed

04. HOMEBRID-GENATIVETOERC

No message checking Acknowledged

Possible Denial of Service ✓ Fixed

05. BRIDGEVALIDA-TORS

Redundant checks and code ✓ Fixed

Function does not generate the event ✓ Fixed

06. GENERAL ISSUES

Double spending ✓ Fixed

Possible Validators/RequiredSignatures desync No issue

APPENDIX 1. TERMINOLOGY

Severity

Source code

Object	Location
POA Network bridge. Branch refactor_v1	#2bf70c7e9fd42968aec2dc352017618907834401

Audit methodology

The code of a smart contract has been automatically and manually scanned for known vulnerabilities and logic errors that may cause security threats. The conformity of requirements (e.g., White Paper) and practical implementation has been reviewed as well. More information on the used methodology can be found here.

Auditors

Alexey Pertsev. **PepperSec**.

Discovered vulnerabilities

Below, you can find a table with all the discovered bugs and security issues listed.

Vulnerability description	Severity	See paragraph
Double spending	Major	**General issues**
Contract does not prevent an accidental token transferring	Medium	**ERC677BridgeToken**
Possible Denial of Service		**HomeBridgeNativeToErc**
Unnecessary functionality	Minor	**ERC677BridgeToken**
Redundant checks and code		**BridgeValidators**
Function does not generate the event		**BridgeValidators**
No message checking		**HomeBridgeNativeToErc**
Possible Validators/ RequiredSignatures desync	Note	**General issues**

Unnecessary functionality

► **Severity:** Minor

A new version of the openzeppelin `Ownable` contract has the `renounceOwnership` function. For more information, see **here**.

So this function is inherited by your `ERC677BridgeToken` unnoticeably. The function **seems superfluous**. It is worth considering whether the function is necessary for the project?

Recommendations:

1. Consider rewriting `renounceOwnership` to empty implementation (as it has been done with **finishMinting**).

Status:

► Fixed – **PR48**

Contract does not prevent accidental token transferring

► **Severity:** Medium

The `transfer` method does not prevent token transferring to `ForeignBridgeNativeToErc`, but in this case the `UserRequestForAffirmation(userAddr, value)` event will not be fired. `claimTokens` will surely come in handy to reveal accidentally sent tokens. Nonetheless, preventing this kind of sending is still the surefire solution to the problem.

Recomendations:

1. Implement the same **isContract** check and the `onTokenTransfer` call at the `transfer` method. You can use `_to.call(abi.encodeWithSignature(«onTokenTransfer(address,uint256,bytes)», ...))` instead of `_to.onTokenTransfer(...)` to prevent the `revert` if a token receiver does not have the method in question.

See an example **here**.

Status:

► Fixed – **PR50**

No message checking

► **Severity: Minor**

The **BasicHomeBridge.sol#L43** contract does not check the `message` entities within the `submitSignature` function.

Scenario

If an attacker spoofs `recipient` or `value` by a MITM attack between the Ethereum node and the Validator bridge software at the time of event grabbing, then Validator will sign and send the `message` with spoofed args. Due to the lack of `message` checking, the contract will accept this signature without any notification.

Note: MITM is not the only way to spoof those values, but it has been used since it is possible right now for `bridge-nodejs`.

See configuration **here**.

Recommendations:

1. Save `msg.sender`, `msg.value`, and `TxHash` (optional) at the moment of receiving Ether (the **fallback** function) and check it later (the `submitSignature` function).
2. Use https only.

Team's answer:

In Solidity, there is no way to access `txhash`, so an identifier could only be the hash of `msg.sender`, `value`, `now`.

Status:

After discussing all pros and cons of message checking, the team decided that using `https` would be enough.

Possible Denial of Service

► **Severity: Medium**

The *Bridge* contacts do not handle exception situations at the time of token/ether transferring. **HomeBridgeNativeToErc.sol#L45** - if a recipient is a contract that cannot receive ether (e.g., `revert()` at the fallback function), `HomeBridgeNativeToErc` will throw an exception, and the whole TX will be reverted as well. In the case of Bridge Validators software being not ready to handle that, there is a risk of DoS.

Recommendations:

Countermeasures can be implemented at both a smart contract and the Bridge software side.

1. Bridge Software can “revert” the whole process by generating an opposite transfer.
2. The contract may implement transfer via `selfdestruct` of a child contract. See more information on implementation **here.**

Status:

Fixed – **PR51**

Redundant checks and code

► **Severity: Minor**

BridgeValidators.sol#L30. The check is redundant because of **this** and **this.**

Recomendations:

1. Change require(...); to assert(...); or remove it.

Status:

Fixed – PR48

Function does not generate the event

► **Severity: Minor**

The **initialize** function sets the `requiredSignatures` var but does not generate the `RequiredSignaturesChanged(requiredSignatures);` event. However, `setRequiredSignatures` does. The whole situation begs the question, whether it is just not necessary or missed? In addition to it, `OwnershipTransferred` and `ValidatorAdded` are fired.

Recommendations:

1. Consider adding the `RequiredSignaturesChanged` emitting.

Status:

Fixed – **PR48**

Double spending

► **Severity: Major**

Since the signatures for the NativeToERC and ERC20ToERC20 (home to foreign) transfers are not different in any way, they can be used to cause double spending.

Recommendations:

1. Implement different message for the NativeToERC and ERC20ToERC20 transfers. Consider adding `tokenAddress` to the ERC20ToERC20 transfers `message` in order to avoid double spending and changing token provider contract.

Status:

Fixed – **PR57**

Possible Validators/RequiredSignatures desync

► **Severity: None**

Because there is no on-chain way to sync the “Home” and “Foreign” sides in terms of “current Validators list” and/or `RequiredSignatures`, Validator Software should notify of any desync and stop Bridge trade if it happens.

Team’s update:

For this purposes, we use **bridge-monitor** to alert if the required signatures are not matched.

Status:

The issue is irrelevant.

Appendix 1. Terminology

Severity

Severity is the category that described the magnitude of an issue.

		Severity		
Impact	Major	Medium	Major	Critical
	Medium	Minor	Medium	Major
	Minor	None	Minor	Medium
		Minor	Medium	Major
		Likelihood		

Minor

Minor issues are generally subjective in their nature or potentially associated with the topics like “best practices” or “readability”. As a rule, minor issues do not indicate an actual problem or bug in the code.

The maintainers should use their own judgment as to whether addressing these issues will improve the codebase.

Medium

Medium issues are generally objective in their nature but do not represent any actual bugs or security problems.

These issues should be addressed unless there is an apparent reason not to.

Major

Major issues are things like bugs or vulnerabilities. These issues may be unexploitable directly or may require a certain condition to arise to be exploited.

If unaddressed, these issues are likely to cause problems with the operation of the contract or lead to situations which make the system exploitable.

Critical

Critical issues are directly exploitable bugs or security vulnerabilities.

If unaddressed, these issues are likely or guaranteed to cause major problems and ultimately a full failure in the operations of the contract.

About Us

Worried about the security of your project? You're on the right way! The second step is to find a team of seasoned cybersecurity experts who will make it impenetrable. And you've just come to the right place.

PepperSec is a group of whitehat hackers seasoned by many-year experience and have a deep understanding of the modern Internet technologies. We're ready to battle for the security of your project.

LET'S KEEP IN TOUCH


peppersec.com


hello@peppersec.com


Github