EvryHub

Smart Contract Audit Report
Prepared for EvryNet

Date Issued: Nov 9, 2021
Project ID: AUDIT2021018-3
Version: v2.0
Confidentiality Level: Public

inspeX
CYBERSECURITY PROFESSIONAL SERVICE

Report Information

Project ID	AUDIT2021018-3
Version	v2.0
Client	EvryNet
Project	EvryHub
Auditor(s)	Weerawat Pawanawiwat Suvicha Buakhom Patipon Suwanbol Peeraphut Punsuwan
Author	Peeraphut Punsuwan Patipon Suwanbol
Reviewer	Suvicha Buakhom
Confidentiality Level	Public

Version History

Version	Date	Description	Author(s)
2.0	Nov 9, 2021	Update issue id	Peeraphut Punsuwan
1.0	Oct 29, 2021	Full report	Peeraphut Punsuwan Patipon Suwanbol

Contact Information

Company	Inspex
Phone	(+66) 90 888 7186
Telegram	t.me/inspexco
Email	audit@inspex.co

Table of Contents

1. Executive Summary

As requested by EvryNet, Inspex team conducted an audit to verify the security posture of the EvryHub smart contracts between Sep 20, 2021 and Sep 30, 2021. During the audit, Inspex team examined all smart contracts and the overall operation within the scope to understand the overview of EvryHub smart contracts. Static code analysis, dynamic analysis, and manual review were done in conjunction to identify smart contract vulnerabilities together with technical & business logic flaws that may be exposed to the potential risk of the platform and the ecosystem. Practical recommendations are provided according to each vulnerability found and should be followed to remediate the issue.

1.1. Audit Result

In the initial audit, Inspex found 3 high, 1 medium 1 very low, and 4 info-severity issues. With the mitigation solutions and fixes confirmed by the project team. Therefore, Inspex trusts that EvryHub smart contracts have sufficient protections to be safe for public use. However, in the long run, Inspex suggests resolving all issues found in this report.

1.2. Disclaimer

This security audit is not produced to supplant any other type of assessment and does not guarantee the discovery of all security vulnerabilities within the scope of the assessment. However, we warrant that this audit is conducted with goodwill, professional approach, and competence. Since an assessment from one single party cannot be confirmed to cover all possible issues within the smart contract(s), Inspex suggests conducting multiple independent assessments to minimize the risks. Lastly, nothing contained in this audit report should be considered as investment advice.

2. Project Overview

2.1. Project Introduction

EvryNet is an intelligent financial services platform providing infrastructure that enables developers and businesses to build an unlimited number of Centralised/Decentralised Finance (CeDeFi) applications, interoperable with many of the world's leading blockchains for "evryone".

EvryHub is built to bridge assets between different blockchains, allowing users to transfer their assets to and from another blockchain easily.

Scope Information:

Project Name	EvryHub
Website	https://evrynet.io/
Smart Contract Type	Ethereum Smart Contract
Chain	Ethereum, Binance Smart Chain
Programming Language	Solidity

Audit Information:

Audit Method	Whitebox
Audit Date	Sep 20, 2021 - Sep 30, 2021
Reassessment Date	Oct 28, 2021

The audit method can be categorized into two types depending on the assessment targets provided:

1. **Whitebox**: The complete source code of the smart contracts are provided for the assessment.
2. **Blackbox**: Only the bytecodes of the smart contracts are provided for the assessment.

2.2. Scope

The following smart contracts were audited and reassessed by Inspex in detail:

BSC

Contract	Location (URL)
Migration	https://public.inspex.co/audit/EvryNet_EvryHub/bsc_contracts/Migrations.sol
BridgeBank	https://public.inspex.co/audit/EvryNet_EvryHub/bsc_contracts/BridgeBank/BridgeBank.sol
BridgeBankPausable	https://public.inspex.co/audit/EvryNet_EvryHub/bsc_contracts/BridgeBank/BridgeBankPausable.sol
BridgeToken	https://public.inspex.co/audit/EvryNet_EvryHub/bsc_contracts/BridgeBank/BridgeToken.sol
BscBank	https://public.inspex.co/audit/EvryNet_EvryHub/bsc_contracts/BridgeBank/BscBank.sol

Ethereum

Contract	Location (URL)
Migration	https://public.inspex.co/audit/EvryNet_EvryHub/ethereum_contracts/Migrations.sol
BridgeBank	https://public.inspex.co/audit/EvryNet_EvryHub/ethereum_contracts/BridgeBank/BridgeBank.sol
BridgeBankPausable	https://public.inspex.co/audit/EvryNet_EvryHub/ethereum_contracts/BridgeBank/BridgeBankPausable.sol
BridgeToken	https://public.inspex.co/audit/EvryNet_EvryHub/ethereum_contracts/BridgeBank/BridgeToken.sol
EthBank	https://public.inspex.co/audit/EvryNet_EvryHub/ethereum_contracts/BridgeBank/EthBank.sol

EvryNet

Contract	Location (URL)
Migration	https://public.inspex.co/audit/EvryNet_EvryHub/evrynet_contracts/Migrations.sol
BridgeBank	https://public.inspex.co/audit/EvryNet_EvryHub/evrynet_contracts/BridgeBank/BridgeBank.sol

BridgeBankPausable	https://public.inspex.co/audit/EvryNet_EvryHub/evrynet_contracts/BridgeBank/BridgeBankPausable.sol
BridgeToken	https://public.inspex.co/audit/EvryNet_EvryHub/evrynet_contracts/BridgeBank/BridgeToken.sol
EthBank	https://public.inspex.co/audit/EvryNet_EvryHub/evrynet_contracts/BridgeBank/EthBank.sol

Please note that the EvryHub smart contracts were initially in EvryNet's private repository. The files have been uploaded to Inspex's storage for public access.

EvryHub Reassessment: (commit: 86601724f42cfdd5abaf6cd179f29eb0d4971469)

BSC

Contract	Location (URL)
Migration	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/bsc_contracts/contracts/Migrations.sol
BridgeBank	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/bsc_contracts/contracts/BridgeBank/BridgeBank.sol
BridgeBankPausable	https://github.com/Evry-Finance/evry-hub/blob/86601724f42/smart_contracts/bsc_contracts/contracts/BridgeBank/BridgeBankPausable.sol
BridgeToken	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/bsc_contracts/contracts/BridgeBank/BridgeToken.sol
BscBank	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/bsc_contracts/contracts/BridgeBank/BscBank.sol
Validator	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/bsc_contracts/contracts/BridgeBank/Validator.sol

Ethereum

Contract	Location (URL)
Migration	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/ethereum_contracts/contracts/Migrations.sol
BridgeBank	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/ethereum_contracts/contracts/BridgeBank/BridgeBank.sol
BridgeBankPausable	https://github.com/Evry-Finance/evry-hub/blob/86601724f42/smart_contracts/ethereum_contracts/contracts/BridgeBank/BridgeBankPausable.sol
BridgeToken	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/eth

	ereum_contracts/contracts/BridgeBank/BridgeToken.sol
EthBank	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/ethereum_contracts/contracts/BridgeBank/EthBank.sol
Validator	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/ethereum_contracts/contracts/BridgeBank/Validator.sol

EvryNet

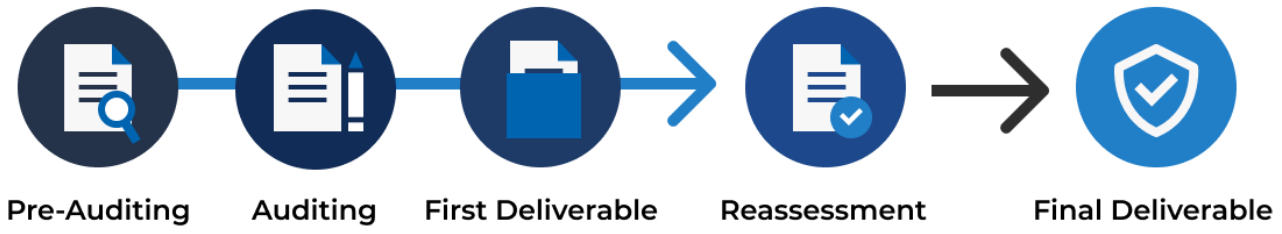
Contract	Location (URL)
Migration	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/evrynet_contracts/contracts/Migrations.sol
BridgeBank	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/evrynet_contracts/contracts/BridgeBank/BridgeBank.sol
BridgeBankPausable	https://github.com/Evry-Finance/evry-hub/blob/86601724f42/smart_contracts/evrynet_contracts/contracts/BridgeBank/BridgeBankPausable.sol
BridgeToken	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/evrynet_contracts/contracts/BridgeBank/BridgeToken.sol
EthBank	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/evrynet_contracts/contracts/BridgeBank/EthBank.sol
Validator	https://github.com/Evry-Finance/evry-hub/blob/86601724f4/smart_contracts/evrynet_contracts/contracts/BridgeBank/Validator.sol

The assessment scope covers only the in-scope smart contracts and the smart contracts that they inherit from.

3. Methodology

Inspex conducts the following procedure to enhance the security level of our clients' smart contracts:

1. **Pre-Auditing**: Getting to understand the overall operations of the related smart contracts, checking for readiness, and preparing for the auditing
2. **Auditing**: Inspecting the smart contracts using automated analysis tools and manual analysis by a team of professionals
3. **First Deliverable and Consulting**: Delivering a preliminary report on the findings with suggestions on how to remediate those issues and providing consultation
4. **Reassessment**: Verifying the status of the issues and whether there are any other complications in the fixes applied
5. **Final Deliverable**: Providing a full report with the detailed status of each issue



3.1. Test Categories

Inspex smart contract auditing methodology consists of both automated testing with scanning tools and manual testing by experienced testers. We have categorized the tests into 3 categories as follows:

1. **General Smart Contract Vulnerability (General)** - Smart contracts are analyzed automatically using static code analysis tools for general smart contract coding bugs, which are then verified manually to remove all false positives generated.
2. **Advanced Smart Contract Vulnerability (Advanced)** - The workflow, logic, and the actual behavior of the smart contracts are manually analyzed in-depth to determine any flaws that can cause technical or business damage to the smart contracts or the users of the smart contracts.
3. **Smart Contract Best Practice (Best Practice)** - The code of smart contracts is then analyzed from the development perspective, providing suggestions to improve the overall code quality using standardized best practices.

3.2. Audit Items

The following audit items were checked during the auditing activity.

General
Reentrancy Attack
Integer Overflows and Underflows
Unchecked Return Values for Low-Level Calls
Bad Randomness
Transaction Ordering Dependence
Time Manipulation
Short Address Attack
Outdated Compiler Version
Use of Known Vulnerable Component
Deprecated Solidity Features
Use of Deprecated Component
Loop with High Gas Consumption
Unauthorized Self-destruct
Redundant Fallback Function
Insufficient Logging for Privileged Functions
Invoking of Unreliable Smart Contract
Advanced
Business Logic Flaw
Ownership Takeover
Broken Access Control
Broken Authentication
Use of Upgradable Contract Design
Improper Kill-Switch Mechanism

Improper Front-end Integration
Insecure Smart Contract Initiation
Denial of Service
Improper Oracle Usage
Memory Corruption
Best Practice
Use of Variadic Byte Array
Implicit Compiler Version
Implicit Visibility Level
Implicit Type Inference
Function Declaration Inconsistency
Token API Violation
Best Practices Violation

3.3. Risk Rating

OWASP Risk Rating Methodology[1] is used to determine the severity of each issue with the following criteria:

- **Likelihood**: a measure of how likely this vulnerability is to be uncovered and exploited by an attacker.
- **Impact**: a measure of the damage caused by a successful attack

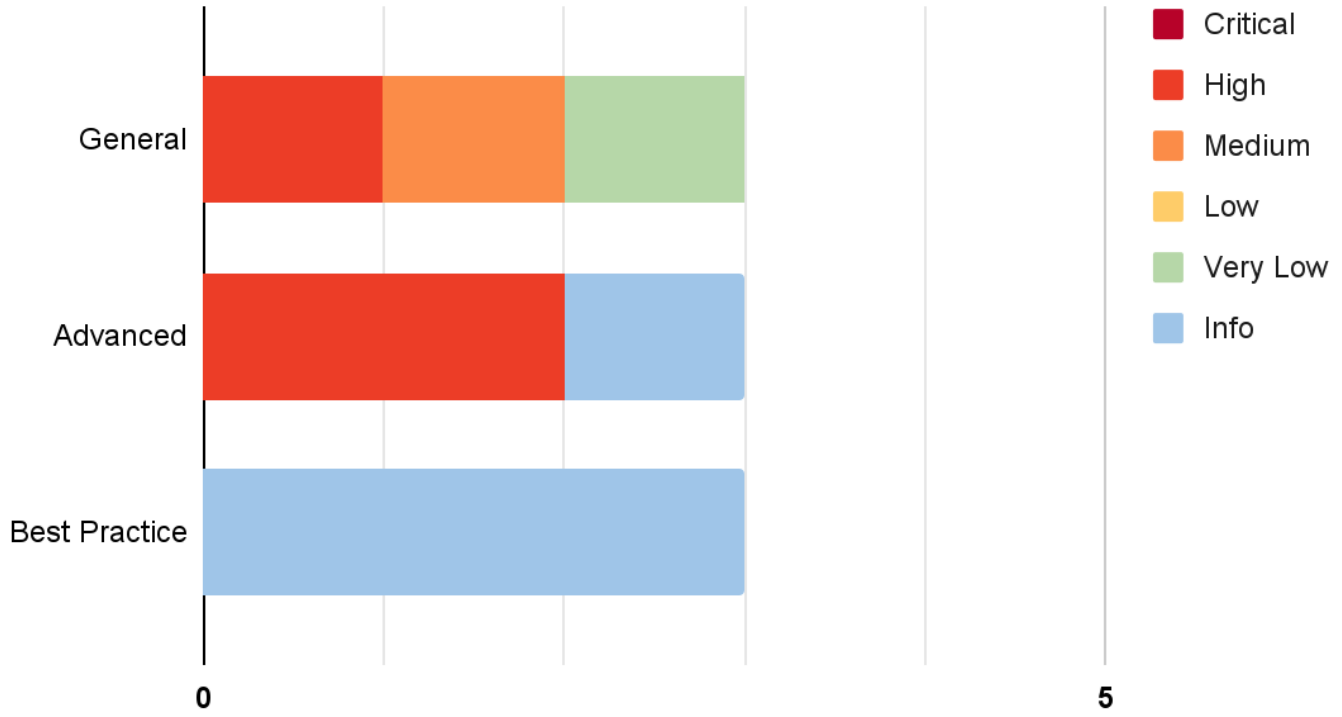
Both likelihood and impact can be categorized into three levels: **Low**, **Medium**, and **High**.

Severity is the overall risk of the issue. It can be categorized into five levels: **Very Low**, **Low**, **Medium**, **High**, and **Critical**. It is calculated from the combination of likelihood and impact factors using the matrix below. The severity of findings with no likelihood or impact would be categorized as **Info**.

Impact \ Likelihood	Low	Medium	High
Low	**Very Low**	**Low**	**Medium**
Medium	**Low**	**Medium**	**High**
High	**Medium**	**High**	**Critical**

4. Summary of Findings

From the assessments, Inspex has found 9 issues in three categories. The following chart shows the number of the issues categorized into three categories: **General**, **Advanced**, and **Best Practice**.



The statuses of the issues are defined as follows:

Status	Description
Resolved	The issue has been resolved and has no further complications.
Resolved *	The issue has been resolved with mitigations and clarifications. For the clarification or mitigation detail, please refer to Chapter 5.
Acknowledged	The issue's risk has been acknowledged and accepted.
No Security Impact	The best practice recommendation has been acknowledged.

The information and status of each issue can be found in the following table:

ID	Title	Category	Severity	Status
IDX-001	Use of Upgradable Contract Design	Advanced	**High**	Resolved *
IDX-002	Centralized Control of State Variable	General	**High**	Resolved *
IDX-003	Improper Design for Operator Privilege	Advanced	**High**	Resolved
IDX-004	Reentrancy Attack	General	**Medium**	Resolved
IDX-005	Insufficient Logging for Privileged Functions	General	**Very Low**	Resolved
IDX-006	Whitelisting of Token Address	Advanced	**Info**	No Security Impact
IDX-007	Modifier Without Effect	Best Practice	**Info**	Resolved
IDX-008	Improper Function Visibility	Best Practice	**Info**	No Security Impact
IDX-009	Inexplicit Solidity Compiler Version	Best Practice	**Info**	No Security Impact

* The mitigations or clarifications by EvryNet can be found in Chapter 5.

5. Detailed Findings Information

5.1. Use of Upgradable Contract Design

ID	IDX-001
Target	BridgeBank (BSC, Ethereum, EvryNet)
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity:** High **Impact:** High The logic of affected contract can be arbitrarily changed. This allows the proxy owner to perform malicious actions e.g., stealing the user funds anytime they want. **Likelihood:** Medium This action can be performed by the proxy owner without any restriction.
Status	Resolved * EvryNet team has confirmed that timelock will be used to own the contracts. This will allow the users to monitor the changes to the smart contracts and act accordingly before the changes are applied. The smart contracts are not yet deployed at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock before using the platform.

5.1.1. Description

Smart contracts are designed to be used as agreements that cannot be changed forever. When a smart contract is upgraded, the agreement can be changed from what was previously agreed upon.

As the `BridgeBank` contract can be deployed through a proxy contract, the logic of it could be modified by the owner anytime, making the smart contract untrustworthy.

5.1.2. Remediation

Inspex suggests deploying the contract without the proxy pattern or any solution that can make smart contract upgradable.

However, if the upgradability is needed, Inspex suggests mitigating this issue by implementing a timelock mechanism with a sufficient length of time to delay the changes. This allows the platform users to monitor the timelock and be notified of the potential changes being done on the smart contracts.

5.2. Centralized Control of State Variable

ID	IDX-002
Target	BridgeBank (BSC, Ethereum, EvryNet)
Category	General Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The controlling authorities can change the critical state variables to gain high privilege and get through restrictions, allowing the owner to withdraw funds from the contract. Thus, it is unfair to the other users. **Likelihood: Medium** There is nothing to restrict the changes from being done; however, these actions can only be performed by the contract owner.
Status	Resolved * EvryNet team has confirmed that timelock will be used to own the contracts. This will allow the users to monitor the changes to the smart contracts and act accordingly before the changes are applied. The smart contracts are not yet deployed at the time of the reassessment. Therefore, Inspex suggests the platform users to confirm the usage of the timelock before using the platform.

5.2.1. Description

Critical state variables can be updated any time by the controlling authorities. Changes in these variables can cause impacts to the users, so the users should accept or be notified before these changes are effective.

However, there is no constraint to prevent the authorities from modifying these variables without notifying the users.

The controllable privileged state update functions are as follows:

File	Contract	Function	Modifier
BridgeBank.sol (L:45)	BridgeBank	changeOperator()	isOwner
BridgeBank.sol (L:257)	BridgeBank	setEmergencyWithdrawDelayTime()	isOwner

5.2.2. Remediation

In the ideal case, the critical state variables should not be modifiable to keep the integrity of the smart contract. However, if modifications are needed, Inspex suggests limiting the use of these functions via the following options:

- Implementing a community-run governance to control the use of these functions
- Using a `Timelock` contract to delay the changes for a sufficient amount of time, e.g., 24 hours

Please note that the `pause()` and `unpause()` functions are under the same `isOwner` access control modifier, and another role should be implemented for these functions if they are needed to be executed promptly for emergency situations.

5.3. Improper Design for Operator Privilege

ID	IDX-003
Target	BridgeBank (BSC, Ethereum, EvryNet)
Category	Advanced Smart Contract Vulnerability
CWE	CWE-284: Improper Access Control
Risk	**Severity: High** **Impact: High** The holder of the `operator` address's private key can execute critical functions freely, allowing that address to transfer funds from the contract to any address. **Likelihood: Medium** These functions can only be called by the authorized party, but there is nothing to restrict malicious actions from being done.
Status	**Resolved** EvryNet team has resolved this issue in commit 86601724f42cfdd5abaf6cd179f29eb0d4971469 by implementing a multi-signature validation. For the operator to execute the functions, 2/3 of the trusted validators must sign the message to ensure that the execution is properly validated by the majority of the trusted validators.

5.3.1. Description

In the `BridgeBank` contract, the `operator` can use critical functions, such as the `unlock()` function, to transfer any token to any recipient. These privileged functions have very high impact as all funds in the contract are solely controlled by the `operator` address. From the current design, the `operator` is a wallet address with the private key controlled by EvryNet's off-chain relayer. Therefore, the holder of the `operator` address's private key can freely drain the funds from the `BridgeBank` contract.

The `operator` can execute the following critical functions:

File	Contract	Function
BridgeBank.sol (L:149)	BridgeBank	unlock()
BridgeBank.sol (L:177)	BridgeBank	emergencyWithdraw()
BridgeBank.sol (L:208)	BridgeBank	refund()

5.3.2. Remediation

Inspex suggests implementing a consensus mechanism to verify that the transaction is approved by the majority of trusted validators before transferring the token out from the `BridgeBank` contract to prevent unverified actions from being done.

For example, this can be done by storing a list of trusted validators on the smart contract. The trusted validators then sign each transaction with a signature of their own, and submit them to the smart contract via the operator. Those signatures can then be verified on the smart contract using `ecrecover()` function, comparing them with the list of trusted validators.

5.4. Reentrancy Attack

ID	IDX-004
Target	BridgeBank (BSC, Ethereum, EvryNet)
Category	General Smart Contract Vulnerability
CWE	CWE-841: Improper Enforcement of Behavioral Workflow
Risk	**Severity:** Medium **Impact:** High The attacker can call the `lock()` function repeatedly via reentrancy attack to record excessively high amount of token locked to gain tokens in the destination chain, resulting in a massive monetary loss to the platform. **Likelihood:** Low This attack will have an impact only when the bridge accepts ERC-777 tokens or any other tokens with callback hook to the token sender on transfer.
Status	Resolved EvryNet team has resolved this issue in commit 86601724f42cfdd5abaf6cd179f29eb0d4971469 by applying a `nonReentrant` modifier from the OpenZeppelin's `ReentrancyGuardUpgradeable` contract to prevent the contract from being exploited by a reentrancy attack.

5.4.1. Description

In the `BridgeBank` contract, the `lock()` function can be used by the users to lock tokens inside the contract and let the relayer bridge the tokens to another chain. Since the `_token` parameter can be the address of any token, an attacker can hijack the execution flow by locking the tokens with callback hooking and perform reentrancy attack to call the `lock()` function again from the token transfer function at line 115 - 119.

BridgeBank.sol

```
81  function lock(
82      address _recipient,
83      address _token,
84      uint256 _amount,
85      string memory _chainName
86  ) public payable availableNonce whenNotPaused {
87      string memory symbol;
88
89      // ETH deposit
90      if (msg.value > 0) {
91          require(
92              _token == address(0),
93              "Ethereum deposits require the 'token' address to be the null
```

```
address"
94          );
95          require(
96              msg.value == _amount,
97              "The transactions value must be equal the specified amount (in
wei)"
98          );
99          symbol = "EVRY";
100
101         lockFunds(
102         payable(msg.sender),
103         _recipient,
104         _token,
105         symbol,
106         _amount,
107         _chainName
108         );
109
110     }// ERC20 deposit
111     else {
112
113         uint beforeLock = BridgeToken(_token).balanceOf(address(this));
114
115         BridgeToken(_token).safeTransferFrom(
116             msg.sender,
117             address(this),
118             _amount
119         );
120
121         uint afterLock = BridgeToken(_token).balanceOf(address(this));
122
123         // Set symbol to the ERC20 token's symbol
124         symbol = BridgeToken(_token).symbol();
125
126         lockFunds(
127         payable(msg.sender),
128         _recipient,
129         _token,
130         symbol,
131         afterLock - beforeLock,
132         _chainName
133         );
134     }
135 }
```

For example, this flaw can be exploited using valid ERC-777 tokens (https://eips.ethereum.org/EIPS/eip-777) by creating a malicious contract with a `tokensToSend()` hook function that has a reentrant calling to the `lock()`

function. The attacker then transfers ERC-777 tokens to the malicious contract, and uses that contract to call the `lock()` function of `BridgeBank` contract. Before the token is transferred, the `tokensToSend()` hook will be invoked, giving the control of execution flow to the malicious contract, allowing the `lock()` function to be called again from within the function execution. This causes the `afterLock` amount of multiple function callings to be inflated, making it higher than the actual amount of token transferred.

5.4.2. Remediation

Inspex suggests inheriting OpenZeppelin's `ReentrancyGuardUpgradeable` contract and use the `nonReentrant` modifier to prevent repeated callings of functions, for example:

BridgeBank.sol

```
5  import "./EthBank.sol";
6  import "./BridgeBankPausable.sol";
7  import "./Ownable.sol";
8  import "./LockTimer.sol";
9  import
10 "../../node_modules/@openzeppelin/contracts-upgradeable/proxy/utils/Initializab
11 le.sol";
   import
   "../../node_modules/@openzeppelin/contracts/token/ERC20/utils/SafeERC20.sol";
   import
   "../../node_modules/@openzeppelin/contracts-upgradeable/security/ReentrancyGuar
   dUpgradeable.sol";
```

BridgeBank.sol

```
18 contract BridgeBank is ReentrancyGuardUpgradeable, EthBank, BridgeBankPausable,
19 Ownable, LockTimer{
20     using SafeERC20 for BridgeToken;
21
22     address public operator;
23
24     /*
25      * @dev: Constructor, sets operator
26      */
27     function initialize(address _operatorAddress) public payable initializer {
28         __ReentrancyGuard_init();
29         operator = _operatorAddress;
30         owner = msg.sender;
31         lockBurnNonce = 0;
32         _paused = false;
   }
```

BridgeBank.sol

```
81 function lock(
82     address _recipient,
```

```
83      address _token,
84      uint256 _amount,
85      string memory _chainName
86  ) public payable availableNonce whenNotPaused nonReentrant {
```

BridgeBank.sol

```
145 function unlock(
146     address payable _recipient,
147     address tokenAddress,
148     string memory _symbol,
149     uint256 _amount,
150     bytes32 _interchainTX
151 ) public onlyOperator whenNotPaused nonReentrant {
```

BridgeBank.sol

```
174 function emergencyWithdraw(
175     address tokenAddress,
176     uint256 _amount
177 ) public onlyOperator whenPaused isAbleToWithdraw nonReentrant {
```

BridgeBank.sol

```
205 function refund(
206     address payable _recipient,
207     address _tokenAddress,
208     string memory _symbol,
209     uint256 _amount,
210     uint256 _nonce
211 ) public onlyOperator whenNotPaused nonReentrant {
```

5.5. Insufficient Logging for Privileged Functions

ID	IDX-005
Target	BridgeBank (BSC, Ethereum, EvryNet)
Category	General Smart Contract Vulnerability
CWE	CWE-778: Insufficient Logging
Risk	**Severity:** Very Low **Impact:** Low Privileged functions' executions cannot be monitored easily by the users. **Likelihood:** Low It is not likely that the execution of the privileged functions will be a malicious action.
Status	Resolved EvryNet team has resolved this issue in commit 86601724f42cfdd5abaf6cd179f29eb0d4971469 by emiting events for all critical functions.

5.5.1. Description

Privileged functions that are executable by the controlling parties are not logged properly by emitting events. Without events, it is not easy for the public to monitor the execution of those privileged functions, allowing the controlling parties to perform actions that cause big impacts to the platform.

For example, the owner can modify the operator contract address by executing `changeOperator()` function in the `BridgeBank` contract, and no event is emitted.

The privileged functions without sufficient logging are as follows:

File	Contract	Function	Modifier
BridgeBank.sol (L:45)	BridgeBank	changeOperator()	isOwner
BridgeBank.sol (L:257)	BridgeBank	setEmergencyWithdraw DelayTime()	isOwner

5.5.2. Remediation

Inspex suggests emitting events for the execution of privileged functions, for example:

BridgeBank.sol

```
44  event ChangeOperator(address _oldOperator, address _newOperator);
45  function changeOperator(address _newOperator)
46      public
```

```
47      isOwner
48  {
49      emit ChangeOperator(operator, _newOperator);
50      operator = _newOperator;
51  }
```

5.6. Whitelisting of Token Address

ID	IDX-006
Target	BridgeBank (BSC, Ethereum, EvryNet)
Category	Advanced Smart Contract Vulnerability
CWE	CWE-20: Improper Input Validation
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this recommendation.

5.6.1. Description

In the `BridgeBank` contract, the users can lock any token inside the contract using the `lock()` function to bridge that token to another blockchain.

BridgeBank.sol

```
74  /*
75   * @dev: Locks received EVRY/ERC20 funds.
76   *
77   * @param _recipient: representation of destination address.
78   * @param _token: token address in origin chain (0x0 if ethereum)
79   * @param _amount: value of deposit
80   */
81  function lock(
82      address _recipient,
83      address _token,
84      uint256 _amount,
85      string memory _chainName
86  ) public payable availableNonce whenNotPaused {
87      string memory symbol;
88
89      // ETH deposit
90      if (msg.value > 0) {
91          require(
92              _token == address(0),
93              "Ethereum deposits require the 'token' address to be the null
    address"
94          );
95          require(
```

```
96              msg.value == _amount,
97              "The transactions value must be equal the specified amount (in
    wei)"
98          );
99          symbol = "EVRY";
100
101         lockFunds(
102         payable(msg.sender),
103         _recipient,
104         _token,
105         symbol,
106         _amount,
107         _chainName
108         );
109
110     }// ERC20 deposit
111     else {
112
113         uint beforeLock = BridgeToken(_token).balanceOf(address(this));
114
115         BridgeToken(_token).safeTransferFrom(
116             msg.sender,
117             address(this),
118             _amount
119         );
120
121         uint afterLock = BridgeToken(_token).balanceOf(address(this));
122
123         // Set symbol to the ERC20 token's symbol
124         symbol = BridgeToken(_token).symbol();
125
126         lockFunds(
127         payable(msg.sender),
128         _recipient,
129         _token,
130         symbol,
131         afterLock - beforeLock,
132         _chainName
133         );
134     }
135 }
```

However, there is no checking mechanism whether the token is supported or not in the `lock()` function, and not all tokens are supported in the destination blockchain. Therefore, unsupported tokens can be incorrectly locked inside the contract without being bridged to the destination blockchain.

As the bridge operates with the operator, the operator will need to execute the `refund()` function to return the tokens to the users if those tokens are not supported.

BridgeBank.sol

```
197 /*
198  * @dev: refunds EVRY and ERC20 tokens held on the contract.
199  *
200  * @param _recipient: recipient's is an evry address
201  * @param _token: token contract address
202  * @param _symbol: token symbol
203  * @param _amount: wei amount or ERC20 token count
204  */
205 function refund(
206     address payable _recipient,
207     address _tokenAddress,
208     string memory _symbol,
209     uint256 _amount,
210     uint256 _nonce
211 ) public onlyOperator whenNotPaused {
212     require(
213         refundCompleted[_nonce].isRefunded == false,
214         "This refunds has been processed before"
215     );
216     require(
217         refundCompleted[_nonce].tokenAddress == _tokenAddress,
218         "This refunds has been processed before"
219     );
220     require(
221         refundCompleted[_nonce].sender == _recipient,
222         "This refunds has been processed before"
223     );
224
225     // Check if it is EVRY
226     if (_tokenAddress == address(0)) {
227         address thisadd = address(this);
228         require(
229             thisadd.balance >= _amount,
230             "Insufficient ethereum balance for delivery."
231         );
232     } else {
233         require(
234             BridgeToken(_tokenAddress).balanceOf(address(this)) >= _amount,
235             "Insufficient ERC20 token balance for delivery."
236         );
237     }
238     refunds(_recipient, _tokenAddress, _symbol, _amount, _nonce);
239 }
```

This action costs gas for the operator to execute the function, and the users need to wait for their token to be returned from the contract.

5.6.2. Remediation

Inspex suggests implementing a whitelist of allowed tokens to prevent the locking of unsupported tokens.

5.7. Modifier Without Effect

ID	IDX-007
Target	BscBank (BSC, EvryNet) EthBank (Ethereum, EvryNet)
Category	Smart Contract Best Practice
CWE	CWE-1164: Irrelevant Code
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	Resolved EvryNet team has resolved this issue as suggested in commit 166f5ba9b610cccb6b349300e2227de74c1b44a7 by removing the unnecessary modifier.

5.7.1. Description

In the `BscBank` and `EthBank` contracts, the `availableNonce` modifier is implemented.

EthBank.sol

```
73  modifier availableNonce() {
74      require(lockBurnNonce + 1 > lockBurnNonce, "No available nonces.");
75      _;
76  }
```

However, this modifier has no effect because the condition will always be evaluated to true. Therefore, the functions with the `availableNonce` modifier will have their gas usage increased unnecessarily.

5.7.2. Remediation

Inspex suggests removing the unnecessary modifier or edit it to follow the intended business design.

5.8. Improper Function Visibility

ID	IDX-008
Target	Migrations (BSC, Ethereum, EvryNet)
Category	Smart Contract Best Practice
CWE	CWE-710: Improper Adherence to Coding Standards
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this best practice recommendation.

5.8.1. Description

Functions with public visibility copy calldata to memory when being executed, while external functions can read directly from calldata. Memory allocation uses more resources (gas) than reading directly from calldata.

For example, the following source code shows that the `lock()` function of the `BridgeBank` contract is set to public and it is never called from any internal function.

BridgeBank.sol

```
81  function lock(
82      address _recipient,
83      address _token,
84      uint256 _amount,
85      string memory _chainName
86  ) public payable availableNonce whenNotPaused {
```

The following table contains all functions that have `public` visibility and are never called from any internal function.

File	Contract	Function
Migrations.sol (L:19)	Migrations	setCompleted()
Migrations.sol (L:23)	Migrations	upgrade()
BridgeBank.sol (L:26)	BridgeBank	initialize()
BridgeBank.sol (L:45)	BridgeBank	changeOperator()
BridgeBank.sol (L:61)	BridgeBank	pause()

BridgeBank.sol (L:70)	BridgeBank	unpause()
BridgeBank.sol (L:81)	BridgeBank	lock()
BridgeBank.sol (L:145)	BridgeBank	unlock()
BridgeBank.sol (L:174)	BridgeBank	emergencyWithdraw()
BridgeBank.sol (L:205)	BridgeBank	refund()
BridgeBank.sol (L:257)	BridgeBank	setEmergencyWithdrawDelayTime()
Ownable.sol (L:20)	Ownable	transferOwnership()

5.8.2. Remediation

Inspex suggests changing all functions' visibility to `external` if they are not called from any `internal` function as shown in the following example:

BridgeBank.sol

```
81  function lock(
82      address _recipient,
83      address _token,
84      uint256 _amount,
85      string memory _chainName
86  ) external payable availableNonce whenNotPaused {
```

5.9. Inexplicit Solidity Compiler Version

ID	IDX-009
Target	Migrations (BSC, Ethereum, EvryNet)
Category	Smart Contract Best Practice
CWE	CWE-1104: Use of Unmaintained Third Party Components
Risk	**Severity:** Info **Impact: None** **Likelihood: None**
Status	No Security Impact EvryNet team has acknowledged this best practice recommendation.

5.9.1. Description

The Solidity compiler versions declared in the smart contracts were not explicit. Each compilation may be done using different compiler versions, which may potentially result in compatibility issues, for example:

Migrations.sol

```
1 pragma solidity ^0.8.0;
2 //SPDX-License-Identifier: MIT
```

The following table contains all targets which the inexplicit compiler version is declared.

Contract	Version
Migrations	^0.8.0
BridgeBank	^0.8.0
BridgeBankPausable	^0.8.0
EthBank	^0.8.0
BscBank	^0.8.0
BridgeToken	^0.8.0

5.9.2. Remediation

Inspex suggests fixing the solidity compiler to the latest stable version. At the time of the audit, the latest stable version of Solidity compiler in major 0.8 is v0.8.9[2].

6. Appendix

6.1. About Inspex

Inspex is formed by a team of cybersecurity experts highly experienced in various fields of cybersecurity. We provide blockchain and smart contract professional services at the highest quality to enhance the security of our clients and the overall blockchain ecosystem.

Follow Us On:

Website	https://inspex.co
Twitter	@InspexCo
Facebook	https://www.facebook.com/InspexCo
Telegram	@inspex_announcement

6.2. References

[1] "OWASP Risk Rating Methodology." [Online]. Available: https://owasp.org/www-community/OWASP_Risk_Rating_Methodology. [Accessed: 08-May-2021]

[2] Ethereum, "Releases · ethereum/solidity." [Online]. Available: https://github.com/ethereum/solidity/releases. [Accessed: 29-Sep-2021]

inspeX

CYBERSECURITY PROFESSIONAL SERVICE