CYBER UNIT
Cyber Security Strategic Partner

www.cyberunit.tech

Chainport

Smart Contract Audit

Date: June 14, 2021

Report for: Chainport

By: CyberUnit.Tech

This document may contain confidential information about IT systems and the customer's intellectual property and information about potential vulnerabilities and methods of their exploitation.

The report containing confidential information can be used internally by the customer or disclosed publicly after all vulnerabilities are fixed upon the customer's decision.

Scope and Code Revision Date

Repository	https://github.com/chainport/smart-contracts
Files	BridgeMintableToken.sol, ChainportBridgeBsc.sol, ChainportBridgeEth.sol, MaintainersRegistry.sol, Validator.sol
Initial Audit Commit	1295db595eeb9be363b3fa0ded84c9b5b60173d2
Initial Audit Date	01.06.2021

Table of contents

Introduction

This report presents the findings of the security assessment of the Customer` s smart contract and its code review conducted between May 19 2021– June 1, 2021.

Scope

The scope of the project is Chainport, which can be found in the repo:

https://github.com/chainport/smart-contracts

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

- Reentrancy
- Timestamp Dependence
- Gas Limit and Loops
- DoS with (Unexpected) Throw
- DoS with Block Gas Limit
- Transaction-Ordering Dependence
- Style guide violation
- Transfer forwards all gas
- ERC20 API violation
- Compiler version not fixed
- Unchecked external call – Unchecked math
- Unsafe type inference
- Implicit visibility level

Executive Summary

According to the assessment, Chainport's protocol security risk is severe; two critical, one medium, one low, and one informed issues were found for the smart contract. This

contract considers security risk as very severe and cannot be used without developer edits.

Our team analyzed code functionality, manual audit, and automated checks. All issues found during automated investigation manually reviewed, and application vulnerabilities presented in the Audit overview section. A general overview is presented in the AS-IS section, and you can find all encountered matters in the Audit overview section.

Severity Definitions

Risk Level	Description
Critical	Critical vulnerabilities are usually straightforward to exploit and can lead to tokens loss.
High	High-level vulnerabilities are difficult to exploit. However, they also significantly impact smart contract execution, e.g., public access to crucial functions.
Medium	Medium-level vulnerabilities are essential to fix; however, they can't lead to tokens loss.
Low	Low-level vulnerabilities are mostly related to outdated or unused code snippets.
Lowest / Code Style / Best Practice	Lowest-level vulnerabilities, code style violations, and info statements can't affect smart contract execution and can generally be ignored.

AS-IS overview

BridgeMintableToken.sol

BridgeMintableToken is a smart contract ERC20 standard.

Contract BridgeMintableToken is Ownable

BridgeMintableToken contract has following functions:

- constructor string tokenName_, string tokenSymbol_, uint8 decimals_;
- mint address _to;

Audit overview

Critical

1. [Fixed] The function mint is not protected. This vulnerability makes it possible an uncontrolled token mint (see Appendix A pic. 1 for evidence).

High

No high issues were found.

Medium

No medium issues were found.

Low

No low issues were found.

Lowest / Code style / Best Practice

1. [Fixed]Missing event declaration for mint method – the mint event is not declared, and as a result, this leads to records not adding to logs (see Appendix A pic. 2 for evidence).

ChainportBridgeBsc.sol

ChainportBridgeBsc is BSC network adapter.

ChainportBridgeBsc is Ownable.

ChainportBridgeBsc has the following parameters and structs:

- IValidator public signatureValidator
- mapping(address => address) public erc20ToBep20Address
- mapping(string => uint256) public functionNameToNonce
- mapping(address => bool) public isCreatedByTheBridge
- bool public isFrozen

BridgeMintableToken contract has following functions:

- constructor string tokenName_, string tokenSymbol_, uint8 decimals_;
- mint address _to;

BridgeMintableToken contract has following functions:

- initialize – set initial addresses
- freezeBridge – freeze bsc bridge function
- unfreezeBridge – unfreeze bsc bridge function
- mintNewToken – mint new token function
- mintTokens – mint token function
- burnTokens – burn token function

Audit overview

Critical

No critical issues were found.

High

No high issues were found.

Medium

No medium issues were found.

Low

No low issues were found.

ChainportBridgeEth.sol

ChainportBridgeEth is ETH network adapter.

ChainportBridgeEth is Ownable.

ChainportBridgeEth has the following parameters and structs:

- PendingWithdrawal uint256 amount address beneficiary uint256 unlockingTime

ChainportBridgeEth contract has the following functions:

- initialize – Initialization function
- setAssetProtection – function to mark specific asset as protected
- unfreezeBridge – unfreeze ethereum bridge function
- setTimeLockLength – timelock setter function
- setThreshold – threshold setter function
- freezeToken – freeze tiken function
- releaseTokensByMaintainer – bytes signature, address token, uint256 amount, address beneficiary, uint256 nonce
- releaseTokensTimelockPassed – check if freeze time has passed
- releaseTokens – function to release tokens
- approveWithdrawalAndTransferFunds – function for congress to approve withdrawal and transfer funds
- rejectWithdrawal – function to reject withdrawal from congress
- isAboveThreshold – function to check if the amount is above a threshold
- getTokenBalance – address token Get contract balance of specific token

Audit overview

Critical

1. [Fixed] Replay vulnerability at the signature verification stage (see Appendix A pic. 3 for evidence).

High

No high issues were found.

Medium

1. [Fixed]Unused return for functions:
   IERC20(token).transfer(beneficiary,amount)
   (contracts/ChainportBridgeEth.sol#163)
   IERC20(token).transfer(p.beneficiary,p.amount)
   (contracts/ChainportBridgeEth.sol#184)
   IERC20(token).transfer(beneficiary,amount)
   (contracts/ChainportBridgeEth.sol#224)
   IERC20(token).transfer(p.beneficiary,p.amount)
   (contracts/ChainportBridgeEth.sol#241) (see Appendix A pic. 4 for evidence)

Low

1. [Fixed]Reentrancy vulnerabilities. The state variable changes after the contract invoke the translation. An attacker uses a function that is automatically executed after the token is passed from the target contract to re-execute the function before the state changes (see Appendix A pic. 5 for evidence)

.

MaintainersRegistry.sol

MaintainersRegistry is a maintainers database.

MaintainersRegistry is Ownable.

MaintainersRegistry has the following parameters and structs:

- chainportCongress chainport congress authorized address to modify maintainers
- allMaintainers

MaintainersRegistry contract has the following functions:

- initialize – the function to perform initialization
- addMaintainer – the function that serves for adding maintainer
- addMaintainerInternal – the function that serves for adding maintainer
- removeMaintainer – the function that serves for removing maintainer
- isMaintainer – the function to check if the wallet is the maintainer

Audit overview

Critical

No critical issues were found.

High

No high issues were found.

Medium

No medium issues were found.

Low

No low issues were found.

Validator.sol

Validator.sol is a smart contract that allows to validate digital signatures

Validator.sol is Ownable.

Validator contract has the following functions:

- initialize – set initial signatory address and Chainport congress
- setSignatoryAddress – set change signatory address
- verifyWithdraw – function to verify withdraw parameters and if signatory signed message
- recoverSignature – function to can check who signed the message
- recoverHash – recover signer message from the signature.

Audit overview

Critical

No critical issues were found.

High

No high issues were found.

Medium

No medium issues were found.

Low

No low issues were found.

Conclusion

Smart contracts within the scope were manually reviewed and analyzed with static analysis tools. A high-level description of functionality is presented in the As-is overview section of the report for the contract.

The audit report contains all found security vulnerabilities and other issues in the reviewed code.

Security engineers found two critical, one medium, one low, and one informed issue on the smart contract. This contract considers security risk as very severe and cannot be used without developer edits.

Disclaimer

The smart contracts given for audit have analyzed following the best industry practices at the date of this report, concerning: cybersecurity vulnerabilities and issues in smart contract source code, the details of which disclosed in this report, (Source Code); the Source Code compilation, deployment, and functionality (performing the intended functions).

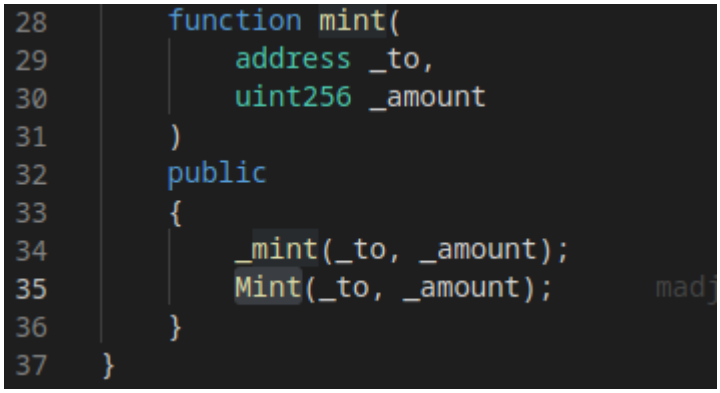
The audit doesn't make warranties on the security of the code. It also cannot be considered a sufficient assessment regarding the utility and safety of the system, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is essential to note that you should not rely on this report only. We recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.

Technical Disclaimer

Smart contracts are deployed and executed on a blockchain platform. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. Thus, the audit can’t guarantee the specific security of the audited smart contracts.

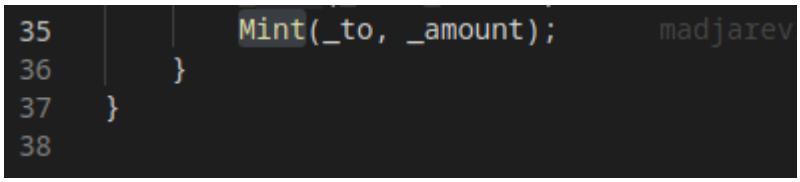
Appendix A. Evidence

Pic 1. The function mint is not protected:



```
28      function mint(
29          address _to,
30          uint256 _amount
31      )
32      public
33      {
34          _mint(_to, _amount);
35          Mint(_to, _amount);
36      }
37  }
```

Pic 2. Missing event declaration for mint method:

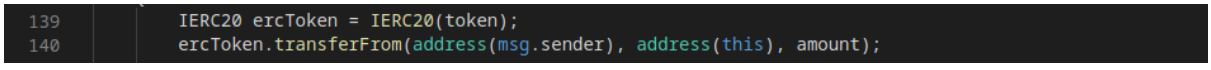


```
35          Mint(_to, _amount);
36      }
37  }
38
```

Pic 3. Replay vulnerability at the signature verification stage:

```
145     function releaseTokensByMaintainer(
146         bytes memory signature,
147         address token,
148         uint256 amount,
149         address beneficiary,
150         uint256 nonce
151     )
152     public
153     onlyMaintainer
154     isNotFrozen
155     {
156         require(isTokenHavingPendingWithdrawal[token] == false, "Token is currently having pending withdrawal.");
157
158         require(nonce == functionNameToNonce["mintTokens"] + 1);
159         functionNameToNonce["mintTokens"] = nonce;
160
161         bool isMessageValid = signatureValidator.verifyWithdraw(signature, token, amount, beneficiary);
```

Pic 4. Ignores return:



```
139         IERC20 ercToken = IERC20(token);
140         ercToken.transferFrom(address(msg.sender), address(this), amount);
```

```
162         require(isMessageValid == true, "Error: Signature is not valid.");
163         IERC20(token).transfer(beneficiary, amount);
164
```

```
184                 IERC20(token).transfer(p.beneficiary, p.amount);
185                 emit TokensUnfreezed(token, p.beneficiary, p.amount);
```

```
241         IERC20(token).transfer(p.beneficiary, p.amount);
242         // Emit events
243         emit TokensUnfreezed(token, p.beneficiary, p.amount);
```

Pic 5. Reentrancy vulnerabilities:

```
229     // Function for congress to approve withdrawal and transfer funds
230     function approveWithdrawalAndTransferFunds(
231         address token
232     )
233     public
234     onlyChainportCongress
235     isNotFrozen
236     {
237         require(isTokenHavingPendingWithdrawal[token] == true);
238         // Get current pending withdrawal attempt
239         PendingWithdrawal memory p = tokenToPendingWithdrawal[token];
240         // Transfer funds to user
241         IERC20(token).transfer(p.beneficiary, p.amount);
242         // Emit events
243         emit TokensUnfreezed(token, p.beneficiary, p.amount);
244         emit WithdrawalApproved(token, p.beneficiary, p.amount);
245
246         // Clear up the state and remove pending flag
247         delete tokenToPendingWithdrawal[token];      madjarevicn, a month ago • Add newes
248         isTokenHavingPendingWithdrawal[token] = false;
249     }
250
```

```
168     function releaseTokensTimelockPassed(
169         bytes memory signature,
170         address token,
171         uint256 amount
172     )
173     public
174     isNotFrozen
175     {
176         // Check if freeze time has passed and same user is calling again
177         if(isTokenHavingPendingWithdrawal[token] == true) {
178             PendingWithdrawal memory p = tokenToPendingWithdrawal[token];
179             if(p.amount == amount && p.beneficiary == msg.sender && p.unlockingTime <= block.timestamp) {
180                 // Verify the signature user is submitting
181                 bool isMessageValid = signatureValidator.verifyWithdraw(signature, token, amount, p.beneficiary);
182                 require(isMessageValid == true, "Error: Signature is not valid.");
183 
184                 IERC20(token).transfer(p.beneficiary, p.amount);
185                 emit TokensUnfreezed(token, p.beneficiary, p.amount);
186                 // Clear up the state and remove pending flag
187                 delete tokenToPendingWithdrawal[token];
188                 isTokenHavingPendingWithdrawal[token] = false;
```