SOFTWARE AUDIT REPORT

for

BHOP CONSULTANTING PTE. LTD.

Prepared By: Shuxiao Wang

Hangzhou, China
December 9, 2020

Document Properties

Client	BHOP Consultanting Pte. Ltd.
Title	Software Audit Report
Target	HBTC OpenSwap
Version	0.1
Author	Ruiyi Zhang
Auditors	Ruiyi Zhang, Xuxian Jiang, Jeff Liu
Reviewed by	Jeff Liu
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
0.1	December 9, 2020	Ruiyi Zhang	Initial Draft

Contact

For more information about this document and its contents, please contact PeckShield Inc.

Name	Shuxiao Wang
Phone	+86 173 6454 5338
Email	contact@peckshield.com

Contents

1 Introduction 4

1.1 About HBTC OpenSwap 4

1.2 About PeckShield 5

1.3 Methodology 5

1.4 Disclaimer 7

2 Findings 9

2.1 Summary 9

2.2 Key Findings 10

3 Detailed Results 11

3.1 Improved Sanity Checks of MsgAddLiquidity() 11

3.2 Possible Front-Running For Reduced Return 12

3.3 Lack of Accepting Liquidity Donation in Current Pools 13

4 Conclusion 14

References 15

1 Introduction

Given the opportunity to review the **HBTC OpenSwap** design document and related source code, we outline in the report our systematic approach to evaluate potential security issues in the smart contract implementation, expose possible semantic inconsistencies between smart contract code and design document, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About HBTC OpenSwap

HBTC Chain presents the next-generation blockchain-based technology for decentralized asset custody and clearing. Inspired by `Uniswap`, HBTC OpenSwap defines native transactions to support features similar to `Uniswap` on HBTC Chain. In other words, it introduces the basic DeFi infrastructure and procedure to use on-chain automatic market makers for flexible and powerful token swap features. HBTC OpenSwap is currently implemented as a `Cosmos`-based module that greatly advances the HBTC Chain ecosystem by developing an essential DeFi infrastructure-level building block.

The basic information of HBTC OpenSwap is as follows:

Table 1.1: Basic Information of HBTC OpenSwap

Item	Description
Issuer	BHOP Consultanting Pte. Ltd.
Website	https://chain.hbtc.com/
Module	HBTC OpenSwap
Coding Language	Go
Audit Method	Whitebox
Latest Audit Report	December 9, 2020

In the following, we show the Git repository of reviewed files and the commit hash value used in this audit.

- https://github.com/hbtc-chain/bhchain/tree/main/x/openswap (74f34d0)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of the current blockchain ecosystems by offering top-notch, industry-leading services and products (including the service of smart contract auditing). We are reachable at Telegram (https://t.me/peckshield), Twitter (http://twitter.com/peckshield), or Email (contact@peckshield.com).

Table 1.2: Vulnerability Severity Classification

	Likelihood		
Impact	*High*	*Medium*	*Low*
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on the OWASP Risk Rating Methodology [6]:

- Likelihood represents how likely a particular vulnerability is to be uncovered and exploited in the wild;
- Impact measures the technical loss and business damage of a successful attack;
- Severity demonstrates the overall criticality of the risk.

Likelihood and impact are categorized into three ratings: *H*, *M* and *L*, i.e., *high*, *medium* and *low* respectively. Severity is determined by likelihood and impact and can be classified into four categories accordingly, i.e., *Critical*, *High*, *Medium*, *Low* shown in Table 1.2.

To evaluate the risk, we go through a checklist of items and each would be labeled with a severity category. For one check item, if our tool or analysis does not identify any issue, the contract is considered safe regarding the check item. For any discovered issue, we might further deploy

Table 1.3: The Full Audit Checklist

Category	Checklist Items
Basic Coding Bugs	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use Of Untrusted Libraries
	(Unsafe) Use Of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
Semantic Consistency Checks	Semantic Consistency Checks
Advanced DeFi Scrutiny	Business Logics Review
	Functionality Checks
	Authentication Management
	Access Control & Authorization
	Oracle Security
	Digital Asset Escrow
	Kill-Switch Mechanism
	Operation Trails & Event Generation
	ERC20 Idiosyncrasies Handling
	Frontend-Contract Integration
	Deployment Consistency
	Holistic Risk Management
Additional Recommendations	Avoiding Use of Variadic Byte Array
	Using Fixed Compiler Version
	Making Visibility Level Explicit
	Making Type Inference Explicit
	Adhering To Function Declaration Strictly
	Following Other Best Practices

contracts on our private testnet and run tests to confirm the findings. If necessary, we would additionally build a PoC to demonstrate the possibility of exploitation. The concrete list of check items is shown in Table 1.3.

In particular, we perform the audit according to the following procedure:

- Basic Coding Bugs: We first statically analyze given smart contracts with our proprietary static code analyzer for known coding bugs, and then manually verify (reject or confirm) all the issues found by our tool.
- Semantic Consistency Checks: We then manually check the logic of implemented smart contracts and compare with the description in the white paper.
- Advanced DeFi Scrutiny: We further review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.
- Additional Recommendations: We also provide additional suggestions regarding the coding and development of smart contracts from the perspective of proven programming practices.

To better describe each issue we identified, we categorize the findings with Common Weakness Enumeration (CWE-699) [5], which is a community-developed list of software weakness types to better delineate and organize weaknesses around concepts frequently encountered in software development. Though some categories used in CWE-699 may not be relevant in smart contracts, we use the CWE categories in Table 1.4 to classify our findings.

1.4 Disclaimer

Note that this audit does not give any warranties on finding all possible security issues of the given smart contract(s), i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contract(s). Last but not least, this security audit should not be used as investment advice.

Table 1.4: Common Weakness Enumeration (CWE) Classifications Used in This Audit

Category	Summary
Configuration	Weaknesses in this category are typically introduced during the configuration of the software.
Data Processing Issues	Weaknesses in this category are typically found in functionality that processes data.
Numeric Errors	Weaknesses in this category are related to improper calculation or conversion of numbers.
Security Features	Weaknesses in this category are concerned with topics like authentication, access control, confidentiality, cryptography, and privilege management. (Software security is not security software.)
Time and State	Weaknesses in this category are related to the improper management of time and state in an environment that supports simultaneous or near-simultaneous computation by multiple systems, processes, or threads.
Error Conditions, Return Values, Status Codes	Weaknesses in this category include weaknesses that occur if a function does not generate the correct return/status code, or if the application does not handle all possible return/status codes that could be generated by a function.
Resource Management	Weaknesses in this category are related to improper management of system resources.
Behavioral Issues	Weaknesses in this category are related to unexpected behaviors from code that an application uses.
Business Logic	Weaknesses in this category identify some of the underlying problems that commonly allow attackers to manipulate the business logic of an application. Errors in business logic can be devastating to an entire application.
Initialization and Cleanup	Weaknesses in this category occur in behaviors that are used for initialization and breakdown.
Arguments and Parameters	Weaknesses in this category are related to improper use of arguments or parameters within function calls.
Expression Issues	Weaknesses in this category are related to incorrectly written expressions within code.
Coding Practices	Weaknesses in this category are related to coding practices that are deemed unsafe and increase the chances that an exploitable vulnerability will be present in the application. They may not directly introduce a vulnerability, but indicate the product has not been carefully developed or maintained.

2 Findings

2.1 Summary

Here is a summary of our findings after analyzing HBTC OpenSwap. As mentioned earlier, we in the first phase of our audit studied HBTC OpenSwap source code (including related libraries) and ran our in-house static code analyzer through the codebase. The purpose here is to statically identify known coding bugs, and then manually verify (reject or confirm) issues reported by our tools. After that, we manually review business logic, examine system operations, and place operation-specific aspects under scrutiny to uncover possible pitfalls and/or bugs.

Table 2.1: The Severity of Our Findings

Severity	# of Findings	
Critical	0	
High	0	
Medium	1	🟦
Low	1	🟦
Informational	1	🟦
Total	3	

We have so far identified a list of potential issues: some of them involve subtle corner cases that might not be previously thought of, while others refer to unusual interactions among multiple modules. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined several issues of varying severities that need to be brought up and paid more attention to. These issues are categorized in the above Table 2.1. More information can be found in the next subsection, and the detailed discussions of each of them are in Section 3.

2.2 Key Findings

Overall, these smart contracts are well-designed and engineered, though the implementation can be improved by resolving the identified issues (shown in Table 2.2), including 1 medium-severity vulnerability, 1 low-severity vulnerability, and 1 informational recommendation.

Table 2.2: Key HBTC OpenSwap Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Informational	Improved Sanity Checks of MsgAddLiquidity()	Coding Practices	Confirmed
PVE-002	Medium	Possible Front-Running For Reduced Return	Business Logic	Confirmed
PVE-003	Low	Lack of Accepting Liquidity Donation in Current Pools	Business Logic	Confirmed

Beside the identified issues, we emphasize that for any user-facing applications and services, it is always important to develop necessary risk-control mechanisms and make contingency plans, which may need to be exercised before the mainnet deployment. The risk-control mechanisms should kick in at the very moment when the contracts are being deployed on mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Improved Sanity Checks of MsgAddLiquidity()

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A

- Target: `MsgAddLiquidity`
- Category: Coding Practices [3]
- CWE subcategory: CWE-391 [1]

Description

The HBTC OpenSwap module is inspired by `Uniswap` and implements an AMM-based token-swapping subsystem as a basic DeFi infrastructure-level service on HBTC Chain. In particular, it defines the eleven fundamental message types and implements the essential logic for their handling. Among them, `MsgAddLiquidity` and `MsgRemoveLiquidity` message types allow liquidity providers to contribute or de-contribute their liquidity.

In the following, We show below the message type `MsgAddLiquidity`'s validity check routine: `ValidateBasic()`. Apparently, there should be a check on `MaxTokenAAmount` and `MaxTokenBAmount`. But it only ensures that the `MaxTokenAAmount` is positive (line 280).

```
270 func (msg MsgAddLiquidity) ValidateBasic() sdk.Error {
271   if !msg.From.IsValidAddr() {
272     return sdk.ErrInvalidAddr(fmt.Sprintf("from address: %s is invalid", msg.From.String
          ()))
273   }
274   if !msg.TokenA.IsValid()  !msg.TokenB.IsValid() {
275     return sdk.ErrInvalidSymbol("invalid token symbol")
276   }
277   if msg.TokenA == msg.TokenB {
278     return sdk.ErrInvalidSymbol("token a and token b cannot be equal")
279   }
280   if !msg.MaxTokenAAmount.IsPositive()  !msg.MaxTokenAAmount.IsPositive() {
281     return sdk.ErrInvalidAmount("token amount should be positive")
282   }
283   return nil
}
```

Listing 3.1: openswap/types/msg.go

Recommendation Ensure the associated `ValidateBasic()` as shown in the following example:

```
func (msg MsgAddLiquidity) ValidateBasic() sdk.Error {
  if !msg.From.IsValidAddr() {
    return sdk.ErrInvalidAddr(fmt.Sprintf("from address: %s is invalid", msg.From.String()))
  }
  if !msg.TokenA.IsValid()  !msg.TokenB.IsValid() {
    return sdk.ErrInvalidSymbol("invalid token symbol")
  }
  if msg.TokenA == msg.TokenB {
    return sdk.ErrInvalidSymbol("token a and token b cannot be equal")
  }
  if !msg.MaxTokenAAmount.IsPositive()  !msg.MaxTokenBAmount.IsPositive() {
    return sdk.ErrInvalidAmount("token amount should be positive")
  }
  return nil
}
```

Listing 3.2: openswap/types/msg.go

Status The issue has been confirmed.

3.2 Possible Front-Running For Reduced Return

- ID: PVE-002
- Severity: Medium
- Likelihood: Medium
- Impact: Medium

- Target: `handler.go`
- Category: Business Logic [4]
- CWE subcategory: CWE-666 [2]

Description

In the OpenSwap contract, we notice that the swap operation respects the `minAmountOut` threshold that specifies the expected minimal token amount out of this trade of selling `AmountIn`. We'd like to point out that such trading provides a certain protection against price slippage but may not be sufficient against sophisticated front-running attacks that could just meet the `minAmountOut` requirement while still lead to a smaller return for trading users.

We emphasize that this is a common issue plaguing current AMM-based DEX solutions. Specifically, a large trade may be sandwiched by a preceding sell to reduce the market price and a tailgating buy-back of the same amount plus the trade amount. Such sandwiching behavior unfortunately causes a loss and brings a smaller return to the trading user because the swap rate is lowered by the preceding sell. As a mitigation, we may consider specifying the restriction on possible slippage caused by the trade or implementing a `TWAP` or `time-weighted average price` reference (similar in `Perpetual Protocol`). Nevertheless, we need to acknowledge that this is largely inherent to current blockchain infrastructure and there is still a need to continue the search efforts for an effective defense.

Recommendation Develop effective mitigation to the above front-running attack to better protect the interests of trading users.

Status This issue has been confirmed. For the sake of mainnet stablility, the team decided to keep it as is for the time being.

3.3 Lack of Accepting Liquidity Donation in Current Pools

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `liquidity`
- Category: Business Logic [4]
- CWE subcategory: CWE-666 [2]

Description

All trading pairs in HBTC OpenSwap can be created in a distributed, trustless manner. As mentioned in Section 3.1, HBTC OpenSwap provides a dedicated message type `MsgCreateTradingPair` to allow for dynamic creation of trading pairs and this is consistent with the original `Uniswap` design.

While reviewing the logic behind liquidity additions and removals, we notice that the current execution path does not support liquidity donation. While there is no issue regarding current implementation, it is indeed a design choice that needs to make regarding whether liquidity donation should be accepted or not.

Our suggestion is that liquidity donation is helpful to increase pool valuation, reduce trading slippage, and provide a better user experience. With that, we'd recommend to accept liquidity donations.

Recommendation Suggest to accept donated liquidity.

Status This issue has been confirmed. For the sake of mainnet stablility, the team decided to keep it as is for the time being.

4 Conclusion

In this audit, we have analyzed the HBTC OpenSwap design and implementation. HBTC OpenSwap defines native transactions to support features similar to `Uniswap` on HBTC Chain. During the audit, we notice that the current code base is well structured and neatly organized, and the identified issues are promptly confirmed and fixed.

As a final precaution, we need to emphasize that smart contracts as a whole are still in an early, but exciting stage of development. To improve this report, we greatly appreciate any constructive feedbacks or suggestions, on our methodology, audit findings, or potential gaps in scope/coverage.

References

[1] MITRE. CWE-391: Unchecked Error Condition. https://cwe.mitre.org/data/definitions/391.html.

[2] MITRE. CWE-666: Operation on Resource in Wrong Phase of Lifetime. https://cwe.mitre.org/data/definitions/666.html.

[3] MITRE. CWE CATEGORY: Bad Coding Practices. https://cwe.mitre.org/data/definitions/1006.html.

[4] MITRE. CWE CATEGORY: Business Logic Errors. https://cwe.mitre.org/data/definitions/840.html.

[5] MITRE. CWE VIEW: Development Concepts. https://cwe.mitre.org/data/definitions/699.html.

[6] OWASP. Risk Rating Methodology. https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology.

[7] PeckShield. PeckShield Inc. https://www.peckshield.com.