SMART CONTRACT AUDIT REPORT

for

Timeswap V2

Prepared By: Xiaomi Huang

PeckShield

March 13, 2023

Document Properties

Client	Timeswap Labs Ltd.
Title	Smart Contract Audit Report
Target	Timeswap V2
Version	1.0
Author	Xiaotao Wu
Auditors	Xiaotao Wu, Xuxian Jiang
Reviewed by	Xiaomi Huang
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	March 13, 2023	Xiaotao Wu	Final Release
1.0-rc	December 1, 2022	Xiaotao Wu	Release Candidate #1

Contact

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

Name	Xiaomi Huang
Phone	+86 183 5897 7782
Email	contact@peckshield.com

Contents

1 Introduction

Given the opportunity to review the design document and related smart contract source code of the Timeswap V2 protocol, 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 Timeswap V2

Timeswap is a fixed time preference protocol for users to manage their ERC20 tokens over discrete time. It works as a zero-liquidation, fixed maturity money market and options market in one. Timeswap utilizes a unique constant sum options specification and a duration weighted constant product automated market maker (AMM) similar to Uniswap AMM. Timeswap V2 is an improved version of Timeswap V1 and increases the capital efficiency of the AMM by 4x–5x while still maintaining all the base layer properties such as permissionless, oracleless and being highly capital efficient. The basic information of the audited protocol is as follows:

Table 1.1: Basic Information of The Timeswap V2

Item	Description
Name	Timeswap Labs Ltd.
Website	https://www.timeswap.io
Type	EVM Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	March 13, 2023

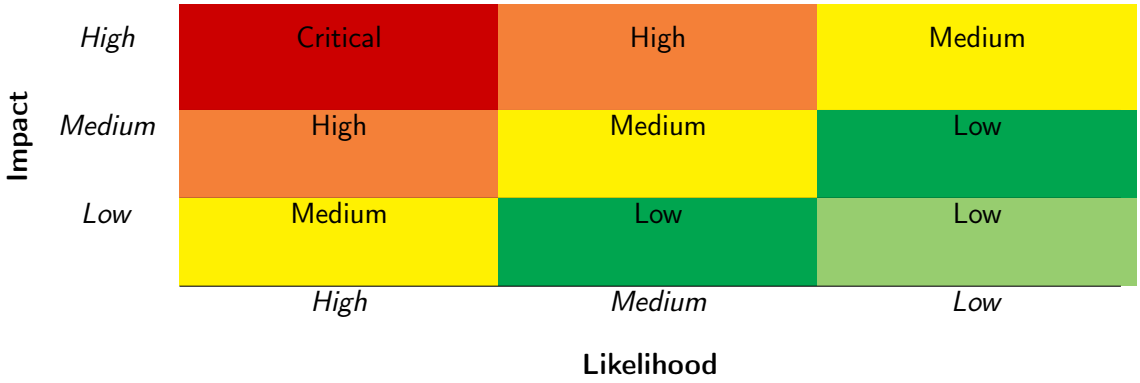
In the following, we show the Git repository of reviewed files and the commit hash values used in this audit:

- https://github.com/Timeswap-Labs/Timeswap-V2-Monorepo/tree/dev (b13afc7)

1.2 About PeckShield

PeckShield Inc. [7] is a leading blockchain security company with the goal of elevating the security, privacy, and usability of 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



1.3 Methodology

To standardize the evaluation, we define the following terminology based on 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 list of check 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

Table 1.3: The Full List of Check Items

Category	Check Item
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

deploy 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 security audit is not designed to replace functional tests required before any software release, and does not give any warranties on finding all possible security issues of the given smart contract(s) or blockchain software, 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 Logics	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 the design and implementation of the `Timeswap V2` protocol. During the first phase of our audit, we study the smart contract source code and run 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 tool. We further manually review business logics, examine system operations, and place DeFi-related aspects under scrutiny to uncover possible pitfalls and/or bugs.

Severity	# of Findings	
Critical	1	■
High	0	
Medium	1	■
Low	6	■■■■■■
Informational	1	■
Total	9	

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 contracts. For each uncovered issue, we have therefore developed test cases for reasoning, reproduction, and/or verification. After further analysis and internal discussion, we determined a few issues of varying severities that need to be brought up and paid more attention to, which are categorized in the above table. 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 2.1), including 1 critical-severity vulnerability, 1 medium-severity vulnerability 6 low-severity vulnerabilities, and 1 informational suggestion.

Table 2.1: Key Timeswap V2 Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Critical	Incorrect Balance Check Logic in TimeswapV2Option::swap()	Business Logic	Fixed
PVE-002	Medium	Incorrect Position Transfer Target in TimeswapV2Token::burn()	Business Logic	Fixed
PVE-003	Low	Incorrect Long Position Receiver in TimeswapV2PeripheryTransform	Business Logic	Fixed
PVE-004	Low	Revisited Logic in PeripheryUniswapV3Transform	Business Logic	Fixed
PVE-005	Low	Incorrect Implementation Logic in timeswapV2PeripheryTransformInternal()	Business Logic	Fixed
PVE-006	Low	Incorrect token0 Receiver in PeripheryUniswapV3Redeem::redeem()	Business Logic	Fixed
PVE-007	Low	Incorrect Rebalance Input in PeripheryUniswapV3Rebalance::rebalance()	Business Logic	Fixed
PVE-008	Informational	Improved Sanity Checks in TimeswapV2Pool::transferFees()	Coding Practices	Fixed
PVE-009	Low	Improved Implementation Logic in borrowGivenPrincipal()	Business Logic	Fixed

Besides recommending specific countermeasures to mitigate these issues, we also emphasize that 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 need to kick in at the very moment when the contracts are being deployed in mainnet. Please refer to Section 3 for details.

3 Detailed Results

3.1 Incorrect Balance Check Logic in TimeswapV2Option::swap()

- ID: PVE-001
- Severity: critical
- Likelihood: High
- Impact: High
- Target: `TimeswapV2Option`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

In `Timeswap V2`, an option pool is fully collateralized with either `token0` or `token1`. When the option has locked `token0`, it gives the holder the right but not the obligation to swap `token1` for `token0` before the maturity of the option. When the option has locked `token1`, the holder can swap `token0` for `token1`. When the swap is exercised, the token that is deposited becomes the new locked token of the option, thus giving the holder the right but not the obligation to swap the tokens back before maturity. This design gives the holder the ability to swap `token0` and `token1` back and forth as many times as the holder wants. While examining the option swap logic, we notice an issue that a user can withdraw asset from the option pool without locking the corresponding collateral.

To elaborate, we show below the `swap()` routine. Suppose a user wants to swap `token0` for `token1`, the execution logic is rather straightforward: it firstly transfers `token1` from the option pool to the `param.tokenTo`, and then call the `timeswapV2OptionSwapCallback()` function to ask the `msg.sender` to transfer `token0` to this option pool. Lastly, the option pool should check if the `token1` balance target is achieved. However, the current implementation logic incorrectly checks the `token0` balance (lines 273-276).

```
216 /// @inheritdoc ITimeswapV2Option
217   function swap(TimeswapV2OptionSwapParam calldata param)
218     external
219     override
```

```
220     noDelegateCall
221     returns (
222       uint256 token0AndLong0Amount,
223       uint256 token1AndLong1Amount,
224       bytes memory data
225     )
226   {
227     ...
228     // transfer token to recipient.
229     IERC20(param.isLong0ToLong1 ? token0 : token1).safeTransfer(
230       param.tokenTo,
231       param.isLong0ToLong1 ? token0AndLong0Amount : token1AndLong1Amount
232     );

234     // ask the msg.sender to transfer token0 or token1 to this contract.
235     data = ITimeswapV2OptionSwapCallback(msg.sender).timeswapV2OptionSwapCallback(
236       TimeswapV2OptionSwapCallbackParam({
237         strike: param.strike,
238         maturity: param.maturity,
239         isLong0ToLong1: param.isLong0ToLong1,
240         token0AndLong0Amount: token0AndLong0Amount,
241         token1AndLong1Amount: token1AndLong1Amount,
242         data: param.data
243       })
244     );

246     // check if the token0 or token1 balance target is achieved.
247     Error.checkEnough(
248       IERC20(param.isLong0ToLong1 ? token0 : token1).balanceOf(address(this)),
249       param.isLong0ToLong1 ? currentProcess.balance0Target : currentProcess.
              balance1Target
250     );

252     ...
253   }
```

Listing 3.1: `TimeswapV2Option::swap()`

Recommendation Check the correct token balance after the option swap is executed.

Status This issue has been fixed.

3.2 Incorrect Position Transfer Target in TimeswapV2Token::burn()

- ID: PVE-002
- Severity: Medium
- Likelihood: High
- Impact: Low
- Target: `TimeswapV2Token`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

In `Timeswap V2`, a user can transfer the long/short option to the `TimeswapV2Token` contract and mint the corresponding amount of `TimeswapV2` tokens for the user. A user can also withdraw the long/short option from the `TimeswapV2Token` contract by burning the `TimeswapV2` tokens. Our analysis with the `burn()` function shows its current implementation is not correct.

To elaborate, we show below the code snippet of the `burn()` routine. Its logic is rather straightforward in burning the `TimeswapV2` tokens representing the long/short position and transfering the underlying equivalent long/short position amount to address of the recipient. However, the recipient of the short position is not correct. Specifically, when the `TimeswapV2` tokens representing short position have been burnt, the short position should be sent to `param.shortTo`, instead of current `param.long1To`.

```
255   /// @inheritdoc ITimeswapV2Token
256   function burn(TimeswapV2TokenBurnParam calldata param) external override {
257     // start the guard to prevent reentrancy
258     raiseGuard(param.token0, param.token1, param.strike, param.maturity);
259
260     ParamLibrary.check(param);
261
262     address optionPair = OptionFactoryLibrary.getWithCheck(optionFactory, param.token0,
          param.token1);
263
264     ...
265
266     // case when the short position is to be burned
267     if (param.shortAmount != 0) {
268       TimeswapV2TokenPosition memory timeswapV2TokenPosition = TimeswapV2TokenPosition({
269         token0: param.token0,
270         token1: param.token1,
271         strike: param.strike,
272         maturity: param.maturity,
273         position: TimeswapV2OptionPosition.Short
274       });
275
276       // burn the TimeswapV2Token representing short position
```

```
277         _burn(msg.sender, _timeswapV2TokenPositionIds[timeswapV2TokenPosition.toKey()],
                param.shortAmount);
278
279         // transfer the underlying equivalent short position amount to address of the
                recipient of short position.
280         ITimeswapV2Option(optionPair).transferPosition(
281           timeswapV2TokenPosition.strike,
282           timeswapV2TokenPosition.maturity,
283           param.long1To,
284           TimeswapV2OptionPosition.Short,
285           param.shortAmount
286         );
287       }
288
289       // stop the guard of reentrancy
290       lowerGuard(param.token0, param.token1, param.strike, param.maturity);
291     }
```

Listing 3.2: TimeswapV2Token::burn()

Recommendation Send the short position to the right recipient when the `TimeswapV2` tokens representing short position have been burnt.

Status This issue has been fixed.

3.3 Incorrect Long Position Receiver in TimeswapV2PeripheryTransform

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `TimeswapV2PeripheryTransform`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

As mentioned in Section 3.1, the `Timeswap Constant Sum Option` gives the option owners the right but not the obligation to withdraw and deposit tokens from the option pool at any time and at however many times the owners want, for a fixed duration, as long as the resulting locked tokens follow a specific formula. The `TimeswapV2PeripheryUniswapV3Transform` contract provides an external `transform()` function to facilitate the token swap. While examining the routine, we notice the long position receiver is set incorrectly.

To elaborate, we show below the code snippet of the abstract implementation for the `transform()` function. It comes to our attention that the swapped long position is sent to the wrong recipient.

Specifically, the swapped long position should be sent to `address(this)`, instead of current `param.longTo` (line 66). The `TimeswapV2PeripheryUniswapV3Transform` contract will then send the swapped long position to the `TimeswapV2Token` contract and mint the corresponding amount of `TimeswapV2` tokens for the `param.longTo`.

```
49  function transform(TimeswapV2PeripheryTransformParam memory param)
50    internal
51    returns (
52      uint256 token0AndLong0Amount,
53      uint256 token1AndLong1Amount,
54      bytes memory data
55    )
56  {
57    address optionPair = OptionFactoryLibrary.getWithCheck(optionFactory, param.token0,
          param.token1);
58
59    data = abi.encode(param.token0, param.token1, param.tokenTo, param.longTo, param.
          data);
60
61    (token0AndLong0Amount, token1AndLong1Amount, data) = ITimeswapV2Option(optionPair).
          swap(
62      TimeswapV2OptionSwapParam({
63        strike: param.strike,
64        maturity: param.maturity,
65        tokenTo: param.tokenTo,
66        longTo: param.longTo,
67        isLong0ToLong1: param.isLong0ToLong1,
68        transaction: param.isLong0ToLong1
69          ? TimeswapV2OptionSwap.GivenToken0AndLong0
70          : TimeswapV2OptionSwap.GivenToken1AndLong1,
71        amount: param.positionAmount,
72        data: data
73      })
74    );
75  }
```

Listing 3.3: TimeswapV2PeripheryTransform::transform()

Recommendation Set the correct recipient for the swapped long position.

Status This issue has been fixed.

3.4 Revisited Logic in PeripheryUniswapV3Transform

- ID: PVE-004
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `TimeswapV2PeripheryUniswapV3Transform`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

As mentioned in Section 3.3, the `TimeswapV2PeripheryUniswapV3Transform` contract provides an external `transform()` function to facilitate the token swap. While reviewing its logic, we notice the current implementation needs to be revisited.

In the following, we show the related code snippet of the `transform()` routine. The correct approach needs to firstly burn the `TimeswapV2` tokens from the `msg.sender` and then transfer the long position from the `TimeswapV2Token` contract to the `TimeswapV2PeripheryUniswapV3Transform` contact. Otherwise, the option swap operation in `TimeswapV2PeripheryTransform::transform()` (lines 61-74) will revert as the swap operation needs to deduct the long position amount from the caller of the `ITimeswapV2Option(optionPair).swap()` function, i.e., the `TimeswapV2PeripheryUniswapV3Transform` contract.

```
49  function transform(TimeswapV2PeripheryTransformParam memory param)
50    internal
51    returns (
52      uint256 token0AndLong0Amount,
53      uint256 token1AndLong1Amount,
54      bytes memory data
55    )
56  {
57    address optionPair = OptionFactoryLibrary.getWithCheck(optionFactory, param.token0,
          param.token1);
58
59    data = abi.encode(param.token0, param.token1, param.tokenTo, param.longTo, param.
          data);
60
61    (token0AndLong0Amount, token1AndLong1Amount, data) = ITimeswapV2Option(optionPair).
          swap(
62      TimeswapV2OptionSwapParam({
63        strike: param.strike,
64        maturity: param.maturity,
65        tokenTo: param.tokenTo,
66        longTo: param.longTo,
67        isLong0ToLong1: param.isLong0ToLong1,
68        transaction: param.isLong0ToLong1
```

```
69              ? TimeswapV2OptionSwap.GivenToken0AndLong0
70              : TimeswapV2OptionSwap.GivenToken1AndLong1,
71            amount: param.positionAmount,
72            data: data
73          })
74      );
75  }
```

Listing 3.4: TimeswapV2PeripheryTransform::transform()

Recommendation Add necessary validation for the above-mentioned function.

Status This issue has been fixed.

3.5 Incorrect Implementation Logic in timeswapV2PeripheryTransformInternal()

- ID: PVE-005
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `TimeswapV2PeripheryUniswapV3Transform`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

The `Timeswap V2` periphery contracts take advantage of the `UniswapV3 DEX` to swap tokens whenever necessary for improved fluidity and capital efficiency of the transactions in `Timeswap`. One example is the `TimeswapV2PeripheryUniswapV3Transform` contract. It implements a helper routine, i.e., `timeswapV2PeripheryTransformInternal()`, which utilizes the `UniswapV3` to swap between token0 and token1 if the swap can bring more benefits for the user. While reviewing its logic, we notice the current implementation is not correct.

In the following, we show the related code snippet of the `timeswapV2PeripheryTransformInternal()` routine. Specifically, the unsafe sub execution in line 139 will overflow as this operation will only be executed if `tokenAmountIn > (isToken0 ? param.token0AndLong0Amount : param.token1AndLong1Amount)` (line 136).

```
95  function timeswapV2PeripheryTransformInternal(
        TimeswapV2PeripheryTransformInternalParam memory param)
96      internal
97      override
98      returns (bytes memory data)
99  {
```

```
100     (address msgSender, uint24 uniswapV3Fee, address tokenTo, bool isToken0) = abi.
            decode(
101       param.data,
102       (address, uint24, address, bool)
103     );
104
105     address pool = UniswapV3FactoryLibrary.getWithCheck(uniswapV3Factory, param.token0,
            param.token1, uniswapV3Fee);
106
107     data = abi.encode(
108       CacheForUniswapV3SwapCallback(
109         msgSender,
110         msg.sender,
111         param.token0,
112         param.token1,
113         uniswapV3Fee,
114         tokenTo,
115         isToken0,
116         isToken0 ? param.token0AndLong0Amount : param.token1AndLong1Amount
117       )
118     );
119
120     uint256 tokenAmountIn;
121     uint256 tokenAmountOut;
122     (tokenAmountIn, tokenAmountOut) = pool.swap(
123       UniswapV3SwapParam({
124         recipient: isToken0 == param.isLong0ToLong1 ? param.optionPair : address(this),
125         zeroForOne: param.isLong0ToLong1,
126         exactInput: isToken0 != param.isLong0ToLong1,
127         amount: isToken0 ? param.token1AndLong1Amount : param.token0AndLong0Amount,
128         strikeLimit: 0,
129         data: data
130       })
131     );
132
133     uint256 tokenAmount;
134     bool hasTokenDeposit;
135     if (isToken0 == param.isLong0ToLong1) {
136       if (tokenAmountIn > (isToken0 ? param.token0AndLong0Amount : param.
              token1AndLong1Amount)) {
137         hasTokenDeposit = true;
138         tokenAmount = (isToken0 ? param.token0AndLong0Amount : param.
                token1AndLong1Amount).unsafeSub(tokenAmountIn);
139       } else tokenAmount = tokenAmountIn.unsafeSub(isToken0 ? param.token0AndLong0Amount
              : param.token1AndLong1Amount);
140     } else {
141       if (tokenAmountOut < (isToken0 ? param.token0AndLong0Amount : param.
              token1AndLong1Amount)) {
142         hasTokenDeposit = true;
143         tokenAmount = (isToken0 ? param.token0AndLong0Amount : param.
                token1AndLong1Amount).unsafeSub(tokenAmountOut);
144       } else tokenAmount = tokenAmountOut.unsafeSub(isToken0 ? param.
```

```
            token0AndLong0Amount : param.token1AndLong1Amount);
145     }
146
147     data = abi.encode(hasTokenDeposit, tokenAmount);
148   }
```

Listing 3.5: TimeswapV2PeripheryUniswapV3Transform::timeswapV2PeripheryTransformInternal()

Recommendation Revisit the implementation of the above-mentioned routine.

Status This issue has been fixed in the following commit: `pull393`

3.6 Incorrect token0 Receiver in PeripheryUniswapV3Redeem::redeem()

- ID: PVE-006
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `TimeswapV2PeripheryUniswapV3Redeem`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

In `Timeswap V2`, if a user owns both long and short option positions, he/she can burn both to withdraw asset from the option pool before maturity. The `TimeswapV2PeripheryUniswapV3Transform` contract provides an external `redeem()` function to facilitate the asset withdrawal for users. While examining the routine, we notice the current receiver is set incorrectly.

To elaborate, we show below the code snippet of the `redeem()` function. It comes to our attention that the recipient of the withdrawed `token0` is set incorrectly (line 72). Specifically, the withdrawed `token0` should be sent to `address(this)` if `param.isToken0 == false` and sent to `param.to` if `param.isToken0 == true`. If the withdrawed `token0` is sent to `param.to` when `param.isToken0 == false`, the execution of the `swapGetTotalToken()` routine (lines 79-89) will revert.

```
63  function redeem(TimeswapV2PeripheryUniswapV3RedeemParam calldata param) external
        returns (uint256 tokenAmount) {
64    ...
65
66    redeem(
67      TimeswapV2PeripheryRedeemParam({
68        token0: param.token0,
69        token1: param.token1,
70        strike: param.strike,
71        maturity: param.maturity,
```

```
72          token0To: param.to,
73          token1To: address(this),
74          token0AndLong0Amount: param.token0AndLong0Amount,
75          token1AndLong1Amount: param.token1AndLong1Amount
76        })
77      );
78
79      tokenAmount = swapGetTotalToken(
80        param.token0,
81        param.token1,
82        param.strike,
83        param.uniswapV3Fee,
84        param.to,
85        param.isToken0,
86        param.token0AndLong0Amount,
87        param.token1AndLong1Amount,
88        true
89      );
90
91      if (tokenAmount < param.minTokenAmount) revert MinTokenReached(tokenAmount, param.
          minTokenAmount);
92    }
```

Listing 3.6: TimeswapV2PeripheryUniswapV3Redeem::redeem()

Recommendation Set the correct recipient for the withdrawed `token0` asset.

Status This issue has been fixed.

3.7 Incorrect Rebalance Input in PeripheryUniswapV3Rebalance::rebalance()

- ID: PVE-007
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `TimeswapV2PeripheryUniswapV3Rebalance`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

In `Timeswap V2`, each `AMM` pool has a fixed transaction price (`K`) or strike price denominated as `token1` over `token0`. The transition price determines the risk profile of a `Timeswap` pool. `Timeswap V2` incentivizes arbitrageurs to withdraw all the long `token0` or long `token1` from the pool, whichever has higher value, and replace it with `K` converted equivalent of the other which has lower value. The

TimeswapV2PeripheryUniswapV3Rebalance contract provides an external rebalance() function to facilitate this arbitrage behavior for users. While reviewing its logic, we notice the seventh input argument isLong0ToLong1 for the abstract implementation of the rebalance() function is not correctly set (line 90).

To elaborate, we show below the related code snippet of the redeem() function. Specifically, the input argument isLong0ToLong1 should be set as !zeroForOne, instead of current zeroForOne. If the input argument isLong0ToLong1 is set to true, it means the function caller want to deposit long0 to receive long1.

```
61  function rebalance(TimeswapV2PeripheryUniswapV3RebalanceParam calldata param)
62    external
63    override
64    returns (uint256 tokenAmount, uint256 excessShortAmount)
65  {
66    ...
67
68    (bool zeroForOne, uint256 tokenAmountIn, uint256 tokenAmountOut) = pool.
          calculateSwapForRebalance(
69      UniswapV3CalculateSwapForRebalanceParam({
70        uniswapV3Fee: param.uniswapV3Fee,
71        token0Amount: token0Balance,
72        token1Amount: token1Balance,
73        strikeLimit: param.strike,
74        transactionFee: transactionFee
75      })
76    );
77
78    if (tokenAmountOut == 0) revert NoRebalanceProfit();
79
80    bytes memory data = abi.encode(param.uniswapV3Fee, param.tokenTo, param.isToken0,
          transactionFee);
81
82    (, , excessShortAmount, data) = rebalance(
83      TimeswapV2PeripheryRebalanceParam({
84        token0: param.token0,
85        token1: param.token1,
86        strike: param.strike,
87        maturity: param.maturity,
88        tokenTo: address(this),
89        excessShortTo: param.excessShortTo,
90        isLong0ToLong1: zeroForOne,
91        tokenAmount: tokenAmountIn,
92        data: data
93      })
94    );
95
96    tokenAmount = abi.decode(data, (uint256));
97
98    if (tokenAmount < param.minTokenAmount) revert MinTokenReached(tokenAmount, param.
          minTokenAmount);
```

```
99
100     if (excessShortAmount < param.minExcessShortAmount)
101       revert MinExcessShortReached(excessShortAmount, param.minExcessShortAmount);
102   }
```

Listing 3.7: TimeswapV2PeripheryUniswapV3Rebalance::rebalance()

Recommendation Set the correct input argument for the the abstract implementation of the `rebalance()` function.

Status This issue has been fixed.

3.8 Improved Sanity Checks in TimeswapV2Pool::transferFees()

- ID: PVE-008
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `TimeswapV2Pool`
- Category: Coding Practices [3]
- CWE subcategory: CWE-1126 [1]

Description

In the `TimeswapV2Pool` contract, the `newReward()` function allows for the liquidity provider to transfer his/her earned fees to another address. While reviewing the implementation of this routine, we notice that it can benefit from additional sanity checks.

To elaborate, we show below the code snippet of the `transferFees()` function. Specifically, there is a lack of liquidity verification for the input pool. If the `pool.liquidity` is equal to 0, the `updateDurationWeightBeforeMaturity()/updateDurationWeightAfterMaturity()` execution will revert (lines 244-245)

```
234   function transferFees(
235     Pool storage pool,
236     uint256 maturity,
237     address to,
238     uint256 long0Fees,
239     uint256 long1Fees,
240     uint256 shortFees,
241     uint96 blockTimestamp
242   ) external {
243     // Update the state of the pool first for the short fee growth.
244     if (maturity > blockTimestamp) updateDurationWeightBeforeMaturity(pool,
          blockTimestamp);
245     else if (pool.lastTimestamp < maturity) updateDurationWeightAfterMaturity(pool,
          maturity, blockTimestamp);
246
```

```
247     // Update the fee growth and fees of msg.sender.
248     LiquidityPosition storage liquidityPosition = pool.liquidityPositions[msg.sender];
249
250     liquidityPosition.update(pool.long0FeeGrowth, pool.long1FeeGrowth, pool.
            shortFeeGrowth);
251     liquidityPosition.burnFees(long0Fees, long1Fees, shortFees);
252
253     // Update the fee growth and fees of recipient.
254     liquidityPosition = pool.liquidityPositions[to];
255
256     liquidityPosition.update(pool.long0FeeGrowth, pool.long1FeeGrowth, pool.
            shortFeeGrowth);
257     liquidityPosition.mintFees(long0Fees, long1Fees, shortFees);
258   }
```

Listing 3.8: `Pool::transferFees()`

Recommendation Only execute the `updateDurationWeight` operation when the `pool.liquidity` is greater than 0 in the above mentioned function.

Status This issue has been fixed.

3.9 Improved Implementation Logic in borrowGivenPrincipal()

- ID: PVE-009
- Severity: Low
- Likelihood: Low
- Impact: Low
- Target: `TimeswapV2PeripheryUniswapV3BorrowGivenPrincipal`
- Category: Business Logic [4]
- CWE subcategory: CWE-841 [2]

Description

Within the `Timeswap V2` protocol, the `TimeswapV2PeripheryUniswapV3BorrowGivenPrincipal` contract provides an external `borrowGivenPrincipal()` function to facilitate users borrowing long position from the `Timeswap` pool with the given principal. While examining the routine, we notice the current implementation logic can be improved.

To elaborate, we show below the related code snippet. It comes to our attention that the calculated result of the `tokenAmountOut` variable might be greater than the input argument `param.tokenAmount`, thus causing the execution of the `borrowGivenPrincipal()` function to revert (line 105).

```
72   function borrowGivenPrincipal(TimeswapV2PeripheryUniswapV3BorrowGivenPrincipalParam
         calldata param)
73     external
74     override
```

```
75      returns (uint256 positionAmount)
76    {
77      ...
78      bool exactInput;
79      bool removeStrikeLimit;
80      uint256 tokenAmountIn;
81      uint256 tokenAmountOut;
82      if ((param.isToken0 ? token1Balance : token0Balance) != 0) {
83        (tokenAmountIn, tokenAmountOut) = pool.calculateSwap(
84          UniswapV3CalculateSwapParam({
85            uniswapV3Fee: param.uniswapV3Fee,
86            zeroForOne: !param.isToken0,
87            exactInput: false,
88            amount: param.tokenAmount,
89            strikeLimit: param.strike
90          })
91        );
92
93        if (tokenAmountIn > (param.isToken0 ? token1Balance : token0Balance))
94          (tokenAmountIn, tokenAmountOut) = pool.calculateSwap(
95            UniswapV3CalculateSwapParam({
96              uniswapV3Fee: param.uniswapV3Fee,
97              zeroForOne: !param.isToken0,
98              exactInput: (exactInput = true),
99              amount: param.isToken0 ? token1Balance : token0Balance,
100             strikeLimit: param.strike
101           })
102       );
103   }
104
105   if (param.tokenAmount - tokenAmountOut > (param.isToken0 ? token0Balance :
          token1Balance)) {
106     removeStrikeLimit = true;
107
108     UniswapV3CalculateSwapParam memory internalParam = UniswapV3CalculateSwapParam({
109       uniswapV3Fee: param.uniswapV3Fee,
110       zeroForOne: !param.isToken0,
111       exactInput: (exactInput = false),
112       amount: param.tokenAmount - (param.isToken0 ? token0Balance : token1Balance),
113       strikeLimit: 0
114     });
115
116     (tokenAmountIn, tokenAmountOut) = pool.calculateSwap(internalParam);
117   }
118   ...
119 }
```

Listing 3.9: TimeswapV2PeripheryUniswapV3BorrowGivenPrincipal::borrowGivenPrincipal()

Recommendation Revise the above branch condition (line 105). An example revision is shown as follows:

```
72  function borrowGivenPrincipal(TimeswapV2PeripheryUniswapV3BorrowGivenPrincipalParam
        calldata param)
73    external
74    override
75    returns (uint256 positionAmount)
76  {
77    ...
78
79    if (param.tokenAmount > (param.isToken0 ? token0Balance : token1Balance) +
          tokenAmountOut) {
80      removeStrikeLimit = true;
81
82      UniswapV3CalculateSwapParam memory internalParam = UniswapV3CalculateSwapParam({
83        uniswapV3Fee: param.uniswapV3Fee,
84        zeroForOne: !param.isToken0,
85        exactInput: (exactInput = false),
86        amount: param.tokenAmount - (param.isToken0 ? token0Balance : token1Balance),
87        strikeLimit: 0
88      });
89
90      (tokenAmountIn, tokenAmountOut) = pool.calculateSwap(internalParam);
91    }
92    ...
93  }
```

Listing 3.10: TimeswapV2PeripheryUniswapV3BorrowGivenPrincipal::borrowGivenPrincipal()

Status This issue has been addressed as the `Timeswap` teams confirms that the calculated `tokenAmountOut` will be always less than or equal to the `param.tokenAmount`.

4 Conclusion

In this audit, we have analyzed the design and implementation of the Timeswap V2 protocol. Timeswap V2 is an improved version of Timeswap V1 and increases the capital efficiency of the AMM by 4x–5x while still maintaining all the base layer properties such as permissionless, oracleless and being highly capital efficient. The current code base is well structured and neatly organized. Those identified issues are promptly confirmed and addressed.

Meanwhile, we need to emphasize that Solidity-based 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-1126: Declaration of Variable with Unnecessarily Wide Scope. https://cwe.mitre.org/data/definitions/1126.html.

[2] MITRE. CWE-841: Improper Enforcement of Behavioral Workflow. https://cwe.mitre.org/data/definitions/841.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.