SMART CONTRACT AUDIT REPORT

for

SET PROTOCOL

Prepared By: Shuxiao Wang

Hangzhou, China

Jan. 8, 2020

Document Properties

Client	Set Protocol
Title	Smart Contract Audit Report
Target	RBSetV2
Version	1.0
Author	Chiachih Wu
Auditors	Chiachih Wu, Xuxian Jiang
Reviewed by	Jeff Liu
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	Jan. 8, 2020	Chiachih Wu	Final Release
0.3	Jan. 8, 2020	Chiachih Wu	Status Update
0.2	Dec. 30, 2019	Chiachih Wu	Status Update, Add Findings
0.1	Dec. 23, 2019	Chiachih Wu	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 **5**
- 1.1 About RBSetV2 . . . 5
- 1.2 About PeckShield . . . 6
- 1.3 Methodology . . . 6
- 1.4 Disclaimer . . . 8

2 Findings **10**
- 2.1 Summary . . . 10
- 2.2 Key Findings . . . 11

3 Detailed Results **12**
- 3.1 Business Logic Error in FailRebalance . . . 12
- 3.2 Missing SetToken Whitelist . . . 13
- 3.3 approve()/transferFrom() Race Condition . . . 14
- 3.4 Rounding Error in startRebalance . . . 16
- 3.5 Upfront Check of Bid Quantity . . . 18
- 3.6 Optimization Suggestions . . . 20
- 3.7 Excessive Contract Inheritance . . . 23
- 3.8 Improved Gas Efficiency in CoreAdmin . . . 24
- 3.9 Better Handling of Manager Privilege Transfer . . . 26
- 3.10 Centralized Governance . . . 28
- 3.11 Misleading Embedded Code Comments . . . 29
- 3.12 Misleading Assertion Messages . . . 32
- 3.13 Misleading Function Header Comments . . . 34
- 3.14 Excessive Function Returns . . . 37
- 3.15 Missing Owner Check For Sufficient Allowance . . . 39
- 3.16 Redundant State Checks . . . 41
- 3.17 Missing Elapsed Time Check in LinearAuction . . . 43
- 3.18 Other Suggestions . . . 44

4 Conclusion 45

5 Appendix 46
5.1 Basic Coding Bugs . . . 46
5.1.1 Constructor Mismatch . . . 46
5.1.2 Ownership Takeover . . . 46
5.1.3 Redundant Fallback Function . . . 46
5.1.4 Overflows & Underflows . . . 46
5.1.5 Reentrancy . . . 47
5.1.6 Money-Giving Bug . . . 47
5.1.7 Blackhole . . . 47
5.1.8 Unauthorized Self-Destruct . . . 47
5.1.9 Revert DoS . . . 47
5.1.10 Unchecked External `Call` . . . 48
5.1.11 Gasless `Send` . . . 48
5.1.12 `Send` Instead Of `Transfer` . . . 48
5.1.13 Costly Loop . . . 48
5.1.14 (Unsafe) Use Of Untrusted Libraries . . . 48
5.1.15 (Unsafe) Use Of Predictable Variables . . . 49
5.1.16 Transaction Ordering Dependence . . . 49
5.1.17 Deprecated Uses . . . 49
5.2 Semantic Consistency Checks . . . 49
5.3 Additional Recommendations . . . 49
5.3.1 Avoid Use of Variadic Byte Array . . . 49
5.3.2 Use Fixed Compiler Version . . . 50
5.3.3 Make Visibility Level Explicit . . . 50
5.3.4 Make Type Inference Explicit . . . 50
5.3.5 Adhere To Function Declaration Strictly . . . 50

References 51

1 Introduction

Given the opportunity to review the **RBSetV2** design document and related smart contract source code, we in the report outline 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 RBSetV2

Set Protocol is an asset management protocol that allows the creation of ERC20 tokens that can periodically rebalance using various trade execution strategies. With Rebalancing Sets V1 (RBSetV1), the rebalance could be done through a proposal and dutch auction mechanism. Rebalancing Set V2 (RBSetV2) is an enhanced version of the original Rebalancing SetToken. The enhancements include enabling of fee collection to a fee recipient, simplification of the rebalancing flow by removing the proposal phase, and enabling alternative types of trade execution through singleton liquidator smart contracts.

The basic information of RBSetV2 is as follows:

Table 1.1: Basic Information of RBSetV2

Item	Description
Issuer	Set Protocol
Website	https://www.tokensets.com
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	Jan. 8, 2020

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

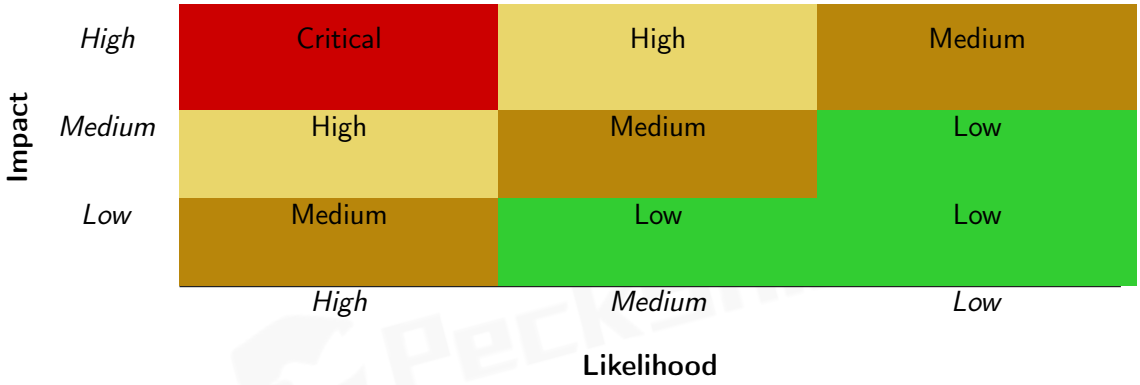
this audit:

- https://github.com/SetProtocol/set-protocol-contracts.git (3daac88)
- https://github.com/SetProtocol/set-protocol-contracts.git (e0e8124)
- https://github.com/SetProtocol/set-protocol-contracts.git (e176af3)

1.2 About PeckShield

PeckShield Inc. [25] 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 [20]:

- 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.

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

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 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) [19], 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 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 an 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 RBSetV2 implementation. During the first phase of our audit, we studied the smart contract source code 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 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	0	
High	1	■
Medium	1	■
Low	3	■■■
Informational	12	■■■■■■■■■■■■
Total	17	

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 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 high-severity vulnerability, 1 medium-severity vulnerability, 3 low-severity vulnerabilities, and 12 informational recommendations.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	High	Business Logic Error in FailRebalance	Coding Practices	Fixed
PVE-002	Low	Missing SetToken Whitelist	Error Conditions	Confirmed
PVE-003	Low	approve()/transferFrom() Race Condition	Time and State	Confirmed
PVE-004	Medium	Rounding Error in startRebalance	Numeric Errors	Confirmed
PVE-005	Info	Upfront Check of Bid Quantity	Coding Practices	Confirmed
PVE-006	Info	Optimization Suggestions	Coding Practices	Fixed
PVE-007	Info	Excessive Contract Inheritance	Coding Practices	Confirmed
PVE-008	Info	Improved Gas Efficiency in CoreAdmin	Coding Practices	Confirmed
PVE-009	Info	Better Handling of Ownership Transfer	Security Features	Confirmed
PVE-010	Info	Centralized Governance	Security Features	Confirmed
PVE-011	Info	Misleading Embedded Code Comments	Coding Practices	Confirmed
PVE-012	Info	Misleading Assertion Messages	Coding Practices	Fixed
PVE-013	Info	Misleading Function Header Comments	Coding Practices	Confirmed
PVE-014	Info	Excessive Function Returns	Coding Practices	Fixed
PVE-015	Low	Missing Owner Check for Sufficient Allowance	Behavioral Problems	Confirmed
PVE-016	Info	Redundant State Checks	Security Features	Confirmed
PVE-017	Info	Missing Elapsed Time Check in LinearAuction	Behavioral Problems	Fixed

Please refer to Section 3 for details.

3 Detailed Results

3.1 Business Logic Error in FailRebalance

- ID: PVE-001
- Severity: High
- Likelihood: Medium
- Impact: High
- Target: `FailRebalance.sol`
- Category: Coding Practices [15]
- CWE subcategory: CWE-1068 [5]

Description

While doing state transaction between the four possible states throughout the Rebalancing Set life cycle, the `transitionToNewState()` is invoked with `_newRebalanceState` as the destination state. As a fail-over mechanism, `reissueSetIfRevertToDefault()` is called in line 90 to handle the case of revert to default.

```
85      function transitionToNewState(
86          RebalancingLibrary.State _newRebalanceState
87      )
88          internal
89      {
90          reissueSetIfRevertToDefault(_newRebalanceState);
91
92          setWithdrawComponentsIfDrawdown(_newRebalanceState);
93
94          rebalanceState = _newRebalanceState;
95          rebalanceIndex = rebalanceIndex.add(1);
96          lastRebalanceTimestamp = block.timestamp;
97
98          nextSet = ISetToken(address(0));
99          hasBidded = false;
100     }
```

Listing 3.1: FailRebalance.sol

However, there is a business logic error in `reissueSetIfRevertToDefault()`. The way `issueQuantity` is calculated in line 142 uses the `nextSet` amount instead of the `currentSet` amount. This leads

to a wrong quantity to reissue in line 145, which breaks the rebalancing mechanism. Since the `issueQuantity` of `currentSet` is likely different from `nextSet`, an attacker could exploit this vulnerability to undermine the issuances of both `currentSet` and `nextSet`.

```
136     function reissueSetIfRevertToDefault(
137         RebalancingLibrary.State _newRebalanceState
138     )
139         private
140     {
141         if (_newRebalanceState == RebalancingLibrary.State.Default) {
142             uint256 issueQuantity = calculateNextSetIssueQuantity();
143
144             // If bid not placed, reissue current Set
145             core.issueInVault(
146                 address(currentSet),
147                 issueQuantity
148             );
149         }
150     }
```

Listing 3.2: FailRebalance.sol

Recommendation Calculate `issueQuantity` based on `currentSet`, instead of `nextSet`.

3.2 Missing SetToken Whitelist

- ID: PVE-002
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Target: `SetToken.sol`
- Category: Error Conditions, Return Values, Status Codes [17]
- CWE subcategory: CWE-391 [10]

Description

When creating the Set tokens, we need to have a whitelist mechanism in place to regulate component tokens, instead of allowing users to create Set tokens from arbitrary components. (What if a component token is based on a rebalancing token?) Note that the current implementation did ensure the address of each component is not 0. It, however, only checks one of the ERC20 methods — `decimals`. Even though the `decimals()` call might fail, the Set can still be created with a zero `minDecimals` (line 125). In other words, the ERC20-compatibility validation may not be sufficient.

```
118             (bool success, ) = currentComponent.call(abi.encodeWithSignature("decimals()
                    "));
119             if (success) {
120                 currentDecimals = ERC20Detailed(currentComponent).decimals();
```

```
121                 minDecimals = currentDecimals < minDecimals ? currentDecimals :
                        minDecimals;
122             } else {
123                 // If one of the components does not implement decimals, we assume the
                        worst
124                 // and set minDecimals to 0
125                 minDecimals = 0;
126             }
```

Listing 3.3: SetToken.sol

Besides, the `constructor()` of `SetToken` may validate additional parameters. For example, the `_units` should not exceed the `totalSupply` of the corresponding component token. Currently, the `constructor()` only requires all `_units` greater than zero (line 105).

```
101         for (uint256 i = 0; i < unitCount; i++) {
102             // Check that all units are non-zero
103             uint256 currentUnits = _units[i];
104             require(
105                 currentUnits > 0,
106                 "SetToken.constructor: Units must be positive"
107             );
```

Listing 3.4: SetToken.sol

It is worthwhile to note that while having a whitelist mechanism could solve the problem, the whitelist itself still needs to be validated. This is also applicable to the whitelist mechanism in `RebalancingSetTokenV2`.

Recommendation Implement a whitelist mechanism to validate the components when creating Set tokens.

3.3 approve()/transferFrom() Race Condition

- ID: PVE-003
- Severity: Low
- Likelihood: Low
- Impact: Medium
- Target: `ERC20.sol`
- Category: Time and State [14]
- CWE subcategory: CWE-362 [9]

Description

There is a known race condition issue regarding `approve()` / `transferFrom()` [2]. Specifically, when a user intends to reduce the allowed spending amount previously approved from, say, 10 SET to 1 SET. The previously approved spender might race to transfer the amount you initially approved (the 10 SET) and then additionally spend the new amount you just approved (1 SET). This breaks the

user's intention of restricting the spender to the new amount, **not** the sum of old amount and new amount.

The original Set Protocol white paper has explicitly discussed this issue, and suggested that "When users are approving tokens to the TransferProxy, it is recommended that users utilize the increaseApproval and decreaseApproval non-ERC20 functions on the token versus the traditional approve function." It is worth mentioning that both the Set token implementation `SetToken.sol` and the Rebalancing Set token implementation `RebalancingSetTokenV2.sol` are based on the OpenZeppelin version, i.e., `ERC20` and `ERC20Detailed`. This version unfortunately suffers from the very same race condition.

Specifically, the the OpenZeppelin version of `ERC20` indeed implemented the known workaround (e.g., `increaseApproval()`/`decreaseApproval()`). However, it does not add necessary sanity checks when entering the `approve()` function.

```
74      /**
75       * @dev See {IERC20-approve}.
76       *
77       * Requirements:
78       *
79       * - `spender` cannot be the zero address.
80       */
81      function approve(address spender, uint256 amount) public returns (bool) {
82          _approve(_msgSender(), spender, amount);
83          return true;
84      }
```

Listing 3.5: ERC20.sol

Recommendation Add additional sanity checks in `approve()` besides the given workaround functions `increaseApproval()`/`decreaseApproval()`.

```
74      /**
75       * @dev See {IERC20-approve}.
76       *
77       * Requirements:
78       *
79       * - `spender` cannot be the zero address.
80       */
81      function approve(address spender, uint256 amount) public returns (bool) {
82          require(amount == 0 || _allowances[_msgSender()][spender] == 0);
83          _approve(_msgSender(), spender, amount);
84          return true;
85      }
```

Listing 3.6: Revised ERC20.sol

3.4 Rounding Error in startRebalance

- ID: PVE-004
- Severity: Medium
- Likelihood: High
- Impact: Low
- Target: `startRebalance.sol`
- Category: Numeric Errors [18]
- CWE subcategory: CWE-197 [6]

Description

In both RBSetV1 and RBSetV2, when `startRebalance()` is initiated, the Rebalancing Set Token calls redeem on its balance of existing Set. These components are stored in the `Vault` under the Rebalancing Set Token's custody. However, the balance of existing Set is calculated in a way that might have small leftover amount.

```
295     function redeemCurrentSet(
296         address _currentSet,
297         address _coreAddress,
298         address _vaultAddress
299     )
300         public
301         returns (uint256)
302     {
303         // Get remainingCurrentSets and make it divisible by currentSet natural unit
304         uint256 currentSetBalance = IVault(_vaultAddress).getOwnerBalance(
305             _currentSet,
306             address(this)
307         );
308
309         // Calculates the set's natural unit
310         uint256 currentSetNaturalUnit = ISetToken(_currentSet).naturalUnit();
311
312         // Rounds the redemption quantity to a multiple of the current Set natural unit
                and sets variable
313         uint256 remainingCurrentSets = currentSetBalance.div(currentSetNaturalUnit).mul(
                currentSetNaturalUnit);
314
315         ICore(_coreAddress).redeemInVault(
316             _currentSet,
317             remainingCurrentSets
318         );
319
320         return remainingCurrentSets;
321     }
```

Listing 3.7: StartRebalanceLibrary.sol

In RBSetV1, line 313 shows that `remainingCurrentSets` is calculated as the rounding result of `currentSetBalance` to a multiple of `currentSetNaturalUnit`. Later on, the rounded redemption quantity

is passed into `redeeemInVault()`, which makes `remainingCurrentSets` of `_currentSet` being burned and corresponding components being redeemed. However, the calculation may result in small amount of leftover when `currentSetBalance` is not divisible by `currentSetNaturalUnit`. The leftover tokens would be permanently lost since there is no book-keeping mechanism in place to further track them.

```
140     function startRebalance(
141         ISetToken _nextSet
142     )
143         external
144         onlyManager
145     {
146         StartRebalance.validateStartRebalance(_nextSet);
148         uint256 startingCurrentSetQuantity = StartRebalance.calculateStartingSetQuantity
                ();
150         StartRebalance.redeemCurrentSet(startingCurrentSetQuantity);
152         StartRebalance.liquidatorStartRebalance(_nextSet, startingCurrentSetQuantity);
154         StartRebalance.transitionToRebalance(_nextSet);
155     }
```

Listing 3.8: RebalancingSetTokenV2.sol

In RBSetV2, `startingCurrentSetQuantity` is calculated by `calculateStartingSetQuantity()`.

```
103     function calculateStartingSetQuantity()
104         internal
105         view
106         returns (uint256)
107     {
108         uint256 currentSetBalance = vault.getOwnerBalance(address(currentSet), address(
                this));
109         uint256 currentSetNaturalUnit = currentSet.naturalUnit();
111         // Rounds the redemption quantity to a multiple of the current Set natural unit
112         return currentSetBalance.sub(currentSetBalance.mod(currentSetNaturalUnit));
113     }
```

Listing 3.9: StartRebalance.sol

Inside `calculateStartingSetQuantity()`, the amount `currentSetBalance.mod(currentSetNaturalUnit)` is subtracted from `currentSetBalance`, which also introduces a rounding issue. (And there is no logic in current codebase to deal with the leftover amount.) We point out that `naturalUnit` is controllable by the user who creates the Set. Those leftover components might not be negligible. After the rebalance process, they are forever locked in the old Set contract.

Recommendation Book-keeping all the quantities of components when each user issues an amount of Set. How to redistribute the leftover may subject to various design choices, including

returning back to users while doing rebalance, donating to project-related communities, or simply burning.

3.5 Upfront Check of Bid Quantity

- ID: PVE-005
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `RebalanceAuctionModule.sol`
- Category: Coding Practices [15]
- CWE subcategory: CWE-1068 [5]

Description

During rebalancing, when a user participates in the bidding process, the user has the option of specifying whether the bid allows partial filling. If no partial filling is intended, i.e., `_allowPartialFill == false`, the bid's `_quantity` is directly returned in line 289. However, `_quantity` may not be greater than `remainingCurrentSets` as this condition will be checked later on. It is suggested to move this particular check upfront.

```
263     function calculateExecutionQuantity(
264         address _rebalancingSetToken,
265         uint256 _quantity,
266         bool _allowPartialFill
267     )
268         internal
269         view
270         returns (uint256)
271     {
272         // Make sure the rebalancingSetToken is tracked by Core
273         require(
274             coreInstance.validSets(_rebalancingSetToken),
275             "RebalanceAuctionModule.bid: Invalid or disabled SetToken address"
276         );

278         // Receive bidding parameters of current auction
279         uint256[] memory biddingParameters = IRebalancingSetToken(_rebalancingSetToken).
                getBiddingParameters();
280         uint256 minimumBid = biddingParameters[0];
281         uint256 remainingCurrentSets = biddingParameters[1];

283         if (_allowPartialFill && _quantity > remainingCurrentSets) {
284             // If quantity is greater than remainingCurrentSets round amount to nearest
                    multiple of
285             // minimumBid that is less than remainingCurrentSets
286             uint256 executionQuantity = remainingCurrentSets.div(minimumBid).mul(
                    minimumBid);
287             return executionQuantity;
```

```
288         } else {
289             return _quantity;
290         }
291     }
```

Listing 3.10: RebalanceAuctionModule.sol

Recommendation Validate `_quantity <= remainingCurrentSets` upfront when `_allowPartialFill` is `false`.

```
263     function calculateExecutionQuantity(
264         address _rebalancingSetToken,
265         uint256 _quantity,
266         bool _allowPartialFill
267     )
268         internal
269         view
270         returns (uint256)
271     {
272         // Make sure the rebalancingSetToken is tracked by Core
273         require(
274             coreInstance.validSets(_rebalancingSetToken),
275             "RebalanceAuctionModule.bid: Invalid or disabled SetToken address"
276         );

278         // Receive bidding parameters of current auction
279         uint256[] memory biddingParameters = IRebalancingSetToken(_rebalancingSetToken).
                getBiddingParameters();
280         uint256 minimumBid = biddingParameters[0];
281         uint256 remainingCurrentSets = biddingParameters[1];

283         if (_allowPartialFill && _quantity > remainingCurrentSets) {
284             // If quantity is greater than remainingCurrentSets round amount to nearest
                    multiple of
285             // minimumBid that is less than remainingCurrentSets
286             uint256 executionQuantity = remainingCurrentSets.div(minimumBid).mul(
                    minimumBid);
287             return executionQuantity;
288         } else {
289             require(_quantity <= remainingCurrentSets);
290             return _quantity;
291         }
292     }
```

Listing 3.11: Revised RebalanceAuctionModule.sol

3.6 Optimization Suggestions

- ID: PVE-006
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Vault.sol`, `Issuance.sol`, `KyberNetworkWrapper.sol`
- Category: Coding Practices [15]
- CWE subcategory: CWE-1041 [3]

Description

Case I: The condition `balances[_token][_owner] >= _quantity` is always true when `_quantity == 0`. Therefore, line 139-143 has no effect when `_quantity == 0`. By moving the balance requirement check under `_quantity > 0`, it makes `decrementTokenOwner` consistent with other routines, i.e., `incrementTokenOwner`, `transferBalance`, and `withdrawTo`.

```
131     function decrementTokenOwner(
132         address _token,
133         address _owner,
134         uint256 _quantity
135     )
136         public
137         onlyAuthorized
138     {
139         // Require that user has enough unassociated tokens to withdraw tokens or issue
                Set
140         require(
141             balances[_token][_owner] >= _quantity,
142             "Vault.decrementTokenOwner: Insufficient token balance"
143         );
145         if (_quantity > 0) {
146             // Decrement balances state variable subtracting _quantity to user's token
                    amount
147             balances[_token][_owner] = balances[_token][_owner].sub(_quantity);
148         }
149     }
```

Listing 3.12: Vault.sol

Recommendation Check for insufficient token balance only when `_quantity > 0`.

```
131     function decrementTokenOwner(
132         address _token,
133         address _owner,
134         uint256 _quantity
135     )
136         public
137         onlyAuthorized
138     {
```

```
139         if (_quantity > 0) {
140             // Require that user has enough unassociated tokens to withdraw tokens or
                    issue Set
141             require(
142                 balances[_token][_owner] >= _quantity,
143                 "Vault.decrementTokenOwner: Insufficient token balance"
144             );
146             // Decrement balances state variable subtracting _quantity to user's token
                    amount
147             balances[_token][_owner] = balances[_token][_owner].sub(_quantity);
148         }
149     }
```

Listing 3.13: Vault.sol

Case III: The state checks in line 67-75 could be consolidated into one check for `rebalanceState == RebalancingLibrary.State.Default` since the `Default` state is the only state that allows a valid mint operation.

```
58     function validateMint()
59         internal
60         view
61     {
62         require(
63             msg.sender == address(core),
64             "Mint: Sender must be core"
65         );
67         require(
68             rebalanceState != RebalancingLibrary.State.Rebalance,
69             "Mint: Cannot mint during Rebalance"
70         );
72         require(
73             rebalanceState != RebalancingLibrary.State.Drawdown,
74             "Mint: Cannot mint during Drawdown"
75         );
76     }
```

Listing 3.14: Issuance.sol

Recommendation Consolidate the state checks.

```
58     function validateMint()
59         internal
60         view
61     {
62         require(
63             msg.sender == address(core),
64             "Mint: Sender must be core"
65         );
```

```
67          require(
68              rebalanceState == RebalancingLibrary.State.Default,
69              "Mint: Only mint during Default"
70          );
71      }
```

Listing 3.15: Issuance.sol

Case IV: In `conversionRate`, the lengths of `_sourceTokens`, `_destinationTokens`, and `_quantities` should be the same. Otherwise, the loop in line 103-109 may fail.

```
90      function conversionRate(
91          address[] calldata _sourceTokens,
92          address[] calldata _destinationTokens,
93          uint256[] calldata _quantities
94      )
95          external
96          view
97          returns (uint256[] memory, uint256[] memory)
98      {
99          uint256 rateCount = _sourceTokens.length;
100         uint256[] memory expectedRates = new uint256[](rateCount);
101         uint256[] memory slippageRates = new uint256[](rateCount);

103         for (uint256 i = 0; i < rateCount; i++) {
104             (expectedRates[i], slippageRates[i]) = KyberNetworkProxyInterface(
                    kyberNetworkProxy).getExpectedRate(
105                 _sourceTokens[i],
106                 _destinationTokens[i],
107                 _quantities[i]
108             );
109         }
```

Listing 3.16: KyberNetworkWrapper.sol

Recommendation Validate the lengths of `_sourceTokens`, `_destinationTokens`, and `_quantities`.

3.7 Excessive Contract Inheritance

- ID: PVE-007
- Severity: Informational
- Likelihood: High
- Impact: N/A
- Targets: `CoreAdmin`, `CoreAccounting`, `CoreIssuance`, `RebalancingSetTokenV2`, `FailRebalance`
- Category: Coding Practices [15]
- CWE subcategory: CWE-1041 [3]

Description

The SET protocol followed a well-engineered approach to organize various contracts and external dependencies. Meanwhile, it brings certain complexities when analyzing contract inheritance graph. In Figure 3.1, we show the `Core`-related inheritance graph. The presence of triangle relationship among smart contracts indicates possible redundant inheritance. Note that any excessive inheritance likely introduces unnecessarily convoluted dependency and makes it harder to reason or infer derived function implementations.

In particular, the above inheritance graph indicates that `CoreAdmin` inherits `Ownable`, `CoreState`, and `TimeLockUpgrade`. The `CoreAdmin`'s inheritance from `Ownable` is redundant as `TimeLockUpgrade` also inherits from `Ownable`. Similarly, `CoreAccounting` inherits `CoreState`, `CoreOperationState`, and `ReentrancyGuard`. Its inheritance from `CoreState` is redundant. Moreover, `CoreIssuance` inherits `CoreState`, `CoreOperationState`, and `ReentrancyGuard`. Its inheritance from `CoreState` is also redundant.

Figure 3.1: The `Core` Contract Inheritance Graph

We also performed similar analysis on `RebalancingSetTokenV2` and its inheritance graph is shown in Figure 3.2. The graph reveals the presence of a few more triangle relationships. Specifically, `RebalancingSetTokenV2`'s inheritances from `ERC20`, `SettleRebalance`, and `RebalancingSetState` are redundant and can be safely removed. In addition, `FailRebalance`'s inheritance from `RebalancingSetState` is unnecessary and can be removed as well.

Figure 3.2: The `RebalancingSetTokenV2` Inheritance Graph

Recommendation Remove redundant inheritances from the above identified smart contracts. Specifically, `CoreAdmin` does not need to inherit from `Ownable`; `CoreAccounting` does not need to inherit from `CoreState`; `CoreIssuance` does not need to inherit from `CoreState`; `RebalancingSetTokenV2` does not need to inherit from `ERC20`, `SettleRebalance`, and `RebalancingSetState`. Moreover, `FailRebalance` does not need to inherit from `RebalancingSetState`. The removal of these redundant inheritances simplifies the contract dependency tracking and better reveals the inherited functionalities from parent contracts.

3.8 Improved Gas Efficiency in CoreAdmin

- ID: PVE-008
- Severity: Informational
- Likelihood: High
- Impact: N/A
- Target: `CoreAdmin.sol`
- Category: Coding Practices [15]
- CWE subcategory: None

Description

The SET protocol has a set of well-defined library functions that greatly facilitate the design and implementation. One example is `AddressArrayUtils`, which collects a number of `address[]`-related routines, such as `indexOf`, `contains`, `extend`, `append`, `intersect`, `union`, `difference`, `pop`, `remove`, `hasDuplicate`, and `isEqual`.

The `CoreAdmin` contract makes extensive use of `AddressArrayUtils`. Though quite useful, the use of `append` in this particular contract may unnecessarily waste gas uses. Specifically, the `append` routine, as the name indicates, simply appends a new item to the end of an address array. This operation can be achieved with a more gas-efficient version, i.e., `push`. Note that `push` is an opcode directly supported in Solidity.

As an example, `addFactory` and `removeFactory` are two essential routines that are used to maintain a dynamic list of whitelisted factories in SET protocol. The `addFactory` routine calls `append`, which basically iterates the entire list, makes a memory copy, appends the new item to the end, and then writes back the copy to the storage. Apparently, it is much more gas-efficient to simply use `push`. According to the Solidity 0.5.7 documentation, " Dynamic storage arrays and bytes (not string) have a member function called push that you can use to append an element at the end of the array. The element will be zero-initialised. The function returns the new length."

```
91      function addFactory(
92          address _factory
93      )
94          external
95          onlyOwner
96          timeLockUpgrade
97      {
98          require(
99              !state.validFactories[_factory]
100         );

102         state.validFactories[_factory] = true;

104         state.factories = state.factories.append(_factory);

106         emit FactoryAdded(
107             _factory
108         );
109     }
```

Listing 3.17: CoreAdmin.sol

```
56      /**
57       * Returns the array with a appended to A.
58       * @param A The first array
59       * @param a The value to append
60       * @return Returns A appended by a
61       */
62      function append(address[] memory A, address a) internal pure returns (address[]
            memory) {
63          address[] memory newAddresses = new address[](A.length + 1);
64          for (uint256 i = 0; i < A.length; i++) {
65              newAddresses[i] = A[i];
66          }
67          newAddresses[A.length] = a;
68          return newAddresses;
69      }
```

Listing 3.18: AddressArrayUtils.sol

Recommendation Replace related `append` operations with the `push` version. Note that similar replacement patterns apply to `addFactory`, `addExchange`, `addModule`, `reenableSet`, and `addPriceLibrary`.

```
91      function addFactory(
92          address _factory
93      )
94          external
95          onlyOwner
96          timeLockUpgrade
97      {
98          require(
99              !state.validFactories[_factory]
100         );

102         state.validFactories[_factory] = true;

104         state.factories.push(_factory);

106         emit FactoryAdded(
107             _factory
108         );
109     }
```

Listing 3.19: CoreAdmin.sol

3.9 Better Handling of Manager Privilege Transfer

- ID: PVE-009
- Severity: Informational
- Likelihood: Low
- Impact: N/A
- Targets: `RebalancingSetToken`, `RebalancingSetState`
- Category: Security Features [13]
- CWE subcategory: CWE-282 [8]

Description

Both RBSetV1 and RBSetV2 smart contracts implement a rather basic access control mechanism that allows a privileged account, i.e., `manager`, to be granted exclusive access to typically sensitive functions (e.g., the setting of Liquidator and FeeRecipient). Because of the `manager`-level access and the implications of these sensitive functions, the `manager` account is essential for the protocol-level safety and operation.

Within this contract, a specific function, i.e., `setManager()`, allows for the manager update. However, current implementation achieves its goal within a single transaction. This is reasonable under the assumption that the `_newManager` parameter is always correctly provided. However, in the unlikely situation, when an incorrect `_newManager` is provided, the contract manager may forever lost, which might be devastating for both RBSetV1 and RBSetV2 operation and maintenance.

As a common best practice, instead of achieving the manager update within a single transaction, it is suggested to split the operation into two steps. The first step initiates the manager update intent and the second step accepts and materializes the update. Both steps should be executed in two separate transactions. By doing so, it can greatly alleviate the concern of accidentally transferring the contract manager to an uncontrolled address. In other words, this two step procedure ensures that a manager public key cannot be nominated unless there is an entity that has the corresponding private key. This is intended to prevent unintentional errors in the manager transfer process.

```
491     /*
492      * Set new manager address
493      *
494      * @param  _newManager       The address of the new manager account
495      */
496     function setManager(
497         address _newManager
498     )
499         external
500     {
501         require(
502             msg.sender == manager,
503             "RebalancingSetToken.setManager: Sender must be the manager"
504         );

506         emit NewManagerAdded(_newManager, manager);
507         manager = _newManager;
508     }
```

Listing 3.20: RebalancingSetToken.sol

```
156     /*
157      * Set new manager address.
158      */
159     function setManager(
160         address _newManager
161     )
162         external
163         onlyManager
164     {
165         emit NewManagerAdded(_newManager, manager);
166         manager = _newManager;
167     }
```

Listing 3.21: RebalancingSetState.sol

Recommendation Implement a two-step approach for manager update (or transfer): `setManager()` and `acceptManager()`.

```
156     address public _newManager;

158     /*
```

```
159      * Set new manager address.
160      */
161     function setManager(
162         address _newManager
163     )
164         external
165         onlyManager
166     {
167         require(newManager != address(0) , "setManager: new manager is the zero address"
                );
168         require(newManager != _newManager, "setManager: new manager is the same as
                previous owner");

170         _newManager = newManager;
171     }

173     function acceptManager() public {
174         require(msg.sender == _newManager);

176         emit NewManagerAdded(_newManager, manager);

178         manager = _newManager;
179         _newManager = 0x0;

181     }
```

Listing 3.22: Revised RebalancingSetState.sol

3.10 Centralized Governance

- ID: PVE-010
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Targets: `CoreAdmin`, `OracleWhiteList`
- Category: Security Features [13]
- CWE subcategory: CWE-654 [12]

Description

Throughout the whole RBSetV2 system (inherited from the original RBSetV1), the `_owner` is the account who can access or execute all the privileged functions (via the `onlyOwner` modifier). However, some privileged functions are not necessary to be controlled by the `_owner` account. For example, the access to `addTokenOraclePair/removeTokenOraclePair` or `addFactory/removeFactory` functions could be assigned to an `operator/manager` account or controlled by a multisig account. The current centralized implementation makes this system not compatible to the usual setup of multiple oracles for reduced risks or shared responsibilities.

Note that the original Set Protocol white paper has explicitly discussed this issue and stated that "Set Protocol intends to begin with a centralized governance in the early, formative days and shift over time to a community-based governance system." It will be greatly helpful to materialize necessary plan to have a community-based governance, which could move the system one step further toward ultimate decentralization.

Recommendation Add necessary decentralized mechanisms to reduce or separate overly centralized privileges around `_owner`. In the meantime, develop a long-time plan for eventual community-based governance.

3.11 Misleading Embedded Code Comments

- ID: PVE-011
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: Multiple Contracts
- Category: Coding Practices [15]
- CWE subcategory: CWE-1059 [4]

Description

There are a few misleading comments embedded among lines of solidity code, which make it more difficult to understand and/or maintain the software.

A few example comments can be found in line 108 of `core/TransferProxy::batchTransfer()`, line 203 of `core/Vault::batchWithdrawTo()`, line 243 of `core/Vault::batchIncrementTokenOwner()`, line 283 of `core/Vault::batchDecrementTokenOwner()`, and line 324 of `core/Vault::batchTransferBalance()`.

```
108         // Confirm and empty _tokens array is not passed
109         require(
110             tokenCount > 0,
111             "TransferProxy.batchTransfer: Tokens must not be empty"
112         );
```

Listing 3.23: core/TransferProxy.sol

Recommendation Modify the comments from "`Confirm and empty _tokens`" to "`Confirm an empty _tokens`".

```
108         // Confirm an empty _tokens array is not passed
109         require(
110             tokenCount > 0,
111             "TransferProxy.batchTransfer: Tokens must not be empty"
112         );
```

Listing 3.24: core/TransferProxy.sol

Additional example comments can also be found in line 90 of `core/tokens/rebalancing-v2/StartRebalance::validateStartRebalance()` and line 127 of `core/tokens/rebalancing-libraries/ProposeLibrary::validateProposal()`.

```
90          // Done to make sure that when calculating token units there will are no rounding errors.
91          require(
92              naturalUnitsAreValid(currentSet, _nextSet),
93              "StartRebalance: Invalid natural unit"
94          );
```

Listing 3.25: core/tokens/rebalancing-v2/StartRebalance.sol

```
127         // Done to make sure that when calculating token units there will are no rounding errors.
128         uint256 currentNaturalUnit = ISetToken(_proposalContext.currentSet).naturalUnit();
129         uint256 nextSetNaturalUnit = ISetToken(_nextSet).naturalUnit();
130         require(
131             Math.max(currentNaturalUnit, nextSetNaturalUnit).mod(
132                 Math.min(currentNaturalUnit, nextSetNaturalUnit)
133             ) == 0,
134             "ProposeLibrary.validateProposal: Invalid proposed Set natural unit"
135         );
```

Listing 3.26: core/tokens/rebalancing- libraries /ProposeLibrary.sol

Recommendation Modify the comments from "`will are no rounding errors`" to "`will be no rounding errors`".

```
90          // Done to make sure that when calculating token units there will are no rounding errors.
91          require(
92              naturalUnitsAreValid(currentSet, _nextSet),
93              "StartRebalance: Invalid natural unit"
94          );
```

Listing 3.27: core/tokens/rebalancing-v2/StartRebalance.sol

```
127         // Done to make sure that when calculating token units there will be no rounding errors.
128         uint256 currentNaturalUnit = ISetToken(_proposalContext.currentSet).naturalUnit();
129         uint256 nextSetNaturalUnit = ISetToken(_nextSet).naturalUnit();
130         require(
131             Math.max(currentNaturalUnit, nextSetNaturalUnit).mod(
132                 Math.min(currentNaturalUnit, nextSetNaturalUnit)
133             ) == 0,
134             "ProposeLibrary.validateProposal: Invalid proposed Set natural unit"
135         );
```

Listing 3.28: core/tokens/rebalancing- libraries /ProposeLibrary.sol

More comments can also be found in line 46 of `helper/RebalancingSetEthBidder.sol`.

```
43      // Address and instance of RebalanceAuctionModule contract
44      IRebalanceAuctionModule public rebalanceAuctionModule;
45
46      // Address and instance of RebalanceAuctionModule contract
47      ITransferProxy public transferProxy;
48
49      // Address and instance of Wrapped Ether contract
50      IWETH public weth;
```

Listing 3.29: helper/RebalancingSetEthBidder.sol

Recommendation Modify the comments from "`Address and instance of RebalanceAuctionModule contract`" to "`Address and instance of TransferProxy contract`".

```
43      // Address and instance of RebalanceAuctionModule contract
44      IRebalanceAuctionModule public rebalanceAuctionModule;
45
46      // Address and instance of TransferProxy contract
47      ITransferProxy public transferProxy;
48
49      // Address and instance of Wrapped Ether contract
50      IWETH public weth;
```

Listing 3.30: helper/RebalancingSetEthBidder.sol

More comments can also be found in line 45-46 of `core/lib/ExchangeWrapperLibraryV2.sol` and line 49-50 of `core/lib/ExchangeWrapperLibrary.sol`.

```
44      /**
45       * components                    A list of the acquired components from exchange
             wrapper
46       * componentQuantities           A list of the component quantities acquired
47       */
48      struct ExchangeResults {
49          address[] receiveTokens;
50          uint256[] receiveTokenAmounts;
51      }
52  }
```

Listing 3.31: core/lib/ExchangeWrapperLibraryV2.sol

```
48      /**
49       * components                    A list of the acquired components from exchange
             wrapper
50       * componentQuantities           A list of the component quantities acquired
51       */
52      struct ExchangeResults {
53          address[] receiveTokens;
54          uint256[] receiveTokenAmounts;
55      }
```

Listing 3.32: core/lib/ExchangeWrapperLibrary.sol

Recommendation Modify the comments from "`components` and `componentQuantities`" to "`receiveTokens` and `receiveTokenAmounts`".

```
44      /**
45       * receiveTokens
46       * receiveTokenAmounts
47       */
48      struct ExchangeResults {
49          address[] receiveTokens;
50          uint256[] receiveTokenAmounts;
```

```
51      }
52  }
```

Listing 3.33: core/lib/ExchangeWrapperLibraryV2.sol

```
48      /**
49       * receiveTokens
50       * receiveTokenAmounts
51       */
52      struct ExchangeResults {
53          address[] receiveTokens;
54          uint256[] receiveTokenAmounts;
55      }
```

Listing 3.34: core/lib/ExchangeWrapperLibrary.sol

More comments can also be found in line 76-77 of `core/lib/auction-price-libraries/LinearAuctionPriceCurve.sol` since the value `MIN_PIVOT_PRICE_DIVISOR` is 5 instead of 2 here.

```
76          // Require pivot price to be greater than 0.5 * price denominator
77          // Equivalent to oldSet/newSet = 0.5
78          require(
79              _auctionPriceParameters.auctionPivotPrice > priceDivisor.div(
                    MIN_PIVOT_PRICE_DIVISOR),
80              "LinearAuctionPriceCurve.validateAuctionPriceParameters: Pivot price too low
                    "
81          );
```

Listing 3.35: core/lib/auction-price- libraries /LinearAuctionPriceCurve.sol

```
76          // Require pivot price to be greater than 0.2 * price denominator
77          // Equivalent to oldSet/newSet = 0.2
78          require(
79              _auctionPriceParameters.auctionPivotPrice > priceDivisor.div(
                    MIN_PIVOT_PRICE_DIVISOR),
80              "LinearAuctionPriceCurve.validateAuctionPriceParameters: Pivot price too low
                    "
81          );
```

Listing 3.36: core/lib/auction-price- libraries /LinearAuctionPriceCurve.sol

3.12 Misleading Assertion Messages

- ID: PVE-012
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: Multiple Contracts
- Category: Coding Practices [15]
- CWE subcategory: CWE-1059 [4]

Description

There are a few misleading assertion messages scattered in various places, which make it more difficult to understand and/or maintain the software.

A few example assertion messages can be found in line 154 of `lib/OracleWhiteList::editTokenOraclePair()`.

```
152         require(
153             oracleWhiteList[_tokenAddress] != address(0),
154             "OracleWhiteList.addTokenOraclePair: Token and Oracle pair already exists."
155         );
```

Listing 3.37: lib/OracleWhiteList.sol

Recommendation Modify `addTokenOraclePair` to `editTokenOraclePair`

```
152         require(
153             oracleWhiteList[_tokenAddress] != address(0),
154             "OracleWhiteList.editTokenOraclePair: Token and Oracle pair already exists."
155         );
```

Listing 3.38: lib/OracleWhiteList.sol

Additional example assertion messages can also be found in line 198 of `lib/OracleWhiteList::getOracleAddressesByToken()`.

```
195         // Check that passed array length is greater than 0
196         require(
197             arrayLength > 0,
198             "OracleWhiteList.areValidAddresses: Array length must be greater than 0."
199         );
```

Listing 3.39: lib/OracleWhiteList.sol

Recommendation Modify `areValidAddresses` to `getOracleAddressesByToken`.

```
195         // Check that passed array length is greater than 0
196         require(
197             arrayLength > 0,
198             "OracleWhiteList.getOracleAddressesByToken: Array length must be greater
                than 0."
199         );
```

Listing 3.40: lib/OracleWhiteList.sol

Another example assertion messages is in line 115 of `core/liquidators/impl/LinearAuction::validateRebalanceComponents()`.

```
105     function validateRebalanceComponents(
106         ISetToken _currentSet,
107         ISetToken _nextSet
108     )
```

```
109         internal
110         view
111     {
112         address[] memory combinedTokenArray = Auction.getCombinedTokenArray(_currentSet,
                _nextSet);
113         require(
114             oracleWhiteList.areValidAddresses(combinedTokenArray),
115             "ExponentialPivotAuctionLiquidator.startRebalance: Passed token does not
                    have matching oracle."
116         );
117     }
```

Listing 3.41: core/ liquidators /impl/LinearAuction.sol

Recommendation Modify `ExponentialPivotAuctionLiquidator` to `LinearAuctionLiquidator`.

```
105     function validateRebalanceComponents(
106         ISetToken _currentSet,
107         ISetToken _nextSet
108     )
109         internal
110         view
111     {
112         address[] memory combinedTokenArray = Auction.getCombinedTokenArray(_currentSet,
                _nextSet);
113         require(
114             oracleWhiteList.areValidAddresses(combinedTokenArray),
115             "LinearAuctionLiquidator.startRebalance: Passed token does not have matching
                    oracle."
116         );
117     }
```

Listing 3.42: core/ liquidators /impl/LinearAuction.sol

3.13 Misleading Function Header Comments

- ID: PVE-013
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: Multiple Contracts
- Category: Coding Practices [15]
- CWE subcategory: CWE-1059 [4]

Description

There are a few misleading function header comments scattered in various places, which make it more difficult to understand and/or maintain the software.

For example, the second parameter `_offset` is missing in the function header comments of `core/lib/ExchangeHeaderLibrary::parseExchangeHeader()`.

```
53      /**
54       * Function to convert bytes into ExchangeHeader
55       *
56       * @param _orderData      Bytes representing the order body information
57       * @return ExchangeHeader  Struct containing data for a batch of exchange orders
58       */
59      function parseExchangeHeader(
60          bytes memory _orderData,
61          uint256 _offset
62      )
```

Listing 3.43: core/lib/ExchangeHeaderLibrary.sol

Recommendation Add `_offset` into the function header comments.

```
53      /**
54       * Function to convert bytes into ExchangeHeader
55       *
56       * @param _orderData      Bytes representing the order body information
57       * @param _offet
58       * @return ExchangeHeader  Struct containing data for a batch of exchange orders
59       */
60      function parseExchangeHeader(
61          bytes memory _orderData,
62          uint256 _offset
63      )
```

Listing 3.44: core/lib/ExchangeHeaderLibrary.sol

Another example is located in line 164 of `core/tokens/rebalancing-v2/StartRebalance::naturalUnitsAreValid()`.

```
164     * Done to make sure that when calculating token units there will are no rounding errors.
165     *
166     * @param _currentSet                The current base SetToken
167     * @param _nextSet                   The proposed SetToken
168     */
169    function naturalUnitsAreValid(
170        ISetToken _currentSet,
171        ISetToken _nextSet
172    )
```

Listing 3.45: core/tokens/rebalancing-v2/StartRebalance.sol

Recommendation Modify the comments from "`will are no rounding errors`" to "`will be no rounding errors`".

```
164     * Done to make sure that when calculating token units there will be no rounding errors.
165     *
166     * @param _currentSet                The current base SetToken
167     * @param _nextSet                   The proposed SetToken
168     */
169    function naturalUnitsAreValid(
170        ISetToken _currentSet,
171        ISetToken _nextSet
172    )
```

Listing 3.46: core/tokens/rebalancing-v2/StartRebalance.sol

Another example can also be found in line 205 of `core/tokens/rebalancing-libraries/StartRebalanceLibrary::calculateCombinedUnitArrays()`.

```
204     /**
205      * Create arrays that represents all components in currentSet and nextSet.
206      * Calcualate unit difference between both sets relative to the largest natural
207      * unit of the two sets.
208      *
209      * @param _currentSet               Information on currentSet
210      * @param _nextSet                  Information on nextSet
211      * @param _minimumBid               Minimum bid amount
212      * @param _auctionLibrary           Address of auction library being used in
             rebalance
213      * @param _combinedTokenArray       Array of component tokens involved in rebalance
214      * @return                          Unit inflow/outflow arrays for current and next
             Set
215      */
216     function calculateCombinedUnitArrays(
217         SetTokenLibrary.SetDetails memory _currentSet,
218         SetTokenLibrary.SetDetails memory _nextSet,
219         uint256 _minimumBid,
220         address _auctionLibrary,
221         address[] memory _combinedTokenArray
222     )
```

Listing 3.47: core/tokens/rebalancing– libraries /StartRebalanceLibrary.sol

Recommendation Modify the comments from "`Create arrays that represents`" to "`Create arrays that represent`".

```
204     /**
205      * Create arrays that represent all components in currentSet and nextSet.
206      * Calcualate unit difference between both sets relative to the largest natural
207      * unit of the two sets.
208      *
209      * @param _currentSet               Information on currentSet
210      * @param _nextSet                  Information on nextSet
211      * @param _minimumBid               Minimum bid amount
212      * @param _auctionLibrary           Address of auction library being used in
             rebalance
213      * @param _combinedTokenArray       Array of component tokens involved in rebalance
214      * @return                          Unit inflow/outflow arrays for current and next
             Set
215      */
216     function calculateCombinedUnitArrays(
217         SetTokenLibrary.SetDetails memory _currentSet,
218         SetTokenLibrary.SetDetails memory _nextSet,
219         uint256 _minimumBid,
220         address _auctionLibrary,
221         address[] memory _combinedTokenArray
222     )
```

Listing 3.48: core/tokens/rebalancing– libraries /StartRebalanceLibrary.sol

More similar function header comments can be found in line 261 of `core/modules/RebalanceAuctionModule::calculateExecutionQuantity()`, line 321 of `core/tokens/RebalancingSetToken::placeBid()`, line 161 of `core/tokens/RebalancingSetTokenV2::getBidPrice()`, line 187 of `core/tokens/RebalancingSetTokenV2::placeBid()`, and line 190 of `core/interfaces/IRebalancingSetToken::placeBid()`.

```
261      * @return executionQuantity       Array of token addresses invovled in rebalancing
262      */
263     function calculateExecutionQuantity(
264         address _rebalancingSetToken,
265         uint256 _quantity,
266         bool _allowPartialFill
267     )
```

Listing 3.49: core/modules/RebalanceAuctionModule.sol

Recommendation Modify the comments from "`Array of token addresses invovled in rebalancing`" to "`Array of token addresses involved in rebalancing`".

```
261      * @return executionQuantity       Array of token addresses involved in rebalancing
262      */
263     function calculateExecutionQuantity(
264         address _rebalancingSetToken,
265         uint256 _quantity,
266         bool _allowPartialFill
267     )
```

Listing 3.50: core/modules/RebalanceAuctionModule.sol

3.14 Excessive Function Returns

- ID: PVE-014
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `PlaceBidLibrary.sol`
- Category: Coding Practices [15]
- CWE subcategory: CWE-1041 [3]

Description

Throughout the function `validatePlaceBid()`, multiple `require` statements are used to validate the states and the parameters while placing a bid. Those `require` statements throw exceptions when there is something unexpected. And the entire function does not return any value, which makes the declaration in line 54 and related comments in line 45 – 46 unnecessary.

```
38     /*
39      * Place bid during rebalance auction. Can only be called by Core.
40      *
```

```
41      * @param _quantity                The amount of currentSet to be rebalanced
42      * @param _coreAddress             Core address
43      * @param _biddingParameters       Struct containing relevant data for calculating
            token flows
44      * @return inflowUnitArray         Array of amount of tokens inserted into system
            in bid
45      * @return outflowUnitArray        Array of amount of tokens taken out of system in
             bid
46      */
47     function validatePlaceBid(
48         uint256 _quantity,
49         address _coreAddress,
50         RebalancingLibrary.BiddingParameters memory _biddingParameters
51     )
52         public
53         view
54         returns (uint256)
55     {
56         // Make sure sender is a module
57         require(
58             ICore(_coreAddress).validModules(msg.sender),
59             "RebalancingSetToken.placeBid: Sender must be approved module"
60         );
61
62         // Make sure that bid amount is greater than zero
63         require(
64             _quantity > 0,
65             "RebalancingSetToken.placeBid: Bid must be > 0"
66         );
67
68         // Make sure that bid amount is multiple of minimum bid amount
69         require(
70             _quantity.mod(_biddingParameters.minimumBid) == 0,
71             "RebalancingSetToken.placeBid: Must bid multiple of minimum bid"
72         );
73
74         // Make sure that bid Amount is less than remainingCurrentSets
75         require(
76             _quantity <= _biddingParameters.remainingCurrentSets,
77             "RebalancingSetToken.placeBid: Bid exceeds remaining current sets"
78         );
79     }
```

Listing 3.51: PlaceBidLibrary.sol

Recommendation Remove excessive returns from the function declaration and function header comments.

```
38     /*
39      * Place bid during rebalance auction. Can only be called by Core.
40      *
41      * @param _quantity                The amount of currentSet to be rebalanced
```

```
42      * @param _coreAddress              Core address
43      * @param _biddingParameters        Struct containing relevant data for calculating
            token flows
44      */
45     function validatePlaceBid(
46         uint256 _quantity,
47         address _coreAddress,
48         RebalancingLibrary.BiddingParameters memory _biddingParameters
49     )
50         public
51         view
52     {
53         // Make sure sender is a module
54         require(
55             ICore(_coreAddress).validModules(msg.sender),
56             "RebalancingSetToken.placeBid: Sender must be approved module"
57         );
58
59         // Make sure that bid amount is greater than zero
60         require(
61             _quantity > 0,
62             "RebalancingSetToken.placeBid: Bid must be > 0"
63         );
64
65         // Make sure that bid amount is multiple of minimum bid amount
66         require(
67             _quantity.mod(_biddingParameters.minimumBid) == 0,
68             "RebalancingSetToken.placeBid: Must bid multiple of minimum bid"
69         );
70
71         // Make sure that bid Amount is less than remainingCurrentSets
72         require(
73             _quantity <= _biddingParameters.remainingCurrentSets,
74             "RebalancingSetToken.placeBid: Bid exceeds remaining current sets"
75         );
76     }
```

Listing 3.52: PlaceBidLibrary.sol

3.15 Missing Owner Check For Sufficient Allowance

- ID: PVE-015
- Severity: Low
- Likelihood: High
- Impact: None
- Target: `ERC20Wrapper.sol`
- Category: Behavioral Problems [16]
- CWE subcategory: CWE-440 [11]

Description

In the ERC20 wrapper library implementation, the `ensureAllowance()` function is used to ensure that the `_owner` has granted enough allowance. When the owner has not granted enough allowance, the ERC20 standard operation, `approve()`, is called for granting more allowance (line 166). However, the `approve()` operation can only **allow** the `_spender` to withdraw from the `msg.sender` instead of an arbitrary `_owner`. This wrapper library cannot function properly while `_owner` is not `address(this)`. Fortunately, all the callers of `ensureAllowance()` pass `address(this)` as the `_owner`. The library should check the `_owner` in the case that the allowance is not enough. Otherwise, the wrapper library cannot work as it declares (i.e., ensure the **owner** has granted enough allowance).

```
147     /**
148      * Ensure's the owner has granted enough allowance for system to
149      * transfer tokens.
150      *
151      * @param  _token          The address of the ERC20 token
152      * @param  _owner          The address of the token owner
153      * @param  _spender        The address to grant/check allowance for
154      * @param  _quantity       The amount to see if allowed for
155      */
156     function ensureAllowance(
157         address _token,
158         address _owner,
159         address _spender,
160         uint256 _quantity
161     )
162         internal
163     {
164         uint256 currentAllowance = allowance(_token, _owner, _spender);
165         if (currentAllowance < _quantity) {
166             approve(
167                 _token,
168                 _spender,
169                 CommonMath.maxUInt256()
170             );
171         }
172     }
```

Listing 3.53: ERC20Wrapper.sol

Recommendation Ensure that the `_owner` is `address(this)` before calling `approve()`.

```
147     /**
148      * Ensure's the owner has granted enough allowance for system to
149      * transfer tokens.
150      *
151      * @param  _token          The address of the ERC20 token
152      * @param  _owner          The address of the token owner
153      * @param  _spender        The address to grant/check allowance for
154      * @param  _quantity       The amount to see if allowed for
```

```
155      */
156     function ensureAllowance(
157         address _token,
158         address _owner,
159         address _spender,
160         uint256 _quantity
161     )
162         internal
163     {
164         uint256 currentAllowance = allowance(_token, _owner, _spender);
165         if (currentAllowance < _quantity) {
166             require(_owner == address(this));
167             approve(
168                 _token,
169                 _spender,
170                 CommonMath.maxUInt256()
171             );
172         }
173     }
```

Listing 3.54: ERC20Wrapper.sol

3.16 Redundant State Checks

- ID: PVE-016
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `CoreAccounting.sol`
- Category: Security Features [13]
- CWE subcategory: CWE-269 [7]

Description

In `batchDeposit()`, the `whenOperational` modifier (line 121) is redundant since the underlying handler, `batchDepositInternal()`, has the same modifier applied (line 198). Since `batchDeposit()` does not have anything other than invoking `batchDepositInternal()`, the redundant state check could be removed.

```
115     function batchDeposit(
116         address[] calldata _tokens,
117         uint256[] calldata _quantities
118     )
119         external
120         nonReentrant
121         whenOperational
122     {
123         // Call internal batch deposit function
124         batchDepositInternal(
125             msg.sender,
126             msg.sender,
```

```
127             _tokens,
128             _quantities
129         );
130     }
```

Listing 3.55: CoreAccounting.sol

```
191     function batchDepositInternal(
192         address _from,
193         address _to,
194         address[] memory _tokens,
195         uint256[] memory _quantities
196     )
197         internal
198         whenOperational
```

Listing 3.56: CoreAccounting.sol

Recommendation Remove the redundant modifier in `batchDeposit()`.

```
115     function batchDeposit(
116         address[] calldata _tokens,
117         uint256[] calldata _quantities
118     )
119         external
120         nonReentrant
121     {
122         // Call internal batch deposit function
123         batchDepositInternal(
124             msg.sender,
125             msg.sender,
126             _tokens,
127             _quantities
128         );
129     }
```

Listing 3.57: CoreAccounting.sol

3.17 Missing Elapsed Time Check in LinearAuction

- ID: PVE-017
- Severity: Informational
- Likelihood: Low
- Impact: None
- Target: `LinearAuction`
- Category: Behavioral Problems [16]
- CWE subcategory: CWE-440 [11]

Description

In the LinearAuction implementation, the `getNumerator()` function is used to get the linear auction price based on the current timestamp. However, the linear auction price could exceed the "end price" when timestamp exceeds "endTime". Specifically, when `getNumerator()` happens to be invoked in a timestamp that `elapsed` is greater than `auctionPeriod`, the `elapsedPrice` would be greater than `range` (line 164). Therefore, `_linearAuction.startNumerator.add(elapsedPrice)` would be greater than `_linearAuction.endNumerator`. Fortunately, the callers of `getNumerator()` could check if the specific `_linearAuction` has failed or not by calling the `hasAuctionFailed()` function such that the abnormal auction price would not be used in a real bid. But, the `getNumerator()` should not return a price larger than "end price" as it declares.

```
161     function getNumerator(State storage _linearAuction) internal view returns (uint256)
            {
162         uint256 elapsed = block.timestamp.sub(_linearAuction.auction.startTime);
163         uint256 range = _linearAuction.endNumerator.sub(_linearAuction.startNumerator);
164         uint256 elapsedPrice = elapsed.mul(range).div(auctionPeriod);
165
166         return _linearAuction.startNumerator.add(elapsedPrice);
167     }
```

Listing 3.58: LinearAuction.sol

Recommendation Return `_linearAuction.endNumerator` when `elapsed >= auctionPeriod`.

```
161     function getNumerator(State storage _linearAuction) internal view returns (uint256)
            {
162         uint256 elapsed = block.timestamp.sub(_linearAuction.auction.startTime);
163
164         if ( elapsed >= auctionPeriod ) {
165             return _linearAuction.endNumerator;
166         }
167         else {
168             uint256 range = _linearAuction.endNumerator.sub(_linearAuction.
                    startNumerator);
169             uint256 elapsedPrice = elapsed.mul(range).div(auctionPeriod);
170
171             return _linearAuction.startNumerator.add(elapsedPrice);
```

```
172         }
173     }
```

Listing 3.59: LinearAuction.sol

3.18 Other Suggestions

Due to the fact that compiler upgrades might bring unexpected compatibility or inter-version consistencies, it is always suggested to use fixed compiler versions whenever possible. As an example, we highly encourage to explicitly indicate the Solidity compiler version, e.g., `pragma solidity 0.5.7;` instead of `pragma solidity ^0.5.7;`.

Moreover, we strongly suggest not to use experimental Solidity features or third-party unaudited libraries. If necessary, refactor current code base to only use stable features or trusted libraries. In case there is an absolute need of leveraging experimental features or integrating external libraries, make necessary contingency plans.

4 | Conclusion

In this audit, we thoroughly analyzed the RBSetV2 documentation and implementation. The audited system does involve various intricacies in both design and implementation. The current code base is well organized and those identified issues are promptly confirmed and fixed.

Meanwhile, we need to emphasize that smart contracts 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.

5 Appendix

5.1 Basic Coding Bugs

5.1.1 Constructor Mismatch

- Description: Whether the contract name and its constructor are not identical to each other.
- Result: Not found
- Severity: Critical

5.1.2 Ownership Takeover

- Description: Whether the set owner function is not protected.
- Result: Not found
- Severity: Critical

5.1.3 Redundant Fallback Function

- Description: Whether the contract has a redundant fallback function.
- Result: Not found
- Severity: Critical

5.1.4 Overflows & Underflows

- Description: Whether the contract has general overflow or underflow vulnerabilities [21, 22, 23, 24, 26].
- Result: Not found
- Severity: Critical

5.1.5 Reentrancy

- Description: Reentrancy [27] is an issue when code can call back into your contract and change state, such as withdrawing ETHs.
- Result: Not found
- Severity: Critical

5.1.6 Money-Giving Bug

- Description: Whether the contract returns funds to an arbitrary address.
- Result: Not found
- Severity: High

5.1.7 Blackhole

- Description: Whether the contract locks ETH indefinitely: merely in without out.
- Result: Not found
- Severity: High

5.1.8 Unauthorized Self-Destruct

- Description: Whether the contract can be killed by any arbitrary address.
- Result: Not found
- Severity: Medium

5.1.9 Revert DoS

- Description: Whether the contract is vulnerable to DoS attack because of unexpected `revert`.
- Result: Not found
- Severity: Medium

5.1.10 Unchecked External `Call`

- Description: Whether the contract has any external `call` without checking the return value.
- Result: Not found
- Severity: Medium

5.1.11 Gasless `Send`

- Description: Whether the contract is vulnerable to gasless `send`.
- Result: Not found
- Severity: Medium

5.1.12 `Send` Instead Of `Transfer`

- Description: Whether the contract uses `send` instead of `transfer`.
- Result: Not found
- Severity: Medium

5.1.13 Costly Loop

- Description: Whether the contract has any costly loop which may lead to `Out-Of-Gas` exception.
- Result: Not found
- Severity: Medium

5.1.14 (Unsafe) Use Of Untrusted Libraries

- Description: Whether the contract use any suspicious libraries.
- Result: Not found
- Severity: Medium

5.1.15 (Unsafe) Use Of Predictable Variables

- Description: Whether the contract contains any randomness variable, but its value can be predicated.
- Result: Not found
- Severity: Medium

5.1.16 Transaction Ordering Dependence

- Description: Whether the final state of the contract depends on the order of the transactions.
- Result: Not found
- Severity: Medium

5.1.17 Deprecated Uses

- Description: Whether the contract use the deprecated `tx.origin` to perform the authorization.
- Result: Not found
- Severity: Medium

5.2 Semantic Consistency Checks

- Description: Whether the semantic of the white paper is different from the implementation of the contract.
- Result: Not found
- Severity: Critical

5.3 Additional Recommendations

5.3.1 Avoid Use of Variadic Byte Array

- Description: Use fixed-size byte array is better than that of `byte[]`, as the latter is a waste of space.
- Result: Not found
- Severity: Low

5.3.2 Use Fixed Compiler Version

- Description: Use fixed compiler version is better.
- Result: Not found
- Severity: Low

5.3.3 Make Visibility Level Explicit

- Description: Assign explicit visibility specifiers for functions and state variables.
- Result: Not found
- Severity: Low

5.3.4 Make Type Inference Explicit

- Description: Do not use keyword `var` to specify the type, i.e., it asks the compiler to deduce the type, which is not safe especially in a loop.
- Result: Not found
- Severity: Low

5.3.5 Adhere To Function Declaration Strictly

- Description: Solidity compiler (version 0.4.23) enforces strict ABI length checks for return data from `calls()` [1], which may break the the execution if the function implementation does NOT follow its declaration (e.g., no return in implementing `transfer()` of ERC20 tokens).
- Result: Not found
- Severity: Low

References

[1] axic. Enforcing ABI length checks for return data from calls can be breaking. https://github.com/ethereum/solidity/issues/4116.

[2] HaleTom. Resolution on the EIP20 API Approve / TransferFrom multiple withdrawal attack. https://github.com/ethereum/EIPs/issues/738.

[3] MITRE. CWE-1041: Use of Redundant Code. https://cwe.mitre.org/data/definitions/1041.html.

[4] MITRE. CWE-1059: Incomplete Documentation. https://cwe.mitre.org/data/definitions/1059.html.

[5] MITRE. CWE-1068: Inconsistency Between Implementation and Documented Design. https://cwe.mitre.org/data/definitions/1068.html.

[6] MITRE. CWE-197: Numeric Truncation Error. https://cwe.mitre.org/data/definitions/197.html.

[7] MITRE. CWE-269: Improper Privilege Management. https://cwe.mitre.org/data/definitions/269.html.

[8] MITRE. CWE-282: Improper Ownership Management. https://cwe.mitre.org/data/definitions/282.html.

[9] MITRE. CWE-362: Concurrent Execution using Shared Resource with Improper Synchronization ('Race Condition'). https://cwe.mitre.org/data/definitions/362.html.

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

[11] MITRE. CWE-440: Expected Behavior Violation. https://cwe.mitre.org/data/definitions/440.html.

[12] MITRE. CWE-654: Reliance on a Single Factor in a Security Decision. https://cwe.mitre.org/data/definitions/654.html.

[13] MITRE. CWE CATEGORY: 7PK - Security Features. https://cwe.mitre.org/data/definitions/254.html.

[14] MITRE. CWE CATEGORY: 7PK - Time and State. https://cwe.mitre.org/data/definitions/361.html.

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

[16] MITRE. CWE CATEGORY: Behavioral Problems. https://cwe.mitre.org/data/definitions/438.html.

[17] MITRE. CWE CATEGORY: Error Conditions, Return Values, Status Codes. https://cwe.mitre.org/data/definitions/389.html.

[18] MITRE. CWE CATEGORY: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

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

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

[21] PeckShield. ALERT: New batchOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10299). https://www.peckshield.com/2018/04/22/batchOverflow/.

[22] PeckShield. New burnOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-11239). https://www.peckshield.com/2018/05/18/burnOverflow/.

[23] PeckShield. New multiOverflow Bug Identified in Multiple ERC20 Smart Contracts (CVE-2018-10706). https://www.peckshield.com/2018/05/10/multiOverflow/.

[24] PeckShield. New proxyOverflow Bug in Multiple ERC20 Smart Contracts (CVE-2018-10376). https://www.peckshield.com/2018/04/25/proxyOverflow/.

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

[26] PeckShield. Your Tokens Are Mine: A Suspicious Scam Token in A Top Exchange. https://www.peckshield.com/2018/04/28/transferFlaw/.

[27] Solidity. Warnings of Expressions and Control Structures. http://solidity.readthedocs.io/en/develop/control-structures.html.