SMART CONTRACT AUDIT REPORT

for

DFORCE NETWORK

Prepared By: Shuxiao Wang

Feb. 27, 2020

Document Properties

Client	dForce Network
Title	Smart Contract Audit Report
Target	DIP001
Version	1.0
Author	Huaguo Shi
Auditors	Chiachih Wu, Huaguo Shi
Reviewed by	Chiachih Wu
Approved by	Xuxian Jiang
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	Feb. 27, 2020	Huaguo Shi	Final Release
0.3	Feb. 27, 2020	Huaguo Shi	Status Update
0.2	Feb. 26, 2020	Huaguo Shi	Status Update, More Findings Added
0.1	Feb. 15, 2020	Huaguo Shi	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

Given the opportunity to review the **DIP001** 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 DIP001

DIP001 is a protocol that unlocks collaterals from an initiated collateralized DeFi protocol and supply those collaterals into designated yield generating protocols (i.e., Lendf.Me, Compound, dydx etc.) With a DAO scheme, DIP001 allows DF holders to vote for managing the protocol (the management contract is not implemented yet).

The basic information of DIP001 is as follows:

Table 1.1: Basic Information of DIP001

Item	Description
Issuer	dForce Network
Website	https://dforce.network/
Type	Ethereum Smart Contract
Platform	Solidity
Audit Method	Whitebox
Latest Audit Report	Feb. 27, 2020

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

- https://github.com/dforce-network/DIP001/tree/audit (513d6c5)
- https://github.com/dforce-network/DIP001/tree/audit_v0.2 (830e89d)

- https://github.com/dforce-network/DIP001/tree/audit (267ee75)

Table 1.2: Audit Scope

Folder	Files
contracts	Dispatcher.sol
contracts	DispatcherEntrance.sol
contracts/DSLibrary	*.*
contracts/interface	*.*
contracts/CompoundHandler	CompoundHandler.sol
contracts/lendFMeHandler	lendFMeHandler.sol

1.2 About PeckShield

PeckShield Inc. [22] 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 1.3: Vulnerability Severity Classification

Impact \ Likelihood	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

1.3 Methodology

To standardize the evaluation, we define the following terminology based on OWASP Risk Rating Methodology [17]:

- 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 1.3.

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 1.4.

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) [16], 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 1.5 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: 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

Table 1.5: 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 DIP001 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	0	
Medium	1	■
Low	0	
Informational	8	■■■■■■■■
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 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 medium-severity vulnerability, and 8 informational recommendations.

Table 2.1: Key Audit Findings

ID	Severity	Title	Category	Status
PVE-001	Info.	Misleading Return Code in Dispatcher	Error Conditions, Return Values, Status Codes	Resolved
PVE-002	Info.	Missing Check before Withdrawing Principle	Error Conditions, Return Values, Status Codes	Resolved
PVE-003	Medium	Wrong Proportion After Adding/Removing Target Handlers	Business Logics	Resolved
PVE-004	Info.	Excessive Owner Privileges	Business Logics	Confirmed
PVE-005	Info.	Gas Consumption Optimization	Resource Management	Confirmed
PVE-006	Info.	Wrong Proportion After Setting Aimed Proportion	Business Logics	Confirmed
PVE-007	Info.	Insufficient Validation to Target Handler	Error Conditions, Return Values, Status Codes	Confirmed
PVE-008	Info.	Redundant Code in Dispatcher	Coding Practices	Resolved
PVE-009	Info.	Optimization Suggestions	Behavioral Issues	Confirmed

Please refer to Section 3 for details.

3 | Detailed Results

3.1 Misleading Return Code in Dispatcher

- ID: PVE-001
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Dispatcher.sol`
- Category: Error Conditions, Return Values, Status Codes [14]
- CWE subcategory: CWE-394 [6]

Description

In DIP001, the `Dispatcher` is designed to distribute digital assets between different yield generating protocols. Specifically, the `trigger()` function is used to trigger the re-balance process when the amount of reserved assets is below `reserveMin` or above `reserveMax`. However, the function always returns `true` whether `internalDeposit()` or `withdrawPrinciple()` are literally triggered or not. This makes the return code meaningless.

```
function trigger () external returns (bool) {
  uint256 reserve = getReserve();
  uint256 denominator = reserve.add(getPrinciple());
  uint256 reserveMax = reserveUpperLimit * denominator / 1000;
  uint256 reserveMin = reserveLowerLimit * denominator / 1000;
  uint256 amounts;
  if (reserve > reserveMax) {
    amounts = reserve - reserveMax;
    amounts = amounts / executeUnit * executeUnit;
    if (amounts != 0) {
      internalDeposit(amounts);
    }
  } else if (reserve < reserveMin) {
    amounts = reserveMin - reserve;
    amounts = amounts / executeUnit * executeUnit;
    if (amounts != 0) {
      withdrawPrinciple(amounts);
    }
  }
```

```
    return true;
  }
```

Listing 3.1: contracts/Dispatcher.sol

Recommendation Return `true` when something is really triggered. Return `false` when nothing happened.

```
  function trigger () external returns (bool) {
    uint256 reserve = getReserve();
    uint256 denominator = reserve.add(getPrinciple());
    uint256 reserveMax = reserveUpperLimit * denominator / 1000;
    uint256 reserveMin = reserveLowerLimit * denominator / 1000;
    uint256 amounts;
    if (reserve > reserveMax) {
      amounts = reserve - reserveMax;
      amounts = amounts / executeUnit * executeUnit;
      if (amounts != 0) {
        internalDeposit(amounts);
        return true;
      }
    } else if (reserve < reserveMin) {
      amounts = reserveMin - reserve;
      amounts = amounts / executeUnit * executeUnit;
      if (amounts != 0) {
        withdrawPrinciple(amounts);
        return true;
      }
    }
    return false;
  }
```

Listing 3.2: contracts/Dispatcher.sol

3.2 Missing Check before Withdrawing Principle

- ID: PVE-002
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Dispatcher.sol`
- Category: Error Conditions, Return Values, Status Codes [14]
- CWE subcategory: CWE-391 [5]

Description

In the `Dispatcher` contract, the `trigger()` function calls `deposit()`/ `withdraw()` of the corresponding target handler to re-balance the digital assets distribution. We noticed that in `internalDeposit()`, the amount to be deposited is validated such that the underlying `deposit()` is only invoked when the amount is greater than 0. However, the validation is not applied on the `withdraw()` case. Specifically, the `withdrawPrinciple()` does not validate the amount to be withdrew before calling the underlying `withdraw()`.

```
function withdrawPrinciple (uint256 _amount) internal { //
  uint256 i;
  uint256 _amounts = _amount;
  uint256 amountsFromTH;
  uint256 thCurrentBalance;
  uint256 amountsToSatisfiedAimedPropotion;
  uint256 totalBalanceAfterWithdraw = getPrinciple().sub(_amounts);
  TargetHandler memory _th;
  for(i = 0; i < ths.length; ++i) {
    _th = ths[i];
    amountsFromTH = 0;
    thCurrentBalance = getTHPrinciple(i);
    amountsToSatisfiedAimedPropotion = totalBalanceAfterWithdraw.mul(_th.
        aimedPropotion) / 1000;
    if (thCurrentBalance < amountsToSatisfiedAimedPropotion) {
      continue;
    } else {
      amountsFromTH = thCurrentBalance - amountsToSatisfiedAimedPropotion;
      if (amountsFromTH > _amounts) {
        amountsFromTH = _amounts;
        _amounts = 0;
      } else {
        _amounts -= amountsFromTH;
      }
      ITargetHandler(_th.targetHandlerAddr).withdraw(amountsFromTH);
    }
  }
}
```

Listing 3.3: contracts/Dispatcher.sol

Recommendation Ensure `amountsFromTH > 0` in `withdrawPrinciple()` before calling `withdraw()`. For better maintenance, we suggest to change the `amountsFromTH !=0` check in `internalDeposit()` to `amountsFromTH > 0` regardless the fact that `amountsFromTH` is an unsigned integer.

```
function withdrawPrinciple (uint256 _amount) internal { //
  uint256 i;
  uint256 _amounts = _amount;
  uint256 amountsFromTH;
  uint256 thCurrentBalance;
  uint256 amountsToSatisfiedAimedPropotion;
  uint256 totalBalanceAfterWithdraw = getPrinciple().sub(_amounts);
  TargetHandler memory _th;
  for(i = 0; i < ths.length; ++i) {
    _th = ths[i];
    amountsFromTH = 0;
    thCurrentBalance = getTHPrinciple(i);
```

```
            amountsToSatisfiedAimedPropotion = totalBalanceAfterWithdraw.mul(_th.
                aimedPropotion) / 1000;
            if (thCurrentBalance < amountsToSatisfiedAimedPropotion) {
              continue;
            } else {
              amountsFromTH = thCurrentBalance - amountsToSatisfiedAimedPropotion;
              if (amountsFromTH > _amounts) {
                amountsFromTH = _amounts;
                _amounts = 0;
              } else {
                _amounts -= amountsFromTH;
              }
              if(amountsToTH > 0) {
                ITargetHandler(_th.targetHandlerAddr).withdraw(amountsFromTH);
              }
            }
        }
    }
```

Listing 3.4: contracts/Dispatcher.sol

3.3 Wrong Proportion After Adding/Removing Target Handlers

- ID: PVE-003
- Severity: Medium
- Likelihood: Medium
- Impact: Medium
- Target: `Dispatcher.sol`
- Category: Business Logics [13]
- CWE subcategory: CWE-841 [9]

Description

While initializing the `Dispatcher` contract, we can add multiple target handlers with an array along with the corresponding proportion array. Essentially, the constructor of `Dispatcher` ensure the sum of the proportion bound with each target handler is 1000, which makes 100% of the digital assets deposited into target handlers are distributed. Beyond the initialization process, `removeTargetHandler()`/ `addTargetHandle()` could be used to dynamically remove/add target handlers. However, the current implementation of `removeTargetHandler()`/ `addTargetHandle()` does not validate the proportion after adding/removing a target handler, leading to invalid proportion settings. For example, when the privileged user adds or removes a target handler but forgets to re-org the proportion settings with `setAimedPropotion`, the sum of all `aimedPropotion` would be not equal to 1000. Moreover, even if the privileged user does re-org the proportion settings, there's still a time window that the proportion settings is in a wrong state. This leads to a possible front-running attack.

```
function removeTargetHandler(address _targetHandlerAddr, uint256 _index) external auth returns (bool) {
  uint256 length = ths.length;
  require(length != 1, "can not remove the last target handler");
  require(_index < length, "not the correct index");
  require(ths[_index].targetHandlerAddr == _targetHandlerAddr, "not the correct index or address");
  require(getTHPrinciple(_index) == 0, "must drain all balance in the target handler");
  ths[_index] = ths[length - 1];
  ths.length--;
  return true;
}
```

Listing 3.5: removeTargetHandler() contracts/Dispatcher.sol

Recommendation Set the proportion of each target handler whenever a target handler is added or removed and make sure the total `aimedPropotion` is 1000 after the adding/removing operation. Here, we use `removeTargetHandler()` as an example.

```
function removeTargetHandler(address _targetHandlerAddr, uint256 _index, uint256[] calldata _thPropotion) external auth returns (bool) {
  uint256 length = ths.length;
  uint256 sum = 0;
  uint256 i;
  TargetHandler memory _th;

  require(length > 1, "can not remove the last target handler");
  require(_index < length, "not the correct index");
  require(ths[_index].targetHandlerAddr == _targetHandlerAddr, "not the correct index or address");
  require(getTHPrinciple(_index) == 0, "must drain all balance in the target handler");
  ths[_index] = ths[length - 1];
  ths.length--;

  require(ths.length == _thPropotion.length, "wrong length");
  for(i = 0; i < _thPropotion.length; ++i) {
    sum = add(sum, _thPropotion[i]);
  }
  require(sum == 1000, "the sum of propotion must be 1000");
  for(i = 0; i < _thPropotion.length; ++i) {
    _th = ths[i];
    _th.aimedPropotion = _thPropotion[i];
    ths[i] = _th;
  }
  return true;
}
```

Listing 3.6: removeTargetHandler() contracts/Dispatcher.sol

3.4 Excessive Owner Privileges

- ID: PVE-004
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Dispatcher.sol`, `DispatcherEntrance.sol`
- Category: Business Logics [13]
- CWE subcategory: CWE-708 [8]

Description

The current version of DIP001 does not implement the management contract which applies DAO management scheme. With that being said, all privileged functions in `Dispatcher` and `DispatcherEntrance` are controlled by the user having the auth key. That powerful auth key can be used to change the aimed proportion, set the beneficiary address, etc. It would be a single point of failure if the privileged user is compromised, leading to security risks to users' assets.

Recommendation Deploy the management contract and apply the DAO scheme to achieve decentralized governance.

3.5 Gas Consumption Optimization

- ID: PVE-005
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `CompoundHandler.sol`
- Category: Resource Management [15]
- CWE subcategory: CWE-920 [10]

Description

In `CompoundHandler`, the `deposit()` does not validate the `_amounts`, which is waste of gas. Specifically, in the case that `_amounts = 0` , the `principle` would not change after some no-effect code which consumes gas.

```
// token deposit
function deposit(uint256 _amounts) external auth returns (uint256) {
  if (IERC20(token).balanceOf(address(this)) >= _amounts) {
    if(ILendFMe(targetAddr).supply(address(token), _amounts) == 0) {
      principle = add(principle, _amounts);
      return 0;
    }
  }
  return 1;
```

```
}
```

Listing 3.7: contracts/handlers/CompoundHandler.sol

Recommendation Ensure `_amounts` is not 0, which optimizes gas consumption.

```
// token deposit
function deposit(uint256 _amounts) external auth returns (uint256) {
  if (_amounts != 0 && IERC20(token).balanceOf(address(this)) >= _amounts) {
    if(ILendFMe(targetAddr).supply(address(token), _amounts) == 0) {
      principle = add(principle, _amounts);
      return 0;
    }
  }
  return 1;
}
```

Listing 3.8: contracts/handlers/CompoundHandler.sol

3.6 Wrong Proportion After Setting Aimed Proportion

- ID: PVE-006
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Dispatcher.sol`
- Category: Business Logics [13]
- CWE subcategory: CWE-841 [9]

Description

When `setAimedPropotion()` is used to set a new set of aimed proportion, the amount of principle may not be compatible to the new settings. For example, there're three target handlers having `2:3:5` proportion settings and the privileged user changes the settings to `4:1:5` with `setAimedPropotion()`. Since the total reserved assets are not changed before or after the `setAimedPropotion()` operation, the `trigger()` function has no effect to re-balance the proportion (i.e., `reserveMin <= reserve <= reserveMax`), leading to the amount of principle being incompatible to the aimed proportion settings until the next deposit or withdrawal.

3.7 Insufficient Validation to Target Handler

- ID: PVE-007
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `Dispatcher.sol`
- Category: Error Conditions, Return Values, Status Codes [14]
- CWE subcategory: CWE-391 [5]

Description

While adding a new target handler, we noticed that DIP001 validates the `_targetHandlerAddr` by calling the `getTargetAddress()` to ensure that the contract has the corresponding interface implemented (line 329). However, the validation is insufficient here. For example, if the contract is set to be controlled by a malicious owner, the assets deposited into it could be in risks.

```
319   function addTargetHandler(address _targetHandlerAddr, uint256[] calldata _thPropotion)
          external auth returns (bool) {
320     uint256 length = ths.length;
321     uint256 sum = 0;
322     uint256 i;
323     TargetHandler memory _th;
324
325     for(i = 0; i < length; ++i) {
326       _th = ths[i];
327       require(_th.targetHandlerAddr != _targetHandlerAddr, "exist target handler");
328     }
329     ths.push(TargetHandler(_targetHandlerAddr, ITargetHandler(_targetHandlerAddr).
          getTargetAddress(), 0));
330
331     require(ths.length == _thPropotion.length, "wrong length");
332     for(i = 0; i < _thPropotion.length; ++i) {
333       sum += _thPropotion[i];
334     }
335     require(sum == 1000, "the sum of propotion must be 1000");
336     for(i = 0; i < _thPropotion.length; ++i) {
337       _th = ths[i];
338       _th.aimedPropotion = _thPropotion[i];
339       ths[i] = _th;
340     }
341     return true;
342   }
```

Listing 3.9: contracts/Dispatcher.sol

Recommendation Check the integrity of the target handler to be added.

3.8 Redundant Code in Dispatcher

- ID: PVE-008
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Targets: `Dispatcher.sol`
- Category: Coding Practices [11]
- CWE subcategory: CWE-1041 [4]

Description

The `DSMath` library is redundant in `Dispatcher` contract since `DSLibrary/DSMath.sol` could be included and used directly.

```
library DSMath {
  function add(uint x, uint y) internal pure returns (uint z) {
    require((z = x + y) >= x, "ds-math-add-overflow");
  }
  function sub(uint x, uint y) internal pure returns (uint z) {
    require((z = x - y) <= x, "ds-math-sub-underflow");
  }
  function mul(uint x, uint y) internal pure returns (uint z) {
    require(y == 0 || (z = x * y) / y == x, "ds-math-mul-overflow");
  }
}
```

Listing 3.10: contracts/Dispatcher.sol

3.9 Optimization Suggestions

- ID: PVE-009
- Severity: Informational
- Likelihood: N/A
- Impact: N/A
- Target: `CompoundHandler.sol`
- Category: Behavioral Issues [12]
- CWE subcategory: CWE-440 [7]

Description

In `CompoundHandler`, one of the target handler, the `getProfit()` function could be optimized by reducing the calculation in the case `_balance == _principle`. Specifically, when `_balance == _principle`, the `_amounts` in line 99 would be 0 which means line 100 is not necessary.

```
92  function getProfit() public view returns (uint256) {
93      uint256 _balance = getBalance();
94      uint256 _principle = getPrinciple();
95      uint256 _unit = IDispatcher(dispatcher).getExecuteUnit();
```

```
        if (_balance < _principle) {
            return 0;
        } else {
          uint256 _amounts = sub(_balance, _principle);
          _amounts = _amounts / _unit * _unit;
            return _amounts;
        }
    }
```

Listing 3.11: contracts/handlers/CompoundHandler.sol

Recommendation Return 0 directly when `_balance == _principle`.

```
    function getProfit() public view returns (uint256) {
        uint256 _balance = getBalance();
        uint256 _principle = getPrinciple();
        uint256 _unit = IDispatcher(dispatcher).getExecuteUnit();
        if (_balance <= _principle) {
            return 0;
        } else {
          uint256 _amounts = sub(_balance, _principle);
          _amounts = _amounts / _unit * _unit;
            return _amounts;
        }
    }
```

Listing 3.12: contracts/handlers/CompoundHandler.sol

3.10 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 0.5.4;` instead of `pragma solidity ^0.5.4;`.

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.

Based on the nature of DeFi, some security risks may exist while integrating different DeFi components. Currently, DIP001 integrates Lendf.me [3] and Compound [2], which works smoothly so far. If some new Defi components are needed to be integrated in the future, dForce Network should consider the security risks and the liquidity of them. It would be a good idea to conduct a security assessment before integrating each new component.

4 Conclusion

In this audit, we thoroughly analyzed the DIP001 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 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.

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 [18, 19, 20, 21, 23].
- Result: Not found
- Severity: Critical

5.1.5 Reentrancy

- Description: Reentrancy [24] 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] Inc. Compound Labs. Compound. https://compound.finance/.

[3] dForce Network. Lendf. https://www.lendf.me.

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

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

[6] MITRE. CWE-394: Unexpected Status Code or Return Value. https://cwe.mitre.org/data/definitions/394.html.

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

[8] MITRE. CWE-708: Incorrect Ownership Assignment. https://cwe.mitre.org/data/definitions/708.html.

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

[10] MITRE. CWE-920: Improper Restriction of Power Consumption. https://cwe.mitre.org/data/definitions/920.html.

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

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

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

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

[15] MITRE. CWE CATEGORY: Resource Management Errors. https://cwe.mitre.org/data/definitions/399.html.

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

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

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

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

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

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

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

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

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