SMART CONTRACT AUDIT REPORT

for

DFORCE DIP001

Prepared By: Shuxiao Wang

Feb. 27, 2020

Document Properties

客户	dForce Network
报告	Smart Contract Audit Report
目标	dForce DIP001
版本	1.0
作者	Huaguo Shi
审计人员	Chiachih Wu, Huaguo Shi
复审	Chiachih Wu
审批	Xuxian Jiang
分类	Confidential

版本信息

版本	日期	审计人	描述
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

联系方式

关于本审计报告的更多详细信息，请联系PeckShield [7] 。

联系人	Shuxiao Wang
电话	+86 173 6454 5338
电子邮件	contact@peckshield.com

Contents

1 介绍

我们（**PeckShield** [7]）受客户委托对**dForce DIP001**的代码进行安全审计。根据我们的安全审计规范和客户需求，我们将在报告中列出用于检测潜在安全问题的系统性方法，并根据检测结果给出相应的建议或推荐以修复或改善代码质量。 分析结果表明，该特定版本的智能合约代码存在若干安全隐患，在包括ERC20标准兼容性、安全性以及性能方面存在一定的改进空间。修复的方式请参考第 3章检测结果详情部分的内容。本文档对审计结果作了分析和阐述。

1.1 关于 dForce DIP001

DIP001 目的是将抵押在合约中的资产，例如USDx协议中抵押的USDC/PAX/TUSD，通过治理合约转出一部分倒去中心化金融产品进行生息，获得的利益可以作为合约产品参与人的利息补偿。DIP001 是由治理合约管理的去中心化协议，DF持有者通过投票来管理。（治理合约功能暂未实现）

dForce DIP001的基本信息如下：

Table 1.1: dForce DIP001的基本信息

条目	描述
发行方	dForce Network
网站	https://dforce.network/
类型	Ethereum 智能合约
平台	Solidity
审计方法	白盒
审计完成时间	Feb. 27, 2020

以下是审计对象（即智能合约）相关信息：

- Github: https://github.com/dforce-network/DIP001/tree/audit (513d6c5)
- Github: https://github.com/dforce-network/DIP001/tree/audit_v0.2 (830e89d)
- Github: https://github.com/dforce-network/DIP001/tree/audit (267ee75)

- 审计范围:

Table 1.2: 审计范围

文件夹	文件
contracts	Dispatcher.sol
contracts	DispatcherEntrance.sol
contracts/DSLibrary	*.*
contracts/interface	*.*
contracts/CompoundHandler	CompoundHandler.sol
contracts/lendFMeHandler	lendFMeHandler.sol

1.2 关于PeckShield

PeckShield (派盾) 是面向全球的业内顶尖区块链安全团队，以提升区块链生态整体的安全性、隐私性以及可用性为己任，通过发布行业趋势报告、实时监测生态安全风险，负责任曝光0day漏洞，以及提供相关的安全解决方案和服务等方式帮助社区抵御新兴的安全威胁。可以通过下列联系方式联络我们： Telegram (https://t.me/peckshield), Twitter (twitter), or Email (contact@peckshield.com).

1.3 评估方法和模型

为了检测评估的标准化，我们根据OWASP Risk Rating Methodology [2]定义下列术语:

- 可能性： 表示某个特定的漏洞被发现和利用的可能性;
- 影响力: 度量了（利用该漏洞的）一次成功的攻击行动造成的损失;
- 危害性： 显示该漏洞的危害的严重程度;

可能性和影响力各自被分为三个等级: 高、中和低。危害性由可能性和影响力确定，分为四个等级: 致命、高危、中危、低危，如表 1.3所示。

我们整理了常见或具备一定危害性的检测项，并按照如下流程进行审计:

- 基本编码漏洞检测: 首先以自研发的自动化静态检测工具分析智能合约代码，而后人工确认漏洞是否真实存在。
- 语义一致性检查: 人工确认智能合约的实现与白皮书的描述是否一致。
- 高级DeFi审查: 我们通过对业务逻辑、系统运行以及其它DeFi相关的内容展开进一步的审查以发现潜在的隐患或漏洞。

Table 1.3: 危害性定义

影响力 \ 可能性	高	中	低
高	致命	高危	中危
中	高危	中危	低危
低	中危	低危	低危

- 其它建议：从业已被实践所证明的良好开发实践出发，针对智能合约代码的编写和部署给出建议或者意见。

为了评估风险，我们对检测项的危害性做了标记。对任一检测项，如果我们的工具没有查找出任何安全问题，则我们认为合约通过了该项检测。对任何发现的问题，我们可能会在我们自己的私有测试链上实际部署予以确认。如果必要的话，我们将编写PoC代码验证漏洞利用的可能性。具体的检测列表见表 1.4。

1.4 免责声明

请注意该审计报告并不保证能够发现该智能合约中存在的一切安全问题，即评估结果并不能保证在未来不会发现新的安全问题。我们一向认为单次审计结果可能并不全面，因而推荐采取多个独立的审计和公开的漏洞奖赏计划相结合的方式来确保合约的安全性。最后必须要强调的是，审计结果仅针对智能合约代码的安全性，不构成任何投资建议。

Table 1.4: 完整的检测项列表

检测类型	检测项
基本编码漏洞检测	Constructor Mismatch
	Ownership Takeover
	Redundant Fallback Function
	Overflows & Underflows
	Reentrancy
	Money-Giving Bug
	Blackhole
	Short Address Bug
	Unauthorized Self-Destruct
	Revert DoS
	Unchecked External Call
	Gasless Send
	Send Instead Of Transfer
	Costly Loop
	(Unsafe) Use of Untrusted Libraries
	Approve / TransferFrom Race Condition
	(Unsafe) Use of Predictable Variables
	Transaction Ordering Dependence
	Deprecated Uses
语义一致性检查	语义一致性检查
高级DeFi审查	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
其它建议	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

2 检测结果

2.1 总结

Severity	# of Findings	
严重	0	
高危	0	
中危	1	🟦
低危	0	
参考	8	🟦🟦🟦🟦🟦🟦🟦🟦
总计	9	

2.2 主要发现

我们在本次审计中我们发现了9个安全隐患的存在，包括1个中危的漏洞，主要问题已经修复，如下表 2.1所示：

Table 2.1: 主要发现

编号	严重性	名称	类型	状态
PVE-001	参考	合约Dispatcher trigger() 缺少返回值逻辑	建议	已修复
PVE-002	参考	调用 withdraw 缺少金额正确性校验	建议	已修复
PVE-003	中	添加/移除 借贷协议Handler时资产配比异常	漏洞	已修复
PVE-004	参考	管理权限过于集中	建议	已确认
PVE-005	参考	优化gas使用问题	建议	已确认
PVE-006	参考	抵押资产配比不一致	建议	已确认
PVE-007	参考	新增借贷入口合约接口验证不充分	建议	已确认
PVE-008	参考	Dispatcher 包含冗余代码	建议	已修复
PVE-009	参考	getProfit 逻辑优化	建议	已确认

具体细节请参考第 3章。

3 检测结果详情

3.1 合约Dispatcher trigger() 缺少返回值逻辑

- ID: PVE-001
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

漏洞描述

Dispatcher 是用来对 token 资产进行分配的合约，合约里的函数 trigger() 触发当前资产的分配逻辑，如列表中代码 第 62 行所示 trigger() 函数声明第返回值为 boolean 类型，但从代码中看无论程序是否执行internalDeposit() 和 withdrawPrinciple() 程序始终返回 true ，这样函数返回值失去了判断的意义。

```
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

修改建议 增加正确的返回值逻辑，函数默认返回值为`false`，只有真正触发资产分配逻辑才返回 `true`， 代码修改如下:

```
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 调用 withdraw 缺少金额正确性校验

- ID: PVE-002
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

漏洞描述

在 `Dispatcher` 合约里，通过函数 `trigger()` 根据不同逻辑分支触发借贷入口合约 `ITargetHandler` 的`deposit()`/ `withdraw()`，借贷接口合约的存款/提现功能都需要传入合理参数，即数额大于0，函数 `internalDeposit()` 的代码在调用 `deposit()` 之前有对输入参数是否为0的逻辑判断，而在 `withdrawPrinciple()` 函数调用 `withdraw()` 时，则缺少对amountsFromTH的判断逻辑。（代码列表中第 139 行）

```
116  function withdrawPrinciple (uint256 _amount) internal { //
117    uint256 i;
118    uint256 _amounts = _amount;
119    uint256 amountsFromTH;
120    uint256 thCurrentBalance;
121    uint256 amountsToSatisfiedAimedPropotion;
122    uint256 totalBalanceAfterWithdraw = getPrinciple().sub(_amounts);
123    TargetHandler memory _th;
124    for(i = 0; i < ths.length; ++i) {
125      _th = ths[i];
126      amountsFromTH = 0;
127      thCurrentBalance = getTHPrinciple(i);
128      amountsToSatisfiedAimedPropotion = totalBalanceAfterWithdraw.mul(_th.
           aimedPropotion) / 1000;
129      if (thCurrentBalance < amountsToSatisfiedAimedPropotion) {
130        continue;
131      } else {
132        amountsFromTH = thCurrentBalance - amountsToSatisfiedAimedPropotion;
133        if (amountsFromTH > _amounts) {
134          amountsFromTH = _amounts;
135          _amounts = 0;
136        } else {
137          _amounts -= amountsFromTH;
138        }
139        ITargetHandler(_th.targetHandlerAddr).withdraw(amountsFromTH);
140      }
141    }
142  }
```

Listing 3.3: contracts/Dispatcher.sol

修改建议 在 `withdrawPrinciple()` 函数调用 `withdraw()` 之前增加对参数 amountsFromTH 是否大于0的判断, 为保持代码风格一致，虽然输入参数 `amountsFromTH` 为无符号整数，但仍建议同样将`internalDeposit()` 判断 `amountsFromTH !=0` 改为 `amountsFromTH >0` 。

```
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 添加/移除 借贷协议Handler时资产配比异常

- ID: PVE-003
- 危害性: 中
- 可能性: 中
- 影响力: 中

漏洞描述

在 `Dispatcher` 合约在初始化时，可以添加 Target Handler 数组来设置多个借贷合约以通过合约进行自动借贷生息，并在初始时将分配了每个借贷合约的资产配比，各借贷合约的资产配比总和设值为1000(总比例100%)。 同时，合约中提供两个函数 `removeTargetHandler()`/ `addTargetHandle()` 该接口可由项目方调用进行动态添加和移除所支持的借贷合约接口。`removeTargetHandler()`/ `addTargetHandle()` 函数实现里，只是从支持的借贷合约数组中添加/移除，但相应的资产配比并没有改变，虽然合约中提供函数 `setAimedPropotion` 可以进行资产配比的重新设置，但仍然会出现资产配比不是100%的情况出现： 1. 操作者失误忘记在添加/移除借贷合约后，重新设置资产配比； 2. 调用添加/移除函数 和 调用 `setAimedPropotion` 之间的时间空隙。 这种情况会导致暂时与合约设计不符，产生资产配比异常的情况（移除，则配比资产总和小于100% ； 新增，则新增借贷合约无法进行借贷），还有可能会被利用。

```
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

修改建议 函数 `removeTargetHandler()`/ `addTargetHandle()` 增加参数来重新调整aimedPropotion比例，保持修改后的 ths 数组 aimedPropotion 总和始终为1000，以`removeTargetHandler()` 为例，修改如下：

```
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 管理权限过于集中

- ID: PVE-004
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

漏洞描述

由于当前审计的DIP001版本暂未实现治理合约功能，在资产配置合约 `Dispatcher` 合约和入口合约`DispatcherEntrance` 代码中看到，所有管理权限和资产分配API 都由同一个 auth key 作为权限控制，auth key拥有者既可以管理DIP001的借贷协议资产配比，又可以指定收益地址获取收益。在未集成治理合约的情况下，这种集中化的权限管理如果auth key丢失等其他人为因素，可能会给用户资产带来一定风险。

修改建议 部署实现治理合约：通过治理合约实现设置资产留存比例、资产分配（到借贷合约）比例、设置收益地址等治理功能。实现去中心化管理。

3.5 优化gas使用问题

- ID: PVE-005
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

DIP001 引入了借贷入口合约（`Target Handler`），该合约实例封装了与三方借贷合约交互的接口，在审计Handler实例（`CompoundHandler/lendFMeHandler`）的代码时发现函数 `deposit()` 没有对参数 `_amounts` 是否为0进行判断，当`_amounts =0` 时，即使其他条件均满足，最终计算 `principle` 值仍然没有变化，增加无效计算耗费gas资源。如下面代码列表中 `CompoundHandler.sol` 第 35 行所示：

```
35  // token deposit
36  function deposit(uint256 _amounts) external auth returns (uint256) {
37    if (IERC20(token).balanceOf(address(this)) >= _amounts) {
38      if(ILendFMe(targetAddr).supply(address(token), _amounts) == 0) {
39        principle = add(principle, _amounts);
40        return 0;
41      }
42    }
43    return 1;
44  }
```

Listing 3.7: contracts/handlers/CompoundHandler.sol

修改建议 在函数入口处增加判断 `_amounts` 是否为0，可以有效优化gas使用。

```
35  // token deposit
36  function deposit(uint256 _amounts) external auth returns (uint256) {
37    if (_amounts != 0 && IERC20(token).balanceOf(address(this)) >= _amounts) {
38      if(ILendFMe(targetAddr).supply(address(token), _amounts) == 0) {
39        principle = add(principle, _amounts);
40        return 0;
41      }
42    }
43    return 1;
44  }
```

Listing 3.8: contracts/handlers/CompoundHandler.sol

3.6 抵押资产配比不一致

- ID: PVE-006
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

DIP001 中可以动态对借贷入口合约（`Target Handler`）进行借贷合约的抵押资产比例进行设置。审计过程中发现，当调用接口 `setAimedPropotion()` 时，会造成短期内借贷入口合约资产配比和实际的抵押资产（`principle`）配比不一致。 例如：Dispacher中有三个 `Target Handler`，资产配比为：（`2:3:5`），当前抵押资产比例也和资产配比相符。 此时通过接口 `setAimedPropotion()`

设置资产配比变为 `(4:1:5)`，就会导致两者比例出现差异，此时在没有改变token保有量时，即 `trigger()` 没有真正触发执行 `internalDeposit()/withdrawPrinciple()`，会出现实际的抵押资产和最新抵押资产配比不一致的情况。

3.7 新增借贷入口合约接口验证不充分

- ID: PVE-007
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

DIP001 中可以动态对借贷入口合约（`Target Handler`）进行添加、删除以及借贷合约的抵押资产比例进行设置。 我们审计发现，添加接口中有对所添加地址的有效性做检查。如下列代码第 329 行：

```
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/handlers/CompoundHandler.sol

上述代码中通过 `(_targetHandlerAddr).getTargetAddress()` 接口来检测当前输入的地址是否合规：可以检测出当前地址是否为有效的合约地址，这样可以有效防范因手工操作输入错误的地

址。但是我们认为，这仍然会有添加其他恶意地址的可能，比如误添加非owner控制权限的恶意合约，会有存在资产丢失的风险。

修改建议 为避免上述风险，我们建议将添加借贷入口合约逻辑中增加对合约地址的有效性验证：增加个合约地址主动发起的操作行为，用于验证新增合约地址的有效性。

3.8 Dispatcher 包含冗余代码

- ID: PVE-008
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

漏洞描述

`contracts/Dispatcher.sol` 中包含重复定义的代码实现 `DSMath` ，可以通过直接引用 `contracts/DSLibrary/DSMath.sol`，减少代码冗余。重复代码如下：

```
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 getProfit 逻辑优化

- ID: PVE-009
- 危害性: 参考
- 可能性: N/A
- 影响力: N/A

DIP001 引入了借贷入口合约——Target Handler，该合约实例封装了与三方借贷合约交互的接口，在审计Handler实例（CompoundHandler/lendFMeHandler）的代码时发现函数 getProfit() 对 _balance 与 _principle 比较判断不充分，当两者相等时也应该直接返回0，避免后续的运算。如 CompoundHandler.sol 代码第 96 行所示：

```
92   function getProfit() public view returns (uint256) {
93       uint256 _balance = getBalance();
94       uint256 _principle = getPrinciple();
95       uint256 _unit = IDispatcher(dispatcher).getExecuteUnit();
96       if (_balance < _principle) {
97           return 0;
98       } else {
99         uint256 _amounts = sub(_balance, _principle);
100        _amounts = _amounts / _unit * _unit;
101          return _amounts;
102      }
103  }
```

Listing 3.11: contracts/handlers/CompoundHandler.sol

修改建议 将 CompoundHandler/lendFMeHandler 的代码 _balance 判断都改为小于等于 _principle ,如果是相等也直接返回，减少后续运算量。修改如下：

```
92   function getProfit() public view returns (uint256) {
93       uint256 _balance = getBalance();
94       uint256 _principle = getPrinciple();
95       uint256 _unit = IDispatcher(dispatcher).getExecuteUnit();
96       if (_balance <= _principle) {
97           return 0;
98       } else {
99         uint256 _amounts = sub(_balance, _principle);
100        _amounts = _amounts / _unit * _unit;
101          return _amounts;
102      }
103  }
```

Listing 3.12: contracts/handlers/CompoundHandler.sol

3.10 其他建议

不同编译器版本编译可能会出现编译运行结果不一致的问题，所以强烈建议指定编译器版本，例如，将 Dispatcher.sol 中的 pragma solidity ^0.5.4; 修改为 pragma solidity 0.5.4;

此外，我们也强烈建议不要使用 Solidity 实验版的功能特性或未通过安全认证的第三方代码库。开发应该基于稳定的编译器版本特性和授信三方代码库，如果必须要用到非稳定版本的编译器特性和不可信三方代码库时，必须要予以审计和制定应急预案。

基于DeFi项目的特性，存在许多不同项目结合时产生的安全隐患，目前dForce DIP001已接入Lendf.me及Compound，如果未来需要新的项目，需要考虑该项目本身的安全性，资产流通性等问题，必要时也可再经由第三方机构审计之后再接入。

4 结论

我们对dForce DIP001进行了安全审计，截至该报告撰写为止，该版本的智能合约代码存在的必须解决的问题已修复。整体来看，DIP001的代码整体逻辑以及架构设计是很严谨和专业的。

但正如免责声明 1.4所述，该审计报告的结果并不不意味着该智能合约 不不存在其它安全问题，亦不不构成任何投资建议。

最后，对于报告内容和格式，我们欢迎任何建设性的反馈或建议。

5 附录

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 [3, 4, 5, 6, 8].
- Result: Not found
- Severity: Critical

5.1.5 Reentrancy

- Description: Reentrancy [9] 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 Make Visibility Level Explicit

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

5.3.3 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

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

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

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

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

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

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

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

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