SMART CONTRACT AUDIT REPORT

for

DEFIS NETWORK

Prepared By: Shuxiao Wang

Hangzhou, China
Oct. 29, 2020

Document Properties

Client	DeFis Network
Title	Smart Contract Audit Report
Target	YFC 机枪池
Version	1.0-rc1
Author	Shawn Li, Huaguo Shi
Auditors	Shawn Li, Huaguo Shi
Reviewed by	Chiachih Wu
Approved by	Chiachih Wu
Classification	Confidential

Version Info

Version	Date	Author(s)	Description
1.0-rc1	Oct. 29, 2020	Shawn Li, Huaguo Shi	Release Candidate #1

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

目录

1 介绍

我们（**PeckShield** [12]）受客户委托对 **YFC 机枪池**智能合约进行安全审计。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安全问题或提高安全性。 分析结果表明，该特定版本的智能合约发现1处高危漏洞，即合约执行部分项目流动性挖矿的功能不可用；并且存在若干安全隐患，包括预约提取资金的时间没有最小值检查和存在 CPU 资源被恶意消耗等问题，需要予以关注或修复。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YFC 机枪池

DeFis Network 的使命是基于区块链技术，打造一个开放式的金融网络。其整合了一系列 DeFi 协议，包括通用结算货币发行协议、流动性协议等。本次审计的 YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DeFis Network 上进行流动性挖矿，实现获利。用户代币存入合约后，在合约空闲期可自由提取，在执行流动性挖矿期间无法提取，但可预约延后取回，并由项目方后台配合合约完成。 YFC 机枪池智能合约基本信息如下：

表 1.1: YFC 机枪池的基本信息

条目	描述
发行方	DeFis Network
发行平台	EOS
项目网站	https://yfc.one/guns
合约语言	C++
审计方法	白盒
审计完成时间	Oct. 29, 2020

以下是审计对象（即EOS智能合约）相关信息：

- 合约代码及文档信息 v3 (2020-10-21):

$MD5$: $0x6872d1a39940ced422983da56edc14d4$
$SHA-256$: $0xb24a137052b744c4b7de3cde30fdbd9f7bcd3ca80b30461a5d87cf52a9b183e0$
部署$code\ hash$: $0x30de365e62f0da91c2c352fcb5a9bf8f9dac8c5bddb32c459d6040c63c5d7dd7$

- 合约代码及文档信息 v2 (2020-10-12):
  $MD5$: $0x4e34a0b23d23ae0f3161efc80279fa86$
  $SHA-256$: $0x9916ed37271e2bbe789256609ab56ebcda8611b3e7cca6f3161bfdbe2d1177ec$
  部署$code\ hash$: $0x21bcd828c28b9f099c612f7b72aea50a8f7a165947cce761b6ba2b2ca91e20d9$

- 合约代码及文档信息 v1 (2020-10-10):
  $MD5$: $0x08cf9dc063ae68fbf9b9f960f00befc6$
  $SHA-256$: $0x16a1442bf0c210b8058add48fb4fe34f292283dc85a4fd4095b61af46c52fee1$
  部署$code\ hash$: $0x503666ec723745422544db89b502215f8b4527829a4cb23c711f528e65e06131$

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 (http://twitter.com/peckshield), 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 [11]定义下列术语：

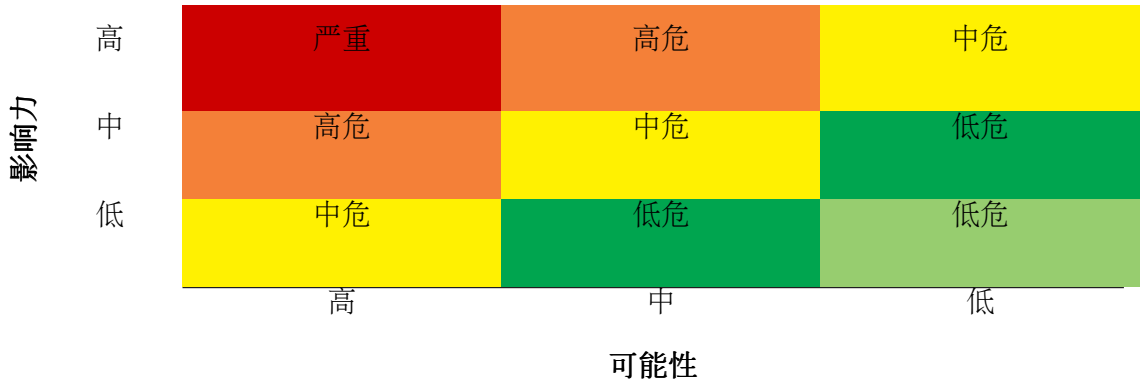
- 可能性： 表示某个特定的漏洞被发现和利用的可能性；
- 影响力： 度量了（利用该漏洞的）一次成功的攻击行动造成的损失；
- 危害性： 显示该漏洞的危害的严重程度；

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：严重、高危、中危、低危，如表 1.2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，而后人工确认漏洞是否真实存在。
- 代码及业务安全性检测：我们通过对业务逻辑、系统运行以及其它相关的内容展开进一步的审查以发现潜在的隐患或漏洞。

表 1.2: 危害性定义



- 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，则我们会进一步人工分析代码确认。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 1.3。

1.4 免责声明

请注意该审计报告并不保证能够发现 YFC 机枪池智能合约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 YFC 机枪池智能合约的安全性，不构成任何投资建议。

表 1.3: 完整的检测项列表

检测类型	检测项
基本漏洞检测	apply 验证漏洞
	权限校验漏洞
	假转账攻击漏洞
	伪造转账通知
	数值溢出
	交易回滚攻击
	交易阻塞攻击
	soft_fail 攻击检测
	hard_fail 攻击检测
	异常 memo 检测
	资源异常消耗检测
	随机数安全问题检测
代码及业务安全性检测	资产安全
	加解密算法强度检测
	业务逻辑风险分析
	代码功能验证
	帐号权限控制
	敏感行为检测
	敏感信息泄露检测
	熔断机制
	黑名单机制
	系统 API 调用检测
	合约部署一致性检查
其它建议	语义一致性检测
	代码建议与优化

2 检测结果

2.1 总结

严重性	发现个数	
严重	0	
高危	1	■
中危	2	■■
低危	1	■
参考	3	■■■
总计	7	

2.2 主要发现

我们在本次审计中发现1处高危漏洞，即合约执行部分项目流动性挖矿的功能不可用；另外发现了6个安全隐患，其中2个中危漏洞、1个低危漏洞和4个建议项，如下表 2.1所示。具体细节请参考第 3章。

表 2.1: 主要发现

编号	严重性	名称	状态
PVE-001	高危	DFS流动性挖矿不可用	已修复
PVE-002	中危	CPU资源存在被恶意消耗的风险	已确认
PVE-003	中危	预约提取缺少最小值检查	已修复
PVE-004	参考	过载检测实现和文档不一致	已确认
PVE-005	低危	price计算存在除0风险	已确认
PVE-006	参考	内联调用层数可能过深	已确认
PVE-007	参考	参与流动性挖矿存在本金损失风险	已确认

3 检测结果详情

3.1 DFS流动性挖矿不可用

- ID: PVE-001
- 危害性: 高
- 可能性: 高
- 影响力: 中
- 目标: `ebonyvsivory.cpp`
- 类型: Behavioral Issues [7]
- CWE 子类: CWE-440 [4]

描述

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS(DeFis Network) 上进行流动性挖矿，实现获利。用户代币存入合约后，项目方后台会调用合约在一定时间内将其用于流动性挖矿。代币在流动性挖矿期间无法提取，但可预约延后取回，并由项目方后台配合合约完成。另外，合约会有短暂的空闲期，此时的代币用户可自由提取。合约中代币的获利来自流动性挖矿，而审计发现合约的 DFS 流动性挖矿无法执行，因此相关获利无法实现。

YFC 机枪池智能合约中函数 `lu()` 实现了针对不同挖矿项目计算两种代币所需的数量，然后分别调用 `lu_dfs()` 和 `lu_box()` 来执行真正的挖矿操作。函数中对参与 Defibox 和 DFS 的流动性挖矿分别以 `Ebony` 和 `Ivory` 为 nick 名称来区分，如代码中第144 – 155行所示。但是，从上述代码逻辑中能够发现，nick 名称不等于 Ebony 时，就执行后面的 else 语句异常退出，导致等于 Ivory 的情况始终无法执行，也就是无法完成针对 DFS 的流动性挖矿。

```
87 void ebonyvsivory::lu(string nick, name swap, uint64_t mid)
88 {

90    asset token0_bal = utils::get_balance(token0_contract, get_self(), token0_symbol.code
          ());
91    asset token1_bal = utils::get_balance(token1_contract, get_self(), token1_symbol.code
          ());

93    uint64_t reserve0 = 0;
94    uint64_t reserve1 = 0;
95    // fetch market info
```

```
96      if (nick == "Ivory")
97      {
98         markets _markets(swap, swap.value);
99         auto m_itr = _markets.require_find(mid, "ebonyvsivory lu_dfs: market does not
               exist.");
100        reserve0 = m_itr->reserve0.amount;
101        reserve1 = m_itr->reserve1.amount;
102     }
103     if (nick == "Ebony")
104     {
105        pairs _markets(swap, swap.value);
106        auto m_itr = _markets.require_find(mid, "ebonyvsivory lu_box: market does not
               exist.");
107        reserve0 = m_itr->reserve0.amount;
108        reserve1 = m_itr->reserve1.amount;
109     }

111     double price = (double)reserve1 / reserve0;


114     uint64_t amount0_desired = token0_bal.amount;
115     uint64_t amount1_desired = token1_bal.amount;
116     uint64_t amount0 = 0;
117     uint64_t amount1 = 0;

119     if (reserve0 == 0 && reserve1 == 0)
120     {
121        amount0 = amount0_desired;
122        amount1 = amount1_desired;
123     }
124     else
125     {
126        uint64_t amount1_optimal = quote(amount0_desired, reserve0, reserve1);
127        if (amount1_optimal <= amount1_desired)
128        {
129           amount0 = amount0_desired;
130           amount1 = amount1_optimal;
131        }
132        else
133        {
134           uint64_t amount0_optimal = quote(amount1_desired, reserve1, reserve0);
135           check(amount0_optimal <= amount0_desired, "math error"); // both token0 and token1
                  insufficient, shoule never happen
136           amount0 = amount0_optimal;
137           amount1 = amount1_desired;
138        }
139     }

141     token0_bal.amount = amount0;
142     token1_bal.amount = amount1;

144     if (nick == "Ivory")
```

```
145     {
146         lu_dfs(token0_bal, token1_bal);
147     }
148     if (nick == "Ebony")
149     {
150         lu_box(token0_bal, token1_bal);
151     }
152     else
153     {
154         check(false, "lu error");
155     }
```

Listing 3.1: ebonyvsivory.cpp

修复方法 正常的逻辑是判断 nick 名称不为 Ivory 或 Ebony 时才异常退出。因此，在代码第148行的 if 前加上 else 即可。

3.2 CPU资源存在被恶意消耗的风险

- ID: PVE-002
- 危害性: 中
- 可能性: 中
- 影响力: 低
- 目标: ebonyvsivory.cpp
- 类型: Resource Management [10]
- CWE 子类: CWE-400 [3]

描述

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS 上进行流动性挖矿，实现获利。用户代币存入合约后，项目方后台会调用合约在一定时间内将其用于流动性挖矿。代币在流动性挖矿期间无法提取，但可预约延后取回。项目方会根据用户的预约延后时间，就近选择挖矿间歇期调用合约完成代币返还。但是代币返还的操作都是由项目方调用完成，并且没有检查收取代币的账号是否为合约账号。因此项目方账号的 CPU 资源存在被恶意消耗的风险。由于流动性挖矿的策略，每次提取间隔在 1 小时左右，因此恶意用户单个账号无法短时间内持续消耗。如果增加恶意的合约账号数量，也会增加攻击者部署合约的成本，但仍可以实现。

具体来看，在 YFC 机枪池智能合约中，函数 `execorder()` 实现了在合约挖矿的间歇期，由项目方后台调用合约实现代币的预约提取。具体的提取操作由 `withdraw()` 函数完成转账。整个调用逻辑消耗的是 `DFS_DEV` 账号的 CPU 资源，因此恶意用户可以在接收转账时恶意消耗 CPU 资源。

```
386 void ebonyvsivory::execorder(name user, symbol_code sym)
387 {
388     require_auth(DFS_DEV);
389     orders _orders(get_self(), sym.raw());
390     auto itr = _orders.require_find(user.value, "You don't have any order.");
391     check(itr->when.sec_since_epoch() <= current_time_point().sec_since_epoch(), "pls
          wait");
392     withdraw(user, sym, itr->amount);
393     _orders.erase(itr);
394 }
```

Listing 3.2: ebonyvsivory.cpp

修复方法 在执行用户的代币提取前判断该账号是否为合约账号即可避免 CPU 恶意消耗的风险。同时可通过白名单的形式增加灵活性，避免正常合约账号的提取。另外需要注意该函数的调用频度，避免短时间内同一账号的重复执行。

3.3 预约提取缺少最小值检查

- ID: PVE-003
- 危害性: 中危
- 可能性: 高
- 影响力: 低
- 目标: ebonyvsivory.cpp
- 类型: Behavioral Issues [7]
- CWE 子类: CWE-440 [4]

描述

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS 上进行流动性挖矿，实现获利。用户代币存入合约后，项目方后台会调用合约在一定时间内将其用于流动性挖矿。代币在流动性挖矿期间无法提取，但可预约延后取回，并由项目方后台配合合约完成。预约延后取回的时间规定在 1～12 小时之间。但合约中并没有检查延后的时间为最小值0的情况。另外提取的频度完全由项目方后台控制，如果后台不断循环执行可提取的账号，可能导致和 CPU 资源被滥用的问题配合，持续消耗项目方账号的 CPU。

在 YFC 机枪池智能合约中，函数 `schedule()` 中实现了用户自己预约延后提取代币的操作。但是如代码第401行所示，只检查了最大值为12的情况，并没有检查最小值。因此，用户可设定延后时间为0。

```
396 ACTION ebonyvsivory::schedule(name user, symbol_code sym, uint64_t amount, uint64_t hour
        )
397 {
398     require_auth(user);

400     check(amount > 0, "invaid liquidity token amount");
401     check(hour <= 12, "invaid hour: 1 ~ 12");

403     guns _guns(get_self(), get_self().value);
404     auto g_itr = _guns.require_find(sym.raw(), "Gun does not exist.");

406     uint64_t reserve0 = g_itr->bullet.amount;
407     check(g_itr->lpt > 0, "INSUFFICIENT_LIQUIDITY_TOKEN");
408     uint64_t amount0 = (uint64_t)((uint128_t)amount * (uint128_t)reserve0 / (uint128_t)
            g_itr->lpt);

410     check(amount0 > 0, "INSUFFICIENT_LIQUIDITY_BURNED");
411     check(amount0 <= reserve0, "INSUFFICIENT_LIQUIDITY_RESERVE");


414     bullets _bullets(get_self(), sym.raw());
415     auto liq_itr = _bullets.require_find(user.value, "User liquidity does not exist.");
416     check(liq_itr->lpt >= amount, "Invalid token amount.");
417     check(liq_itr->lpt > 0, "Liquidity token is zero.");
```

Listing 3.3: ebonyvsivory.cpp

修复方法 根据上述分析，函数 `schedule()` 中增加一个 hour 大于0的检查即可。

3.4 过载检测实现和文档不一致

- ID: PVE-004
- 危害性: 参考
- 可能性: 未知
- 影响力: 未知
- 目标: `ebonyvsivory.cpp`
- 类型: Documentation Issues [8]
- CWE 子类: CWE-1068 [1]

描述

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS 上进行流动性挖矿，实现获利。合约中代币的数量和风险完全由项目方控制。为了控制存入的代币数量，文档中有说明"过载检测参数: 2%"，但是合约中有代币数量最大值的限制，没有该参数的检测。

在 YFC 机枪池智能合约中，函数 `loaded()` 中根据配置好的代币最大值进行检测。不同的代币有不同的数量限制，如代码第308 – 311行所示，没有具体的过载参数检测，是完全依赖于配置值。具体的限制值则是由 `setconfig()` 函数完成，因此额度均由项目方随时控制。

```
298 ACTION ebonyvsivory::loaded(name code)
299 {
300     require_auth(get_self());

302     configs _configs(_self, _self.value);
303     auto itr = _configs.begin();

305     guns _guns(get_self(), get_self().value);
306     auto m_itr_0 = _guns.require_find(token0_symbol.code().raw(), "Gun0 does not exist.")
            ;
307     auto m_itr_1 = _guns.require_find(token1_symbol.code().raw(), "Gun1 does not exist.")
            ;
308     if (code == token0_contract)
309         check(m_itr_0->bullet <= itr->bal0_max, "token0_overload");
310     if (code == token1_contract)
311         check(m_itr_1->bullet <= itr->bal1_max, "token1_overload");
312 }
```

Listing 3.4: ebonyvsivory.cpp

修复方法 根据上述分析，应该删除"过载检测参数: 2%"的说明，并说明由项目方后台风控机制控制。

3.5 price计算存在除0风险

- ID: PVE-005
- 危害性: 低
- 可能性: 低
- 影响力: 中
- 目标: `ebonyvsivory.cpp`
- 类型: Numeric Errors [9]
- CWE 子类: CWE-369 [2]

描述

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS 上进行流动性挖矿，实现获利。用户代币存入合约后，项目方后台会调用合约在一定时间内将其用于流动性挖矿。代币在流动性挖矿期间无法提取，但可预约延后取回。项目方会根据用户的预约延后时间，就近选择挖矿间歇期调用合约完成代币返还。合约在进行流动性挖矿前，会获取挖矿项目交易对的代币数量，并计算代币价格，用于后续的熔断检查。理论上说，如果交易对中代币数量均为0，价格计算会出现除0风险，导致后续存入的代币比例出现问题，熔断检查也会失去作用。

YFC 机枪池智能合约中函数 `lu()` 实现了针对不同挖矿项目计算交易对中代币所需的数量，然后分别调用 `lu_dfs()` 和 `lu_box()` 来执行真正的挖矿操作，同时保存当前代币价格，作为后续熔断检查的基础。计算代币当前价格的代码如第96 – 111行所示。其中 price 的计算在第111行，`reserve0` 表示某种代币的数量，因此理论上有为0的可能。

```
87  void ebonyvsivory::lu(string nick, name swap, uint64_t mid)
88  {

90      asset token0_bal = utils::get_balance(token0_contract, get_self(), token0_symbol.code
            ());
91      asset token1_bal = utils::get_balance(token1_contract, get_self(), token1_symbol.code
            ());

93      uint64_t reserve0 = 0;
94      uint64_t reserve1 = 0;
95      // fetch market info
96      if (nick == "Ivory")
97      {
98          markets _markets(swap, swap.value);
99          auto m_itr = _markets.require_find(mid, "ebonyvsivory lu_dfs: market does not
                exist.");
100         reserve0 = m_itr->reserve0.amount;
101         reserve1 = m_itr->reserve1.amount;
102     }
103     if (nick == "Ebony")
104     {
105         pairs _markets(swap, swap.value);
106         auto m_itr = _markets.require_find(mid, "ebonyvsivory lu_box: market does not
                exist.");
107         reserve0 = m_itr->reserve0.amount;
108         reserve1 = m_itr->reserve1.amount;
109     }

111     double price = (double)reserve1 / reserve0;


114     uint64_t amount0_desired = token0_bal.amount;
115     uint64_t amount1_desired = token1_bal.amount;
116     uint64_t amount0 = 0;
117     uint64_t amount1 = 0;

119     if (reserve0 == 0 && reserve1 == 0)
120     {
121         amount0 = amount0_desired;
122         amount1 = amount1_desired;
123     }
124     else
125     {
126         uint64_t amount1_optimal = quote(amount0_desired, reserve0, reserve1);
127         if (amount1_optimal <= amount1_desired)
128         {
129             amount0 = amount0_desired;
130             amount1 = amount1_optimal;
131         }
132         else
133         {
134             uint64_t amount0_optimal = quote(amount1_desired, reserve1, reserve0);
```

```
135             check(amount0_optimal <= amount0_desired, "math error"); // both token0 and token1
                    insufficient, shoule never happen
136             amount0 = amount0_optimal;
137             amount1 = amount1_desired;
138         }
139     }

141     token0_bal.amount = amount0;
142     token1_bal.amount = amount1;
```

Listing 3.5: ebonyvsivory.cpp

修复方法 根据上述分析，需要特殊处理挖矿项目中交易对的代币数量为0的情况，比如发现时异常退出，避免价格计算出现问题。

3.6 内联调用层数可能过深

- ID: PVE-006
- 危害性: 参考
- 可能性: 未知
- 影响力: 未知
- 目标: `ebonyvsivory.cpp`
- Category: Behavioral Issues [7]
- CWE subcategory: CWE-440 [4]

描述

在 EOS 公链上，一个交易（Transaction）中可以包含多个 Action，在 Action 内部可以再调用其他 Action，称为内联调用（Inline Action）。默认的内联调用层数最大为4，即当交易中内联调用的层级达到4时就会抛出异常。而且在公链上有配置项 `max_inline_action_depth` 可以控制内联调用深度。在 EOS 主网中此值被设置为 10。开发者需要注意在合约中内联调用的深度，避免交易无法执行成功。

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS 上进行流动性挖矿，实现获利。用户代币存入合约后，项目方后台会调用合约在一定时间内将其用于流动性挖矿，之后再取回代币。合约中还有熔断检查的机制，在挖矿目标的交易对价格偏离过大时会及时取回代币。但是从挖矿合约中取回代币的逻辑中还增加了兑换 YFC 代币的操作，导致内联调用深度超过4，小于10，虽然还在主网允许的范围内，但兑换 YFC 的内联调用深度不受本合约的控制。因此需要及时关注关联合约的变化和 EOS 主网配置的变化，避免出现执行异常。

在 YFC 机枪池智能合约中，函数 `pullout()` 实现合约退出流动性挖矿。举例从 Defibox 中退出流动性挖矿分析，函数中会调用 `withdraw_box` 取回代币。在取回代币的逻辑中，如代码第257行所示会接着调用 `buyyfc2`，然后内部调用 DFS 的 YFC 代币兑换逻辑，本身已经有三层调用。在 DFS 的代币兑换逻辑中也牵扯到挖矿、日志记录等内联调用，内联调用深度较深。类似的 `supervise()` 执行熔断检查发现异常时，也会出现上面的退出流动性挖矿的情况。

```
242 void ebonyvsivory::withdraw_box()
243 {
244     // withdraw from dfs swap
245     asset lp_token_bal = utils::get_balance(name("lptoken.defi"), get_self(), symbol_code
        ("BOXL"));
246     utils::inline_transfer(name("lptoken.defi"), get_self(), BOX_SWAP_ACCOUNT,
        lp_token_bal, string(""));

248     // erase task
249     tasks _tasks(get_self(), get_self().value);
250     auto t_itr = _tasks.begin();
251     _tasks.erase(t_itr);

253     // claim all from box
```

```
254     action(permission_level{get_self(), "active"_n}, name("lptoken.defi"), name("claimall
            "), make_tuple(get_self(), symbol_code("BOXL"), 20)).send();

256     // sell all and buy yfc
257     action(permission_level{get_self(), "active"_n}, get_self(), name("buyfc2"),
            make_tuple()).send();

259     // update status
260     update_status(1, current_time_point());

262     // sync
263     action(permission_level{get_self(), "active"_n}, get_self(), name("sync"), make_tuple
            ()).send();
264 }

266 ACTION ebonyvsivory::buyfc2()
267 {
268     require_auth(get_self());
269     asset box_bal = utils::get_balance(name("token.defi"), get_self(), symbol_code("BOX")
            );
270     if (box_bal.amount > 0)
271     {
272         utils::inline_transfer(name("token.defi"), get_self(), DFS_SWAP_ACCOUNT, box_bal,
                string("swap:303-329:0"));
273         action(permission_level{get_self(), "active"_n}, get_self(), name("sliceyfc"),
                make_tuple()).send();
274     }
275 }
```

Listing 3.6: ebonyvsivory.cpp

修复方法 根据上述分析，可将兑换 YFC 代币的逻辑独立出来，减少内联调用的深度。如果暂不修复的话，需要及时关注关联合约的变化和 EOS 主网配置的变化，保证交易的正常执行。

3.7 参与流动性挖矿存在本金损失风险

- ID: PVE-007
- 危害性: 参考
- 可能性: 未知
- 影响力: 未知
- 目标: `ebonyvsivory.cpp`
- 类型: Security Features [6]
- CWE 子类: CWE-693 [5]

描述

YFC 机枪池智能合约是构建在 EOS 公链上的自动化挖矿工具，即利用用户存入代币，在 Defibox 和 DFS 上进行流动性挖矿，实现获利。用户代币存入合约后，项目方后台会调用合约在一定时间内将其用于流动性挖矿。代币在流动性挖矿期间无法提取，但可预约延后取回。项目方会根据用户的预约延后时间，就近选择挖矿间歇期调用合约完成代币返还。合约在进行流动性挖矿前，会获取挖矿项目交易对的代币数量，并计算代币价格，用于后续的熔断检查。当发现当前 DFS 中交易对价格与之前保存的流动性挖矿项目的交易对价格偏离超过3%时，就会触发自动退出挖矿操作。但触发检查的时间和频率完全由项目方账号控制，依赖于项目方后台的实时监测。如果出现检查不及时不准确，可能会导致用户代币损失。

YFC 机枪池智能合约中函数 `supervise()` 实现了熔断检查的策略。如代码第12 – 24行所示，熔断检查是基于 DFS 项目中交易对里代币的价格和之前保存的交易对价格的比较。受限于挖矿项目本身流动性问题，可能出现短时间内代币价格被控制，导致代币兑换比例异常，致使 YFC 机枪池合约退出时因代币兑换比例变化出现损失。

```
4  ACTION ebonyvsivory::supervise()
5  {
6     require_auth(DFS_DEV);

8     tasks _tasks(get_self(), get_self().value);
9     auto t_itr = _tasks.begin();
10    check(t_itr != _tasks.end(), "task no found");

12    markets _markets(DFS_SWAP_ACCOUNT, DFS_SWAP_ACCOUNT.value);
13    const auto m_itr = _markets.require_find(mid_of_dfs, "ebonyvsivory supervise: market
          does not exist.");
14    double price = (double)m_itr->reserve1.amount / m_itr->reserve0.amount;

16    // 3double diff = 0;
17    if (price > t_itr->fuse)
18    {
19       diff = (price - t_itr->fuse) / t_itr->fuse;
20    }
21    else
22    {
23       diff = (t_itr->fuse - price) / t_itr->fuse;
24    }

26    check(diff > SYSTEM_FUSE, " supervise: don't worry, very safe now ");

28    if (t_itr->gun == "Ivory")
29    {
30       withdraw_dfs();
31    }
32    if (t_itr->gun == "Ebony")
33    {
34       withdraw_box();
35    }

37    // update status into cool down
38    const time_point_sec next{current_time_point().sec_since_epoch() + COOL_DOWN};
39    update_status(1, next);
40 }
```

Listing 3.7: ebonyvsivory.cpp

修复方法 根据上述分析，触发熔断检查的时间和频率完全由项目方账号控制，依赖于项目方后台的实时监测。项目方后台除了需要做实时的价格监测外，还需要增加多个代币价格的获取渠道，避免代币价格被恶意控制。同时注意监测流动性挖矿项目交易对的流动性情况，如果太少，需要停止挖矿操作。

3.8 其它建议

由于 EOS 公链账号体系及权限设计的原因，EOS 上的普通账号和合约账号的权限和使用方法一致。普通账号在调用系统账号 eosio 的 `setcode` 后，会变成合约账号。而且，该合约账号可以在任意时间调用 `setcode` 更改合约执行逻辑。因此，需要特别注意合约的 `setcode` 动作，并及时确认是否符合预期。

另外，合约账号中如果保存有大量代币资产时，需要注意，一般情况下该账号是能够直接转移账号中所有资产的，不需要该合约代码中有额外的逻辑。

因此，综合来看，针对项目的合约账号和保存代币资产的资金账号建议采用多签的方式来管理账号的权限，一方面能避免单个私钥泄漏导致的资产丢失，另一方面也能避免合约和资金账号拥有者直接转移资产。其中，多签的方式举例来说，可以设置合约账号或者专门的管理账号拥有较高的权重，再加上若干 BP（Block Producer）账号的权重后才能超过阈值控制账号，而独立的几个 BP 权重相加无法超过阈值，就不能直接控制合约账号。

最后，YFC 机枪池智能合约与关联合约的操作较多，而且大量代币会直接转账给其他合约，需要特别关注和监测关联合约的变化情况。避免流动性挖矿项目合约的资金池被掏空等情况发生时，仍然继续参与挖矿。

4 结论

我们对 YFC 机枪池的智能合约进行了安全审计，截至该报告撰写为止，该特定版本的智能合约发现1处高危漏洞，即合约执行部分项目流动性挖矿的功能不可用；并且存在若干安全隐患，包括预约提取资金的时间没有最小值检查和存在 CPU 资源被恶意消耗等问题，需要予以关注或修复。但正如免责声明 1.4所述，我们欢迎任何建设性的反馈或建议。

5 附录

5.1 基本漏洞检测

5.1.1 apply 验证漏洞

- 描述: 在合约调用入口 apply 验证的安全性。
- 结果: 未发现
- 危害性: 严重

5.1.2 权限校验漏洞

- 描述: 对外部可调用函数的权限检查。
- 结果: 未发现
- 危害性: 严重

5.1.3 假转账攻击漏洞

- 描述: 检测合约是否存在假转账攻击漏洞。
- 结果: 未发现
- 危害性: 严重

5.1.4 伪造转账通知

- 描述: 检测合约是否存在伪造转账通知漏洞。
- 结果: 未发现
- 危害性: 严重

5.1.5 交易回滚攻击

- 描述: 检测合约是否存在交易回滚的缺陷。
- 结果: 未发现
- 危害性: 严重

5.1.6 交易阻塞攻击

- 描述: 检测合约是否存在交易阻塞攻击的缺陷。
- 结果: 未发现
- 危害性: 严重

5.1.7 soft_fail 攻击检测

- 描述: 检测合约是否存在soft_fail 攻击的缺陷。
- 结果: 未发现
- 危害性: 严重

5.1.8 hard_fail 攻击检测

- 描述: 检测是否存在hard_fail 攻击的漏洞。
- 结果: 未发现
- 危害性: 严重

5.1.9 异常 memo 检测

- 描述: 检测合约交易是否存在异常 memo 的漏洞。
- 结果: 未发现
- 危害性: 严重

5.1.10 随机数安全问题检测

- 描述: 检测合约代码中随机数生成的安全性。
- 结果: 未发现
- 危害性: 严重

5.2 代码及业务安全性检测

5.2.1 加解密算法强度检测

- 描述: 验证合约代码加解密算法强调是否存在问题。
- 结果: 未发现
- 危害性: 高危

5.2.2 帐号权限控制

- 描述: 验证合约代码是否存在帐号权限控制问题。
- 结果: 未发现
- 危害性: 中危

5.2.3 敏感行为检测

- 描述: 验证合约代码敏感行为是否存在问题。
- 结果: 未发现
- 危害性: 中危

5.2.4 敏感信息泄漏检测

- 描述: 验证合约代码敏感信息是否存在泄露问题。
- 结果: 未发现
- 危害性: 中危

5.2.5 系统 API 调用检测

- 描述: 验证合约代码系统 API 调用是否存在问题。
- 结果: 未发现
- 危害性: 低危

References

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

[2] MITRE. CWE-369: Divide By Zero. https://cwe.mitre.org/data/definitions/369.html.

[3] MITRE. CWE-400: Uncontrolled Resource Consumption. https://cwe.mitre.org/data/definitions/400.html.

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

[5] MITRE. CWE-684: Protection Mechanism Failure. https://cwe.mitre.org/data/definitions/693.html.

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

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

[8] MITRE. CWE CATEGORY: Documentation Issues. https://cwe.mitre.org/data/definitions/1225.html.

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

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

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

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