SMART CONTRACT AUDIT REPORT

for

DEFIS NETWORK

Prepared By: Shuxiao Wang

Hangzhou, China

Jul. 31, 2020

Document Properties

Client	DeFis Network
Title	Smart Contract Audit Report
Target	DeFis Network Swap
Version	1.0
Author	Shawn Li, Huaguo Shi
Auditors	Shawn Li, Jun Li
Reviewed by	Chiachih Wu
Approved by	Chiachih Wu
Classification	Public

Version Info

Version	Date	Author(s)	Description
1.0	Jul. 31, 2020	Shawn Li, Huaguo Shi	Final Release

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** [11]）受客户委托对 **DeFis Network Swap** 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存在无交易费挖矿的高危漏洞，并存在若干安全隐患，包括交易对数量过多时无法创建新交易对等问题，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DeFis Network Swap

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 DeFis Network Swap 智能合约即是流动性协议的实现。基于此合约，用户可以自由地创建两种代币之间的交易对，实现方便快速的币币交易，同时对于提供流动性的用户按注入的资金比例分享交易产生的手续费收入。DeFis Network Swap 智能合约基本信息如下：

表 1.1: DeFis Network Swap的基本信息

条目	描述
发行方	DeFis Network
发行平台	EOS
合约语言	C++
审计方法	白盒
审计完成时间	Jul. 31, 2020

以下是审计对象（即EOS智能合约）相关信息：

- 合约代码信息 v1 (2020-07-20):
  $MD5$：$0x3a81c87c81b907654d5945a39aa36427$
  $SHA-256$：$0x4642bd9e079aba54390c615c8c671f5673c72f1ae26b68da2653c6ccd127d502$

- 合约代码信息 v2 (2020-07-27):
  $MD5$：$0x2f16d56b2439488c18fc0e927c4af1dc$
  $SHA-256$：$0x537d8a0af770066cdea68a1d72f613a4064c4936e81bc97a6a7ce86143c411ef$

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).

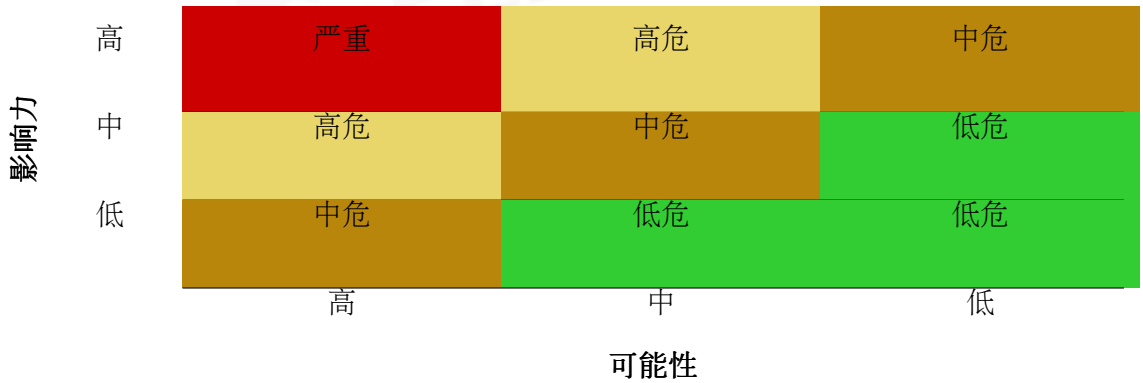
1.3 评估方法和模型

为了检测评估的标准化，我们根据OWASP Risk Rating Methodology [10]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。

表 1.2: 危害性定义



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。
- 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 DeFis Network Swap 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 DeFis Network Swap 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	1	■
低危	2	■■
参考	3	■■■■
总计	7	

2.2 主要发现

我们在本次审计中发现存在无交易费挖矿的高危漏洞，同时发现了其他6个安全隐患，其中1个中危漏洞、2个低危漏洞和3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	存在无交易费挖矿漏洞	已修复
PVE-002	低危	币币兑换时可能存在假充值漏洞	已修复
PVE-003	低危	流动性计算存在算术溢出可能	已修复
PVE-004	参考	建议添加熔断机制	已确认
PVE-005	参考	建议添加账号黑名单机制	已确认
PVE-006	中危	交易对过多时，无法创建新的交易对	已确认
PVE-007	参考	计算存入的代币数量时可能存在数值截断错误	已确认

3 检测结果详情

3.1 存在无交易费挖矿漏洞

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

描述

DFS 是 DeFis Network 的平台币，功能不仅用于社区治理和支付利息费用，还可用于交易挖矿、流动性激励、抵押分红等。DFS 代币最大发行量为10亿，并通过挖矿算法线性释放。具体来说，在用户使用 DeFis Network Swap 智能合约进行币币交易时，会同步调用其他合约账号进行挖矿，获得代币。由于每次币币交易都会有交易手续费，在一定程度上抑制用户无节制地挖矿。但是在 DeFis Network Swap 智能合约中，允许用户自由地创建交易对，恶意用户可以创建无价值的代币与 EOS 兑换的交易对，同时由于此交易对的交易手续费扣除的是无价值代币，导致用户可以在无交易费的情况下挖矿。

DeFis Network Swap 智能合约代码中所示，函数 `do_swap()` 实现了币币交易的逻辑，其中交易费由 `PROTOCOL_FEE` 和 在 `get_amount_out()`中参与计算的 `TRADE_FEE` 构成。但两者都是依赖于输入的代币数量 `amount_in` 来计算的，如代码第269-275行所示。 另外在判断该交易是否能参与挖矿的 `mining_dfs()` 函数中仅限定了交易对中有 EOS 代币，且交易额大于 1 EOS。因此用户可通过创建无价值代币与 EOS 的交易对，使用无价值代币不停兑换 EOS 即可实现无交易费挖矿。

```
263 extended_asset swap::do_swap(uint64_t mid, name from, asset quantity, name code, std::
        string memo)
264 {
265     auto m_itr = _markets.require_find(mid, "Market does not exist.");
266     check((code == m_itr->contract0 || code == m_itr->contract1), "contract error");
267     check((quantity.symbol == m_itr->sym0 || quantity.symbol == m_itr->sym1), "symbol
            error");
268     uint64_t amount_in = quantity.amount;
269     uint64_t tmp = safemath::mul(amount_in, PROTOCOL_FEE);
270     uint64_t protocol_fee = tmp / 10000;
271     if (protocol_fee > 0)
272     {
273         amount_in = safemath::sub(amount_in, protocol_fee);
274         utils::inline_transfer(code, get_self(), PROTOCOL_FEE_ACCOUNT, asset(
                protocol_fee, quantity.symbol), std::string("swap protocol fee"));
275     }
276     uint64_t reserve0 = m_itr->reserve0.amount;
277     uint64_t reserve1 = m_itr->reserve1.amount;
278     uint64_t amount_out = 0;
279     asset quantity_out;
280     extended_asset quantity_out_ex;
281     if (code == m_itr->contract0 && quantity.symbol == m_itr->sym0)
282     {
283         amount_out = get_amount_out(amount_in, reserve0, reserve1);
284         check(amount_out < reserve1, "invalid amount_out1");
285         quantity_out = asset(amount_out, m_itr->sym1);
286         quantity_out_ex = extended_asset(quantity_out, m_itr->contract1);
287         update(mid, reserve0 + amount_in, reserve1 - amount_out, reserve0, reserve1);
288     }
289     else
290     {
291         amount_out = get_amount_out(amount_in, reserve1, reserve0);
292         check(amount_out < reserve0, "invalid amount_out0");
293         quantity_out = asset(amount_out, m_itr->sym0);
294         quantity_out_ex = extended_asset(quantity_out, m_itr->contract0);
295         update(mid, reserve0 - amount_out, reserve1 + amount_in, reserve0, reserve1);
296     }

298     // mining
299     extended_asset quantity_in_ex = extended_asset(quantity, code);
300     mining_dfs(mid, from, quantity_in_ex, quantity_out_ex);
```

Listing 3.1: swap.cpp

修复方法 一方面可以建立白名单机制，给有价值的代币交易挖矿赋予较高的权重。另一方面也可以在计算挖矿手续费时，收取以 EOS 为计算交易手续费的单位。

3.2 币币兑换时可能存在假充值漏洞

- ID: PVE-002
- 危害性: 低
- 可能性: 低
- 影响力: 中
- 目标: swap.cpp
- 类型: Behavioral Issues [7]
- CWE 子类: CWE-440 [4]

描述

DeFis Network Swap 协议中用户可自由创建币币兑换的交易对，通过注入流动性资金，使其形成有效的交换市场，并依据注入的流动性资金的比例赚取交易手续费。由于交易对创建的自由性，需谨慎对待交易对中的代币发行合约。在 EOS 公链中，代币的识别通过其发行合约和 symbol 来确定，因此代币发行合约可以发行与其他合约同名的代币。如在自己的代币合约中发行 EOS 名称的代币作为测试。DeFis Network Swap 智能合约在币币交换过程中，没有严格的比对合约和代币名称相同，导致存在假充值的可能。

在 DeFis Network Swap 智能合约中，函数 `do_swap()` 实现了币币交易的逻辑，其中判断输入的代币是否与交易对中的合约及代币 symbol 匹配的逻辑如代码第266-267行所示。EOS 公链上的代币判断是要求合约和 symbol 同时相同，但代码中是分别判断输入的代币是否符合条件，即先判断合约是否匹配，再判断 symbol。这就有可能出现输入的代币是 `contract0` 的 `sym1` 也符合条件。因此交易对有被假充值的可能。但由于充入的假币被当作 `sym1` 的代币，因此兑换回的是 `sym0` 代币，即假币和 `sym0` 都是同一个合约发行，因此发生的概率很低。

```
263 extended_asset swap::do_swap(uint64_t mid, name from, asset quantity, name code, std::
        string memo)
264 {
265     auto m_itr = _markets.require_find(mid, "Market does not exist.");
266     check((code == m_itr->contract0 || code == m_itr->contract1), "contract error");
267     check((quantity.symbol == m_itr->sym0 || quantity.symbol == m_itr->sym1), "symbol
            error");
268     uint64_t amount_in = quantity.amount;
269     uint64_t tmp = safemath::mul(amount_in, PROTOCOL_FEE);
270     uint64_t protocol_fee = tmp / 10000;
271     if (protocol_fee > 0)
272     {
273         amount_in = safemath::sub(amount_in, protocol_fee);
274         utils::inline_transfer(code, get_self(), PROTOCOL_FEE_ACCOUNT, asset(
                protocol_fee, quantity.symbol), std::string("swap protocol fee"));
275     }
```

Listing 3.2: swap.cpp

修复方法 根据上述分析，在币币兑换交易中，判断代币时，同时检查代币合约和 symbol 即可。

3.3 流动性计算存在算术溢出可能

- ID: PVE-003
- 危害性: 低危
- 可能性: 中
- 影响力: 低
- 目标: `swap.cpp`
- 类型: Numeric Errors [8]
- CWE 子类: CWE-191 [1]

描述

DeFis Network Swap 协议中用户可自由创建币币兑换的交易对，通过注入流动性资金，使其形成有效的交换市场，并依据注入的流动性资金的比例赚取交易手续费。用户新创建的交易对流动性为空，即交易对中两种代币的数量均为0，此时交易对兑换的价格由新存入的两种代币的数量来决定。合约会要求首次注入的流动性必须不小于 `MINIMUM_LIQUIDITY`，其中 `MINIMUM_LIQUIDITY` 数量算做交易对中首次注入流动性的手续费，留给合约本身，剩余的数量才算做用户的。如果注入的流动性低于 `MINIMUM_LIQUIDITY`，则会造成溢出。

DeFis Network Swap 智能合约中，增加流动性的代码在 `add_liquidity()` 函数中实现。新建的交易对流动性为0，即代码中 `total_liquidity_token` 为0，此时计算用户的流动性如代码第196-197行所示。如果 `((uint64_t)(sqrt((uint128_t)amount0 * (uint128_t)amount1)))` 小于 `MINIMUM_LIQUIDITY` 就会造成负值导致溢出。好在后续两次的 `mint_liquidity_token()` 计算恰好检查到有溢出，但仍需注意此问题。

```
147 void swap::add_liquidity(name user)
148 {
149     auto itr = _orders.require_find(user.value, "You don't have any order.");
150     auto m_itr = _markets.require_find(itr->mid, "Market does not exist.");

152     // step1: get amount0 and amount1
153     uint64_t amount0_desired = itr->quantity0.amount;
154     uint64_t amount1_desired = itr->quantity1.amount;
155     uint64_t amount0 = 0;
156     uint64_t amount1 = 0;
157     uint64_t reserve0 = m_itr->reserve0.amount;
158     uint64_t reserve1 = m_itr->reserve1.amount;
159     uint64_t refund_amount0 = 0;
160     uint64_t refund_amount1 = 0;
161     if (reserve0 == 0 && reserve1 == 0)
162     {
163         amount0 = amount0_desired;
164         amount1 = amount1_desired;
165     }
166     else
167     {
168         uint64_t amount1_optimal = quote(amount0_desired, reserve0, reserve1);
169         if (amount1_optimal <= amount1_desired)
170         {
171             amount0 = amount0_desired;
172             amount1 = amount1_optimal;
173             refund_amount1 = amount1_desired - amount1_optimal;
174         }
175         else
176         {
177             uint64_t amount0_optimal = quote(amount1_desired, reserve1, reserve0);
178             check(amount0_optimal <= amount0_desired, "math error"); // both token0 and
                    token1 insufficient, shoule never happen
179             amount0 = amount0_optimal;
180             amount1 = amount1_desired;
181             refund_amount0 = amount0_desired - amount0_optimal;
182         }
183     }

185     // step2: refund
186     if (refund_amount0 > 0)
187         utils::inline_transfer(m_itr->contract0, get_self(), user, asset(refund_amount0,
                m_itr->sym0), std::string("extra deposit refund"));
188     if (refund_amount1 > 0)
189         utils::inline_transfer(m_itr->contract1, get_self(), user, asset(refund_amount1,
                m_itr->sym1), std::string("extra deposit refund"));

191     //  step3: mint liquidity token
192     uint64_t token_mint = 0;
193     uint64_t total_liquidity_token = m_itr->liquidity_token;
194     if (total_liquidity_token == 0)
195     {
```

```
196         token_mint = ((uint64_t)(sqrt((uint128_t)amount0 * (uint128_t)amount1))) -
                MINIMUM_LIQUIDITY;
197         mint_liquidity_token(itr->mid, get_self(), MINIMUM_LIQUIDITY); // permanently
                lock the first MINIMUM_LIQUIDITY tokens
198     }
199     else
200     {
201         uint64_t x = (uint64_t)((uint128_t)amount0 * (uint128_t)total_liquidity_token /
                (uint128_t)reserve0);
202         uint64_t y = (uint64_t)((uint128_t)amount1 * (uint128_t)total_liquidity_token /
                (uint128_t)reserve1);
203         token_mint = std::min(x, y);
204     }
205     check(token_mint > 0, "INSUFFICIENT_LIQUIDITY_MINTED");
206     // step4: update user liquidity token
207     mint_liquidity_token(itr->mid, user, token_mint);
```

Listing 3.3: swap.cpp

修复方法 根据上述分析，在交易对首次存入代币计算用户的流动性时，可以先判断注入的流动性是否不小于 `MINIMUM_LIQUIDITY`，再做后续的计算。

3.4 建议添加熔断机制

- ID: PVE-004
- 危害性: 参考
- 可能性: 无
- 影响力: 无
- 目标: `swap.cpp`
- 类型: Security Features [6]
- CWE 子类: CWE-693 [5]

描述

在 EOS 主链上，当前针对链上智能合约的攻击或者利用漏洞薅羊毛的行为频繁发生，一旦发现智能合约中的运行数据出现异常，需要及时关闭合约相关功能，排查原因，避免造成大范围的损失。虽然智能合约可以随时更改并重新部署，关闭合约的方式就可以通过部署空合约或者立即修改合约代码再部署来实现，但频繁地更新合约一方面可能会造成用户正常的使用受阻，另一方面也会影响用户对项目方的信任，同时也有可能会造成合约接收了用户代币而没有执行相关功能的问题，导致后续的追查返还。而且有时真正出现问题的，可能仅是部分功能，仅需暂停此功能，系统仍能稳定运行。因此智能合约需要一套完整的熔断机制来辅助合约的长期稳定运行，在必要时暂停部分功能，拒绝用户的代币转入或转出，在排除异常后可以马上恢复运行。

DeFis Network Swap 协议中用户可自由创建币币兑换的交易对，通过注入流动性资金，使其形成有效的交换市场，并依据注入的流动性资金的比例赚取交易手续费。DeFis Network Swap 智能合约中涉及操作包括创建交易对，用户存入代币提供流动性，提取已存入的代币减少流动性以及币币兑换，交易挖矿等，各种操作之间有些是相互独立的，但没有相关的机制来保障在出现异常时关闭相关功能。举例来说，如有人不停创建垃圾交易对，其实并不影响其他用户正常使用币币兑换的功能；再比如交易挖矿出现异常，也不影响用户创建交易对并实现币币兑换。

修复方法 建议根据智能合约的功能划分添加合理的熔断机制，在需要时快速启停部分功能或阻止某些交易。

3.5 建议添加账号黑名单机制

- ID: PVE-005
- 危害性: 参考
- 可能性: 无
- 影响力: 无
- 目标: `swap.cpp`
- 类型: Security Features [6]
- CWE 子类: CWE-693 [5]

描述

自从 EOS 主网上线以来，智能合约相关的黑客攻击事件已发生上百起，因此智能合约的自保护机制尤为重要。一般建议在 EOS 智能合约中拥有熔断机制和黑名单机制。当发现智能合约运行出现严重问题时，可及时关停智能合约，避免项目方和用户遭受进一步的损失。另一方面，在系统正常的运行过程中，如果发现某些用户账号的交易出现异常时，可通过黑名单机制及时禁止该账号的操作，同时不会影响其他账号的使用，保持系统整体运行的稳定。如曾经发生过的利用合约的 CPU 进行挖矿的行为，通过黑名单机制屏蔽恶意用户的合约账号，可以有效防止恶意用户的同时不影响其他用户的正常使用。另外黑名单机制中，当异常账号的问题查清后，还可以恢复其正常使用。因此以黑名单机制来增加熔断机制的灵活性，保障系统的持续平稳安全地运行。

在 DeFis Network Swap 智能合约中，用户创建新的交易对，存入代币提供流动性，提取代币等操作，消耗的都是用户的 CPU 和 RAM 资源，避免了用户恶意消耗合约资源的可能。但用户恶意大量创建垃圾交易对或是利用漏洞薅羊毛的行为仍可能存在。因此建议可根据智能合约的实际情况添加黑名单机制，来抵御未知的风险，同时增加智能合约防护的灵活性。

修复方法 建议在 DeFis Network Swap 智能合约中添加黑名单机制，限制异常账号的交易行为。

3.6 交易对过多时，无法创建新的交易对

- ID: PVE-006
- 危害性: 中危
- 可能性: 低
- 影响力: 中
- 目标: `swap.cpp`
- 类型: Resource Management [9]
- CWE 子类: CWE-400 [3]

描述

在 EOS 主链上，单笔交易的执行时间是有限制的，当超出允许的执行时间时，会导致该交易执行失败，因此需要避免在交易中出现循环耗时操作。一般在循环体中涉及了数据表的操作，

耗时会迅速增加，而当此操作需要循环的次数较多时，就会导致执行时间过长而出现本次交易执行失败，使此次交易所属功能执行失败。

在 DeFis Network Swap 智能合约中任何用户都可以创建任意两种代币的交易对。为了避免交易对之间的重复，在创建前会循环遍历已有的交易对来判断和新交易对是否相同。当交易对过多时会出现执行超时，导致无法创建新的交易对。

```
3  ACTION swap::newmarket(name creator, name contract0, name contract1, symbol sym0, symbol
       sym1)
4  {
5      require_auth(creator);
6      check((contract0 != contract1) || (sym0 != sym1), "invalid pair");
7      auto supply0 = utils::get_supply(contract0, sym0.code());
8      check(supply0.amount > 0, "invalid token0");
9      check(supply0.symbol == sym0, "invalid symbol0");
10     auto supply1 = utils::get_supply(contract1, sym1.code());
11     check(supply1.amount > 0, "invalid token1");
12     check(supply1.symbol == sym1, "invalid symbol1");

14     auto itr = _markets.begin();
15     bool pair_exists = false;
16     while (itr != _markets.end())
17     {
18         if (itr->contract0 == contract0 && itr->sym0 == sym0 && itr->contract1 ==
               contract1 && itr->sym1 == sym1)
19         {
20             pair_exists = true;
21             break;
22         }
23         if (itr->contract0 == contract1 && itr->sym0 == sym1 && itr->contract1 ==
               contract0 && itr->sym1 == sym0)
24         {
25             pair_exists = true;
26             break;
27         }
28         itr++;
29     }
30     check(!pair_exists, "market already exists");

32     _markets.emplace(creator, [&](auto &a) {
33         a.mid = get_mid();
34         a.contract0 = contract0;
35         a.contract1 = contract1;
36         a.sym0 = sym0;
37         a.sym1 = sym1;
38         a.reserve0.symbol = sym0;
39         a.reserve1.symbol = sym1;
40         a.last_update = current_time_point();
41     });
42 }
```

Listing 3.4: swap.cpp

DeFis Network Swap 智能合约中，如上代码第16 – 29行所示，在创建新的交易对前，先循环遍历已有的交易对判断要创建的交易对是否存在。在本地 EOS 测试网中，发现当创建的交易对在 500 个左右时，会出现执行超时，不能再增加新的交易对。当然此操作的耗时会根据机器的配置不同有较大的差异，需要注意并防止恶意用户的攻击。

修复方法 根据上述分析，可将交易对信息字符串拼接或创建哈希，并作为交易对表的一项索引。当创建新的交易对时，根据信息从索引中判断交易对是否存在。

3.7 计算存入的代币数量时可能存在数值截断错误

- ID: PVE-007
- 危害性: 参考
- 可能性: 低
- 影响力: 未知
- 目标: `swap.cpp`
- 类型: Numeric Errors [8]
- CWE 子类: CWE-197 [2]

描述

DeFis Network Swap 协议中用户可自由创建币币兑换的交易对，通过注入流动性资金，使其形成有效的交换市场，并依据注入的流动性资金的比例赚取交易手续费。用户新创建的交易对流动性为空，即交易对中两种代币的数量均为0，此时交易对兑换的价格由新存入的两种代币的数量来决定。如果用户存入的两种代币兑换比例差距过大，之后再存入大量代币时可能导致出现数据截断错误。

在 DeFis Network Swap 智能合约中处理用户提供流动性的逻辑在 `add_liquidity()` 函数中，其中用来计算两种代币数量的逻辑如代码 `quote()` 所示。当前正存入的两种代币数量分别为 `amount0` 和 `amount1`，未包含此次存入的两种代币的余额数量分别为 `reserve0` 和 `reserve1`。如代码第369行所示，如果 `amount0` 和 `reserve1` 过大，而 `reserve0` 很小，可能在将结果转成 `uint64_t` 时出现数据截断错误。

```
365 uint64_t swap::quote(uint64_t amount0, uint64_t reserve0, uint64_t reserve1)
366 {
367     check(amount0 > 0, "INSUFFICIENT_AMOUNT0");
368     check(reserve0 > 0 && reserve1 > 0, "INSUFFICIENT_LIQUIDITY");
369     return (uint64_t)((uint128_t)amount0 * (uint128_t)reserve1 / (uint128_t)reserve0);
370 }
```

Listing 3.5: swap.cpp

修复方法 根据上述分析，可在将结果转成 `uint64_t` 之前，先检查当前的计算结果是否大于 `UINT64_MAX`，如果是可抛出异常，否则可继续正常执行。

4 结论

我们对 DeFis Network Swap 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存在无交易费挖矿的高危漏洞已修复，其他存在的若干安全隐患，包括交易对数量过多时无法创建新交易对等问题，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 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

5.1.4 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

5.1.5 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

5.1.6 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

5.1.7 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

5.1.8 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

5.1.9 资源异常消耗检测

- 描述: 检测合约资源是否会被异常消耗。
- 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-191: Integer Underflow (Wrap or Wraparound). https://cwe.mitre.org/data/definitions/191.html.

[2] MITRE. CWE-197: Numeric Truncation Error. https://cwe.mitre.org/data/definitions/197.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: Numeric Errors. https://cwe.mitre.org/data/definitions/189.html.

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

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

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