行业周期与机遇

一、Web3 周期性机遇

Web3 创业今年是一件代表趋势的事，无论在中国还是美国，很多互联网大厂的年轻人纷纷辞职，投身 web3。但最近也有不少人觉得 web3 实在太难了，可能撑不下去了，又开始考虑重回 web2 找工作。

web3 目前的基本面分析

web3 到底是什么？做技术的看到的是分布式，做商业的看到更多是控制权、所有权。

我们团队经过实践和研究，认为 web3 跟 web2 有四个层次的不同：

底层技术是去中心化，准确来说，应该是分布式

底层技术是去中心化，准确来说，应该是分布式。以前机房由一个公司提供，现在分散了，很多人参与。

web3 应用都有金融属性

web3 应用都有金融属性，从比特币开始，就跟互联网有很大的不同。互联网更多是技术属性，后来才有商业属性，到支付宝、微信支付时才有了金融属性，但区块链从一诞生开始就是金融属性。

区块链的商业模式

区块链的商业模式因为底层技术的差异，跟互联网也有很大差异。

治理权

用户和平台之间的关系。互联网的应用，用户按平台的规则来走，平台制定规则，控制所有的一切，包括用户的数据。

web3 的商业周期、规律和机会

每个周期结束后，就算在周期的低位，与上一个周期比，从技术成熟度、应用数量、用户数量、链上资产的规模，都有一个量级的增长。虽然有泡沫的出现与破灭，整体在螺旋上升，应用的赛道、领域在不停更迭，演进。

Web3 的核心是，要把 web2 忘掉，先看看 web3 应该怎么做，然后再把 web2 的成功经验引入进来。

现在区块链的用户进入门槛太高了，用户体验是这个行业很缺的。

从投资方向看机会分布

- L1，公链
- 中间层
- 应用层
- 加密金融

web3 周期下的技术变革

比特币

分布式账本

比特币脚本

以太坊

智能合约

Defi

Ordinals

指一个以独特方式对“聪”排序、创造出 NFT 所需的非同质化属性的系统。 简单地说，Ordinals 为每一聪分配一个特定的数字，使之与其他聪区分开来。 这种编号系统确保每聪都有一个独特的标识符。 铭文代表附在聪上的实际内容。

- 铭文

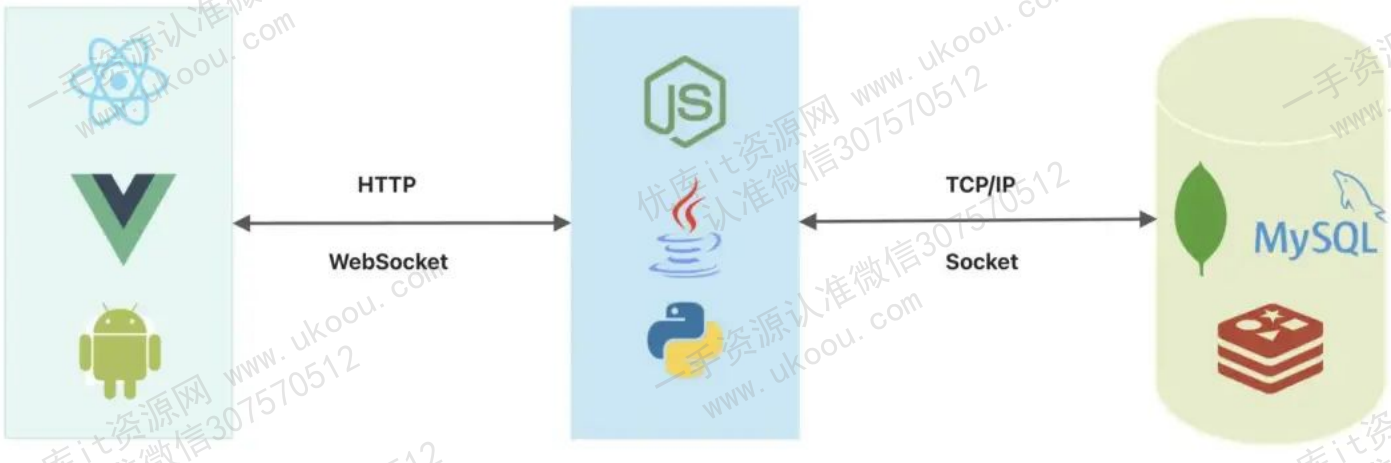
UniSwap

Uniswap 是在以太坊区块链中运行的一系列计算机程序，支持去中心化代币交换。

- DEX

二、Web3 基础设施和架构

我们在聊 Web3 基础架构之前，要先看一下 Web2 的架构



Web 和 Web3 原则的演变

我们深入研究基础设施之前，多谈一点 Web3 的原则可能是有用的——毕竟，这就是所有项目的建设目的。

数字化：

今天，我们正处于从物理信息和资产到数字的过渡状态，Web3 存在于一个数据流动的时代。物理数据可以被测量，与数字信息结合并执行。后端计算将处理不同来源的信息——AR/VR、物联网传感器和自主设备等等。

新兴技术的融合和成熟：

技术不是无根之水，加密货币和区块链技术是架构和价值观的一个关键部分。但其他技术也在不断成熟——5G、AI、边缘计算、物联网。结合更高效、更智能地处理更大的数据集、灵活的基础设施和新的商业模式（例如代币）将为下一代应用提供基础。

数据和资产的所有权：

Web3 将有助于提供数据、资产和工作的真正所有权。区块链是革命性的，因为它可以追踪和证明数字所有权。数字资产——无论是法币、加密货币，还是 NFT，都可以通过钱包由用户控制。可追踪性允许对资产或个人的贡献进行测量和控制，与数字身份相结合，这是很强大的。

增加控制和治理的去中心化：

加密货币和区块链从根本上说是基于去中心化、信任和协调的问题。一个代币代表的不仅仅是一种资产，还往往是一种投票权，或会员证。中心化有其自身的好处，但随着 DAO 的迭代，公司治理、网络控制和分布式决策的新模式正在形成。更加去中心化的控制将允许平台的建设者、运营商和用户能够拥有并对平台的发展有发言权。

可组合性：

在互联网之前，是有围墙的局域网。越来越多的区块链似乎也是如此，互操作性是一个关键焦点。通过开源代码，Web3 的构建考虑到了可组合性。这意味着团队可以采取现有的项目或程序，并在其基础上构建。这允许更快的开发和跨应用的沟通（互操作性）。

更成熟的 Web3 将为改变网络上的数据、价值和所有权关系奠定基础。企业、用户和网络可以更自由地分享或互动数据或转移价值。因此，Web3 有很多事情要做，而今天我们正在观察一个处于发展的空间，这就是人们关心 Web3 基础设施的原因。

Web3 的基础设施架构

我们研究 Web3 的基础设施时，有几个指导性的问题支撑着这个框架，作为思考不断发展的前景的一种方式：

项目团队通常使用哪些 Web3 服务和工具（“架构”）以及哪些数据、存储或计算（“核心基础架构”）系统支持这些？

Web3 中的建设者或机构存在哪些运营挑战？ 已经开发了哪些“关键任务”的解决方案？

对于某些用例（例如 DeFi、NFT）已经并且需要建立哪些基础服务才能大规模成功？

现有公司和项目在哪些能力上投入资源——无论是通过并购还是有机建设？

Web3“基础设施”的组件如何相互交互？ 每个组成部分的建立程度如何（即该领域中的项目数量、通用标准、解决方案的成熟度）？

在 Web3 中，人们可以考虑在基础计算（区块链）层之上的三类服务和产品中的数字基础设施——数据、价值/流动性和参与者/区块链支持服务。

计算层

Web3 的基础是计算层。这包括区块链和受区块链启发的网络（第 1 层，子网/侧链和扩展解决方案），其他一切都建立在其上。因此出现了跨链/全链协议以及链上消息传递项目，以允许价值和信息从一条链交换到另一条链。这一点很重要，区块链网络以一种更无信任的方式，在不同的利益相关者之间交织着数据和价值的转移。关于这个正在开发的计算/区块链层含有大量信息，但可以说这是花费了大量时间并将继续投资的地方。

1. 数据参与者：为团队建设和运营 Web3 项目提供核心基础设施、架构和工具；

2. 价值和流动性参与者：为资本在 Web3 应用程序和参与者之间的流动建立区块，减少摩擦；

3. "互动者 "或区块链支持基础设施：支持和维护区块链/计算网络的利益相关者。

数据基础设施

Web3 是语义学的一个演变。这意味着机器和计算机将需要分析、传递和计算大量的数据。

对于数据基础设施，有更多的 "核心 "或既定的数据基础设施和 "新兴 "的服务和工具，形成开发团队的架构。这里的数据基础设施包括各种工具、服务和组件，使区块链数据易于利用或建立在其之上。

应用层

1. 应用发展方向去中心化主旋律不变

Defi、DAO、NFT、铭文

2. 应用层架构

三、从 Defi 到铭文，新周期下的 Web3 有何特点

在过去的几年里，DeFi 经历了飞速的发展和演变。从最初的实验性质的项目，到现在成为 Crypto 领域必不可少的基石。

Uniswap、Curve、Aave、Compoud 等众多项目在这一过程中脱颖而出，但这一赛道的竞争也日益剧烈，如 DEX 不断

降低手续费吸引交易量、借贷协议提高贷款价值比来提高资本效率，各个项目也都积极开发新的产品来抢占更多市场。

理解 Defi 的发展过程，更有助于我们判断在 2024 年可能展现出哪些趋势？

协议平台化

随着 DeFi 领域的发展和成熟，主要的 DeFi 协议开始不满足于其核心业务，希望从单一功能的项目转向提供一揽子综合性服务的平台。

在过去一年，大家所熟悉的 DeFi 协议中，MakerDAO 的 SubDAO Spark 上线，截至 12 月 29 日，在以太坊上的 TVL 达到 16.5 亿美元，成为主要的借贷协议。

Curve 和 Aave 分别开发了自己的稳定币 crvUSD 和 GHO，Uniswap 推出了自己的钱包应用且之前还收购了 NFT 平台 Genie。新公链 Aptos 上的 Thala 凭一己之力开发了稳定币、DEX、Launchpad、流动性质押功能，几乎包含除借贷外的所有常用 DeFi 业务。

DeFi 协议的平台化已成为一种趋势，这也是 DeFi 发展成熟和不断内卷的象征，这一趋势很可能在未来延续并愈演愈烈。

头部 DEX 与借贷协议将继续保持优势

Uniswap、Aave、MakerDAO 等头部 DeFi 协议均是上一轮牛市之前的产物，它们已经在市场的持续演化中加强了自己的地位，展现了很强的网络效应和品牌影响力，并在不断更新。一段时间内，它们仍然会占据主要的市场份额，很难被取代。

Uniswap 宣布了 v4 版本，允许通过「钩子」增加各种自定义的功能；Uniswap X 提出了一种和 Cowswap 类似的在链外签署订单，并通过荷兰式拍卖进行链上结算的方案。Aave v3 提高了资本效率，在多个链上扩张，进一步巩固了其作为 DeFi 生态系统中主要借贷平台中的地位。

Dune 联合创始人 hagaetc 的仪表板显示，Uniswap 在主要 EVM 链的 DEX 中仍占据 55% 左右的市场份额。

流动性挖矿逐渐成过去式，资金会流向更高效的地方

在以太坊、Solana、BNB 链这类已有成熟生态的公链中，流动性挖矿已经逐渐成为过去。项目依靠「真实收益」来吸引资金，而资金更可能流向更高效的地方。

近期，Solana 中 SOL 的价格上涨和生态发展引发了对以太坊及其生态的 FUD。在 MEME 币频繁交易的背景下，Solana 上 DEX 展现了很强的资本效率。目前的流动性提供者主要依靠真实的、由交易手续费产生的收入，那么这些项目很可能在短期内吸引更多资金。

以 12 月 30 日数据为例，过去 24 小时，Orca 中流动性最多的 SOL/USDC、SOL/USDT 池，提供流动性仅靠交易手续费的每日平均收入均接近或超过 0.5%，且手续费比例为 0.04% 的 SOL/USDC 池收取的手续费单日达到流动性的 2.396%。

这在其它链上是难以想象的，如以太坊上，ETH/ 稳定币流动性前 3 的交易对，提供流动性的每日收入分别为流动性 0.068%、0.077%、0.127%。

在盈利能力完全不对等的情况下，专业的流动性提供者更可能转向盈利能力强、资本效率更高的地方。这和上一点并不矛盾，头部 DeFi 项目基本面更好，更安全和稳定，但增速也相对慢。新兴项目在风口来临时会保持更快的增速，同时对未来增长的预期也会反应到代币价格上，但这种增长能维持多久却是一个问题。

LST 将引领新公链的 TVL 增长

虽然在众多采用权益证明机制的区块链上早已有流动性质押类项目出现，但流动性质押代币（LST）在以太坊上海升级前才开始被集中讨论，如今流动性质押龙头 Lido 已经成为 TVL 最高的项目，没有之一。

同样，在 Solana 上也出现了这个趋势，两个流动性质押项目 Marinade 和 Jito 分别占据了 Solana 生态 TVL 的前两名。流动性质押类项目也引领了最近 Solana TVL 的增长，一方面，Jito 发币前的空投预期吸引了质押量；另一方面，Marinade、Jito 和其它流动性质押项目都在持续激励 LST 在 Solana DeFi 协议中的使用，促进了 Solana TVL 的整体提高。

其它想要提升 TVL 的公链似乎也发现了 LST 对生态促进的秘诀。如 Sui 生态中，Cetus 上的 haSUI-SUI 交易对的 APR 为 49.04%，其中 48.09% 都来自 Sui 官方奖励的 SUI 代币。Avalanche 生态中，借贷龙头 Benqi 也开发了 LST 业务，目前 LST 带来的 TVL 已经超过借贷。

Perp DEX 可能会出现有竞争力的项目

去中心化永续合约交易所，即 Perp DEX，曾被很多人看好，也跑出了 dYdX、Synthetix、GMX 等项目。dYdX 为订单簿类型，就流动性池类型的 Synthetix 和 GMX 而言，虽然已经是主要的 Perp DEX，但使用起来仍各有优缺点。

GMX v1 被诟病在单边行情时多空比例失衡，对流动性提供者不友好；同时做多、多空均需收取借币费用、交易手续费比例高，对交易员也不够友好。但它无滑点流动性的特性却是其它项目不具备的。

GMX v2 为了多空平衡引入了交易滑点，使多空平衡的交易将得到补偿，使多空失衡的交易将被惩罚。但用户在开仓时无法预估平仓时多空是否平衡，这就带来了不确定性，惩罚性的交易滑点可能达到仓位的 0.8% 甚至更高。考虑到杠杆倍数，如 10 倍杠杆，滑点 0.8%，单次交易就会损失本金的 8%。

和 GMX v2 相比，Synthetix 中的资金费率波动更大，同样，用户在开仓后可能会因为资金费率增加造成损失。另外，Synthetix 使用了 Pyth 的链下预言机，从下单到执行之间存在 8 秒钟的延迟，不能做到所见即所得。

最近的一些 Perp DEX 展现出了吸引人的特点，如 Drift 的 DLP 池，其中 BONK-PERP 显示 30 天提供流动性的回报达到 2000%，HNT-PERP 的回报为 439%。虽然在 Drift 的 DLP 池使用杠杆提供流动性风险很大，可能亏掉全部本金，但也可能获得更高的回报。另外，Aark Digital、MXY Finance 等项目提供了资金效率更高的 Perp DEX 方案。

现实世界资产

现实世界资产（RWA）其实是存在争议的一类项目。首先，它存在链下部分，可能需要依赖于单一实体，也可能面临监管，这和 DeFi 去中心化的特点并不完全符合。

虽然我们相信现实世界中有更好的机会，万物皆可代币化，但就现阶段而言，美债似乎是唯一能得到大规模应用的方向。其它房地产、艺术品等，虽然也可以代币化之后放到链上，但由于是非标准化的产品，原来不具备流动性，在链上依然没有流动性。

随着美国的加息预期，短期美债收益率预计在 2024 年将会大幅下降，这将直接影响 MakerDAO 等 RWA 产品的收益率。而加密市场在这期间可能进入牛市，对稳定币的需求会增加，这类产品的吸引力可能下降。从 MakerDAO 近期的数据看，DAI 的发行量从 10 月下旬以来已经开始下降。

但这并不妨碍 Crypto 创业者们对这一赛道的探索和兴趣，这一过程中可能会为 RWA 引入有力的传统金融机构作为合作者，这至少会是一个伟大的叙事。

五、安全攻击案例看 Web3 项目特点趋势

业务逻辑设计不当

Pair 代币意外销毁

近期因为业务逻辑设计不当发生的安全事件，大多都是由于 pair 代币余额异常导致的。

一些代币项目，会在业务设计中添加交易手续费或代币通缩的功能，也就是在代币转移过程中，会收取相关比例的手续费或者直接销毁部分转移的代币。这本是一个项目的创收或激励用户持有代币或其他有助于项目发展的业务逻辑，但如果代码设计不够完善，便会出现严重的问题。

例如，在转账过程中，有些项目进行额外代币扣除时，没考虑到 pair 意外扣除将导致严重后果。该类项目的转账函数逻辑一般是发送方和接收方进行正常的代币转移，并且额外扣除某个地址部分费用，用于手续费或销毁。但额外扣除地址为 pair 时，会导致 pair 中其中一个代币余额通过非交易的方式变少，k 值异常变化，从而使用少量代币便能兑换出大量另一个代币。

以下列项目代码为例：

该代币合约中存在意外销毁 pair 余额的问题，当代币转移发起者不是 pair 合约时，会进行更新池子的操作，该更新池子的方式是将 pair 的代币余额扣除交易金额的 1%，再更新储备量。黑客可以通过调用 transfer 函数自己给自己转账，反复操作，将会使得 pair 中的一个代币消耗得极少，最后利用极少代币将 pair 中大量的价值币兑换出来。

```
1176    function _transfer(
1177        address sender,
1178        address recipient,
1179        uint256 amount
1180    ) private returns (bool) {
1181        require(sender != address(0), "ERC20: transfer from the zero address");
1182        require(recipient != address(0), "ERC20: transfer to the zero address");
1183        //Trade start check
1184        if (!tradingOpen) {...}
1190
1191        if (inSwapAndLiquify) {...
1193        } else {
1194            if (sender == addressDev && recipient == uniswapPair) {
1195                sale = block.number;
1196            }
1197
1198            if (sender == uniswapPair && recipient != addressDev) {...}
1203
1204            if (sender != owner() && recipient != owner())
1205                _checkTxLimit(sender, amount);
1206
1207            uint256 currentTokenBalance = balanceOf(address(this));
1208            bool overMinimumTokenBalance = currentTokenBalance >=
1209                minimumTokensBeforeSwap;
1210
1211            if (!isMarketPair[sender] && sale > 0) updatePool(amount);
1212
1213            if (
1214                overMinimumTokenBalance &&
1215                !inSwapAndLiquify &&
1216                !isMarketPair[sender] &&
1217                swapAndLiquifyEnabled
1218            ) {...}
1223
1224            _balances[sender] = _balances[sender].sub(
1225                amount,
1226                "Insufficient Balance"
1227            );
1228
1229            uint256 finalAmount = (isExcludedFromFee[sender] ||
1230                isExcludedFromFee[recipient])
1231                ? amount
1232                : takeFee(sender, recipient, amount);
1233
1234            if (checkWalletLimit && !isWalletLimitExempt[recipient])
1235                require(balanceOf(recipient).add(finalAmount) <= _walletMax);
1236
1237            _balances[recipient] = _balances[recipient].add(finalAmount);
1238
1239            emit Transfer(sender, recipient, finalAmount);
1240            return true;
1241        }
1242    }
```

以下是真实安全事件过程：

首先使用 2555 枚 WBNB 兑换了 1390 亿枚 Bamboo 代币；

From 0x7F7D46...564FF5A2 To PancakeSwap V2: BAMBOO 41 For 2,555.757523479346390471 $546,026.24 Wrapped BNB... (WBNB...)

From PancakeSwap V2: BAMBOO 41 To 0x7F7D46...564FF5A2 For 139,762,309,088.674044298 Bamboo AI... (BAMBOO...)

不断通过 transfer 将代币发送给自己，导致 pair 中 Bamboo 代币异常减小；

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,691,584,774.22836492 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,626,007,243.440926943 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,561,072,372.45520586 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,496,773,863.205144843 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,433,105,479.345734424 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,370,061,045.648146227 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,307,634,447.400794394 Bamboo AI... (BAMBOO...)

From 0x7F7D46...564FF5A2 To 0x7F7D46...564FF5A2 For 6,245,819,629.81626661 Bamboo AI... (BAMBOO...)

使用 139B 枚 Bamboo 代币兑换回 2772 枚 WBNB，完成攻击。

From 0x7F7D46...564FF5A2 To PancakeSwap V2: BAMBOO 41 For 139,762,309,088.674044298 Bamboo AI... (BAMBOO...)

From PancakeSwap V2: BAMBOO 41 To 0x7F7D46...564FF5A2 For 2,772.093418146348543682 $592,245.44 Wrapped BNB... (WBNB...)

再以另一个意外销毁 pair 代币的例子说明一下，如下图代码。合约会判断当前交易类型，如果为 2（to 地址为 pair 合约，相当于卖币），合约会记录一个交易量的 20% 作为销毁数量，后续可以调用 goDead() 函数将 pair 中这个累积数量销毁，该功能对稳定币价能起部分作用。

但是合约没考虑到一些特殊情况，例如直接向 pair 里面转币，这时会使得合约误以为用户是在卖币，这部分币可以通过 pair 的 skim 函数全部提取出来。相当于 pair 没有任何变化，再次调用 goDead() 函数时，却能使得 pair 中的代币意外减少，重复操作，便能耗尽 pair 其中一种代币。

```
514     function _transfer(
515         address _from,
516         address _to,
517         uint256 _amount
518     ) internal {
519         uint256 amountx = _amount;
520         if(_amount == 0) {...}
524         require(!bkks[_from] && !bkks[_to], "had bkk");
525
526         if (inSwapAndLiquify) {...}
530
531         uint256 _transferType = _getTransferType(_from, _to);
532         if (_transferType == 0){...}
536
537         if (_transferType == 3){...}
542
543         if ( _transferType == 4) {...}
550
551         bool isSwap = shouldSwapToDiv(_transferType);
552         if (isSwap) {...}
555
556         if (frees[_from] || frees[_to] ) {...}
561
562         if (_transferType == 1){...}
567
568
569         if ( 1 == 1 ) {...}
597
598         if (_transferType == 1){...}
603
604         setShareAndProcess(_from,_to,isSwap);
605
606         if (_transferType == 2){
607             if (isOpenToDead){
608                 amountToDead += amountx * 20 /100;
609             }
610
611         }
612     }
```

```
614     function goDead() public{
615         if (amountToDead > 0){
616             _rawTransfer(address(pair), address(0xdEaD), amountToDead);
617             pair.sync();
618             amountToDead = 0;
619         }
```

以下同样是真实安全事件过程：

不停通过 transfer 与 skim 操作，将 amountToDead 值控制得异常大。

From 0x45aA25...Dd4779C0 To PancakeSwap V2: BSC-USD-APEDAO 3 For 422.460775987218251304 APEDAO... (APEDAO...)

From PancakeSwap V2: BSC-USD-APEDAO 3 To 0x45aA25...Dd4779C0 For 0 $0.00 Binance-Peg ... (BSC-US...) transfer

From PancakeSwap V2: BSC-USD-APEDAO 3 To 0xB47955...6C460092 For 8.449215519744365026 APEDAO... (APEDAO...)

From PancakeSwap V2: BSC-USD-APEDAO 3 To 0x91418d...B4fc7299 For 8.449215519744365026 APEDAO... (APEDAO...)

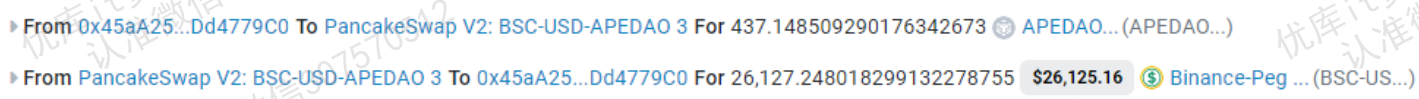
From PancakeSwap V2: BSC-USD-APEDAO 3 To 0xB47955...6C460092 For 4.224607759872182513 APEDAO... (APEDAO...) skim

From PancakeSwap V2: BSC-USD-APEDAO 3 To 0x45aA25...Dd4779C0 For 401.337737187857338739 APEDAO... (APEDAO...)

调用 goDead() 函数将 pair 中的代币销毁掉。

From PancakeSwap V2: BSC-USD-APEDAO 3 To Null: 0x000...dEaD For 3,708.941327967849359474 APEDAO... (APEDAO...)

使用少量代币兑换了大量的 USDT。



总结：

代币的 transfer 函数一定要认真设计代币转移逻辑，一定要避免 pair 余额意外扣除的情况，pair 余额仅在添加流动性、移除流动性以及交易过程中才能改变，并且改变数量尽量不要大于或小于传入数量，才能保障 pair 资金健康。

二、价格操控问题

对于价格操控的攻击事件，主要是由于项目方合约在获取价格的时候没有考虑到价格被意外控制的情况。特别是对于存在两个逆向计算的业务逻辑函数，如 stake&unstake、deposit&withdraw 等，两种操作一般都使用相同的参数但相反的计算逻辑。

而在这两种操作之间，如果能通过其他函数或合约将其中的部分或全部参数进行不对等的控制，那么两种操作之间便可能存在价格差，从而导致资产被盗。

1、只读重入

重入一直是区块链生态中最常见也是危害最大的漏洞之一。在以太坊最开始盛行的时候，重入攻击大多是因为以太坊转账调用的目标函数 fallback() 中再次发起了对该项目合约的调用，此时项目合约变量还未更新，可以绕过相关检查，如 DAO 事件。

随着以太坊生态越来越丰富，各个项目方更加注重这一方面的防护以及 solidity0.8.0 版本之后对溢出的主动检查，现在更多的攻击事件是通过重入来控制某些价格或参数，来达到攻击的目的。

例如近期发生频繁的只读重入，便是通过重入前后变量未修改完全，使得重入中调用的计算价格的函数出现异常，从而导致资产被盗。

有兴趣的读者可以参考如下链接：安全审计必备知识 | 难以防范的 “只读重入攻击” 是什么？

2、Pair 余额依赖

Pair 可以最直观的表现出代币价格变化，也是最准确的价格获取渠道，但 pair 中的价格是瞬时价格，并且可以通过大量代币兑换来控制，导致瞬间暴涨或暴跌。如果依赖 pair 中的价格作为业务中的参数参与计算，那么便可能导致异常的结果。

如下代码，burnForEth 函数通过依赖 pair 中能兑换出的数量来作为本合约发送给调用者的 ETH 数量，该数量是通过 pair 中两种代币余额来进行计算。正常情况下这是一个最准确的价格，但是如果提前使用 ETH 将 pair 中的另一个代币大量兑换出来，会导致 ETH 在该 pair 中的价格暴跌，从而计算出能兑换的 ETH 就会异常大，使得合约给调用者发送更多的 ETH。最后，调用者通过 pair 将 ETH 兑换回来，使价格恢复正常。

```
184 ▾   function burnForEth(uint256 amount) public returns (bool) {
185         require(burnEnabled, "Burn not enabled");
186         require(balanceOf(_msgSender()) >= amount, "not enough funds to burn");
187
188         address[] memory path = new address[](2);
189         path[0] = address(this);
190         path[1] = router.WETH();
191
192         uint256[] memory a = router.getAmountsOut(amount, path);
193
194         uint256 cap;
195 ▸       if (address(this).balance <= 1 ether) {⟷
197 ▸       } else {⟷}
200
201         require(a[a.length - 1] <= cap, "amount greater than cap");
202         require(
203             address(this).balance >= a[a.length - 1],
204             "not enough funds in contract"
205         );
206
207         transferToAddressETH(payable(msg.sender), a[a.length - 1]);
208         super._burn(_msgSender(), amount);
209
210         totalBurnRewards += a[a.length - 1];
211         totalBurned += amount;
212
213         emit BurnedTokensForEth(_msgSender(), amount, a[a.length - 1]);
214         return true;
215     }
```

以下是真实安全事件过程：

攻击者使用闪电贷的 ETH 将大量 WAX 代币兑换出来，导致 WAX 价格暴涨。

- From 0x6bd56a...13CFa3b9 To Uniswap V2: WAX 3 For 10 ($15,778.80) Wrapped Ethe... (WETH...)
- From Uniswap V2: WAX 3 To 0xFf0679...f85E4283 For 3,162,408.507319883383899272 Candle... (WAX...)
- From Uniswap V2: WAX 3 To 0x6bd56a...13CFa3b9 For 60,085,761.639077784294086178 Candle... (WAX...)

攻击者不断调用 burnForEth 函数，销毁 WAX 来取出 ETH，此时使用的 WAX 价格时暴涨过后的，可以获得更多的 ETH。

- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 551,716.232991616559409672 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 494,610.224031331955013634 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 443,593.906278244479897987 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 397,983.063138148242664918 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 357,177.100556986906608039 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 320,647.540616612547148353 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 287,928.336461743179217038 Candle... (WAX...)

  - Transfer 0.459817587021942203 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9
  - Transfer 0.413835828319747983 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9
  - Transfer 0.372452245487773185 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9
  - Transfer 0.335207020938995866 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9
  - Transfer 0.30168631884509628 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9
  - Transfer 0.271517686960586652 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9
  - Transfer 0.244365918264527986 ETH From 0xFf0679...f85E4283 To 0x6bd56a...13CFa3b9

最后，攻击者将 ETH 兑换出来，归还闪电贷并获利。

- From 0x6bd56a...13CFa3b9 To 0xFf0679...f85E4283 For 2,733,195.632382054003325713 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Uniswap V2: WAX 3 For 51,930,717.015259026063188551 Candle... (WAX...)
- From 0x6bd56a...13CFa3b9 To Null: 0x000...000 For 60,085,761.639077784294086178 Wax_Dividend... (Wax_Di...)
- From Uniswap V2: WAX 3 To 0x6bd56a...13CFa3b9 For 9.443127922748690779 ($14,900.12) Wrapped Ethe... (WETH...)
- From 0x6bd56a...13CFa3b9 To Balancer: Vault For 10 ($15,778.80) Wrapped Ethe... (WETH...)
- From 0x6bd56a...13CFa3b9 To 0xdAb930...F03FF84f For 4.041303792968112817 ($6,376.69) Wrapped Ethe... (WETH...)

总结：

安全事件一直都是区块链生态最大的威胁之一，需要项目方时刻保持警惕。对于需要计算价格的业务逻辑要有严格的流程把控，尽量避免直接使用 pair 余额进行价格计算，调用外部函数获取价格也需要考虑其是否能被操控。项目方也可以与专业的安全合作伙伴合作，及时发现并应对潜在的安全威胁。