幻象函数和数十亿美元风险的无操作

（Phantom Functions and the Billion-Dollar No-op）

由 Dedaub 团队提供

1 月 10 日，我们披露了 Multichain 项目（原“AnySwap”）的重大漏洞。Multichain 已发布公告，重点关注对其客户的影响及缓解措施。声明发布之后，网络攻击和 flashbots 大战接踵而至。目前损失的资金总额大约为最初直接暴露资金的 0.5%。

[建议：如果您使用过 Multichain/Anyswap，请检查/撤销您对易受攻击代币的许可。如果有任何不清楚的地方，请务必检查所有链接并阅读完整的说明。]

我们将用单独的年表记录攻击和防御，等待威胁完全缓解之后发布。这篇短文旨在说明漏洞的技术要素，即攻击向量的情况。

据我们所知，攻击向量是新的。Solidity/EVM 开发者和安全社区应该意识到这种威胁。

在 Multichain 合约的个案中，攻击向量指向两个独立的主要漏洞，一个主要在 WETH（“包装的以太币”）流动资金库合约（AnyswapV5ERC20 的例子）中，另一个在向其他链发送通证的路由器合约（AnyswapV4Router）中。威胁是巨大的和多方面的——对单个合约来说几乎是“至大的威胁”：

- 仅以太坊（Ethereum）上，仅从 3 个受害者的 WETH 账户上就将有 4.31 亿美元通过一次直接交易被盗。我们在披露之前在本地分支上对此进行了演示。（余额和估值计算截至 1 月 12 日本说明撰写时为止。主要的潜在受害者账户 AnySwap Fantom Bridge 本身持有超过 3.67 亿美元。该合约在本文发表之时持有 12 亿美元。）

- 不同通证和多个区块链已部署了相同的合约，包括 Polygon、BSC、Avalanche 和 Fantom。（基于其他包装的原生代币的流动性合约，如 WBNB、WAVAX、WMATIC 等也容易受到攻击。）这些其他网络的潜在风险后来估值约为 4000 万美元。

- 主要的潜在受害者账户 AnySwap Fantom Bridge 托管已转移到 Fantom 区块链的通证。这意味着攻击者可以将任意金额转移至 Fantom，然后在以太坊将其连同当前 3.67 亿美元的跨链桥（以及来自其他受害者的数千万美元）一起偷回。被移动的通证在 Fantom 或是其他被移动到的地方仍将是有效的（并且很有价值）。这使得攻击的潜在影响在理论上决无止境（“无限的”）：除了从以太坊受害者那里盗走的 4.31 亿美元以及在其他链上窃取的若干金额之外，任何“投入”金额皆可以翻倍。

- 近 5000 个不同的账户已经给予（以太坊）易受攻击的合约无限的 WETH 授权。这一数字此后已大幅下降（尤其是在持有资产的账户中），但是仍然存在着威胁：直到批准被撤销之前，这些账户曾经获得的任何 WETH 都容易受到攻击。

鉴于上述情况，潜在的实际影响（如果漏洞被充分利用的话）可以说在数十亿美元的范

围内。考虑到理论上威胁无穷无尽，这将是有史以来最大的黑客攻击之一，我们不再进行更详细的评估。

攻击向量

简单来说：
调用者不应依赖 permit 撤销任意通证。

调用 token.permit(…)不会复归如下通证

•不执行 permit 的
•具有（不归复）回退函数的

最值得注意的是，WETH（以太币的 ERC-20 形式）就是这样一种通证。

我们将这种模式称为幻象函数（Phantom Functions）——例如，我们说“WETH 有一个幻象 permit”或“permit 是 WETH 合约的一个幻象函数”。带有幻象函数的合约并不真正地定义函数，而是接受任何对它的调用而不进行还原。在以太坊上，其他具有幻象许可的高价值通证是 BNB 和 HEX。其他链上的原生等效代币（如 WBNB、WAVAX）也可能表现幻象许可。

更详细地说：

用 Solidity 写的智能合约可以包含一个回退函数（fallback function）。这是在合约上调用任意函数 f ( )而合约并未定义 f ( )时需要调用的代码。

在当前的 Solidity 中，回退函数是相当奇特的功能。不过在更早版本的 Solidity 中，包含回退函数很常见，因为回退函数也是合约接收以太币时调用的代码。（在较新的 Solidity 版本中，使用明确的 receive function 作为代替。）事实上，回退函数以前是没有名称的，只是 function()。例如，WETH 合约包含如下定义的回退功能：

```
1  function() public payable {
2        deposit();
3  }
4  function deposit() public payable {
5        balanceOf[msg.sender] += msg.value;
6        Deposit(msg.sender, msg.value);
7  }
```

WETH9-fallback.sol hosted with ❤ by GitHub view raw

这个函数在收到以太币时被调用（只是将其存储，用来换成包装的以太币），但至关重要的是，在 WETH 合约调用未定义的函数时也会调用这个函数。

问题是，如果要依赖未定义的函数来执行重要的安全检查，该怎么办呢？

就 AnySwap/MultiChain 的代码而言，最简单的易受攻击的合约包含诸如以下代码：

```
function deposit() external returns (uint) {
    uint _amount = IERC20(underlying).balanceOf(msg.sender);
    IERC20(underlying).safeTransferFrom(msg.sender, address(this), _amount);
    return _deposit(_amount, msg.sender);
}
...
function depositWithPermit(address target, uint256 value, uint256 deadline, uint8 v, bytes32 
    IERC20(underlying).permit(target, address(this), value, deadline, v, r, s);
    IERC20(underlying).safeTransferFrom(target, address(this), value);
    return _deposit(value, to);
}
```

AnyswapV5ERC20WETH.sol hosted with ❤ by GitHub view raw

这意味着常规存款路径（函数 deposit）从外部调用者（msg.sender）将资金转移到该合约，它需要被批准为支出者。这个存款行为一向是安全的，但是会使客户产生一种虚假的安全感：他们批准合约以转移其资金，因为他们确信只有在他们发起调用，也就是他们是 msg.sender 时转款才会发生。

然而，第二种存入资金的途径，函数 depositWithPermit 只要 permit 调用成功，就允许存入属于其他人（target）的资金。

对于支持它的 ERC-20 通证，permit 是标准的批准（approve）调用的替代方案：它允许使用一个链外的安全签名来注册许可。许可人通过签署许可请求的方式批准受益人花钱。permit 方式有若干优点：不需要单独的交易（花费 gas）来批准支出者，许可有截止日期以及转账可以批量进行等等。

如前所述，本例中的问题是 WETH 通证有一个幻象许可，因此在调用它时是一个无故障的不操作。不过，这应该没问题，对吧？不操作会有什么损失？既然 permit 没有执行，也不应该存在对使用目标资金的批准/许可。

然而不幸的是，合约已经获得了所有使用过第一个存款路径（函数 deposit）的客户的批准！

所有这类客户的 WETH 全都可能被盗，只需调用 depositWithPermit，之后 withdraw 即可。（为避免抢先交易，攻击者可能会将这两个拆分为不同的交易，这样收益就不会立即显现。）

笔记：

两个单独的漏洞是基于上述的攻击向量。我们已在上文概述了第一个漏洞。第二个 AnySwap 路由器合约上的漏洞更难以利用，因为需要模仿特定类型的通证。我们之所以没有

详细说明，是因为这篇速成短文的目的是向社区通报攻击向量，而非说明攻击的细节。

我们已经详尽地搜索了具有类似易受攻击代码和风险敞口的其他服务。这包括除 WETH 以外对有幻象许可的通证进行批准的容易受到攻击的合约。尽管我们发现了其他易受攻击的代码模式，但是这些合约目前在以太坊上的批准率非常之低，或近乎为零。（正是运用了 contract-library.com 分析系统我们才能快速完成这一类型的研究。）我们对其他链的搜索没有那么彻底，因为我们没有现成的所有已部署合约的索引存储库。然而，我们最好的指标显示，除了 AnySwap/Multichain 合约之外并没有太大的风险敞口。

结束语

我们已披露了两个漏洞，每一个都获得了 Multichain 悬赏的最高 100 万美元的漏洞赏金。（感谢您能慷慨地承认这一巨大的威胁！）

我们首先对模型提出怀疑，然后在实际部署的合约中加以搜寻并由此发现了这一攻击。虽然事后看来，攻击向量是简单直接的，但是当我们刚开始考虑之时，它并不是那么简单。事实上，我们最初在周日凌晨 2:30 交流想法的时候，是这样说的：

“关于漏洞我有一个疯狂的想法。您想完整检查一下基本情况吗？”

鉴于这会导致历史上最大的黑客攻击之一，确实，这很疯狂。