SharkTeam | In Math，We Trust ！

闪电贷攻防与智能合约审计

分享人：SE.Adam
日期：2021-07-07

团队介绍

SharkTeam

SharkTeam专注于智能合约安全，由拥有多年一线网络安全实战经验的团队成员组成，精通区块链和智能合约底层原理，具备完善的区块链漏洞挖掘和智能合约审计能力，可提供全面的威胁建模、合约审计、应急响应服务，已帮助多个知名区块链项目发现并修复安全漏洞，致力于保护用户数字资产安全与隐私安全。

目录

SharkTeam

01 · 什么是闪电贷？

02 · 闪电贷攻击事件分析

03 · 闪电贷攻击防范建议

04 · 智能合约审计

什么是闪电贷？

SharkTeam

概念：闪电贷，顾名思义就是在极短的时间内完成贷款与还款的操作。在DeFi领域，闪电贷为金融业务提供了免抵押借款服务，是DeFi世界的一款利器。闪电贷的概念最早是由Marble协议提出来的，并由Aave、dYdX、Uniswap等协议进行了普及，第一笔闪电贷操作来自于Aave协议。

原理：闪电贷是通过智能合约的原子性，来完成零风险贷款的业务，因此闪电贷业务只能由智能合约来实现。简单来说，就是在一个区块交易中，同时完成借款和还款操作这两个操作，无需抵押任何资产，只需支付手续费即可。用户在借到款之后，可以利用借到的资产进行其他操作，比如套利、偿还抵押借款、自清算等操作。然后在交易结束的时候，用户只要把借到的款项及手续费及时归还就可以，否则该笔交易就会回滚，就像什么也没有发生过一样。

原子性：原子性是指具有原子性的事务。以太坊的每比交易都是具有原子性的，原子事务是不可分割的，在事务执行完毕之前，不会被任 何其他操作所中断。而该事务中的所有操作，要么全部被成功执行，要么全部执行失败，不会出现一部分成功而另一部分失败的情况。

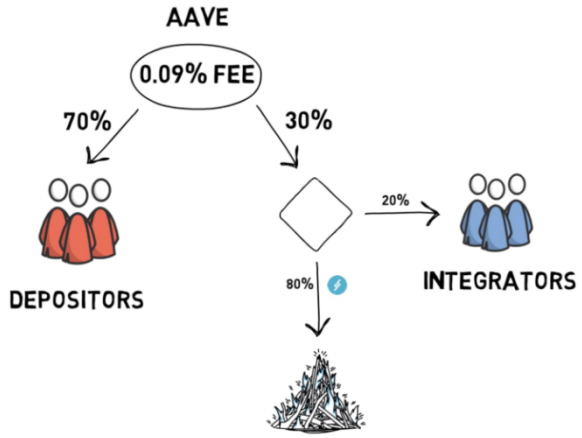
- 向dYdX进行借款、还款;
- 在Compound上提供ETH并借出DAI;
- 在Curve上将借出的DAI的一半换成USDC;
- 在Uniswap上为DAI/USDC池提供流动性等等。

所有这些操作都可以同时在一个交易中完成，而其中只要有一个操作失败，则整笔交易都会失败回滚，就像什么都没有发生一样。单笔交易只受Gas费用限制，以及以太坊区块容量的限制，所以理论上可以有成千上万个操作步骤。

闪电贷与闪电兑

SharkTeam

Flash Loan（以Aave为例）	Flash Swap
0.09% fee	Swap fee
Minimum 3 operations	More flexibility
Payback same asset	Payback pair asset
Optimistic transfer	Optimistic transfer



闪电贷（Flash Loan）必须以你借入的同一资产偿还：如果借入Dai，就需要偿还Dai。生息协议Aave对闪电贷收取0.09%的费用。它至少需要进行三个操作：

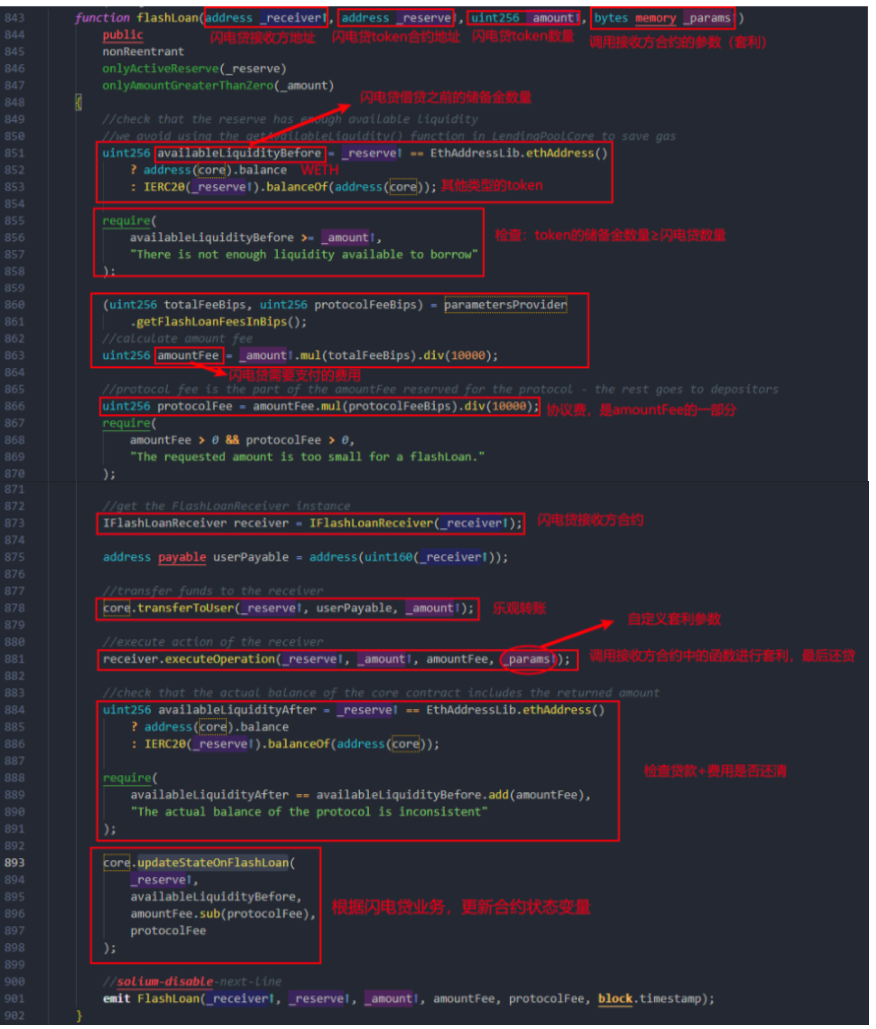
1. 在Aave上借款
2. 使用从Aave通过闪电贷借来的资产（套利/更换抵押品/自清算）
3. 向Aave支付手续费，并还款

闪电兑(Flash Swap) 就像在Uniswap上所做的那样，闪电交换没有固定费用，而是收取交换费。与闪电贷相比，闪电兑换可以看作是“免费”贷款，因为兑换手续费是从交易订单中扣除的，我们不必单独付款。另一点区别：我们可以偿还兑换的任何资产，如果我们使用闪电兑换拿ETH买入Dai，那么用Dai或ETH偿还都可以，这使我们可以执行更复杂的操作。

闪电兑换和闪电贷均采用乐观转帐（Optimistic Tranfer），乐观转账使用户可以进行无抵押贷款或兑换交易，只要用户在交易结束时还清借款，就可以执行交易。乐观转账是闪电兑换和闪电贷的核心。

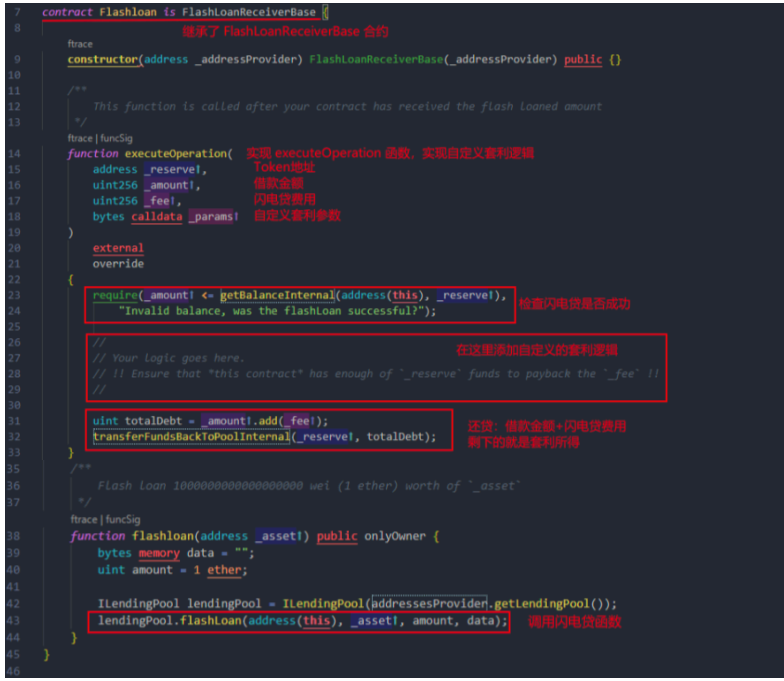
闪电贷实现与使用（以AAVE为例）

SharkTeam



以Aave项目为例，Aave项目实现了闪电贷业务的主要合约是LendingPool.sol 合约，关键函数为 flashLoan函数。

闪电贷接收方合约 IFlashLoanReceiver (接口合约)的实现是需要闪电贷发起方自定义的。自定义闪电贷合约需要继承 IFlashLoanReceiver 接口合约或者 FlashLoanReceiverBase 合约。



自定义闪电贷合约需要实现 IFlashLoanReceiver 接口中的 executeOperation 函数。自定义闪电贷合约部署在区块链中的地址即为 flashloan 函数的参数 _receiver。

https://github.com/aave/flashloan-box/blob/master/contracts/Flashloan.sol

闪电兑实现与使用（以Uniswap为例）

SharkTeam

```
ftrace | funcSig
function swap(uint amount0Out!, uint amount1Out!, address to!, bytes calldata data!) external lock {
    require(amount0Out! > 0 || amount1Out! > 0, 'UniswapV2: INSUFFICIENT_OUTPUT_AMOUNT');
    (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
    require(amount0Out! < _reserve0 && amount1Out! < _reserve1, 'UniswapV2: INSUFFICIENT_LIQUIDITY');

    uint balance0;
    uint balance1;
    { // scope for _token{0,1}, avoids stack too deep errors
    address _token0 = token0;
    address _token1 = token1;
    require(to! != _token0 && to! != _token1, 'UniswapV2: INVALID_TO');
    if (amount0Out! > 0) _safeTransfer(_token0, to!, amount0Out!); // optimistically transfer tokens
    if (amount1Out! > 0) _safeTransfer(_token1, to!, amount1Out!); // optimistically transfer tokens
    if (data!.length > 0) IUniswapV2Callee(to!).uniswapV2Call(msg.sender, amount0Out!, amount1Out!, data!);
    balance0 = IERC20(_token0).balanceOf(address(this));
    balance1 = IERC20(_token1).balanceOf(address(this));
    }
    uint amount0In = balance0 > _reserve0 - amount0Out! ? balance0 - (_reserve0 - amount0Out!) : 0;
    uint amount1In = balance1 > _reserve1 - amount1Out! ? balance1 - (_reserve1 - amount1Out!) : 0;
    require(amount0In > 0 || amount1In > 0, 'UniswapV2: INSUFFICIENT_INPUT_AMOUNT');
    { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
    uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
    uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
    require(balance0Adjusted.mul(balance1Adjusted) >= uint(_reserve0).mul(_reserve1).mul(1000**2), 'UniswapV2: K');
    }

    _update(balance0, balance1, _reserve0, _reserve1);
    emit Swap(msg.sender, amount0In, amount1In, amount0Out!, amount1Out!, to!);
}
```

调用接收方合约的参数
闪电兑换接收方地址
闪电兑换token1数量
闪电兑换token0数量
接收方地址不能是token0和token1地址
乐观转账
套利后当前合约的token0和token1的余额
调用接收方合约实现套利最后返还借款+兑换费用
校验还款金额≥借款+兑换费用
更新储备金

- UniswapV2项目实现闪电兑换业务的主要是UniswapV2Pair合约的swap函数。
- 闪电贷接收方合约 IUniswapV2Callee(接口合约)是需要闪电贷发起方自定义的。自定义闪电贷合约需要继承该接口合约，并实现其中的uniswapV2Call函数。
- 自定义闪电贷合约部署在区块链中的地址即为 swap 函数的参数 to。

```
contract ExampleFlashSwap is IUniswapV2Callee {
    IUniswapV1Factory immutable factoryV1;
    address immutable factory;
    IWETH immutable WETH;

    constructor(address _factory, address _factoryV1, address router) public {
        factoryV1 = IUniswapV1Factory(_factoryV1);
        factory = _factory;
        WETH = IWETH(IUniswapV2Router01(router).WETH());
    }

    // needs to accept ETH from any V1 exchange and WETH. ideally this could be enforced, as in the router,
    // but it's not possible because it requires a call to the v1 factory, which takes too much gas
    receive() external payable {}

    // gets tokens/WETH via a V2 flash swap, swaps for the ETH/tokens on V1, repays V2, and keeps the rest!
    function uniswapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external override {
        address[] memory path = new address[](2);
        uint amountToken;
        uint amountETH;
        { // scope for token{0,1}, avoids stack too deep errors
        address token0 = IUniswapV2Pair(msg.sender).token0();
        address token1 = IUniswapV2Pair(msg.sender).token1();
        assert(msg.sender == UniswapV2Library.pairFor(factory, token0, token1)); // ensure that msg.sender is actual
        assert(amount0 == 0 || amount1 == 0); // this strategy is unidirectional
        path[0] = amount0 == 0 ? token0 : token1;
        path[1] = amount0 == 0 ? token1 : token0;
        amountToken = token0 == address(WETH) ? amount1 : amount0;
        amountETH = token0 == address(WETH) ? amount0 : amount1;
        }

        assert(path[0] == address(WETH) || path[1] == address(WETH)); // this strategy only works with a V2 WETH pair
        IERC20 token = IERC20(path[0] == address(WETH) ? path[1] : path[0]);
        IUniswapV1Exchange exchangeV1 = IUniswapV1Exchange(factoryV1.getExchange(address(token))); // get V1 exchange

        if (amountToken > 0) {
            (uint minETH) = abi.decode(data, (uint)); // slippage parameter for V1, passed in by caller
            token.approve(address(exchangeV1), amountToken);
            uint amountReceived = exchangeV1.tokenToEthSwapInput(amountToken, minETH, uint(-1));
            uint amountRequired = UniswapV2Library.getAmountsIn(factory, amountToken, path)[0];
            assert(amountReceived > amountRequired); // fail if we didn't get enough ETH back to repay our flash loan
            WETH.deposit{value: amountRequired}();
            assert(WETH.transfer(msg.sender, amountRequired)); // return WETH to V2 pair
            (bool success,) = sender.call{value: amountReceived - amountRequired}(new bytes(0)); // keep the rest!
            assert(success);
        } else {
            (uint minTokens) = abi.decode(data, (uint)); // slippage parameter for V1, passed in by caller
            WETH.withdraw(amountETH);
            uint amountReceived = exchangeV1.ethToTokenSwapInput{value: amountETH}(minTokens, uint(-1));
            uint amountRequired = UniswapV2Library.getAmountsIn(factory, amountETH, path)[0];
            assert(amountReceived > amountRequired); // fail if we didn't get enough tokens back to repay our flash
            assert(token.transfer(msg.sender, amountRequired)); // return tokens to V2 pair
            assert(token.transfer(sender, amountReceived - amountRequired)); // keep the rest! (tokens)
        }
    }
}
```

继承 IUniswapV2Callee 接口
UniswapV2路由合约地址
UniswapV1 工厂合约地址
UniswapV2 工厂合约地址
WETH合约地址
合约可以接收转账
实现了 uniswapV2Call 函数
交易对：WETH/Token
调用地址为token0/token1交易对合约地址
Token金额
WETH金额
获得V1的Exchange
V2 上的 Token 兑换 V1 上的 ETH
V1兑换得到的ETH数量
还贷
V2闪电贷需要的ETH数量
剩余的ETH就是套利所得
V2 上的ETH 兑换 V1 上的 Token
V1兑换得到的Token数量
还贷
V2闪电贷需要的Token数量
剩余的Token就是套利所得

闪电贷应用

SharkTeam

ARBITRAGE

套利，即闪电贷可以放大成功套利的利润。例如，Uniswap 和 Curve 之间的 DAI/USDC 池存在价格差异。你可以在 Curve 上用 1 个 DAI 换取 1 个 USDC，但在 Uniswap 上您只需要 0.99 个 DAI 就可以买到 1 个 USDC。现在你可以尝试执行以下套利:

1. 通过闪电贷向 Aave 借 100,000 DAI。
2. 在 Uniswap 上将 100,000 DAI 换成 USDC，收到 101,010 USDC。
3. 在 Curve 上将 101,010 USDC 换成 101,010 DAI
4. 偿还初始 100,000 DAI + 0.09% Fees = 100,090 DAI
5. 利润 101,010 DAI - 100,090 DAI = 920 DAI

SWAPPING COLLATERAL

假设你用 ETH 作为抵押品从 Compound 借了 DAI 。你可以通过以下方式将抵押品从 ETH 换成例如BAT :

1. 发起一笔闪电贷借出 DAI
2. 用借来的 DAI 偿还 Compound 中的贷款
3. 提取你的 ETH
4. 在 Uniswap 上将你的 ETH 换成 BAT
5. 提供 BAT 作为 Compound 的抵押品
6. 以你的 BAT 抵押品借出 DAI
7. 用借来的 DAI + Fees 偿还步骤1的闪电贷通过这样一轮操作，额外付出借款金额的 0.09% 就能成功把你的抵押品从 ETH 换成了 BAT

SELF-LIQUIDATION

假设你在 Compound 上有一笔以 ETH 作为抵押借出 DAI 的贷款。ETH 价格不断下跌，已经接近清算水平。你也没有或不想存入更多的 ETH 来降低你的清算水平，你也没有偿还贷款所需的 DAI。现在，与其让 MakerDAO 合约清算你的抵押品并向你收取清算费，不如采取以下步骤:

1. 发起闪电贷，借出一笔 DAI，金额是你在 Compound 上的欠债
2. 偿还 Compound 上的 DAI 贷款，提取 ETH
3. 将足够的 ETH 换成 DAI，以偿还步骤1的 闪电贷+费用
4. 拿住你剩下的 ETH　经过以上步骤，就可以实现自我清算，避免了合约清算以及清算费用。

闪电贷攻击事件

SharkTeam

编号	日期	主攻协议	闪电贷协议	攻击类型
1	2020/2/15	bZx	dYdX	哄抬套利
2	2020/2/18	bZx	bZx	操纵预言机
3	2020/6/28	Balancer	dYdX	哄抬套利
4	2020/10/26	Harvest	Uniswap V2	操纵预言机
5	2020/11/6	Cheese Bank	dYdX	操纵预言机
6	2020/11/12	Akropolis	dYdX	重入攻击
7	2020/11/14	Value.DeFi	Aave/Uniswap V2	操纵预言机
8	2020/11/17	OUSD	dYdX	重入攻击
9	2020/12/18	Warp Finance	Uniswap V2/dYdX	操纵预言机
10	2021/2/4	Yearn.Finance	dYdX/Aave	哄抬套利
11	2021/2/13	Alpha.Finance	Aave	技术漏洞
12	2021/5/2	Spartan	PancakeSwap	哄抬套利
13	2021/5/8	Value.DeFi	Value.DeFi	技术漏洞
14	2021/5/8	Rari Capital	dYdX	重入攻击
15	2021/5/13	xToken	dYdX	操纵预言机
16	2021/5/16	bEarn Fi	Cream.Finance	技术漏洞
17	2021/5/20	PancakeBunny	PancakeSwap	操纵预言机
18	2021/5/22	Bogged Finance	PancakeSwap	技术漏洞
19	2021/5/25	AutoShark	PancakeSwap	技术漏洞+操纵预言机
20	2021/5/28	BurgerSwap	PancakeSwap	重入攻击
21	2021/5/28	JulSwap	JulSwap	操纵预言机
22	2021/5/30	Belt Finance	PancakeSwap	操纵预言机
23	2021/6/5	BurgerSwap	PancakeSwap	重入攻击
24	2021/6/10	EvoDeFi	PancakeSwap	技术漏洞
25	2021/6/19	Impossible Finance	PancakeSwap	技术漏洞
26	2021/6/23	Eleven Finance	ApeSwap	技术漏洞
27	2021/6/25	xWin Finance	PancakeSwap	操纵预言机
28	2021/7/2	XDXSwap	XDXSwap	技术漏洞

- DeFi项目的业务逻辑设计复杂，涉及的经济学计算和参数较多，却不同项目和协议之间可组合性极其丰富，很难预测，非常容易出现安全漏洞。2020年以来，利用闪电贷这种新型产品进行攻击的DeFi事件在过去的一年里层出不穷，已增加到27起。通常闪电贷攻击包括“哄抬套利”、“操纵预言机”、“重入攻击”和“技术漏洞”四种

- 「哄抬套利」Pump and Arbitrage，先设法借助他人的资本抬高自有资产的价格再高价卖出获利。DeFi 生态中此类经济攻击事件之所以能够成功，与Flash loans（短时支配大量数字资产）和AMM（价格滑点）紧密关联。

- 「操纵预言机」攻击可以认为是「哄抬套利」攻击的一种对称操作，其必要条件为：攻击目标依赖 AMM 提供的信息对其内部资产进行定价。

- 「重入攻击」恶意合约通过恶意的fallback函数对受害合约发起递归调用，从而破坏操作的原子性，绕过检查以重复获得转账收益。

- 「技术漏洞」开发人员和测试人员对数值计算、地址校验等实现层面的疏忽而造成的漏洞。

PancakeBunny

SharkTeam

1) Flash Loan From PancakeSwap and ForTub
Borrow 2323K BNB From PancakSwap and 2962K USDT from ForTub 2

2) Add Liquidity to PancakeSwap BNB-USDT Pool
7744 BNB + 2962K USDT ==> 144K BNB-USDT LP-Token 2

3) Swap BNB To USDT in BNB-USDT Pool BNB价格极低
2316K BNB ==> 3826K USDT 3

4) Call getReward function of VaultFlipToFlip contract 3

5) Remove Liquidity From BNB-USDT Pool
144K BNB-USDT LP-Token ==> 7744 BNB + 2962K USDT 3

6) Swap USDT To BNB in BNB-USDT Pool
2962K USDT ==> 2312K BNB

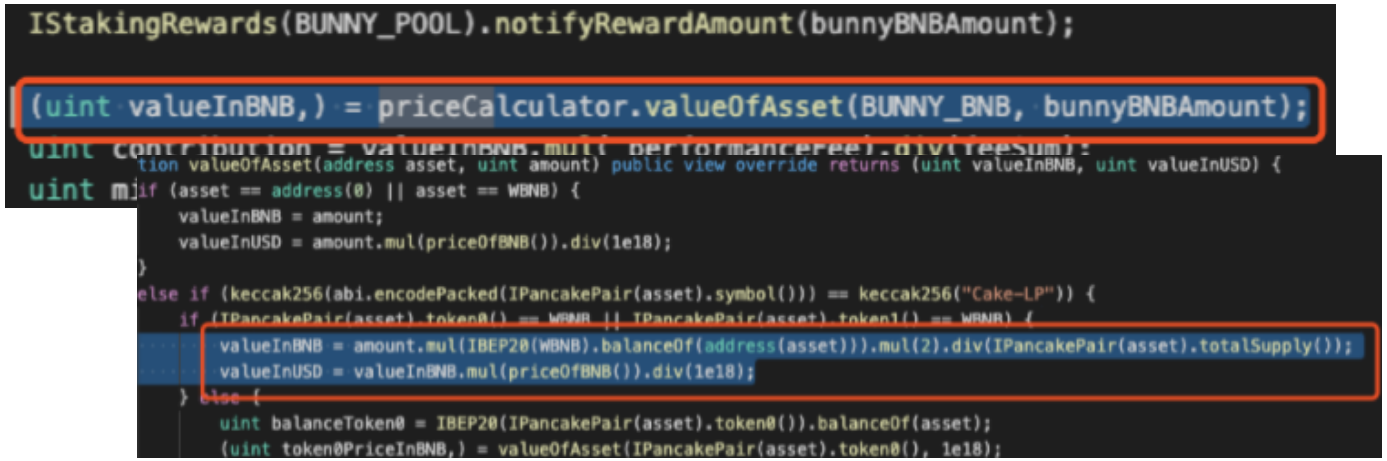
7) Swap BNB To BUNNY and then add liquidity to BNB-BUNNY Pool
2312/2K BNB ==> 177K BUNNY + 2312/2K BNB ==> 105K LP Token
7744/2 BNB ==> 36 BUNNY + 7744/2 BNB ==> 352 LP Token

8) Add All LP Tokens to PancakeSwap BunnyPool
106K BNB-BUNNY LP-Token ==> 6972K BUNNY

9) Exchange BUNNY to USDT and BNB to repay flash loan.
Transfer remianing tokens to attacker address.

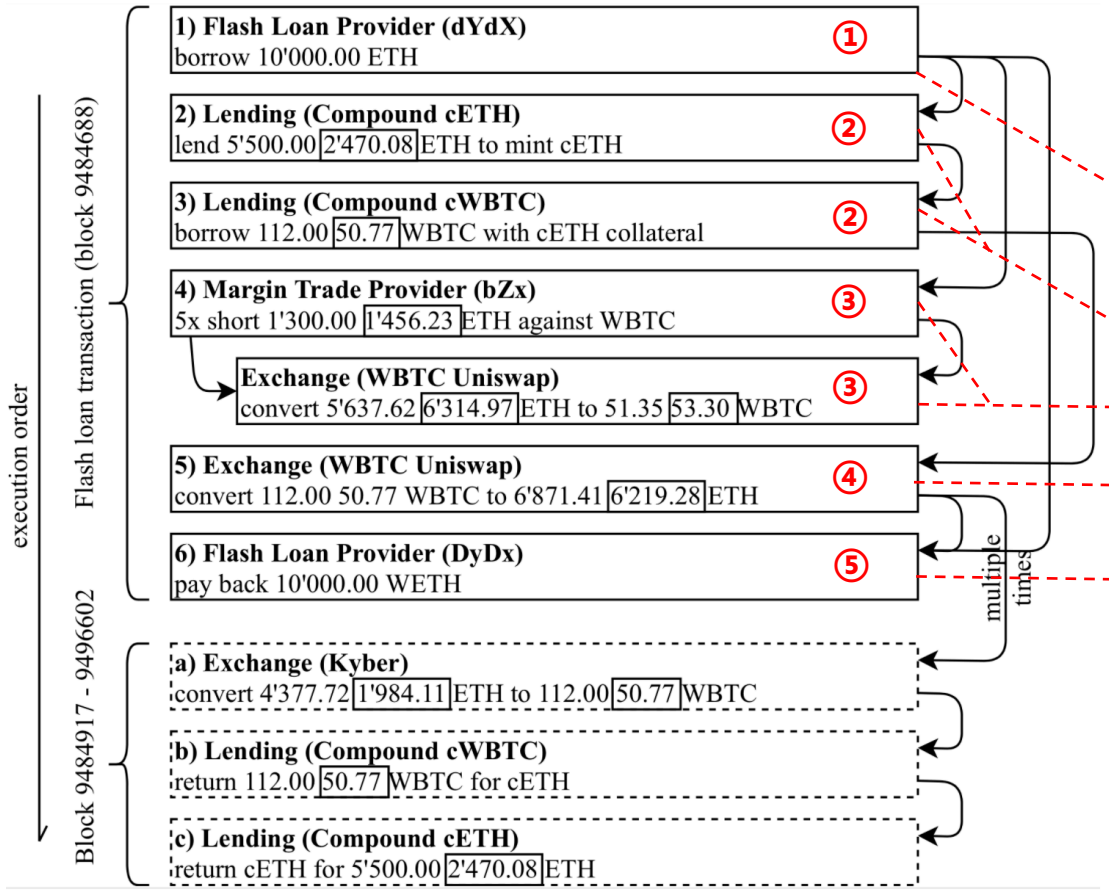
- 1. 攻击者首先进行了一次抵押，此时会产生抵押奖励；
- 2. 然后攻击者从闪电贷平台借出大量 BNB 和 USDT，通过 PancakeSwap 的交易对去添加流动性获取lp代币（144,445），并将lp代币留在了交易对合约中；
- 3. 因为攻击者进行过抵押，然后通过调用 VaultFlipToFlip 合约的 getReward 函数来获取 BUNNY 代币奖励并取回先前抵押的流动性，然而在奖励计算时，获取铸币数量的逻辑出现问题导致产生大量的 BUNNY 代币（6972455），并通过 PancakeSwap 兑换成BNB，造成了 BUNNY 价格暴跌；
- 4. 通过3步骤得到 BNB 之后归还到闪电贷地址，并从中获利 114631 个 BNB，价值大约 4000W 美元。

- 本次攻击事件，攻击者在一笔交易中完成了一系列闪电贷、兑换、获取奖励、归还闪电贷等操作，其主要原因还是**项目在计算抵押奖励时获取 lp 价格出现了逻辑问题**，导致黑客利用这一漏洞进行了攻击。



bZx

SharkTeam



	WBTC	ETH
Step 0	0	0
Step 1	0	10000
Step 2	112	4500
Step 3	112	3200
Step 4	0	10071
Step 5		71

其他需要注意点
⑥ 恒定乘积做市商与价格滑点：X*Y=K
⑦ 黑客窃取的数字资产是哪里来的？谁为此次攻击事件买单？

- 以0.5 ETH的GAS交易费，在区块9484688 完成闪电贷攻击，黑客共累计进行74比交易，获得1193.69 ETH 的利润。
- 此次攻击的核心思路是在去中心化交易平台bZx上进行杠杆交易，拉升去中心化交易所Uniswap上的WBTC/ETH的价格，创造套利机会。黑客首先从dYdX上利用闪电贷借到 10,000 枚ETH，其中一部分 ETH 以 5 倍杠杆在 bZx 做空 ETH，另一部分在 Compound上置换出 WBTC；在ETH价格降低之时，再用 WBTC 置换回 ETH，完成利用闪电贷的哄抬套利。
- 后续待价格稳定后，在 Kyber 上换回 WBTC，归还 Compound 中的ETH贷款。

安全防范建议

SharkTeam

Dragonfly：全面解读闪电贷，DeFi 安全模式全然改变
Flash Loans: Why Flash Attacks will be the New Normal

闪电贷攻击的根本原因在于被攻击的项目的智能合约存在漏洞，黑客利用了其中的漏洞完成价格操纵或技术攻击，而闪电贷只是提供了攻击的资金，降低了黑客攻击的成本。**因此，闪电贷攻击并没有直接根除的防范措施，只能尽可能的通过多轮的智能合约审计让项目的业务逻辑和合约代码不存在漏洞。**

解决闪电贷攻击的设想：1. 说服所有的闪电贷协议停止提供这种服务　但这几乎是不可能的。提供闪电贷款符合每个协议的利益，并且其用户有合理原因的希望使用此功能。2. 迫使关键交易跨越两个区块　如果一个资本密集型交易需要跨越至少两个区块，用户需要至少在两个区块时间段取出贷款，闪电攻击就成为不可能（如EOS中的 defer transaction）。3. 要求提供链上证明，证明完成闪电贷后用户的账户余额未出现变化　如果可以通过某种方法来检测用户的实际余额是多少（即在他们贷款之前和还款之后的余额分别是多少），我们就可以战胜闪电攻击。在原生 EVM 机制中无法执行此操作，但是可以对其进行修改。你要做的是：在用户与你的协议进行交互之前，你要求其提供 Merkle 证明，证明在上一个区块末尾时他们有足够的余额来偿还当前使用的资金。 你需要针对每个区块中的每个用户跟踪此情况。　这种方法一定程度上是奏效的。但验证这些链上证据在链上的成本极高 ，少有人会愿意提供这类证明，并为整个过程支付 gas 费用。另外用户完全可能有合理的原因，在上个区块想调整其余额。

防范闪电贷攻击的可行方法：1. 加权平均价格（WAP），像 Uniswap 或 OasisDEX 等市场价预言机，由于闪电攻击的可能性，你任何情况下不能把当前市价中位数当成喂价。 攻击者只需要一笔交易就能轻而易举地大幅改变市价中位数，让预言机失灵。对此最好的解决方案是通过时间加权平均价格（TWAP）或 成交量加权平均价格（VWAP）计算上一批 X 区块的加权平均数。2. 代币锁定期，通常如果治理代币进入某一贷款池，攻击者可以偷走大量选票，得到自己想要的结果。 我们应该要求所有治理行动要有代币锁定期 ，以阻止闪电贷攻击。更关键的是，所有治理代币应该有时间锁（timelocks），时间锁迫使所有的执行决策在生效前都有一段等候期。（例如 Compound 的时间锁是 2 天 ）如果遭遇意料之外的治理攻击，这一机制让系统有了容错时间 。3.合约在上线前测试时，应设置闪电贷攻击的 test case，尽量放大测试时的数值，并进行数学计算的核算。

智能合约审计

SharkTeam

1、闪电贷攻击并没有直接根除的防范措施，应尽可能的通过多轮审计让项目的业务逻辑和合约代码尽量不存在漏洞。

2、语法层面重点关注的审计项：重入、地址校验、权限控制等。

3、业务和计算层面重点关注的审计项：预言机、铸币/销毁、推荐奖励、清算等涉及到净值计算的数学运算逻辑，尤其重点检查计算中分母部分的安全性和可靠性。闪电贷实现逻辑、治理实现逻辑、分红实现逻辑等。

SharkTeam | In Math，We Trust ！

谢 谢 观 看