CD SECURITY

AUDIT REPORT

Blerb, a Hybrid DeFi protocol

May 2024

Introduction

A time-boxed security review of **Blerb, a Hybrid DeFi protocol** was done by **CD Security**, with a focus on the security aspects of the application's implementation.

Disclaimer

A smart contract security review can never verify the complete absence of vulnerabilities. This is a time, resource, and expertise-bound effort where we try to find as many vulnerabilities as possible. We can not guarantee 100% security after the review or even if the review will find any problems with your smart contracts. Subsequent security reviews, bug bounty programs, and on-chain monitoring are strongly recommended.

About **Blerb**

Hybrid DeFi is a new concept that combines an ERC721 NFT, `BLERB`, with an ERC20 Token `$BLERB`. The token and NFT after launch are interchangeable. One NFT can always be burnt for `90 000 $BLERB` and one `BLERB` NFT can be minted for the same `90 000 $BLERB` tokens. Pre-launch users will be able to buy the `BLERB` NFT using different referral codes.

Severity classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact - the technical, economic, and reputation damage of a successful attack

Likelihood - the chance that a particular vulnerability gets discovered and exploited

Severity - the overall criticality of the risk

Security Assessment Summary

***review commit hash* - 2ddc4249aebde6a805fe8a55b78c47202ad6b0dd**

Scope

The following smart contracts were in scope of the audit:

All contracts under:

- `./contracts`

The following number of issues were found, categorized by their severity:

- Critical & High: 1 issues
- Medium: 3 issues
- Low: 1 issues

Findings Summary

ID	Title	Severity
[H-01]	Initializing the pool in uniswap will lock all ETH in bridge	High
[M-01]	Referrer can steal commission from another referrer	Medium
[M-02]	No way to collect uniswap fees	Medium
[M-03]	Centralization issue	Medium
[L-01]	`closeMint` doesn't follow CEI	Low
[I-01]	No event emitted on important state changes	Informational
[I-02]	Misspelled event	Informational
[I-03]	Misspelled error message	Informational

Detailed Findings

[H-01] Initializing the pool in uniswap will lock all ETH in bridge

Severity

Impact: High

Likelihood: Medium

Description

The rollout of `$BLERB` tokens and `BLERB` NFTs work in stages. During the first stage users can mint NFTs using different referral codes. Each NFT will cost `0.1 ETH`. These ETH are split in three chunks, 50% goes to the protocol in what's called a genesis share. 25% will go to the referrers and the last 25% will be used to initialize a Uniswap v3 trading pool with initial liquidity.

The Uniswap pool is initialized when the admin calls `Bridge::closeMint`, then initialized with the newly minted `$BLERB` tokens and `wETH`:

```
    function _mintInitialPosition(
        uint256 amountWeth,
        uint256 amountToken
    ) private {
...
        INonfungiblePositionManager.MintParams
            memory params = INonfungiblePositionManager.MintParams({
                token0: token0,
                token1: token1,
                fee: FEE_TIER,
                tickLower: MIN_TICK,
                tickUpper: MAX_TICK,
                amount0Desired: amount0Desired,
                amount1Desired: amount1Desired,
                amount0Min: (amount0Desired * 90) / 100,
                amount1Min: (amount1Desired * 90) / 100,
                recipient: address(this),
                deadline: block.timestamp
            });

        (uint256 tokenId, uint256 liquidity, , ) = manager.mint(params);
...
    }
```

The issue is the slippage protection, `amountMin: (amountDesired * 90) / 100`. Someone can create the pool ahead of time with an initial price that will cause the Uniswap mint call to fail on `Price slippage check`. This will cause the whole call to `Bridge::closeMint` to fail and the `Bridge` to be forever stuck in the open minting state. Since the `$BLERB` tokens aren't minted until `closeMint` is called there is no easy way to recover from this.

The only possible recovery is for the protocol to allow an admin address to mint in `Token.sol`. Then mint some initial tokens and trade them in the pool to return the price. This would still leave the pool vulnerable to price manipulation before the call to `closeMint` is done. Thus it's not a waterproof way and would require the protocol to diverge from the plan to only mint `100_000 $BLERB` per NFT as this unplanned mint would be in excess of that.

PoC

Add this test to `Close.ts`:

```
it("Anyone can prevent closeMint by calling uniswap initialize directly", async () => {
  const fixture = await loadFixture(deployHybridFixture);
  const { bridge, token, others } = fixture;

  // nft mint happens
  await mintNfts(fixture);
  const positionManagerAddr =
"0x03a520b32C04BF3bEEf7BEb72E919cf822Ed34f1";
  const positionManager = await ethers.getContractAt(
    "INonfungiblePositionManager",
    positionManagerAddr
  );

  const wethAddr = "0x4200000000000000000000000000000000000006";
  const tokenAddr = await token.getAddress();
  const [token0, token1] =
    tokenAddr < wethAddr ? [tokenAddr, wethAddr] : [wethAddr, tokenAddr];

  const maxSqrtPrice = ethers.parseUnits(
    "1461446703485210103287273052203988822378723970341",
    0
  );

  // someone creates the pool early with another price
  const other = others[0];
  await positionManager
    .connect(other)
    .createAndInitializePoolIfNecessary(token0, token1, 10_000,
maxSqrtPrice);

  // admin cannot call close mint as pool price is wrong
  await expect(bridge.closeMint()).to.be.revertedWith("Price slippage
check");

  // tokens have not been minted
  expect(await token.totalSupply()).to.be.equal(0);
});
```

Recommendations

Consider initializing the pool yourself as soon as the token contract is created.

It's predetermined what the price will be because regardless of how many NFTs are minted the proportions between `$BLERB` and `wETH` will be the same, `9000 $BLERB` for every `0.025 wETH`. Hence the pool can be initialized to this price as soon as the address of the token contract is known. Preferably in the same tx to avoid any possibility of someone starting the pool at an unfavorable price. This price is then impossible to change until `closeMint` is called, as there are no `$BLERB` tokens to trade before then.

Blerb:

Fixed

[M-01] Referrer can steal commission from another referrer

Severity

Impact: High

Likelihood: Low

Description

Minting NFTs uses a referral system. Where the user minting can chose which referrer should get the commission from their mint. A mint costs `0.1 ETH` and the referrer gets 25% of this, `0.025 ETH`.

A user gets tied to their first referrer and any subsequent mints will always send the mint commissions to this referrer, in `ReferralMap::trySetReferrerForUser`:

```
        // Only set referrer ID once per user
        ShortString existingId = map._users[user];
        if (!isValidId(map, existingId)) {
            map._users[user] = storedId;

            // Emit Event for off-chain tracking
            emit UserOnbardedWithId(storedId, user);
            return true;
        }
        return false;
```

Hence, if a referrer sees that a user is minting multiple NFTs for another referrer they can mint an NFT for this user first using their own referral code. As long as the original mint was for >= 5 NFTs they will profit from this. Since each mint costs `0.1 ETH` and they gain `0.025 ETH` in commission per NFTs. So for 4 mints they would break even and 5 mints gain `0.025 ETH`.

PoC

```
it("lets other referrer can steal mint commission", async () => {
  const fixture = await loadFixture(deployHybridFixture);
  const { user, bridge, referrer, others, referral, genesis, nft } =
fixture;

  const other = others[0];
  await referral.generateReferrerId(other.address);
  const otherReferral = await referral.getReferralId(other.address);

  // malicious referrer front runs the original tx by minting one NFT for
the user with their referral
  await bridge.connect(other).mintAndForward(user.address, otherReferral,
{
    value: ethers.parseEther("0.1"),
  });

  // original tx using a referrer is executed
  await bridge
    .connect(user)
    .mintAndForward(user.address, referrer, { value:
ethers.parseEther("1") });

  // user still has 10 NFTs (but only paid for 9)
  expect(await nft.balanceOf(user.address)).to.be.equal(10);

  // original referrer got 0
  expect(
    await referral.getCommisson(genesis.address, ethers.ZeroAddress)
  ).to.be.equal(0);

  // front running referrer got all the commission but only spent 0.1,
thus made a 0.15 eth profit
  expect(
    await referral.getCommisson(other.address, ethers.ZeroAddress)
  ).to.be.equal(ethers.parseEther("0.25"));
});
```

Recommendations

Consider tracking the commission per purchase not only the first purchase.

Blerb:

Fixed

[M-02] No way to collect uniswap fees

Severity

Impact: Low

Likelihood: High

Description

When the mint stage is done an admin calls `Bridge::closeMint`, this will end the minting stage and divert some funds to initializing a Uniswap pool with a first LP position in `Bridge::_mintInitialPosition`:

```
609:        (uint256 tokenId, uint256 liquidity, , ) =
manager.mint(params);
610:
611:        lpTokenInfo.tokenId = uint80(tokenId);
612:        lpTokenInfo.liquidity = uint128(liquidity);
613:        lpTokenInfo.tickLower = MIN_TICK;
614:        lpTokenInfo.tickUpper = MAX_TICK;
```

Here the position is created and the details are saved. There is however no way to collect the fees that this position will gradually build up from trading happening in the pool.

Recommendations

Consider implementing a call to `NonfungiblePositionManager::collect` which can be restricted to just the owner and send the rewards to any address.

Blerb:

Fixed

[M-03] Centralization issues

Severity

Impact: High

Likelihood: Low

Description

The Hybrid NFT protocol uses OpenZeppelin AccessManager to guard sensitive calls withing the contracts.

These include: `Token::mint`, `Token::setTrusted`, `NFT::mint`, `NFT::burn`, `Referral::generateReferrerId`, `Bridge::closeMint`, `Bridge::enableBridge`, `Bridge::pullGenesisShare`.

Were the admin account in control of `AccessManager` to be compromised all the above calls would be available to an attacker which would cause great harm to the project.

Recommendations

We recommend the protocol to use at least a multisig for the admin control over `AccessManager` or a DAO setup. Also consider having a timelock setup so that users can react to changes done to the protocol.

Blerb:

Acknowledged

[L-01] `closeMint` doesn't follow CEI

To end the mint stage the admin calls `Bridge::closeMint`. There a couple of things are done but at the end the remaining `ETH` in the contract is sent to the admin then the state is change to `mintEnded`:

```
        if (address(this).balance > 0) {
            Address.sendValue(payable(msg.sender), address(this).balance);
        }

        // Close mint phase
        mintEnded = true;
```

The issue is that check-effects-interactions(CEI) is not followed here since the `mintEnded` state is changed after the external call (via `ETH` transfer) is done. Hence the admin could, albeit unlikely, reenter into `closeMint` to mint more `$BLERB` and create a new LP position (given the provide more `ETH` to the contract).

We recommend to move the state change before the transfer:

```diff
+       // Close mint phase
+       mintEnded = true;

        if (address(this).balance > 0) {
            Address.sendValue(payable(msg.sender), address(this).balance);
        }

-       // Close mint phase
-       mintEnded = true;
```

Blerb:

Fixed

[I-01] No event emitted on important state changes

Events are important for off-chain tracking of what is happening on-chain and it's good practice to emit events for any important state changes to the contract. Some admin calls in `Bridge.sol` are missing events.

Consider adding events to these three calls:

- `Bridge::closeMint`
- `Bridge::enableBridge`
- `Bridge::pullGenesisShare`

Blerb:

Fixed

[I-02] Misspelled event

When a user uses a referrer it is stored in a map. This also emits an even in `ReferralMap::trySetReferrerForUser`

```
            // Emit Event for off-chain tracking
            emit UserOnbardedWithId(storedId, user);
```

`UserOnbardedWithId` is however misspelled and should be `UserOnboardedWithId`. This could complicate tracking of these events as the tracker would have to remember that it is misspelled.

Consider renaming it to `UserOnboardedWithId`

Blerb:

Fixed

[I-03] Misspelled error message

The error message in `Referral::pullCommissions` is misspelled:

```
        require(amount >= assetConfig[asset].minAmount, "amout too low");
```

We recommend you change it to `amount`

Blerb:

Fixed