yAcademy LlamaLend V2 Review

Residents:

- blockdev
- engn33r

Fellows

- prady

Table of Contents

Review Summary

LlamaLend V2

LlamaLend is a protocol to borrow ETH against NFTs. Most of the NFT liquidity protocols don’t support small collections and need oracle support. LlamaLend allows anyone to run their own oracle for any collection, so NFT projects can create pools to let their holders borrow ETH against the NFTs instead of selling.

The contracts of the LlamaLend Repo were reviewed between January 9, 2023 and January 20, 2023. The review was limited to the latest commit at the start of the review.

This was commit eb8034f88a410d0dd3d6bce4e442cc69e0e12ea3 for the LlamaLend repo.

Scope

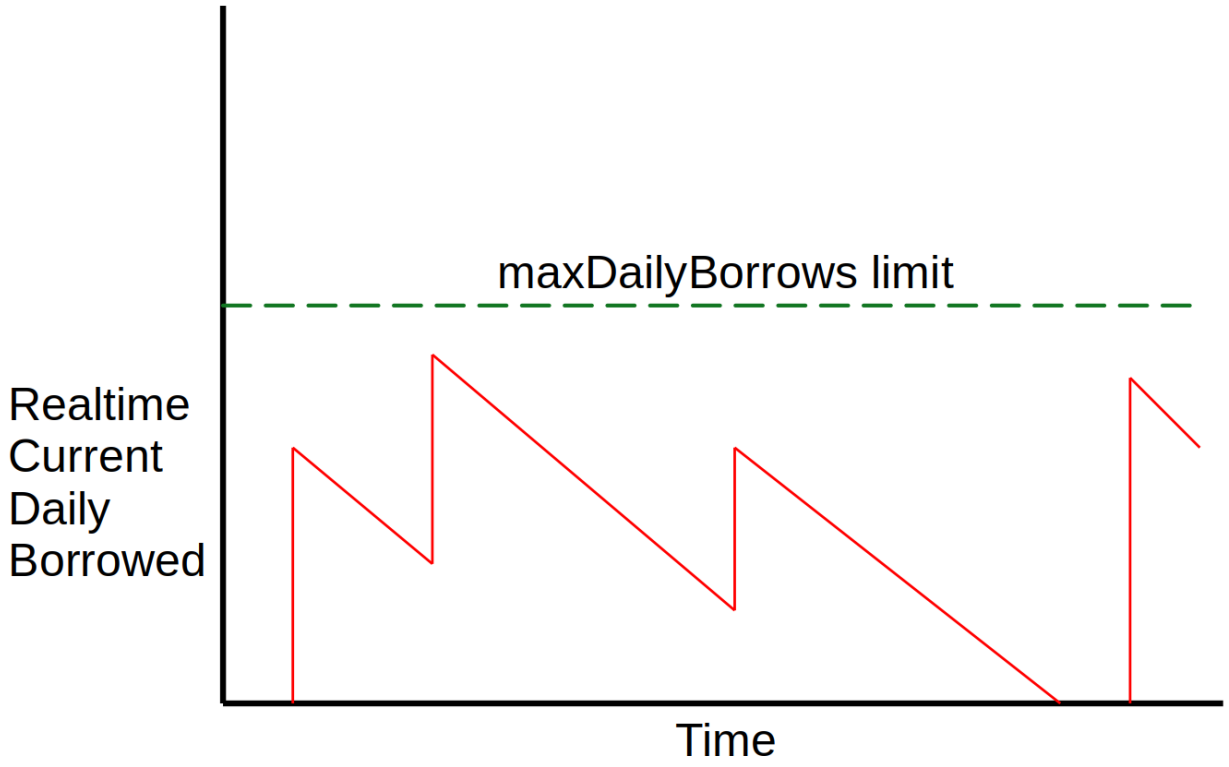
The scope of the review consisted of the following contracts at the specific commit. Note the oracle used by LlamaLend was not in scope of the review and is planned to be a custom off-chain solution:

- LlamaLendFactory.sol
- LendingPool.sol

Some of the unique features of LlamaLend are:

- the lending pools are all isolated
- 100% of the interest from lending goes to the pool owner
- The interest due on a loan is fixed at the time the loan is created and is independent of events in the pool afterwards
- the lending can use a wide variety of NFTs as collateral, not limiting collateral to just the most popular NFT collections
- flashloans with NFTs as collateral are free
- the lending period if short (roughly 2 weeks) but there is no immediate liquidation due to undercollateralization

One key variable in the LlamaLend design is the limitation on the amount that can be borrowed per day, or `maxDailyBorrows`. Each time a borrow happens, this increases the `currentDailyBorrows` value, and time that elapses without borrowing acts as a “cooldown” to reduce the effective `currentDailyBorrows` value. A visualization of how the math works in `addDailyBorrows()` is below.



This review is a code review to identify potential vulnerabilities in the code. The reviewers did not investigate security practices or operational security and assumed that privileged accounts could be trusted. The reviewers did not evaluate the security of the code relative to a standard or specification. The review may not have identified all potential attack vectors or areas of vulnerability. The review was a pro bono effort and is not guaranteed to follow the same process or to have been reviewed for the same amount of time as a typical yAcademy audit.

yAcademy and the residents make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAcademy and the residents do not represent nor imply to third parties that the code has been audited nor that the code is free from defects. By deploying or using the code, Rage Trade and users of the contracts agree to use the code at their own risk.

Findings Explanation

Findings are broken down into sections by their respective impact:

- Critical, High, Medium, Low impact
  - These are findings that range from attacks that may cause loss of funds, impact control/ownership of the contracts, or cause any unintended

consequences/actions that are outside the scope of the requirements

- Gas savings
  - Findings that can improve the gas efficiency of the contracts
- Informational
  - Findings including recommendations and best practices

Critical Findings

None.

High Findings

1. High - `createPool()` always reverts (blockdev)

Technical Details

`LLamalendFactory.createPool()` sends ether to the LendingPool contract:

```
payable(address(pool)).sendValue(msg.value);
```

Since the LendingPool contract doesn't have a `receive()` function, this transfer reverts, in turn reverting `createPool()`.

Impact

High. The factory cannot create lending pools.

Recommendation

Use `pool.deposit()` function to transfer ether to the pool in `createPool()`.

Developer Response

Acknowledged and fixed.

Medium Findings

1. Medium - Frontrunning borrow changes borrower's interest rate (engn33r)

The interest rate that a borrower pays for a loan is fixed. The rate depends on the amount borrowed before the loan was issued and the amount that will be borrowed in the loan. This means that if a borrow is frontrun, the borrower will have to pay a higher interest rate for the loan compared to if they were not frontrun. Furthermore, because the lender receives 100% of the interest from the interest of loans, the lender has minimal disincentive to perform the frontrunning because they will receive the interest that they have to pay for the loan.

Technical Details

The LlamaLend protocol calculates the interest rate of a loan in `calculateInterest()`. The rate curve is a linear 1-piece curve (unlike protocols like Aave that use a 2-part curve, with a steeper slope after a certain amount of borrowing). The interest rate for a loan is fixed, not variable like many other lending protocols where the rate depends on the total amount borrowed. To determine the fixed rate, LlamaLend uses the midpoint between the current borrowed amount of the new borrowed amount after the borrow action is complete. This calculation is in line 98. This introduces dependency on the amount borrowed at the time of the loan, or `totalBorrowed`. Frontrunning a borrow with another borrow can change the value of `totalBorrowed`.

Impact

Medium. The `maxInterest` argument in `borrow()` allows users to limit slippage, but frontrunning a borrow can cause a revert (temporary denial of service) or can cost the borrower more than expected.

Recommendation

The issue is caused by the combination of 1. fixed rate loans depending on the existing borrowed amount and 2. 100% of the interest getting paid to the lender. If one of these existing assumptions is modified, this issue would be resolved. For a comparison to other protocols, Aave or Compound does not return 100% of the interest to the lenders, which prevents this vector. And Uniswap takes a fee from each swap that disincentivize unlimited swapping which could be used for frontrunning.

Developer Response

Acknowledged and mitigated with a 10% platform fee. This is also fixed from frontend by setting slippage to 0% when calls are made, thus reverting the tx if the interest rate increased due to frontrunning. While this could create a DoS attack vector, the attacker

would lose money when performing the DoS due to tx+platform fees, so the cost would likely not be worth the benefit.

2. Medium - No support for cryptopunks (prady)

Technical Details

Cryptopunks are at the core of the NFT ecosystem. As one of the first NFTs, it embodies the culture of NFT marketplaces. But cryptopunks does not adhere to the ERC721 standard, that limits the contract ability to accept cryptopunks as collateral.

Impact

Medium.

Recommendation

Here is an example implementation on how cryptopunks can be integrated. Be sure to use an implementation that is safe for this specific cryptopunks issue.

Developer Response

Acknowledged but won't fix. LlamaLend targets the long tail of NFT collections. Cryptopunks are a huge collection that can be borrowed against in protocols such as JPEG'd, which are subsidizing rates for it massively and driving them towards 3-5%, which are insanely low compared to collections on llamalend usually (50-100%). Cryptopunks won't be supported.

Low Findings

1. Low - `baseURI` can be made a modifiable state variable (blockdev)

If `baseURI` is a constant variable, it cannot be updated if the server to which it points to becomes inaccessible.

Technical Details

`LendingPool.tokenURI()` uses this `baseURI`:

```
string private constant baseURI = "https://nft.llamalend.com/nft2/";
```

This is a constant variable and if access to `https://nft.llamalend.com/nft2/` is lost, `tokenURI()` will start returning an inaccessible URI.

Impact

Low. It doesn't impact any smart contract functionality, but UI can break if it depends on this URI.

Recommendation

- Make `baseURI` a state variable and define a setter function to set `baseURI` which can only be called through factory (or factory owner).
- Or, define `baseURI` in factory and make LendingPool fetch it from the factory so that you need to set and update it only once.

Developer Response

Acknowledged and fixed.

2. Low - Consider removing `LlamaLendFactory.receive()` (blockdev, prady)

Technical Details

There is no need to send ether to LlamaLendFactory contract explicitly, so `receive()` can be removed, but even if someone sends ETH by mistake to the contract address, any one can extract them by passing empty data to `repay`.

Impact

Low. It doesn't impact any smart contract functionality, but any ETH sent to LlamaLendFactory can be used to repay loans.

Recommendation

Remove `receive()` function in LlamaLendFactory.

Developer Response

Disputed, because llamalend pool contracts send ETH back to the factory so the factory needs to have `receive()` to receive that ETH.

3. Low - `calculateInterest()` does not account for pending interest payments (engn33r)

`calculateInterest()` calculates the interest for borrowing. One of the values in this calculation is the total value owned by the pool. This value does not consider the pending interest payments that existing borrowers own, which underestimates the total assets owned by the pool.

Technical Details

When a pool has an outstanding borrowed amount, there is a corresponding amount of interest that is expected to be paid in the future. When the interest is calculated for new loans, the total value of the pool is calculated as the current pool balance plus the amount of value borrowed, which is the denominator in the calculation of `variableRate`. This total value does not consider the pending interest payments that are due. The result is that the total pool balance is slightly underestimated. While the existing approach of ignoring future interest payments until they are paid may have some benefit, it creates a discontinuity where the total assets held by the pool before and after a user calls `repay()` results in an immediate increase in pool assets.

Impact

Low. The pending interest payments are not considered part of the pool balance.

Recommendation

Document that the interest owed to the pool is not included in calculations until the amount is actually received by the pool.

Developer Response

Acknowledged, fixed by adding a comment to document this.

4. Low - Risk of bad debt (engn33r)

`doEffectiveAltruism()` liquidates loans that are overdue and have not been paid. This function can only be called on overdue loans, which means that undercollateralized loans cannot be liquidated until they mature beyond `loan.deadline`.

Technical Details

The README in the LlamaLend repository has this line in the “Risks for LPs” section

> You are selling put options on NFTs, if NFT price drops >66% before some loan expires, user will likely not repay and you’ll get the NFT at a loss.

Lending protocols like Aave and Compound have liquidation policies that limit the amount of bad debt that can accumulate in the protocol. This is no such protection in LlamaLend, so liquidation is only possible after `loan.deadline` regardless of how much bad debt accumulates. The LlamaLend README suggest a `maxLoanLength` value of 2 weeks will be used, but this is not hardcoded into the contracts, meaning that a longer deadline could increase the risk of bad debt and loss of value for the lender.

Impact

Low. The risk is mitigated by the short borrowing timeframes and this risk is already documented in LlamaLend docs.

Recommendation

Monitor the volatility of NFT floor prices for the largest LlamaLend pools to verify that the chosen `maxLoanLength` values are safe and are not at risk of undercollateralization in adverse market condition.

Developer Response

Acknowledged, no fix needed because this risk is part of the protocol design. This is the reason why APRs are usually high, because llamalend loans also offer protection against price plunges as a feature of the protocol.

5. Low - Oracle data replay attack (engn33r)

The data from the oracle is valid with `checkOracle()` within a certain time window, roughly 20 minutes. During this time window, any oracle data can be reused.

Technical Details

There is no requirement in `checkOracle()` that only the latest oracle data can be used. Instead, there is a `deadline` and any oracle data in the timeframe allowed by `deadline` is accepted. A user can look at the data from the last 20 minutes and choose the oracle data with a price that is most beneficial to them. This lets users “turn back to clock” if they don’t like the latest data that the oracle is returning. This means that the oracle must be sure not to return a zero value or underestimated value for the NFT floor price in any edge case scenario, such as before the NFT floor price data is available. For comparison, when Chainlink oracle data is used, it is normally recommended to use only the latest round of oracle data, which removes the ability for a user to choose which datapoint among recent data to use.

Impact

Low. Oracle data can be reused within a certain timeframe, but since the off-chain oracle itself is out of scope of this review, the full impact of this cannot be determined.

Recommendation

Test the off-chain oracle for all edge cases, including the behaviour before NFT floor prices are available, and consider reducing the deadline timeframe to a shorter amount, say 5 minutes or less.

Developer Response

Acknowledged, the oracle deadline is reduced to 10 minutes, the only downside is that if the tx stays in the mempool for too long it will revert. But with a 10 minute deadline the chances should be quite low. If an oracle returns a 0 value or underestimates NFT prices, nothing will happen (and actually oracles regularly do this) as it will just make the loans not attractive due to giving too little ETH. The only danger is if the oracle is overestimating the price, since it would be possible to drain the pool by taking loans and buying more NFTs.

6. Low - NFT can get locked in contract if `from` is a smart contract (prady)

Technical Details

User will receive the collateral NFT when repay() is called. However, if `from` is a contract address that does not support ERC721, the collateral NFT can be frozen in the contract.

As per EIP-721: "A wallet/broker/auction application MUST implement the wallet interface if it will accept safe transfers."

```
    function _repay(Loan calldata loan, address from)
        internal
        returns (uint256)
    {
        ...
        IERC721(loan.nftContract).transferFrom(address(this), from, loan.nft);
        ...
    }
```

Impact

Low.

Recommendation

Use `safeTransferFrom` instead of `transferFrom`.

Developer Response

Acknowledged, won't fix. The downside of using `safeTransferFrom()` is that any contracts that don't implement ERC721 receiver functions will be unable to repay loans. I think the main purpose of safeTransferFrom() was to prevent sending NFTs to third parties that

couldn't use them, such as when making regular transfers, where you can send to NFTs to contracts that would have no awareness of them. However in this case, the only contracts that will ever call repay are:

1. contracts that can do arbitrary calls -> these will have no issue transferring the NFTs
2. contracts that specifically integrated llamalend -> by definition these will know how to handle NFTs

So that logic doesn't apply here. IMO this change would do more harm than good by preventing repayments.

Gas Savings Findings

1. Gas - Use custom errors (blockdev)

Technical Details

Custom errors are more gas efficient than error strings.

Impact

Gas savings.

Recommendation

Replace all error strings with custom errors.

Developer Response

Acknowledged, won't fix to keep code simple and more similar to llamalend v1.

2. Gas - Variables are auto-initialized to `0` (blockdev)

Technical Details

LendingPool.sol#L277-L279 creates an `else` clause to assign `0` to a return value which is already `0`:

```
} else {
    lateFees = 0;
}
```

Impact

Gas savings.

Recommendation

Remove this `else` clause.

Developer Response

Acknowledged and fixed.

3. Gas - Extra event is emitted by `_burnWithoutBalanceChanges()`

Technical Details

In a normal `_burn()` function, only a `Transfer` event is emitted. `_burnWithoutBalanceChanges()` emits 2 events: `Transfer` and `Approval`. The reason being it calls `_approve()` to clear approval, but `_burn()` function just deletes `_tokenApprovals` mapping.

Impact

Gas savings.

Recommendation

Explicitly delete `_tokenApprovals` mapping instead of calling `_approve()` function in `_burnWithoutBalanceChanges()`.

Developer Response

Acknowledged and fixed.

4. Gas - Use unchecked if no underflow risk (engn33r)

There is a subtraction operation that can use unchecked for gas savings.

Technical Details

Because `lastUpdateDailyBorrows` is only modified by setting the variable equal to block.timestamp, `block.timestamp - lastUpdateDailyBorrows` can never underflow. This line and this line can be unchecked. The same logic applies to `loan.startTime` so this line can be unchecked.

Impact

Gas savings.

Recommendation

Use unchecked arithmetic if there is no overflow or underflow risk for gas savings.

Developer Response

Acknowledged, won't fix. The problem here is that this suggestion relies on the assumption that timestamps in the blockchain will be monotonically increasing, however when miners submit a timestamp they have some leeway on the timestamp they pick. Currently this optimization would work because geth's block validation rejects blocks that have a timestamp lower than the timestamp of the previous block (source: https://github.com/ethereum/go-ethereum/blob/master/consensus/ethash/consensus.go#L274), however other chains could remove that check or it could be removed in future versions of ethereum or other chains, and if that was removed it would be possible to submit blocks that have timestamps that are lower than the previous block. If that happens it would introduce a huge security vuln due underflow, so this just isn't worth the risk.

5. Gas - `factory` can be marked as `immutable` (prady)

Technical Details

Since there is no function in the contract that updates the `factory` variable address, it can be marked as immutable.

Impact

Gas savings.

Recommendation

```
- address public factory;
+ address immutable public factory;
```

Developer Response

Disputed, can't be made immutable because contracts use proxies, so all variables need to be initialized in `initialize()`, which doesn't support immutable vars.

Informational Findings

1. Informational - Foundry and hardhat tests not working (blockdev)

Unit tests verify the correctness of the functionality, also making sure that functionality doesn't break in future code changes.

Technical Details

Hardhat and Foundry tests are not working as they refer to non-existent contracts and variables, or call functions with different number of arguments.

Impact

Informational. Even though the severity is informational, the finding "`createPool()` *reverts*" would have been caught if the tests were working. It's important to keep the tests up-to-date.

Recommendation

Fix tests.

2. Informational - `_burnWithoutBalanceChanges()` shadows `owner` (blockdev)

Technical Details

The second argument in `_burnWithoutBalanceChanges(tokenId, owner)` shadows the global variable `owner`.

Impact

Informational.

Recommendation

Rename the local variable `owner` to `_owner` to not confuse it with the global variable of the same name.

3. Informational - Consider allowing anyone to repay any loan (blockdev)

Currently only the loan owner is allowed to initiate `repay()`. You can consider allowing anyone to repay a loan, hence removing the `from` restriction.

Technical Details

LendingPool.sol#L203, LendingPool.sol#L169

Impact

Informational.

Recommendation

Let any address pay for a loan.

4. Informational - Update OpenZeppelin dependencies (engn33r)

The package.json file shows old OpenZeppelin dependencies are used. Update these to the latest available version, which is currently v4.8.1.

Technical Details

The package.json file shows a dependency of openzeppelin/contracts v4.2.0 and openzeppelin/contracts-upgradeable v4.8.0-rc1. All dependencies should be updated to the latest version available.

Impact

Informational.

Recommendation

Use the latest version of OpenZeppelin libraries for production.

5. Informational - Flashloans are free (engn33r)

Interest for the repayment of a loan is calculated with `block.timestamp - loan.startTime`. This value is zero when a flashloan happens, so the borrower would pay no interest.

Technical Details

The interest due on a loan is calculated in `_repay()` and `infoToRepayLoan()`. The interest due for a flashloan is zero, meaning there is no fee for this borrowing operation. Whether this is intentional is unclear.

Impact

Informational.

Recommendation

If the design is intended to allow this, document that flashloans have no fees so users can more clearly see this as an advantage, or consider building in a specialized flashloan function to make this feature easier to use. If this is an unintentional side effect of the design, consider adjusting `_repay()` and `infoToRepayLoan()` to set an interest value based on a constant flashloan fee in the case where `block.timestamp == loan.startTime`.

6. Informational - Interest rate “renegotiation” after a repayment (engn33r)

If a pool has three loans with different deadlines, the third loan (the most recent) will have a high interest rate than the earlier loans. But when the first loan is repaid, there is

an opportunity for the third loan to repay and immediately borrow a new loan for the same amount but with a lower interest rate.

Technical Details

Assume the following scenario:

1. Alice takes out a loan on day 1 with 5% interest
2. Bob takes out a loan on day 3 with 7% interest
3. Carol takes out a loan on day 5 with 9% interest

If the loan deadline is 14 days from the start date, then on day 15 Alice will repay her loan. At this point, Bob and Carol can compete to get lower interest rates and the result may be that Bob has a new loan for the same amount with 5% interest while Carol's loan is with 7% interest. This is because `calculateInterest()` depends on the amount that has been borrowed from the pool, and this amount is reduced when a repayment happens.

Impact

Informational.

Recommendation

This is an artifact of the LlamaLend incentives with the current design, so there may be no need to redesign incentives to modify this if it is intended behaviour. If this is unintended, consider triggered a new function when repayment happens to lower the interest rate amount for existing loans to avoid the need for borrowers to recreate their loan to acquire the lower rate.

7. Informational - Inaccurate comment (engn33r)

The word "overflow" should be changed to "underflow" in one comment.

Technical Details

The word overflow in this comment that reads "overflow checks implictly check that amount is enough" should be changed underflow.

Impact

Informational.

Recommendation

Improve comment accuracy.

8. Informational - Align `_burnWithoutBalanceChanges()` with `_burn()` (engn33r)

The ERC721 `_burn()` is slightly different from the custom `_burnWithoutBalanceChanges()`.

Technical Details

Although it does not change the end result, the following change in `_burnWithoutBalanceChanges()` can be made to use the same approach as `_burn()`

```
- _approve(address(0), tokenId);
+ delete _tokenApprovals[tokenId];
```

Impact

Informational.

Recommendation

Align the implementation of `_burnWithoutBalanceChanges()` to the standard `_burn()`.

9. Informational - Missing events for critical operations (prady)

Several critical operations do not trigger events. As a result, it is difficult to check the behavior of the contracts.

Technical Details

Without events, users and blockchain-monitoring systems cannot easily detect suspicious behavior. Ideally, the following critical operations should trigger events:

- withdraw()
- setMaxDailyBorrows()
- setOracle()
- emergencyShutdown()
- doEffectiveAltruism()

Impact

Informational.

Recommendation

Add events for all critical operations. Events aid in contract monitoring and the detection of suspicious behavior.

10. Informational - Revert operation performed without proper error message (prady)

Several function performs `require` check, without reverting with any error message.

Technical Details

For ex: In function `borrow()` it checks that current interest rate should be less than maximumInterestRate, but it does not revert with error message, which can be confusing to use, as error message helps to understand the reason why transaction failed.

```
function borrow() {
    ...
    require(interest <= maxInterest);
    ...
}
```

Impact

Informational.

Recommendation

User proper error messaging.

Final remarks

blockdev

The protocol's design is intelligent and puts well-reasoned trust on different actors.

engn33r

The LlamaLend protocol takes a very simplistic approach to a lending and borrowing protocol, removing substantial amounts of complexity that many other borrowing protocols implement. While this simplification does cut a couple of corners, the compromises seem reasonable given the scale and intended use cases of the protocol.

About yAcademy

yAcademy is an ecosystem initiative started by Yearn Finance and its ecosystem partners to bootstrap sustainable and collaborative blockchain security reviews and to nurture aspiring security talent. yAcademy includes a fellowship program, a residents program, and a guest auditor program. In the fellowship program, fellows perform a series of periodic security reviews and presentations during the program. Residents are past fellows who continue to gain experience by performing security reviews of contracts submitted to yAcademy for review (such as this contract). Guest auditors are experts with a track record in the security space who temporarily assist with the review efforts.