yAudit Llamapay V2 Review

Review Resources:

- LlamaPay V1 Docs

Auditors:

- blockdev
- engn33r

Fellows:

- spalen
- pandadefi
- prady
- hasanza
- PraneshASP

Table of Contents

1. Review Summary
2. Scope
3. Code Evaluation Matrix
4. Findings Explanation
5. Critical Findings
    a. 1. Critical - Infinite withdrawals using function `withdrawAllWithRedirect(uint256 _id)` (spalen)
        a. Technical Details
        b. Impact
        c. Recommendation
        d. Developer Response
6. High Findings

a 1. High - Inaccurate internal token balance for weird ERC20 tokens (engn33r)
    a Technical Details
    b Impact
    c Recommendation
    d Developer Response
7 Medium Findings
  a 1. Medium - `burnStream()` shouldn’t burn a stream if it has some debt (prady)
    a Technical Details
    b Impact
    c Recommendation
    d Developer Response
  b 2. Medium - `Payee` funds get locked if stream is resumed after being burned (prady)
    a Technical Details
    b Impact
    c Recommendation
    d Developer Response
8 Low Findings
  a 1. Low - Llama Pay V2 is incompatible with high decimal tokens (engn33r)
    a Technical Details
    b Impact
    c Recommendation
    d Developer Response
  b 2. Low - `permitToken` fails in case of tokens that doesn’t follow `IERC20Permit` standard (prady)
    a Technical Details
    b Impact
    c Recommendation
    d Developer Response
  c 3. Low - Handle funds transferred via `transfer` method (prady)

a Technical Details
b Impact
c Recommendation
d Developer Response

d 4. Low - Missing Input Validation (hasanza)
a Technical Details
b Impact
c Recommendation
d Developer Response

e 5. Low - `_updateToken` updates `token.lastUpdate` even if the token doesn’t exists (prady)
a Technical Details
b Impact
c Recommendation
d Developer Response

f 6. Low - `BoringBatchable.batch()` is `payable` (blockdev)
a Technical Details
b Impact
c Recommendation
d Developer Response

g 7. Low - `modifyStream()` should revert for inactive stream (blockdev)
a Technical Details
b Impact
c Recommendation
d Developer Response

h 8. Low - `withdrawable()` can revert (blockdev, pandadefi)
a Technical Details
b Impact
c Recommendation
d Developer Response

i 9. Low - Inaccuracies in `withdrawable()` (blockdev, PraneshASP)
  a Technical Details
  b Impact
  c Recommendation
  d Developer Response
j 10. Low - `streams` can be resumed even after `burn` (prady)
  a Technical Details
  b Impact
  c Recommendation
  d Developer Response
k 11. Low - `modifyStream` can create a new `stream` if it doesn’t exist (prady)
  a Technical Details
  b Impact
  c Recommendation
  d Developer Response
l 12. Low - `payerWhitelists` addresses should have access to all withdraw operations (engn33r)
  a Technical Details
  b Impact
  c Recommendation
  d Developer Response
m 13. Low - Rogue `payerWhitelists` address can steal value (engn33r)
  a Technical Details
  b Impact
  c Recommendation
  d Developer Response
9 Gas Savings Findings
  a 1. Gas - Use cheaper comparison (spalen)
    a Technical Details

    b Impact
    c Recommendation
  b 2. Gas - Incorrect function visibility (pandadefi, engn33r)
    a Technical Details
    b Impact
    c Recommendation
  c 3. Gas - `owner` should be a immutable variable (pandadefi)
    a Technical Details
    b Impact
    c Recommendation
  d 4. Gas - Return stream value after updating (spalen)
    a Technical Details
    b Impact
    c Recommendation
  e 5. Gas - Use unchecked if no underflow risk (engn33r)
    a Technical Details
    b Impact
    c Recommendation
  f 6. Gas - Do not load whole struct if not required (prady)
    a Technical Details
    b Impact
    c Recommendation
  g 7. Gas - Pass struct as a params to `_createStream()` (prady)
    a Technical Details
    b Impact
    c Recommendation
  h 8. Gas - `if` condition can be optimized in `withdrawable()` (blockdev)
    a Technical Details
    b Impact

c Recommendation

i 9. Gas - Use `calldata` instead of `memory` if function param remains unchanged (prady)

a Technical Details

b Impact

c Recommendation

j 10. Gas - Consider and compare UX when streams are ERC1155 tokens (blockdev)

a Technical Details

b Impact

c Recommendation

10 Informational Findings

a 1. Informational - Missing Indexed Event Parameters (hasanza)

a Technical Details

b Impact

c Recommendation

b 2. Informational - External createStream functions should return the streamId (hasanza)

a Technical Details

b Impact

c Recommendation

c 3. Informational - Update BoringBatchable to latest version (engn33r)

a Technical Details

b Impact

c Recommendation

d 4. Informational - Add tests for weird ERC20 tokens (engn33r)

a Technical Details

b Impact

c Recommendation

e 5. Informational - Sanitisation to avoid false event emission (prady)

    a. Technical Details
    b. Impact
    c. Recommendation
f. 6. Informational - `tokenURI()` should revert for non-existent tokens (blockdev)
    a. Technical Details
    b. Impact
    c. Recommendation
g. 7. Informational - Missing events for critical operations (prady)
    a. Technical Details
    b. Impact
    c. Recommendation
h. 8. Informational - Lack of zero check on functions (prady)
    a. Technical Details
    b. Impact
    c. Recommendation
i. 9. Informational - `onlyOwnerAndWhitelisted` should be renamed `onlyOwnerOrWhitelisted` (blockdev)
    a. Technical Details
    b. Impact
    c. Recommendation
j. 10. Informational - `unchecked` has no effect (blockdev)
    a. Technical Details
    b. Impact
    c. Recommendation
k. 11. Informational - Use consistent decimals format in functions (prady)
    a. Technical Details
    b. Impact
    c. Recommendation
l. 12. Informational - Use function to reduce duplicacy of code (prady)

a Technical Details
b Impact
c Recommendation
m 13. Informational - Redirect recipient cannot receive funds directly (engn33r)
a Technical Details
b Impact
c Recommendation
n 14. Informational - Typos (engn33r)
a Technical Details
b Impact
c Recommendation
o 15. Informational - Stream owner should be able to call `updateStream()` (blockdev)
a Technical Details
b Impact
c Recommendation
11 Final remarks
a blockdev
b engn33r
12 About yAcademy

Review Summary

LlamaPay V2

LlamaPay V2 provides a solution to streaming payments, as well as scheduling streams with a start and end date. This is designed to extend the use cases of LlamaPay V1, which focuses on automating salary transactions from protocols to employees. Each LlamaPayV2Payer is an ERC721, unlike the V1 version of the Payer contract.

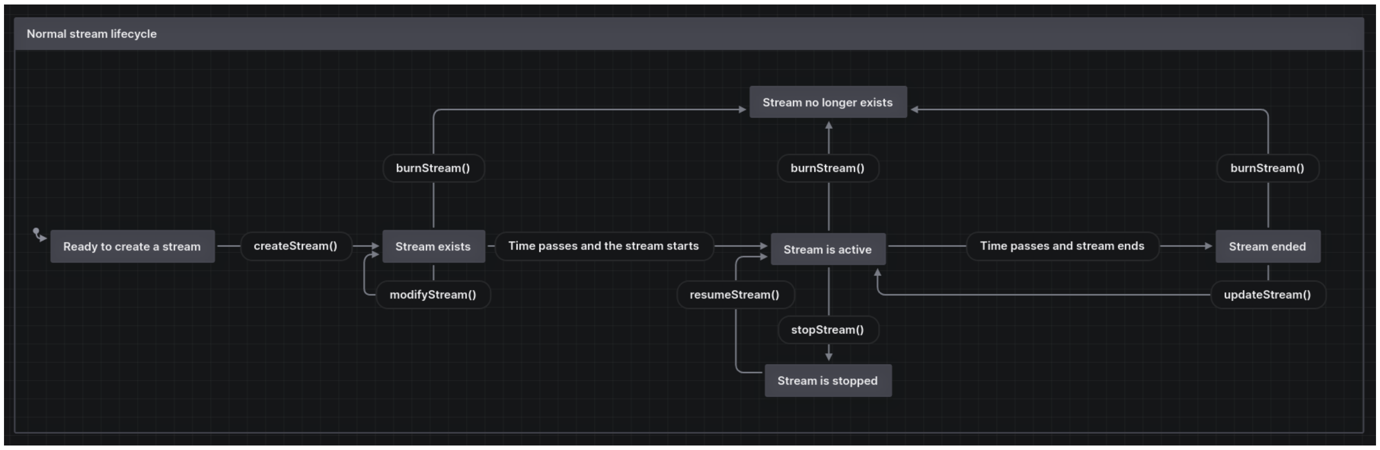
The contracts of the LlamaPay V2 Repo were reviewed over 7 days. The code review was performed by 2 auditors and several Fellows between December 11 and December 18, 2022. The repository was under active development during the review, but the review was limited to the latest commit at the start of the review. This was commit fb4a171d6969062519e92c579a8f4cdedb99ab77 for the LlamaPay V2 repo.

Scope

The scope of the review consisted of the following contracts at the specific commit:

- LlamaPayV2Factory.sol
- LlamaPayV2Payer.sol
- BoringBatchable.sol

The stream is time-sensitive in the sense that specific actions are intended to happen at specific times in the proper sequence of operations. Below is a visualization of the intended call sequence of stream functions and a list of functions that cannot be called at certain points during the stream's duration.



1 Before start of stream (lastPaid == block.timestamp && stream.starts > block.timestamp)

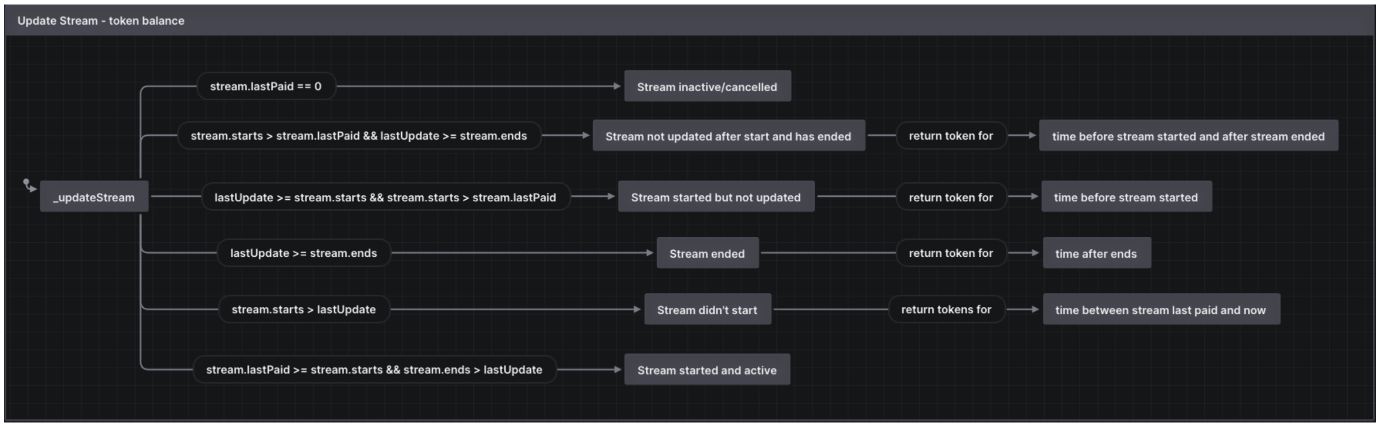
- modifyStream()
- ~~stopStream()~~
- ~~resumeStream()~~
- burnStream()
- updateStream()

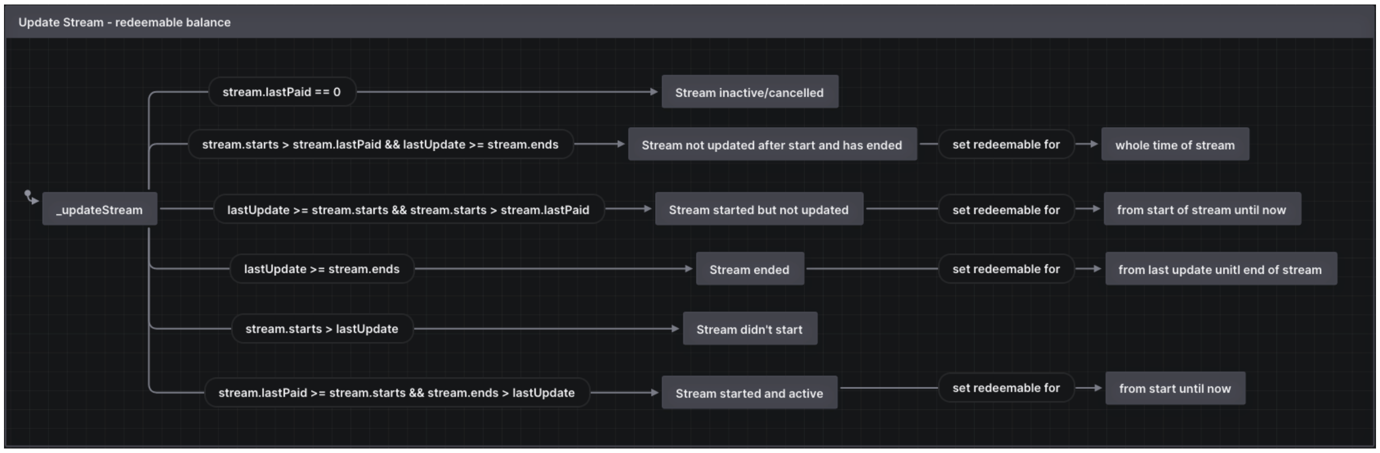
2 Between start and end of stream (lastUpdate >= stream.starts)

- modifyStream()

    - stopStream()
    - resumeStream() → only callable if stream is stopped (stream.lastPaid == 0)
    - ~~burnStream()~~
    - updateStream()
3. After end of stream (lastPaid == 0 && block.timestamp >= stream.ends)
    - modifyStream()
    - ~~stopStream()~~
    - ~~resumeStream()~~
    - burnStream()
    - updateStream()

At a lower level, `_updateStream()` plays an important role in updating the stream parameters when a stream is stopped, modified, or withdrawals happen. Diagrams visualizing the logic flow of `_updateStream()` in different scenarios is shown below.


Update Stream - token balance
_updateStream
stream.lastPaid == 0
Stream inactive/cancelled
stream.starts > stream.lastPaid && lastUpdate >= stream.ends
Stream not updated after start and has ended
return token for
time before stream started and after stream ended
lastUpdate >= stream.starts && stream.starts > stream.lastPaid
Stream started but not updated
return token for
time before stream started
lastUpdate >= stream.ends
Stream ended
return token for
time after ends
stream.starts > lastUpdate
Stream didn't start
return tokens for
time between stream last paid and now
stream.lastPaid >= stream.starts && stream.ends > lastUpdate
Stream started and active


Update Stream - redeemable balance
_updateStream
stream.lastPaid == 0
Stream inactive/cancelled
stream.starts > stream.lastPaid && lastUpdate >= stream.ends
Stream not updated after start and has ended
set redeemable for
whole time of stream
lastUpdate >= stream.starts && stream.starts > stream.lastPaid
Stream started but not updated
set redeemable for
from start of stream until now
lastUpdate >= stream.ends
Stream ended
set redeemable for
from last update unitl end of stream
stream.starts > lastUpdate
Stream didn't start
stream.lastPaid >= stream.starts && stream.ends > lastUpdate
Stream started and active
set redeemable for
from start until now

After the findings were presented to the LlamaPay development team, fixes were made and included in several PRs.

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 yAudit audit.

yAudit and the auditors make no warranties regarding the security of the code and do not warrant that the code is free from defects. yAudit and the auditors 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, LlamaPay V2 and users of the contracts agree to use the code at their own risk.

Code Evaluation Matrix

Category	Mark	Description
Access Control	Good	The only access controls are for the owners of the individual LlamaPayV2Payer contracts (often this will mean a separate contract for each employer) and any addresses they add to their unique whitelist. There are no admin access controls for the LlamaPay team because the contracts are designed to be immutable and funds managed by the individual payers.
Mathematics	Average	No abnormally complex math was used, but internal accounting is designed to properly keep track of token balanced held by LlamaPayV2Payer. Some of the internal accounting should be improved to account for different ERC20 edge cases.
Complexity	Low	There are many withdraw functions with slightly different logic and complex branching logic in `_updateStream()`. Given the current level of code coverage, it is likely that issues exist in edge cases that are hard to detect unless all

Category	Mark	Description
		permutations of the branching logic is examined. The branching and tests should be written in a way that it is easy to determine that all cases are covered. Consider a visual showing the points in time or the criteria that must be satisfied for certain functions to be called. Positive and negative tests should then test each of these criteria in different permutations to confirm the functions can only be called when that specific set of criteria is met.
Libraries	Good	Only solmate token libraries and BoringBatchable were imported to the LlamaPayV2Payer contract, which is less dependencies than most protocols.
Decentralization	Good	The LlamaPay V2 contracts are designed to be immutable and only the payment sender (normally the employer) has control over the stream.
Code stability	Average	LlamaPay V1 is already in production with 7 figure TVL, but LlamaPay V2 is still a work in progress with commits made on the same day as the start of this review.
Documentation	Good	All functions had solid NatSpec comments, and the complex function `_createStream()` and `_updateToken()` had inline comments for further clarification.
Monitoring	Average	Some important functions that modify state variables are missing events, including `cancelDebt()`, `repayAllDebt()`, `repayDebt()`, and `updateStream()`. Adding events to important functions enables easier monitoring and on-chain analysis.
Testing and verification	Low	Foundry code coverage showed only 74% of LlamaPayV2Payer lines and 52.22% of LlamaPayV2Payer branches were covered by tests. 0% of BoringBatchable.sol was covered by tests. Test coverage should be as close to 100% before production deployment as possible, and the current coverage is considered relatively low.

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

1. Critical - Infinite withdrawals using function `withdrawAllWithRedirect(uint256 _id)` (spalen)

The function `withdrawAllWithRedirect(uint256 _id)` enables the user to withdraw the token, but it doesn’t update `redeemables` for the given token `_id`.

Technical Details

In the following example, Bob can withdraw all tokens in the contract. The first call only withdraws the funds allocated to him, but because it’s not recorded, Bob can call withdraw again. Here’s an extended test.

```
function testWithdrawAllWithRedirect() external {
    vm.prank(alice);
    llamaPayV2Payer.createStream(
        address(llamaToken),
        bob,
        1e20,
        10000,
        10000000
    );
    vm.prank(bob);
    llamaPayV2Payer.addRedirectStream(0, steve);
    vm.warp(15000);
    vm.prank(bob);
    llamaPayV2Payer.withdrawAllWithRedirect(0);
    (uint256 balance, , , uint48 lastUpdate) = llamaPayV2Payer.tokens(
        address(llamaToken)
    );
    assertEq(llamaToken.balanceOf(address(llamaPayV2Payer)), 5000 * 1e18);
    assertEq(llamaToken.balanceOf(steve), 5000 * 1e18);
    assertEq(balance, 5000 * 1e20);
    assertEq(lastUpdate, 15000);
    // redeemables should be 0 but it's not
    assertGt(llamaPayV2Payer.redeemables(0), 0);

    // Withdraw all again
    vm.prank(bob);
    llamaPayV2Payer.withdrawAllWithRedirect(0);
    assertEq(llamaToken.balanceOf(address(llamaPayV2Payer)), 0);
    assertEq(llamaToken.balanceOf(steve), 2 * 5000 * 1e18);
    assertEq(lastUpdate, 15000);

    // redeemables should be 0 but it's not
    assertGt(llamaPayV2Payer.redeemables(0), 0);
}
```

Impact

Critical. The user can call `withdrawAllWithRedirect(uint256 _id)` multiple times and drain token defined in the stream from the contract.

Recommendation

The function should set `redeemables[id]` to 0 after LlamaPayV2Payer.sol#L276.

```
uint256 toRedeem = redeemables[_id] / tokens[stream.token].divisor;
```

```
redeemables[_id] = 0;
```

Also, it would be good to check the value `redeemables` in the tests.

Developer Response

Fixed.

High Findings

1. High - Inaccurate internal token balance for weird ERC20 tokens (engn33r)

LlamaPayV2Payer maintains an internal balance of the tokens held by the contract. The internal balance will be inaccurate in the case of fee-on-transfer tokens and rebasing tokens.

Technical Details

When a token is deposited into LlamaPayV2Payer, the internal balance is updated and the transfer of tokens into the contract happens. The key lines are:

```
tokens[_token].balance += _amount * tokens[_token].divisor;
token.safeTransferFrom(msg.sender, address(this), _amount);
```

The problem is that if a fee-on-transfer exists, the `tokens[_token].balance` will be greater than the balanceOf token amount in the contract, because the `tokens[_token].balance` did not subtract the fee-on-transfer amount. A receiver could withdraw more tokens than they should have access to with these steps:

1. The payer deposits `X` number of fee-on-transfer tokens into the LlamaPayV2Payer contract
2. The internal balance thinks the contract holds `X` tokens, but actually the contract holds `X - tokenFee` tokens
3. The contract accounting does not consider the fee involved for fee-on-transfer tokens. If the receiver is eligible to withdraw the `X` amount of tokens, they can only withdraw `X - tokenFee` tokens because this is the amount held by the contract. If fee-on-transfer tokens are permitted, the accounting should only permit a withdrawal of `X - tokenFee`.
4. The payer deposits `Y` number of fee-on-transfer tokens into the LlamaPayV2Payer

contract.

5. If `X - tokenFee` + `Y - tokenFee` > `X`, then the receiver can now withdraw the `X` amount of tokens because the contract holds enough tokens to allow this withdrawal. If fee-on-transfer tokens are permitted, the accounting should only permit a withdrawal of `X - tokenFee`, but with the current code, the receiver with withdraw `X` tokens.

Likewise, if a rebasing token is used and the balanceOf token amount in the contract is a function of time, the `tokens[_token].balance` value is not increased or decreased outside of deposit or withdraw events and therefore the internal accounting may underestimate the tokens held by the contract. This latter case could lock funds in the contract, for example:

1. The payer deposits `X` number of rebasing tokens into the LlamaPayV2Payer contract
2. Over time, the `X` rebasing tokens grow to `1.25 * X` tokens
3. Once the stream ends and the receiver can withdraw the full token balance, the internal account balance still assumes the contract holds only `X` tokens, so only `X` tokens and not `1.25 * X` tokens can be withdrawn.

Impact

High. If rebasing tokens are supplied to a LlamaPayV2Payer contract, like Aave aTokens, tokens can be locked in the contract. A receiver could withdraw more fee-on-transfer tokens than they should have access to.

Recommendation

To prevent the case of fee-on-transfer tokens, add a check such as `require(tokens[_token].balance == token.balanceOf(address(this)));` at the end of `deposit()` to prevent a mismatch between the internal balance and the actual contract balance. To allow compatibility with rebasing tokens, consider an ownerOnly withdraw function that could use the current token balance held by the contract instead of the internal accounting balance, retrievable only when there are no active streams with this token (after block.timestamp is greater than the end time of any stream using this token).

Developer Response

Will document limitation.

Medium Findings

1. Medium - `burnStream()` shouldn’t burn a stream if it has some debt (prady)

`burnStream()` only burns a stream if it is inactive and `redeemables[_id] == 0`, but it doesn’t check if the payer has to pay debt for that stream.

Technical Details

Alice creates a stream for Bob which ended but has some debt accrued. Some time after the stream ends, Alice decides to burn that stream. Later Bob checks that Alice has to pay some debt for stream and calls `repayAllDebt()` to pay the debt for that stream. But now as the stream has been burnt by Alice, Bob cannot redeem the debt from stream, and those amount of funds are locked into that stream.

Impact

Medium.

Recommendation

Do not burn a stream if `debts[_id] > 0`.

- In `burnStream()` add this:

  ```
  +if (redeemables[_id] > 0 || streams[_id].lastPaid > 0 || debts[_id] > 0)
      revert STREAM_ACTIVE_OR_REDEEMABLE();
  ```

- Do not allow `owner` or `payerWhitelists[msg.sender]` to pay debt for a stream that has been burnt. In all repayDebt functions add this:

  ```
  + require(ownerOf(_id) > address(0), "STREAM_ENDED_OR_NOT_STARTED");
  ```

Developer Response

Fixed here and here.

2. Medium - `Payee` funds get locked if stream is resumed after being burned (prady)

If the payer `burns` the stream, and then later `resumes` the stream, then the funds that are redeemebale by the `payee` get locked into the contract, as payee can’t access those funds using any available withdraw function.

Technical Details

`Alice` (payer) creates a stream for `Bob` (payee), and later `Alice` decided to burn the stream as the payout time was over. Now later `Alice` decides to resume the stream to `Bob`, which in turns makes the stream active again. But `Bob` can not accesss the redeemable funds as due to the burn the `ownerOf(_id)` is `address(0)`, and has not been reallocated in `resumeStream()`.

```
function testBurnAndResumeStream() external {
    vm.startPrank(alice);
    // Alice creates a stream for Bob
    llamaPayV2Payer.createStream(
        address(llamaToken),
        bob,
        1e20,
        10000,
        50000
    );
    vm.warp(12000);
    // Alice stops the stream
    llamaPayV2Payer.stopStream(0, false);

    // Alice pays Bob
    llamaPayV2Payer.withdrawAll(0);

    // checking that bob is the owner of stream
    address nftOnwer = llamaPayV2Payer.ownerOf(0);
    assertEq(nftOnwer, bob);

    // Alice burns the stream
    llamaPayV2Payer.burnStream(0);

    // Alice resumes the stream, now Bob can again withdraw after some time
    llamaPayV2Payer.resumeStream(0);
    vm.warp(20000);
    vm.stopPrank();
    // now Bob tries to withdraw from stream, which fails as the owner of nft is
address(0)
    vm.startPrank(bob);
    llamaPayV2Payer.withdrawAll(0);
    vm.stopPrank();
}
```

Impact

Medium.

Recommendation

- Do not allow stream to be resumed if it is burned.
- If Payer is burning the stream, do not call `_burn()` function of ERC721.
- If Payer resumes the stream, reallocate the nftOwner to `payee`.
- Do not allow function interactions if stream is burned.

Developer Response

`_updateStream()` will now check if stream has been burned, therefore forbidding interactions if stream is burned.

Low Findings

1. Low - Llama Pay V2 is incompatible with high decimal tokens (engn33r)

If LlamaPayV2Payer is used with an ERC20 token that has a `decimals()` value greater than 20, `deposit()` will revert.

Technical Details

The key line is:

```
if (tokens[_token].divisor == 0) {
    tokens[_token].divisor = uint208(10**(20 - token.decimals()));
}
```

If `token.decimals()` is greater than 20, then the subtraction `20 - token.decimals()` will underflow and the transaction will revert. This prevents depositing tokens like YAM-V2 into

Llama Pay V2.

Impact

Low. Certain tokens are not compatible with Llama Pay V2.

Recommendation

If this limitation is not acceptable, use a constant multiplier on top of the existing `decimals()` value for each ERC20. For example, use a precision constant of 10**18.

Developer Response

Acknowledged but won't fix. Most tokens that will be used are under 20 decimals.

2. Low - `permitToken` fails in case of tokens that doesn't follow `IERC20Permit` standard (prady)

The `BoringBatchable` uses `IERC20Permit` standard for `permit` but these `tokens` use DAI standard.

Technical Details

`permitToken()` fails in case of DAI.

IERC20Permit:

```
function permit(
    address owner,
    address spender,
    uint256 value,
    uint256 deadline,
    uint8 v,
    bytes32 r,
    bytes32 s
) external;
```

DaiPermit:

```
function permit(
    address holder,
    address spender,
    uint256 nonce,
    uint256 expiry,
    bool allowed,
    uint8 v,
    bytes32 r,
    bytes32 s
) external;
```

Impact

Low.

Recommendation

Consider using `permit2` by Uniswap. Another option is to check whether the token is DAI or not, and use DAI permit standard if it is.

Developer Response

Acknowledged. Will use second option if applicable.

3. Low - Handle funds transferred via `transfer` method (prady)

If someone sends ERC20 to the LlamaPayV2Payer using the ERC20's built-in `transfer()` function, those tokens get stuck in the contract and cannot be retrieved.

Technical Details

Introduce a new variable `X` that tracks the net amount that exist in the contract for each token.

1. Updating LlamaPayV2Payer.sol#L183 with `ERC20(token_).balanceOf(address(this))` solves this by making those extra funds withdraw-able.

```
- uint256 toSend = token.balance / token.divisor;
+ uint256 toSend = (token.balance / token.divisor) +
(ERC20(_token).balanceOf(address(this)) - X);
```

1. Rebalance token balance in `_updateToken` function.

```
function _updateToken(address _token) private {
    Token storage token = tokens[_token];
    token.balance += (ERC20(_token).balanceOf(address(this)) - X);
    ...
}
```

Impact

Low.

Recommendation

Allow owner to withdraw the unused balance of token that was added via `transfer` method. Rebalance the balance in `_updateToken()`.

Developer Response

This solution won't work as most of the time balanceOf will be > token.balance.

4. Low - Missing Input Validation (hasanza)

Lack of input validation can lead to funds or tokens being minted/sent to the zero address, thereby burning said tokens and making them irrecoverable for the user.

Technical Details

None of the external `createStream` functions or the internal one `createStream` perform input validation for the `_to` address. This address is supplied by the user and is the recipient of the stream token. It is not uncommon for users or front-ends/ clients to set 0 as values for input forms. This can lead to a user creating a stream but accidentally burning it in the process as well.

Impact

Low.

Recommendation

Put a zero-address check for the `_to` address and the `_amountPerSec` at the beginning of `_createStream`. The already existing `INVALID_ADDRESS()` error can be used for the address check, and a new error `INVALID_AMOUNT` may be used for the amount check. It is worth noting here that the `INVALID_ADDRESS()` error is not used anywhere in the code at this moment.

```
function _createStream(
    address _token,
    address _to,
    uint208 _amountPerSec,
    uint48 _starts,
    uint48 _ends
) private onlyOwnerAndWhitelisted returns (uint256 id) {
    if (_to == address(0)) revert INVALID_ADDRESS()
    if (_amountPerSec == uint208(0)) revert INVALID_AMOUNT()
    //...
}
```

Developer Response

Added checks.

5. Low - `_updateToken` updates `token.lastUpdate` even if the token doesn't exists (prady)

If the token is not yet added to the `tokens` mapping, `_updateToken` can update the `token.lastUpdate`.

Technical Details

```
function _updateToken(address _token) private {
    Token storage token = tokens[_token];

    // @audit add this check
    require(token.divisor > 0, "NOT_ADDED");
    // rest same
}
```

Impact

Low.

Recommendation

Check whether the token has been added to the mapping or not `(token.divisor > 0)`.

Developer Response

Added check.

6. Low - `BoringBatchable.batch()` is `payable` (blockdev)

`BoringBatchable.batch()` is `payable`, while no other function in Llamapay-v2 is `payable`. If `revertOnFail` is set to `false`, it can lead to locked Ether in the contract.

Technical Details

BoringBatchable.sol#L46

Impact

Low. `payable` function can lead to locked Ether in the contract.

Recommendation

Remove `payable` keyword from `BoringBatchable.batch()`.

Developer Response

Acknowledged, will keep original version of BoringBatchable.

7. Low - `modifyStream()` should revert for inactive stream (blockdev)

If `modifyStream()` is used to extend a stream that is inactive, or becomes inactive after `_updateStream()` (in other words, `lastPaid==0`), then no amount is streamed towards the modified stream.

Technical Details

For example, take a stream `s` with id `id` with this configuration:

- `s.lastPaid < s.starts`
- `lastUpdate > s.ends`
- `s.amountPerSec == _oldAmountPerSec`

Now, `modifyStream(id, _newAmountPerSec, _newEnd)` is called for this stream with `_newEnd` in future. After `_updateStream()`, `lastPaid` becomes `0`, and now even though the stream's end is in future, it is not streamed any funds as the stream remains inactive.

Impact

Low. The payer may think that extending a stream into future will result in streaming token for the extra duration, but that's not true in every case.

Recommendation

Revert `modifyStream()` if `lastPaid == 0` once `_updateStream()` is called.

Developer Response

Acknowledged but won't fix as it is expected behavior to allow for payer to modify stream before resuming stream.

8. Low - `withdrawable()` can revert (blockdev, pandadefi)

`withdrawable()` can revert in a few cases.

Technical Details

Cases where `withdrawable()` can revert:

- When no token is deposited in `LlamaPayV2Payer`, and a stream is created.
  - reverts due to division by `0` as `token.divisor` is `0`.
- When execution reaches in this `else` block, and `stream.starts > lastUpdate`.
  - reverts due to arithmetic underflow.
  - This implicitly assumes that `lastUpdate` lies between `starts` and `ends`.

Impact

Low. Caller needs to handle revert in these cases.

Recommendation

Update `withdrawable()` function to handle these cases and return meaningful values.

Developer Response

Using pandadefi's refactor.

9. Low - Inaccuracies in `withdrawable()` (blockdev, PraneshASP)

1.) Debt is calculated incorrectly for the case `block.timestamp >= stream.ends`:

```
debt = (stream.ends * lastUpdate) * stream.amountPerSec;
```

2.) Use `token.lastUpdate` instead of `lastUpdate` to calculate the total amount of streamed tokens.

```
streamed = (block.timestamp - lastUpdate) * token.totalPaidPerSec;
```

Technical Details

LlamaPayV2Payer.sol#L650

LlamaPayV2Payer.sol#L603

Impact

Low. Will result in incorrect calculations.

Recommendation

Apply these diff:

```
-debt = (stream.ends * lastUpdate) * stream.amountPerSec;
+debt = (stream.ends - lastUpdate) * stream.amountPerSec;
```

```
-stream = (block.timestamp - lastUpdate) * token.totalPaidPerSec;
+stream = (block.timestamp - token.lastUpdate) * token.totalPaidPerSec;
```

Developer Response

Using pandadefi's refactor.

10. Low - `streams` can be resumed even after `burn` (prady)

Since `burnStream` doesn't frees up storage related to that particular stream, it is possible to resume burnt stream.

Technical Details

```
function testBurnAndResumeStream() external {
    vm.startPrank(alice);
    llamaPayV2Payer.createStream(
        address(llamaToken),
        bob,
        1e20,
        10000,
        50000
    );
    vm.warp(12000);
    llamaPayV2Payer.stopStream(0, false);
    llamaPayV2Payer.withdrawAll(0);
    // burning the stream
    llamaPayV2Payer.burnStream(0);
    // resuming the stream
    llamaPayV2Payer.resumeStream(0);
    vm.stopPrank();
}
```

```
function burnStream(uint256 _id) external {
    if (
        msg.sender != owner &&
        payerWhitelists[msg.sender] != 1 &&
        msg.sender != ownerOf(_id)
    ) revert NOT_OWNER_OR_WHITELISTED();
    /// Prevents somebody from burning an active stream or a stream with balance in
it
    if (redeemables[_id] > 0 || streams[_id].lastPaid > 0)
        revert STREAM_ACTIVE_OR_REDEEMABLE();

    _burn(_id);
    emit BurnStream(_id);
}
```

As above code shows that data related to that particular burnt `_id` still stays in contract, it can be again resumed.

Impact

Low.

Recommendation

Apply this:

```
function burnStream(uint256 _id) external {
    if (
        msg.sender != owner &&
        payerWhitelists[msg.sender] != 1 &&
        msg.sender != ownerOf(_id)
    ) revert NOT_OWNER_OR_WHITELISTED();
    /// Prevents somebody from burning an active stream or a stream with balance in
it
    if (redeemables[_id] > 0 || streams[_id].lastPaid > 0)
        revert STREAM_ACTIVE_OR_REDEEMABLE();

    // free up storage
+   delete streams[_id];
+   delete debts[_id];
+   delete redeemables[_id];
+   delete redirects[_id];

    _burn(_id);
    emit BurnStream(_id);
}
```

Developer Response

`_updateStream()` will now check if stream has been burned, therefore forbidding interactions if stream is burned. Freed up storage.

11. Low - `modifyStream` can create a new `stream` if it doesn't exist (prady)

Technical Details

Since `modifyStream` doesn't check whether the stream is present or not, it creates a new stream if not present.

```
function testModifyStream() external {
    vm.prank(alice);
    llamaPayV2Payer.modifyStream(0, 2e20, 20000);
    (uint208 amountPerSec, , , , uint256 ends) = llamaPayV2Payer.streams(
        0
    );
    assertEq(amountPerSec, 2e20);
    assertEq(ends, 20000);
}
```

Impact

Low.

Recommendation

Apply this:

```
function modifyStream(
    uint256 _id,
    uint208 _newAmountPerSec,
    uint48 _newEnd
) external onlyOwnerAndWhitelisted {
    _updateStream(_id);
    Stream storage stream = streams[_id];
+   require(stream.token > address(0), "STREAM_NOT_AVAILABLE");
    // ...
}
```

Developer Response

`_updateStream()` will now check if stream does not exist, therefore forbidding interactions if stream does not exist.

12. Low - `payerWhitelists` addresses should have access to all withdraw operations (engn33r)

`withdrawPayer()` and `withdrawPayerAll()` permit an address in `payerWhitelists` to call them. The other withdraw functions `withdraw()`, `withdrawAll()`, `withdrawWithRedirect()`, and `withdrawAllWithRedirect()` are more permissive but do not allow `payerWhitelists` addresses to call them.

Technical Details

`withdraw()`, `withdrawAll()`, `withdrawWithRedirect()`, and `withdrawAllWithRedirect()` do not permit an address in `payerWhitelists` to call them. Most likely this was an oversight, because the addresses in `payerWhitelists` should have similar permissions as the contract owner. The payer whitelist addresses could take funds directly from the contract already, so they should be able to assist with transfering the funds to the intended recipient.

Impact

Low. Some withdraw operations should likely permit `payerWhitelists` addresses but don't.

Recommendation

Modify `withdraw()`, `withdrawAll()`, `withdrawWithRedirect()`, and `withdrawAllWithRedirect()` with:

```
if (
          msg.sender != nftOwner &&
          msg.sender != owner &&
+         payerWhitelists[msg.sender] != 1 &&
          streamWhitelists[_id][msg.sender] != 1
) revert NOT_OWNER_OR_WHITELISTED();
```

Developer Response

Fixed using new internal function.

13. Low - Rogue `payerWhitelists` address can steal value (engn33r)

Addresses added to the mapping `payerWhitelists` are considered trusted addresses that can act on behalf of the contract owner. However, in the event that one of these addresses goes rogue and turns malicious (or is compromised), the address in the payer whitelist can backrun a deposit to create a new stream and take value from the contract.

Technical Details

Any function protected by the `onlyOwnerAndWhitelisted` modifier treats the owner and whitelisted payer addresses as the same privilege level. This includes the internal `_createStream()` function that can redirect value to a recipient. If an owner deposits value into the Llamapay contract, an address in the `payerWhitelists` mapping can:

1. Backrun the deposit to create a stream where the recipient is an address that is controlled by the whitelisted address. The stream end date can be in the past so that it

is possible to instantly withdraw.
2. After the stream is created, the recipient can withdraw the funds.

The end result is the same as calling `withdrawPayer()` directly, but this is a less expected approach.

Impact

Low. Owner may not realize that the addresses added to `payerWhitelists` can take tokens out of the contract for themselves in multiple ways.

Recommendation

Clearly warn the owner when an address is added to `payerWhitelists` that the address should be trusted as highly as the owner because it could withdraw value from the contract.

Developer Response

Will clearly document and warn owner.

Gas Savings Findings

1. Gas - Use cheaper comparison (spalen)

It is cheaper to use `==` instead of `!=`.

Technical Details

At L251 and L280 logic can be switched to save some gas.

Impact

Gas savings.

Recommendation

Change the logic at L251 and L280.

```
- if (redirect != address(0)) {
-     to = redirect;
- } else {
-     to = nftOwner;
- }
```

```
+ if (redirect == address(0)) {
+     to = nftOwner;
+ } else {
+     to = redirect;
+ }
```

2. Gas - Incorrect function visibility (pandadefi, engn33r)

`tokenURI()` and `withdrawable()` visibility is set to public without any internal call to these two functions. Visibility should be set to external.

Technical Details

`tokenURI()` and `withdrawable()` should be external functions in LlamaPayV2Payer. In LlamaPayV2Factory, `calculateLlamaPayAddress()` should be an external function.

Impact

Gas savings.

Recommendation

Use `external` instead of `public`.

3. Gas - `owner` should be a immutable variable (pandadefi)

The contract owner will never change, to reflect that, `owner` should be an immutable variable.

Technical Details

`owner` is never updated; it should be defined as an immutable to save gas.

Impact

Gas savings.

Recommendation

Add `immutable` to the owner variable definition.

4. Gas - Return stream value after updating (spalen)

Almost after each function call `_updateStream(uint256 _id)`, the same stream is loading again for additional changes. Additional loading of the stream can be skipped by returning the stream from the function `_updateStream(uint256 _id)`.

Technical Details

If the `_updateStream(uint256 _id)` is changed to return the update stream, small gas optimization is possible. At L201, L222, L243, L272, L400, L425, L446, L222 a stream value could be returned instead of loading again stream in the next line.

Impact

Gas savings.

Recommendation

Change stream update function to return stream with updated value:

```
function _updateStream(uint256 _id) private returns (Stream storage)
```

Use returned value stream from update function instead of loading it again:

```
- _updateStream(_id);
- Stream storage stream = streams[_id];
+ Stream storage stream = _updateStream(_id);
```

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

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

Technical Details

Unchecked can be applied to this line and other similar locations. The subtraction will not overflow because `lastUpdate` is zero. `block.timestamp - lastUpdate` has a max value of `type(uint48).max` and token.totalPaidPerSec has a max value of `type(uint208).max`, the maximum product is `type(uint256).max` which doesn't overflow.

```
- uint256 streamed = (block.timestamp - lastUpdate) * token.totalPaidPerSec;
+ unchecked { uint256 streamed = (block.timestamp - lastUpdate) *
token.totalPaidPerSec; }
```

Impact

Gas savings.

Recommendation

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

6. Gas - Do not load whole struct if not required (prady)

Technical Details

Following function loads `Stream` struct, but loading `Stream.token` works.

- `withdrawAllWithRedirect`
- `withdrawWithRedirect`
- `withdrawAll`
- `withdraw`

```
- Stream storage stream = streams[_id];
+ address token = streams[_id].token;
```

Impact

Gas savings.

Recommendation

Load struct in `memory` rather than in `storage`, wherever possible.

7. Gas - Pass struct as a params to `_createStream()` (prady)

Technical Details

Update `_createStream` and `createStream` this way:

```
function createStream(
    address _token,
    address _to,
    uint208 _amountPerSec,
    uint48 _starts,
    uint48 _ends
) external {
    uint256 id = _createStream(
        Stream({
            amountPerSec: _amountPerSec,
            token: _token,
            lastPaid: 0,
            starts: _starts,
            ends: _ends
        }),
        _to
    );
    emit CreateStream(id, _token, _to, _amountPerSec, _starts, _ends);
}

function _createStream(Stream memory stream, address _to)
    private
    onlyOwnerAndWhitelisted
    returns (uint256 id)
{
    if (stream.starts >= stream.ends) revert INVALID_TIME();
    _updateToken(stream.token);
    Token storage token = tokens[stream.token];

    if (block.timestamp > token.lastUpdate) revert PAYER_IN_DEBT();

    _safeMint(_to, id = nextTokenId);

    /// calculate owed if stream already ended on creation
    uint256 owed;
    if (block.timestamp > stream.ends) {
        owed = (stream.ends - stream.starts) * stream.amountPerSec;
    } else if (block.timestamp > stream.starts) {
        /// calculated owed if start is before block.timestamp
        owed = (block.timestamp - stream.starts) * stream.amountPerSec;
        tokens[stream.token].totalPaidPerSec += stream.amountPerSec;
        stream.lastPaid = uint48(block.timestamp);
    } else if (stream.starts >= block.timestamp) {
        /// If started at timestamp or starts in the future
        tokens[stream.token].totalPaidPerSec += stream.amountPerSec;
```

```
            stream.lastPaid = uint48(block.timestamp);
        }

        unchecked {
            /// If can pay owed then directly send it to payee
            if (token.balance >= owed) {
                tokens[stream.token].balance -= owed;
                redeemables[id] = owed;
            } else {
                /// If cannot pay debt, then add to debt and send entire balance to payee
                uint256 balance = token.balance;
                tokens[stream.token].balance = 0;
                debts[id] = owed - balance;
                redeemables[id] = balance;
            }
            nextTokenId++;
            streams[id] = stream;
        }
    }
```

Impact

Gas savings. (168576 -> 168392)

Recommendation

Use of struct as parameter to function.

8. Gas - `if` condition can be optimized in `withdrawable()` (blockdev)

To calculate debt and withdrawable amount, `withdrawable()` function goes through a series of if/else conditions. One such condition can be optimized:

```
stream.lastPaid >= stream.starts && stream.ends > block.timestamp
```

By the time, execution reaches, LlamaPayV2Payer.sol#L662, `stream.ends > block.timestamp` is always true due to the previous `else if` condition:

```
block.timestamp >= stream.ends
```

Technical Details

LlamaPayV2Payer.sol#L662

Impact

Gas savings.

Recommendation

Apply this diff:

```
else if (
-   stream.lastPaid >= stream.starts && stream.ends > block.timestamp
+   stream.lastPaid >= stream.starts
) {
```

9. Gas - Use `calldata` instead of `memory` if function param remains unchanged (prady)

Technical Details

Following function uses `memory` to pass string param, `calldata` can be used as the string remains unchanged during function execution.

- `createStreamWithReason`
- `createStreamWithheldWithReason`

```
- string memory _reason
+ string calldata _reason
```

Impact

Gas savings.

Recommendation

`calldata` is less expensive than `memory`.

10. Gas - Consider and compare UX when streams are ERC1155 tokens

(blockdev)

ERC1155 can have multiple owners of the same token ID, because a token need not be unique. However, it can always be restricted that way in ERC1155. It's more gas efficient than ERC721 since it doesn't have to maintain `ownerOf` mapping. So gas costs savings are quite substantial.

Technical Details

ERC1155 doesn't have the same interface as ERC721, so if doing this replacement, a code refactor may be needed. Another thing to consider would be to use ERC1155D (see this repo and announcement) which provides compatibility with ERC721 interface, however, it's not audited. So if you consider ERC1155D, you can either self-review it, take inspiration or get it audited.

Impact

Gas savings.

Recommendation

Consider using ERC1155 instead of ERC721, but be aware of the needed UX change for vanilla ERC1155, and the associated risks with ERC1155D.

Informational Findings

1. Informational - Missing Indexed Event Parameters (hasanza)

Event parameters should have at least one indexed parameter. This is so that the graphing and fetching apps are able to filter through event data easily.

Technical Details

All the events are missing any indexed parameters.

Impact

Informational.

Recommendation

Make event parameters such as address and/or tokenIds indexed, in all the events.

2. Informational - External createStream functions should return the streamId (hasanza)

Functions create streams and store the returned streamId in a local variable, but do not return it.

Technical Details

Right now, the createStream functions namely `createStream`, `createStreamWithReason`, `createStreamWithheld` and `createStreamWithheldWithReason`, do not return the id of the newly created stream. Returning the id of the stream upon creation could result in better integration with other protocols and a better UX, where the user immediately knows the id of his stream once it is created.

Impact

Informational.

Recommendation

The functions: `createStream`, `createStreamWithReason`, `createStreamWithheld` and `createStreamWithheldWithReason` should return the id of the newly created stream.

3. Informational - Update BoringBatchable to latest version (engn33r)

The BoringBatchable library in Llama Pay V2 is missing the latest commit from the Boring Solidity repository.

Technical Details

The difference between BoringBatchable in Llama Pay V2 and the main Boring Solidity repo is this PR https://github.com/boringcrypto/BoringSolidity/pull/16/files.

Impact

Informational.

Recommendation

Consider using the latest release of any third party libraries as a security best practice.

4. Informational - Add tests for weird ERC20 tokens (engn33r)

If Llama Pay V2 is intended to be used for a wide variety of DeFi tokens, consider adding tests for weird ERC20 tokens to avoid unexpected scenarios.

Technical Details

The weird-erc20 repo has many sample contracts that mimic unusual ERC20 token behavior. Consider writing tests to check the compatibility of LlamaPay V2 with all of these contracts.

Impact

Informational.

Recommendation

Add testing for a variety of weird ERC20 tokens.

5. Informational - Sanitisation to avoid false event emission (prady)

- `addPayerWhitelist` and `removePayerWhitelist` doesn't check if `payerWhitelists[_toAdd]` is present or not.
- Same applies to `addRedirectStream`, `removeRedirectStream`, `addStreamWhitelist`, `removeStreamWhitelist`.

Technical Details

```
function addPayerWhitelist(address _toAdd) external onlyOwner {
    require(payerWhitelists[_toAdd] == 0, "ALREADY_WHITELISTED");
    payerWhitelists[_toAdd] = 1;
    emit AddPayerWhitelist(_toAdd);
}

function removePayerWhitelist(address _toRemove) external onlyOwner {
    require(payerWhitelists[_toAdd] == 1, "NOT_WHITELISTED");
    payerWhitelists[_toRemove] = 0;
    emit RemovePayerWhitelist(_toRemove);
}
```

Impact

Informational.

Recommendation

Using checks can reduce false events.

6. Informational - `tokenURI()` should revert for non-existent tokens (blockdev)

As per ERC721 standard, `tokenURI()` needs to revert if `tokenId` doesn't exist. The current code returns empty string for all inputs.

Technical Details

LlamaPayV2Payer.sol#L135

Impact

Informational.

Recommendation

Revert if `_ownerOf[tokenid] == address(0)`.

7. 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:

- `cancelDebt`
- `repayAllDebt`
- `repayDebt`
- `updateStream`

Impact

Informational.

Recommendation

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

8. Informational - Lack of zero check on functions (prady)

Certain setter functions fails to validate incoming argument, so owner can accidentally set zero address as whitelisted address.

Technical Details

For e.x.,

```
function addPayerWhitelist(address _toAdd) external onlyOwner {
    payerWhitelists[_toAdd] = 1;
    emit AddPayerWhitelist(_toAdd);
}
```

Similarly following function doesn't check for valid input address.

- `addPayerWhitelist`
- `removePayerWhitelist`
- `addRedirectStream`
- `addStreamWhitelist`
- `removeStreamWhitelist`

Impact

Informational.

Recommendation

Add zero-value checks to the functions mentioned above to ensure owner cannot accidentally set incorrect values.

9. Informational - `onlyOwnerAndWhitelisted` should be renamed `onlyOwnerOrWhitelisted` (blockdev)

`onlyOwnerAndWhitelisted` allows calls from the owner or from a whitelisted address. Hence, a more suitable name for this modifier is `onlyOwnerOrWhitelisted`.

Technical Details

LlamaPayV2Payer.sol#L129

Impact

Informational.

Recommendation

Rename `onlyOwnerAndWhitelisted` to `onlyOwnerOrWhitelisted`.

10. Informational - `unchecked` has no effect (blockdev)

There is no arithmetic done in the highlighted `unchecked` code block below.

Technical Details

LlamaPayV2Payer.sol#L454

Impact

Informational.

Recommendation

Remove `unchecked` as it has no effect.

11. Informational - Use consistent decimals format in functions (prady)

Most of the functions uses native token decimals format for input params, and then converts it to 20 decimals format. But `repayDebt()` uses 20 decimals format for input params.

Technical Details

```
-    /// @param _amount amount to repay (20 decimals)
+    /// @param _amount  amount to repay (native token decimals)
     function repayDebt(uint256 _id, uint256 _amount) external {
         if (
             msg.sender != owner &&
             payerWhitelists[msg.sender] != 1 &&
             msg.sender != ownerOf(_id)
         ) revert NOT_OWNER_OR_WHITELISTED();
         address token = streams[_id].token;

+         /// convert _amount into 20 decimals format
+         _amount = _amount * tokens[token].divisor;

         /// Update token to update balances
         _updateToken(token);
         /// Reverts if debt cannot be paid
         tokens[token].balance -= _amount;
         /// Reverts if paying too much debt
         debts[_id] -= _amount;
         /// Add to redeemable to payee
```

```
        redeemables[_id] += _amount;
    }
```

Impact

Informational.

Recommendation

Use consistent input params across all functions, to aviod confusions.

12. Informational - Use function to reduce duplicacy of code (prady)

Use `function` to reduce duplicacy of code.

Technical Details

```
if (
    msg.sender != nftOwner &&
    msg.sender != owner &&
    streamWhitelists[_id][msg.sender] != 1
) revert NOT_OWNER_OR_WHITELISTED();
```

Following code snippet can be converted into a function.

```
function ownerOrNftOwnerOrWhitelisted(uint _id, address _nftOwner) internal {
    if (
        msg.sender != nftOwner &&
        msg.sender != owner &&
        streamWhitelists[_id][msg.sender] != 1
    ) revert NOT_OWNER_OR_WHITELISTED();
}
```

Applicable to following functions:

- `withdraw`
- `withdrawAll`
- `withdrawWithRedirect`

- `withdrawAllWithRedirect`

Impact

Informational.

Recommendation

Reduce duplicacy of code.

13. Informational - Redirect recipient cannot receive funds directly (engn33r)

The `redirects` mapping allows funds to be received by an address other than the owner of the NFT. However, the recipient of the redirect has no permission to directly withdraw the funds from the contract. They must wait for someone else to call `withdrawWithRedirect()` or `withdrawAllWithRedirect()` to receive their funds.

Technical Details

`addRedirectStream()` and `removeRedirectStream()` allow the receiver of the stream's tokens to sent to a different address with `withdrawWithRedirect()` or `withdrawAllWithRedirect()`. `withdrawWithRedirect()` and `withdrawAllWithRedirect()` have the same access controls as other withdraw functions, meaning only the NFT owner, the contract owner, or a stream whitelist address can trigger the transfer of funds to the redirect address. This means the redirect address does not have control over these funds but is instead relying on someone else to deliver the funds. This is a less than ideal experience for the redirect receiver, unless the intent of this feature is that the redirect address and NFT address should have the same person (or entity) owning the two addresses. If that is a hidden intent when using the redirect address, it should be clearly documented.

Impact

Informational.

Recommendation

If the redirect recipient is supposed to be able to retrieve funds directly, modify `withdrawWithRedirect()` and `withdrawAllWithRedirect()` with:

```
if (
            msg.sender != nftOwner &&
            msg.sender != owner &&
+           msg.sender != redirects[_id] &&
            streamWhitelists[_id][msg.sender] != 1
) revert NOT_OWNER_OR_WHITELISTED();
```

Note that if the redirect recipient does have this control over retrieving funds, selling a Llamapay V2 NFT on OpenSea may result in a surprise. A buyer may not realize a redirect address is set, which the seller could use to steal funds after selling the NFT. One way to mitigate this is overriding the default `_transfer()` to delete the redirect address for all ids.

14. Informational - Typos (engn33r)

There are some typos in comments.

Technical Details

- strean -> stream
- redeeemable -> redeemable

Impact

Informational.

Recommendation

Fix typos.

15. Informational - Stream owner should be able to call `updateStream()` (blockdev)

`updateStream()` can only be called by the owner or by an address in payer whitelist. However, it can be made an open function that anyone can call. If there is some legal or tax related concern, the permission can be extended to the stream owner.

Technical Details

LlamaPayV2Payer.sol#L478

Impact

Informational.

Recommendation

Consider removing `onlyOwnerAndWhitelisted` modifier from `updateStream()` function, or extend this permission to the stream owner.

Final remarks

blockdev

Overall, the code is easy to understand and well designed. Adding documentation and explaining the difference in behaviour of the protocol depending on the relative position of parameters, like current time, stream start and end, will help anyone understand the code easily. Documenting this may also help simplify the code as done in this refactor PR.

engn33r

LlamaPay V2 is building on the solid foundation of LlamaPay V1. Some of the design differences still have some rough edges to iron out, but the overall approach is reasonable. One of the difficult elements in this code is validating that the payment stream can only follow a specific call flow where unintended call flows should not be possible. This requires extensive testing for the many permutations and possible edge cases. Visualizing and documentating how the call flow should look could help create positive and negative tests to validate the code matches the intended flow.

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.