Llama Security Review

Auditors

Noa Marconi, Lead Security Researcher

Xmxanuel, Security Researcher

M4rio.eth, Security Researcher

Report prepared by: Pablo Misirov

August 20, 2023

Contents

1 About Spearbit **2**

2 Introduction **2**

3 Risk classification **2**
3.1 Impact . . . 2
3.2 Likelihood . . . 2
3.3 Action required for severity levels . . . 2

4 Executive Summary **3**

5 Findings **4**
5.1 Medium Risk . . . 4
5.1.1 `LlamaAbsolutePeerReview.validateActionCreation` uses incorrect quantity timestamp for the creator quantity . . . 4
5.2 Low Risk . . . 4
5.2.1 `isActionExpired` can return incorrect boolean result . . . 4
5.2.2 `salt` and `name` reuse for strategies/accounts creation . . . 5
5.2.3 `Core._newCastCount` is not overflow resistant in all cases . . . 5
5.2.4 Improve sanity checks within the `LlamaFactory` . . . 6
5.3 Informational . . . 6
5.3.1 `LlamaAccount` cannot use its own funds in a `call` with `data` . . . 6
5.3.2 forceApprovalRoles are not based on snapshots . . . 7
5.3.3 Docs are out of sync with `LlamaCore.authorizeStrategy` implementation . . . 7
5.3.4 It is not possible to roll back to previous version after updating to a new `ILlamaPolicyMetadata` config . . . 7
5.3.5 Use inheritance for code reuse among strategies . . . 8
5.3.6 `PolicyholderCheckpoints.push` and `PolicyholderCheckpoints._insert` have opposite order of args . . . 8
5.3.7 Checkpoints `push` interface can require desired types . . . 9
5.3.8 a disapproval voter might need to first queue an action to vote against it . . . 9
5.3.9 approvalEndTime should return 0 for non existing actions . . . 10
5.3.10 no `LlamaCore.cancelActionBySig` function exists . . . 10
5.3.11 Some function parameters within the `LlamaLens` are not explicit enough . . . 10
5.3.12 Script for recovering assets stuck in the executor. . . . 11
5.3.13 Unnecessary `_infoHash` implementation . . . 11

1 About Spearbit

Spearbit is a decentralized network of expert security engineers offering reviews and other security related services to Web3 projects with the goal of creating a stronger ecosystem. Our network has experience on every part of the blockchain technology stack, including but not limited to protocol design, smart contracts and the Solidity compiler. Spearbit brings in untapped security talent by enabling expert freelance auditors seeking flexibility to work on interesting projects together.

Learn more about us at spearbit.com

2 Introduction

Llama is an onchain governance and access control framework for smart contracts. Using Llama, protocol teams can deploy self-governed instances that define granular roles, permissions, and strategies for executing actions.

Disclaimer: This security review does not guarantee against a hack. It is a snapshot in time of llama according to the specific commit. Any modifications to the code will require a new security review.

3 Risk classification

Severity level	Impact: High	Impact: Medium	Impact: Low
Likelihood: high	Critical	High	Medium
Likelihood: medium	High	Medium	Low
Likelihood: low	Medium	Low	Low

3.1 Impact

- High - leads to a loss of a significant portion (>10%) of assets in the protocol, or significant harm to a majority of users.
- Medium - global losses <10% or losses to only a subset of users, but still unacceptable.
- Low - losses will be annoying but bearable--applies to things like griefing attacks that can be easily repaired or even gas inefficiencies.

3.2 Likelihood

- High - almost certain to happen, easy to perform, or not easy but highly incentivized
- Medium - only conditionally possible or incentivized, but still relatively likely
- Low - requires stars to align, or little-to-no incentive

3.3 Action required for severity levels

- Critical - Must fix as soon as possible (if already deployed)
- High - Must fix (before deployment if not already deployed)
- Medium - Should fix
- Low - Could fix

4 Executive Summary

Over the course of 5 days in total, llama engaged with Spearbit to review commit 28d6e94e1. In this period of time a total of **18** issues were found.

Summary

Project Name	llama
Commit	28d6e9...618a
Type of Project	Governance, DeFi and NFT
Audit Timeline	July 24 to July 28

Issues Found

Severity	Count	Fixed	Acknowledged
Critical Risk	0	0	0
High Risk	0	0	0
Medium Risk	1	1	0
Low Risk	4	2	2
Gas Optimizations	0	0	0
Informational	13	8	5
Total	**18**	**11**	**7**

5 Findings

5.1 Medium Risk

5.1.1 `LlamaAbsolutePeerReview.validateActionCreation` uses incorrect quantity timestamp for the creator quantity

Severity: Medium Risk

Context: LlamaAbsolutePeerReview.sol#L46

Description: `Llama` uses various strategies to implement different voting rules for the `approval` or `disapproval` of actions. The `LlamaAbsolutePeerReview` strategy doesn't allow the creator of an action to participate in the voting.To determine if other `policyholders` can reach the required `quantity` to approve/disapprove, the creator's quantity needs to be subtracted from the `totalQuantity` of all policyholders.

The voting power is calculated based on a snapshot taken at `actionCreatedTimestamp - 1`. However, the quantity of the creator in the `validateActionCreation` function uses the latest timestamp. This will lead to an incorrect validation if the quantity of the `creator` got changed in the same `block.timestamp` before the create action.

Recommendation:

```
- uint256 actionCreatorApprovalRoleQty = llamaPolicy.getQuantity(actionInfo.creator, approvalRole);
+ uint256 actionCreatorApprovalRoleQty = llamaPolicy.getPastQuantity(actionInfo.creator, approvalRole,
↪  block.timestamp - 1);
if (minApprovals > approvalPolicySupply - actionCreatorApprovalRoleQty) revert
↪  InsufficientApprovalQuantity();
- uint256 actionCreatorDisapprovalRoleQty = llamaPolicy.getQuantity(actionInfo.creator,
↪  disapprovalRole);
+ uint256 actionCreatorDisapprovalRoleQty = llamaPolicy.getPastQuantity(actionInfo.creator,
↪  disapprovalRole, block.timestamp - 1);
```

Llama: Completed in commit 9728b1 ,PR 451.

Spearbit: Resolved.

5.2 Low Risk

5.2.1 `isActionExpired` can return incorrect boolean result

Severity: Low Risk

Context: LlamaRelativeStrategyBase#L263

Description: In Llama both the relative and absolute strategies have a view function `isActionExpired` to determine if an action is expired. An action is expired if it has passed the `experiationTimestamp` and it got never executed after the approval.

```
function isActionExpired(ActionInfo calldata actionInfo) external view virtual returns (bool) {
  Action memory action = llamaCore.getAction(actionInfo.id);
  return block.timestamp > action.minExecutionTime + expirationPeriod;
}
```

However, the function will return `true` if the `action.minExecutionTime == 0` for any reasonable `expirationPeriod`. The `minExecutionTime` time is only set in the `queueAction` step.

This implies the function could incorrectly return `true` for all states prior to queuing. The Llama contracts currently only use this function if the action is already `queued`. Therefore, there are no direct implications at present.

Recommendation:

```
  function isActionExpired(ActionInfo calldata actionInfo) external view virtual returns (bool) {
+  if (action.minExecutionTime == 0) return false;
    Action memory action = llamaCore.getAction(actionInfo.id);
    return block.timestamp > action.minExecutionTime + expirationPeriod;
  }
```

Llama: We acknowledge this issue, but chose not to take action. Strategy view functions are not meant to be called by any onchain or offchain service other than the core. We will make sure to document this.

Spearbit: Acknowledged.

5.2.2 `salt` and `name` reuse for strategies/accounts creation

Severity: Low Risk

Context: LlamaCore.sol#L273-L277 | LlamaCore.sol#L744

Description: The `Strategy` and `Account` deployment flow is using `Clones.cloneDeterministic` approach, where they rely on the `CREATE2` opcode to deploy the addresses. This approach requires a `salt` to generate uniquely generate the addresses.

- Scenario 1: The `LlamaCore` instance is using only the config parameters of the strategy for the salt. This can impose a weak salt if someone wants to deploy the same strategy implementation, with the same config parameters, as a governance action.

Deactivating and old strategy and deploying a new strategy with previously used config can be useful for allowing a clean reset of the deactivated strategy. The same strategy parameters could be redeployed with a different `salt` to receive a clean new strategy address.

The `permissionId` in Llama includes the strategy address, therefore permissions with old strategy address would be still inactive.

- Scenario 2: Account deployments use a string variable `name` which can be used to deploy different accounts with the same logic and different name. Accounts with unique names is desirable. A name collision between accounts can occur if a new `LlamaAccount` implementation is used.

Recommendation:

- Scenario 1: For the strategy deployments, consider adding a unique parameter, like a `name` that can be used to differentiate between 2 strategies deployments.
- Scenario 2: Consider restricting Account names within each `LlamaCore` so that reuse of names across `LlamaAccount` implementations is not possible.

Llama: We acknowledge this issue, but choose not to resolve it. We're ok with requiring instances to reset `permissionIds` if that's desirable and reuse of account names if that's also desirable.

Spearbit: Acknowledged.

5.2.3 `Core._newCastCount` is not overflow resistant in all cases

Severity: Low Risk

Context: LlamaCore.sol#L696

Description: The helper function `_newCastCount` is used to calculate the new total approval or disapproval when a policyholder votes.

In case `type(uint96).max` is reached it should just return the maximum value instead of adding the new approval.

In the current implementation, the function would revert if `currentCount + quantity` > `type(uint96).max`.

Theoretically, with the current `policy._setRoleHolder` implementation, it should be prevented, that the sum of the individual policyholder `totalQuantity` can be higher than `type(uint96).max`.

However, we would still recommend not relying on the correct implementation of another contract if it is not needed and adding the protection inside the `_newCastCount` function.

Recommendation: If the function should be completely overflow resistant and always return `type(uint96).max` instead of reverting with an `overflow`

```
function _newCastCount(uint96 currentCount, uint96 quantity) internal pure returns (uint96) {
  if (uint104(currentCount) + uint104(quantity) >= type(uint96).max) return type(uint96).max;
  return currentCount + quantity;
}
```

Llama: Completed in commit 68ad56.

Spearbit: Fixed.

5.2.4 Improve sanity checks within the `LlamaFactory`

Severity: Low Risk

Context: LlamaFactory.sol#L58-L60

Description: Currently, the `LlamaFactory` uses a deploy function in a permissionless manner to deploy `Llama` instances. This function does a check that the first role is the bootstrap role, which is a role needed to assign more roles after the deploy.

This function should contain more checks on the initial parameters, as these can make the instance being a faulty instance after deploy.

For example, having the `instanceConfig.strategyLogic` mistakenly as `address(0)` will not revert the call, therefore, resulting in a faulty instance being deployed.

Recommendation: Consider either adding more checks within this function for the initial parameters, especially that this is a permissionless function, or consider adding more checks within the deploy script, the script that the `Llama` team will recommend to the users to use to deploy their instances.

Llama: Completed in commit 31a797. The strategies, accounts, and policy metadata initialize functions now have returns (bool). By doing that the solidity checks the return size which should be 32 bytes for a bool. If a faulty implementation (or address 0) is used, it will revert due to the aforementioned check.

Spearbit: Fixed.

5.3 Informational

5.3.1 `LlamaAccount` cannot use its own funds in a `call` with `data`

Severity: Informational

Context: LlamaAccount.sol#L329

Description: The `LlamaAccount` is restricted in the varieties of external calls it can make. Calls were value is sent (e.g. an ether dex swap or `WETH9.deposit`) are not supported due to `msg.value` coming from the sender who calls `LlamaCore.executeAction` rather than the balance of ether held by `LlamaAccount`.

Recommendation: `LlamaAccount` offers means of sending ether externally through `LlamaAccount.transferNativeToken` meaning no funds are trapped. This finding is simply UX related.

Consider where allowing for use cases such as ether dex swaps, weth deposits, etc. are desirable for `LlamaAccount` to support. Modifying to support the use of its own balance would enable use cases commonly supported by smart contract based accounts.

Llama: Completed in commit 70bb95, PR 468.

Spearbit: Acknowledged.

5.3.2 forceApprovalRoles are not based on snapshots

Severity: Informational

Context: LlamaRelativeUniqueHolderQuorum.sol#L28

Description: If a new action is created in`Llama`, the `creationTimestamp - 1` is used as a snapshot to determine the eligible policyholders who can participate in votings. A policyholder needs to hold a `role` and a `quantity` at `creationTimestamp - 1`. The quantity defines the voting power of a policyholder.

Each strategy in `Llama` has a special role called `forceApprovalRoles`. A policyholder with the `forceApprovalRoles` can approve any action for this strategy without the approval of any other policyholder.

Currently, the `forceApprovalRoles` are not following the `creationTimestamp - 1` snapshot. It is only relevant that the policyholder holds the `forceApprovalRoles` at `block.timestamp`.

All current existing strategies require defining the `forceApprovalRoles` and `forceDisapprovalRoles` in the initialization step of the strategy. A later modification is not possible in the existing strategy.

Recommendation: For future strategies which might allow modifying the `forceApprovalRoles` and `forceDisapprovalRoles` roles, it should be documented that these roles are not following the `creationTimestamp - 1` snapshot.

Llama: Completed in commit 3c2b5c.

Spearbit: Acknowledged.

5.3.3 Docs are out of sync with `LlamaCore.authorizeStrategy` implementation

Severity: Informational

Context: LlamaCore.sol#L500

Description: The docs indicate `Strategies cannot be explicitly deleted or unauthorized in the Llama system`, while the new implementation allows for unauthorizing a strategy.

Recommendation: Update documentation to reflect the new capability.

Llama: Completed in commit 2d5b9d, PR 453.

Spearbit: Acknowledged.

5.3.4 It is not possible to roll back to previous version after updating to a new `ILlamaPolicyMetadata` config

Severity: Informational

Context: File.sol#L123

Description: Setting and initializing a Policy's metadata attempts to deploy with `create2`. In the event a previously deployed metadata configuration is desired, there is no simple way to revert back to it.

Recommendation: Document the constraint and include instructions on how to circumvent the issue through a minor bytecode edit (as modified bytecode leads to a new `create2` address).

Llama: We acknowledge, but will not take action on this issue. Teams can always deploy a new `policyMetadataLogic` contract with a minor bytecode edit if they needed to roll back to a previous configuration.

Spearbit: Acknowledged.

5.3.5 Use inheritance for code reuse among strategies

Severity: Informational

Context: LlamaRelativeHolderQuorum.sol, LlamaRelativeQuantityQuorum.sol, LlamaRelativeUniqueHolderQuorum.sol, LlamaAbsoluteStrategyBase.sol

Description: Many of the strategy implementations share code that is not included in the `RelativeBase` strategy. Namely approval / disapproval quantities:

```
function getApprovalQuantityAt(address policyholder, uint8 role, uint256 timestamp)
  external
  view
  virtual
  returns (uint96)
{
  if (role != approvalRole && !forceApprovalRole[role]) return 0;
  uint96 quantity = policy.getPastQuantity(policyholder, role, timestamp);
  return quantity > 0 && forceApprovalRole[role] ? type(uint96).max : quantity;
}
```

Recommendation: For code reuse, relying on the base strategy is recommended.

Additionally, the variations noted in the Strategy Comparison Table, would be good candidates to include as internal functions in a utility lib:

- Policyholder weight: Role quantity
- Policyholder weight: 1
- Supply: Total role quantity
- Supply: Total role holders
- etc.

Llama: We acknowledge but chose to not take action on this issue. We believe we struck the right balance between DRY and introducing unnecessary abstractions. This could change as we develop more strategy logics.

Spearbit: Acknowledged.

5.3.6 `PolicyholderCheckpoints.push` and `PolicyholderCheckpoints._insert` have opposite order of args

Severity: Informational

Context: PolicyholderCheckpoints.sol#L66

Description: `push` and `_insert` have opposite order of args (`quantity/expiration` vs `expiration/quantity`).

Recommendation: Inline comments to caution developers that future edits will need to be made cautiously.

Llama: Fixed in commit 53d3e4b, PR 470. We have added a comment, see the PR description for more info.

Spearbit: Acknowledged.

5.3.7 Checkpoints `push` interface can require desired types

Severity: Informational

Context: PolicyholderCheckpoints.sol#L66, SupplyCheckpoints.sol#L70

Description: In `LlamaPolicy`, where both checkpoints are `push`ed from, the arguments are already appropriately typed.

Recommendation: The typecasting can be removed from both `push` functions:

```
-function push(History storage self, uint256 numberOfHolders, uint256 totalQuantity) internal {
+function push(History storage self, uint96 numberOfHolders, uint96 totalQuantity) internal {
-    _insert(self._checkpoints, LlamaUtils.toUint64(block.timestamp),
↪  LlamaUtils.toUint96(numberOfHolders), LlamaUtils.toUint96(totalQuantity));
+    _insert(self._checkpoints, LlamaUtils.toUint64(block.timestamp), numberOfHolders, totalQuantity);
}
```

and

```
-function push(History storage self, uint256 quantity, uint256 expiration) internal returns (uint96,
↪  uint96) {
+function push(History storage self, uint96 quantity, uint64 expiration) internal returns (uint96,
↪  uint96) {
-    return _insert(self._checkpoints, LlamaUtils.toUint64(block.timestamp),
↪  LlamaUtils.toUint64(expiration), LlamaUtils.toUint96(quantity));
+    return _insert(self._checkpoints, LlamaUtils.toUint64(block.timestamp), expiration, quantity);
}
```

Llama: We chose to acknowledge but close this issue. Leaving it as a `uint256` does add minimal gas overhead but it also minimizes the diff between the original OZ library.

Spearbit: Acknowledged.

5.3.8 a disapproval voter might need to first queue an action to vote against it

Severity: Informational

Context: LlamaCore.sol#L359

Description: In `Llama` an approved action needs to be `queued` first before voters with the `disapprovalRole` can actively vote against it. The queuing step happens via a public callable `LlamaCore.queueAction` function. This can lead to the scenario, in which a voter who wants to vote against the action needs to first queue it.

Queuing can be seen as a step closer to execution, which is counter-intuitive from a disapproval perspective.

Recommendation: Automatically call the `queueAction` function in `LlamaCore.castApproval` if the action is approved after the `cast`

Llama: Completed in commit a81186.

Spearbit: Resolved.

5.3.9 approvalEndTime should return 0 for non existing actions

Severity: Informational

Context: LlamaAbsoluteStrategyBase.sol#L278

Description: Currently, the `approvalEndTime` method returns the `approvalPeriod` which is a time interval for non-existing actions.

Recommendation: It would be cleaner to return `0` in the non-existing case.

```
  function approvalEndTime(ActionInfo calldata actionInfo) public view virtual returns (uint256) {
    Action memory action = llamaCore.getAction(actionInfo.id);
+   if(action.creationTime == 0) return 0;
    return action.creationTime + approvalPeriod;
  }
```

Llama: We acknowledge this issue, but chose not to take action. Strategy view functions are not meant to be called by any onchain or offchain service other than the core. We will make sure to document this.

Spearbit: Acknowledged

5.3.10 no `LlamaCore.cancelActionBySig` function exists

Severity: Informational

Context: LlamaCore.sol#L403

Description: A creator can create an action with a `signature` instead of relying just on `msg.sender` by using the `LlamaCore.createActionBySig` function. Currently, it is not possible for the creator to cancel the action afterwards with a signature.

The existing `cancelAction` is based on `msg.sender`.

Recommendation: Implement a `cancelActionBySig` function for creators who want to interact based on signatures with Llama.

Llama: Completed in commit 206e36.

Spearbit: Fixed.

5.3.11 Some function parameters within the `LlamaLens` are not explicit enough

Severity: Informational

Context: LlamaLens.sol#L117 | LlamaLens.sol#L135

Description: Within the `LlamaLens`, various functions were created to help a caller determine addresses of different contracts within a `Llama` instance. The parameters for some of them are a bit misleading in naming.

```
function computeLlamaStrategyAddress(address llamaStrategyLogic, bytes memory strategy, address
↪  llamaCore)
```

For example this one should be `strategyConfig` as strategy might mean a strategy address for a caller.

Recommendation: Consider renaming the parameters a bit more explicit.

Llama: Completed in commit 31a797.

Spearbit: Fixed.

5.3.12 Script for recovering assets stuck in the executor.

Severity: Informational

Context: LlamaExecutor.sol#L34

Description: The `LlamaExecutor` will be the caller to the targets, most of the time and it's very probable that some of the calls returns the assets to the caller, in this case being the executor.

Currently the only way to get these assets out is by used authorized scripts that it will be triggered via the governance process.

Recommendation: Consider adding a script for the community to use it to rescue funds that are stuck in the `LlamaExecutor`.

Llama: We acknowledge this issue, but chose not to resolve because adding a rescue function would mean that the core can make calls to an untrusted, third-party contract. We believe this situation is most likely to occur in the executor and that can be solved using a script.

Spearbit: Acknowledged.

5.3.13 Unnecessary `_infoHash` implementation

Severity: Informational

Context: LlamaCore.sol#L764

Description: Currently, there are 2 implementations of the `_infoHash` function, the one on line L764 being used just once on L622.

Recommendation: Consider removing the implementation on L764 for keeping the code cleaner.

Llama: Completed in commit a31444.

Spearbit: Fixed.