Llama Security Review

Auditors

Noah Marconi, Lead Security Researcher

M4rio.eth, Security Researcher

Xmxanuel, Security Researcher

TheMystery, Junior Security Researcher

Parth Patel, Junior Security Researcher

Report prepared by: Pablo Misirov

June 3, 2023

Contents

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 a governance system for onchain organizations. It uses non-transferable NFTs to encode access control, features programmatic control of funds, and includes a modular framework to define action execution rules.

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 10 days in total, llama engaged with Spearbit to review the Llama framework. In this period of time a total of **43** issues were found.

Summary

Project Name	llama
Repository	llama
Commit	9c87a7...fe62
Type of Project	Governance, DeFi and NFT
Audit Timeline	May 1 to May 12
Two week fix period	May 12 - May 26

Issues Found

Severity	**Count**	**Fixed**	**Acknowledged**
Critical Risk	2	2	0
High Risk	1	1	0
Medium Risk	6	5	1
Low Risk	14	10	4
Gas Optimizations	5	2	3
Informational	15	12	3
Total	**43**	**32**	**11**

5 Findings

5.1 Critical Risk

5.1.1 The `castApprovalBySig` and `castDisapprovalBySig` functions can revert

Severity: Critical Risk

Context: LlamaCore.sol#L683-L685

Description: The `castApprovalBySig` and `castDisapprovalBySig` functions are used to cast an approve or disapprove via an off-chain signature.

Within the `_preCastAssertions` a check is performed against the `strategy` using `msg.sender` instead of `policyholder`, the strategy (e.g. `AbsoluteStrategy`) uses that argument to check if the cast sender is a `policyholder`.

```
isApproval
  ? actionInfo.strategy.isApprovalEnabled(actionInfo, msg.sender)
  : actionInfo.strategy.isDisapprovalEnabled(actionInfo, msg.sender);
```

While this works for normal cast, using the ones with signatures will fail as the sender can be anyone who calls the method with the `signature` signed off-chain.

Recommendation: Consider sending the `policyholder` instead of `msg.sender`

Llama: Fixed in commit 4bb184 and PR 285.

Spearbit: Resolved.

5.1.2 The `castApproval/castDisapproval` doesn't check if `role` parameter is the `approvalRole`

Severity: Critical Risk

Context: LlamaCore.sol#L642

Description: A policyholder should be able to `cast` their approval for an `action` if they have the `approvalRole` defined in the strategy. It should not be possible for other `roles` to cast an action.

The `_castApproval` method verifies if the policyholder has the `role` passed as an argument but doesn't check if it actually has `approvalRole` which is eligible to cast an approval.

This means any role in the llama contract can participate in the approval with completely different `quantities` (weights).

The same problem occurs for the `castDisapproval` function as well.

Recommendation: The check could be added inside the strategy in the `getApprovalQuantityAt` function: RelativeStrategy.sol#L174

```
  function getApprovalQuantityAt(address policyholder, uint8 role, uint256 timestamp) external view
  ↪  returns (uint128) {
+   if (role != approvalRole) return 0;
    uint128 quantity = policy.getPastQuantity(policyholder, role, timestamp);
    return quantity > 0 && forceApprovalRole[role] ? type(uint128).max : quantity;
 }
```

If the passed `role` doesn't equal the `approvalRole` a quantity of zero could be returned.

```
if (role != approvalRole) return 0;
```

Llama: Fixed in commit 38b5a9.

Spearbit: Resolved.

5.2 High Risk

5.2.1 Reducing the `quantity` of a `policyholder` results in an increase instead of a decrease in `totalQuantity`

Severity: High Risk

Context: LlamaPolicy.sol#L388

Description: In Llama `policyholder` can approve or disapprove actions. Each policyholder has a `quantity` which represents their approval casting power. It is possible to update the `quantity` of individual `policyholder` with the `setRoleHolder` function in the LlamaPolicy.

The `_setRoleHolder` method is not handling the `decrease` of quantity correctly for the `totalQuantity`.

The `totalQuantity` describes the sum of the quantities of the individual policyholders for a specific role.

In the case of a quantity change, the difference is calculated as follows:

```
uint128 quantityDiff = initialQuantity > quantity ? initialQuantity - quantity : quantity -
↪  initialQuantity;
```

However, the `quantityDiff` is always added instead of being subtracted when the quantity is reduced.

This results in an incorrect tracking of the `totalQuantity`.

Adding the `quantityDiff` should only happen in the increase case.

See: LlamaPolicy.sol#L388

```
// case: willHaveRole=true, hadRoleQuantity=true
newTotalQuantity = currentRoleSupply.totalQuantity + quantityDiff;
```

Recommendation: The increase and decrease case of quantity should be handled in separate if-conditions or using an `int` variable for calculating the difference.

Decrease:

```
if(hadRoleQuantity && willHaveRole && initialQuantity > quantity) {
  newTotalQuantity = currentRoleSupply.totalQuantity - quantityDiff;
}
```

Increase:

```
if(hadRoleQuantity && willHaveRole && initialQuantity < quantity) {
  newTotalQuantity = currentRoleSupply.totalQuantity + quantityDiff;
}
```

Llama: Resolved by commit eb90d2 and PR 291.

Spearbit: Resolved.

5.3 Medium Risk

5.3.1 `LlamaPolicy.revokePolicy` cannot be called repeatedly and may result in burned tokens retaining active roles

Severity: Medium Risk

Context: LlamaPolicy.sol#L199)

Description: Llama has two distinct `revokePolicy` functions.

The first `revokePolicy` function removes all roles of a policyholder and burns the associated token.

This function iterates over all existing roles, regardless of whether a policyholder still holds the role. In the next step the token is burned. If the total number of roles becomes too high, this transaction might not fit into one block.

A second version of the revokePolicy function allows users to pass an array of roles to be removed. This approach should enable the function to be called multiple times, thus avoiding an "out-of-gas" error.

An `out-of-gas` error is currently not very likely considering the maximum possible role number of `255`.

However, the method exists and could be called with a subset of the roles a policyholder.

The method contains the following check:

```
if (balanceOf(policyholder) == 0) revert AddressDoesNotHoldPolicy(policyholder);
```

Therefore, it is not possible to call the method multiple times.

The result of a call with a subset of roles would lead to an inconsistent state.

The token of the policyholder is burned, but the policyholder could still use the remaining roles in Llama.

Important methods like `LlamaPolicy.hasRole` don't check if the token has been burned. (See LlamaPolicy.sol#L250)

Recommendation: Add a separate method `revokePolicyWithoutBurn` which only revokes the roles. This method can be provided as part of `llama` script. The script could check the `numRoles` and splits the call into multiple `revokePolicyWithoutBurn` ensuring each role will be revoked. In the last call the `token` should be burned.

Llama: Resolved by commit ad0391 and PR 294.

Spearbit: Resolved.

5.3.2 Role, permission, strategy, and guard management or config errors may prevent creating/approving/queuing/executing actions

Severity: Medium Risk

Context: Creating

- LlamaCore.sol#L492
- LlamaCore.sol#L583 Policyholder check
- LlamaCore.sol#L589 Strategy validateActionCreation
- LlamaCore.sol#L592 Guard check

Approving

- LlamaCore.sol#L492
- LlamaCore.sol#L544-L552 `getActionState`'s calls to the strategy
- LlamaCore.sol#L680 Policyholder check

Queuing

- LlamaCore.sol#L492

- LlamaCore.sol#L544-L552 `getActionState`'s calls to the strategy
- LlamaCore.sol#L283 Getting `minExecutionTime` for queuing

Executing

- LlamaCore.sol#L492
- LlamaCore.sol#L544-L552 `getActionState`'s calls to the strategy
- LlamaCore.sol#L303 Pre-execution guard
- LlamaCore.sol#L346 Post-execution guard

Description: `LlamaCore` deployment from the factory will only succeed if one of the roles is the `BOOTSTRAP_ROLE`. As the comments note:

```
// There must be at least one role holder with role ID of 1, since that role ID is initially
// given permission to call `setRolePermission`. This is required to reduce the chance that an
// instance is deployed with an invalid configuration that results in the instance being unusable.
// Role ID 1 is referred to as the bootstrap role.
```

There are still several ways a user can misstep and lose access to `LlamaCore`.

- **Bootstrap Role Scenarios** While the bootstrap role is still needed:

1. Setting an expiry on the bootstrap role's policyholder `RoleHolderData` and allowing the timestamp to pass. Once passed any caller may remove the `BOOTSTRAP_ROLE` from expired policyholders.
2. Removing the `BOOTSTRAP_ROLE` from all policyholders.
3. Revoking the role's permission with `setRolePermission(BOOTSTRAP_ROLE, bootstrapPermissionId, false)`.

- **General Roles and Permissions** Similarly, users may allow other permissions to expire, or remove/revoke them, which can leave the contract in a state where no permissions exist to interact with it. The `BOOTSTRAP_ROLE` would need to be revoked or otherwise out of use for this to be a problem.
- **Misconfigured Strategies** A misconfigured strategy may also result in the inability to process new actions. For example:

1. Setting minApprovals too high.
2. Setting `queuingPeriod` unreasonably high
3. Calling `revokePolicy` when doing so would make `policy.getRoleSupplyAsQuantitySum(approvalRole)` fall below `minApprovals` (or fall below `minApprovals` - `actionCreatorApprovalRoleQty`).

1 & 2 but applied to disapprovals.

And more, depending on the strategy (e.g. if a strategy always responded `true` to `isActive`).

- **Removal of Strategies** It should not be possible to remove the last strategy of a Llama instance It is possible to remove all strategies from an llama instance. It would not be possible to create a new `action` afterward. An `action` is required to add other strategies back.

As a result, the instance would become unusable, and access to funds locked in the `Accounts` would be lost.

- **Misconfigured Guards**

An accidentally overly aggressive guard could block all transactions. There is a built-in protection to prevent guards from getting in the way of basic management `if (target == address(this) || target == address(policy)) revert CannotUseCoreOrPolicy();`.

Again, the `BOOTSTRAP_ROLE` would need to be revoked or otherwise out of use for this to be a problem.

Recommendation:

- **Bootstrap Role** Given the importance of the bootstrap role, and the factory-level enforcement of its existence, preventing it from expiring would prevent accidental loss of the role.

In general, exercise extreme caution when removing or disabling this role.

- **General Roles and Permissions** In addition to any potential code changes, documentation and UI should highlight the foot guns present when removing role permissions and roles from policyholders (i.e. care must be taken to ensure the remaining permissions/roles support the ongoing operation of `LlamaCore`).
- **Misconfigured Strategies** Again, document and surface risks in UI at a minimum. Consider adding on-chain checks if ever possible to prevent strategies that are unable to create/approve/execute actions.
- **Removal of Strategies** It should not be possible to remove the last strategy of a Llama instance

Risk of removing strategies should be documented and surfaced in the UI, as even if one strategy remains, it may not be the strategy needed to support the remaining permissions.

- **Misconfigured Guards**

Similar to strategies, document and surface risks in UI at minimum. Consider adding on-chain checks if ever possible to prevent guards that are unable to create/approve/execute actions.

Llama: We made all possible changes to accommodate these findings, not everything can be solved onchain but we will have offchain infrastructure to help prevent these situations. The changes that were performed:

1. Expect the bootstrap role to be at index 0
2. Require that it's expiration is type(uint64).max

Spearbit: Acknowledged.

5.3.3 `LlamaPolicy.hasRole` doesn't check if a policyholder holds a token

Severity: Medium Risk

Context: LlamaPolicy.sol#L250

Description: Incorrect usage of the `revokePolicy` function can result in a case, where the token of a policyholder is already burned but still holds a role.

The `hasRole` function doesn't check if in addition to the role the `policyholder` still holds the token to be active.

The `role` could still be used in the Llama system.

Recommendation: Add `if (balanceOf(policyholder) == 0) return false` to the `hasRole` function.

Llama: Resolved by commit ad0391 and PR 294.

Spearbit: Resolved.

5.3.4 Incorrect `isActionApproved` behavior if new policyholders get added after the `createAction` in the same `block.timestamp`

Severity: Medium Risk

Context: RelativeStrategy#L162

Description: Llama utilizes `Checkpoints` to store approval quantities per timestamp. If the current quantity changes, the previous values are preserved. The `block.timestamp` of `createAction` is used as a snapshot for the approval. (See: LlamaCore.sol#L597)

Thus, in addition to the Checkpoints, the `totalQuantity` or `numberOfHolders` at the `createAction` are included in the snapshot.

However, if new policyholders are added or their quantities change after the `createAction` within the same `block.timestamp`, they are not considered in the snapshot but remain eligible to cast an approval.

For example, if there are four policyholders together 50% minimum approval: If a new action is created and two policyholders are added subsequently within the same `block.timestamp`.

The `numberOfHolders` would be 4 in the snapshot instead of 6. All 6 policyholders could participate in the approval, and two approvals would be sufficient instead of 4.

Adding new policyholders together with creating a new action could happen easily in a llama script, which allows to bundle different actions.

If a separate action is used to add a new policyholder, the final execution happens via a public callable function. An attacker could exploit this by trying to execute the add new policyholder action if a new action is created

Recommendation: The correctness of the approval casting mechanism should not be impacted by other actions.

A simple solution could be to store for each role the timestamp of `lastActionCreated`. The `setRoleHolder` function should revert in case `lastActionCreated == block.timestamp`.

It would still be possible to add new policyholders and create an action in the same Llama script. However, it would ensure that policyholders are always added before the create action.

Llama: Resolved by commit 337672 and commit bee182.

Spearbit: Resolved.

5.3.5 LlamaCore `delegate calls` can bring Llama into an unusable state

Severity: Medium Risk

Context: LlamaCore.sol#L337

Description: The core contract in Llama allows the execution of actions through a `delegate_call`.

An action is executed as a `delegate_call` when the `target` is added as an `authorizedScript`.

This enables batching multiple tasks into a contract, which can be executed as a single action.

In the `delegate_call`, a script contract could modify arbitrary any slot of the core contract.

The Llama team is aware of this fact and has added additional safety-checks to see if the `slot0` has been modified by the `delegate_call`.

The `slot0` contains values that should never be allowed to change.

```
bytes32 originalStorage = _readSlot0();
(success, result) = actionInfo.target.delegatecall(actionInfo.data);
if (originalStorage != _readSlot0()) revert Slot0Changed();
```

A script might be intended to modify certain storage slots. However, incorrect `SSTORE` operations can completely break the contracts.

For example, setting `actionsCount = type(uint).max` would prevent creating any new actions, and access to funds stored in the `Account` would be lost.

Recommendation: Users should be aware of the risks associated with using the `delegate_call` in the LlamaCore contract.

We acknowledge the benefits of using a `delegate_call` in certain situations.

The risks of errors or malicious code in scripts must be clearly documented.

`SSTORE` operations in scripts should be reviewed carefully.

To further mitigate these risks, the LlamaCore contract could be separated into two individual contracts.

For not allowing certain variables to be changed at all.

Llama: Resolved by adding LlamaExecutor in commit d0ba59 and PR 298.

Spearbit: Resolved.

5.3.6 The execution opcode of an action can be changed from `call` to `delegate_call` after approval

Severity: Medium Risk

Context: LlamaCore.sol#L309

Description: In Llama an `action` only defines the `target` address and the function which should be called. An `action` doesn't implicitly define if the opcode should be a `call` or a `delegate_call`.

This only depends on whether the `target` address is added to `authorizedScripts` mapping. However, adding a `target` to the `authorizedScripts` can be done after the approval in a different action.

The `authorizedScript` action could use a different set of signers with a different approval strategy.

The change of adding a target to `authorizedScript` should not impact `actions` which are already `approved` and in the `queuing` state.

This could lead to security issues when policyholders approved the action under the assumption the `opcode` will be a call instead of a delegate call.

Recommendation: If policyholders approve or disapprove of an action, the execution opcode should already be defined. It should not be possible for other actions with different signers to indirectly change it without explicitly modifying the current action.

Llama: Resolved by commit e2c9ed.

Spearbit: Resolved.

5.4 Low Risk

5.4.1 `LlamaFactory` is governed by `Llama` itself

Severity: Low Risk

Context: LlamaFactory.sol#L159

Description: `Llama` uses their own governance system to govern the `LlamaFactory` contract. The `LlamaFactory` contract is responsible for authorizing new `LlamaStrategies`.

We can identify several potential drawbacks with this approach.

If only a single strategy contract is used and a critical bug is discovered, the implications could be significant.

In such a scenario, it would mean a broken strategy contract needs to be used by the Factory governance to deploy a fixed version of the strategy contract or enable other strategies.

The likelihood for this to happen is still low but implications could be critical.

Recommendation: Analyze the likelihood of such a scenario. One idea could be to decouple the Factory `governance` from Llama. This could be achieved by making the `Factory Ownable` which could be optional set to Llama governance.

This would allow us to use for example `gnosis safe` in the beginning and later on, migrate as indented to Llama Governance.

Llama: We are confident in the Llama governance system to govern the Llama Factory.

Spearbit: Acknowledged.

5.4.2 The permissionId doesn't include call or delegate-call for LlamaAccount.execute

Severity: Low Risk

Context: LlamaAccount.sol#L236)

Description: The decision if LlamaAccount.execute is a delegate_call depends on the bool flag parameter withDelegatecall. This parameter is not included in the permissionId, which controls role permissions in Llama.

The permissionId in Llama is calculated in the following way:

```
PermissionData memory permission = PermissionData(target, bytes4(data), strategy);
bytes32 permissionId = keccak256(abi.encode(permission));
```

The permissionId required for a role to perform an action only includes the function signature but not the parameters themselves.

It is impossible to define the opcode as part of the permissionId.

Recommendation: Include the opcode for the execution as part of permissionId or split the LlamaAcccount.execute into two separate functions.

Another alternative idea could be to use Llama guards for Accounts to verify to opcode.

Llama: In practice accounts will likely require more granular control than solely separating the permissions of a call vs. delegatecall from the account. Even an arbitrary call is sufficient to drain an account, so including call vs delegatecall in permission ID doesn't buy you much security. What's more likely is that each Account will have its own guard, and that guard can have its own permission system with any desired characteristics.

Spearbit: Acknowledged.

5.4.3 Nonconforming EIP-712 typehash

Severity: Low Risk

Context: LlamaCore.sol#L97, LlamaCore.sol#L102

Description: Incorrect strings used in computing the EIP-712 typehash.

1. The strings contain space( ) after comma(,) which is not standard EIP-712 behaviour.
2. ActionInfo is not used in typehash. There will be a mismatch when comparing to hashes produced by JS libs or solidity (if implemented), etc..

Not adhering to EIP-712 spec means wallets will not render correctly and any supporting tools will produce a different typehash.

Recommendation: Remove spaces( ) after commas(,) and append the ActionInfo struct.

Llama: Resolved in commit fbc9b4.

Spearbit: Resolved.

5.4.4 Various events do not add the `role` as parameter

Severity: Low Risk

Context: LlamaCore.sol#L52-L77

Description:

> *Note: During the audit, the client discovered an issue that affects their offchain infrastructure.*

Various events do not emit the `role` as parameter:

1. `event ActionCreated(uint256 id, address indexed creator, ILlamaStrategy indexed strategy, address indexed target, uint256 value, bytes data, string description);`
2. `event ApprovalCast(uint256 id, address indexed policyholder, uint256 quantity, string reason);`
3. `event DisapprovalCast(uint256 id, address indexed policyholder, uint256 quantity, string reason);`

Recommendation: Consider adding the role as a parameter to the aforementioned events.

Llama: Fixed in commit 34e493 and PR 305.

Spearbit: Resolved.

5.4.5 `LlamaCore` doesn't check if `minExecutionTime` returned by `strategy` is in the past

Severity: Low Risk

Context: LlamaCore.sol#L283

Description: The `minExecutionTime` returned by a strategy is not validated.

Recommendation: An additional `require(minExecutionTime >= block.timestamp)` safety check could ensure more protection. The Llama strategies could be seen as a bit outside of the system and a lot of new strategies could be added.

An additional check would ensure the correct behavior of strategies.

Llama: Resolved by commit 312337 and PR 292.

Spearbit: Resolved.

5.4.6 Address parsing from `tokenId` to address string does not account for leading 0s

Severity: Low Risk

Context: LlamaPolicyMetadata.sol#L37, LlamaPolicyMetadata.sol#L27

Description: Policy `tokenIds` are derived from the holder's account address. The address is intended to be displayed in the svg generated when calling `tokenURI`.

Currently, leading 0s are truncated rendering the incorrect address string: e.g. `0x015b...` vs `0x0000...be60` for address `0x0000000000015B23C7e20b0eA5eBd84c39dCbE60`.

Recommendation: Instead of converting the `tokenId` from `uint256` to a hex string, first cast it to an `address` and make use of the `address` to hex string function from `LibString`:

```diff
- string memory policyholder = LibString.toHexString(tokenId);
+ string memory policyholder = LibString.toHexString(address(uint160(tokenId)));
```

Llama: Resolved by commit 0e5654 and PR 295

Spearbit: Resolved.

5.4.7 The `ALL_HOLDERS_ROLE` can be set as a force role by mistake

Severity: Low Risk

Context: RelativeStrategy.sol#L141-L145, AbsoluteStrategy.sol#L130-L142

Description: During the initialization, an array of roles that must be assigned as force approval/disapproval can be sent. The logic does not account for `ALL_HOLDERS_ROLE` (which is role id 0, the default value of uint8) which can be sent as a mistake by the user.

This is a low issue as if the above scenario happens, the strategy can become obsolete which will render the owner redeploy the strategy with correct initialization configs.

We must mention that the force roles can not be changed after they are set within the initialization.

Recommendation: Consider making an explicit logic for the owner to set `ALL_HOLDERS_ROLE` (e.g. an array with one role, `ALL_HOLDERS_ROLE`, must be sent for this functionality to be permitted, otherwise revert in case the array contains a 0)

Llama: The ability to set `ALL_HOLDERS_ROLE` as a force (dis)approval role has been prevented in commit f2cad6 and PR 303.

Spearbit: Resolved.

5.4.8 `LlamaPolicy.setRolePermission` allows to set permissions for non existing roles

Severity: Low Risk

Context: LlamaPolicy.sol#L396

Description: It is possible to set a permission for a role that doesn't exist, yet. In other functions like assigning a role to a policyholder, this check happens. (See: LlamaPolicy.sol#L343)

A related issue, very close to this, is the `updateRoleDescription` method which can emit an event for a role that does not exists. This is just an informational issue as it does not affect with anything the on-chain logic, might affect off-chain logic if any logic will ever rely on it.

Recommendation: Add the following check to the function

```
if (role > numRoles) revert RoleNotInitialized(role);
```

Llama: Resolved by commit 998d14 and PR 299.

Spearbit: Resolved.

5.4.9 The `RoleAssigned` event in LlamaPolicy emits the `currentRoleSupply` instead of the `quantity`

Severity: Low Risk

Context: LlamaPolicy.sol#L393

Description:

> *During the audit, the client discovered an issue that affects their off-chain infrastructure.*

The `RoleAssigned` event in `LlamaPolicy` emits the `currentRoleSupply` instead of the `quantity`.

From an off-chain perspective, there is currently no way to get the quantity assigned for a role to a policyholder at Role Assignment time. The event would be more useful if it emitted quantity instead of currentRoleSupply (since the latter can be just be calculated off-chain from the former).

Recommendation: Change `emit RoleAssigned(policyholder, role, expiration, currentRoleSupply);` to `emit RoleAssigned(policyholder, role, expiration, quantity);`

Llama: Fixed in commit de7168 and PR 288.

Spearbit: Resolved.

5.4.10 ETH can remain in the contract if `msg.value` is greater than expected

Severity: Low Risk

Context: LlamaCore.sol#L297

Description: When an action is created, the creator can specify an amount of `ETH` that needs to be sent when executing the transaction. This is necessary in order to forward `ETH` to a target call.

Currently, when executing the action the `msg.value` is checked to be at least the required amount of `ETH` needed to be forwarded. `if (msg.value < actionInfo.value) revert InsufficientMsgValue();`

This can result in ETH remaining in the contract after the execution. From our point of view, LlamaCore should not hold any balance of ETH.

Recommendation: Consider changing the `require` to an equal or forwarding the whole `msg.value` to the target.

Llama: Fixed in commit 23d376.

Spearbit: Resolved.

5.4.11 Cannot re-authorize an unauthorized strategy config

Severity: Low Risk

Context: LlamaCore.sol#L492 and LlamaCore.sol#L709-L710

Description: Strategies are deployed using a create2 salt. The salt is derived from the strategy config itself (see LlamaCore.sol#L709-L710).

This means that any unauthorized strategy cannot be used in the future, even if a user decides to re-enable it.

Recommendation: Consider if this is desired behavior. If not, allow re-authorization OR skip the create2 deployment attempt if the strategy address exists already.

Llama: Resolved by commit fa7bbf and PR 300.

We ended up removing the `unauthorizeStrategies` method and the concept of `authorizing` strategies and went with implicit unauthorization by unassigning all permission IDs using a given strategy

Spearbit: Resolved.

5.4.12 Signed messages may not be cancelled

Severity: Low Risk

Context: LlamaCore.sol#L143

Description: Creating, approving, and disapproving actions may all be done by signing a message and having another account call the relevant `*BySig` function. Currently, there is no way for a signed message to be revoked without a successful `*BySig` function call containing the nonce of the message to be revoked.

Recommendation: Provide a means of incrementing nonces to allow signers to revoke a signed message.

Llama: Resolved by commit c7b948 and PR 315.

Spearbit: Resolved.

5.4.13 LlamaCore `name` open to squatting or impersonation

Severity: Low Risk

Context: LlamaFactory.sol#L137

Description: When deploying a `LlamaCore` clone, the create2 salt is derived from the `name`. This means that no two may have the same name, and name squatting, or impersonation, may occur.

Recommendation: May be handled in the UI by being selective in what names are displayed.

Alternatively, a combination of name and salt could be used to form the create2 salt. This would still leave impersonation open but would prevent squatting as duplicate names would be allowed.

Finally, in discussion with the Llama team, a mitigation using `msg.sender` plus `name` to create the create2 salt would prevent squatting by allowing duplicate `name`s and tying addresses to the deployer account.

Llama: Since the deploy function is governed by Llama, we feel comfortable that we'll have the proper safeguards to prevent this issue.

Spearbit: Acknowledged.

5.4.14 Expired `policyholder`s are active until they are explicitly revoked

Severity: Low Risk

Context: LlamaPolicy.sol#L180

Description: Each policyholder in Llama has an `expiration` timestamp. However, `policyholder` can still use the power of their `role` after the `expiration` has passed.

The final revoke only happens after the public `LlamaPolicy.revokeExpiredRole` method is called. Anyone can call this method after the expiration timestamp is passed.

For the Llama system to function effectively with role expiration, it is essential that external keepers vigilantly monitor the contract and promptly revoke expired roles.

A final revoke exactly at the `expiration` can not be guaranteed.

Recommendation: Consider if this is the desired behavior, if not consider taking into account the expiration behavior of a role.

Llama: We acknowledge this issue as it's a trade-off we are ok with. The definition of a valid role is

- Role had quantity at action creation.
- Role may be revokable at action creation, but if not yet revoked, it's still active.

Spearbit: Acknowledged.

5.5 Gas Optimization

5.5.1 Gas optimizations

Severity: Gas Optimization

Context: General

Description: Throughout the codebase we've identified gas improvements that were aggregated into one issue for a better management.

RelativeStrategy.sol#L159

- The `if (disapprovalPolicySupply == 0) revert RoleHasZeroSupply(disapprovalRole);` check and `actionDisapprovalSupply[actionInfo.id] = disapprovalPolicySupply;` can be wrapped in an `if` block in case `disapprovals` are enabled
- The `uint128 newNumberOfHolders;` and `uint128 newTotalQuantity;` variables are obsolete as the updates on the `currentRoleSupply` can be done in the if branches.

LlamaPolicy.sol#L380-L392

- The `exists` check is redundant LlamaPolicy.sol#L252
- The `_validateActionInfoHash(action.infoHash, actionInfo);` is redundant as it's already done in the `getActionState` LlamaCore.sol#L292

LlamaCore.sol#L280

LlamaCore.sol#L672

- Finding the `BOOTSTRAP_ROLE` in the `LlamaFactory._deploy` could happen by expecting the role at a certain position like `position 0` instead of paying gas for an on-chain search operation to iterate the array. LlamaFactory.sol#L205
- `quantityDiff` calculation guaranteed to not overflow as the ternary checks `initialQuantity > quantity` before subtracting.
- Infeasible for `numberOfHolders` and `totalQuantity` to overflow. See also LlamaPolicy.sol#L422-L423
- Infeasible for `numberOfHolders` to overflow.

Recommendation: Consider implementing the gas optimizations.

- `approvalRole` can be read from memory instead of reloading from storage.

Llama: Resolved in PR 284 and commit c0303c. We decided to acknowledge the first issue as it would add too much complexity to the code.

Spearbit: Resolved.

5.5.2 Unused code

Severity: Gas Optimization

Context: Description: Various parts of the code is unused or unnecessary.

- `CallReverted` and `MissingAdmin` in LlamaPolicy.sol#L27-L29
- `DisapprovalThresholdNotMet` in RelativeStrategy.sol#L28
- Unused errors in LlamaCore.sol `InvalidCancelation`, `ProhibitedByActionGuard`, `ProhibitedByStrategy`, `ProhibitedByStrategy(bytes32 reason)` and `RoleHasZeroSupply(uint8 role)`
- `///   - Action creators are not allowed to cast approvals or disapprovals on their own actions`, The comment is inaccurate, this strategy, the creators have no restrictions on their actions. RelativeStrategy.sol#L19

Recommendation: Consider removing the unnecessary code blocks.

Llama: Resolved in PR 227 and commit 5071e5. diff.

Spearbit: Resolved.

5.5.3 Duplicate storage reads and external `call`s

Severity: Gas Optimization

Context: `castApproval` sequence diagram, `createAction` sequence diagram

Description: When creating, approving, disapproving, queuing, and executing actions, there are calls between the various contracts in the system. Due to the external calls, the compiler will not cache storage reads, meaning the gas cost of warm `sloads` is incurred multiple times.

The same is true for `view` function calls between the contracts. A number of these calls are returning the same value multiple times in a transaction.

Recommendation: Consider where simple changes can be made to cache reads or avoid duplicate calls. Where entirely eliminating duplicate reads, we agree that there is a trade-off, and saving 100 gas for a warm read may not be worth added complexity for some areas.

Llama: We addressed some of these findings in commit e5d5ee, but in some cases we decided to optimize for readability over optimization.

Spearbit: Acknowledged.

5.5.4 Consider `clones-with-immutable-args`

Severity: Gas Optimization

Context: LlamaCore.sol#L108-L117, and elsewhere.

Description: The cloned contracts have immutable values that are written to storage on initialization due to proxies being used. Reading from storage costs extra gas but also puts some of the storage values at risk of being overwritten when making delegate calls.

Recommendation: Consider if `clones-with-immutable-args` is appropriate for the project.

Llama: We chose not to use one of the `clones-with-immutable-args` implementations because:

1. The contracts are not audited (they have been involved with audits but never directly audited themselves)
2. Infrequent governance actions are typically less gas-sensitive than e.g. critical path DeFi methods called by every user, which we think makes this change not worth the risk due to (1)

Spearbit: Acknowledged. The remaining delegate call risks are discussed in another ticket.

5.5.5 The `domainSeperator` may be cached

Severity: Gas Optimization

Context: LlamaCore.sol#L620, LlamaCore.sol#L460, LlamaCore.sol#L403

Description: The `domainSeperator` is computed for each use. Some gas may be saved by using caching and deferring to the cached value.

Recommendation: Consider using one of OpenZeppelin or Solady which are both project dependencies.

Llama: The domain separator would be cached as immutable in the case of using the OpenZeppelin library. But in Llama's case, all the Llama Cores are minimal proxies which would each need their own domain separators.

Spearbit: Acknowledged.

5.6 Informational

5.6.1 Prefer on-chain `SVGs` or `IPFS` links over server links for `contractURI`

Severity: Informational

Context: LlamaPolicyMetadata.sol#L97

Description: Llama uses on-chain SVG for `LlamaPolicy.tokenURI`. The same could be implemented for `LlamaPolicy.contractURI` as well.

In general IPFS links or on-chain SVG for visual representations provide better properties than centralized server links.

Recommendation: Change hardcoded `URLs` in `contractURI` to SVG's or an `IPFS` link.

Llama: These images are only used by NFT marketplace frontends and are cached anyway by their CDNs. We lean towards leaving it to preserve the flexibility if the logo changes in the future.

Spearbit: Acknowledged.

5.6.2 Consider making the `delegate-call` scripts functions only callable by `delegate-call`

Severity: Informational

Context: GovernanceScript.sol#L63

Description: An additional safety check could be added to scripts if a function should be only callable via a `delegate-call`.

Recommendation: The following modifier could be added to `delegate-call` only script functions.

```
address public immutable SELF;
constructor () {
  SELF = address(this);
}

modifier onlyDelegateCall {
  require(address(this) != SELF);
  _;
}
```

Llama: Fixed in commit 5834a2.

Spearbit: Resolved.

5.6.3 Missing tests for `SingleUseScript.sol`

Severity: Informational

Context: SingleUseScript.sol#L10

Description: There are no tests for `SingleUseScript.sol` in Llama.

Recommendation: Each contract used for Llama should be tested with high coverage.

Llama: Fixed in PR 331.

Spearbit: Resolved.

5.6.4 Role not available to `Guards`

Severity: Informational

Context: Structs.sol#L29-L36, ILlamaStrategy.sol#L14

Description: Use cases where `Guards` require knowing the creation or approval role for the action are not supported.

`ActionInfo` does reference the strategy, and the two implemented strategies do have public functions referencing the `approvalRole`, allowing for a workaround. However, this is not mandated by the `ILlamaStrategy` interface and is not guaranteed to be present in future strategies.

Recommendation: Consider exposing role to guards.

Llama: Action creation role was added to the ActionInfo struct in PR 281. If a guard wanted the (dis)approval role it could try querying the strategy (there's no guarantee that the strategy has the approval or disapproval role exposed, however, both strategies currently implemented do).

Spearbit: Resolved.

5.6.5 Global `guards` are not supported

Severity: Informational

Context: LlamaCore.sol#L146

Description: Other protocols use of guards applies them to the account (i.e. globally). In other words, if global guards existed and if there are some properties you know to apply to the entire `LlamaCore` instance a global guard could be applied.

The current implementation allows granular control, but it also *requires* granular control with no ability to set global guards.

Recommendation: Note in docs and/or consider if global guards are desirable.

Llama: We explicitly avoided global guards to reduce the risk of unintended consequences. If users want to generalize their guards they can accomplish that by using a script as an abstraction layer on administrative functions and guard that script.

Spearbit: Acknowledged.

5.6.6 Consider using `_disableInitializers` in constructor

Severity: Informational

Context: LlamaCore.sol#L152

Description: OpenZeppelin added the `_disableInitializers()` in 4.6.0 which prevents initialization of the implementation contract and recommends its use.

Recommendation: Remove the `initializer` modifier from the constructor and add `_disableInitializers()` in the constructor body.

Llama: Resolved in commit e3da48 and PR 293.

Spearbit: Resolved.

5.6.7 Consider renaming `initialAccounts` to `initialAccountNames` for clarity

Severity: Informational

Context: LlamaCore.sol#L168

Description: Consider renaming `initialAccounts` to `initialAccountNames` for clarity. The more verbose argument name makes it clear what to pass as input.

Llama: Resolved by commit 5f3d2c and PR 313.

Spearbit: Resolved.

5.6.8 Revoking and setting a `role` edge cases

Severity: Informational

Context: LlamaPolicy.sol#LL158C12-L158C25, /LlamaPolicy.sol#L180

Description: This issue highlights a number of edge-case behaviors

1. Calling `setRoleHolder` passing in an account with `balanceOf == 0`, 0 quantity, and 0 expiration results in minting the NFT.
2. Revoking all policies through `revokeExpiredRole` leaves an address with no roles except for the `ALL_HOLDERS_ROLE` and a `balanceOf == 1`.
3. Revoking may be conducted on policies the address does not have (building on the previous scenario):

- Alice is given role 1 with expiry.
- Expiry passes.
- Anyone calls `revokeExpiredRole`.
- Role is revoked but Alice still has `balanceOf == 1`.
- `LlamaCore` later calls `revokePolicy` with roles array of `[2]`.
- A role Alice never had is revoked.
- The NFT is burned.

Recommendation: Document the edge cases and the recommended means of revoking.

Llama: Acknowledged 1 & 2 as this is intentional behavior to allow minting an account a policy that only has the `ALL_HOLDERS_ROLE`, and resolved 3 with commit 68538b and PR 318.

Spearbit: Acknowledged.

5.6.9 Use built in `string.concat`

Severity: Informational

Context: LlamaPolicyMetadata.sol#L48, LlamaPolicyMetadata.sol#L63-L66, LlamaPolicyMetadata.sol#L85, LlamaPolicyMetadata.sol#L99-L100

Description: The solidity version used has a built-in `string.concat` which can replace the instances of `string(abi.encodePacked(...))`.

The client notes there are no gas implications of this change while the change does offer semantic clarity.

Recommendation: Use built in `string.concat`.

Llama: Fixed in commit c305ce.

Spearbit: Resolved.

5.6.10 Inconsistencies

Severity: Informational

Context: Description: Throughout the codebase, we've encountered some inconsistencies that we decided to point out.

`for(uint256 i = 0...` is not used everywhere e.g. AbsoluteStrategy.sol#L130

- Sometimes, a returned value is not named. e.g. named return value

```
function createAction(
  uint8 role,
  ILlamaStrategy strategy,
  address target,
  uint256 value,
  bytes calldata data,
  string memory description
) external returns (uint256 actionId) {
```

unnamed return value

```
function createActionBySig(
  uint8 role,
  ILlamaStrategy strategy,
  address target,
  uint256 value,
  bytes calldata data,
  address policyholder,
  uint8 v,
  bytes32 r,
  bytes32 s
) external returns (uint256) {
```

- Missing `NatSpec` on various functions. e.g. LlamaPolicy.sol#L102
- `_uncheckedIncrement` is not used everywhere.
- **Naming of modifiers** In all contracts the `onlyLlama` modfiier only refers to the `llamaCore`. The only exception is `LlamaPolicyMetadataParamRegistry` which has the same name but refers to `llamaCore` and `rootLlama` but is called `onlyLlama`. See LlamaPolicyMetadataParamRegistry.sol#L16
- **Console.log debug output in RelativeStrategy** `console.log` in `RelativeStrategy` See: RelativeStrategy.sol#L215
- In `GovernanceScript.sol` both of `SetRolePermission` and `SetRoleHolder` mirror structs defined in the shared `lib/Structs.sol` file.

Additionally, some contracts declare their own structs over inheriting all structs from `lib/Structs.sol`:

- LlamaAccount
- GovernanceScript
- LlamaPolicy

Recommend removing duplicate structs and, where relevant, continue making use of the shared `Structs.sol` for struct definitions.

Recommendation: Consider keeping the code consistent for a better readability

Llama: PR 321 fixed with the following notes:

- We've chosen not to consistently use named or unnamed returns. We think named returns are useful in certain contexts but not always necessary.
- We've updated the NatSpec for all contracts in PR 349.

- Modifier naming and the leftover console.log statement were addressed in these two PRs respectively: PR 279 and PR 277.
- We chose to remove duplicate structs, but only make use of the shared `Structs.sol` for structs that appear in more than one file in the src directory. For structs that appear in a single file we're ok with declaring them in that file.

Spearbit: Resolved.

5.6.11 Policyholders with large quantities may not both create and exercise their large quantity for the same action

Severity: Informational

Context: AbsoluteStrategy.sol#L178

Description: The `AbsoluteStrategy` removes the action creator from the set of policyholders who may approve / disapprove an action. This is a departure from how the `RelativeStrategy` handles action creators.

Not permitting action creators to approve / disapprove is simple to reason about when each policyholder has a quantity of 1; creating can even be thought of an implicit approval and may be factored in when choosing a `minApprovals` value. However, in scenarios where a policyholder has a large quantity (in effect a large weight to their casted approval), creating an action means they forfeit the use of the vast majority of their quantity for that particular action.

Recommendation: For the strategy as is, emphasize the behavior in documentation so that action creators are not surprised by the loss of influence.

A code based solution would be to consider adding a strategy that allows the creator's quantity to count as casting approval. Documenting the difference between a strategy like this and the current `AbsoluteStrategy` would be important.

Finally, without modifications to the codebase, the issue may be circumvented by asking policyholders with large quantities to provide two addresses, one for exercising their large quantity and a second for creating actions. Assigning both addresses the same role, with appropriate quantities, would allow the policyholder to both create an action and exercise their large quantity.

Llama: A new strategy has been added called `AbsoluteQuorum`. The old `AbsoluteStrategy` has been split into 2 contracts `AbsoluteStrategyBase` and `PeerReview`. Fixed in PR 345.

Spearbit: Resolved.

5.6.12 The `roleBalanceCheckpoints` can run out of gas

Severity: Informational

Context: LlamaPolicy.sol#L244-L246

Description: The `roleBalanceCheckpoints` function returns the Checkpoints history of a balance. This check will copy into memory the whole history which can end up in a out of gas error.

This is an informational issue as this function was designed for off-chain usage and the caller can use `eth_call` with a higher gas limit.

Recommendation: Consider adding a paginated version of this function, furthermore, consider adding an extra function that returns the length of the `_checkpoints`.

Llama: Fixed in commit e055e9 and PR 310.

Spearbit: Resolved.

5.6.13 `GovernanceScript.revokeExpiredRoles` should be avoided in favor of calling `LlamaPolicy.revokeExpiredRole` from EOA

Severity: Informational

Context: GovernanceScript.sol#L211

Description: `GovernanceScript.revokeExpiredRoles` is intended to be delagate called from `LlamaCore`. Given that `LlamaPolicy.revokeExpiredRole` is already public and without access controls, it will always be cheaper, and less complex, to call directly from an EOA or batching a multicall, again from an EOA.

Recommendation: Prefer calling `LlamaPolicy.revokeExpiredRole` directly or via multicall over using `GovernanceScript.revokeExpiredRoles`.

Llama: Resolved with commit 2e4de6.

Spearbit: Resolved.

5.6.14 The `InvalidActionState` can be improved

Severity: Informational

Context: LlamaCore.sol#L281, LlamaCore.sol#L295

Description: Currently, the `InvalidActionState` includes the expected state as an argument, this is unnecessary as you can derive the state from the method call, would make more sense to take the current state instead of the expected state.

Recommendation: Consider adding the current state to the revert error instead of the expected state.

Llama: Fixed in commit 5fa77c and PR 311.

Spearbit: Resolved.

5.6.15 `_uncheckedIncrement` function written in multiple contracts

Severity: Informational

Context: LlamaCore.sol#L771, LlamaFactory.sol#L243, LlamaAccount.sol#L284, LlamaPolicy.sol#L432, RelativeStrategy.sol#L287, AbsoluteStrategy.sol#L285

Description: Multiple contracts make use of an `_uncheckedIncrement`function and each duplicates the function definition.

Similarly the `slot0` function appears in both `LlamaAccount` and `LlamaCore` and `_toUint64` appears in the two strategy contracts plus `LlamaCore`.

Recommendation: Move shared functions to a local lib to DRY out the codebase.

Llama: Resolved by commit 0a0aa4.

Spearbit: Resolved.