Llama
Findings & Analysis Report

2023-07-26

Table of contents

Overview

About C4

Code4rena (C4) is an open organization consisting of security researchers, auditors, developers, and individuals with domain expertise in smart contracts.

A C4 audit is an event in which community participants, referred to as Wardens, review, audit, or analyze smart contract logic in exchange for a bounty provided by sponsoring projects.

During the audit outlined in this document, C4 conducted an analysis of the Llama smart contract system written in Solidity. The audit took place between June 6—June 14 2023.

Wardens

50 wardens contributed reports to the Llama Audit:

1. 0xHati
2. 0xSmartContract
3. 0xcm
4. 0xnev
5. Atree
6. BLOS
7. BRONZEDISC
8. CoOnan
9. DavidGiladi
10. Go-Langer

11. JCN
12. K42
13. Madalad
14. MiniGlome
15. QiuhaoLi
16. Rageur
17. Raihan
18. Rolezn
19. SAAJ
20. SAQ
21. SM3_SS
22. Sathish9098
23. T1MOH
24. Toshii
25. Udsen
26. VictoryGod
27. auditor0517
28. dirk_y
29. ernestognw
30. flacko
31. hunter_w3b
32. joestakey
33. ktg
34. kutugu
35. libratus
36. lsaudit
37. mahdirostami
38. matrix_0wl
39. minhquanym

40. n1punp
41. naman1778
42. neko_nyaa
43. peanuts
44. petrichor
45. qpzm
46. rvierdiiev
47. sces60107
48. sebghatullah
49. shamsulhaq123
50. xuwinnie

This audit was judged by gzeon.

Final report assembled by PaperParachute.

Summary

The C4 analysis yielded an aggregated total of 5 unique vulnerabilities. Of these vulnerabilities, 2 received a risk rating in the category of HIGH severity and 3 received a risk rating in the category of MEDIUM severity.

Additionally, C4 analysis included 13 reports detailing issues with a risk rating of LOW severity or non-critical. There were also 17 reports recommending gas optimizations.

All of the issues presented here are linked back to their original finding.

Scope

The code under review can be found within the C4 Llama repository, and is composed of 23 smart contracts written in the Solidity programming language and includes 2096 lines of Solidity code.

Severity Criteria

C4 assesses the severity of disclosed vulnerabilities based on three primary risk categories: high, medium, and low/non-critical.

High-level considerations for vulnerabilities span the following key areas when conducting assessments:

- Malicious Input Handling
- Escalation of privileges
- Arithmetic
- Gas use

For more information regarding the severity criteria referenced throughout the submission review process, please refer to the documentation provided on the C4 website, specifically our section on Severity Categorization.

High Risk Findings (2)

[H-01] In `LlamaRelativeQuorum`, the governance result might be incorrect as it counts the wrong approval/disapproval

Submitted by auditor0517, *also found by* Toshii, kutugu, 0xnev, *and* T1MOH

https://github.com/code-423n4/2023-06-llama/blob/9d641b32e3f4092cc81dbac7b1c451c695e78983/src/strategies/LlamaRelativeQuorum.sol#L223
https://github.com/code-423n4/2023-06-llama/blob/9d641b32e3f4092cc81dbac7b1c451c695e78983/src/strategies/LlamaRelativeQuorum.sol#L242

Proof of Concept

The `LlamaRelativeQuorum` uses approval/disapproval thresholds that are specified as percentages of total supply and the approval/disapproval supplies are set at `validateActionCreation()` during the action creation.

```
function validateActionCreation(ActionInfo calldata actionInfo
  LlamaPolicy llamaPolicy = policy; // Reduce SLOADs.
```

```
        uint256 approvalPolicySupply = llamaPolicy.getRoleSupplyAsNu
        if (approvalPolicySupply == 0) revert RoleHasZeroSupply(appr

        uint256 disapprovalPolicySupply = llamaPolicy.getRoleSupplyA
        if (disapprovalPolicySupply == 0) revert RoleHasZeroSupply(d

        // Save off the supplies to use for checking quorum.
        actionApprovalSupply[actionInfo.id] = approvalPolicySupply;
        actionDisapprovalSupply[actionInfo.id] = disapprovalPolicySu
    }
```

As we can see, `actionApprovalSupply` and `actionDisapprovalSupply` are set using `getRoleSupplyAsNumberOfHolders` which means the total number of role holders.

But while counting for `totalApprovals/totalDisapprovals` in `getApprovalQuantityAt()/getDisapprovalQuantityAt()`, it adds the quantity instead of role holders(1 for each holder).

```
function getApprovalQuantityAt(address policyholder, uint8 role,
    if (role != approvalRole && !forceApprovalRole[role]) return
    uint128 quantity = policy.getPastQuantity(policyholder, role
    return quantity > 0 && forceApprovalRole[role] ? type(uint12
}
```

So the governance result would be wrong with the below example.

1. There are 3 role holders(Alice, Bob, Charlie) and Alice has 2 quantities, others have 1.
2. During the action creation with the `LlamaRelativeQuorum` strategy, `actionApprovalSupply = 3` and there should be 2 approved holders at least when `minApprovalPct = 51%`.
3. But if Alice approves the action, the result of `getApprovalQuantityAt()` will be 2 and the action will be approved with only one approval.

It's because `getApprovalQuantityAt()` return the quantity although `actionApprovalSupply` equals `NumberOfHolders`.

Recommended Mitigation Steps

`getApprovalQuantityAt()` and `getDisapprovalQuantityAt()` should return 1 instead of `quantity` for the positive quantity.

I think we can modify these functions like below.

```
function getApprovalQuantityAt(address policyholder, uint8 role
  if (role != approvalRole && !forceApprovalRole[role]) return
  uint128 quantity = policy.getPastQuantity(policyholder, role

  if (quantity > 1) quantity = 1;

  return quantity > 0 && forceApprovalRole[role] ? type(uint12
}

function getDisapprovalQuantityAt(address policyholder, uint8
  external
  view
  returns (uint128)
{
  if (role != disapprovalRole && !forceDisapprovalRole[role])
  uint128 quantity = policy.getPastQuantity(policyholder, role

  if (quantity > 1) quantity = 1;

  return quantity > 0 && forceDisapprovalRole[role] ? type(uin
}
```

AustinGreen (Llama) disputed and commented:

> This is actually how we intend this strategy to work but we're open to feedback! Here's an example:

- An instance has 10 role holders and a 50% min approval percentage. Each role holder's quantity is 1, so 5 role holders can approve this action.
- 2 of the role holders have their quantity increased to 2.
- This means that if each of these role holders cast approvals, then their approval power will count as 4. That means just one other role holder is needed to cast approval to approve the action.

> In this system quantity can be used to provide granular approval weights to role holders.

gzeon (Judge) commented:

> @AustinGreen- I don't think this make sense. Sure, if each holder's quantity is 1, then `getRoleSupply` is same as `getRoleSupplyAsNumberOfHolders` and what you said is valid. However, if you have 10 holders each with quantity 10 at snapshot, then your `actionApprovalSupply` is set to 10 (number of holder) and any of their approval (10 quantity) would hit quorum.

AustinGreen (Llama) commented:

> @gzeon- Yes that's exactly how the design is intended to work!

gzeon (Judge) commented:

> @AustinGreen- This sounds weird, is this design documented anywhere? From what I can see in the code comments it seems to be hard for anyone (including potential user/dao) to understand such logic.

> In the code, there is a comment

> Minimum percentage of `totalApprovalQuantity / totalApprovalSupplyAtCreationTime` required for the action to be queued

> I think it is fair for one to assume `totalApprovalQuantity` and `totalApprovalSupplyAtCreationTime` would be using the same metric, instead of one using the raw count and the other using `AsNumberOfHolders`.

AustinGreen (Llama) commented:

> Although this is the intended design for this strategy, we decided to create an additional strategy that Llama instances can adopt that follows the warden's recommendations. It uses total (dis)approval quantity for the quorum calculation as specified.

[H-02] Anyone can change approval/disapproval threshold for any action using LlamaRelativeQuorum strategy

Submitted by ktg, *also found by* auditor0517 *and* dirk_y

Proof of Concept

When a new action is created with `LlamaRelativeQuorum` strategy, `LlamaCore` will call function `validateActionCreation` which is currently implemented as below:

```
function validateActionCreation(ActionInfo calldata actionInfo) {
    LlamaPolicy llamaPolicy = policy; // Reduce SLOADs.
    uint256 approvalPolicySupply = llamaPolicy.getRoleSupplyAsNu
    if (approvalPolicySupply == 0) revert RoleHasZeroSupply(appr

    uint256 disapprovalPolicySupply = llamaPolicy.getRoleSupplyA
    if (disapprovalPolicySupply == 0) revert RoleHasZeroSupply(d

    // Save off the supplies to use for checking quorum.
    actionApprovalSupply[actionInfo.id] = approvalPolicySupply;
    actionDisapprovalSupply[actionInfo.id] = disapprovalPolicySu
  }
```

The last 2 lines of code is to `Save off the supplies to use for checking quorum`. The 2 variables `actionApprovalSupply` and `actionDisapprovalSupply` are described as `Mapping of action ID to the supply of the approval/disapproval role at the time the action was created.`

This means the strategy will save the total supply of approval/disapproval role at creation time and then use them to calculate the approval/disapproval threshold, which equals to (approval/disapproval percentage) * (total supply of approval/disapproval).

However, since the function `validateActionCreation`'s scope is `external` and does not require any privilege to be called, any user can call this function and update the total supply of approval/disapproval role to the current timestamp and break the intention to keep total supply of approval/disapproval role `at the time the action was created`. This issue is highly critical because many Llama protocol's functions depend on these 2 variables to function as intended.

For example, if the total supply of approval role is 10 at the creation of action and the `minApprovalPct` = 100% - which means requires all policy holders to approve the action to pass it.

If it then be casted 9 votes (1 vote short), the action's state is still Active (not approved yet).

However, if 1 user is revoked their approval/role, anyone can call function `validateActionCreation` and update the required threshold to 9 votes and thus the action's state becomes Approved.

Below is a POC for the above example, for ease of testing, place this test case under file `LlamaStrategy.t.sol` , contract `IsActionApproved` :

```
function testAnyoneCanChangeActionApprovalSupply() public {
    // Deploy a relative quorum strategy
    uint256 numberOfHolders = 10;

    // Assign 10 users role of TestRole1
    for (uint256 i=0; i< numberOfHolders; i++){
      address _policyHolder = address(uint160(i + 100));
      if (mpPolicy.balanceOf(_policyHolder) == 0) {
        vm.prank(address(mpExecutor));
        mpPolicy.setRoleHolder(uint8(Roles.TestRole1), _policyHo
      }
    }


    // Create  a LlamaRelativeQuorum strategy
    // in this minApprovalPct = 10_000 (meaning we require all 1
    LlamaRelativeQuorum.Config memory testStrategyData = LlamaRe
      approvalPeriod: 2 days,
      queuingPeriod: 2 days,
      expirationPeriod: 8 days,
      isFixedLengthApprovalPeriod: true,
      minApprovalPct: 10000, // require all policyholder to appr
      minDisapprovalPct: 2000,
      approvalRole: uint8(Roles.TestRole1),
      disapprovalRole: uint8(Roles.TestRole1),
      forceApprovalRoles: new uint8[](0),
      forceDisapprovalRoles: new uint8[](0)
    });
```

```
    ILlamaStrategy testStrategy = lens.computeLlamaStrategyAddre
      address(relativeQuorumLogic), DeployUtils.encodeStrategy(t
    );

    LlamaRelativeQuorum.Config[] memory testStrategies
    = new LlamaRelativeQuorum.Config[](1);
    testStrategies[0] = testStrategyData;
    vm.prank(address(mpExecutor));
    mpCore.createStrategies(relativeQuorumLogic, DeployUtils.enc

    // create action
    ActionInfo memory actionInfo = createAction(testStrategy);
    assertEq(LlamaRelativeQuorum(address(testStrategy)).actionAp

    // Suppose that 9 policyholder approve
    // the action lacks 1 more approval vote so isActionApproved
    approveAction(9, actionInfo);
    assertEq(LlamaRelativeQuorum(address(testStrategy)).isAction

    // Revoke 1 user
    vm.prank(address(mpExecutor));
    mpPolicy.revokePolicy(address(100));

    // Now anyone can update the actionApprovalSupply and theref
    // change the approval threshold
    address anyOne = address(12345);
    vm.prank(anyOne);
    LlamaRelativeQuorum(address(testStrategy)).validateActionCre

    // The actionApproval for the above action is reduced to 9
    // and the action state changes to approved
    assertEq(LlamaRelativeQuorum(address(testStrategy)).actionAp
    assertEq(LlamaRelativeQuorum(address(testStrategy)).isAction
  }
```

Recommended Mitigation Steps

Since the intention is to keep values `actionApprovalSupply` and `actionDisapprovalSupply` snapshot at creation time for every action and `LlamaCore` only call `validateActionCreation` at creation time, I think the easiest way is to allow only `llamaCore` to call this function.

AustinGreen (Llama) confirmed and commented:

> This finding was addressed in this PR: https://github.com/llamaxyz/llama/pull/384 (note our repo is private until we launch)

Medium Risk Findings (3)

[M-01] It is not possible to execute actions that require ETH (or other protocol token)

Submitted by libratus, *also found by* Udsen, flacko, joestakey, n1punp, Go-Langer, QiuhaoLi, sces60107, Toshii, rvierdiiev, minhquanym, Madalad, BRONZEDISC, 0xcm, ernestognw, Co0nan, T1MOH, *and* MiniGlome

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaCore.sol#L334
https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaExecutor.sol#L29

Actions can have value attached to them. That means when action is being executed, a certain amount of ETH (or other protocol token) need to be sent by the caller with the contract call. This is why `LlamaCore.executeAction` is payable.

```
function executeAction(ActionInfo calldata actionInfo) externa
```

However, when LlamaCore executes the action it doesn't pass value to the downstream call to LlamaExecutor

```
// Execute action.
(bool success, bytes memory result) =
  executor.execute(actionInfo.target, actionInfo.value, acti
```

LlamaExecutor's `execute` is not payable even though it does try to pass value to the downstream call

```
function execute(address target, uint256 value, bool isScript,
  external
```

```
    returns (bool success, bytes memory result)
  {
    if (msg.sender != LLAMA_CORE) revert OnlyLlamaCore();
    (success, result) = isScript ? target.delegatecall(data) : ta
  }
```

This will of course revert because LlamaExecutor is not expected to have any ETH balance.

Proof of Concept

To reproduce the issue based on the existing tests we can do the following changes:

```
diff --git a/test/LlamaCore.t.sol b/test/LlamaCore.t.sol
index 8135c93..6964846 100644
--- a/test/LlamaCore.t.sol
+++ b/test/LlamaCore.t.sol
@@ -77,9 +77,9 @@ contract LlamaCoreTest is LlamaTestSetup, Llama
   function _createAction() public returns (ActionInfo memory ac
     bytes memory data = abi.encodeCall(MockProtocol.pause, (tru
     vm.prank(actionCreatorAaron);
-    uint256 actionId = mpCore.createAction(uint8(Roles.ActionCr
+    uint256 actionId = mpCore.createAction(uint8(Roles.ActionCr
     actionInfo =
-      ActionInfo(actionId, actionCreatorAaron, uint8(Roles.Acti
+      ActionInfo(actionId, actionCreatorAaron, uint8(Roles.Acti
     vm.warp(block.timestamp + 1);
   }

@@ -107,7 +107,7 @@ contract LlamaCoreTest is LlamaTestSetup, Lla
   function _executeAction(ActionInfo memory actionInfo) public
     vm.expectEmit();
     emit ActionExecuted(actionInfo.id, address(this), actionInf
-    mpCore.executeAction(actionInfo);
+    mpCore.executeAction{value: actionInfo.value}(actionInfo);

     Action memory action = mpCore.getAction(actionInfo.id);
     assertEq(action.executed, true);

diff --git a/test/mock/MockProtocol.sol b/test/mock/MockProtocol
index 1636808..f6b0e0f 100644
--- a/test/mock/MockProtocol.sol
+++ b/test/mock/MockProtocol.sol
@@ -21,7 +21,7 @@ contract MockProtocol {
```

```
    return msg.value;
  }

-  function pause(bool isPaused) external onlyOwner {
+  function pause(bool isPaused) external payable onlyOwner {
    paused = isPaused;
  }
```

Now we can run any test that executes this action, for example:

```
forge test -m test_RevertIf_ActionExecuted
```

The test fails with “EvmError: OutOfFund”.

Recommended Mitigation Steps

It seems like an important part of protocol functionality that is not working, therefore suggested **High** severity.

The fix is straightforward, making LlamaExecutor.execute payable and passing value in LlamaCore:

```
diff --git a/src/LlamaCore.sol b/src/LlamaCore.sol
index 89d60de..05f1755 100644
--- a/src/LlamaCore.sol
+++ b/src/LlamaCore.sol
@@ -331,7 +331,7 @@ contract LlamaCore is Initializable {

     // Execute action.
     (bool success, bytes memory result) =
-      executor.execute(actionInfo.target, actionInfo.value, act
+      executor.execute{value: msg.value}(actionInfo.target, act

     if (!success) revert FailedActionExecution(result);

diff --git a/src/LlamaExecutor.sol b/src/LlamaExecutor.sol
index f92ebc0..fe7127e 100644
--- a/src/LlamaExecutor.sol
+++ b/src/LlamaExecutor.sol
@@ -28,6 +28,7 @@ contract LlamaExecutor {
   /// @return result The data returned by the function being ca
   function execute(address target, uint256 value, bool isScript
     external
```

```
+     payable
      returns (bool success, bytes memory result)
    {
      if (msg.sender != LLAMA_CORE) revert OnlyLlamaCore();
```

AustinGreen (Llama) confirmed and commented:

> This was resolved in this PR: https://github.com/llamaxyz/llama/pull/367 (note repo is currently private but will be made public before launch)

gzeon (Judge) reduced severity to Medium and commented:

> Valid issue, actions that require the executor to forward a call value would not work. However, fund is secure and not stuck since this does not impact the functionality of `LlamaAccount.transferNativeToken` which take the amount from calldata.

[M-02] User with disapproval role can gas grief the action executor

Submitted by **dirk_y**, *also found by* **rvierdiiev**

Because disapprovals can be cast after the minimum queue time has expired (i.e. the action is now executable), a user with the disapproval role can frontrun any execute calls to push the action into the disapproved state and cause the execute call to fail, hence gas griefing the execute caller. This is particularly easy to achieve if a user has a force disapproval role.

Proof of Concept

During calls to `castDisapproval` there is a call to `_preCastAssertions` which checks that the action is in a queued state. The purpose of this check is to ensure that disapprovals can only be cast after the action was first approved and then queued for execution.

However, the issue is that the action remains in the queue state even after the `minExecutionTime` has been passed. The result is that a malicious user can disapprove an action once it is ready to execute.

Below is a diff to the existing test suite that shows how an action that is ready to be executed could be disapproved just before execution. This isn't demonstrated with a force disapproval role, but that case would be the most harmful in terms of gas griefing.

```
diff --git a/test/LlamaCore.t.sol b/test/LlamaCore.t.sol
index 8135c93..34fd630 100644
--- a/test/LlamaCore.t.sol
+++ b/test/LlamaCore.t.sol
@@ -1015,8 +1015,12 @@ contract ExecuteAction is LlamaCoreTest {
     mpCore.queueAction(actionInfo);
     vm.warp(block.timestamp + 6 days);

-    vm.expectEmit();
-    emit ActionExecuted(0, address(this), mpStrategy1, actionCr
+    vm.prank(disapproverDave);
+    mpCore.castDisapproval(uint8(Roles.Disapprover), actionInfo
+    vm.prank(disapproverDrake);
+    mpCore.castDisapproval(uint8(Roles.Disapprover), actionInfo
+
+    vm.expectRevert();
     mpCore.executeAction(actionInfo);
   }
```

Recommended Mitigation Steps

I suggest that disapprovals should only be allowed to be cast whilst the timestamp is still less than the `minExecutionTime` of the action. Effectively there is a specified disapproval window. The following lines could be added to `_preCastAssertions`:

```
if (!isApproval) {
    require(block.timestamp < action.minExecutionTime, "Missed d
}
```

AustinGreen (Llama) confirmed and commented:

> We confirm this and are working on a fix. It is a duplicate of https://github.com/code-423n4/2023-06-llama-findings/issues/80

> Not sure if it should be medium or not but don't feel strongly. Llama is a trusted system so this would require malicious user intent or user error.

CoOnan (Warden) commented:

> This is more of an improved design than a security issue. `disapproval` role is a highly privileged role as per the design of the system.

> The `minExecutionTime` is meant to prevent someone from executing the action early but is not designed to prevent the `disApproval` role. Either he disapproved early or after `minExecutionTime` passed this doesn't break the logic of the function at all, it will be excepted to cancel the action in this case. I believe this is a valid QA.

AustinGreen (Llama) confirmed and commented:

> We removed the ability to disapprove after minExecutionTime to address this finding.

[M-03] LlamaPolicy could be DOS by creating large amount of actions

Submitted by ktg, *also found by* auditor0517, BLOS, Atree, Toshii, xuwinnie, *and* 0xnev

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L404-#L409
https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaCore.sol#L516-#L562

Proof of Concept

Currently, when Executor want to set role for a user, he call function `LlamaPolicy._setRoleHolder`, this in turn will first call function `_assertNoActionCreationsAtCurrentTimestamp`:

```
    /// @dev Because role supplies are not checkpointed for simplici
      /// if each of the below is executed within the same timestamp
```

```
//    1. An action is created that saves off the current role
//    2. A policyholder is given a new role.
//    3. Now the total supply in that block is different than
// As a result, we disallow changes to roles if an action was
function _assertNoActionCreationsAtCurrentTimestamp() internal
  if (llamaExecutor == address(0)) return; // Skip check durin
  address llamaCore = LlamaExecutor(llamaExecutor).LLAMA_CORE(
  uint256 lastActionCreation = LlamaCore(llamaCore).getLastAct
  if (lastActionCreation == block.timestamp) revert ActionCrea
}
```

As stated in the comment, the protocol disallows changes to roles if an action was created in the same block. However, function `LlamaCore._createAction` does not limit the number of actions a user could create. Consequently, a user with createAction role can DOS protocol's policy by creating large amount of actions. A user can create 24 * 3600 * 30 ~ 2.5 mils actions to DOS a system in a month, this is definitely a not too big number, especially when the protocol is deployed in low fee blockchains. (I notice that the folder `script` is organized as `script/input/{blockchainId}/*.json` so I assume that the protocol will be used across different blockchains).

This will prevents the revoking of expired roles, revoke policy,... because they all use `_setRoleHolder` function.

Below is a POC, for ease of testing, place this test case under file LlamaStrategy.t.sol, contract IsActionApproved:

```
function testDOSByCreatingManyAction() public {
    ILlamaStrategy testStrategy = deployTestStrategy();
    uint256 numberOfHolders = 10;
    generateAndSetRoleHolders(numberOfHolders);

    // create action
    bytes32 newPermissionId = keccak256(abi.encode(address(mockP
    vm.prank(address(mpExecutor));
    mpPolicy.setRolePermission(uint8(Roles.ActionCreator), newPe
    bytes memory data = abi.encodeCall(MockProtocol.pause, (true
    vm.prank(actionCreatorAaron);
    uint256 actionId = mpCore.createAction(uint8(Roles.ActionCre
    console.logUint(actionId);
    // revert if we try to set role
```

```
        vm.prank(address(mpExecutor));
        vm.expectRevert(LlamaPolicy.ActionCreationAtSameTimestamp.se
        mpPolicy.setRoleHolder(uint8(Roles.TestRole1), address(12345


        // Pass time
        vm.warp(block.timestamp + 1);

        // Create action again
        vm.prank(actionCreatorAaron);
        actionId = mpCore.createAction(uint8(Roles.ActionCreator), t
        console.logUint(actionId);
        // policy can't set role again
        vm.prank(address(mpExecutor));
        vm.expectRevert(LlamaPolicy.ActionCreationAtSameTimestamp.se
        mpPolicy.setRoleHolder(uint8(Roles.TestRole1), address(12345


    }
```

Recommended Mitigation Steps

I recommend limiting the number of active actions a user can create.

AustinGreen (Llama) confirmed and commented:

> We are tracking the issue here and deciding on a fix:
> https://github.com/llamaxyz/llama/issues/393

> This is a duplicate of https://github.com/code-423n4/2023-06-llama-findings/issues/209

gzeon (Judge) commented:

> Keeping this as M given this is a valid DOS vector and the cost to DOS is linear. There are EVM chains with low enough gas fee which can make this a feasible attack.

AustinGreen (Llama) commented:

> We addressed this issue by adding an additional checkpoint that strategies can use to get a (dis)approval role's total number of holders and total quantity in the past.

> This allowed us to remove the `_assertNoActionCreationsAtCurrentTimestamp` check.

Low Risk and Non-Critical Issues

For this audit, 9 reports were submitted by wardens detailing low risk and non-critical issues. The **report highlighted below** by **Rolezn** received the top score from the judge.

The following wardens also submitted reports: **libratus**, **0xSmartContract**, **QiuhaoLi**, **DavidGiladi**, **kutugu**, **Sathish9098**, **minhquanym**, *and* **matrix_0wl** .

Summary

Low Risk Issues

	Issue	Contexts
[L-01]	External calls in an un-bounded `for-` loop may result in a DOS	19
[L-02]	Missing Contract-existence Checks Before Low-level Calls	4
[L-03]	Protect `LlamaPolicy.sol` NFT from copying in POW forks	4
[L-04]	Unbounded loop	7
[L-05]	Inconsistent documentation to actual function logic	3

Total: 37 contexts over 5 issues

Non-critical Issues

	Issue	Contexts
[N-01]	Critical Changes Should Use Two-step Procedure	9
[N-02]	Large or complicated code bases should implement fuzzing tests	1
[N-03]	Initial value check is missing in Set Functions	9
[N-04]	Use @inheritdoc rather than using a non-standard annotation	55
[N-05]	Function name should contain `InitializeRoles` instead of `NewRoles`	1
[N-06]	Add to `blacklist` function	1

Total: 76 contexts over 6 issues

[L-01] External calls in an un-bounded `for-` loop may result in a DOS

Consider limiting the number of iterations in `for-` loops that make external calls.

Proof Of Concept

▶ Details

[L-02] Missing Contract-existence Checks Before Low-level Calls

Low-level calls return success if there is no code present at the specified address.

Proof Of Concept

▶ Details

Recommended Mitigation Steps

In addition to the zero-address checks, add a check to verify that `<address>.code.length > 0`

[L-03] Protect `LlamaPolicy.sol` NFT from copying in POW forks

Ethereum has performed the long-awaited “merge” that will dramatically reduce the environmental impact of the network

There may be forked versions of Ethereum, which could cause confusion and lead to scams as duplicated NFT assets enter the market.

If the Ethereum Merge, which took place in September 2022, results in the Blockchain splitting into two Blockchains due to the ‘THE DAO’ attack in 2016, this could result in duplication of immutable tokens (NFTs).

In any case, duplicate NFTs will exist due to the ETH proof-of-work chain and other potential forks, and there's likely to be some level of confusion around which assets are 'official' or 'authentic.'

Even so, there could be a frenzy for these copies, as NFT owners attempt to flip the proof-of-work versions of their valuable tokens.

As ETHPOW and any other forks spin off of the Ethereum mainnet, they will yield duplicate versions of Ethereum's NFTs. An NFT is simply a blockchain token, and it can work as a deed of ownership to digital items like artwork and collectibles. A forked Ethereum chain will thus have duplicated deeds that point to the same tokenURI.

About Merge Replay Attack:
https://twitter.com/elerium115/status/1558471934924431363?s=20&t=RRheaYJwo-GmSnePwofgag

Proof Of Concept

▶ Details

Recommended Mitigation Steps

Add the following check:

```
    if(block.chainid != 1) {
        revert();
    }
```

[L-04] Unbounded loop

New items are pushed into the following arrays but there is no option to `pop` them out. Currently, the array can grow indefinitely. E.g. there's no maximum limit and there's no functionality to remove array values.

If the array grows too large, calling relevant functions might run out of gas and revert. Calling these functions could result in a DOS condition.

Proof Of Concept

▶ Details

Recommended Mitigation Steps

Add a functionality to delete array values or add a maximum size limit for arrays.

[L-05] Inconsistent documentation to actual function logic

It is mentioned in documentation of the function `validateActionCreation` that the param `actionInfo` is used.

```
/// @notice Reverts if action creation is not allowed.
/// @param actionInfo Data required to create an action.
function validateActionCreation(ActionInfo calldata actionInfo
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/interfaces/ILlamaStrategy.sol#L33-L35

However, in `LlamaAbsoluteQuorum.sol` the param is commented out and is not used in the function.

```
function validateActionCreation(ActionInfo calldata /* actionI
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaAbsoluteQuorum.sol#L27

The same applies to `isApprovalEnabled` and `isDisapprovalEnabled`.

[N-01] Critical Changes Should Use Two-step Procedure

The critical procedures should be two step process.

See similar findings in previous Code4rena audits for reference: https://code4rena.com/reports/2022-06-illuminate/#2-critical-changes-should-use-two-step-procedure

Proof Of Concept

▶ Details

Recommended Mitigation Steps

Lack of two-step procedure for critical operations leaves them error-prone. Consider adding two step procedure on the critical functions.

[N-02] Large or complicated code bases should implement fuzzing tests

Large code bases, or code with lots of inline-assembly, complicated math, or complicated interactions between multiple contracts, should implement fuzzing tests. Fuzzers such as Echidna require the test writer to come up with invariants which should not be violated under any circumstances, and the fuzzer tests various inputs and function calls to ensure that the invariants always hold. Even code with 100% code coverage can still have bugs due to the order of the operations a user performs, and fuzzers, with properly and extensively-written invariants, can close this testing gap significantly.

Proof Of Concept

Various in-scope contract files.

[N-03] Initial value check is missing in Set Functions

A check regarding whether the current value and the new value are the same should be added.

Proof Of Concept

▶ Details

[N-04] Use @inheritdoc rather than using a non-standard annotation

Proof Of Concept

▶ Details

[N-05] Function name should contain `InitializeRoles` instead of `NewRoles`

The function `createNewStrategiesAndNewRolesAndSetRoleHoldersAndSetRolePermissions` should be `createNewStrategiesAndInitializeRolesAndSetRoleHoldersAndSetRolePermissions` as it calls `initializeRoles(description);`.

Similar to the function `createNewStrategiesAndInitializeRolesAndSetRoleHolders`

Proof Of Concept

```
function createNewStrategiesAndInitializeRolesAndSetRoleHolder
  CreateStrategies calldata _createStrategies,
  RoleDescription[] calldata description,
  RoleHolderData[] calldata _setRoleHolders
) external onlyDelegateCall {
  (LlamaCore core,) = _context();
  core.createStrategies(_createStrategies.llamaStrategyLogic,
  initializeRoles(description);
  setRoleHolders(_setRoleHolders);
}
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/llama-scripts/LlamaGovernanceScript.sol#L120-L129

```
function createNewStrategiesAndNewRolesAndSetRoleHoldersAndSetR
  CreateStrategies calldata _createStrategies,
  RoleDescription[] calldata description,
  RoleHolderData[] calldata _setRoleHolders,
  RolePermissionData[] calldata _setRolePermissions
) external onlyDelegateCall {
  (LlamaCore core,) = _context();
  core.createStrategies(_createStrategies.llamaStrategyLogic,
  initializeRoles(description);
  setRoleHolders(_setRoleHolders);
  setRolePermissions(_setRolePermissions);
```

```
    }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/llama-scripts/LlamaGovernanceScript.sol#L140-L151

[N-06] Add to `blacklist` function

It is noted that in this project: `LlamaPolicy.sol is an NFT.`

NFT thefts have increased recently, so with the addition of hacked NFTs to the platform, NFTs can be converted into liquidity. To prevent this, I recommend adding the blacklist function.

Marketplaces such as Opensea have a blacklist feature that will not list NFTs that have been reported theft, NFT projects such as Manifold have blacklist functions in their smart contracts.

Here is the project example; Manifold

Manifold Contract
https://etherscan.io/address/0xe4e4003afe3765aca8149a82fc064c0b125b9e5a#code

```
    modifier nonBlacklistRequired(address extension) {
        require(!_blacklistedExtensions.contains(extension), "E
        _;
    }
```

Recommended Mitigation Steps

Add to Blacklist function and modifier.

AustinGreen (Llama) commented:

> L-01: These external calls are to the internal Llama system so this finding is incorrect.
> L-03: We plan to deploy Llama on multiple EVM chains so this check would not make sense.

Gas Optimizations

For this audit, 17 reports were submitted by wardens detailing gas optimizations. The report highlighted below by **JCN** received the top score from the judge.

The following wardens also submitted reports: naman1778, 0xSmartContract, sebghatullah, SM3_SS, shamsulhaq123, hunter_w3b, SAQ, petrichor, Rageur, Raihan, SAAJ, lsaudit, DavidGiladi, Sathish9098, Rolezn, *and* VictoryGod .

Summary

A majority of the optimizations were benchmarked via the protocol's tests, i.e. using the following config: `solc version 0.8.17`, `optimizer on`, and `1300 runs`. Optimizations that were not benchmarked are explained via EVM gas costs and opcodes.

Note

- Only optimizations for state-mutating functions (i.e. non `view` / `pure`) and `view` / `pure` functions called within state-mutating functions have been highlighted below.
- Some code snippets may be truncated to save space. Code snippets may also be accompanied by @audit tags in comments to aid in explaining the issue.

Table of Contents

Number	Issue	Instances	Gas Saved
[G-05]	Forgo internal function to save 1 `STATICCALL`	4	400
[G-06]	Multiple accesses of a mapping/array should use a local variable cache	1	116
[G-07]	Refactor `If` / `require` statements to save SLOADs in case of early revert	4	-

Total Estimated Gas Saved: 29915

[G-01] State variables can be cached instead of re-reading them from storage

Caching of a state variable replaces each `Gwarmaccess (100 gas)` with a much cheaper stack read.

Total Instances: `5`

Estimated Gas Saved: `5 * 100 = 500`

Note: These are instances missed by the Automated Report.

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaCore.sol#L542-L559

Use already cached `actionId` to save 1 SLOAD

```
File: src/LlamaCore.sol
542:     actionId = actionsCount; // @audit: 1st sload
...
559:     actionsCount = LlamaUtils.uncheckedIncrement(actionsCoun
```

```
diff --git a/src/LlamaCore.sol b/src/LlamaCore.sol
index 89d60de..049f66d 100644
--- a/src/LlamaCore.sol
+++ b/src/LlamaCore.sol
```

```
@@ -556,7 +556,7 @@ contract LlamaCore is Initializable {
       newAction.isScript = authorizedScripts[target];
     }

-    actionsCount = LlamaUtils.uncheckedIncrement(actionsCount);
+    actionsCount = LlamaUtils.uncheckedIncrement(actionId); // 

     emit ActionCreated(actionId, policyholder, role, strategy, 
   }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaAbsolutePeerReview.sol#L56-L58

Cache `minDisapprovals` to save 1 SLOAD

```
File: src/strategies/LlamaAbsolutePeerReview.sol
56:       if (
57:         minDisapprovals != type(uint128).max // @audit: 1st s
58:           && minDisapprovals > disapprovalPolicySupply - acti
```

```
diff --git a/src/strategies/LlamaAbsolutePeerReview.sol b/src/st
index 85feb92..c8426aa 100644
--- a/src/strategies/LlamaAbsolutePeerReview.sol
+++ b/src/strategies/LlamaAbsolutePeerReview.sol
@@ -53,9 +53,10 @@ contract LlamaAbsolutePeerReview is LlamaAbso
       if (minApprovals > approvalPolicySupply - actionCreatorAp

       uint256 actionCreatorDisapprovalRoleQty = llamaPolicy.get
+      uint128 _minDisapprovals = minDisapprovals;
       if (
-        minDisapprovals != type(uint128).max
-          && minDisapprovals > disapprovalPolicySupply - action
+        _minDisapprovals != type(uint128).max
+          && _minDisapprovals > disapprovalPolicySupply - actio
       ) revert InsufficientDisapprovalQuantity();
     }
   }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaRelativeQuorum.sol#L220-L223

Cache `forceApprovalRole[role]` to save 1 SLOAD

```
File: src/strategies/LlamaRelativeQuorum.sol
220:   function getApprovalQuantityAt(address policyholder, uint8
221:     if (role != approvalRole && !forceApprovalRole[role]) re
222:     uint128 quantity = policy.getPastQuantity(policyholder,
223:     return quantity > 0 && forceApprovalRole[role] ? type(ui
```

```
diff --git a/src/strategies/LlamaRelativeQuorum.sol b/src/strate
index d796ae9..8d74c92 100644
--- a/src/strategies/LlamaRelativeQuorum.sol
+++ b/src/strategies/LlamaRelativeQuorum.sol
@@ -218,9 +218,10 @@ contract LlamaRelativeQuorum is ILlamaStrat

    /// @inheritdoc ILlamaStrategy
    function getApprovalQuantityAt(address policyholder, uint8 ro
-     if (role != approvalRole && !forceApprovalRole[role]) retur
+     bool _forceApprovalRole = forceApprovalRole[role];
+     if (role != approvalRole && !_forceApprovalRole) return 0;
      uint128 quantity = policy.getPastQuantity(policyholder, rol
-     return quantity > 0 && forceApprovalRole[role] ? type(uint1
+     return quantity > 0 && _forceApprovalRole ? type(uint128).m
    }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaRelativeQuorum.sol#L240-L242

Cache `forceDisapprovalRole[role]` to save 1 SLOAD

```
File: src/strategies/LlamaRelativeQuorum.sol
240:     if (role != disapprovalRole && !forceDisapprovalRole[rol
241:     uint128 quantity = policy.getPastQuantity(policyholder,
242:     return quantity > 0 && forceDisapprovalRole[role] ? type
```

```
diff --git a/src/strategies/LlamaRelativeQuorum.sol b/src/strate
index d796ae9..e1c3927 100644
--- a/src/strategies/LlamaRelativeQuorum.sol
+++ b/src/strategies/LlamaRelativeQuorum.sol
```

```
@@ -237,9 +237,10 @@ contract LlamaRelativeQuorum is ILlamaStrat
     view
     returns (uint128)
   {
-    if (role != disapprovalRole && !forceDisapprovalRole[role])
+    bool _forceDisapprovalRole = forceDisapprovalRole[role];
+    if (role != disapprovalRole && !_forceDisapprovalRole) retu
     uint128 quantity = policy.getPastQuantity(policyholder, rol
-    return quantity > 0 && forceDisapprovalRole[role] ? type(ui
+    return quantity > 0 && _forceDisapprovalRole ? type(uint128
   }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaFactory.sol#L260-L263

Cache `llamaCount` to save 1 SLOAD

```
File: src/LlamaFactory.sol
260:     emit LlamaInstanceCreated(
261:       llamaCount, name, address(llamaCore), address(llamaExe
262:     );
263:     llamaCount = LlamaUtils.uncheckedIncrement(llamaCount);
```

```
diff --git a/src/LlamaFactory.sol b/src/LlamaFactory.sol
index 0cc4cfd..269e4cb 100644
--- a/src/LlamaFactory.sol
+++ b/src/LlamaFactory.sol
@@ -256,11 +256,12 @@ contract LlamaFactory {
     llamaExecutor = llamaCore.executor();

     policy.finalizeInitialization(address(llamaExecutor), boots
-
+
+    uint256 _llamaCount = llamaCount;
     emit LlamaInstanceCreated(
-      llamaCount, name, address(llamaCore), address(llamaExecut
+      _llamaCount, name, address(llamaCore), address(llamaExecu
     );
-    llamaCount = LlamaUtils.uncheckedIncrement(llamaCount);
+    llamaCount = LlamaUtils.uncheckedIncrement(_llamaCount);
   }
```

[G-02] Cache state variables outside of loop to avoid reading/writing storage on every iteration

Reading from storage should always try to be avoided within loops. In the following instances, we are able to cache state variables outside of the loop to save a `Gwarmaccess (100 gas)` per loop iteration. In addition, for some instances we are also able to increment the cached variable in the loop and update the storage variable outside the loop to save 1 `SSTORE` per loop iteration.

Total Instances: `3`

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L227

Gas Savings for `LlamaPolicy.revokePolicy` *, obtained via protocol's tests: Avg 724 gas*

	Med	Max	Avg	# calls
Before	69040	110067	59837	11
Before	68180	108992	59113	11

Cache `numRoles` outside of loop to save 1 SLOAD per iteration

```
File: src/LlamaPolicy.sol
227     for (uint256 i = 1; i <= numRoles; i = LlamaUtils.uncheck
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
index 3fca63e..7e47189 100644
--- a/src/LlamaPolicy.sol
+++ b/src/LlamaPolicy.sol
@@ -224,7 +224,8 @@ contract LlamaPolicy is ERC721NonTransferabl
      // We start from i = 1 here because a value of zero is rese
      // that will get removed automatically when the token is bu
      // the last role is also revoked.
-     for (uint256 i = 1; i <= numRoles; i = LlamaUtils.unchecked
+     uint8 _numRoles = numRoles;
+     for (uint256 i = 1; i <= _numRoles; i = LlamaUtils.unchecke
```

```
        if (hasRole(policyholder, uint8(i))) _setRoleHolder(uint8
      }
      _burn(_tokenId(policyholder));
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L151-L168

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L393-L396

Gas Savings for `LlamaFactory.deploy` *, obtained via protocol's tests: Avg 4468 gas*

	Med	Max	Avg	# calls
Before	5101157	5406425	5015882	412
After	5096893	5281811	5011414	412

To benchmark this instance we will bring the logic from `_initializeRole` *into the construtor in order to refactor the logic. Note that another way of achieving this is by refactoring the logic of the* `_initializeRole` *directly and every other function that calls* `_initializeRole` *.*

Cache `numRoles` outside loop, increment cached variable in loop, and update storage outside loop to save 2 SLOADs + 1 SSTORE per iteration

```
File: src/LlamaPolicy.sol
151:     for (uint256 i = 0; i < roleDescriptions.length; i = Lla
152:       _initializeRole(roleDescriptions[i]); // @audit: sload
153:     }
...
168:     if (numRoles == 0 || getRoleSupplyAsNumberOfHolders(ALL_

393:   function _initializeRole(RoleDescription description) inte
394:     numRoles += 1; // @audit: sload + sstore for `numRoles`
395:     emit RoleInitialized(numRoles, description); // @audit:
   }
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
```

```
index 3fca63e..af2129b 100644
--- a/src/LlamaPolicy.sol
+++ b/src/LlamaPolicy.sol
@@ -148,9 +148,12 @@ contract LlamaPolicy is ERC721NonTransferab
   ) external initializer {
     __initializeERC721MinimalProxy(_name, string.concat("LL-", 
     factory = LlamaFactory(msg.sender);
+    uint8 _numRoles = numRoles;
     for (uint256 i = 0; i < roleDescriptions.length; i = LlamaU
-      _initializeRole(roleDescriptions[i]);
+      _numRoles += 1;
+      emit RoleInitialized(_numRoles, roleDescriptions[i]);
     }
+    numRoles = _numRoles;

     for (uint256 i = 0; i < roleHolders.length; i = LlamaUtils.
       _setRoleHolder(
@@ -165,7 +168,7 @@ contract LlamaPolicy is ERC721NonTransferabl
     // Must have assigned roles during initialization, otherwis
     // we do not check that roles were assigned "properly" as t
     // this is more of a sanity check, not a guarantee that the
-    if (numRoles == 0 || getRoleSupplyAsNumberOfHolders(ALL_HOL
+    if (_numRoles == 0 || getRoleSupplyAsNumberOfHolders(ALL_HO
   }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L161-L163

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L490-L491

Gas Savings for `LlamaFactory.deploy` *, obtained via protocol's tests: Avg 677 gas*

	Med	Max	Avg	# calls
Before	5101157	5406425	5015882	412
After	5101175	5120119	5015205	412

To benchmark this instance we will refactor the logic of the `_setRolePermission` *internal function directly and also refactor every other function that calls* `_setRolePermission` *. Another way of achieving this would be to move the logic of* `_setRolePermission` *into the construtor and refactoring it there.*

Cache `numRoles` outside loop to save 1 SLOAD per iteration

```
File: src/LlamaPolicy.sol
161:     for (uint256 i = 0; i < rolePermissions.length; i = Llama
162:       _setRolePermission(rolePermissions[i].role, rolePermis
163:     }

490:   function _setRolePermission(uint8 role, bytes32 permission
491:     if (role > numRoles) revert RoleNotInitialized(role); //
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
index 3fca63e..8a3273a 100644
--- a/src/LlamaPolicy.sol
+++ b/src/LlamaPolicy.sol
@@ -157,15 +157,16 @@ contract LlamaPolicy is ERC721NonTransferal
          roleHolders[i].role, roleHolders[i].policyholder, roleH
       );
     }
-
+
+    uint8 _numRoles = numRoles;
     for (uint256 i = 0; i < rolePermissions.length; i = LlamaUt
-      _setRolePermission(rolePermissions[i].role, rolePermissio
+      _setRolePermission(_numRoles, rolePermissions[i].role, ro
     }

     // Must have assigned roles during initialization, otherwis
     // we do not check that roles were assigned "properly" as t
     // this is more of a sanity check, not a guarantee that the
-    if (numRoles == 0 || getRoleSupplyAsNumberOfHolders(ALL_HOL
+    if (_numRoles == 0 || getRoleSupplyAsNumberOfHolders(ALL_HO
   }

   // ============================================
@@ -181,7 +182,7 @@ contract LlamaPolicy is ERC721NonTransferabl
     if (llamaExecutor != address(0)) revert AlreadyInitialized(

     llamaExecutor = _llamaExecutor;
-    _setRolePermission(BOOTSTRAP_ROLE, bootstrapPermissionId, t
+    _setRolePermission(numRoles, BOOTSTRAP_ROLE, bootstrapPermi
   }

   // -------- Role and Permission Management -------
```

```
@@ -205,7 +206,7 @@ contract LlamaPolicy is ERC721NonTransferable
    /// @param permissionId Permission ID to assign to the role.
    /// @param hasPermission Whether to assign the permission or
    function setRolePermission(uint8 role, bytes32 permissionId,
-     _setRolePermission(role, permissionId, hasPermission);
+     _setRolePermission(numRoles, role, permissionId, hasPermiss
    }

    /// @notice Revokes a policyholder's expired role.
@@ -487,8 +488,8 @@ contract LlamaPolicy is ERC721NonTransferable
    }

    /// @dev Sets a role's permission along with whether that per
-   function _setRolePermission(uint8 role, bytes32 permissionId,
-     if (role > numRoles) revert RoleNotInitialized(role);
+   function _setRolePermission(uint8 _numRoles, uint8 role, byte
+     if (role > _numRoles) revert RoleNotInitialized(role);
      canCreateAction[role][permissionId] = hasPermission;
      emit RolePermissionAssigned(role, permissionId, hasPermissi
    }
```

[G-03] Multiple address mappings can be combined into a single mapping of an address to a struct, where appropriate

We can combine multiple mappings below into structs. This will result in cheaper storage reads since multiple mappings are accessed in functions and those values are now occupying the same storage slot, meaning the slot will become warm after the first SLOAD. In addition, when writing to and reading from the struct values we will avoid a `Gsset (20000 gas)` and `Gcoldsload (2100 gas)` since multiple struct values are now occupying the same slot.

Note: This instance was missed by the automated report.

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaRelativeQuorum.sol#L130-L133

Gas Savings for `LlamaCore.executeAction`*, obtained via protocol's tests: Avg 21838 gas*

	Med	Max	Avg	# calls
Before	5186172	23819570	4807541	430
After	5164334	32081589	4803180	430

```
File: src/strategies/LlamaRelativeQuorum.sol
130:   mapping(uint8 => bool) public forceApprovalRole;
131:
132:   /// @notice Mapping of roles that can force an action to b
133:   mapping(uint8 => bool) public forceDisapprovalRole;
```

```
diff --git a/src/strategies/LlamaRelativeQuorum.sol b/src/strate
index d796ae9..2cbeb0c 100644
--- a/src/strategies/LlamaRelativeQuorum.sol
+++ b/src/strategies/LlamaRelativeQuorum.sol
@@ -125,12 +125,13 @@ contract LlamaRelativeQuorum is ILlamaStra

    /// @notice The role that can disapprove an action.
    uint8 public disapprovalRole;
+
+   struct ForceRoles {
+     bool forceApprovalRole;
+     bool forceDisapprovalRole;
+   }

-   /// @notice Mapping of roles that can force an action to be a
-   mapping(uint8 => bool) public forceApprovalRole;
-
-   /// @notice Mapping of roles that can force an action to be d
-   mapping(uint8 => bool) public forceDisapprovalRole;
+   mapping(uint8 => ForceRoles) forceRoles;

    /// @notice Mapping of action ID to the supply of the approva
    mapping(uint256 => uint256) public actionApprovalSupply;
@@ -146,6 +147,15 @@ contract LlamaRelativeQuorum is ILlamaStrat
      _disableInitializers();
    }

+   // @audit: Getters used for benchmarking purposes
+   function forceApprovalRole(uint8 role) external view returns
+     return forceRoles[role].forceApprovalRole;
+   }
+
```

```
+  function forceDisapprovalRole(uint8 role) external view retur
+    return forceRoles[role].forceDisapprovalRole;
+  }
+
   // ========================================
   // ======== Interface Implementation ========
   // ========================================
@@ -178,7 +188,7 @@ contract LlamaRelativeQuorum is ILlamaStrate
       uint8 role = strategyConfig.forceApprovalRoles[i];
       if (role == 0) revert InvalidRole(0);
       _assertValidRole(role, numRoles);
-      forceApprovalRole[role] = true;
+      forceRoles[role].forceApprovalRole = true;
       emit ForceApprovalRoleAdded(role);
     }

@@ -186,7 +196,7 @@ contract LlamaRelativeQuorum is ILlamaStrate
       uint8 role = strategyConfig.forceDisapprovalRoles[i];
       if (role == 0) revert InvalidRole(0);
       _assertValidRole(role, numRoles);
-      forceDisapprovalRole[role] = true;
+      forceRoles[role].forceDisapprovalRole = true;
       emit ForceDisapprovalRoleAdded(role);
     }

@@ -213,14 +223,14 @@ contract LlamaRelativeQuorum is ILlamaStra

   /// @inheritdoc ILlamaStrategy
   function isApprovalEnabled(ActionInfo calldata, address, uint
-    if (role != approvalRole && !forceApprovalRole[role]) rever
+    if (role != approvalRole && !forceRoles[role].forceApproval
   }

   /// @inheritdoc ILlamaStrategy
   function getApprovalQuantityAt(address policyholder, uint8 ro
-    if (role != approvalRole && !forceApprovalRole[role]) retur
+    if (role != approvalRole && !forceRoles[role].forceApproval
     uint128 quantity = policy.getPastQuantity(policyholder, rol
-    return quantity > 0 && forceApprovalRole[role] ? type(uint1
+    return quantity > 0 && forceRoles[role].forceApprovalRole ?
   }

   // -------- When Casting Disapproval --------
@@ -228,7 +238,7 @@ contract LlamaRelativeQuorum is ILlamaStrate
   /// @inheritdoc ILlamaStrategy
   function isDisapprovalEnabled(ActionInfo calldata, address, u
```

```
      if (minDisapprovalPct > ONE_HUNDRED_IN_BPS) revert Disappro
-     if (role != disapprovalRole && !forceDisapprovalRole[role])
+     if (role != disapprovalRole && !forceRoles[role].forceDisap
    }

    /// @inheritdoc ILlamaStrategy
@@ -237,9 +247,9 @@ contract LlamaRelativeQuorum is ILlamaStrate
      view
      returns (uint128)
    {
-     if (role != disapprovalRole && !forceDisapprovalRole[role])
+     if (role != disapprovalRole && !forceRoles[role].forceDisap
      uint128 quantity = policy.getPastQuantity(policyholder, rol
-     return quantity > 0 && forceDisapprovalRole[role] ? type(ui
+     return quantity > 0 && forceRoles[role].forceDisapprovalRol
    }

    // -------- When Queueing --------
```

[G-04] Cache calldata/memory pointers for complex types to avoid offset calculations

The function parameters in the following instances are complex types (i.e. arrays which contain structs) and thus will result in more complex offset calculations to retrieve specific data from calldata/memory. We can avoid peforming some of these offset calculations by instantiating calldata/memory pointers.

Total Instances: `2`

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L155-L159

Gas Savings for `LlamaPolicy.deploy` *, obtained via protocol's tests: Avg 484 gas*

	Med	Max	Avg	# calls
Before	5101157	5406425	5015882	412
After	5101034	5256589	5015398	412

File: src/LlamaPolicy.sol

```
155:     for (uint256 i = 0; i < roleHolders.length; i = LlamaUti
156:       _setRoleHolder(
157:         roleHolders[i].role, roleHolders[i].policyholder, ro
158:       );
159:     }
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
index 3fca63e..b46c68e 100644
--- a/src/LlamaPolicy.sol
+++ b/src/LlamaPolicy.sol
@@ -153,8 +153,9 @@ contract LlamaPolicy is ERC721NonTransferabl
     }

     for (uint256 i = 0; i < roleHolders.length; i = LlamaUtils.
+      RoleHolderData calldata roleHolder = roleHolders[i];
       _setRoleHolder(
-        roleHolders[i].role, roleHolders[i].policyholder, roleH
+        roleHolder.role, roleHolder.policyholder, roleHolder.qu
       );
     }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L161-L163

Gas Savings for `LlamaPolicy.deploy` *, obtained via protocol's tests: Avg 708 gas*

	Med	Max	Avg	# calls	
Before	5101157	5406425	5015882	412	
After	5101157	5116924	5015174	412	

```
File: src/LlamaPolicy.sol
161:     for (uint256 i = 0; i < rolePermissions.length; i = Llam
162:       _setRolePermission(rolePermissions[i].role, rolePermis
163:     }
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
index 3fca63e..c6df227 100644
--- a/src/LlamaPolicy.sol
```

```
+++ b/src/LlamaPolicy.sol
@@ -159,7 +159,8 @@ contract LlamaPolicy is ERC721NonTransferabl
     }

     for (uint256 i = 0; i < rolePermissions.length; i = LlamaUt
-      _setRolePermission(rolePermissions[i].role, rolePermissio
+      RolePermissionData calldata rolePermission = rolePermissi
+      _setRolePermission(rolePermission.role, rolePermission.pe
     }
```

[G-05] Forgo internal function to save 1 `STATICCALL`

The `_context` internal function performs two external calls and returns both of the return values from those calls. Certain functions invoke `_context` but only use the return value from the first external call, thus performing an unnecessary extra external call. We can forgo using the internal function and instead only perform our desired external call to save 1 `STATICCALL (100 gas)`.

Total Instances: `4`

Estimated Gas Saved: `4 * 100 = 400`

https://github.com/code-423n4/2023-06-llama/blob/main/src/llama-scripts/LlamaGovernanceScript.sol#L111-L115

Only perform `address(this).LLAMA_CORE()` to save 1 `STATICCALL`

```
File: src/llama-scripts/LlamaGovernanceScript.sol
111:  function createNewStrategiesAndSetRoleHolders(
112:    CreateStrategies calldata _createStrategies,
113:    RoleHolderData[] calldata _setRoleHolders
114:  ) external onlyDelegateCall {
115:    (LlamaCore core,) = _context(); // @audit: return value
```

```
diff --git a/src/llama-scripts/LlamaGovernanceScript.sol b/src/l
index 820872e..f886bf7 100644
--- a/src/llama-scripts/LlamaGovernanceScript.sol
+++ b/src/llama-scripts/LlamaGovernanceScript.sol
```

```
@@ -112,7 +112,7 @@ contract LlamaGovernanceScript is LlamaBaseS
     CreateStrategies calldata _createStrategies,
     RoleHolderData[] calldata _setRoleHolders
   ) external onlyDelegateCall {
-    (LlamaCore core,) = _context();
+    LlamaCore core = LlamaCore(LlamaExecutor(address(this)).LLA
     core.createStrategies(_createStrategies.llamaStrategyLogic,
     setRoleHolders(_setRoleHolders);
   }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/llama-scripts/LlamaGovernanceScript.sol#L120-L125

Only perform `address(this).LLAMA_CORE()` to save 1 `STATICCALL`

```
File: src/llama-scripts/LlamaGovernanceScript.sol
120:   function createNewStrategiesAndInitializeRolesAndSetRoleHo
121:     CreateStrategies calldata _createStrategies,
122:     RoleDescription[] calldata description,
123:     RoleHolderData[] calldata _setRoleHolders
124:   ) external onlyDelegateCall {
125:     (LlamaCore core,) = _context(); // @audit: return value
```

```
diff --git a/src/llama-scripts/LlamaGovernanceScript.sol b/src/l
index 820872e..f886bf7 100644
--- a/src/llama-scripts/LlamaGovernanceScript.sol
+++ b/src/llama-scripts/LlamaGovernanceScript.sol
@@ -122,7 +122,7 @@ contract LlamaGovernanceScript is LlamaBaseS
     RoleDescription[] calldata description,
     RoleHolderData[] calldata _setRoleHolders
   ) external onlyDelegateCall {
-    (LlamaCore core,) = _context();
+    LlamaCore core = LlamaCore(LlamaExecutor(address(this)).LLA
     core.createStrategies(_createStrategies.llamaStrategyLogic,
     initializeRoles(description);
     setRoleHolders(_setRoleHolders);
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/llama-scripts/LlamaGovernanceScript.sol#L131-L135

Only perform `address(this).LLAMA_CORE()` to save 1 `STATICCALL`

```
File: src/llama-scripts/LlamaGovernanceScript.sol
131:   function createNewStrategiesAndSetRolePermissions(
132:     CreateStrategies calldata _createStrategies,
133:     RolePermissionData[] calldata _setRolePermissions
134:   ) external onlyDelegateCall {
135:     (LlamaCore core,) = _context(); // @audit: return value
```

```
diff --git a/src/llama-scripts/LlamaGovernanceScript.sol b/src/l
index 820872e..f886bf7 100644
--- a/src/llama-scripts/LlamaGovernanceScript.sol
+++ b/src/llama-scripts/LlamaGovernanceScript.sol
@@ -132,7 +132,7 @@ contract LlamaGovernanceScript is LlamaBaseS
     CreateStrategies calldata _createStrategies,
     RolePermissionData[] calldata _setRolePermissions
   ) external onlyDelegateCall {
-    (LlamaCore core,) = _context();
+    LlamaCore core = LlamaCore(LlamaExecutor(address(this)).LLA
     core.createStrategies(_createStrategies.llamaStrategyLogic,
     setRolePermissions(_setRolePermissions);
   }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/llama-scripts/LlamaGovernanceScript.sol#L140-L146

Only perform `address(this).LLAMA_CORE()` to save 1 `STATICCALL`

```
File: src/llama-scripts/LlamaGovernanceScript.sol
140:   function createNewStrategiesAndNewRolesAndSetRoleHoldersAn
141:     CreateStrategies calldata _createStrategies,
142:     RoleDescription[] calldata description,
143:     RoleHolderData[] calldata _setRoleHolders,
144:     RolePermissionData[] calldata _setRolePermissions
145:   ) external onlyDelegateCall {
146:     (LlamaCore core,) = _context(); // @audit: return value
```

```
diff --git a/src/llama-scripts/LlamaGovernanceScript.sol b/src/l
```

```
index 820872e..f886bf7 100644
--- a/src/llama-scripts/LlamaGovernanceScript.sol
+++ b/src/llama-scripts/LlamaGovernanceScript.sol
@@ -143,7 +143,7 @@ contract LlamaGovernanceScript is LlamaBaseS
     RoleHolderData[] calldata _setRoleHolders,
     RolePermissionData[] calldata _setRolePermissions
   ) external onlyDelegateCall {
-    (LlamaCore core,) = _context();
+    LlamaCore core = LlamaCore(LlamaExecutor(address(this)).LLA
     core.createStrategies(_createStrategies.llamaStrategyLogic,
     initializeRoles(description);
     setRoleHolders(_setRoleHolders);
```

[G-06] Multiple accesses of a mapping/array should use a local variable cache

Caching a mapping's value in a storage pointer when the value is accessed multiple times saves ~40 gas per access due to not having to perform the same offset calculation every time. Help the Optimizer by saving a storage variable's reference instead of repeatedly fetching it.

To achieve this, declare a storage pointer for the variable and use it instead of repeatedly fetching the reference in a map or an array. As an example, instead of repeatedly calling `stakes[tokenId_]`, save its reference via a storage pointer: `StakeInfo storage stakeInfo = stakes[tokenId_]` and use the pointer instead.

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L443-L448

Gas Savings for `LlamaPolicy.revokePolicy`*, obtained via protocol's tests: Avg 116 gas*

	Med	Max	Avg	# calls
Before	69040	110067	59837	11
After	68916	109912	59721	11

```
File: src/LlamaPolicy.sol
443:     uint128 initialQuantity = roleBalanceCkpts[tokenId][role
```

```
444:     bool hadRole = initialQuantity > 0;
445:     bool willHaveRole = quantity > 0;
446:
447:     // Now we update the policyholder's role balance checkpo
448:     roleBalanceCkpts[tokenId][role].push(willHaveRole ? quan
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
index 3fca63e..7674061 100644
--- a/src/LlamaPolicy.sol
+++ b/src/LlamaPolicy.sol
@@ -440,12 +440,13 @@ contract LlamaPolicy is ERC721NonTransferal
      // checking if the quantity is nonzero, and we don't need t
      // the `hadRole` and `willHaveRole` variables.
      uint256 tokenId = _tokenId(policyholder);
-     uint128 initialQuantity = roleBalanceCkpts[tokenId][role].l
+     Checkpoints.History storage _roleBalanceCkpts = roleBalance
+     uint128 initialQuantity = _roleBalanceCkpts.latest();
      bool hadRole = initialQuantity > 0;
      bool willHaveRole = quantity > 0;

      // Now we update the policyholder's role balance checkpoint
-     roleBalanceCkpts[tokenId][role].push(willHaveRole ? quantit
+     _roleBalanceCkpts.push(willHaveRole ? quantity : 0, expirat

      // If they don't hold a policy, we mint one for them. This
      // and 0 expiration, a policy is still minted even though t
```

[G-07] Refactor `If` / `require` statements to save SLOADs in case of early revert

Checks that involve calldata should come before checks that involve state variables, function calls, and calculations. By doing these checks first, the function is able to revert before using excessive gas in a call that may ultimately revert in an unhappy case.

Total Instances: `4`

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaAbsoluteQuorum.sol#L27-L35

The check in line 35 performs an SLOAD, while the check in lines 32-33 perform an external call and two SLOADs. We can move the check in line 35 above lines 32-33 to potentially save an SLOAD & External call in the unhappy path.

Note: This view function is called in the state mutating `_createAction` *function in* `LlamaCore.sol`

```
File: src/strategies/LlamaAbsoluteQuorum.sol
27:   function validateActionCreation(ActionInfo calldata /* acti
28:     LlamaPolicy llamaPolicy = policy; // Reduce SLOADs.
29:     uint256 approvalPolicySupply = llamaPolicy.getRoleSupplyA
30:     if (approvalPolicySupply == 0) revert RoleHasZeroSupply(a
31:
32:     uint256 disapprovalPolicySupply = llamaPolicy.getRoleSupp
33:     if (disapprovalPolicySupply == 0) revert RoleHasZeroSuppl
34:
35:     if (minApprovals > approvalPolicySupply) revert Insuffici
```

```
diff --git a/src/strategies/LlamaAbsoluteQuorum.sol b/src/strate
index 66130c0..aee2ce3 100644
--- a/src/strategies/LlamaAbsoluteQuorum.sol
+++ b/src/strategies/LlamaAbsoluteQuorum.sol
@@ -29,10 +29,11 @@ contract LlamaAbsoluteQuorum is LlamaAbsolut
     uint256 approvalPolicySupply = llamaPolicy.getRoleSupplyAsQ
     if (approvalPolicySupply == 0) revert RoleHasZeroSupply(app

+    if (minApprovals > approvalPolicySupply) revert Insufficien
+
     uint256 disapprovalPolicySupply = llamaPolicy.getRoleSupply
     if (disapprovalPolicySupply == 0) revert RoleHasZeroSupply(

-    if (minApprovals > approvalPolicySupply) revert Insufficien
     if (minDisapprovals != type(uint128).max && minDisapprovals
       revert InsufficientDisapprovalQuantity();
     }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/strategies/LlamaAbsolutePeerReview.sol#L74-L82

The check in line 79 accesses storage, while the check in line 80 only accesses calldata. Move the check in line 80 above line 79 to potentially save an SLOAD in the unhappy path.

Note: This view function is called in the state mutating `_preCastAssertions` *function in* `LlamaCore.sol`

```
File: src/strategies/LlamaAbsolutePeerReview.sol
74:   function isDisapprovalEnabled(ActionInfo calldata actionInf
75:     external
76:     view
77:     override
78:   {
79:     if (minDisapprovals == type(uint128).max) revert Disappro
80:     if (actionInfo.creator == policyholder) revert ActionCrea
81:     if (role != disapprovalRole && !forceDisapprovalRole[role
82:   }
```

```
diff --git a/src/strategies/LlamaAbsolutePeerReview.sol b/src/st
index 85feb92..2df24ec 100644
--- a/src/strategies/LlamaAbsolutePeerReview.sol
+++ b/src/strategies/LlamaAbsolutePeerReview.sol
@@ -76,8 +76,9 @@ contract LlamaAbsolutePeerReview is LlamaAbsol
     view
     override
   {
-    if (minDisapprovals == type(uint128).max) revert Disapprova
     if (actionInfo.creator == policyholder) revert ActionCreato
+    if (minDisapprovals == type(uint128).max) revert Disapprova
     if (role != disapprovalRole && !forceDisapprovalRole[role])
   }
 }
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaPolicy.sol#L412-L418

The check in line 414 accesses storage, while the check in line 418 only accesses a stack variable. Move the check in line 418 above line 414 to potentially save 1 SLOAD on the unhappy path.

Note: This view function is called within state mutating functions in `LlamaPolicy.sol`.

```
File: src/LlamaPolicy.sol
412:  function _assertValidRoleHolderUpdate(uint8 role, uint128 
413:    // Ensure role is initialized.
414:    if (role > numRoles) revert RoleNotInitialized(role); //
415:
416:    // Cannot set the ALL_HOLDERS_ROLE because this is handl
417:    // create duplicate entries if set here.
418:    if (role == ALL_HOLDERS_ROLE) revert AllHoldersRole(); /
```

```
diff --git a/src/LlamaPolicy.sol b/src/LlamaPolicy.sol
index 3fca63e..443e74c 100644
--- a/src/LlamaPolicy.sol
+++ b/src/LlamaPolicy.sol
@@ -410,13 +410,13 @@ contract LlamaPolicy is ERC721NonTransferal

   /// @dev Checks if the conditions are met for a `role` to be 
   function _assertValidRoleHolderUpdate(uint8 role, uint128 qua
-    // Ensure role is initialized.
-    if (role > numRoles) revert RoleNotInitialized(role);
-
     // Cannot set the ALL_HOLDERS_ROLE because this is handled 
     // create duplicate entries if set here.
     if (role == ALL_HOLDERS_ROLE) revert AllHoldersRole();

+    // Ensure role is initialized.
+    if (role > numRoles) revert RoleNotInitialized(role);
+
     // An expiration of zero is only allowed if the role is bei
     // the quantity is zero. In other words, the relationships 
     // quantity and expiration fields are:
```

https://github.com/code-423n4/2023-06-llama/blob/main/src/LlamaCore.sol#L317-L324

The check in line 324 accesses calldata, the check in line 323 accesses storage, and the check in lines 320-322 accesses storage at least once and potentially multiple times. To save at least one SLOAD in unhappy path, place the checks in the following order:

1. Check in line 324
2. Check in line 323
3. Check in lines 320-322

```
File: src/LlamaCore.sol
317:  function executeAction(ActionInfo calldata actionInfo) ext
318:    // Initial checks that action is ready to execute.
319:    Action storage action = actions[actionInfo.id];
320:    ActionState currentState = getActionState(actionInfo); /
321:
322:    if (currentState != ActionState.Queued) revert InvalidAc
323:    if (block.timestamp < action.minExecutionTime) revert Mi
324:    if (msg.value != actionInfo.value) revert IncorrectMsgVa
```

```
diff --git a/src/LlamaCore.sol b/src/LlamaCore.sol
index 89d60de..594e9f4 100644
--- a/src/LlamaCore.sol
+++ b/src/LlamaCore.sol
@@ -316,12 +316,13 @@ contract LlamaCore is Initializable {
   /// @param actionInfo Data required to create an action.
   function executeAction(ActionInfo calldata actionInfo) externa
     // Initial checks that action is ready to execute.
+    if (msg.value != actionInfo.value) revert IncorrectMsgValue
+
     Action storage action = actions[actionInfo.id];
-    ActionState currentState = getActionState(actionInfo);
+    if (block.timestamp < action.minExecutionTime) revert MinExe

+    ActionState currentState = getActionState(actionInfo);
     if (currentState != ActionState.Queued) revert InvalidActio
-    if (block.timestamp < action.minExecutionTime) revert MinExe
-    if (msg.value != actionInfo.value) revert IncorrectMsgValue

     action.executed = true;
```

Audit Analysis

For this audit, 13 analysis reports were submitted by wardens. An analysis report examines the codebase as a whole, providing observations and advice on such topics

as architecture, mechanism, or approach. The report highlighted below by **0xnev** received the top score from the judge.

The following wardens also submitted reports: peanuts, dirk_y, 0xSmartContract, joestakey, libratus, QiuhaoLi, K42, ktg, mahdirostami, kutugu, xuwinnie, neko_nyaa, *and* VictoryGod .

1. Analysis of Codebase

The Llama governance system provides a unique way for protocol to leverage policies (represented by a non-transferable NFT) to permission action creation till execution. It primarily focuses on 2 mechanisms, Action creation and policy management. To summarize the protocol, here is a step-by-step flow:

1. Protocol owners give policy and set roles (via `_setRoleHolder()`)
2. Protocol owner set permissions (via `_setRolePermissions()`)
3. Permissioned policy holders can create actions (via `createAction/createActionBySig`)
4. Strategy and custom guards validate action creation, if passes action can be queued (via Strategy and Guard function `validateActionCreation()`)
5. Policy holders with approval/disapproval cast votes during approval period (via `castApproval()/castDisapproval()`)
6. Strategies validate approval/disapproval against minimum thresholds `via isActionApproved()/isActionDisapproved()`
7. If approved and meets minimum execution time and action is not expired, action can now be executed, if not action is canceled

2. Architecture Improvements

The following architecture improvements and feedback could be considered:

2.1 Incorporate ERC20 tokens for action execution that requires value

Could consider incorporating payment of action execution with common ERC-20 tokens (USDC, USDT, BNB ...). The tokens incorporated can be whitelisted to prevent ERC-20 tokens with other caveats from interacting with protocol until support is implemented (e.g. rebasing, fee-on-transfer)

2.2 Create a new type of strategy for flexibility

Could consider creating a new type of Llama strategy in which approval/disapproval thresholds are specified as percentages of total supply and action creators are not allowed to cast approvals or disapprovals on their own actions for more flexibility

2.3 Checkpoints contracts are deprecated by OpenZeppelin

Checkpoint contracts seems to be deprecated by OpenZeppelin, not sure how this affects Llama contracts but since it affects core parts of the contract logic such as retrieving `quantity` and `expiration` data of roles, it might be worth noting.

2.4 Consider changing `quantity` check logic

Consider changing logic for action creation and checks for role before action creation. Protocol owners cannot set role with 0 quantity coupled with an expiration due to checks in `_assertValidRoleHolderUpdate()`.

Only the `approvalRole` is required to have quantity. All other roles that do not have approval power but have `quantity` assigned to them will only incur unecessary computation.

Based on current implementation of `_setRoleHolder`, protocol owners can never give a policyholder a role with an expiry with no quantity that represents approval/disapproval casting power. In the event where protocol owner wants to give policyholder a role that has 0 `quantity` of votes, they can never do so.

Furthermore, `hasPermissionId()` also checks for quantity before allowing creation of actions. This means policyholders can only create actions if some `quantity` of approval/disapproval votes is assigned to them. Sementically, I don't think the quantity used for voting has relation to action creation.

Although that particular policy holder cannot vote unless `approvalRole /disapprovalRole` is assigned to them, it can cause confusion where policy holders might think they can vote since some `quantity` is assigned to them.

The following adjustments can be made:

- You could consider adding a third case in `_assertValidRoleHolderUpdate()` such as the following:

```
case3 = quantity == 0 && expiration > block.timestamp;
```

- Remove `quantity > 0` check in `LlamaPolicy.hasPermissionId()` to allow action creators to create roles even when no quantity is assigned to them, since permissions to create actions are required to be set for policy holders via `setRolePermissions()` anyway.
- `hasRole()` can simply check for expiration to determine if policy holder has role
- A separate `hasCastRole()` can be created to specifically check for approval/disapproval Role

This way, `quantity` will only ever need to be assigned to policyholders assigned with the approval/disapproval role.

2.5 No actual way to access role descriptions via mapping

In the `policy-management.md` doc it states that:

> When roles are created, a description is provided. This description serves as the plaintext mapping from description to role ID, and provides semantic meaning to an otherwise meaningless unsigned integer.

However, there is no actual way to access roleId via role descriptions in contract. Policy holders cannot access role descriptions and roleIds convieniently except via protocol UI.

Hence, protocol could consider adding a new mapping to map roleIds to description and add logic to return role description and Id in `LlamaPolicy.updateRoleDescriptions()`.

2.6 Consider increasing number of unique roles

Since Id 0 is reserved for the bootstrap `ALL_HOLDERS_ROLE`, the protocol owner could infact only have 254 unique roles. So it may be good to consider using `uint16` to allow 65534 unique roles.

3. Centralization risks

3.1 Policy holders with forceApproval/forceDisapproval role can force approvals and disapproval

Policy holders will approval/disapproval role and quantity of `type(uint128).max` can force approval/disapproval of actions via `forceApprovalRole/forceDisapproval` mapping.

3.2 Protocol owners can revoke roles of policyholders anytime

Protocol owners can revoke policyholders anytime via `LlamaPolicy.revokePolicy()` and prevent action creation/queuing/execution and approval/disapproval. It should be noted that as long as action is created, that action can be executed regardless policyholder is revoked or not, unless action is explicitly cancelled or disapproved.

3.3 Any guards can be set for actions before execution

The type of guards can be customized by protocol owners, so at any point of time specific guards can be set for specific action based on data input (selector) and possibly unfairly prevent execution of action via `LlamaCore.setGuard()`.

4. Time Spent

A total of 4 days were spent to cover this audit, broken down into the following:

- 1st Day: Understand protocol docs, action creation flow and policy management
- 2nd Day: Focus on linking docs logic to `LLamaCore.sol` and `LlamaPolicy.sol`, coupled with typing reports for vulnerabilities found
- 3rd Day: Focus on different types of strategies contract coupled with typing reports for vulnerabilities found
- 4th Day: Sum up audit by completing QA report and Analysis

Time spent:

96 hours

Disclosures

C4 is an open organization governed by participants in the community.

C4 audits incentivize the discovery of exploits, vulnerabilities, and bugs in smart contracts. Security researchers are rewarded at an increasing rate for finding higher-risk issues. Audit submissions are judged by a knowledgeable security researcher and solidity developer and disclosed to sponsoring developers. C4 does not conduct formal verification regarding the provided code but instead provides final verification.

C4 does not provide any guarantee or warranty regarding the security of this project. All smart contract software should be used at the sole risk and responsibility of users.

Top