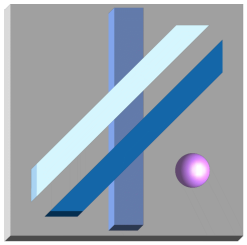

Fixed Point Solutions, LLC

Nova Protocol Assessment

2021/08/25
Prepared by: Kurt Barry

1. Scope

Nova is a protocol for trustlessly relaying arbitrary smart contract calls from the Optimistic Ethereum rollup to mainnet Ethereum. It includes a special facility to allow a relayer on L1 to use tokens they hold to complete the request and be reimbursed with equivalent tokens on L2. It also allows requests to be canceled, or sped up with a higher gas price.

Audited commit: 5a609bd93ca9889c0b299932a092707f133486bb of Rari-Capital/nova
Files: contracts/ except for contracts/mocks/ and contracts/echidna/; certain dependencies

Additionally, fixes for reported issues were reviewed (noted as commit hashes in responses to findings), as well as the following commits:
7a68bd4df20a1d2465eaa4fcae804eb6e6d13ed8
debe85b772e0117d8c9c3da238ea57f9d5d438cd
48418f45793b7c1b1aede35204e08fcfb47e09ef.

The assessment was performed using 12 hours of auditor effort by a single auditor.

2. Limitations

No assessment can guarantee the absolute safety or security of a software-based system. Further, a system can become unsafe or insecure over time as it and/or its environment evolves. This assessment aimed to discover as many issues and make as many suggestions for improvement as possible within the specified timeframe. Undiscovered issues, even serious ones, may remain. Issues may also exist in components and dependencies not included in the assessment scope.

3. Findings

Findings and recommendations are listed in this section, grouped into broad categories. It is up to the team behind the code to ultimately decide whether the items listed here qualify as issues that need to be fixed, and whether any suggested changes are worth adopting. When a response from the team regarding a finding is available, it is provided.

Findings are given a severity rating based on their likelihood of causing harm in practice and the potential magnitude of their negative impact. Severity is only a rough guideline as to the risk an issue presents, and all issues should be carefully evaluated.

Severity Level Determination		Impact		
		High	Medium	Low
Likelihood	High	**Critical**	**High**	**Medium**
	Medium	**High**	**Medium**	**Low**
	Low	**Medium**	**Low**	**Low**

Issues that do not present any quantifiable risk (as is common for issues in the Code Quality category) are given a severity of **Informational**.

3.1 Security and Correctness

Findings that could lead to harmful outcomes or violate the intentions of the system.

SC.1 Relayers' Funds Can Be Stolen If They Approve `L1_NovaExecutionManager` for Tokens With Certain Non-Standard Functions

Severity: Medium

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L1_NovaExecutionManager.sol#L215

Description: Relayers are required to approve the `L1_NovaExecutionManager` contract to transfer their tokens on behalf of certain strategies. This is to protect relayers against potentially malicious strategies, reducing the need for trust in the system. A side effect is that because a token contract can be used as a strategy, direct calls to `transferFrom(address, address, uint256)` must be prevented in the execution manager, otherwise anyone may craft a request that allows stealing a relayer's approved tokens ("approval snatching"). However, the check to

simply prevent a call to `transferFrom` is insufficient to protect relayers, because some tokens have methods other than `transferFrom` for transferring or burning tokens based on approvals. Two common examples include the DAI and MKR tokens, which implement `pull(address src, uint wad)` as an alias for `transferFrom(src, msg.sender, wad)` and a `burn` function that allows one address to destroy another's tokens if an approval has been granted. There is no reason to believe that these are the only two tokens which may violate the assumption that `transferFrom` is the only way to take advantage of an exposed approval, and there is no way to prevent more such tokens from being deployed in the future, or to prevent upgradeable tokens from being modified to introduce such functionality.

Consider using a separate contract to house the `transferFromRelayer` function so that relayers do not need to make approvals on a contract that calls arbitrary code.

Response: Fixed in commit 990fbad4f2ee9f7257660464bbaf232c2e18b4d8.

SC.2 If A Sped-Up Request's Uncle Executes First, the Additional ETH Supplied for the Sped-Up Request Is Permanently Locked

Severity: **Medium**

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L2_NovaRegistry.sol

Description: When a request is sped-up, additional Ether must be sent, as the new gas price is required to be greater than that of the old one. To help avoid relayers accidentally executing invalid uncles, a time delay is enforced during which the uncled request remains valid and the new request cannot yet be executed. If the uncled request is executed by a relayer during this time, there is no way for a requester to claim the extra gas they submitted with their call to `speedUpRequest` because `areTokensRemoved` will return true for both the original and sped-up requests, and further there is no logic to account for the already-claimed ETH in such a refund.

Response: Fixed in commit a0e10a77bef785306d3574623a4bc79b26b149ef.

SC.3 `requestExecWithTimeout` should be `payable`

Severity: **Low**

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L2_NovaRegistry.sol#L256

Description: This function calls `requestExec` which generally expects a non-zero amount of Ether to be sent with the transaction. Without a `payable` modifier, `requestExecWithTimout` will

fail any time `msg.value` is non-zero. It would, however, be quickly discovered and worked around in practice.

Response: Fixed in commit 3380ec2b1c92620baa33b33c7c3e2a1fb74b0339.

SC.4 Preventing Calls to Optimism's Cross-Domain Messenger Would Be More Secure If Based On Address Rather Than Function Signature

Severity: **Low**

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L1_NovaExecutionManager.sol#L220

Description: It is theoretically possible that Optimism's cross-domain messaging contract may have functions other than `sendMessage` that are unsafe to call directly from the execution manager--it would be more robust to simply require that the strategy address differs from that of `iOVM_CrossDomainMessenger`. This additionally allows `sendMessage` to be a valid target method of receiving strategies (although this is a minor benefit).

Response: Fixed in commit 990fbad4f2ee9f7257660464bbaf232c2e18b4d8.

SC.5 Unchecked Arithmetic Used for L1 Gas Calculations

Severity: **Low**

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L1_NovaExecutionManager.sol#L240
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L1_NovaExecutionManager.sol#L263

Description: While relayers can always perform offline computations to guard against e.g. and unexpectedly high gas limit in the external call caused by an underflow, it is likely more robust to explicitly check math operations in the smart contract, despite slightly higher gas costs. This makes it easier to write safe relayer logic, which is beneficial from an ecosystem perspective as it encourages adoption.

Response: Fixed in commit 3d15c1b253ae7ae314508774a7f1a66a95581360 and safely refined in commit 276af16575a574d4b0798e0648af862057f23f26.

3.2 Usability

Findings that could lead to suboptimal user experience or hinder integration.

None.

3.3 Gas Optimizations

Findings that could reduce the gas costs of interacting with the protocol, potentially on an amortized or averaged basis.

None.

3.4 Code Quality

CQ.1 Inaccurate Comments in NovaExecHashLib.sol

Severity: **Low**

Code Location:
[1]https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/libraries/NovaExecHashLib.sol#L4
[2]https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/libraries/NovaExecHashLib.sol#L8

Description:
[1] This comment omits `gasLimit`.
[2] This comment omits `gasLimit` and the use of packed ABI encoding.

Response: Fixed in commit c1c12c7230d7933d0c403348a815a8087d178e2f.

CQ.2 Spelling/Grammar Issues In Comments

Severity: **Informational**

Code Location:
[1]https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L2_NovaRegistry.sol#L550

Description:
[1] Extra “is”.

Response: Fixed in commit fd5b9e9e7f30ee1f8c4cec1e6b126f15a585aac2.

CQ.3 Technically Incorrect Revert Message in `speedUpRequest`

Severity: **Informational**

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L2_NovaRegistry.sol#L393

Description: The revert message says “LESS_THAN_PREVIOUS_GAS_PRICE”, but the transaction will also revert if the new gas price is equal to the previous gas price.

Response: Fixed in commit 6ea70c72331a8e1b05f8da8575a79b910c5e855f.

CQ.4 Misleading Function Name (`areTokensRemoved`)

Severity: **Informational**

Code Location:
https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/L2_NovaRegistry.sol#L497

Description: The function name `areTokensRemoved` is arguably misleading, since e.g. in the case of a sped-up request with a non-dead uncle, it returns `true`, even though that request never “had” any tokens to remove--more accurately, it has not yet received them. A more accurate name would be `hasNoTokens`, or if all returns values are inverted (and inverted at their sites of use), the even more straightforward name `hasTokens` could be used. It also behaves in an arguably unexpected way for hashes that do not correspond to created requests by indicating that they have tokens.

Response: Fixed in commits 3073f6aa9ecee13d8872d19500761dba86db23c5 and 2f3456c8a63d1866f45bb97738ec830bece46a87.

4. Notes

This section contains general considerations for interacting with or maintaining the system and various conclusions reached or discoveries made during the course of the assessment. Whereas findings generally represent things for the team to consider changing, notes are more informational and may be helpful to those who intend to interact with the system.

4.1 Risks to Relayers From Request Cancellation

Relayers will need to pay close attention to the state of the L2 registry, as if a user cancels their request and withdraws the supplied tokens, then the relayer will not be reimbursed for completing the request on L1. Relayers can protect themselves somewhat by setting a deadline that comes before the earliest possible token withdrawal time for a request. Frontrunning or other competition from other relayers is still a risk, as requests can execute at most once.

4.2 Legacy Gas Pricing

The code as audited assumes gas pricing is done in a pre-EIP 1559 style. This shouldn't present significant practical problems, but is something else for relayers to be aware of.

4.3 Use of `abi.encodePacked()`

https://github.com/Rari-Capital/nova/blob/5a609bd93ca9889c0b299932a092707f133486bb/contracts/libraries/NovaExecHashLib.sol#L16

This usage is safe from hash collisions because only one dynamic type is used, but care should be taken that this is not changed in the future.

4.4 Analysis of Additional Risk From Removal of `nonReentrant` Modifiers

The version of the code audited included reentrancy guards on three functions (`requestExec`, `claimInputTokens`, `withdrawTokens`) within the `L2_NovaRegistry` contract. At the client's request, the additional risk posed by removing these modifiers was evaluated. There appears to be no significant additional risk introduced by removing the modifiers from these functions; the reasoning behind this is explained below. Note that this does not imply that there are not reentrancy-based vulnerabilities in the contract that remain undiscovered; it is rather a statement that the removal of these reentrancy guards likely does not introduce additional vulnerabilities. The assumption that storage locations do not collide is used, and while it may not be strictly necessary, it simplifies the analysis and is anyway necessary for the overall correctness of the contract. In practice this should be a very robust assumption given the largeness of 2^{256} compared to the expected number of storage slots that could feasibly be used by Nova over human timescales. Note that it is not a recommendation of Fixed Point Solutions that any of the modifiers be removed; rather, we simply provide the results of our analysis to the client and other consumers of this report for their own consideration.

`withdrawTokens` → `withdrawTokens`

If the reentrant call is invoked with the same `execHash` as the initial call, the transaction will revert because `areTokensRemoved` will return `true` because the initial call set `getRequestInputTokenRecipientData[execHash].recipient` to a non-zero address before making an external call (and no other method can alter the value once set). If the reentrant call is invoked with a different `execHash` argument, this will behave no differently than making this call in a separate transaction.

`withdrawTokens` → `claimInputTokens`

If the reentrant call is invoked with the same `execHash` as the initial call, the transaction will revert because the initial call set

`getRequestInputTokenRecipientData[execHash].isClaimed` to `true` before making an external call (and no other method can alter the value once set). If the reentrant call is invoked with a different `execHash` argument, this will behave no differently than making this call in a separate transaction.

`withdrawTokens` → `requestExec`

An `execHash` collision is not feasible due to the incrementing of the `systemNonce` variable; thus the only risk would be a storage collision, which is assumed to be unlikely enough to be disregarded.

`claimInputTokens` → `claimInputTokens`

Argument is identical to the `withdrawTokens` → `claimInputTokens` case.

`claimInputTokens` → `withdrawTokens`

If the reentrant call is invoked with the same `execHash` as the initial call, the transaction will revert because `areTokensRemoved` will return `true` because in order for the initial call not to revert, `getRequestInputTokenRecipientData[execHash].recipient` had to be a non-zero address before making an external call (and no other method can alter the value once set). If the reentrant call is invoked with a different `execHash` argument, this will behave no differently than making this call in a separate transaction.

`claimInputTokens` → `requestExec`

Argument is identical to the `withdrawTokens` → `requestExec` case.

`requestExec` → `requestExec`

Argument is identical to the `withdrawTokens` → `requestExec` case.

`requestExec` → `withdrawTokens`

If the reentrant call is invoked with the same `execHash` generated in the initial call, the transaction will revert because `areTokensUnlocked` will return `false` because either:

1. `getRequestUnlockTimestamp[execHash]` will be zero; or
2. `getRequestUnlockTimestamp[execHash]` will be greater than the current `block.timestamp` as `unlockTokens` cannot be called before `getRequestCreator[execHash]` is a non-zero address (which is only true starting from within the current block as it is only set in `requestExec`) and `MIN_UNLOCK_DELAY_SECONDS` is greater than zero.

If the reentrant call is invoked with a different `execHash` argument than that generated in the initial call, this will behave no differently than making this call in a separate transaction.

`requestExec` → `claimInputTokens`

If the reentrant call is invoked with the same `execHash` generated in the initial call, the transaction will revert because `getRequestInputTokenRecipientData[execHash].recipient` will be the zero address, because it is not possible to complete an L1 relay round-trip in a single Optimistic Ethereum transaction. If the reentrant call is invoked with a different `execHash` argument than that generated in the initial call, this will behave no differently than making this call in a separate transaction.