

# Statechannels Nitro

## 1 Executive Summary

- 1.1 Report Layout
- 1.2 Important Remarks

## 2 Scope

- 2.1 Objectives

## 3 Fuzzing and Custom Property Checking

- 3.1 Overview
- 3.2 Instrumentation with Scribble
- 3.3 Properties
- 3.4 Setup
- 3.5 Results
- 3.6 Conclusion

## 4 Smart Contract Recommendations

- 4.1 Improve the semantic meaning of variables in the codebase Open
- 4.2 Improve input validation, fail early on invalid input Open
- 4.3 Use the type system in parameters Open
- 4.4 Use existing internal methods instead of reimplementing them inline Open
- 4.5 Stick to interface naming conventions Open
- 4.6 Public functions may be external Open
- 4.7 Missing declaration of use for SafeMath may break Solidity 0.7.x compatibility Open
- 4.8 Optimization - `_transferAll()` should break out of loop if `balance <= 0` Open

## 5 Smart Contract Findings

- 5.1 Reentrancy in `claimAll` may allow an external address to drain funds Major
- 5.2 Recipient can block transferAsset/payouts for all participants Major
- 5.3 Gas siphoning attack possible in “transfer all” and “claim all”-related methods Medium
- 5.4 A channel established with a `FixedPart.challengeDuration == 0` can be challenged and finalized immediately Medium
- 5.5 Channel parameters may hash to a channelId hinting an external destination potentially losing funds Medium
- 5.6 `AttestationApp` - Bytes comparison should enforce length check Minor
- 5.7 `AttestationApp` - De-duplicate state machine states Minor
- 5.8 Documentation inconsistency Minor Pending
- 5.9 Signature validation may allow invalid signatures if a channel is established with `address(0)` as one of the participants Minor
- 5.10 `_bytesEqual()` should enforce a length check Minor

## 1 Executive Summary

This report presents the results of our engagement with **TheGraph/Statechannels.org** to review the **Nitro Protocol Smart Contracts & The Graph Server Wallet Library**.

The review was conducted over five weeks, from **Oct 12, 2020** to **Nov 13, 2020**.

During the first and second weeks, the assessment team ramped up on auditing the smart contracts, and efforts regarding the verification of specific properties of the smart contract system were initiated. The focus was set on the *Nitro protocol*’s and *ForceMove*’s contracts.

In the third week, the assessment team delivered a preliminary report summarizing the property verification and fuzzing efforts. The final result can be found in [Section 3](#). Additionally, the first results of the smart contract review were informally shared with the client team. The issues can be found in [Section 5](#) and [Section 4](#). Towards the end of the week, the focus of efforts was shifted towards reviewing the off-chain `server-wallet` store component. The off-chain components were under active development and only ready towards the end of the week. Given the codebase’s size and complexity, the assessment team requested multiple meetings, including walk-throughs in which the scope was further clarified.

During the fourth week, the assessment team continued working tightly with the relevant client teams to understand further the systems and codebases involved. This week, it was brought to the assessment team’s attention that two more code repositories existed, housing code that implements the `server-wallet` store that was initially set in scope. As it was found crucial to understand the system’s attack surface better, the assessment team requested access to the `gateway` and `indexer` repositories.

During the beginning of the fifth week that the assessment team updated the commit hash under scrutiny to a new one. Please note this is not the commit hash referenced in the *Scope* section. The renovated commit hash agreed upon between the two teams was: `0c00a9c8eb8b1c61b480385e6ea3a6786b1109d1`, this was especially important for the changes in the `server-wallet` codebase.

During the fifth week, the assessment team, armed with all the previous knowledge gained throughout the engagement, started looking for more business-logic-related edge cases in the `server-wallet` codebase.

### 1.1 Report Layout

For the Smart Contracts, the following artifacts have been produced:

- Results for the [Fuzzing and Custom Property Checking](#) campaign
- [Recommendations](#) for the Smart Contracts
- [Findings](#) for the Smart Contracts

For the Application review, the following artifacts have been produced:

- [Findings](#) for the Application review
- Recommendations and informational findings (i.e., findings that carry no severity) are mentioned alongside issues in the [Application Findings](#) section.

### 1.2 Important Remarks

- The code under review was undergoing constant changes during the engagement. The audit codebase was updated multiple times to catch up with important upstream changes. It should be noted that the application code and `AttestationApp` were a moving target, which led the assessment team, together with the development team, to conclude that it might have been too early to review these components specifically.
- While the Nitro protocol documentation was immensely helpful, the documentation and code annotations for the application (`gateway` / `indexer`) were sparse to non-existent. Expressly, a specification for the applications was not provided. It should be noted that an assessment of this nature exists mostly as verification that an application complies with its specification. This is done to ensure that the client actually coded the system the way it was designed. A lack of documentation and specification makes it hard to reason about the design rationales and may leave important security properties untested.
- The protocol does not specify how messages are being exchanged with each of the participants. The current implementation of gateway and indexer appears to provide an HTTP interface. It should be noted that these interfaces and API server implementations were not reviewed in this engagement (e.g., proper use of transport security, API service and web security-related vulnerabilities, access control and permissions, rate-limiting, ...).
  - In the current implementation, one party can flood the other party with messages, potentially leading to DoS like scenarios. There is no whitelisting or rate-limiting.
- A threat to both applications (`gateway` / `indexer`) is that an unexpected or malicious input may cause an exception that is not caught by either implementation.
- The protocol offloads most of the trust to the actual application implementations and relies on the applications to verify all parameters.

### Book your 1-Day Security Spot Check

BOOK NOW

Date	November 2020
------	---------------

6 Application Findings

6.1 `server-wallet` - `validateTransition` fails to return on turn number check failed Major

6.2 `server-wallet` - Processing states with zero participants may Minor

- Since the state is kept in the application’s database, it could be considered to strip the constant part from inbound messages reducing the size of the message.
  - For example, on channel creation, the channel’s constant parts are negotiated and stored in the database, and any further message refers to the channel by its variable parts.
- The `server-wallet` codebase has a large amount of branching logic crafted specifically to facilitate tests. This becomes problematic because these branches are buried somewhat deep within production-targeted modules. It makes it harder to scour for these test-specific snippets at the time of production deployment.
- The smart contracts should always be the reference implementation of the protocol and not the documents. At least, this will maintain cohesion between on-chain and off-chain components and prevent lack of alignment at a later stage. Even documentation should be a by-product of the on-chain reference implementation.
- The `server-wallet` codebase has a confusing mix of components built for generic `n` player games and `2` player games (this is due to the game’s specific necessities tailored for The Graph). An effort should be made to go either one way or the other. Keeping some elements generic and other specific will lead to confusion in the future.
- There is a general trend of lack of input validation and lax error handling in the `server-wallet` codebase. In the issues presented in this report, we have encountered problems with these topics repeatedly. We believe that the lax error validation in the off-chain components creates an increased attack surface for DoS attacks.

2 Scope

Our review focused on several repositories and commit hashes. Some of them were reviewed while in flux and, as such, were not pinned to a specific commit.

Repository URL	Code under review	Commit Hash
<a href="https://github.com/statechannels/statechannels.git">https://github.com/statechannels/statechannels.git</a>	packages/nitro-protocol	<a href="#">statechannels/statechannels@fe04a09</a>
	packages/server-wallet	
<a href="https://github.com/statechannels/the-graph.git">https://github.com/statechannels/the-graph.git</a>	*	(codebase was in constant flux)
<a href="https://github.com/graphprotocol/indexer.git">https://github.com/graphprotocol/indexer.git</a>	(only used as reference material)	
<a href="https://github.com/graphprotocol/gateway.git">https://github.com/graphprotocol/gateway.git</a>	(only used as reference material)	

The list of main files in scope can also be found in the [Appendix](#).

2.1 Objectives

Together with the StateChannels.org and The Graph team, we identified the following priorities for our review:

1. Review of the Nitro Protocol Smart Contracts
2. Identify known vulnerabilities particular to smart contract systems, as outlined in our [Smart Contract Best Practices](#), and the [Smart Contract Weakness Classification Registry](#).
3. Property verification for the Nitro Protocol using MythX property verification and fuzzing capabilities (Scribble/Harvey)
4. Review of the server wallet store, off-chain component.

3 Fuzzing and Custom Property Checking

3.1 Overview

We checked several custom properties of the Nitro protocol contracts by instrumenting the code using our internal Scribble specification language and running 36h fuzzing campaigns using the Harvey fuzzer on the instrumented code.

Our work was done on commit [fe04a0991aadd902f5395df3d18954e5743bc9bb](#) with the following contracts in scope:

- `packages/nitro-protocol/contracts/ForceMove.sol`
- `packages/nitro-protocol/contracts/NitroAdjudicator.sol`
- `packages/nitro-protocol/contracts/AssetHolder.sol`
- `packages/nitro-protocol/contracts/ETHAssetHolder.sol`
- `packages/nitro-protocol/contracts/ERC20AssetHolder.sol`

In total we formalized `11` invariants to be checked through fuzzing.

In [Section 3.2](#) we give a brief overview of our instrumentation process. In [Section 3.3](#) we list the eleven invariants we checked using fuzzing. In [Section 3.4](#) we describe our fuzzing setup. In [Section 3.5](#) we present the results of the fuzing campaign.

3.2 Instrumentation with Scribble

We formalized `11` invariants (including 5 invariants from previous work using [TLA+](#) in our internal Scribble specification language. The formalized invariants were inserted as annotations in comments before functions. For example, in the listing below we added an annotation to the `NitroAdjudicator.pushOutcome` function. The annotation specifies that `NitroAdjudicator.pushOutcome` can only succeed if called on a channel in the finalized state.

```

/// if_succeeds {:msg "P8"} old(_mode(channelId)) == ChannelMode.Finalized;
function pushOutcome(
    bytes32 channelId,
    uint48 turnNumRecord,
    uint48 finalizesAt,
    bytes32 stateHash,
    address challengerAddress,
    bytes memory outcomeBytes
) public override {
    ...
}

```

In the above annotation the `if_succeeds` keyword specifies that this is an invariant that must hold on successful function calls only. The `{:msg "P9"}` tag can be ignored. The expression itself is almost vanilla Solidity - `old(_mode(channelId)) == ChannelMode.Finalized` .

The only peculiar bit is the `old` keyword, which specifies that we want the value of `_mode(channelId)` at the start of the function.

The above annotation has the semantics of a function post-condition - if the function returns successfully, then the expression `old(_mode(channelId)) == ChannelMode.Finalized` must evaluate to true.

We use an instrumentation tool, which automatically inserts checks for the annotated properties by re-writing the contracts in-place and adding corresponding assertions. For example, after instrumentation the above code would look something like this:

```

function pushOutcome(
    bytes32 channelId,
    uint48 turnNumRecord,
    uint48 finalizesAt,
    bytes32 stateHash,
    address challengerAddress,
    bytes memory outcomeBytes
) public override {
    uint8 oldMode = _mode(channelId);

    _original_pushOutcome(
        channelId,
        turnNumRecord,
        finalizesAt,
        stateHash,
        challengerAddress,
        outcomeBytes);

    if (oldMode != ChannelMode.Finalized) {
        <throw user assertion>
    }
}

function _original_pushOutcome(
    bytes32 channelId,
    uint48 turnNumRecord,
    uint48 finalizesAt,
    bytes32 stateHash,
    address challengerAddress,
    bytes memory outcomeBytes
) public override {
    ...
}

```

Finally as a sanity check for our properties, we made sure the instrumented contracts can successfully pass the Nitro test suite.

### 3.3 Properties

Below is a list of the instrumented properties, along with a short explanation of each and the location(s) where it was inserted.

**1. Calling `forceMove` successfully on an open channel transitions it into the Challenge state. Such calls are only possible of `largestTurnNum` is greater or equal to the current known turn.**

Annotation:

```

if_succeeds {:msg "P1"} let channelId := _getChannelId(fixedPart) in
    let lastTurnNum := old(
        let lastTurnNum, finalizesAt, fingerprint :=
            _getChannelStorage(_getChannelId(fixedPart)) in lastTurnNum) in
    let newTurnNum, finalizesAt, fingerprint :=
        _getChannelStorage(channelId) in
    let oldMode := old(_mode(_getChannelId(fixedPart))) in
    let newMode := _mode(channelId) in
        newTurnNum == largestTurnNum &&
        newMode == ChannelMode.Challenge &&
        (oldMode == ChannelMode.Open ==> newTurnNum >= lastTurnNum);

```

Location:

Annotation above `ForceMove.forceMove` .

Description:

This corresponds to the TLA+ property `Open(n)->forceMove(m,s,p)->Chal(m,s,p) if m>=n`

Note:

This property is actually a little weaker than the TLA+ property - it \*doesn't specify that the stored state hash is also updated correctly. This is difficult to express due to the complex computation involved in coming up with the hash of the encoded `ChannelData` . However we could be further strengthened, by specifying that `finalizesAt` part of the channel data is also updated correctly.

**2. Calling `forceMove` successfully on a channel in the Challenge state**

transitions into the Challenge state. Such calls are only possible of `largestTurnNum` is greater than the current known turn.

Annotation:



```

if_succeeds {:msg "P2"} let channelId := _getChannelId(fixedPart) in
  let lastTurnNum := old(
    let lastTurnNum, finalizesAt, fingerprint :=
      _getChannelStorage(_getChannelId(fixedPart)) in lastTurnNum) in
  let newTurnNum, finalizesAt, fingerprint :=
    _getChannelStorage(channelId) in
  let oldMode := old(_mode(_getChannelId(fixedPart))) in
  let newMode := _mode(channelId) in
    newTurnNum == largestTurnNum &&
    newMode == ChannelMode.Challenge &&
    (oldMode == ChannelMode.Challenge ==> newTurnNum > lastTurnNum);

```

Location: Annotation above `ForceMove.forceMove` .

Description: This corresponds to the TLA+ property `Chal(n,s,p)->ForceMove(m,s',p)->Chal(m,s',p) if m>n` .

### 3. You can only call `forceMove` on a channel in an Open or Challenge state.

Annotation:

```

if_succeeds {:msg "P3"}
  let oldMode := old(_mode(_getChannelId(fixedPart))) in
    oldMode == ChannelMode.Open ||
    oldMode == ChannelMode.Challenge;

```

Location:

Annotation above `ForceMove.forceMove` .

### 4. Calling `checkPoint` successfully, with turn number M, on a channel in an

open or challenge state and turn number N, where `M>N` transitions the channel into an open state with turn number M.

Annotation:

```

if_succeeds {:msg "P4"} let channelId := _getChannelId(fixedPart) in
  let lastTurnNum := old(
    let lastTurnNum, finalizesAt, fingerprint :=
      _getChannelStorage(_getChannelId(fixedPart)) in lastTurnNum) in
  let newTurnNum, finalizesAt, fingerprint :=
    _getChannelStorage(channelId) in
  let oldMode := old(_mode(_getChannelId(fixedPart))) in
  let newMode := _mode(channelId) in
    (oldMode == ChannelMode.Open ||
     oldMode == ChannelMode.Challenge) &&
    newTurnNum == largestTurnNum &&
    newMode == ChannelMode.Open &&
    newTurnNum > lastTurnNum;

```

Location:

Annotation above `ForceMove.checkPoint` .

Description:

This corresponds to the following 2 TLA+ properties: `Open(n,s)->checkpoint(m)->Open(m) if m>n` and

`Chal(n,s,p)->checkpoint(m)->Open(m) if m>n` .

### 5. Successful call to `respond` on a channel in the challenge state with turn number N, transitions the channel into the open state with turn number `N+1` .

Annotation:

```

if_succeeds {:msg "P5"} let channelId := _getChannelId(fixedPart) in
  let lastTurnNum := old(
    let lastTurnNum, finalizesAt, fingerprint :=
      _getChannelStorage(_getChannelId(fixedPart)) in lastTurnNum) in
  let newTurnNum, finalizesAt, fingerprint :=
    _getChannelStorage(channelId) in
  let oldMode := old(_mode(_getChannelId(fixedPart))) in
  let newMode := _mode(channelId) in
    (oldMode == ChannelMode.Challenge &&
     newMode == ChannelMode.Open &&
     newTurnNum == lastTurnNum + 1);

```

Location:

Annotation above `ForceMove.respond` .

Description:

This corresponds to the following TLA+ property: `Chal(n,s,p)->respond(s, s')->Open(n+1) if s->s'` .

Note:

Since the implementation of `_mode` returns `ChannelMode.Challenge` \*iff `finalizesAt < now` , this property also implies that “ `respond` will only succeed if called on a channel before its challenge deadline expires”.

### 6. You can only finalize a channel that is not already finalized.

Annotation:

```

if_succeeds {:msg "P6"}
  let oldMode := old(_mode(_getChannelId(fixedPart))) in
    (oldMode == ChannelMode.Open ||
     oldMode == ChannelMode.Challenge) &&
    _mode(_getChannelId(fixedPart)) == ChannelMode.Finalized

```

Location:

Annotation above `ForceMove.conclude` and `NitroAdjudicator.concludePushOutcomeAndTransferAll` .

Description:

The intent here is to check that a channel can only be finalized once. However, the annotated property is subtly weaker. If on top of the annotated property we could add an annotation to the effect of “a finalized channel never goes into any other state” then together the two would imply that a channel can only be finalized once. However, unfortunately at the moment we cannot express the second property in our language.

7. `forceMove` , `checkpoint` and `respond` only succeed if a signatures is given for every participant.

Annotation:

```
if_succeeds {:msg "P7"}
  sigs.length == whoSignedWhat.length &&
  sigs.length == fixedPart.participants.length;
```

Location:

Annotation above `ForceMove.forceMove` , `ForceMove.checkpoint` , `ForceMove.conclude` , and `NitroAdjudicator.concludePushOutcomeAndTransferAll` .

Description:

Ideally we would like to express the functional property “ `forceMove` , `checkpoint` and `respond` succeed only if a *valid* signature is given for every participant. However our language currently doesn’t support the universal quantifier necessary to express this property, thus we opted for the simpler one. We expect to add support for universal quantification in the coming months.

8. `respond` only succeed if the challenger provides a valid signature.

Annotation:

```
if_succeeds {:msg "P8"}
  let oldTurnNum := old(
    let turnNum, fAt, fp :=
      _getChannelStorage(_getChannelId(fixedPart)) in turnNum) in
  let channelId := _getChannelId(fixedPart) in
  let responder := fixedPart.participants[(oldTurnNum+1) % fixedPart.participants.length] in
  let responseHash :=
    _hashState(
      oldTurnNum+ 1,
      isFinalAB[1],
      channelId,
      fixedPart,
      variablePartAB[1].appData,
      _hashOutcome(variablePartAB[1].outcome)
    ) in
  _recoverSigner(responseHash, sig) == responder;
```

Location:

Annotation above `ForceMove.respond` .

9. `respond` only succeed if the hash of its arguments match an existing challenge hash.

Annotation:

```
if_succeeds {:msg "P9"}
  let oldTurnNum := old(
    let turnNum, fAt, fp :=
      _getChannelStorage(_getChannelId(fixedPart)) in turnNum) in
  let oldFinAt := old(
    let turnNum, fAt, fp :=
      _getChannelStorage(_getChannelId(fixedPart)) in fAt) in
  let channelId := _getChannelId(fixedPart) in
  let challengeHash :=
    _hashState(oldTurnNum,
      isFinalAB[0],
      channelId,
      fixedPart,
      variablePartAB[0].appData,
      _hashOutcome(variablePartAB[0].outcome)) in
  _matchesHash(
    ChannelData(
      oldTurnNum,
      oldFinAt,
      challengeHash,
      challenger,
      _hashOutcome(variablePartAB[0].outcome)),
    old(channelStorageHashes[_getChannelId(fixedPart)]));
```

Location:

Annotation above `ForceMove.respond` .

10. `pushOutcome` / `pushOutcomeAndTransferAll` only succeed on a finalized channel.

Annotation:

```
if_succeeds {:msg "P10"}
  let oldMode := old(_mode(_getChannelId(fixedPart))) in
  oldMode == ChannelMode.Finalized
```

Location:

Annotation above `NitroAdjudicator.pushOutcome` and `NitroAdjudicator.pushOutcomeAndTransferAll` .

11. `deposit()` increases the amount held by an AssetHolder to the expected value.

Annotation:

```
if_succeeds {:msg "P11"}
  let oldHoldings := old(holdings[destination]) in
  oldHoldings < holdings[destination] &&
  holdings[destination] == expectedHeld + amount;
```

Location:

Annotation above `ERC20AssetHolder.deposit` and `ETHAssetHolder.deposit` .

### 3.4 Setup

Since generating correct signatures through fuzzing is impractical, we had to modify the the signature code (specifically `ForceMove._recoverSigner` ) such that it returns a value directly provided by Harvey - thus essentially making the output of the signature checking something that Harvey fuzzes directly.

Due to the complexity of the input space for the contracts under fuzzing, we ran two separate fuzzing campaigns with different setups, to achieve higher coverage.

In both setups, our deployment was based on the Nitro deployment script ( `packages/nitro-protocol/deployment/deploy.js` ), extended with several extra steps to distribute tokens to several known attacker accounts.

#### “Raw” fuzzing campaign

In the raw campaign, we allowed Harvey to directly target the following contracts:

- NitroAdjudicator
- ERC20AssetHolder
- ETHAssetHolder

#### “Structured” fuzzing campaign

In the structured fuzzing campaign, on top of the setup from the “raw” campaign, we also add a `FuzzHarness` contract, that invokes the entry points of the 3 target contracts:

- NitroAdjudicator
- ERC20AssetHolder
- ETHAssetHolder

In this setup Harvey doesn’t directly fuzz the 3 target contracts above. Instead it only targets the `FuzzHarness` entry points, and each `FuzzHarness` entry point invokes a function on the 3 original target contracts, with arguments derived from the random inputs Harvey provided.

At construction time, `FuzzHarness` populates an array of 3 potential participants ( `address[3]` ), and 4 channels ( `FixedPart[4]` ). The 4 channels have 0-3 participants each selected from the potential participants.

`FuzzHarness` has an entry point for each of the following functions:

- ForceMove.forceMove
- ForceMove.respond
- ForceMove.checkpoint
- ForceMove.conclude
- NitroAdjudicator.pushOutcome
- NitroAdjudicator.pushOutcomeAndTransferAll
- NitroAdjudicator.concludePushOutcomeAndTransferAll
- AssetHolder.transferAll
- AssetHolder.claimAll
- AssetHolder.deposit

`FuzzHarness` entry points receive inputs directly from Harvey, from which they generate inputs to their respective target function. For example the entry point `FuzzHarness.transferAll` calls the `AssetHolder.transferAll` function, which expects a `bytes32 channelId` argument:

```
function transferAll(
    bytes32 channelId,
    bytes memory allocationBytes) public override
{
    ...
}
```

However `FuzzHarness.transferAll` doesn’t receive a `bytes32 channelId` from Harvey. Instead it receives a `uint8 channelId` , from which it decides whether to use the id of one of the 4 channels created at construction time, or to use an invalid channel id.

The fuzzing harness similarly makes a random choice from a precomputed set of options for destinations and asset holders. The `uint` amounts are still fuzzed directly by Harvey.

### 3.5 Results

We ran the two fuzzing campaigns for 36 hours. Our coverage of the fuzzed properties is summarized below:

Property	Hit by Testing	Violated in Testing	Hit by Raw Fuzzing	Violated by Raw Fuzzing	Hit by Harness Fuzzing	Violated by Harness Fuzzing
1	Yes	No	Yes	No	Yes	No
2	Yes	No	Yes	No	Yes	No
3	Yes	No	Yes	No	Yes	No
4	Yes	No	Yes	No	Yes	No
5	Yes	No	No	No	Yes	No
6	Yes	No	Yes	No	Yes	No
7	Yes	No	Yes	Yes	Yes	No
8	Yes	No	No	No	Yes	No
9	Yes	No	No	No	Yes	No

Property	Hit by Testing	Violated in Testing	Hit by Raw Fuzzing	Violated by Raw Fuzzing	Hit by Harness Fuzzing	Violated by Harness Fuzzing
10	Yes	No	No	No	No	No
11	Yes	No	Yes	No	No	No

All the properties were exercised by the Nitro protocol test suite, and all but property 10 were hit at least once in fuzzing. During our run we found only one *benign* violation to property 7:

```
if_succeeds {:msg "P7"}
  sigs.length == whoSignedWhat.length &&
  sigs.length == fixedPart.participants.length;
```

The issue is that as formalized our property is too strict. The function will still succeed if we provide additional garbage signatures at the end of `sigs` . The following more-relaxed version should be used instead in the future:

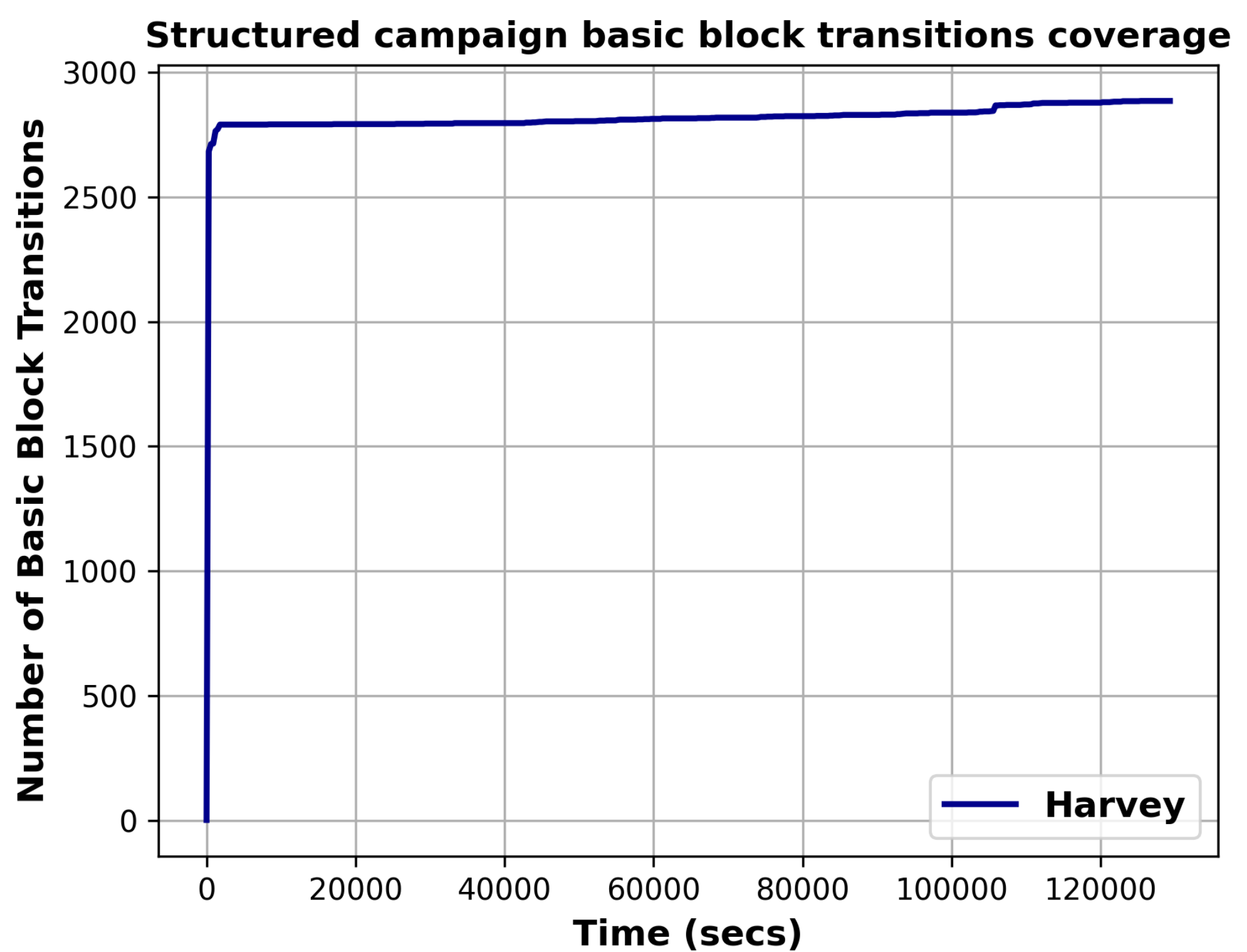
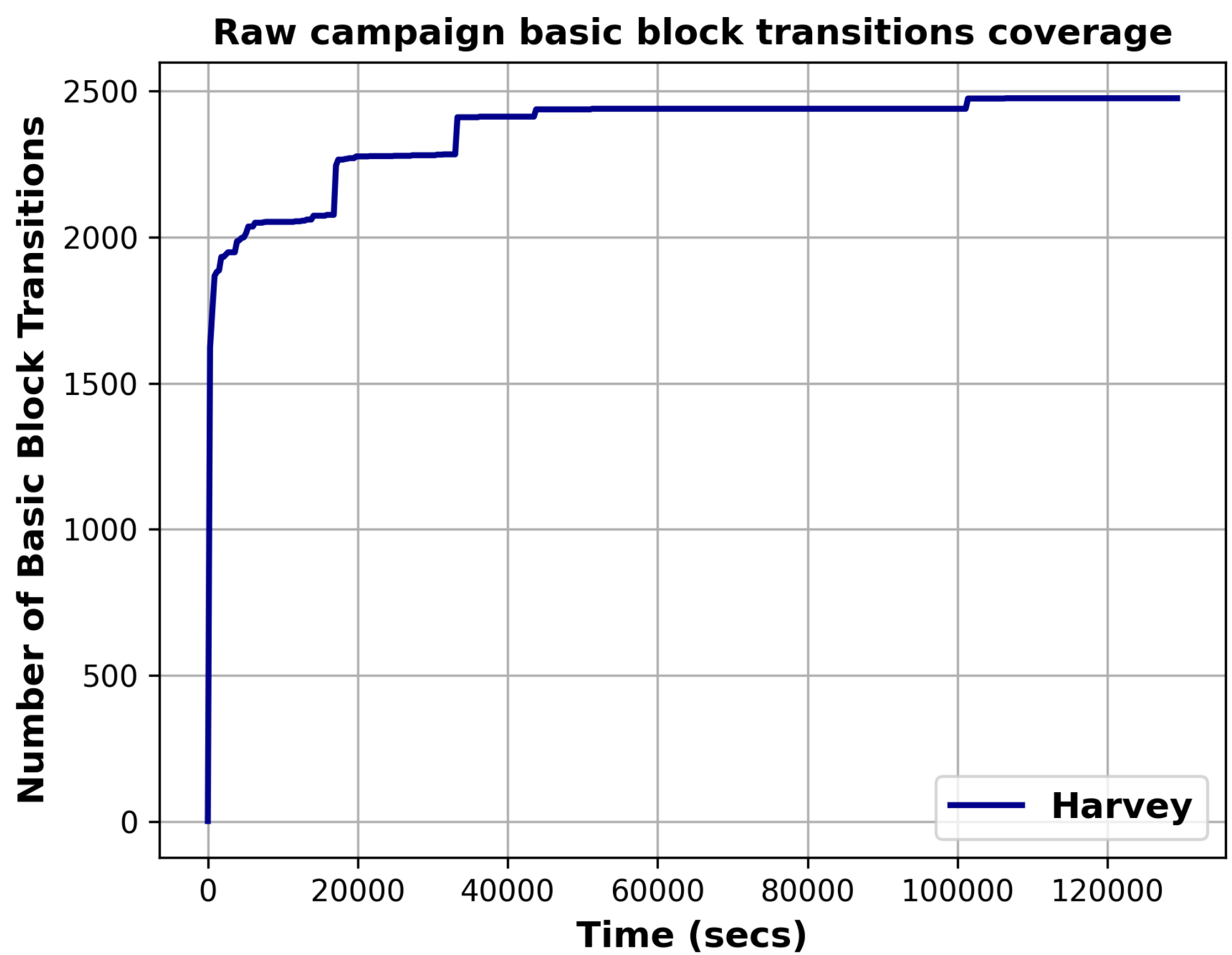
```
if_succeeds {:msg "P7"}
  sigs.length <= whoSignedWhat.length &&
  sigs.length >= fixedPart.participants.length;
```

Thus no security relevant violations were found.

Below we report additional coverage metrics. While the structured fuzzing campaign shows better coverage in terms of all measured metrics, this should be taken with a grain of salt, as the reported coverage also include the `FuzzHarness` contract, which is not present in the raw campaign.

Fuzzing Run	# Covered Paths	# Covered Instructions	# Covered BB Transitions
Raw	56856	23559	2475
Structured	70726	28979	2885

Finally below we report on the BB Transition coverage over time for the Raw and Structured fuzzing campaigns.



The basic block transitions coverage graph of the structured campaign still shows slow growth even after 36h of fuzzing. This suggests that a longer structured fuzzing campaign might be useful in discovering new states.

### 3.6 Conclusion

During our audit we did not encounter serious violations of the outlined properties. All but one of the properties were exercised by our fuzzing campaigns.

Since the property checking campaign was based on fuzzing, this is not a solid guarantee that its not possible to violate the presented properties. Longer fuzzing campaigns, manual auditing and other formal methods would be useful complements to this work.

## 4 Smart Contract Recommendations

The issues are presented in approximate order of priority from highest to lowest.

### 4.1 Improve the semantic meaning of variables in the codebase [Open](#)

Description



In several places in the Nitro codebase, parameter types are void of semantic meaning.

Presenting below a concrete example to illustrate the rationale to follow.

**code/packages/nitro-protocol/contracts/ForceMove.sol:L226-L234**

```
function conclude(
    uint48 largestTurnNum,
    FixedPart memory fixedPart,
    bytes32 appPartHash,
    bytes32 outcomeHash,
    uint8 numStates,
    uint8[] memory whoSignedWhat,
    Signature[] memory sigs
) public override {
```

In the `conclude()` method, inside `ForceMove`, the `numStates` parameter has a type of `uint8`. However, this does not correctly signal the limits of the permitted number of states to be passed onto the underlying functions for validation or any theoretical limits of the system.

For an example on where this gets a contrarian signal we can look at the `checkpoint()` function, which, deeper down the stack ends up calling `_requireValidTransitionChain()`. In `_requireValidTransitionChain()`, as we can see in the example below, the *number of states* we can pass in (the semantic value of the `numStates` variable mentioned above) is “uncapped” (or better, realistically capped at `2^256-1 == MAX_UINT`) because it is derived from the length of the arbitrarily-big dynamically-sized array `variableParts`.

**code/packages/nitro-protocol/contracts/ForceMove.sol:L446-L454**

```
// returns stateHashes array if valid
// else, reverts
uint48 largestTurnNum,
ForceMoveApp.VariablePart[] memory variableParts,
uint8 isFinalCount,
bytes32 channelId,
FixedPart memory fixedPart
) internal pure returns (bytes32[] memory) {
    bytes32[] memory stateHashes = new bytes32[](variableParts.length);
```

Although in a different form, the same line of reasoning can be extended to the fact that the `whoSignedWhat` parameter array in `_validSignatures()` is a dynamically-sized array of the `uint8` type. The rationale for that cap seems arbitrary even though its type is preventing bug-inducing overflows later on in the function call tree.

### Recommendation

This general topic has no easy, clear-cut solution. The audit team’s recommendation would be to reassess both the types of limits imposed on each variable and its business logic meaning. Carefully assess why each component is built or constrained the way it currently is and modify the codebase accordingly, making it more coherent and robust.

## 4.2 Improve input validation, fail early on invalid input Open

### Description

Input validation checks should be explicit and well documented as part of the code’s documentation. This is to make sure that smart-contracts are robust against erroneous inputs and reduce the potential attack surface for exploitation.

It is good practice to verify the method’s input as early as possible and only perform further actions if the validation succeeds.

### Examples

**forceMove**

- Let’s assume someone calls `forceMove` with `variableParts.length == 0`

**code/packages/nitro-protocol/contracts/ForceMove.sol:L46-L73**

```
function forceMove(
    FixedPart memory fixedPart,
    uint48 largestTurnNum,
    ForceMoveApp.VariablePart[] memory variableParts,
    uint8 isFinalCount, // how many of the states are final
    Signature[] memory sigs,
    uint8[] memory whoSignedWhat,
    Signature memory challengerSig
) public override {
    bytes32 channelId = _getChannelId(fixedPart);

    if (_mode(channelId) == ChannelMode.Open) {
        _requireNonDecreasedTurnNumber(channelId, largestTurnNum);
    } else if (_mode(channelId) == ChannelMode.Challenge) {
        _requireIncreasedTurnNumber(channelId, largestTurnNum);
    } else {
        // This should revert.
        _requireChannelNotFinalized(channelId);
    }

    bytes32 supportedStateHash = _requireStateSupportedBy(
        largestTurnNum,
        variableParts,
        isFinalCount,
        channelId,
        fixedPart,
        sigs,
        whoSignedWhat
    );
```

- which calls `_requireStateSupportedBy` which returns an empty `stateHashes` array

```
bytes32[] memory stateHashes = _requireValidTransitionChain(
    largestTurnNum,
    variableParts,
    isFinalCount,
    channelId,
    fixedPart
);
```

- `_validSignatures` then checks whether there are signatures for each state-hash which ultimately fails in an index out of bounds assertion, consuming all gas

```
address signer = _recoverSigner(stateHashes[whoSignedWhat[i]], sigs[i]);
```

The fact that the provided parameters are invalid and would ultimately fail could already be detected early on before spending gas on instructions that ultimately lead to a revert.

Recommendation

Fail early on invalid input.

4.3 Use the type system in parameters Open

Description

Typecasting inside the corpus of a function is unneeded when the type of the parameter is known beforehand.

Examples

```
contract GRTAssetHolder is ERC20AssetHolder {
    IController public Controller;

    constructor(
        address _AdjudicatorAddress,
        address _TokenAddress,
        address _ControllerAddress
    ) public ERC20AssetHolder(_AdjudicatorAddress, _TokenAddress) {
        AdjudicatorAddress = _AdjudicatorAddress;
        Controller = IController(_ControllerAddress);
    }
}
```

```
constructor(address _AdjudicatorAddress, address _TokenAddress) public {
    AdjudicatorAddress = _AdjudicatorAddress;
    Token = IERC20(_TokenAddress);
}
```

Recommendation

Define parameters with the correct type.

4.4 Use existing internal methods instead of reimplementing them inline Open

Description

Duplicated code is hard to maintain and may introduce *out-of-sync* bugs.

A few examples where this is done in the codebase:

- Instead of using `_hashOutcome()` it is sometimes just done directly ( `keccak` ing things)
- `_bytesEqual()` not always used (directly `keccak` ing things)

In another, more representative example, `concludePushOutcomeAndTransferAll()` includes `ForceMove.conclude()` , *ipsis verbis*, in its corpus with just a changed require error message.

This last example very clearly highlights the perils of having code duplication and the difficulty in maintaining different sections of the code that are supposed to be the same in sync.

Recommendation

Call the internal functions directly instead of duplicating code.

4.5 Stick to interface naming conventions Open

Description

Consider prefixing Interface declarations with a capital `I` to make them distinguishable from concrete contract implementations to improve code readability.

For example, `IAssetHolder` , `IForceMove` (is actually an abstract contract) already follow this naming scheme.

Examples

```
interface Adjudicator {
```

code/packages/nitro-protocol/contracts/interfaces/ForceMoveApp.sol:L7-L7

```
interface ForceMoveApp {
```

Recommendation

Indicate that an interface is used by prefixing the interface name with a capital `I`.

4.6 Public functions may be external [Open](#)

Description

Reduce the scope of functions to `external` if they are not meant to be called from within the same contract. This might actually be a non-issue if the listed methods should be callable from a subclass.

Examples

code/packages/nitro-protocol/contracts/ERC20AssetHolder.sol:L29-L34

```
    */
    function deposit(
        bytes32 destination,
        uint256 expectedHeld,
        uint256 amount
    ) public {
```

code/packages/nitro-protocol/contracts/AssetHolder.sol:L114-L114

```
    function transferAll(bytes32 channelId, bytes memory allocationBytes) public override {
```

code/packages/nitro-protocol/contracts/AssetHolder.sol:L152-L156

```
    function claimAll(
        bytes32 guarantorChannelId,
        bytes memory guaranteeBytes,
        bytes memory allocationBytes
    ) public override {
```

- Pushoutcome
- Pushoutcomeandtransferall
- concludePushOutcomeAndTransferAll
- ...

Recommendation

Change `public` to `external` (may require to declare `memory` arrays as `calldata`) for methods that are not meant to be consumed within the same contract.

4.7 Missing declaration of use for SafeMath may break Solidity 0.7.x compatibility [Open](#)

Description

`EthAssetHolder` and `ERC20AssetHolder` are using `SafeMath` for `uint256` but it is not explicitly declared but inherited from the `BaseClass AssetHolder`. Inheriting `using SafeMath for uint256;` declarations will not work with Solidity 0.7.x. It is therefore recommended to explicitly declare it in the inheriting contract as well.

Examples

code/packages/nitro-protocol/contracts/ETHAssetHolder.sol:L8-L17

```
contract ETHAssetHolder is AssetHolder {
    /**
     * @notice Constructor function storing the AdjudicatorAddress.
     * @dev Constructor function storing the AdjudicatorAddress.
     * @param _AdjudicatorAddress Address of an Adjudicator contract, supplied at deploy-time.
     */
    constructor(address _AdjudicatorAddress) public {
        AdjudicatorAddress = _AdjudicatorAddress;
    }
}
```

code/packages/nitro-protocol/contracts/ERC20AssetHolder.sol:L8-L10

```
    */
    contract ERC20AssetHolder is AssetHolder {
        IERC20 public Token;
```

Recommendation

Consider declaring `using SafeMath for uint256;` in contracts using `SafeMath` and inheriting from `AssetHolder`.

4.8 Optimization - `_transferAll()` should break out of loop if `balance <= 0` [Open](#)

Description

The *for* loop mentioned below should break instead of interesting the rest of the allocations. By doing this, we are able to greatly improve the gas efficiency of the routine.

Examples

- There is no benefit of continuing to iterate allocation items when `balance <= 0` :

code/packages/nitro-protocol/contracts/AssetHolder.sol:L24-L40

```

    */
    function _transferAll(bytes32 channelId, bytes memory allocationBytes) internal {
        Outcome.AllocationItem[] memory allocation = abi.decode(
            allocationBytes,
            (Outcome.AllocationItem[])
        );
        uint256 balance = holdings[channelId];
        uint256 numPayouts = 0;
        uint256 numNewAllocationItems = allocation.length;
        uint256 _amount;
        bool overlap;
        uint256 finalPayoutAmount;
        uint256 firstNewAllocationItemAmount;

        for (uint256 i = 0; i < allocation.length; i++) {
            if (balance == 0) {
                // if funds are completely depleted, keep the allocationItem and do not pay out
            }
        }
    }
}
```

- Consider breaking out of the outer loop as well if `balance <= 0` to save some gas:

code/packages/nitro-protocol/contracts/AssetHolder.sol:L197-L204

```

// first increase payouts according to guarantee
for (uint256 i = 0; i < guarantee.destinations.length; i++) {
    // for each destination in the guarantee
    bytes32 _destination = guarantee.destinations[i];
    for (uint256 j = 0; j < allocation.length; j++) {
        if (balance == 0) {
            break;
        }
    }
}
```

Recommendation

Consider breaking from the *for* loop if `balance <= 0` , in the first example.

In the second example, it might be valid to not check `balance` in the outer loop on every iteration to save gas and optimize the *happy path* of the system.

5 Smart Contract Findings

Each issue has an assigned severity:

- **Minor** issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- **Medium** issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- **Major** issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- **Critical** issues are directly exploitable security vulnerabilities that need to be fixed.

5.1 Reentrancy in `claimAll` may allow an external address to drain funds **Major**

Description

When paying out via `claimAll` a recipient may choose to re-enter `claimAll` from `_transferAsset` . This may allow an attacker to get out more funds from the guarantor channel than allowed if there are still funds left in `holdings[guarantorChannelId]` . The `assetOutcomeHashes[guarantee.targetChannelId]` is only set after the transfers are executed and, therefore, the current `assetOutcomeHashes[guarantee.targetChannelId]` is still valid when re-entering.

Examples

additional\_code/GRTAssetHolder/AssetHolder.sol:L259-L294



```

uint256 k = 0;
for (uint256 j = 0; j < allocation.length; j++) {
    // for each destination in the target channel's allocation
    if (allocation[j].amount > 0) {
        newAllocation[k] = allocation[j];
        k++;
    }
    if (payouts[j] > 0) {
        if (_isExternalDestination(allocation[j].destination)) {
            _transferAsset(_bytes32ToAddress(allocation[j].destination), payouts[j]);
            emit AssetTransferred(
                guarantorChannelId,
                allocation[j].destination,
                payouts[j]
            );
        } else {
            holdings[allocation[j].destination] += payouts[j];
        }
    }
}
assert(k == newAllocationLength);

if (newAllocationLength > 0) {
    // store hash
    assetOutcomeHashes[guarantee.targetChannelId] = keccak256(
        abi.encode(
            Outcome.AssetOutcome(
                uint8(Outcome.AssetOutcomeType.Allocation),
                abi.encode(newAllocation)
            )
        )
    );
} else {
    delete assetOutcomeHashes[guarantee.targetChannelId];
}
}

```

### Recommendation

- clear `assetOutcomeHashes[guarantee.targetChannelId]` before performing an external call to a potentially untrusted entity.
- protect the method with a reentrancy guard

## 5.2 Recipient can block transferAsset/payouts for all participants Major

### Description

The push-payment pattern in `_transferAll` and `claimAll` allows one recipient to block all payouts on `_transferAsset`. There are two implementations for an `AssetHolder` that implement `_transferAsset`, one for `EthAssetHolder` performing an `<address>.transfer(<amount>)` that can be forced to throw by the recipient, and one for an `Erc20AssetHolder`. The latter can potentially throw when using `ERC20` tokens with a callback or `ERC20` compliant `ERC777` token variants in the system.

Furthermore, it is not recommended to use `address.send()/address.transfer()` anymore, du to potentially changing gas prices in the future.

### Examples

- `_transferAll`

additional\_code/GRTAssetHolder/AssetHolder.sol:L88-L101

```

uint256 payoutAmount;
for (uint256 m = 0; m < numPayouts; m++) {
    if (overlap && m == numPayouts - 1) {
        payoutAmount = finalPayoutAmount;
    } else {
        payoutAmount = allocation[m].amount;
    }
    if (_isExternalDestination(allocation[m].destination)) {
        _transferAsset(_bytes32ToAddress(allocation[m].destination), payoutAmount);
        emit AssetTransferred(channelId, allocation[m].destination, payoutAmount);
    } else {
        holdings[allocation[m].destination] += payoutAmount;
    }
}
}

```

- `claimAll`

additional\_code/GRTAssetHolder/AssetHolder.sol:L260-L278

```

for (uint256 j = 0; j < allocation.length; j++) {
    // for each destination in the target channel's allocation
    if (allocation[j].amount > 0) {
        newAllocation[k] = allocation[j];
        k++;
    }
    if (payouts[j] > 0) {
        if (_isExternalDestination(allocation[j].destination)) {
            _transferAsset(_bytes32ToAddress(allocation[j].destination), payouts[j]);
            emit AssetTransferred(
                guarantorChannelId,
                allocation[j].destination,
                payouts[j]
            );
        } else {
            holdings[allocation[j].destination] += payouts[j];
        }
    }
}
}

```

- `ETHAssetHolder` implementation

**code/packages/nitro-protocol/contracts/ETHAssetHolder.sol:L64-L66**

```
function _transferAsset(address payable destination, uint256 amount) internal override {
    destination.transfer(amount);
}
```

- `GRTAssetHolder` if `Token` has callbacks or fails to return `true` on `transfer` (broken token) (or `recipient` , `sender` is `address(0x0)` )

**additional\_code/GRTAssetHolder/GRTAssetHolder.sol:L31-L40**

```
function _transferAsset(address payable destination, uint256 amount) internal override {
    IStaking _staking = staking();

    if (_staking.isChannel(destination)) {
        _staking.collect(amount, destination);
        return;
    }

    require(Token.transfer(destination, amount), "GRTAssetHolder: transferring tokens failed");
}
```

Recommendation

use `address.call{value: x}()` instead. Note that this may allow the external contract to reenter ([issue 5.1](#))

5.3 Gas siphoning attack possible in “transfer all” and “claim all”-related methods Medium

Description

Nitro protocol implements a push-style payment mechanism to all the participants in the channel upon its conclusion. This means that one actor is responsible for single-handedly paying the gas costs of the transaction that disburses funds to other players in the channel, which are untrusted by definition.

When disbursing these funds through the *asset holder* there is a necessary external call being made to an address owned by each one of the players. This, in turn, means that we are, potentially, handing over execution control to on-chain code deployed by another player and we are, therefore, at his mercy regarding gas usage.

The other player now has an opportunity to profit off of the gas paid for by the “withdrawer” by minting any flavor of the *gas tokens* available on the main net today.

This attack is, however, only possible when dealing with complete-gas-forwarding methods that are available when dealing with specific implementations of asset holders. More specifically, any modification or augmentation of the ERC20 standard that has callback functionality (e.g., ERC777) or if using the `.call()` method in an asset holder of the native token.

Examples

**code/packages/nitro-protocol/contracts/ERC20AssetHolder.sol:L70-L72**

```
function _transferAsset(address payable destination, uint256 amount) internal override {
    Token.transfer(destination, amount);
}
```

**packages/nitro-protocol/contracts/ETHAssetHolder.sol:L69-L72**

```
function _transferAsset(address payable destination, uint256 amount) internal override {
    (bool success, ) = destination.call{value: amount}('');
    require(success, 'Could not transfer asset');
}
```

- Please note that the second example provided above was pulled from a newer iteration of the codebase and does not correspond to the original commit hash under scrutiny. We have tried to accompany the flux of the codebase and provide real-time aid to the development team.

Recommendation

There are several possible solutions to this issue.

The first, simpler but also less sophisticated, was to provide a stipend to all of these external calls, the same way that Solidity does, by calculating a minimum amount of gas to be forwarded that allows a set of operations we want to permit on the untrusted payee’s side. On native token payments 2300 gas is a standard choice (not allowing to change state), however, on the token side, this calculation gets trickier since it would heavily depend on the token’s implementation.

The second, more involved but also more sound solution would be to keep a counter of how much gas was used on the external call by each participant and then pay the transaction executor accordingly from the payee’s pot.

This second solution is, however, not trivial to implement, and depending on the asset under consideration, the peg to ETH might be hard to calculate. That is why the easiest way to set this sort of stipend tied to the asset would be to make it a new parameter that each player needs to sign on, agreeing what the payout executor would get for each external call per payee.

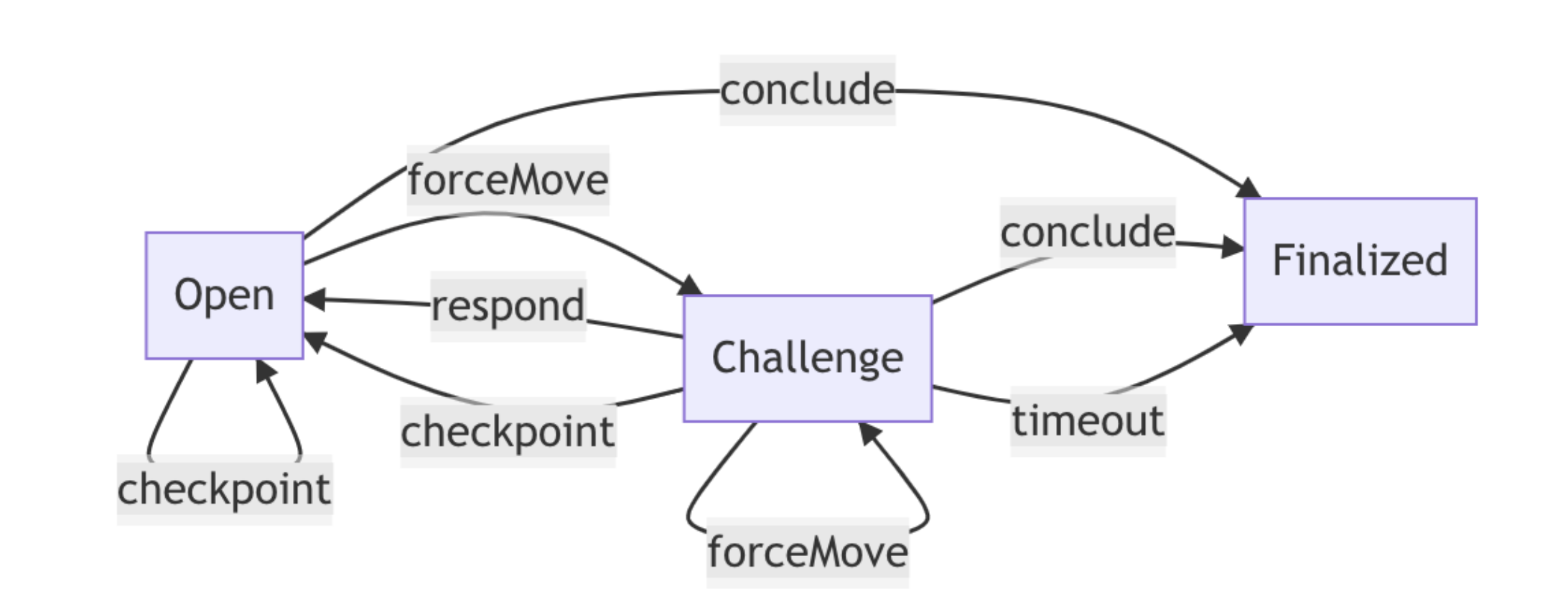
5.4 A channel established with a `FixedPart.challengeDuration == 0` can be challenged and finalized immediately Medium

Description

If a state-channel is challenged by calling `forceMove` it is auto-finalized after `FixedPart.challengeDuration` seconds. Allowing the `challengeDuration` to be zero would allow a participant to challenge and finalize a challenge at-will within one block without

allowing the counterparty to `resolve` or `checkpoint` the challenged channel.

The following state-diagram outlines valid state-channel transitions:



[source](#)

The `challengeDuration` is part of the negotiated channel duration. A counterparty has to agree to the fixed channel parameters to join a channel and may choose not to do so if they don't like the parameters. However, we have not found implementation to actually verify the validity of this parameter.

### Examples

- `challengeDuration` is part of the fixed channel parameters

**code/packages/nitro-protocol/contracts/interfaces/IForceMove.sol:L9-L22**

```
abstract contract IForceMove {
    struct Signature {
        uint8 v;
        bytes32 r;
        bytes32 s;
    }

    struct FixedPart {
        uint256 chainId;
        address[] participants;
        uint48 channelNonce;
        address appDefinition;
        uint48 challengeDuration;
    }
}
```

### Recommendation

Implementations should check for the `challengeDuration` to be within acceptable bounds (multiple blocks, not too long to never finalize). Consider exposing a smart contract function to validate the bare minimum of parameters allowed for valid state-channels. Have implementations check new channels using this method to allow them to detect maliciously configured state channels early. Provide guidelines on what should be considered as invalid state channel parameters.

## 5.5 Channel parameters may hash to a channelId hinting an external destination potentially losing funds Medium

### Description

An external destination may be represented by a `channelId` that has the first 12 bytes set to zero. However, there is a chance that a channel actually hashes to something that would be used as an external destination while it should be used as a `channelId` for internal accounting.

### Examples

**code/packages/nitro-protocol/contracts/ForceMove.sol:L773-L783**

```
/**
 * @notice Computes the unique id of a channel.
 * @dev Computes the unique id of a channel.
 * @param fixedPart Part of the state that does not change
 * @return channelId
 */
function _getChannelId(FixedPart memory fixedPart) internal pure returns (bytes32 channelId) {
    channelId = keccak256(
        abi.encode(fixedPart.chainId, fixedPart.participants, fixedPart.channelNonce)
    );
}
```

### Recommendation

- add a flag to indicate whether it is a `channelId` or external address
- or make sure that whoever generates a `channelId` verifies that the first 12 bytes are non-zero

## 5.6 AttestationApp - Bytes comparison should enforce length check Minor

### Description

Similar to [issue 5.10](#) the bytes comparison should enforce a length check before actually invoking the hashed-comparison. Consider re-using a length-checked variant of `_bytesEqual` as mentioned in [issue 4.4](#).

Examples

additional\_code/the-graph/packages/statechannels-contracts/contracts/AttestationApp.sol:L75-L78

```
require(
    keccak256(abi.encode(computedStateB)) == keccak256(b.appData),
    "AttestationApp: Proposed 'to' appData is not a valid transition from 'from' appData"
);
```

additional\_code/the-graph/packages/statechannels-contracts/contracts/AttestationApp.sol:L86-L89

```
require(
    keccak256(abi.encode(computedOutcomeB)) == keccak256(b.outcome),
    "AttestationApp: Proposed 'to' outcome is not a valid transition from 'from' outcome"
);
```

Recommendation

See [issue 4.4](#) and [issue 5.10](#).

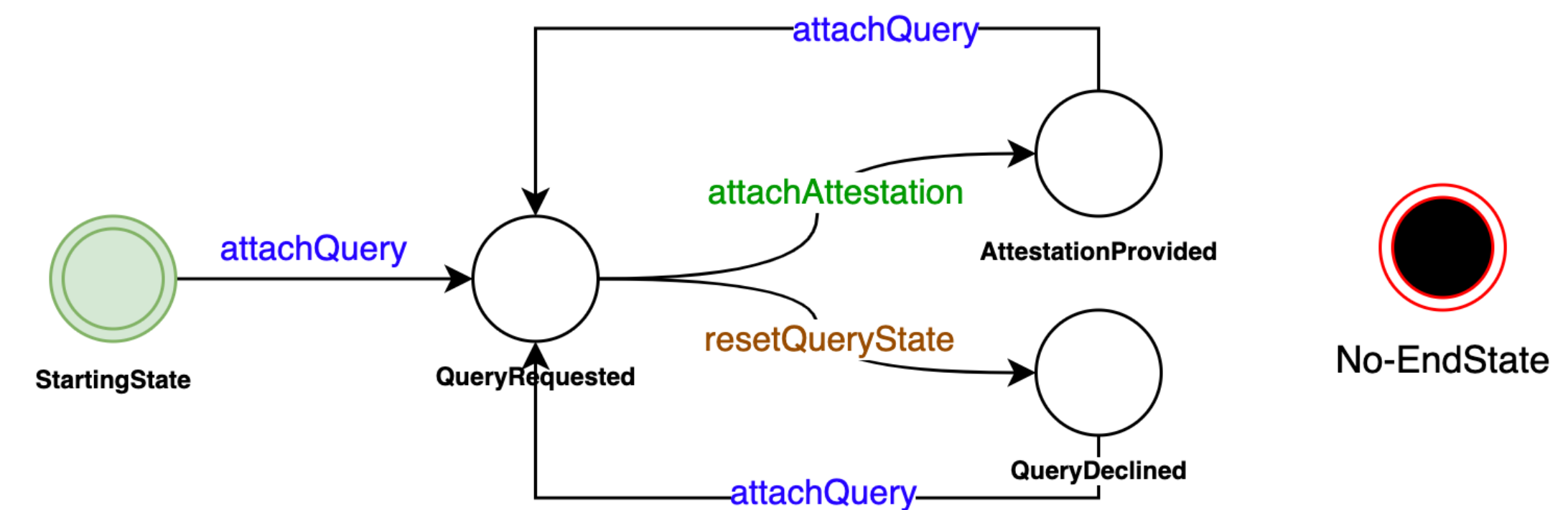
5.7 AttestationApp - De-duplicate state machine states Minor

Description

State-machines typically add complexity and risk to smart contract systems. In order to reduce complexity, it should be considered if any states in a state machine are actually identical. In the `AttestationApp` this seems to be the case for `StartingState` and `QueryDeclined`. The transition `resetQueryState` that moves the AppData to `QueryDeclined` effectively resets all fields as if they were in `StartingState`.

Examples

StartingState, // Starting state  
QueryRequested,  
AttestationProvided,  
QueryDeclined



Recommendation

Simplify the state machine by de-duplicating states `QueryDeclined` and `StartingState`.

5.8 Documentation inconsistency Minor Pending

Resolution
This has been addressed by <a href="#">statechannels/statechannels#2775</a> , however, the documentation seems to be slightly wrong as <code>finalized</code> is <code>finalizesAt &lt;= currentTime</code> instead of <code>finalizesAt &lt; currentTime</code> (documentation).

Description

The [documentation](#) states that a state-channel is finalized if the `finalizesAt >= currrentTime` which is incorrect. The implementation in the smart contract, however, is correct.

Examples



# Channel Modes

- **Open** if and only if `finalizesAt` is null
  - implies that `stateHash` and `challengerAddress` are also null
- **Challenge** if and only if `finalizesAt < currentTime`
  - implies that all other fields are not null
- **Finalized** if and only if `finalizesAt >= currentTime`
  - implies that all other fields are not null

These states can be represented in the following state machine:

code/packages/nitro-protocol/contracts/ForceMove.sol:L650-L662

```
function _mode(bytes32 channelId) internal view returns (ChannelMode) {
    // Note that _getChannelStorage(someRandomChannelId) returns (0,0,0), which is
    // correct when nobody has written to storage yet.

    (, uint48 finalizesAt, ) = _getChannelStorage(channelId);
    if (finalizesAt == 0) {
        return ChannelMode.Open;
    } else if (finalizesAt <= now) {
        return ChannelMode.Finalized;
    } else {
        return ChannelMode.Challenge;
    }
}
```

## Recommendation

Review the documentation and correct the inconsistency:

- Challenge if and only if `finalizesAt > currentTime`
- Finalized if and only if `finalizesAt <= currentTime`

## 5.9 Signature validation may allow invalid signatures if a channel is established with `address(0)` as one of the participants Minor

### Description

The smart contract uses `ecrecover` to recover the signer of a message. In case of an error, the method would return `address(0x0)` . Since the smart contract does not enforce the signature validation to fail if `ecrecover` returns an `address(0x0)` it would therefore accept an invalid signature if the participants array contains the `address(0x0)` . Now, it is highly unlikely that channel participants would agree to establish a channel with one participant being `address(0x0)` , however, we couldn't find a check for this case in an implementation nor does the smart contract clearly rejects this case.

**Note:** It is very unlikely that a channel would be successfully established with one participant being `address(0x0)` , hence the severity classification of Minor .

### Examples

- `_recoverSigner`

code/packages/nitro-protocol/contracts/ForceMove.sol:L386-L390

```
function _recoverSigner(bytes32 _d, Signature memory sig) internal pure returns (address) {
    bytes32 prefixedHash = keccak256(abi.encodePacked(prefix, _d));
    address a = ecrecover(prefixedHash, sig.v, sig.r, sig.s);
    return (a);
}
```

code/packages/nitro-protocol/contracts/ForceMove.sol:L160-L164

```
require(
    _recoverSigner(responseStateHash, sig) ==
        fixedPart.participants[(turnNumRecord + 1) % fixedPart.participants.length],
    'Response not signed by authorized mover'
);
```

code/packages/nitro-protocol/contracts/ForceMove.sol:L282-L292

```
function _requireChallengerIsParticipant(
    bytes32 supportedStateHash,
    address[] memory participants,
    Signature memory challengerSignature
) internal pure returns (address challenger) {
    challenger = _recoverSigner(
        keccak256(abi.encode(supportedStateHash, 'forceMove')),
        challengerSignature
    );
    require(!_isAddressInArray(challenger, participants), 'Challenger is not a participant');
}
```

```
function _validSignatures(
    uint48 largestTurnNum,
    address[] memory participants,
    bytes32[] memory stateHashes,
    Signature[] memory sigs,
    uint8[] memory whoSignedWhat // whoSignedWhat[i] is the index of the state in stateHashes that was signed by participants[i]
) internal pure returns (bool) {
    uint256 nParticipants = participants.length;
    uint256 nStates = stateHashes.length;

    require(
        _acceptableWhoSignedWhat(whoSignedWhat, largestTurnNum, nParticipants, nStates),
        'Unacceptable whoSignedWhat array'
    );
    for (uint256 i = 0; i < nParticipants; i++) {
        address signer = _recoverSigner(stateHashes[whoSignedWhat[i]], sigs[i]);
        if (signer != participants[i]) {
            return false;
        }
    }
    return true;
}
```

Recommendation

Reject invalid signatures by throwing in case `erecover` returns `address(0x0)` . Consider providing a method to allow implementations to verify a minimum set of sane channel parameter and value bounds as outlined in [issue 5.4](#). Disallow `address(0x0)` being part of the participants’ array.

5.10 `_bytesEqual()` should enforce a length check Minor

Description

`_bytesEqual()` verifies whether `A` equals `B` by checking `keccak(A)==keccak(B)` . However, this would potentially return `true` for the unlikely case where `A` is not even of length `B` but still hashes to the same value.

Examples

```
/**
 * @notice Check for equality of two byte strings
 * @dev Check for equality of two byte strings
 * @param left One bytes string
 * @param right The other bytes string
 * @return true if the bytes are identical, false otherwise.
 */
function _bytesEqual(bytes memory left, bytes memory right) internal pure returns (bool) {
    return keccak256(left) == keccak256(right);
}
```

Recommendation

Enforce that `A.length==B.length` . While consuming slightly more gas for the check this has the positive side-effect of wasting less gas if there is a length mismatch.

6 Application Findings

Each issue has an assigned severity:

- Minor issues are subjective in nature. They are typically suggestions around best practices or readability. Code maintainers should use their own judgment as to whether to address such issues.
- Medium issues are objective in nature but are not security vulnerabilities. These should be addressed unless there is a clear reason not to.
- Major issues are security vulnerabilities that may not be directly exploitable or may require certain conditions in order to be exploited. All major issues should be addressed.
- Critical issues are directly exploitable security vulnerabilities that need to be fixed.

6.1 `server-wallet` - `validateTransition` fails to return on turn number check failed Major

Description

If the application is not the Null-App `validateTransition` checks if the `turnNum` was incremented by one. Upon detecting that the `turnNum` was not increased, or increased by an unsupported amount the application prints an error. However, `validateTransition` continues to process the state transition instead of returning `false` indicating that the validation failed.

Examples

```
if (fromState.appDefinition !== constants.AddressZero) {
    // turn numbers not relevant for the null app
    const turnNumCheck = _.isEqual(toState.turnNum, fromState.turnNum + 1);
    if (!turnNumCheck) {
        const VALIDATION_ERROR = `Turn number check failed.`;
        logger.error({fromState, toState, error: Error(VALIDATION_ERROR)}, VALIDATION_ERROR);
    }
}
```

Recommendation

exit validateTransition returning false when detecting a protocol violation.

6.2 server-wallet - Processing states with zero participants may cause an exception Major

Description

validateTransition does not validate that inputs provided to the function are actually within valid bounds. For example, participants may be an empty array leading to participants.length returning zero which will quickly trigger an exception validateTransition or even before.

We also haven't observed any concrete checks for participants.length being non-zero upstream to validateTransition . The wire-format validation checks only check the type structure schema but does not enforce that certain fields are actually set.

Examples

- consider fromState.participants.length=0
- fromMoverIndex = fromState.turnNum % fromState.participants.length; will div zero, hence return NaN
- const fromMover = fromState.participants[NaN].signingAddress; will throw with and TypeError due to an illegal property access undefined.signingAddress

code/packages/server-wallet/src/utilities/validate-transition.ts:L28-L32

```
const fromMoverIndex = fromState.turnNum % fromState.participants.length;
const fromMover = fromState.participants[fromMoverIndex].signingAddress;

const toMoverIndex = toState.turnNum % toState.participants.length;
const toMover = toState.participants[toMoverIndex].signingAddress;
```

There are likely more functions that assume that valid input was provided without actually verifying input variable bounds.

- server-wallet pushMessage only performs wire-data scheme validation without checking bounds for arrays.

code/packages/server-wallet/src/wallet/index.ts:L569-L582

```
async pushMessage(rawPayload: unknown): Promise<MultipleChannelOutput> {
  if (this.walletConfig.workerThreadAmount > 0) {
    return this.workerManager.pushMessage(rawPayload);
  } else {
    const wirePayload = validatePayload(rawPayload);
    return this.pushMessageInternal(wirePayload).catch(e => {
      throw new PushMessageError('Error during pushMessage', {
        thisWalletVersion: WALLET_VERSION,
        payloadWalletVersion: wirePayload.walletVersion,
        cause: e,
      });
    });
  }
}
```

Recommendation

validateTransition is a security-critical function. Make sure that all inputs are properly validated (especially pushMessage and `before using them. Ensure that a message being processed by either component cannot throw an exception that may cause the wallet/gateway/indexer to shut down (Denial-of-Service).

6.3 sever-wallet - validateTransition is missing a check for channelNonce Major

Description

The constantsCheck in validateTransition is missing a check for channelNonce to match.

Examples

- server-wallet - missing check for channelNonce

code/packages/server-wallet/src/utilities/validate-transition.ts:L42-L47

```
const constantsCheck =
  _.isEqual(toState.chainId, fromState.chainId) &&
  _.isEqual(toState.participants, fromState.participants) &&
  _.isEqual(toState.appDefinition, fromState.appDefinition) &&
  _.isEqual(toState.challengeDuration, fromState.challengeDuration);
```

- a note in ForceMove indicating that channelNonce must be checked

code/packages/nitro-protocol/contracts/ForceMove.sol:L500-L500

```
// chainId, participants, channelNonce, appDefinition, challengeDuration
```

Recommendation

Validate that channelNonce matches in fromState and toState . Consider renaming constantsCheck to something more expressive, like isValidConstantFixedPart .

6.4 server-wallet - validateTransition checks differ from ForceMove.\_requireValidTransition Major



## Description

The checks performed in server-wallet `validateTransition` differ from the checks performed in the `ForceMove` contracts `_requireValidTransition`. Our assumption is that these checks should be logically equal.

It is unclear what effect this may have on the security of the system but any deviations should be avoided. Furthermore, having to maintain two implementations of validation checks (Smart Contract, App Implementations) decreases maintainability increasing the risk that not all necessary checks are always performed (or components get out of sync)

## Examples

the smart contract checks the following (considered the reference implementation for the checks)

**code/packages/nitro-protocol/contracts/ForceMove.sol:L492-L534**

```
function _requireValidTransition(
    uint256 nParticipants,
    bool[2] memory isFinalAB, // [a.isFinal, b.isFinal]
    ForceMoveApp.VariablePart[2] memory ab, // [a,b]
    uint48 turnNumB,
    address appDefinition
) internal pure returns (bool) {
    // a prior check on the signatures for the submitted states implies that the following fields are equal for a and b:
    // chainId, participants, channelNonce, appDefinition, challengeDuration
    // and that the b.turnNum = a.turnNum + 1
    if (isFinalAB[1]) {
        require(
            _bytesEqual(ab[1].outcome, ab[0].outcome),
            'InvalidTransitionError: Cannot move to a final state with a different default outcome'
        );
    } else {
        require(
            !isFinalAB[0],
            'InvalidTransitionError: Cannot move from a final state to a non final state'
        );
        if (turnNumB < 2 * nParticipants) {
            require(
                _bytesEqual(ab[1].outcome, ab[0].outcome),
                'InvalidTransitionError: Cannot change the default outcome during setup phase'
            );
            require(
                keccak256(ab[1].appData) == keccak256(ab[0].appData),
                'InvalidTransitionError: Cannot change the appData during setup phase'
            );
        } else {
            require(
                ForceMoveApp(appDefinition).validTransition(
                    ab[0],
                    ab[1],
                    turnNumB,
                    nParticipants
                )
            );
            // reason string not necessary (called function will provide reason for reverting)
        }
    }
    return true;
}
```

- inconsistencies:
  - the order of checks is different; the outcome of checks is different
  - if `toState.isFinal` the smart contract checks that the outcome hasn't changed and returns, otherwise throws. The server-wallet, however, continues validating the transition if `toState.isFinal` and the outcomes are equal. This may even then cause `validateTransition` to return `true` as the EVM validation is skipped if `toState.isFinal = true`.
  - `fromState.final` is only checked in the funding stage in the server-wallet, while the smart contract always checks this requirement.
- as mentioned in [issue 6.3](#), the Smart contracts ensures that constant fields including `ChannelNonce` match, while the application doesn't.
- misleading validation name `signatureValidation` does not actually validate signatures and only checks if an expected mover was found in the `signatures` array. Consider renaming to accurately reflect what check is being performed (e.g. `moverSignaturesProvided`)

**code/packages/server-wallet/src/utilities/validate-transition.ts:L62-L69**

```
const signatureValidation =
    (fromState.signatures || []).some(s => s.signer === fromMover) &&
    (toState.signatures || []).some(s => s.signer === toMover);

if (!signatureValidation) {
    const VALIDATION_ERROR = `Signature validation failed.`;
    logger.error({fromState, toState, error: Error(VALIDATION_ERROR)}, VALIDATION_ERROR);
    return false;
}
```

## Recommendation

Consider calling out to the smart contract to validate state transitions, deduplicating the check, and reducing the risk that one application performs different checks to the smart contract.

## 6.5 server-wallet - Plaintext privatekey stored in database Major

### Description



On at least on occasion a private key is stored in the database unencrypted. Depending on the configuration of the database server this may put users’ credentials at risk. It is highly recommended to avoid storing secrets in the application at all and to apply adequate security measures.

Examples

additional\_code/the-graph/packages/receipt-manager/src/receipt-manager.ts:L35-L49

```
async migrateWalletDB(): Promise<void> {
  this.logger.info('Migrate server-wallet database');
  await this.wallet.dbAdmin().migrateDB();

  try {
    const {address} = new ethers.Wallet(this.privateKey);
    await this.wallet.knex.table('signing_wallets').insert({
      private_key: this.privateKey,
      address
    });
  } catch (err) {
    if (err.constraint !== 'signing_wallets_private_key_unique') {
      throw err;
    }
  }
}
```

code/packages/server-wallet/src/wallet/store.ts:L107-L112

```
const signingWallet = await SigningWallet.query(this.knex)
  .insert({
    privateKey: randomWallet.privateKey,
    address: randomWallet.address,
  })
  .returning('*');
```

Recommendation

Adequately protect data at rest enforcing database encryption with an application-specific key and field-based encryption for sensitive information like a user’s private key.

6.6 server-wallet - The wallet does not account for chain reorgs by waiting for confirmations to events received Major

Description

Chain reorg events can be problematic for the applications built on top of Nitro protocol. This manifests itself in two different forms.

Firstly, since server-wallet persists on-chain state locally, the application needs to make sure that its internal state is befitting with the chain at all times. This can be problematic for the first channel funder as the code might assume that the transaction went through and keeps this state in its local database. Whenever chain reorgs happen and make it so that the funding never actually happened, the application still assumes this is the case. The channel will become stale.

Secondly, all participants in the game must watch events coming from on-chain interactions, and, as described above, persist them in their local database. This means, that they must make sure that there are no reorgs to the chain and that their internal state always mirrors the longest chain. Otherwise, a player might assume that a channel is funded when, actually, a chain reorg made it so that another player has not correctly funded the channel in question. This is what we could, effectively, call a “double spend”.

Recommendation

Generally speaking, the recommendation for both scenarios is that we should wait accordingly after any on-chain event to persist state locally.

More specifically, for the first scenario mentioned, we should make sure that sendTransaction() only persists the state to the internal database after it has seen X blocks being mined after the transaction was included in the chain.

For the second scenario, the recommendation is to make sure that after the events are caught by the server-wallet , it should wait for X blocks after the one on which the received event was emitted.

For both recommendations, and depending on the risk profile of each, specific application to be run on top of Nitro, the minimum amount of blocks to wait should be between 6 to 25 blocks.

6.7 server-wallet - Remove and decouple test code from production code Medium

Description

Avoid mixing test-code and test-functionality with production code as this increases the likelihood of certain security controls not being enforced putting the system at risk.

Examples

- validate-transition has an option to skip evm validation which can be enabled by setting process.env.SKIP\_EVM\_VALIDATION to a value that resolves to boolean true (e.g. any non-zero length string)

code/packages/server-wallet/src/utilities/validate-transition.ts:L97-L108

```
if (!skipEvmValidation && !isInFundingStage && !toState.isFinal) {
  const evmValidation = await validateTransitionWithEVM(
    toNitroState(fromState),
    toNitroState(toState),
    bytecode
  );

  if (!evmValidation) {
    logger.error({fromState, toState}, 'EVM Validation failure');
    return false;
  }
}
```

- in `defaultTestConfig` `process.env.SKIP_EVM_VALIDATION` is enabled by default, which does not seem to be used anywhere.

## code/packages/server-wallet/src/config.ts:L63-L67

```
export const defaultTestConfig = {
  ...defaultConfig,
  skipEvmValidation: (process.env.SKIP_EVM_VALIDATION || 'true').toLowerCase() === 'true',
  postgresPoolSize: {max: 1, min: 0},
};
```

- in `channel-manager.ts` there are many instances of hardcoded values that are unusable in a production environment:

**additional\_code/the-graph/packages/payment-manager/src/channel-manager.ts:L23-L26**

```
export const ATTESTATION_APP_ADDRESS = '0x1111111111111111111111111111111111111111111111111111111111111111';  
export const PAYMENT_CHANNEL_STARTING_AMOUNT = BN.from(1_000_000_000);  
export const PAYMENT_CHANNEL_TOKEN_ADDRESS = ethers.constants.AddressZero; // TODO: Is this the asset holder address? Should it b  
export const GATEWAY_DESTINATION_ADDRESS = ethers.constants.AddressZero; // TODO (Liam) The gateway needs a destination address
```

- open-channel has a `fake` funding strategy that bypasses checks

## code/packages/server-wallet/src/protocols/open-channel.ts:L61-L66

```
function isFunded(ps: ProtocolState): boolean {
  const {funding, supported, fundingStrategy} = ps.app;

  switch (fundingStrategy) {
    case 'Fake':
      return true;
  }
}
```

- Wallet constructor silently falls back to ChainServiceMock if the configuration is missing.

## code/packages/server-wallet/src/wallet/index.ts:L169-L176

```
if (walletConfig?.rpcEndpoint && walletConfig.serverPrivateKey) {
    this.chainService = new ChainService(walletConfig.rpcEndpoint, walletConfig.serverPrivateKey);
} else {
    this.logger.debug(
        'rpcEndpoint and serverPrivateKey must be defined for the wallet to use chain service'
    );
    this.chainService = new MockChainService();
}
```

- FakeChain service

## code/packages/xstate-wallet/src/chain.ts:L67-L67

```
export class FakeChain implements Chain {
```

## Recommendation

Remove hardcoded values and features used solely for testing from production code.

Create dedicated test-classes if there is a need to, for example, disable EVM validation or use addresses hard coded for testing.

## 6.8 Repositories under review leak a number of private keys Minor

Description	
1	1. The first row of the table contains the header information, including the title, author, and date.
2	2. The second row of the table contains the first column of data, which is the name of the first person.
3	3. The third row of the table contains the second column of data, which is the name of the second person.
4	4. The fourth row of the table contains the third column of data, which is the name of the third person.
5	5. The fifth row of the table contains the fourth column of data, which is the name of the fourth person.
6	6. The sixth row of the table contains the fifth column of data, which is the name of the fifth person.
7	7. The seventh row of the table contains the sixth column of data, which is the name of the sixth person.
8	8. The eighth row of the table contains the seventh column of data, which is the name of the seventh person.
9	9. The ninth row of the table contains the eighth column of data, which is the name of the eighth person.
10	10. The tenth row of the table contains the ninth column of data, which is the name of the ninth person.
11	11. The eleventh row of the table contains the tenth column of data, which is the name of the tenth person.
12	12. The twelfth row of the table contains the eleventh column of data, which is the name of the eleventh person.
13	13. The thirteenth row of the table contains the twelfth column of data, which is the name of the twelfth person.
14	14. The fourteenth row of the table contains the thirteenth column of data, which is the name of the thirteenth person.
15	15. The fifteenth row of the table contains the fourteenth column of data, which is the name of the fourteenth person.
16	16. The sixteenth row of the table contains the fifteenth column of data, which is the name of the fifteenth person.
17	17. The seventeenth row of the table contains the sixteenth column of data, which is the name of the sixteenth person.
18	18. The eighteenth row of the table contains the seventeenth column of data, which is the name of the seventeenth person.
19	19. The nineteenth row of the table contains the eighteenth column of data, which is the name of the eighteenth person.
20	20. The twentieth row of the table contains the nineteenth column of data, which is the name of the nineteenth person.
21	21. The twenty-first row of the table contains the twentieth column of data, which is the name of the twentieth person.
22	22. The twenty-second row of the table contains the twenty-first column of data, which is the name of the twenty-first person.
23	23. The twenty-third row of the table contains the twenty-second column of data, which is the name of the twenty-second person.
24	24. The twenty-fourth row of the table contains the twenty-third column of data, which is the name of the twenty-third person.
25	25. The twenty-fifth row of the table contains the twenty-fourth column of data, which is the name of the twenty-fourth person.
26	26. The twenty-sixth row of the table contains the twenty-fifth column of data, which is the name of the twenty-fifth person.
27	27. The twenty-seventh row of the table contains the twenty-sixth column of data, which is the name of the twenty-sixth person.
28	28. The twenty-eighth row of the table contains the twenty-seventh column of data, which is the name of the twenty-seventh person.
29	29. The twenty-ninth row of the table contains the twenty-eighth column of data, which is the name of the twenty-eighth person.
30	30. The thirtieth row of the table contains the twenty-ninth column of data, which is the name of the twenty-ninth person.
31	31. The thirty-first row of the table contains the thirtieth column of data, which is the name of the thirtieth person.
32	32. The thirty-second row of the table contains the thirty-first column of data, which is the name of the thirty-first person.
33	33. The thirty-third row of the table contains the thirty-second column of data, which is the name of the thirty-second person.
34	34. The thirty-fourth row of the table contains the thirty-third column of data, which is the name of the thirty-third person.
35	35. The thirty-fifth row of the table contains the thirty-fourth column of data, which is the name of the thirty-fourth person.
36	36. The thirty-sixth row of the table contains the thirty-fifth column of data, which is the name of the thirty-fifth person.
37	37. The thirty-seventh row of the table contains the thirty-sixth column of data, which is the name of the thirty-sixth person.
38	38. The thirty-eighth row of the table contains the thirty-seventh column of data, which is the name of the thirty-seventh person.
39	39. The thirty-ninth row of the table contains the thirty-eighth column of data, which is the name of the thirty-eighth person.
40	40. The fortieth row of the table contains the thirty-ninth column of data, which is the name of the thirty-ninth person.
41	41. The forty-first row of the table contains the fortieth column of data, which is the name of the fortieth person.
42	42. The forty-second row of the table contains the forty-first column of data, which is the name of the forty-first person.
43	43. The forty-third row of the table contains the forty-second column of data, which is the name of the forty-second person.
44	44. The forty-fourth row of the table contains the forty-third column of data, which is the name of the forty-third person.
45	45. The forty-fifth row of the table contains the forty-fourth column of data, which is the name of the forty-fourth person.
46	46. The forty-sixth row of the table contains the forty-fifth column of data, which is the name of the forty-fifth person.
47	47. The forty-seventh row of the table contains the forty-sixth column of data, which is the name of the forty-sixth person.
48	48. The forty-eighth row of the table contains the forty-seventh column of data, which is the name of the forty-seventh person.
49	49. The forty-ninth row of the table contains the forty-eighth column of data, which is the name of the forty-eighth person.
50	50. The fiftieth row of the table contains the forty-ninth column of data, which is the name of the forty-ninth person.
51	51. The fifty-first row of the table contains the fiftieth column of data, which is the name of the fiftieth person.
52	52. The fifty-second row of the table contains the fifty-first column of data, which is the name of the fifty-first person.
53	53. The fifty-third row of the table contains the fifty-second column of data, which is the name of the fifty-second person.
54	54. The fifty-fourth row of the table contains the fifty-third column of data, which is the name of the fifty-third person.
55	55. The fifty-fifth row of the table contains the fifty-fourth column of data, which is the name of the fifty-fourth person.
56	56. The fifty-sixth row of the table contains the fifty-fifth column of data, which is the name of the fifty-fifth person.
57	57. The fifty-seventh row of the table contains the fifty-sixth column of data, which is the name of the fifty-sixth person.
58	58. The fifty-eighth row of the table contains the fifty-seventh column of data, which is the name of the fifty-seventh person.
59	59. The fifty-ninth row of the table contains the fifty-eighth column of data, which is the name of the fifty-eighth person.
60	60. The sixtieth row of the table contains the fifty-ninth column of data, which is the name of the fifty-ninth person.
61	61. The sixty-first row of the table contains the sixtieth column of data, which is the name of the sixtieth person.
62	62. The sixty-second row of the table contains the sixty-first column of data, which is the name of the sixty-first person.
63	63. The sixty-third row of the table contains the sixty-second column of data, which is the name of the sixty-second person.
64	64. The sixty-fourth row of the table contains the sixty-third column of data, which is the name of the sixty-third person.
65	65. The sixty-fifth row of the table contains the sixty-fourth column of data, which is the name of the sixty-fourth person.
66	66. The sixty-sixth row of the table contains the sixty-fifth column of data, which is the name of the sixty-fifth person.
67	67. The sixty-seventh row of the table contains the sixty-sixth column of data, which is the name of the sixty-sixth person.
68	68. The sixty-eighth row of the table contains the sixty-seventh column of data, which is the name of the sixty-seventh person.
69	69. The sixty-ninth row of the table contains the sixty-eighth column of data, which is the name of the sixty-eighth person.
70	70. The seventieth row of the table contains the sixty-ninth column of data, which is the name of the sixty-ninth person.
71	71. The seventy-first row of the table contains the seventieth column of data, which is the name of the seventieth person.
72	72. The seventy-second row of the table contains the seventy-first column of data, which is the name of the seventy-first person.
73	73. The seventy-third row of the table contains the seventy-second column of data, which is the name of the seventy-second person.
74	74. The seventy-fourth row of the table contains the seventy-third column of data, which is the name of the seventy-third person.
75	75. The seventy-fifth row of the table contains the seventy-fourth column of data, which is the name of the seventy-fourth person.
76	76. The seventy-sixth row of the table contains the seventy-fifth column of data, which is the name of the seventy-fifth person.
77	77. The seventy-seventh row of the table contains the seventy-sixth column of data, which is the name of the seventy-sixth person.
78	78. The seventy-eighth row of the table contains the seventy-seventh column of data, which is the name of the seventy-seventh person.
79	79. The seventy-ninth row of the table contains the seventy-eighth column of data, which is the name of the seventy-eighth person.
80	80. The eightieth row of the table contains the seventy-ninth column of data, which is the name of the seventy-ninth person.
81	81. The eighty-first row of the table contains the eightieth column of data, which is the name of the eightieth person.
82	82. The eighty-second row of the table contains the eighty-first column of data, which is the name of the eighty-first person.
83	83. The eighty-third row of the table contains the eighty-second column of data, which is the name of the eighty-second person.
84</	

The various codebases appear to be leaking private keys that have likely been used for testing purposes but at least one of the associated ethereum accounts either has funds or history of transactions on the ethereum mainnet.

privatekey: 0x7ab741b57...faa6ee8 -> address: 0xD9995BAE12FEe327256FFec1e3184d492bD94C31

Address

0xD9995BAE12FEe327256FFec1e3184d492bD94C31

Buy

Exchange

Earn

Crypto Credit

Sponsored:

DeFi Yield Protocol: Your new DeFi gem with anti-manipulation feature. [Join Now!](#)

Overview

Balance:

0.00000021953595358 Ether

Ether Value:

Less Than \$0.01 (@ \$446.39/ETH)

More Info

My Name Tag:

Not Available, [login to update](#)

Transactions

Internal Txns

Analytics

Comments

Latest 2 from a total of 2 transactions

	Txn Hash	Block	Age	From		To	Value
	<a href="#">0x46c287650a6185dd...</a>	<a href="#">7911081</a>	522 days 4 hrs ago	<a href="#">0xd9995bae12fee3272...</a>	OUT	<a href="#">0xc0302c62ced73abdf...</a>	0.0003898020
	<a href="#">0xa4de983bc5092615...</a>	<a href="#">7910986</a>	522 days 4 hrs ago	<a href="#">0xd9995bae12fee3272...</a>	OUT	<a href="#">0xc0302c62ced73abdf...</a>	0.0004911234

[ Download [CSV Export](#) ]

Medium severity because we assume that this is only leaking keys used for testing purposes.

### Examples

- a collection of `ETHERLIME_ACCOUNTS` at least one with tx on mainnet

#### code/packages/devtools/src/constants.ts:L1-L16

```
import {Account} from './types';

export const ETHERLIME_ACCOUNTS: Account[] = [
  {privateKey: '0x7ab741b57e8d94dd7e1a29055646bafde7010f38a900f55bbd7647880faa6ee8'}, //0xD9995BAE12FEe327256FFec1e3184d492bD94C31
  {privateKey: '0x2030b463177db2da82908ef90fa55ddfcef56e8183caf60db464bc398e736e6f'}, //0xd4Fa489Eacc52BA59438993f37Be9fcC20090E33
  {privateKey: '0x62ecd49c4ccb41a70ad46532aed63cf815de15864bc415c87d507afd6a5e8da2'}, //0x760bf27cd45036a6C486802D30B5D90CfFBE31F
  {privateKey: '0xf473040b1a83739a9c7cc1f5719fab0f5bf178f83314d98557c58aae1910e03a'}, //0x56A32fFf5E5A8B40d6A21538579fB8922DF5258
  {privateKey: '0x823d590ed2cb5e8493bb0efc834771c1cde36f9fc49b9fe3620ebd0754ad6ea2'}, //0xfec44e15328B7d1d8885a8226b0858964358F1D
  {privateKey: '0xd6d710943471e4c37ceb787857e7a2b41ca57f9cb4307ee9a9b21436a8e709c3'}, //0xDA8A06F1C910CAB18aD187be1faA2b8606C2ec6
  // Hub uses this (index 6)
  {privateKey: '0x187bb12e927c1652377405f81d93ce948a593f7d66cfba383ee761858b05921a'}, //0x8199de05654e9afa5C081BcE38F140082C9a773
  {privateKey: '0xf41486fdb04505e7966c8720a353ed92ce0d6830f8a5e915fbde735106a06d25'}, //0x28bF45680cA598708E5cDACc1414FCAc04a3F1e
  {privateKey: '0x6ca40ba4cca775643398385022264c0c414da1abd21d08d9e7136796a520a543'}, //0xf0508F89e26Bd6b00f66a9D467678C7ED16a3C05
  {privateKey: '0xfac0bc9325ad342033afe956e83f0bf8f1e863c1c3e956bc75d66961fe4cd186'} //0x87e0ED760fb316eeb94Bd9cf23D1d2BE87aCe3d8
];
```

- default configuration falls back to hardcoded private keys. The fact that the private key is missing may not be reported causing the application to use a well-known leaked private key that may be shared across multiple implementations putting potential rewards at risk.

#### code/packages/server-wallet/src/config.ts:L43-L48

```
serverSignerPrivateKey:
  process.env.SERVER_SIGNER_PRIVATE_KEY ||
  '0x1b427b7ab88e2e10674b5aa92bb63c0ca26aa0b5a858e1d17295db6ad91c049b',
serverPrivateKey:
  process.env.SERVER_PRIVATE_KEY ||
  '0x7ab741b57e8d94dd7e1a29055646bafde7010f38a900f55bbd7647880faa6ee8',
```

- PrivateKeys in documentation

#### code/packages/docs-website/docs/protocol-tutorial/off-chain-funding.md:L23-L25

```
const mySigningKey = '0x7ab741b57e8d94dd7e1a29055646bafde7010f38a900f55bbd7647880faa6ee8';
const hubSigningKey = '0x2030b463177db2da82908ef90fa55ddfcef56e8183caf60db464bc398e736e6f';
const me = new ethers.Wallet(mySigningKey).address;
```

- DevTools

#### code/packages/devtools/src/utlis/network-setup.ts:L3-L4

```
const privateKeyWithEth = '0xf2f48ee19680706196e2e339e5da3491186e0c4c5030670656b0e0164837257d';
```

- xstate-wallet mock

#### code/packages/xstate-wallet/src/chain.ts:L245-L258

```
export class ChainWatcher implements Chain {
  private _adjudicator?: Contract;
  private _assetHolders: Contract[];
  private mySelectedAddress: string | null = window.ethereum?.selectedAddress ?? null;
  private provider: ReturnType<typeof getProvider>;
  private get signer() {
    if (!this.ethereumIsEnabled) throw new Error('Ethereum not enabled');

    if (window.ethereum.mockingInfuraProvider) {
      return new Wallet(
        '0xccb052837ccafb700e34c0e0cc0f3e5fbee8f078f3fe6b4e5950c7c8acaa7bce',
        this.provider
      );
    }
  }
}
```

### Recommendation

Remove hardcoded private-keys and load them from an encrypted Keystore instead. Note that secrets may still be stored in gist history.

## 6.9 sever-wallet - Custom implementation of object merging methods are dangerous Minor

### Description

Prototype pollution vulnerabilities in Javascript (or Typescript) have almost always surfaced due to faulty implementations of object merging methods. Custom implementations of such methods almost always pose a high security risk in production codebases.

One such example in `server-wallet` is presented below. The helper method `pick()`, even if only used once and in such a way that no prototype pollution is achievable, is prone to such problems if it were to be used in any other way.

### Examples

**code/packages/server-wallet/src/utilities/helpers.ts:L1-L7**

```
export function pick<T, K extends keyof T>(obj: T, ...keys: K[]): Pick<T, K> {
  const ret: any = {};
  keys.forEach(key => {
    ret[key] = obj[key];
  });
  return ret;
}
```

**code/packages/server-wallet/src/wallet/store.ts:L635-L649**

```
const channel = Channel.fromJson(
  pick(
    {
      ...constants,
      assetHolderAddress: undefined,
      chainServiceRequests: [],
      channelId: calculateChannelId(constants),
      fundingLedgerChannelId,
      fundingStrategy,
      signingAddress: signingWallet.address,
      vars: [],
    },
    ...CHANNEL_COLUMNS
  )
);
```

### Recommendation

Since `lodash` is already a dependency of the system, it is advised to always use its object merging capabilities for the simple fact that it is a well-maintained open-source package with wide utilization over production codebases.

## 6.10 On-chain constraints for AttestationApp can be increased

### Description

The attestation app is built specifically for The Graph's needs. Therefore, there is a section in the app's code that checks specifically for the existence of *only one* asset in the outcome struct of the app's data being signed by participants.

**additional\_code/GRTAssetHolder/AttestationApp.sol:L53-L64**

```
// {
//   NOTE: This block limits AttestationChannel to outcomes of type "Allocation"
//         for a single asset type. Extensions for multiple assets out of scope.
Outcome.AssetOutcome memory providedAssetOutcomeA = abi.decode(
  providedOutcomeA[0].assetOutcomeBytes,
  (Outcome.AssetOutcome)
);
Outcome.AllocationItem[] memory providedAllocationsA = abi.decode(
  providedAssetOutcomeA.allocationOrGuaranteeBytes,
  (Outcome.AllocationItem[])
);
// }
```

In the Nitro protocol, apps are described as `pure` functions (i.e., `ForceMoveApp` s) that implement checks on the `AppData` structure to validate if a certain state transition is allowed in the specific *game* being played. Solidity created the definition of a `pure` function as being a completely side-effect-free function, not reading or writing to state so it could make use of the underlying EVM opcode `STATICCALL`.



However, under the hood, `STATICCALL` does allow for state reads (i.e., it works with functions marked with the `view` visibility). Meaning that we could deploy the attestation app with a deploy-time `immutable` variable linked to the deployment address of the `GRTAssetHolder` (the only asset we wish to allow usage of in this specific app) and check for that specific address in the `validTransition()` method being called into by Nitro.

This means that, contrary to the current implementation, we would offload the trust from the `server-wallet` component and rely on the on-chain implementation of the app for security. This becomes especially relevant if we look into the current way this is handled in `channel-manager` :

**additional\_code/the-graph/packages/payment-manager/src/channel-manager.ts:L25**

```
export const PAYMENT_CHANNEL_TOKEN_ADDRESS = ethers.constants.AddressZero; // TODO: Is this the asset holder address? Should it
```

**Recommendation**

Pin `GRTAssetHolder` address to the implementation of `AttestationApp` on-chain instead of relying on off-chain enforcement.

**6.11 server-wallet - Improve input validation, fail early on invalid input**

**Description**

Input validation checks should be explicit and well documented as part of the code’s documentation. This is to make sure that an API is robust against erroneous inputs and reduces the potential attack surface for exploitation.

It is good practice to verify the method’s input as early as possible in the call-stack and only perform further actions if the validation succeeds. Methods can be split into a public API that performs initial checks and subsequently calls an internal method that performs less-stringent checks and ultimately the action.

**Examples**

This section mentions a few select candidates for weak or non-existent input validation. It should be noted that this was seen as a pattern throughout the codebase.

- [issue 6.3](#) - describes a check that was missed during the validation of transitions
- [issue 6.2](#) - describes a potential DoS vector where a peer passes in a message with a zero-length participant array.
- the following code throws on `signedState.signatures.length==0`

**code/packages/server-wallet/src/wallet/store.ts:L198-L208**

```
async addMyState(
  channel: Channel,
  signedState: SignedStateWithHash,
  tx: Transaction
): Promise<Channel> {
  if (
    signedState.signatures.length > 1 ||
    signedState.signatures[0].signer !== channel.myAddress
  ) {
    throw new Error('This state not exclusively signed by me');
  }
}
```

- `addSignedState` could fail as early as it detects that `bytecode` is empty, but instead, it only prints an error and continues until `validateTransition` returns false. Given that this is a security-critical function it should bail as early as it detects an error and not rely on any downstream function call to throw or return false. Since the evm validation is only performed under certain conditions (noSkipEvm, noFundingState, noToState.isFinal) and, therefore, the bytecode is only accessed under these conditions this might in the best case just create a misleading log message.

**code/packages/server-wallet/src/wallet/store.ts:L488-L491**

```
if (!this.skipEvmValidation && !bytecode)
  this.logger.error('Missing bytecode', {
    error: new Error(`No byte code for ${supported.appDefinition}`),
  });
```

**code/packages/server-wallet/src/wallet/store.ts:L176-L179**

```
if (!this.skipEvmValidation && !bytecode)
  this.logger.error('Missing bytecode', {
    error: new Error(`No byte code for ${supported.appDefinition}`),
  });
```

**Recommendation**

Fail early on invalid input.

**6.12 server-wallet - Improve handling of application secrets**

**Recommendation**

**General Advise**

- Avoid having to manage/store/directly use secrets within the application at all.
- Avoid storing secrets in plaintext.
- Ideally use an external Keystore, password manager, or HSM (require the external entity to sign data)
- Require the user to type a password on startup or load the secret from a file (e.g. a secret to unlock a keystore)
  - The secret file must be adequately protected (file permissions; separate encrypted volume; for reference: tekulighthouse are using keyfiles with a password file to unlock the keystore read from users directory, enforcing

- restrictive permissions on that file and refusing to startup otherwise)
- Avoid passing secrets via CLI flags or environment variables as they can be read by an attacker or might be leaked by crash-collectors/logging instances

Code

- Javascript is a garbage-collected language that gives no guarantees about low-level memory management. I.e. it is up to the memory management engine to decide if memory is really wiped, reallocated, remapped, or even duplicated on access. Consider minimizing the use of secrets and therefore leave as little traces of secrets in memory as possible.
- Avoid copying secrets from one variable/instance to another reducing potential low-level memory copies of that string.
- Consider implementing an encrypted KeyStore singleton class that controls access to sensitive information

6.13 server-wallet - Provide means to control channel participation

Description

Currently, an indexer does not have any control over which channels they would want to join. For example, receipt-manager joins any channel that is fed into the application:

additional\_code/the-graph/packages/receipt-manager/src/receipt-manager.ts:L66-L87

```
async inputStateChannelMessage(payload: unknown): Promise<unknown> {
  const handleProposedChannel = async (proposedChannels: ChannelResult[]) => {
    /**
     * Initial request to create a channelResult is received. In this case, join
     * the channel.
     */
    if (proposedChannels.length > 0) {
      this.logger.debug('channels proposed detected. joining channels', {
        channelIds: proposedChannels.map((cr) => cr.channelId)
      });

      const joinChannelMessage = await this.wallet.joinChannels(
        proposedChannels.map((cr) => cr.channelId)
      );

      this.logger.info(`channel join succeeded`, {
        channelIds: proposedChannels.map((cr) => cr.channelId)
      });

      return joinChannelMessage;
    }
  };
};
```

Recommendation

Implement means to control channel participation (e.g. blacklist/whitelist filter or more sophisticated means of control): \* acceptable AssetHolder addresses (e.g. GraphToken , specific ERC20, ETHAssetHolder ) \* acceptable participants, appDefinition, challengeDuration, chainId, ...

Appendix 1 - Artifacts

This section contains some of the artifacts generated during our review by automated tools, the test suite, etc. If any issues or recommendations were identified by the output presented here, they have been addressed in the appropriate section above.

A.1.1 MythX

MythX is a security analysis API for Ethereum smart contracts. It performs multiple types of analysis, including fuzzing and symbolic execution, to detect many common vulnerability types. The tool was used for automated vulnerability discovery for all audited contracts and libraries. More details on MythX can be found at mythx.io.

Below is the raw output of the MythX vulnerability scan:

- Nitro Protocol
- GRTAssetHolder & AttestationApp


A.1.2 Surya

Surya is a utility tool for smart contract systems. It provides a number of visual outputs and information about the structure of smart contracts. It also supports querying the function call graph in multiple ways to aid in the manual inspection and control flow analysis of contracts.

Below is a complete list of functions with their visibility and modifiers:

Nitro Protocol

Contracts Description Table

Contract	Type	Bases		
L	Function Name	Visibility	Mutability	Modifiers
Outcome	Library			
ForceMove	Implementation	IForceMove		
L	getChannelStorage	Public !		NO !
L	forceMove	Public !		NO !

Contract	Type	Bases		
L	respond	Public !	🔴	NO !
L	checkpoint	Public !	🔴	NO !
L	conclude	Public !	🔴	NO !
L	_requireChallengerIsParticipant	Internal 🗝️		
L	_isAddressInArray	Internal 🗝️		
L	_validSignatures	Internal 🗝️		
L	_acceptableWhoSignedWhat	Internal 🗝️		
L	_recoverSigner	Internal 🗝️		
L	_requireStateSupportedBy	Internal 🗝️		
L	_requireValidTransitionChain	Internal 🗝️		
L	_requireValidTransition	Internal 🗝️		
L	_bytesEqual	Internal 🗝️		
L	_clearChallenge	Internal 🗝️	🔴	
L	_requireIncreasedTurnNumber	Internal 🗝️		
L	_requireNonDecreasedTurnNumber	Internal 🗝️		
L	_requireSpecificChallenge	Internal 🗝️		
L	_requireOngoingChallenge	Internal 🗝️		
L	_requireChannelNotFinalized	Internal 🗝️		
L	_requireChannelFinalized	Internal 🗝️		
L	_requireChannelOpen	Internal 🗝️		
L	_requireMatchingStorage	Internal 🗝️		
L	_mode	Internal 🗝️		
L	_hashChannelData	Internal 🗝️		
L	_getChannelStorage	Internal 🗝️		
L	_matchesHash	Internal 🗝️		
L	_hashState	Internal 🗝️		
L	_hashOutcome	Internal 🗝️		
L	_getChannelId	Internal 🗝️		
ETHAssetHolder	Implementation	AssetHolder		
L		Public !	🔴	NO !
L	deposit	Public !	💰	NO !
L	_transferAsset	Internal 🗝️	🔴	
TrivialApp	Implementation	ForceMoveApp		
L	validTransition	Public !		NO !
NitroAdjudicator	Implementation	Adjudicator, ForceMove		
L	pushOutcome	Public !	🔴	NO !
L	pushOutcomeAndTransferAll	Public !	🔴	NO !
L	concludePushOutcomeAndTransferAll	Public !	🔴	NO !
L	_transferAllFromAllAssetHolders	Internal 🗝️	🔴	
L	validTransition	Public !		NO !
CountingApp	Implementation	ForceMoveApp		
L	appData	Internal 🗝️		
L	validTransition	Public !		NO !
AssetHolder	Implementation	IAssetHolder		
L	_transferAll	Internal 🗝️	🔴	
L	transferAll	Public !	🔴	NO !
L	transferAllAdjudicatorOnly	External !	🔴	AdjudicatorOnly
L	claimAll	Public !	🔴	NO !
L	_setAssetOutcomeHash	Internal 🗝️	🔴	
L	setAssetOutcomeHash	External !	🔴	AdjudicatorOnly
L	_transferAsset	Internal 🗝️	🔴	

Contract	Type	Bases		
L	_isExternalDestination	Internal		
L	_addressToBytes32	Internal		
L	_bytes32ToAddress	Internal		
<b>Token</b>	Implementation	ERC20		
L		Public		ERC20
<b>ERC20AssetHolder</b>	Implementation	AssetHolder		
L		Public		NO
L	deposit	Public		NO
L	_transferAsset	Internal		
<b>SingleAssetPayments</b>	Implementation	ForceMoveApp		
L	validTransition	Public		NO
<b>IForceMove</b>	Implementation			
L	forceMove	Public		NO
L	respond	Public		NO
L	checkpoint	Public		NO
L	conclude	Public		NO
<b>ForceMoveApp</b>	Interface			
L	validTransition	External		NO
<b>Adjudicator</b>	Interface			
L	pushOutcome	External		NO
<b>IAssetHolder</b>	Interface			
L	transferAll	External		NO
L	claimAll	External		NO

### AttestationApp

Contracts Description Table

Contract	Type	Bases		
L	<b>Function Name</b>	<b>Visibility</b>	<b>Mutability</b>	<b>Modifiers</b>
<b>AttestationApp</b>	Implementation	ForceMoveApp, AttestationStateMachine		
L		Public		NO
L	deduceActionFromState	Internal		
L	validTransition	External		NO
<b>AttestationUtils</b>	Implementation			
L	recoverAttestationSigner	Public		NO
<b>AttestationStateMachine</b>	Implementation	AttestationUtils		
L	computeNextState	Public		NO
L	attachQuery	Public		NO
L	attachAttestation	Public		NO
L	resetQueryState	Public		NO
L	whoSignedAttestation	Public		NO

### GRTAssetHolder

Contracts Description Table



Contract	Type	Bases		
L	<b>Function Name</b>	<b>Visibility</b>	<b>Mutability</b>	<b>Modifiers</b>
<b>IERC20</b>	Interface			
L	totalSupply	External		NO
L	balanceOf	External		NO



Contract	Type	Bases		
L	transfer	External !		NO !
L	allowance	External !		NO !
L	approve	External !		NO !
L	transferFrom	External !		NO !
IAssetHolder	Interface			
L	transferAll	External !		NO !
L	claimAll	External !		NO !
IGraphToken	Interface	IERC20		
L	burn	External !		NO !
L	mint	External !		NO !
AttestationApp	Implementation	ForceMoveApp, AttestationStateMachine		
L		Public !		NO !
L	deduceActionFromState	Internal		
L	validTransition	External !		NO !
SafeMath	Library			
L	add	Internal		
L	sub	Internal		
L	sub	Internal		
L	mul	Internal		
L	div	Internal		
L	div	Internal		
L	mod	Internal		
L	mod	Internal		
AssetHolder	Implementation	IAssetHolder		
L	_transferAll	Internal		
L	transferAll	Public !		NO !
L	transferAllAdjudicatorOnly	External !		AdjudicatorOnly
L	claimAll	Public !		NO !
L	_setAssetOutcomeHash	Internal		
L	setAssetOutcomeHash	External !		AdjudicatorOnly
L	_transferAsset	Internal		
L	_isExternalDestination	Internal		
L	_addressToBytes32	Internal		
L	_bytes32ToAddress	Internal		
AttestationStateMachine	Implementation	AttestationUtils		
L	computeNextState	Public !		NO !
L	attachQuery	Public !		NO !
L	attachAttestation	Public !		NO !
L	resetQueryState	Public !		NO !
L	whoSignedAttestation	Public !		NO !
IController	Interface			
L	setContractProxy	External !		NO !
L	updateController	External !		NO !
L	getContractProxy	External !		NO !
L	getGovernor	External !		NO !
L	paused	External !		NO !
L	recoveryPaused	External !		NO !

Contract	Type	Bases		
Outcome	Library			
GRTAssetHolder	Implementation	ERC20AssetHolder		
L		Public !		ERC20AssetHolder
L	staking	Public !		NO !
L	approveAll	External !		NO !
L	_transferAsset	Internal 🗝️		
IStaking	Interface			
L	setThawingPeriod	External !		NO !
L	setCurationPercentage	External !		NO !
L	setProtocolPercentage	External !		NO !
L	setChannelDisputeEpochs	External !		NO !
L	setMaxAllocationEpochs	External !		NO !
L	setDelegationCapacity	External !		NO !
L	setDelegationParameters	External !		NO !
L	setDelegationParametersCool down	External !		NO !
L	setDelegationUnbondingPeriod	External !		NO !
L	setSlasher	External !		NO !
L	setOperator	External !		NO !
L	stake	External !		NO !
L	stakeTo	External !		NO !
L	unstake	External !		NO !
L	slash	External !		NO !
L	withdraw	External !		NO !
L	delegate	External !		NO !
L	undelegate	External !		NO !
L	withdrawDelegated	External !		NO !
L	allocate	External !		NO !
L	allocateFrom	External !		NO !
L	settle	External !		NO !
L	collect	External !		NO !
L	claim	External !		NO !
L	hasStake	External !		NO !
L	getIndexerStakedTokens	External !		NO !
L	getIndexerCapacity	External !		NO !
L	getAllocation	External !		NO !
L	getAllocationState	External !		NO !
L	isChannel	External !		NO !
L	getSubgraphAllocatedTokens	External !		NO !
L	getDelegation	External !		NO !
AttestationUtils	Implementation			
L	recoverAttestationSigner	Public !		NO !
ECDSA	Library			
L	recover	Internal 🗝️		
L	toEthSignedMessageHash	Internal 🗝️		
ForceMoveApp	Interface			
L	validTransition	External !		NO !
ERC20AssetHolder	Implementation	AssetHolder		
L		Public !		NO !
L	deposit	Public !		NO !

Legend

Symbol	Meaning
	Function can modify state
	Function is payable

## Appendix 2 - Files in Scope

### A.2.1 Nitro Protocol

This audit covered the following files:

File Name	SHA-1 Hash
code/packages/nitro-protocol/contracts/Outcome.sol	e097dbfe98cd2f3b4976329087710db707375af9
code/packages/nitro-protocol/contracts/ForceMove.sol	c97f81a6c2dac7a1a9def82777c99ef04b04f4c0
code/packages/nitro-protocol/contracts/ETHAssetHolder.sol	9d079b361938a6a5fbda02ed7a985773670ea371
code/packages/nitro-protocol/contracts/TrivialApp.sol	556093c21dd53b5d35832e99a02c4758807788c4
code/packages/nitro-protocol/contracts/NitroAdjudicator.sol	0d9a40952684e14de8a25b3907c7185cbe10cb59
code/packages/nitro-protocol/contracts/CountingApp.sol	9e70efea8f428738e70032501d1c69e19fb051f5
code/packages/nitro-protocol/contracts/AssetHolder.sol	4fb4884d2ef3e7a68fe488353e995632f2383da5
code/packages/nitro-protocol/contracts/Token.sol	bb2889128144aaece63034a78b6098f1e433be08
code/packages/nitro-protocol/contracts/ERC20AssetHolder.sol	bc282aa706b337145e56afb41816e5311931f286
code/packages/nitro-protocol/contracts/examples/SingleAssetPayments.sol	c46cb8b81014b7464248ec0fe88e18f9719e33c1
code/packages/nitro-protocol/contracts/interfaces/IForceMove.sol	4d43f94230c0325da1a6b0d7325e8af3c4e16132
code/packages/nitro-protocol/contracts/interfaces/ForceMoveApp.sol	ba5ea76e93936b0b15eadaafb46146149db4fda7
code/packages/nitro-protocol/contracts/interfaces/Adjudicator.sol	788358d4aa6156a06f042652cf8910551dec8221
code/packages/nitro-protocol/contracts/interfaces/IAssetHolder.sol	02bd319fc0fd7f0b9ecd2d78e8f83f92fd8a04bb

### A.2.2 Attestation App

File Name	SHA-1 Hash
additional_code/the-graph/packages/statechannels-contracts/contracts/AttestationUtils.sol	c8eaa03b0afc0d6dcf1a6c1732ed4732aaabf284
additional_code/the-graph/packages/statechannels-contracts/contracts/AttestationApp.sol	03c6b9daec947fe93079498833be3a5d83a6600f
additional_code/the-graph/packages/statechannels-contracts/contracts/AttestationStateMachine.sol	c369c81a04b94fbe865d3f7fc46ee4816feb4848

### A.2.3 GRTAssetHolder

File Name	SHA-1 Hash
additional_code/GRTAssetHolder/IERC20.sol	c1e074a2655230e26e90ba95cbcd402701ee1854
additional_code/GRTAssetHolder/IAssetHolder.sol	02bd319fc0fd7f0b9ecd2d78e8f83f92fd8a04bb
additional_code/GRTAssetHolder/IGraphToken.sol	6526f2e0cee2e441e346b2aa44cea14cac9458a4
additional_code/GRTAssetHolder/AttestationApp.sol	f46748b43155489b9fabe0c84b7a419100a27464
additional_code/GRTAssetHolder/SafeMath.sol	eda72d381d2fa58c150ea356072cc98f026be5e8
additional_code/GRTAssetHolder/AssetHolder.sol	ee0c8ba84ead90a4f7a14b05da840c9cdea93954
additional_code/GRTAssetHolder/AttestationStateMachine.sol	a6e3775f704ff4a9b9f284e513251452bf6e0329
additional_code/GRTAssetHolder/IController.sol	bf0cc090f4d6cb27d5804b185eefed5aced10b20
additional_code/GRTAssetHolder/Outcome.sol	e097dbfe98cd2f3b4976329087710db707375af9
additional_code/GRTAssetHolder/GRTAssetHolder.sol	2ba176c68afb2f9f33e0f0bab1f467f2dfcbe165
additional_code/GRTAssetHolder/IStaking.sol	52cdaa19d1180593855bca491319a0f3cd7cead5
additional_code/GRTAssetHolder/AttestationUtils.sol	2a5bbacca022c4ea475c5b09128d92b787b52f6f
additional_code/GRTAssetHolder/ECDSA.sol	63676b85187659187ed6b099f9e2a6b6e034fd83
additional_code/GRTAssetHolder/ForceMoveApp.sol	ba5ea76e93936b0b15eadaafb46146149db4fda7
additional_code/GRTAssetHolder/ERC20AssetHolder.sol	96341ff2edf99c8205873c872b6ddfe60a1356d5

## Appendix 3 - Disclosure

ConsenSys Diligence (“CD”) typically receives compensation from one or more clients (the “Clients”) for performing the analysis contained in these reports (the “Reports”). The Reports may be distributed through other means, including via ConsenSys publications and other distributions.

The Reports are not an endorsement or indictment of any particular project or team, and the Reports do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. CD owes no duty to any Third-Party by virtue of publishing these Reports.

**PURPOSE OF REPORTS** The Reports and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty.

CD makes the Reports available to parties other than the Clients (i.e., “third parties”) – on its website. CD hopes that by making these analyses publicly available, it can help the blockchain ecosystem develop technical best practices in this rapidly evolving area of innovation.

**LINKS TO OTHER WEB SITES FROM THIS WEB SITE** You may, through hypertext or other computer links, gain access to web sites operated by persons other than ConsenSys and CD. Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites’ owners. You agree that ConsenSys and CD are not responsible for the content or operation of such Web sites, and that ConsenSys and CD shall have no liability to you or any other person or entity for the use of third party Web sites. Except as described below, a hyperlink from this web Site to another web site does not imply or mean that ConsenSys and CD endorses the content on that Web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the Reports. ConsenSys and CD assumes no responsibility for the use of third party software on the Web Site and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

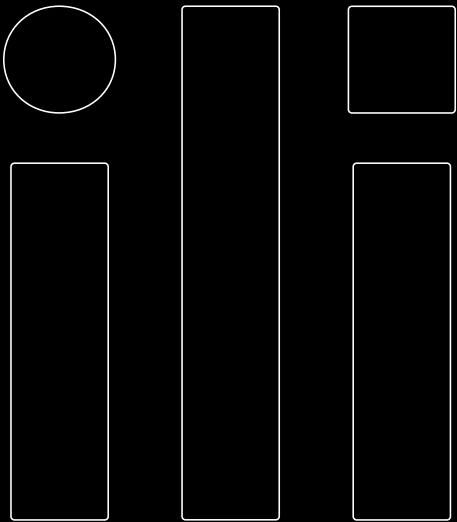
**TIMELINESS OF CONTENT** The content contained in the Reports is current as of the date appearing on the Report and is subject to change without notice. Unless indicated otherwise, by ConsenSys and CD.



# Request a Security Review Today

Get in touch with our team to request a quote for a smart contract audit or a 1-day security review.

CONTACT US



- AUDITS
- BLOG
- TOOLS
- RESEARCH
- ABOUT
- CONTACT
- CAREERS

## Subscribe to Our Newsletter

Stay up-to-date on our latest offerings, tools, and the world of blockchain security.

e-mail address →