Timeboost Auction Contracts

Security Assessment

September 25, 2024

Prepared for:
Offchain Labs

Prepared by: **Gustavo Grieco, Priyanka Bose, and Michael Colburn**

About Trail of Bits

Founded in 2012 and headquartered in New York, Trail of Bits provides technical security assessment and advisory services to some of the world's most targeted organizations. We combine high-end security research with a real-world attacker mentality to reduce risk and fortify code. With 100+ employees around the globe, we've helped secure critical software elements that support billions of end users, including Kubernetes and the Linux kernel.

We maintain an exhaustive list of publications at https://github.com/trailofbits/publications, with links to papers, presentations, public audit reports, and podcast appearances.

In recent years, Trail of Bits consultants have showcased cutting-edge research through presentations at CanSecWest, HCSS, Devcon, Empire Hacking, GrrCon, LangSec, NorthSec, the O'Reilly Security Conference, PyCon, REcon, Security BSides, and SummerCon.

We specialize in software testing and code review projects, supporting client organizations in the technology, defense, and finance industries, as well as government entities. Notable clients include HashiCorp, Google, Microsoft, Western Digital, and Zoom.

Trail of Bits also operates a center of excellence with regard to blockchain security. Notable projects include audits of Algorand, Bitcoin SV, Chainlink, Compound, Ethereum 2.0, MakerDAO, Matic, Uniswap, Web3, and Zcash.

To keep up to date with our latest news and announcements, please follow @trailofbits on Twitter and explore our public repositories at https://github.com/trailofbits. To engage us directly, visit our "Contact" page at https://www.trailofbits.com/contact, or email us at info@trailofbits.com.

Trail of Bits, Inc.
497 Carroll St., Space 71, Seventh Floor
Brooklyn, NY 11215
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

Test Coverage Disclaimer

All activities undertaken by Trail of Bits in association with this project were performed in accordance with a statement of work and agreed upon project plan.

Security assessment projects are time-boxed and often reliant on information that may be provided by a client, its affiliates, or its partners. As a result, the findings documented in this report should not be considered a comprehensive list of security issues, flaws, or defects in the target system or codebase.

Trail of Bits uses automated testing techniques to rapidly test the controls and security properties of software. These techniques augment our manual security review work, but each has its limitations: for example, a tool may not generate a random edge case that violates a property or may not fully complete its analysis during the allotted time. Their use is also limited by the time and resource constraints of a project.

Table of Contents

Project Summary

Contact Information

The following project manager was associated with this project:

Mary O'Brien, Project Manager
mary.obrien@trailofbits.com

The following engineering director was associated with this project:

Josselin Feist, Engineering Director, Blockchain
josselin.feist@trailofbits.com

The following consultants were associated with this project:

Gustavo Grieco, Consultant
gustavo.grieco@trailofbits.com

Priyanka Bose, Consultant
priyanka.bose@trailofbits.com

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Project Timeline

The significant events and milestones of the project are listed below.

Date	Event
August 28, 2024	Delivery of report draft
August 28, 2024	Report readout meeting
September 25, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

Offchain Labs engaged Trail of Bits to review the security of their Timeboost auction contracts. These contracts implement the on-chain components of a sealed-bid, second-price auction for the rights to assign an express lane controller whose transactions are processed faster by the sequencer for a set period of time.

A team of three consultants conducted the review from August 19 to August 27, 2024, for a total of three engineer-weeks of effort. With full access to source code and documentation, we performed static and dynamic testing of the codebase using automated and manual processes.

Observations and Impact

The codebase includes extensive NatSpec documentation, and the project's specification provides a comprehensive description of the system's functionality, the roles involved, and the auction mechanism, though we did note several minor discrepancies between the written specification and the implementation.

During the review, we evaluated the access controls, the accuracy of updating the round timing info, the possibility of malicious bidders evading bidding amount, the potential for introducing bias during the selection of the winning bid, and any possibility that could halt the auction's progression. Our review resulted in three issues ranging in severity from medium to informational. The medium-severity issue highlights an unhandled edge case that could prevent an auction round from being finalized (TOB-ELA-1), and the remaining informational issues are related to discrepancies with the specification (TOB-ELA-2, TOB-ELA-3).

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, Trail of Bits recommends that Offchain Labs take the following steps:

- **Remediate the TOB-ELA-002 issue disclosed in this report.** This finding should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Add a stateful fuzzing test suite.** Trail of Bits considers stateful fuzzing to be a baseline requirement for DeFi protocols and applications and recommends fuzzing the system's user flows and arithmetic calculations.

- **Align implementation with the specification.** A number of discrepancies exist between the specification and the actual implementation. These two should be aligned to eliminate any gaps between them.

Finding Severities and Categories

The following tables provide the number of findings by severity and category.

EXPOSURE ANALYSIS

Severity	*Count*
High	0
Medium	1
Low	0
Informational	2
Undetermined	0

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	3

Project Goals

The engagement was scoped to provide a security assessment of Offchain Labs' Timeboost design. Specifically, we sought to answer the following non-exhaustive list of questions:

- Are proper access controls used in the contracts that users interact with?
- Could an unauthorized user disrupt the auction?
- Could one user withdraw another user's tokens?
- Are any of the components vulnerable to price manipulation?
- Could a user's funds become frozen or stuck?
- Are events used appropriately?
- Is the actual implementation in line with the provided specification?
- Do auction rounds always start and end at the expected time?
- Can a round be resolved multiple times?
- Could the use of signed arithmetic in the system's timekeeping allow the system to be put in an unexpected state?

Project Targets

The engagement involved a review and testing of the following targets.

express-lane-auction (PR 214)

Repository	https://github.com/OffchainLabs/nitro-contracts
Version	`9dc19d21c0ba0df89529cc0085915fa9565ecafd (initial)`
	`84ade5042533fb35c3f30ae7bfec85580eba461d (fixes)`
Type	Solidity
Platform	EVM

timeboost-design

Repository	https://github.com/OffchainLabs/timeboost-design
Version	`02846291b669d559c4bbf13e2ea0d499864d043b`
Type	Markdown

Project Coverage

This section provides an overview of the analysis coverage of the review, as determined by our high-level engagement goals. Our approaches included the following:

- **Timeboost auction smart contracts**. The on-chain part of the Timeboost auction protocol is implemented using several smart contracts, which will allow any user to deposit funds, withdraw their funds, or temporarily select the express lane controller if they win the auction. Privileged users can also perform administrative operations, such as change the protocol parameters or resolve auctions, by submitting bids received off-chain.

  We reviewed these contracts for common Solidity issues as well as any issues related to access control violations, accuracy of updating round timing information, and any deviations of the implementation from the standard. We also explored the potential for malicious bidders to manipulate bidding amounts during the auction, the possibility of fund loss or theft from the contracts, and whether users from the protocol could block bids, or somehow delay or disrupt the auction.

Coverage Limitations

Because of the time-boxed nature of testing work, it is common to encounter coverage limitations. The following list outlines the coverage limitations of the engagement and indicates system elements that may warrant further review:

- While we reviewed the informal specification of the Timeboost auction mechanism for inconsistencies, we did not review some aspects of this mechanism, such as the high-level impact to the rollups or the economic soundness.

- We did not review the off-chain components of the Timeboost protocol.

Codebase Maturity Evaluation

Trail of Bits uses a traffic-light protocol to provide each client with a clear understanding of the areas in which its codebase is mature, immature, or underdeveloped. Deficiencies identified here often stem from root causes within the software development life cycle that should be addressed through standardization measures (e.g., the use of common libraries, functions, or frameworks) or training and awareness programs.

Category	Summary	Result
Arithmetic	Overall, the system does not rely on complex arithmetic, and arithmetic use seems safe. Additionally, the protocol is implemented using Solidity 0.8.9, which has overflow protection by default for arithmetic operations, and the contracts do not include any `unchecked` blocks. The system's operations depend on certain privileged actors manually executing essential tasks (via off-chain executions). However, certain features, such as the round offset that relies on signed numbers, should be properly documented in the code and the standard to ensure that their properties are not violated by other contracts.	**Satisfactory**
Auditing	All system operations emit events that include sufficient context for effective off-chain monitoring. However, there is a general lack of documentation about the implications of each event and what kind of events could indicate unexpected behavior.	**Satisfactory**
Authentication / Access Controls	The system heavily relies on access controls to protect arbitrary users from calling privileged functions. The core contract divides these responsibilities across seven different roles, in addition to the default admin role. Most functions are restricted to either a special privileged user or the DAO governance to call.	**Satisfactory**
Complexity Management	The functions and contracts are organized and scoped appropriately and contain inline documentation that explains their workings.	**Satisfactory**
Decentralization	As the client has indicated, the use of off-chain components and the privileged users lowers the decentralization level of the protocol, but this is well	**Moderate**

	documented. For example, the auctioneer is trusted to not censor bids and resolve auctions according to the specification.	
Documentation	The project has good high-level documentation, a specification indicating the behavior of the system, and good use of NatSpec and inline comments. However, the system's invariants are not specified.	**Satisfactory**
Testing and Verification	The codebase contains a number of unit and integration tests, as well a small amount of fuzz testing properties. However, the tests are insufficient to catch medium-severity issues, which indicates that the test suite should be improved. The codebase would benefit from expanded fuzz testing, as described in appendix D.	**Moderate**
Transaction Ordering	While on-chain auction implementations are usually susceptible to transaction ordering, this does not apply directly since bids are resolved using a trusted user. This user only performs the resolution of the bid on-chain and is supposed to verify the state of the blockchain before submitting transactions to the network to ensure that user balances are sufficient.	**Satisfactory**

Summary of Findings

The table below summarizes the findings of the review, including type and severity details.

ID	Title	Type	Severity
1	Bids cannot be resolved if reservePrice is zero	Data Validation	**Medium**
2	Discrepancies in the structure and use of signatures between the specification and implementation	Data Validation	**Informational**
3	No way to burn auction proceeds	Data Validation	**Informational**

Detailed Findings

1. Bids cannot be resolved if reservePrice is zero

Severity: **Medium**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-ELA-1
Target: `src/express-lane-auction/Balance.sol`	

Description

Auction resolution will revert if the reserve price is zero, blocking the express lane controller address from being assigned for that round.

Express lane controller selection is performed through a second-bid auction, where a privileged user will call the `resolveSingleBidAuction` or `resolveMultiBidAuction` function to resolve an auction using valid signatures from users willing to pay a particular amount.

```
    /// @inheritdoc IExpressLaneAuction
    function resolveSingleBidAuction(Bid calldata firstPriceBid)
        External
 392    /// @inheritdoc IExpressLaneAuction
 393    function resolveSingleBidAuction(Bid calldata firstPriceBid)
 394        external
 395        onlyRole(AUCTIONEER_ROLE)
 ...
 422    /// @inheritdoc IExpressLaneAuction
 423    function resolveMultiBidAuction(Bid calldata firstPriceBid, Bid calldata
secondPriceBid)
 424        external
 425        onlyRole(AUCTIONEER_ROLE)
```

Figure 1.1: The headers of the functions used by the auctioneer to resolve auctions (src/express-lane-auction/ExpressLaneAuction.sol#L392-L425)

When an auction is resolved, the price paid for the bid is taken from the balance of the bidder:

```
 308    function resolveAuction(
 309        bool isMultiBid,
 310        Bid calldata firstPriceBid,
 311        address firstPriceBidder,
 312        uint256 priceToPay,
```

```
313         uint64 biddingInRound,
314         uint64 roundStart,
315         uint64 roundEnd
316     ) internal {
317         // store that a round has been resolved
318         uint64 biddingForRound = biddingInRound + 1;
319         latestResolvedRounds.setResolvedRound(biddingForRound,
firstPriceBid.expressLaneController);
320
321         // first price bidder pays the beneficiary
322         _balanceOf[firstPriceBidder].reduce(priceToPay, biddingInRound);
```

Figure 1.2: A snippet of the `resolveAuction` function showing the price being deducted from the winning bidder's balance (src/express-lane-auction/ExpressLaneAuction.sol#L308-L322)

However, if the price to pay is zero, the balance reduction will revert.

```
77     function reduce(
78         Balance storage bal,
79         uint256 amount,
80         uint64 round
81     ) internal {
82         if (amount == 0) {
83             revert ZeroAmount();
84         }
```

Figure 1.3: A snippet of the `reduce` library function that expects a non-zero deduction (src/express-lane-auction/Balance.sol#L77-L84)

Exploit Scenario

The reserve price is set to zero and Alice is the sole bidder for a round. Since this is a second-price auction, she should pay the reserve price when the auction for this round is resolved. The auctioneer calls the `resolveSingleBidAuction` function and passes in Alice's bid. The call reverts when it attempts to deduct zero tokens from Alice's balance. As a result, no express lane controller can be assigned for that round.

Recommendations

Short term, consider allowing the `reduce` function to decrement zero.

Long term, use fuzz testing to detect unexpected reverts caused by calling different functions of the smart contracts.

Fix Status

Resolved in PR 243. The updated code allows for bids of zero tokens as long as the bidder has a non-zero balance.

2. Discrepancies in the structure and use of signatures between the specification and implementation

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ELA-2
Target: `src/express-lane-auction/ExpressLaneAuction.sol`	

Description

Bids that are submitted to an express lane auction round include signatures that must be validated on-chain. While reviewing the specification and implementation, we identified several minor discrepancies between the documentation and the contract's use of signatures. These differences do not compromise the integrity of the signatures used by the system, but may result in integration issues for third parties attempting to bid.

The written specification defines a bid as containing "(`chainId`, `auctionContractAddress`, `roundNumber`, `bid`, `expressLaneControllerAddress`, `signature`)" and defines the signature as being "signature by the bidder's private key on the tuple (`chainId`, `auctionContractAddress`, `roundNumber`, `bid`, `expressLaneControllerAddress`)". A code comment in the `IExpressLaneAuction` contract further specifies an EIP-191 `personal_sign` message format for the signature using all of the fields mentioned above (though with the order of the last two fields swapped).

However, the contract actually expects an EIP-712 signature with the `chainId` and `auctionContractAddress` in the domain separator, and the actual bid being constructed of the remaining three fields (again, with the order of the last two fields swapped relative to the specification), as shown in figure 2.1.

```
356     function getBidHash(
357         uint64 round,
358         address expressLaneController,
359         uint256 amount
360     ) public view returns (bytes32) {
361         return
362             _hashTypedDataV4(
363                 keccak256(abi.encode(BID_DOMAIN, round, expressLaneController,
amount))
364             );
365     }
```

Figure 2.1: The `getBidHash` function definition showing the order of the bid parameters
`src/express-lane-auction/ExpressLaneAuction.sol#L356-L365`

This also has trickle-down effects on how ties will be broken by the contract. The specification states that if the top two bidders submit bids with identical amounts, then "the tie is broken by hashing the bidder address concatenated with the respective byte-string representation of the bid using the Keccak256 hashing scheme." However, since the contract considers a bid to be constructed differently than the specification, the outcome of the tie breaking process will differ. Additionally, rather than concatenating the bidder's address with the full byte-string representation of the bid, the contract uses a hash of the bid (figure 2.2).

```
460     // when bids have the same amount we break ties based on the bid hash
461     // although we include equality in the check we know this isnt possible due
462     // to the check above that ensures the first price bidder and second price
bidder are different
463     if (
464         firstPriceBid.amount == secondPriceBid.amount &&
465         uint256(keccak256(abi.encodePacked(firstPriceBidder, firstBidHash))) <
466         uint256(keccak256(abi.encodePacked(secondPriceBidder, secondBidHash)))
467     ) {
468         revert TieBidsWrongOrder();
469     }
```

Figure 2.2: A portion of the `resolveMultiBidAuction` function body showing the tie breaking logic `src/express-lane-auction/ExpressLaneAuction.sol#L460-L469`

Recommendations

Short term, update the specification to match the implemented behavior.

Long term, review the Arbitrum-related specifications across components and resolve any discrepancies.

Fix Status

Unresolved.

3. No way to burn auction proceeds

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-ELA-3
Target: `src/express-lane-auction/ExpressLaneAuction.sol`	

Description

According to the Timeboost specification, after deducting the funds from the highest bidder, the contract should "transfer those funds to a `beneficiary` account designated by governance, or burn them if governance specifies that the proceeds are to be burned." However, the contract does not include any mechanism to signal that the proceeds should be burned either at the time of auction resolution or when the funds are later transferred out of the contract. At the time of auction resolution, the funds are escrowed in an internal beneficiary account (figure 3.1) that can be emptied at a later date.

```
308    function resolveAuction(
309        bool isMultiBid,
310        Bid calldata firstPriceBid,
311        address firstPriceBidder,
312        uint256 priceToPay,
313        uint64 biddingInRound,
314        uint64 roundStart,
315        uint64 roundEnd
316    ) internal {
317        // store that a round has been resolved
318        uint64 biddingForRound = biddingInRound + 1;
319        latestResolvedRounds.setResolvedRound(biddingForRound,
firstPriceBid.expressLaneController);
320
321        // first price bidder pays the beneficiary
322        _balanceOf[firstPriceBidder].reduce(priceToPay, biddingInRound);
323        beneficiaryBalance += priceToPay;
```

Figure 3.1: A snippet of the `resolveAuction` function definition showing the proceeds being escrowed (`src/express-lane-auction/ExpressLaneAuction.sol#L308-L323`)

The only way to access the beneficiary account is via the `flushBeneficiaryBalance` function (figure 3.2). Any user can call this function, which will transfer the entire beneficiary balance to the currently designated beneficiary address.

```
291    function flushBeneficiaryBalance() public {
292        uint256 bal = beneficiaryBalance;
293        if (bal == 0) {
```

```
294             revert ZeroAmount();
295         }
296         beneficiaryBalance = 0;
297         biddingToken.safeTransfer(beneficiary, bal);
298     }
```

Figure 3.2: The `flushBeneficiaryBalance` function (`src/express-lane-auction/ExpressLaneAuction.sol#L291-L298`)

Recommendations

Short term, add functionality to explicitly signal that tokens should be burned, or update the specification to reflect that the currently implemented behavior is correct.

Long term, review the Arbitrum-related specifications across components and resolve any discrepancies.

Fix Status

Resolved in PR 242. The Offchain Labs team has added a new `Burner` contract that can be deployed if needed and set as the beneficiary in the event that governance decides to configure the system to burn auction proceeds.

A. Vulnerability Categories

The following tables describe the vulnerability categories, severity levels, and difficulty levels used in this document.

Vulnerability Categories	
Category	**Description**
Access Controls	Insufficient authorization or assessment of rights
Auditing and Logging	Insufficient auditing of actions or logging of problems
Authentication	Improper identification of users
Configuration	Misconfigured servers, devices, or software components
Cryptography	A breach of system confidentiality or integrity
Data Exposure	Exposure of sensitive information
Data Validation	Improper reliance on the structure or values of data
Denial of Service	A system failure with an availability impact
Error Reporting	Insecure or insufficient reporting of error conditions
Patching	Use of an outdated software package or library
Session Management	Improper identification of authenticated users
Testing	Insufficient test methodology or test coverage
Timing	Race conditions or other order-of-operations flaws
Undefined Behavior	Undefined behavior triggered within the system

Severity Levels	
Severity	**Description**
Informational	The issue does not pose an immediate risk but is relevant to security best practices.
Undetermined	The extent of the risk was not determined during this engagement.
Low	The risk is small or is not one the client has indicated is important.
Medium	User information is at risk; exploitation could pose reputational, legal, or moderate financial risks.
High	The flaw could affect numerous users and have serious reputational, legal, or financial implications.

Difficulty Levels	
Difficulty	**Description**
Undetermined	The difficulty of exploitation was not determined during this engagement.
Low	The flaw is well known; public tools for its exploitation exist or can be scripted.
Medium	An attacker must write an exploit or will need in-depth knowledge of the system.
High	An attacker must have privileged access to the system, may need to know complex technical details, or must discover other weaknesses to exploit this issue.

B. Code Maturity Categories

The following tables describe the code maturity categories and rating criteria used in this document.

Code Maturity Categories	
Category	**Description**
Arithmetic	The proper use of mathematical operations and semantics
Auditing	The use of event auditing and logging to support monitoring
Authentication / Access Controls	The use of robust access controls to handle identification and authorization and to ensure safe interactions with the system
Complexity Management	The presence of clear structures designed to manage system complexity, including the separation of system logic into clearly defined functions
Cryptography and Key Management	The safe use of cryptographic primitives and functions, along with the presence of robust mechanisms for key generation and distribution
Decentralization	The presence of a decentralized governance structure for mitigating insider threats and managing risks posed by contract upgrades
Documentation	The presence of comprehensive and readable codebase documentation
Low-Level Manipulation	The justified use of inline assembly and low-level calls
Testing and Verification	The presence of robust testing procedures (e.g., unit tests, integration tests, and verification methods) and sufficient test coverage
Transaction Ordering	The system's resistance to transaction-ordering attacks

Rating Criteria	
Rating	**Description**
Strong	No issues were found, and the system exceeds industry standards.
Satisfactory	Minor issues were found, but the system is compliant with best practices.
Moderate	Some issues that may affect system safety were found.
Weak	Many issues that affect system safety were found.
Missing	A required component is missing, significantly affecting system safety.
Not Applicable	The category is not applicable to this review.
Not Considered	The category was not considered in this review.
Further Investigation Required	Further investigation is required to reach a meaningful conclusion.

C. Code Quality Recommendations

The following recommendations are not associated with specific vulnerabilities. However, they enhance code readability and may prevent the introduction of vulnerabilities in the future.

Smart Contracts

- Update the out-of-date comment at the top of the `Balance` contract that suggests that a pending withdrawal will block future deposits, as the current behavior will allow deposits at any time and remove any pending withdrawals instead.
- Clarify the expected bounds on the `roundDurationSeconds` field of a `RoundTimingInfo` struct. The code comment for the field suggests that the valid range is strictly less than 86400, but the validation in the `setRoundTimingInfoInternal` function accepts a value less than or equal to 86400.

Unit Tests

- The `testSetBeneficiary` test has an unused variable named `newBeneficiary2` that should be removed.

```
1741    function testSetBeneficiary() public {
1742        ResolveSetup memory rs = deployDepositAndBids();
1743        vm.stopPrank();
1744
1745        address newBeneficiary = vm.addr(9090);
1746        address newBeneficiary2 = vm.addr(9091);
1747
1748        bytes memory revertString = abi.encodePacked(
1749            "AccessControl: account ",
1750            Strings.toHexString(uint160(address(this)), 20),
1751            " is missing role ",
1752            Strings.toHexString(uint256(rs.auction.BENEFICIARY_SETTER_ROLE()),
32)
1753        );
1754        vm.expectRevert(revertString);
1755        rs.auction.setBeneficiary(newBeneficiary);
1756
1757        vm.prank(beneficiarySetter);
1758        vm.expectEmit(true, true, true, true);
1759        emit SetBeneficiary(beneficiary, newBeneficiary);
1760        rs.auction.setBeneficiary(newBeneficiary);
1761        assertEq(rs.auction.beneficiary(), newBeneficiary, "new beneficiary");
1762    }
```

Figure C.1: The `testSetBeneficiary` unit test with an unused variable (`test/foundry/ExpressLaneAuction.t.sol#L1741-L1762`)

D. Smart Contract Fuzzing Recommendations

The following recommendations are related to the use of fuzz testing on the Express Lane Auction smart contracts.

- Make sure you cover both stateless and stateful invariants. The current code favors stateless tests, even if the code is used with different states. For instance, this is how the `Balance` contract tests that balances are reduced to zero after the withdrawal round:

```
51    function testBalanceAtRound(
52        uint256 initialBalance,
53        uint64 initialRound,
54        uint64 withdrawalRound
55    ) public {
56        Balance memory bal = Balance(initialBalance, initialRound);
57        BalanceImp b = new BalanceImp(bal);
58        if (withdrawalRound >= initialRound) {
59            assertEq(b.balanceAtRound(withdrawalRound), 0);
60        } else {
61            assertEq(b.balanceAtRound(withdrawalRound), initialBalance);
62        }
63    }
```

Figure D.1: The `testBalanceAtRound` fuzz test for checking how the contract behaves depending on the withdrawal round (`test/foundry/ExpressLaneBalance.t.sol#L51-L63`)

  Though the `Balance` contract can change its state over different transactions, this test is really stateless since the state is simulated using different parameter values instead of actually executing different operations (deployment, `startWithdrawal`, and `endWithdrawal`). This pattern should be avoided; instead, use invariant testing that initializes the `Balance` contract once and checks different types of invariants when different operations are executed (e.g., after a successful withdrawal, the balance of the user should be zero). Additional guidance on introducing stateful fuzzing can be found in Trail of Bits' "Learn how to fuzz like a pro" series on YouTube.

- Consider making a list of invariants of the protocol to check one by one if they are properly tested. We collected the following system invariants from the written specification that can be used as a starting point for defining a good set of invariants:
  - The express lane controller (ELC) is determined by a per-round auction.
  - The auction is implemented as a sealed-bid, second-price auction.

- The auction is managed on-chain by an auction contract, and by an "autonomous auctioneer."
- The auctioneer is designated by governance.
- The auction has a minimum reserve price.
- The minimum reserve price is set by governance.
- The contract stores the minimum and current reserve prices.
- The contract enforces that the current reserve price is not less than the minimum reserve price.
- Governance can designate a reserve pricer.
- The reserve pricer can update the reserve price to any value equal or greater to the minimum.
- There is a blackout period during which the reserve price cannot be updated by the pricer.
- This window is `ReserveSubmissionSeconds` seconds before bidding for a round closes.
    - For a round that starts at time `t`, the blackout period begins at `t - AuctionClosingSeconds - ReserveSubmissionSeconds` and lasts until the round ends.
- Bidders know the reserve price for at least `ReserveSubmissionSeconds` before they must submit their bids.
- A party must deposit funds into the auction contract before bidding.
- Deposits can be made at any time.
- Funds can be added to an existing deposit at any time.
- A party can initiate a withdrawal of all of its deposited funds at any time.
- A withdrawal request submitted during round `i` will be claimable at round `i + 2` or later.
- The auction has a closing time.
- The closing time is determined by `AuctionClosingSeconds`.
- The default value of `AuctionClosingSeconds` is 15 seconds.

- Any party can submit a bid to the auctioneer via RPC until closing time.
- A "bid" contains (`chainID, auctionContractAddress, roundNumber, bid, expressLaneControllerAddress, signature`) where:
  - `chainID` is the chain ID of the target chain
  - `auctionContractAddress` is the address of the auction contract the bid is to be submitted to
  - `roundNumber` is the round being bid for (i.e., the current round + 1)
  - `bid` is the value the party is offering to pay
  - `expressLaneControllerAddress` is the address to set as the express lane controller if their bid is successful
  - `signature` is the signature of the bidder's private key across all other items in the tuple
- If two or more bids meet the reserve price and the bidders have sufficient deposits for their bids, the auctioneer will:
  - call `resolveAuctionMultiBid`, passing the two highest bids
  - if the two bids tie, they are sorted by `keccak256(bidder address, byte-string of bid)`
  - the contract verifies:
    - the signatures of both bids
    - both bids are signed by different addresses
    - both bids are backed by funds deposited in the contract
    - both bids meet the reserve price
  - the contract deducts the second-highest bid from the account of the highest bidder
  - the funds are transferred to a beneficiary account or burned
    - governance specifies the beneficiary or if the tokens are to be burned
  - the `expressLaneControllerAddress` address from the highest bid will be the ELC for the round

- If exactly one bid meets the reserve price and the bidder has sufficient deposits for its bid:
  - the auctioneer will call `resolveAuctionSingleBid`, passing the single bid
  - the contract verifies:
    - the signature of the bid
    - the bid is backed by funds deposited in the contract
    - the bid meets the reserve price
  - the contract deducts the reserve price from the account of the bidder
  - the funds are transferred to a beneficiary account or burned
    - governance specifies the beneficiary or if the tokens are to be burned
  - the `expressLaneControllerAddress` address from the highest bid will be the ELC for the round
- If there are no bids that meet the reserve price or have sufficient deposits for their bids
  - the auctioneer will call no functions
  - no address will be the ELC for the round
- A round lasts `RoundDurationSeconds` seconds.
- The next round begins when the current one ends.
- For a round that becomes active at time `t`:
  - the auctioneer closes the round between timestamps `t - AuctionClosingSeconds` and `t`
  - the auctioneer is trusted to follow the instructions above
  - this may designate a party as the ELC for the round
  - if an ELC is designated, funds are deducted from that party's deposit account
  - the deducted funds are distributed or burned

- at any time between being designated the ELC and the end of the round (i.e., `t + RoundDurationSeconds`), the ELC can transfer the ELC right for that round to a different address
    - the previous ELC retains no rights once transferred
    - this can be done multiple times per round, as long as it is executed by the then-current ELC
- at `t + RoundDurationSeconds`, the sequencer stops accepting express lane messages for the round