Blast

Security Assessment

February 2, 2024

Prepared for:
Blast
MetaLayer Labs

Prepared by: **Anish Naik, Damilola Edwards, and David Pokora**

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:

Anne Marie Barry, Project Manager
annemarie.barry@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:

Anish Naik, Consultant
anish.naik@trailofbits.com

Damilola Edwards, Consultant
damilola.edwards@trailofbits.com

David Pokora, Consultant
david.pokora@trailofbits.com

Project Timeline

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

Date	Event
January 2, 2024	Pre-project kickoff call
January 8, 2024	Status update meeting #1
January 15, 2024	Delivery of report draft
January 15, 2024	Report readout meeting
February 2, 2024	Delivery of comprehensive report

Executive Summary

Engagement Overview

MetaLayer Labs engaged Trail of Bits to review the security of Blast, an Ethereum L2 solution aiming to incentivize developers by offering lower fees for staking L1 deposited funds and redistributing rewards.

A team of three consultants conducted the review from January 2 to January 12, 2024, for a total of four engineer-weeks of effort. Our testing efforts focused on uncovering improper arithmetic related to user funds, undefined behavior, trapped state transitions, and denial-of-service conditions. With full access to the source code and documentation, we performed static testing of Blast, using automated and manual processes. Due to the size of the project and the engagement scope, we gave the project a best-effort review, prioritizing risks noted to be of most importance to the Blast team.

Observations and Impact

Blast includes mature considerations for arithmetic and data validation in the code paths covered in the engagement. We did not discover many of the concerns that typically plague similar systems during the engagement. We did not uncover any EMV-compliance issues with respect to execution semantics (excluding the use of the added precompiled contracts) or issues related to solvency or theft of funds from the Blast bridge.

Recommendations

Based on the codebase maturity evaluation and findings identified during the security review, and considering the size of the project relative to the engagement scope, Trail of Bits recommends that MetaLayer Labs take the following steps prior to mainnet deployment:

- **Remediate the findings disclosed in this report.** These findings should be addressed as part of a direct remediation or as part of any refactor that may occur when addressing other recommendations.
- **Integrate dynamic fuzz testing.** Using dynamic fuzz testing for arithmetic-heavy areas (e.g., gas claiming) and for the properties of the rebasing tokens (USDB and WETH) will significantly aid in validating the expected behavior of the system. The `crytic/properties` repository can be used as a starting point to ensure that ERC-20-related properties are maintained.
- **Develop an EVM tracer to validate the gas tracker.** Using an EVM tracer that can validate the behavior of the gas tracker across various call scopes and contexts will be beneficial in validating the invariants of the gas tracker.

Finding Severities and Categories

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

EXPOSURE ANALYSIS

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

CATEGORY BREAKDOWN

Category	*Count*
Data Validation	3
Undefined Behavior	1

Project Goals

The engagement was scoped to provide a security assessment of the Blast system. Specifically, we sought to answer the following non-exhaustive list of questions:

- Is the arithmetic prone to errors that may allow currency/tokens to be minted?
- Can currency/tokens be caught in a trapped state transition (unable to be withdrawn)?
- Does the system correctly track gas usage across contracts with its `GasTracker` provider?
- Are shares and share prices updated appropriately in the pre-deployed L2 contracts?
- Are there any EVM-compliance issues that would disallow a native Ethereum-compatible smart contract to run on Blast's L2?
- Do the introduced EVM precompile contracts allow for reentrancy? Do they account for user shares appropriately?
- Were any existing RPC endpoints or other code paths modified in a way that might break existing external tooling in the Ethereum ecosystem?
- Are staked funds appropriately tracked? Are rebasing currencies prone to flaws in arithmetic?

Project Targets

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

blast-audit-tob

Repository	https://github.com/blast-io/blast-audit-tob
Version	`be51873cb7740d6044fc41ce218ee4f1bd601c32`
Type	Ethereum
Platforms	Go, Solidity

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:

- **Deposits:** We performed a manual review of the code handling ETH and token deposits from L1 to L2. Specifically, we did the following:
    - Checked whether it is possible to bridge any tokens that are not currently supported by the bridge
    - Analyzed the decimal conversions between stablecoins for general correctness and looked for possible arithmetic edge cases such as divide-by-zero errors and overflows
    - Investigated opportunities to bypass access controls to maliciously allowlist a stablecoin or steal funds
    - Checked whether deposits allow for unexpected arbitrary execution of calldata that could lead to a loss of funds or reentrancy
    - Investigated whether the L2 bridge correctly handles ETH transfers and the minting of USDB
    - Checked that all necessary state variables are updated upon user deposit
    - Evaluated any deviations made from the flow of tokens in vanilla Optimism, which led to the discovery that ETH yield token deposits lose replay capabilities on L2 (TOB-BLAST-2)
- **Withdrawals:** We performed a manual review of the code handling ETH and USDB withdrawals from L2 and L1. Specifically, we did the following:
    - Looked for any opportunities to bypass access controls to steal funds, (un)stake funds, or cause insolvency issues
    - Validated that users can receive, at most, the nominal amount of ETH or USDB burned on L2
    - Evaluated whether it is possible to bypass any step within the withdrawal flow to steal user funds or bypass the dispute period
    - Investigated whether proven withdrawals that are later disputed may adversely affect the solvency of the bridge

- Assessed the `WithdrawalQueue` arithmetic and data validation for general correctness and checked that users cannot use an invalid checkpoint for withdrawals
- Checked whether token or ETH airdrops can adversely affect any state variables or the withdrawal flow
- Evaluated whether it is possible to manipulate share price to receive more funds than expected

- **Yield reporting and accounting:** We performed a manual review of the yield tracking, reporting, and accounting mechanisms. Specifically, we did the following:
  - Checked whether yield reporting correctly accounts for insurance deductions and handles cases of negative yield
  - Analyzed the staking and unstaking logic, which led to the discovery that the Blast admin can stake more funds than expected, causing temporary insolvency issues (TOB-BLAST-3)
  - Evaluated whether yield commits correctly update the necessary state variables
  - Checked whether the USDB and `Shares` pre-deployed contracts correctly update the share price and accept only reports that increase the share price
  - Checked whether it is possible for an attacker to steal insurance funds or funds locked within the yield managers
  - Checked the calculation of the share price to ensure that it cannot exceed 1e27
  - Tracked where the various accounting state variables are updated to ensure that an attacker cannot update them to maliciously affect the state machine and also for general correctness
- **Gas tracking:** We performed a manual review of the gas tracking logic in the `op-geth` project. Specifically, we did the following:
  - Reviewed the gas tracking implementation across all EVM call scopes and calling conventions for a given transaction or block for general correctness
  - Evaluated whether there is a sequence of transactions that could trigger a panic within the state machine and cause block production issues

- Investigated how the gas tracker is affected by reverted transactions, self-destructs, and proxy calls to identify any edge cases where gas is tracked incorrectly

- Verified the gas allocation logic for denial-of-service risks, which led to the discovery of a possible denial-of-service vector due to the fact that the protocol does not charge for `sstore` operations on the `Gas` pre-deployed contract (TOB-BLAST-4)

- **EVM compliance:** We performed a manual review of the system's EVM semantics. Specifically, we did the following:

  - Investigated whether added precompiled contracts violate standard EVM conventions (e.g., allow for reentrant calls, state changing operations in static/read-only contexts, etc.)

  - Checked that added precompiled contracts are not retroactively added to prior forks of Ethereum and do not use a monotonically increased address value, which may collide with official precompiled contracts introduced in future forks of Ethereum

  - Analyzed whether gas costs during EVM execution are altered in a way that affects the number of instructions that can be executed with a given call scope/state

  - Checked for conditions that may trigger a trapped state transition (e.g., inability to call a function)

  - Partially investigated the effects of changes (e.g., to JSON-RPC endpoints) on external standard Ethereum tooling for potential non-compliance

- **Share management:** We performed a manual review of the EVM state object used for tracking rebasing ETH (via shares and remainders). Specifically, we did the following:

  - Checked that changes to yield modes do not lead to the unexpected creation or destruction of ETH

  - Analyzed the arithmetic logic for updating user balances to ensure that there are no risks of division-by-zero errors or precision loss

- **Yield and gas claiming:** We reviewed the yield and gas claiming logic. Specifically, we did the following:

    - Checked whether the `Yield` precompiled contract charges a sufficient amount of gas and whether it has the necessary access controls to ensure that only the `Blast` pre-deployed contract can invoke it
    - Reviewed the `Gas` pre-deployed contract to ensure that the arithmetic calculations honor the "tax rate curve," correctly calculate the integral of gas over time, and do not allow users to claim more gas than expected
- **Rebasing USDB and WETH contracts:** We reviewed the rebasing USDB and WETH pre-deployed contracts. Specifically, we did the following:
    - Evaluated whether the WETH and USDB pre-deployed contracts correctly track user balances across token transfers, mints, and burns regardless of a user's yield mode
    - Analyzed the share price calculations for both tokens for general correctness

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:

- Further dynamic testing of gas tracking across the contracts/calls is needed.
- Increased investigation into possible denial-of-service conditions related to pre-deployed contract storage writes, which users are not charged for, is needed.
- Dynamic fuzz testing of complex arithmetic operations and testing of ERC-20 invariants should be performed.
- Interactions with the Lido and DSR protocols should be reviewed to ensure they are done correctly.
- We did not assess the logic of the `DSRYieldProvider.stakedValue` function; this function needs review.
- Further review of economic incentives as dynamic variables shift (e.g., cost incentives, yields) should be performed.

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	The Solidity bridge contracts and pre-deployed contacts are compiled with version 0.8.x of the Solidity compiler, which includes overflow protection by default. We did not identify any arithmetic issues on the Solidity or `op-geth` side that could lead to rounding errors, division-by-zero errors, theft of funds, or denial-of-service attack vectors (e.g., panics), or that could trigger unexpected behavior. It would be beneficial to have detailed documentation highlighting the arithmetic related to gas claiming and investing efforts into dynamic fuzz testing for additional validation of the arithmetic.	**Satisfactory**
Auditing	Auditing and logging controls were not evaluated during the assessment because the system largely relies on existing event and transaction introspection mechanisms.	**Not Considered**
Authentication / Access Controls	We found that Blast employs appropriate access controls to safeguard sensitive system functions from non-admins of the system. In all cases covered during the review, we found that access controls appropriately safeguard the protocol against reentrancy attacks.	**Strong**
Complexity Management	The Blast system is a fork of Optimism, which means it inherits a lot of the complexity of Optimism (e.g., with regard to upgradeability and storage layouts). In addition to that, the concepts of rebasing currencies, yield reporting, and gas claiming/tracking break traditional EVM semantics and increase the complexity of the system.	**Moderate**

	That said, the changes made by the Blast team, although significant, are structured in a modular fashion, which makes it easy to reason through and test. The changes are generally well documented. Additionally, great care and intention were taken during the development process to generate the smallest necessary changeset possible while still achieving the expected behavior and invariants.	
Configuration	The configuration of the system was not fully evaluated, as the system is not actively in production. Although there is a theoretical denial-of-service vector against `op-geth` (TOB-BLAST-4), the configuration of default values otherwise observed in the system during the engagement were not observed to be problematic.	**Further Investigation Required**
Cryptography and Key Management	There were no changes made to the key management or account authorization logic on either the Solidity or `op-geth` side that required additional review.	**Not Applicable**
Data Handling	Outside of minor instances like TOB-BLAST-1 and TOB-BLAST-3, the Blast codebase has robust data validation and handling. Each individual function and workflow reviewed during the engagement has the necessary guard clauses, validation of user input, and access controls to ensure that no unexpected edge cases can be triggered to break the state machine.	**Satisfactory**
Decentralization	The Blast system relies heavily on the Blast admin to ensure that the bridge and the L2 chain work as expected. Facets such as bridge solvency, (un)staking of user funds, and yield reporting require the manual intervention of the admin. Additionally, the system inherits the centralization risks of Optimism (e.g., centralized sequencer). It is important to note that the decentralization properties of `op-geth` remain the same.	**Weak**
Documentation	The documentation for the Blast bridge is of generally high quality. We were provided with sufficient external and inline documentation to reason through the expected behavior. The documentation surrounding gas tracking and gas claiming, however, should be expanded. Concepts such as “gas seconds” and the “tax curve,” and	**Moderate**

	justification for why gas is allocated or refunded at various instances within the EVM state machine, remain unclear. Additionally, it would be beneficial for the Blast team to document all of the system invariants associated with the bridge and on the L2 side.	
Low-Level Manipulation	Low-level manipulation is used within the Blast system to perform delegate calls, read and write to storage, and execute `unchecked` blocks. We did not identify any issues with the above logic.	**Strong**
Memory Safety and Error Handling	We identified a number of panics introduced by the Blast team to handle cases in which gas tracking is not performed correctly. During the audit, we were unable to identify a series of transactions that could trigger these panics. Creating a custom EVM tracer that can programmatically validate the behavior of the gas tracker during a transaction would be beneficial.	**Further Investigation Required**
Testing and Verification	The project has many unit tests that test the custom functionality introduced by the Blast team. Additionally, all pertaining tests from the original Optimism repository continue to function as expected. However, it would be beneficial for the Blast team to invest additional effort into integration testing and dynamic fuzz testing of areas that have heavy arithmetic complexity (e.g., gas claiming and rebasing tokens). Similarly, developing a custom EVM tracer that can validate the expected behavior of the gas tracker would aid in validating the security properties of the system.	**Moderate**
Transaction Ordering	Transaction ordering risks were not reviewed during this audit. Outside of vectors such as front-running a yield report to extract value, we did not identify changes made by the Blast team that would significantly affect the semantics of traditional transaction ordering risks.	**Not Considered**

Summary of Findings

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

ID	Title	Type	Severity
1	isStakingEnabled returns true when Lido is paused	Data Validation	**Informational**
2	ETH yield token user deposits lose replay capabilities on the L2 bridge	Undefined Behavior	**Informational**
3	Insufficient funds may be available for user withdrawals	Data Validation	**Low**
4	Possible denial-of-service vector through pre-deployed contract storage writes	Data Validation	**Undetermined**

Detailed Findings

1. isStakingEnabled returns true when Lido is paused

Severity: **Informational**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-BLAST-1
Target: `optimism/packages/contracts-bedrock/src/mainnet-bridge/yield-providers/LidoYieldProvider.sol`	

Description

The `isStakingEnabled` function returns `true` when Lido has paused staking on its system. Instead, it should return `false` (figure 1.1).

```
/// @inheritdoc YieldProvider
function isStakingEnabled(address token) public view override returns (bool) {
    return token == address(LIDO) && LIDO.isStakingPaused();
}
```

Figure 1.1: The `isStakingEnabled` function incorrectly returns the status of whether Lido has paused staking.
(optimism/packages/contracts-bedrock/src/mainnet-bridge/yield-providers/LidoYieldProvider.sol#L53-L56)

The function should negate the return value of `LIDO.isStakingPaused`.

Recommendations

Short term, update the `isStakingEnabled` function, as shown in figure 1.2:

```
/// @inheritdoc YieldProvider
function isStakingEnabled(address token) public view override returns (bool) {
    return token == address(LIDO) && !LIDO.isStakingPaused();
}
```

Figure 1.2: This corrected version of the `isStakingEnabled` function correctly assesses whether `Lido` is paused.

Long term, improve the unit test coverage to validate all getter functions and their expected functionality.

2. ETH yield token user deposits lose replay capabilities on the L2 bridge

Severity: **Informational**	Difficulty: **High**
Type: Undefined Behavior	Finding ID: TOB-BLAST-2
Target: `optimism/packages/contracts-bedrock/src/mainnet-bridge/L1BlastBridge.sol`	

Description

ETH yield token deposits on the Blast bridge do not use the `L1CrossDomainMessenger` component to relay the message to L2. As a result, the default transaction replay capabilities provided by the messenger are not available to users.

On Optimism, the traditional flow for token deposits is for the L1 bridge to call the `L1CrossDomainMessenger.sendMessage` function. Internally, the messenger will invoke the `OptimismPortal.depositTransaction` function, where the calldata for the L2 side is for the `L2CrossDomainMessenger.relayMessage` function (figure 2.1). The `relayMessage` function provides transaction replay capabilities if the original transaction fails to execute successfully. These capabilities ensure that users' funds do not become locked in the bridge.

However, on Blast, ETH yield token deposits, such as deposits of stETH, do not go through the `L1CrossDomainMessenger`. Instead, they directly go through the `OptimismPortal.depositTransaction` function (figure 2.1). This function is used because stETH deposits on L1 must result in minted ETH instead of ERC-20 tokens on L2. To access such behavior, the system cannot go through the `L1CrossDomainMessenger`. Because of this deviation, the callee on L2 is not the `L2CrossDomainMessenger.relayMessage` function; instead, it is the `L2BlastBridge.finalizeETHBridgeDirect` function. If this function reverts, the L2 ETH will be locked on the bridge and the user will have no way to gain access to it.

```
function _initiateBridgeERC20(
    address _localToken,
    address _remoteToken,
    address _from,
    address _to,
    uint256 _amount,
    uint32 _minGasLimit,
    bytes memory _extraData
)
```

```
    internal
    override
{
    YieldToken memory usdYieldToken = usdYieldTokens[_localToken];
    YieldToken memory ethYieldToken = ethYieldTokens[_localToken];
    if (usdYieldToken.approved) {
        [...]
    } else if (ethYieldToken.approved) {
        [...]

        // Message has to be sent to the OptimismPortal directly because we have to
        // request the L2 message has value without sending ETH.
        portal.depositTransaction(
            Predeploys.L2_BLAST_BRIDGE,
            _amount,
            RECEIVE_DEFAULT_GAS_LIMIT,
            false,
            abi.encodeWithSelector(
                L2BlastBridge.finalizeBridgeETHDirect.selector,
                _from,
                _to,
                USDConversions._convertDecimals(_amount, ethYieldToken.decimals,
USDConversions.WAD_DECIMALS),
                _extraData
            )
        );

        [...]
}
```

Figure 2.1: ETH yield token deposits bypass the `L1CrossDomainMessenger`. (optimism/packages/contracts-bedrock/src/mainnet-bridge/L1BlastBridge.sol #L161-L241)

Recommendations

Short term, update any user- and developer-facing documentation to inform developers that ETH transfers on L2, especially when ETH yield tokens are deposited on L1, must not revert.

Long term, formally enumerate the invariants introduced across Blast. This may help prevent invariants from being violated with future changes to code, enable deeper internal security review, and assist external developers.

3. Insufficient funds may be available for user withdrawals

Severity: **Low**	Difficulty: **Low**
Type: Data Validation	Finding ID: TOB-BLAST-3
Target: `optimism/packages/contracts-bedrock/src/mainnet-bridge/yield-providers/LidoYieldProvider.sol`, `optimism/packages/contracts-bedrock/src/mainnet-bridge/yield-providers/DSRYieldProvider.sol`	

Description

The Lido and DSR yield providers might stake more funds than what is available, which could prevent user withdrawals from being finalized.

The `LidoYieldProvider` and `DSRYieldProvider` contracts are responsible for staking funds that are available in their respective yield managers into Lido and the DSR pot, respectively. For example, the `LidoYieldProvider.stake` function first checks that the amount of funds to stake is less than the available ETH balance in the `ETHYieldManager` (figure 3.1). Then, the function calls Lido to stake the necessary amount.

```
/// @inheritdoc YieldProvider
function stake(uint256 amount) external override onlyDelegateCall {
    if (amount > YIELD_MANAGER.lockedValue()) {
        revert InsufficientStakableFunds();
    }
    LIDO.submit{value: amount}(address(0));
}
```

Figure 3.1: The `stake` function may stake more funds than what is available in the `ETHYieldManager`. (optimism/packages/contracts-bedrock/src/mainnet-bridge/yield-providers/LidoYieldProvider.sol#L73-L79)

However, the function does not take into account the amount of ETH that is locked by the `WithdrawalQueue` to complete any outstanding withdrawals. Thus, the Blast admin could stake more than the available liquidity in the `ETHYieldManager`. Doing so would affect the solvency of the bridge temporarily since users may be unable to finalize their withdrawals.

Note that this issue also exists in the `DSRYieldManager`.

Exploit Scenario

Alice, the Blast admin, calls the `ETHYieldManager.stake` function with an amount that is more than what is available (including locked funds for withdrawals). Bob, a user of Blast, attempts to finalize his withdrawal request but is unable to do so because there are not enough funds left in the `ETHYieldManager`.

Recommendations

Short term, update the `LidoYieldProvider.stake` (figure 3.2) and `DSRYieldProvider.stake` (figure 3.3) functions as follows:

```
/// @inheritdoc YieldProvider
function stake(uint256 amount) external override onlyDelegateCall {
    if (amount > YIELD_MANAGER.getTokenBalance()) {
        revert InsufficientStakableFunds();
    }
    LIDO.submit{value: amount}(address(0));
}
```

Figure 3.2: This corrected version of the `LidoYieldProvider.stake` function ensures that only liquid funds are used for staking.

```
/// @inheritdoc YieldProvider
function stake(uint256 amount) external override onlyDelegateCall {
    if (amount > YIELD_MANAGER.getTokenBalance()) {
        revert InsufficientStakableFunds();
    }
    if (amount > 0) {
        DSR_MANAGER.join(address(YIELD_MANAGER), amount);
    }
}
```

Figure 3.3: This corrected version of the `DSRYieldProvider.stake` function ensures that only liquid funds are used for staking.

Long term, document all invariants of the system, including those pertaining to token accounting and solvency. Ensure that testing can validate these invariants and cover all edge cases.

4. Possible denial-of-service vector through pre-deployed contract storage writes

Severity: **Undetermined**	Difficulty: **High**
Type: Data Validation	Finding ID: TOB-BLAST-4
Target: `op-geth/core/genesis.go`, `op-geth/core/protocol_params.go`, `op-geth/core/state/statedb.go`	

Description

Blast tracks share prices and counts for each account address using pre-deployed contracts that are set within the chain's genesis block. As the EVM executes a transaction, it uses Blast's `GasTracker` provider to track gas rewards to disperse to developers. The refund provided on gas offers an incentive for developers to adopt Blast as their L2 solution.

However, Blast's pre-deployed contracts' getter and setter functions for these parameters use Ethereum account storage. Because storage writes are computationally expensive operations (due to the triggering of expensive account trie updates), the pre-deployed contracts should charge an appropriate amount in gas costs for each storage write to financially disincentivize attackers from leveraging them as a denial-of-service attack vectors, slowing down transaction processing rates.

Currently, Blast performs multiple account storage writes per unique contract address executed during a transaction. Thus, an attacker may attempt to perform a long list of calls in a transaction in an attempt to trigger storage writes. They may perform more calls than the maximum stack depth by adding a depth limit and iterating over a list of contracts at each depth level.

The severity of this issue is undetermined, as the feasibility of this attack was not determined during the course of this review. We rated the attack complexity as high for the following reasons:

- Contract deployments will be costly to the attacker.
- Performing the call afterwards will add additional financial overhead for the attacker, due to the number of calls performed.
- There must be a financial incentive to perform a denial-of-service attack that produces a larger return than the money spent.

When considering practical attack scenarios, note that some systems involving the pooling of assets (e.g., a liquidity pool) and fluctuating asset pricing based on availability may provide an incentive, if an attacker can disallow other users from funding a pool for some time.

Recommendations

Short term, evaluate the total aggregate cost of such attacks for a given number of contracts/calls to determine whether any attacks may be financially feasible. This could be done by subtracting the cost of the storage write operations from the amount refunded, but in practice, some refunds may not cover the cost fully. Doing so would only narrow the constraints for such attacks and may not eliminate them.

As a result of design constraints within Blast, it may be difficult to find viable solutions. Blast holds interest in maintaining EVM compliance such that traditional gas costs that influence the number of instructions that will be executed before gas runs out will not be violated. As a result, Blast's gas refunds were developed to occur alongside traditional Ethereum gas computations and are applied only at the end of transactions. Otherwise, the most ideal solution would be to immediately charge for the storage writes or ensure that enough gas is provided to account for them so that they are always accounted for.

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. Fix Review Results

When undertaking a fix review, Trail of Bits reviews the fixes implemented for issues identified in the original report. This work involves a review of specific areas of the source code and system configuration, not comprehensive analysis of the system.

From January 25 to January 26, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Blast team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, of the four issues described in this report, Blast has resolved three issues and has partially resolved one issue. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	isStakingEnabled returns true when Lido is paused	**Resolved**
2	ETH yield token user deposits lose replay capabilities on the L2 bridge	**Partially Resolved**
3	Insufficient funds may be available for user withdrawals	**Resolved**
4	Possible denial-of-service vector through pre-deployed contract storage writes	**Resolved**

Detailed Fix Review Results

TOB-BLAST-1: isStakingEnabled returns true when Lido is paused
Resolved in PR #14. The pull request introduced by the Blast team corrects the implementation of the `isStakingEnabled` function by negating the `LIDO.isStakingPaused` expression.

TOB-BLAST-2: ETH yield token user deposits lose replay capabilities on the L2 bridge
Partially resolved in PR #29. The `L1BlastBridge` contract was updated to use the `_minGasLimit` parameter in the `_initiateBridgeERC20` method for both ETH yield tokens and USD yield tokens, instead of a hard-coded gas limit. Although this mitigates the risk of a revert, the user still cannot replay on the L2 side. As highlighted in the finding, we recommend that the Blast team improve its user-facing documentation to highlight this edge case.

TOB-BLAST-3: Insufficient funds may be available for user withdrawals
Resolved in PR #5. The arithmetic surrounding withdrawals was updated to consider funds finalized in the withdrawal queue when computing the total amount of funds available.

TOB-BLAST-4: Possible denial-of-service vector through pre-deployed contract storage writes
Resolved in PR #2, PR #3, and PR #38. PR #2 lowers the cost of gas used by the `Gas` contract by packing the four previously written storage slot variables into a single storage slot. PR #3 introduces the respective changes within the `op-geth` component. Finally, PR #38 introduces an additional measure that penalizes each call scope/frame past the fifth frame, further disincentivizing denial-of-service attacks by increasing their cost. The fixes seem appropriate, as only a maximum of five discounted storage slot writes will be processed per transaction without proportionate costs, which is unlikely to notably delay transaction processing time.

D. Fix Review Status Categories

The following table describes the statuses used to indicate whether an issue has been sufficiently addressed.

Fix Status	
Status	**Description**
Undetermined	The status of the issue was not determined during this engagement.
Unresolved	The issue persists and has not been resolved.
Partially Resolved	The issue persists but has been partially resolved.
Resolved	The issue has been sufficiently resolved.