BOB FusionLock

Security Assessment (Summary Report)

April 3, 2024

Prepared for:
BOB Collective

Prepared by: **Justin Jacob**

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.
228 Park Ave S #80688
New York, NY 10003
https://www.trailofbits.com
info@trailofbits.com

Notices and Remarks

Copyright and Distribution

The sole canonical source for Trail of Bits publications is the Trail of Bits Publications page. Reports accessed through any source other than that page may have been modified and should not be considered authentic.

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:

Jeff Braswell, Project Manager
jeff.braswell@trailofbits.com

The following engineering director was associated with this project:

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

The following consultant was associated with this project:

Justin Jacob, Consultant
justin.jacob@trailofbits.com

Project Timeline

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

Date	Event
March 21, 2024	Pre-project kickoff call
March 25, 2024	Delivery of report draft
March 25, 2024	Report readout meeting
April 3, 2024	Delivery of summary report

Project Targets

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

FusionLock

Repository	https://github.com/bob-collective/fusion-lock
Version	`f65b5c58d495a80cafceab6bfa046b0d10fd90e1`
Type	Solidity
Platform	EVM

Executive Summary

Engagement Overview

The BOB Collective engaged Trail of Bits to review the security of its `FusionLock` contract. The contract is designed to lock users' ETH and ERC-20 deposits and allow them to bridge their funds to the BOB Collective's L2 blockchain. The contract also includes functionality for pausing withdrawals and deposits and withdrawing tokens on the L1 blockchain.

One consultant conducted the review from March 21 to March 22, 2024, for a total of two engineer-days of effort. With full access to source code and documentation, we performed static and dynamic testing of the target, using automated and manual processes.

Observations and Impact

The code is fairly straightforward and simple to understand. We found three minor issues regarding the lack of data validation for ownership transfers, pausing the contract's functions, and contract existence checks. The code relies on correctly interfacing with the Optimism bridge. While we did verify some basic functionality and integration with the bridge, we did not go into detail about attack vectors and scenarios regarding bridging.

Recommendations

Based on the findings in this report, we recommend that the BOB Collective take the following steps:

- **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.
- **Expand the testing suite.** The current testing suite is a good baseline, but further testing, such as fuzz testing tailored to protocol-specific invariants and complex scenarios mimicking real usage, will help uncover edge cases. Guidance on introducing stateful fuzzing can be found in Trail of Bits' Learn how to fuzz like a pro series on YouTube.

Summary of Findings

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

ID	Title	Type	Severity
1	withdrawDepositsToL1 is lacking a pausable modifier	Data Validation	**Informational**
2	Lack of two-step process for ownership transference	Data Validation	**Low**
3	Lack of zero-address checks in setBridgeProxyAddress	Data Validation	**Informational**

A. Incident Response Recommendations

This section provides recommendations on formulating an incident response plan.

- **Identify the parties (either specific people or roles) responsible for implementing the mitigations when an issue occurs (e.g., deploying smart contracts, pausing contracts, upgrading the front end, etc.).**
- **Document internal processes for addressing situations in which a deployed remedy does not work or introduces a new bug.**
    - Consider documenting a plan of action for handling failed remediations.
- **Clearly describe the intended contract deployment process.**
- **Outline the circumstances under which the BOB Collective will compensate users affected by an issue (if any).**
    - Issues that warrant compensation could include an individual or aggregate loss or a loss resulting from user error, a contract flaw, or a third-party contract flaw.
- **Document how the team plans to stay up to date on new issues that could affect the system; awareness of such issues will inform future development work and help the team secure the deployment toolchain and the external on-chain and off-chain services that the system relies on.**
    - Identify sources of vulnerability news for each language and component used in the system, and subscribe to updates from each source. Consider creating a private Discord channel in which a bot will post the latest vulnerability news; this will provide the team with a way to track all updates in one place. Lastly, consider assigning certain team members to track news about vulnerabilities in specific system components.
- **Determine when the team will seek assistance from external parties (e.g., auditors, affected users, other protocol developers) and how it will onboard them.**
    - Effective remediation of certain issues may require collaboration with external parties.
- **Define contract behavior that would be considered abnormal by off-chain monitoring solutions.**

It is best practice to perform periodic dry runs of scenarios outlined in the incident response plan to find omissions and opportunities for improvement and to develop “muscle memory.” Additionally, document the frequency with which the team should perform dry runs of various scenarios, and perform dry runs of more likely scenarios more regularly. Create a template to be filled out with descriptions of any necessary improvements after each dry run.

B. Token Integration Checklist

The following checklist provides recommendations for interactions with arbitrary tokens. Every unchecked item should be justified and its associated risks understood. For an up-to-date version of the checklist, see `crytic/building-secure-contracts`.

For convenience, all Slither utilities can be run directly on a token address, such as the following:

```
slither-check-erc 0xdac17f958d2ee523a2206206994597c13d831ec7 TetherToken --erc erc20
slither-check-erc 0x06012c8cf97BEaD5deAe237070F9587f8E7A266d KittyCore --erc erc721
```

To follow this checklist, use the following output from Slither for the token:

```
slither-check-erc [target] [contractName] [optional: --erc ERC_NUMBER]
slither [target] --print human-summary
slither [target] --print contract-summary
slither-prop . --contract ContractName # requires configuration, and use of Echidna
and Manticore
```

General Considerations

- ❏ **The contract has a security review.** Avoid interacting with contracts that lack a security review. Check the length of the assessment (i.e., the level of effort), the reputation of the security firm, and the number and severity of the findings.
- ❏ **You have contacted the developers.** You may need to alert their team to an incident. Look for appropriate contacts on `blockchain-security-contacts`.
- ❏ **They have a security mailing list for critical announcements.** Their team should advise users when critical issues are found or when upgrades occur.

Contract Composition

- ❏ **The contract avoids unnecessary complexity.** The token should be a simple contract; a token with complex code requires a higher standard of review. Use Slither's `human-summary` printer to identify complex code.
- ❏ **The contract uses `SafeMath` or Solidity 0.8.0+.** Contracts that do not use `SafeMath` require a higher standard of review. Inspect the contract by hand for `SafeMath`/Solidity 0.8.0+ usage.
- ❏ **The contract has only a few non-token-related functions.** Non-token-related functions increase the likelihood of an issue in the contract. Use Slither's `contract-summary` printer to broadly review the code used in the contract.

- ❏ **The token has only one address.** Tokens with multiple entry points for balance updates can break internal bookkeeping based on the address (e.g., `balances[token_address][msg.sender]` may not reflect the actual balance).

Owner Privileges

- ❏ **The token is not upgradeable.** Upgradeable contracts may change their rules over time. Use Slither's `human-summary` printer to determine whether the contract is upgradeable.
- ❏ **The owner has limited minting capabilities.** Malicious or compromised owners can misuse minting capabilities. Use Slither's `human-summary` printer to review minting capabilities, and consider manually reviewing the code.
- ❏ **The token is not pausable.** Malicious or compromised owners can trap contracts relying on pausable tokens. Identify pausable code by hand.
- ❏ **The owner cannot denylist the contract.** Malicious or compromised owners can trap contracts relying on tokens with a denylist. Identify denylisting features by hand.
- ❏ **The team behind the token is known and can be held responsible for misuse.** Contracts with anonymous development teams or teams that reside in legal shelters require a higher standard of review.

ERC-20 Tokens

ERC-20 Conformity Checks

Slither includes a utility, `slither-check-erc`, that reviews the conformance of a token to many related ERC standards. Use `slither-check-erc` to review the following:

- ❏ **`Transfer` and `transferFrom` return a Boolean.** Several tokens do not return a Boolean on these functions. As a result, their calls in the contract might fail.
- ❏ **The `name`, `decimals`, and `symbol` functions are present if used.** These functions are optional in the ERC-20 standard and may not be present.
- ❏ **`Decimals` returns a `uint8`.** Several tokens incorrectly return a `uint256`. In such cases, ensure that the value returned is less than 255.
- ❏ **The token mitigates the known ERC-20 race condition.** The ERC-20 standard has a known ERC-20 race condition that must be mitigated to prevent attackers from stealing tokens.

Slither includes a utility, `slither-prop`, that generates unit tests and security properties that can discover many common ERC flaws. Use `slither-prop` to review the following:

- ❏ **The contract passes all unit tests and security properties from `slither-prop`.** Run the generated unit tests and then check the properties with Echidna and Manticore.

Risks of ERC-20 Extensions

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❏ **The token is not an ERC-777 token and has no external function call in `transfer` or `transferFrom`.** External calls in the transfer functions can lead to reentrancies.
- ❏ **`Transfer` and `transferFrom` should not take a fee.** Deflationary tokens can lead to unexpected behavior.
- ❏ **Potential interest earned from the token is accounted for.** Some tokens distribute interest to token holders. This interest may be trapped in the contract if not accounted for.

Token Scarcity

Reviews of token scarcity issues must be executed manually. Check for the following conditions:

- ❏ **The supply is owned by more than a few users.** If a few users own most of the tokens, they can influence operations based on the tokens' repartition.
- ❏ **The total supply is sufficient.** Tokens with a low total supply can be easily manipulated.
- ❏ **The tokens are in more than a few exchanges.** If all the tokens are in one exchange, a compromise of the exchange could compromise the contract relying on the token.
- ❏ **Users understand the risks associated with a large amount of funds or flash loans.** Contracts relying on the token balance must account for attackers with a large amount of funds or attacks executed through flash loans.
- ❏ **The token does not allow flash minting.** Flash minting can lead to substantial swings in the balance and the total supply, which necessitate strict and comprehensive overflow checks in the operation of the token.

ERC-721 Tokens

ERC-721 Conformity Checks

The behavior of certain contracts may differ from the original ERC specification. Conduct a manual review of the following conditions:

- ❏ **Transfers of tokens to the `0x0` address revert.** Several tokens allow transfers to `0x0` and consider tokens transferred to that address to have been burned; however, the ERC-721 standard requires that such transfers revert.
- ❏ **`safeTransferFrom` functions are implemented with the correct signature.** Several token contracts do not implement these functions. A transfer of NFTs to one of these contracts can result in a loss of assets.
- ❏ **The `name`, `decimals`, and `symbol` functions are present if used.** These functions are optional in the ERC-721 standard and may not be present.
- ❏ **If it is used, `decimals` returns a `uint8(0)`.** Other values are invalid.
- ❏ **The `name` and `symbol` functions can return an empty string.** This behavior is allowed by the standard.
- ❏ **The `ownerOf` function reverts if the `tokenID` is invalid or is set to a token that has already been burned.** The function cannot return `0x0`. This behavior is required by the standard, but it is not always properly implemented.
- ❏ **A transfer of an NFT clears its approvals.** This is required by the standard.
- ❏ **The `tokenID` of an NFT cannot be changed during its lifetime.** This is required by the standard.

Common Risks of the ERC-721 Standard

To mitigate the risks associated with ERC-721 contracts, conduct a manual review of the following conditions:

- ❏ **The `onERC721Received` callback is accounted for.** External calls in the transfer functions can lead to reentrancies, especially when the callback is not explicit (e.g., in `safeMint` calls).
- ❏ **When an NFT is minted, it is safely transferred to a smart contract.** If there is a minting function, it should behave like `safeTransferFrom` and properly handle the minting of new tokens to a smart contract. This will prevent a loss of assets.
- ❏ **The burning of a token clears its approvals.** If there is a burning function, it should clear the token's previous approvals.