Ethereum Foundation Devcon Auction-Raffle Contracts

Security Assessment (Summary Report)

June 18, 2024

Prepared for:
Aya Miyaguchi
Ethereum Foundation

Prepared by: **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

© 2024 by Trail of Bits, Inc.

All rights reserved. Trail of Bits hereby asserts its right to be identified as the creator of this report in the United Kingdom.

This report is considered by Trail of Bits to be public information; it is licensed to the Ethereum Foundation under the terms of the project statement of work and has been made public at the Ethereum Foundation's request. Material within this report may not be reproduced or distributed in part or in whole without the express written permission of Trail of Bits.

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 consultants were associated with this project:

Michael Colburn, Consultant
michael.colburn@trailofbits.com

Project Timeline

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

Date	Event
May 28, 2024	Pre-project kickoff call
June 4, 2024	Delivery of report draft
June 4, 2024	Report readout meeting
June 18, 2024	Delivery of summary report

Project Targets

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

devcon-raffle

Repository	https://github.com/efdevcon/devcon-raffle
Version	`7be621f5ec3a923138d386726e0d112fa9b5b1f7`
Type	Solidity
Platform	EVM

Executive Summary

Engagement Overview

The Ethereum Foundation engaged Trail of Bits to review the security of its Devcon ticket auction and raffle contracts in the `efdevcon/devcon-raffle` repository at commit `7be621f`. The contracts implement a combination auction and raffle in a gas-efficient manner in order to sell Devcon tickets.

One consultant conducted the review from May 28 to June 3, 2024, for a total of one engineer-week 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

Our review of the `AuctionRaffle` contract itself looked for invalid state transitions, any way to bid without meeting the requirements or proper funding, or any way to remove more funds than expected from the system, both before or during the claim period. We also reviewed the use of the max heap to store the top bids to identify any logical errors that could result in the top bids not being accurately tracked.

The system relies on a Chainlink VRF to seed the raffle shuffle. We reviewed this integration to ensure that the request for randomness is submitted properly, that the callback is done properly, and that the raffle cannot start until the randomness is received. Finally, we reviewed the optimized Feistel Shuffle used to select raffle winners to identify any logical errors in the implementation or any inconsistencies with the Solidity reference implementation. Our review was limited to the Solidity smart contracts in the `packages/contracts` directory of the repository. Any files in the `packages/frontend` were considered out of scope for this review.

We did not identify any security issues as part of this review. However, we did note that the settlement of the auction and raffle are overly permissioned due to past versions of the contracts requiring randomness to be passed to the contracts to settle the raffle. These functions could be made permissionless to allow users to move the system into the next state without the owner's intervention.

Recommendations

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

- Clean up unused variables and references to former versions of the auction-raffle contracts.

- Consider removing the access controls from the auction and raffle settlement steps so that the system can fully progress through its lifecycle without any owner intervention.

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	Most of the arithmetic used in the system is straightforward and easy to understand. Even the most complex calculations, which are in the Feistel Shuffle library, are made up of a few simple modular arithmetic operations.	**Satisfactory**
Auditing	The system emits events for any core state-changing operations during the lifecycle of the system until the claim deadline.	**Satisfactory**
Authentication / Access Controls	The contract has one explicit role, its owner, that must settle the auction and raffle and can claim proceeds and unclaimed funds once everything has settled. Aside from roles, the contract also uses off-chain signed attestations as an oracle to prove that users meet the minimum Gitcoin Passport score to participate (which can also be updated post-deployment by the owner). The system also gates any state-changing, publicly exposed functions based on the system's current state.	**Satisfactory**
Complexity Management	The overall system is broken down into functions with clearly defined scopes and logical contract components. The auction and raffle processes follow a well-defined state machine. The contracts do contain some references to an older version of the codebase that could be cleaned up (e.g., the `_raffleWinners` array, comments referencing a 2% fee that is no longer assessed).	**Satisfactory**
Decentralization	While the contract mostly functions autonomously once deployed, it does require the owner to trigger the settlement of both the auction and raffle stages. In the event that access to this key is lost, or inaction by a	**Moderate**

	malicious owner, the auction and raffle would not be able to be settled. As a result, after the claim deadline, funds would only be claimable by the contract owner, even for failed bids that should be fully refunded.	
Documentation	The repository includes high-level documentation of the expected state progression of the auction and raffle; a sequence diagram depicting retrieval of the Gitcoin Passport attestation off-chain; and a link to background on the Feistel Shuffle and how it can be used to make a more efficient raffle compared to past years' Devcon ticket raffles.	**Satisfactory**
Low-Level Manipulation	The actual Feistel Shuffle implementation used by the `AuctionRaffle` contract is a gas-optimized library implemented in assembly. Individual assembly statements are not heavily compounded, and the library is fairly thoroughly commented, which makes it easier to compare with the reference Solidity implementation.	**Satisfactory**
Testing and Verification	The codebase has unit and end-to-end tests, as well as differential testing of the Feistel Shuffle. Over the long term, we also recommend exploring techniques such as fuzzing, especially for the library contracts.	**Satisfactory**
Transaction Ordering	Though the end time of the bidding period is well known in advance and could be subject to last-minute bidding wars and MEV, the requirement to supply an attestation retrieved off-chain makes this more difficult to carry out in practice. An additional mitigating factor is that failed bids are automatically considered for the raffle portion of the ticket distribution.	**Satisfactory**

Summary of Findings

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

ID	Title	Type	Severity
1	Risk of funds becoming trapped if owner key is lost before raffle settlement	Access Controls	Low

Detailed Findings

1. Risk of funds becoming trapped if owner key is lost before raffle settlement

Severity: **Low**	Difficulty: **High**
Type: Access Controls	Finding ID: TOB-DEVCON-1
Target: `AuctionRaffle.com`	

Description

Due to overly restrictive access controls on the functions used to settle the auction and raffle, in the event that access to the owner key is lost before these are settled, there will be no way for users to reclaim their funds, even for unsuccessful bids.

A previous version of the `AuctionRaffle` contract required a random seed value when calling `settleRaffle`, so the settlement functions included the `onlyOwner` modifier. The current version of the contract relies on Chainlink's Verifiable Random Function (VRF) service to request randomness on-chain as part of the raffle settlement flow, and neither the `settleAuction` or `settleRaffle` functions take any parameters (figure 1.1).

```
/**
 * @notice Draws auction winners and changes contract state to AUCTION_SETTLED.
 * @dev Removes highest bids from the heap, sets their WinType to AUCTION and adds
them to _auctionWinners array.
 * Temporarily adds auction winner bidderIDs to a separate heap and then retrieves
them in descending order.
 * This is done to efficiently remove auction winners from _raffleParticipants array
as they no longer take part
 * in the raffle.
 */
function settleAuction() external onlyOwner onlyInState(State.BIDDING_CLOSED) {
    _settleState = SettleState.AUCTION_SETTLED;
    ...
}

/**
 * @notice Initiate raffle draw by requesting a random number from Chainlink VRF.
 */
function settleRaffle() external onlyOwner onlyInState(State.AUCTION_SETTLED)
returns (uint256) {
    ...
}
```

Figure 1.1: The `settleAuction` and `settleRaffle` function declarations (`AuctionRaffle.sol#L122-L161`)

In order for users to recover their funds (for the "golden ticket" winner of the raffle, funds in excess of the reserve price for other raffle winners, and all other unsuccessful bidders), the contract must be in the RAFFLE_SETTLED state, which is the state the contract remains in until the claiming period closes (figure 1.2).

```
function getState() public view returns (State) {
    if (block.timestamp >= _claimingEndTime) {
        return State.CLAIMING_CLOSED;
    }
    if (_settleState == SettleState.RAFFLE_SETTLED) {
        return State.RAFFLE_SETTLED;
    }
    if (_settleState == SettleState.AUCTION_SETTLED) {
        return State.AUCTION_SETTLED;
    }
    if (block.timestamp >= _biddingEndTime) {
        return State.BIDDING_CLOSED;
    }
    if (block.timestamp >= _biddingStartTime) {
        return State.BIDDING_OPEN;
    }
    return State.AWAITING_BIDDING;
}
```

Figure 1.2: The `AuctionRaffle` contract's `getState` function, which lists the contract's states in reverse chronological order (`AuctionRaffle.sol#L327-L324`)

In the unlikely event that the team managing the `AuctionRaffle` contract loses access to the contract's owner key before settling the raffle, according to figure 1.2, the contract state machine will be unable to progress until the current time reaches the value in the `_claimingEndTime` variable. After that point, the only notable function in the contract is `withdrawUnclaimedFunds` (figure 1.3), which can be called only by the contract's owner. As a result, any funds escrowed in the contract as part of the auction and raffle will be unrecoverable.

```
/**
 * @notice Allows the owner to withdraw all funds left in the contract by the
participants.
 * Callable only after the claiming window is closed.
 */
function withdrawUnclaimedFunds() external onlyOwner
onlyInState(State.CLAIMING_CLOSED) {
    uint256 unclaimedFunds = address(this).balance;
    payable(owner()).transfer(unclaimedFunds);
}
```

Figure 1.3: The `withdrawUnclaimedFunds` function body (`AuctionRaffle.sol#L234-L241`)

Exploit Scenario

The team loses access to the owner key of the `AuctionRaffle` contract late in the bidding process. The auction and raffle can no longer be settled due to the `onlyOwner` modifier on the functions that trigger these state transitions, so the contract will not enter the claim period. As a result, all of the funds for successful and unsuccessful bids will be considered "unclaimed" and will be recoverable by the contract owner only if access to the owner key is regained.

Recommendations

Short term, remove the `onlyOwner` modifier from the `settleAuction` and `settleRaffle` functions in the AuctionRaffle contract. This will allow the system to progress through all of its states without owner intervention once deployed.

A. 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.

B. 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.

On June 6, 2024, Trail of Bits reviewed the fixes and mitigations implemented by the Ethereum Foundation team for the issues identified in this report. We reviewed each fix to determine its effectiveness in resolving the associated issue.

In summary, the Ethereum Foundation has resolved the issue identified in this report. For additional information, please see the Detailed Fix Review Results below.

ID	Title	Status
1	Risk of funds becoming trapped if owner key is lost before raffle settlement	**Resolved**

Detailed Fix Review Results

TOB-DEVCON-1: Risk of funds becoming trapped if owner key is lost before raffle settlement

Resolved in PR 67. The `onlyOwner` modifier was removed from the `settleAuction` and `settleRaffle` functions, and they can now be called by anyone.

C. 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.