BlockÂpex

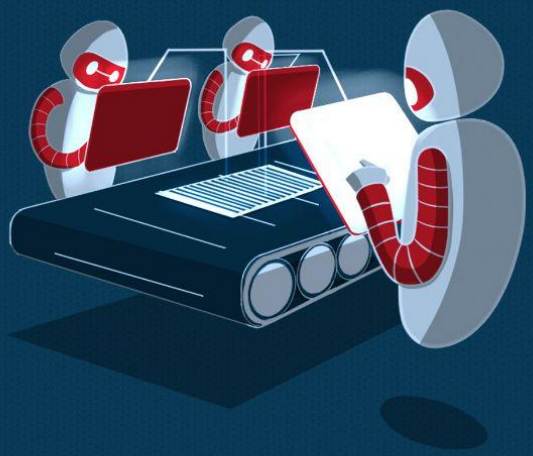
SMART CONTRACT SECURITY ANALYSIS REPORT

```
pragma solidity 0.7.0;
contract Contract {

    function hello() public returns (string) {
        return "Hello World!";
    }

    function findVulnerability() public returns (string) {
        return "Finding Vulnerability";
    }

    function solveVulnerability() public returns (string) {
        return "Solve Vulnerability";
    }
}
```



PREFACE

Objectives

The purpose of this document is to highlight any identified bugs/issues in the provided codebase. This audit has been conducted in a closed and secure environment, free from influence or bias of any sort. This document may contain confidential information about IT systems/architecture and the client's intellectual property. It also contains information about potential risks and the processes involved in mitigating/exploiting the risks mentioned below. The usage of information provided in this report is limited, internally, to the client. However, this report can be disclosed publicly to aid our growing blockchain community; at the client's discretion.

Key understandings

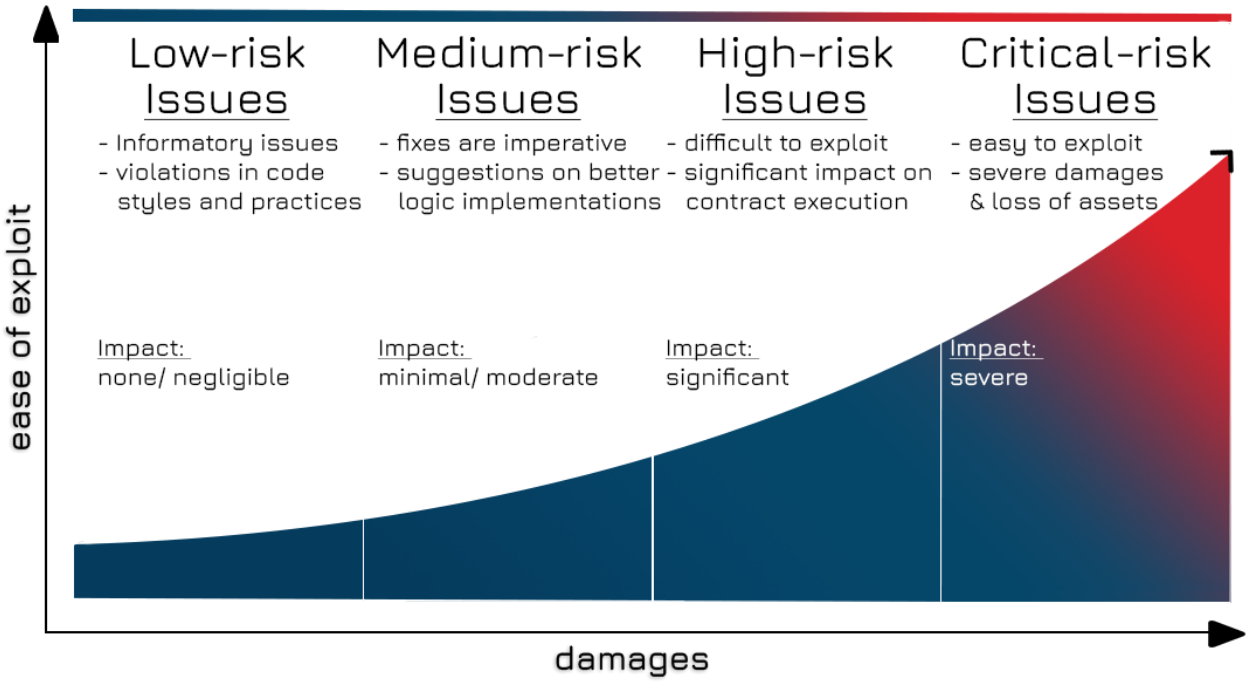


TABLE OF CONTENTS

PREFACE 2
- Objectives 2
- Key understandings 2

TABLE OF CONTENTS 3
INTRODUCTION 4
- Scope 5
- Project Overview 6
  - ENS Arora: 6
  - ENS Arora Public Resolver: 6
  - Ember Domain Registrar Arora: 6
- System Architecture 7
  - ENS Arora: 7
  - ENS Arora Public Resolver: 7
  - Ember Domain Registrar Arora: 7
- Methodology & Scope 8

AUDIT REPORT 9
- Executive Summary 9
- Key Findings 9
- Detailed Overview 11
  - High-risk issues 11
    - Potential DoS in Subdomain Registration Process 11
  - Informatory Issues and Optimizations 13
    - Inefficient Error Handling 13

DISCLAIMER 14

INTRODUCTION

BlockApex (Auditor) was contracted by Ember (Client) for the purpose of conducting a Smart Contract Audit/ Code Review. This document presents the findings of our analysis which started on 1st May '2023

Name
Ember
Auditor
BlockApex
Platform
Ethereum | Solidity
Type of review
Manual Code Review | Automated Tools Analysis
Methods
Architecture Review | Functional Testing | Computer-Aided Verification | Manual Review
Git repository/ Commit Hash
Private Repos with Commits- AroraPublicResolver, DomainRegistrarArora, ENS-Arora
White paper/ Documentation
-
Document log
Initial Audit Completed: May 29 '2023
Final Audit: (Pending)

Scope

The git-repository shared was checked for common code violations along with vulnerability-specific probing to detect major issues/vulnerabilities. Some specific checks are as follows:

Code review		Functional review
Reentrancy	Unchecked external call	Business Logics Review
Ownership Takeover	Fungible token violations	Functionality Checks
Timestamp Dependence	Unchecked math	Access Control & Authorization
Gas Limit and Loops	Unsafe type inference	Escrow manipulation
DoS with (Unexpected) Throw	Implicit visibility level	Token Supply manipulation
DoS with Block Gas Limit	Deployment Consistency	Asset's integrity
Transaction-Ordering Dependence	Repository Consistency	User Balances manipulation
Style guide violation	Data Consistency	Kill-Switch Mechanism
Costly Loop		Operation Trails & Event Generation

Project Overview

ENS Arora is a groundbreaking protocol developed by Ember, offering a comprehensive solution for the Ethereum Name Service (ENS). The ENS Arora project consists of three interconnected components: ENS Arora Public Resolver, Ember Domain Registrar Arora, and ENS Arora. Together, these components provide enhanced functionalities for managing ENS names and subdomains within the .ember domain.

ENS Arora:

ENS Arora serves as the foundation of the trio of projects, providing core functionality for managing ENS resource records. It acts as a registry, storing ownership and resolver information for each domain name. ENS Arora handles various operations on ENS resource records, including reading DNS format names and counting labels in a DNS name. Additionally, it incorporates a Resource Record Iterator for efficiently traversing resource records. This project serves as the backbone of the entire ENS system, ensuring the seamless operation of the other two projects.

ENS Arora Public Resolver:

ENS Arora Public Resolver extends the capabilities of the ENS Arora project by introducing a resolver contract. This contract enables users to query specific information associated with an ENS name. With ENS Arora Public Resolver, users can retrieve address records, access text records, and manage permissions for modifying the records.

Ember Domain Registrar Arora:

Ember Domain Registrar Arora offers a subdomain registrar contract specifically designed for the .ember top-level domain. Leveraging the ENS Arora and ENS Arora Public Resolver projects, this contract enables the registration and management of subdomains within the .ember domain. Users can set the ENS Registry and Resolver contract addresses, register new subdomains, and take ownership of these subdomains. Ember Domain Registrar Arora seamlessly interacts with both the ENS Registry from ENS Arora and the resolver contract from ENS Arora Public Resolver.

System Architecture

The architecture of the ENS Arora protocol has been thoughtfully designed to provide a robust and interconnected system for managing ENS names, with a particular focus on the .ember domain. Let's explore the architecture of each component:

ENS Arora:

ENS Arora utilizes a smart contract-based approach to manage ENS resource records. The core of the architecture revolves around a smart contract that acts as a registry for storing ownership and resolver information. This smart contract incorporates essential logic and functionalities to facilitate smooth operations, ensuring accurate and secure handling of transactions.

ENS Arora Public Resolver:

ENS Arora Public Resolver introduces a resolver contract that extends the functionality of ENS Arora. The resolver contract allows users to connect ENS names with the corresponding data they represent, such as Ethereum addresses or IPFS content hashes. This contract handles the resolution of human-readable ENS names into machine-readable addresses, providing a seamless link between the two.

Ember Domain Registrar Arora:

Ember Domain Registrar Arora builds upon the services provided by ENS Arora and ENS Arora Public Resolver to manage subdomains within the .ember domain. This component utilizes the ENS Registry and Resolver contracts to facilitate the registration and configuration of subdomains. Users can register new subdomains, define owners and resolvers, and transfer ownership to the registrant. Ember Domain Registrar Arora acts as a specialized service tailored for the .ember domain, ensuring efficient subdomain management.

The interconnectedness of these three projects enables a comprehensive system for handling ENS names within the Ember ecosystem. This setup enhances the user-friendliness of the Ethereum Name Service, making it easier to associate human-readable names with Ethereum addresses and other relevant data.

Methodology & Scope

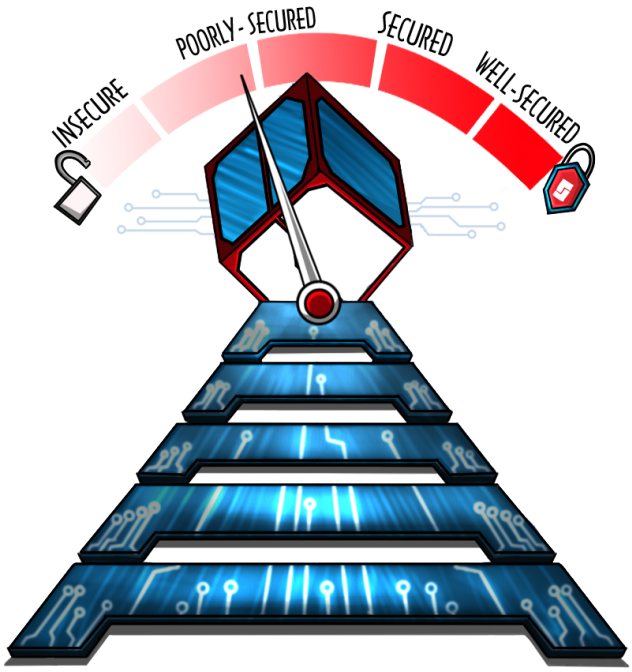
The codebase was audited using a filtered audit technique. A band of three (3) auditors scanned the codebase in an iterative process for a time spanning five (5) days.

Starting with the recon phase, a basic understanding was developed, and the auditors worked on developing presumptions for the developed codebase and the relevant documentation/whitepaper. Furthermore, the audit moved on with the manual code reviews with the motive to find logical flaws in the codebase complemented with code optimizations, software, and security design patterns, code styles, best practices, and identifying false positives that were detected by automated analysis tools.

AUDIT REPORT

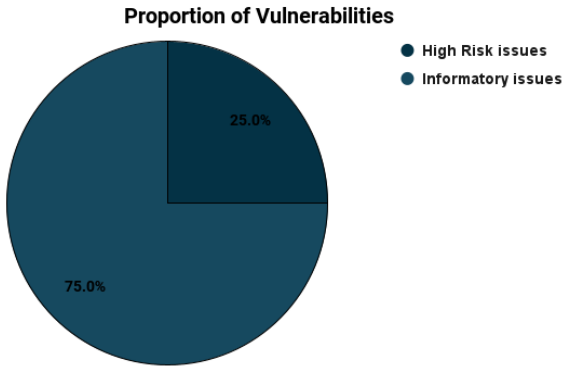
Executive Summary

Our team performed a technique called "Filtered Audit", where the Ember protocols were separately audited by three individuals.
After a thorough and rigorous process of manual testing, an automated review was carried out using Slither for static analysis and Foundry for fuzzing invariants. All the flags raised were manually reviewed and re-tested to identify the false positives.



Our team found:

#Issues	Severity Level
1	High-Risk issue(s)
1	Informatory issue(s)



Key Findings

#	Findings	Risk	Status
1	Potential DoS in Subdomain Registration Process	High	Fixed
2	Inefficient Error Handling	Info	Fixed

Detailed Overview

High-risk issues

ID	1
Title	Potential DoS in Subdomain Registration Process
Path	src/EmberSubdomainRegistrar.sol
Function Name	register

Description: In the register function of the smart contract, there is a potential front-running/ race-condition vulnerability. The function is responsible for registering subdomains of ".ember", where it creates hashes for the subdomains, checks if they already exist, and, if not, register them.

However, the function does not account for the possibility that an attacker might register a subdomain by watching the mempool for the batch registration call by the adminOnly function. In such a case, the attacker could cause the function to revert by registering one of the subdomains that the admin intends to register, forcing the entire transaction to fail repeatedly.

Attack Scenario: A malicious attacker could monitor transactions to identify calls to the register function that have not yet been mined. The attacker could then craft a transaction that registers one of the same subdomains with a higher gas fee to incentivize miners to include it in the blockchain first. If successful, this would cause the original register function call to fail due to the SubdomainAlreadyRegistered revert condition. This could potentially disrupt the normal operation of the contract and might be used in Denial of Service (DoS) attacks.

Vulnerable Code:

```
// Subdomain must not be registered already.
if (ens.recordExists(subdomainNodeHash)) {
    revert SubdomainAlreadyRegistered();
```

```
}

// Registering subdomain. Registering at this address so we can configure
it.
ens.setSubnodeRecord(
    DOMAIN_NODE_HASH,
    subdomainNameHash,
    address(this),
    address(resolver),
    0
);
```

In this code snippet, the function checks if a subdomain is already registered. If not, it registers the subdomain. However, a malicious attacker can register the same subdomain, leading to the potential front-running/race condition vulnerability.

Recommendation: It is recommended that continue/ break statements be utilized in the register function loop to remove any and all possible vulnerabilities of such a kind.

Informatory Issues and Optimizations

ID	2
Title	Inefficient Error Handling
Path	src/EmberSubdomainRegistrar.sol
Function Name	register

Description: In the register function of the smart contract, there is a potential issue with error handling in the loop where subdomains are registered. The function checks for invalid inputs, such as an address being zero or the length of a subdomain being zero, and if found, it reverts the entire transaction.

This might not be the most efficient way to handle these errors, especially in cases where there is a large list of subdomains to be registered. If an invalid entry is encountered halfway through the list, all preceding successful registrations would be needlessly reverted, consuming gas and potentially causing inconvenience for the users.

Vulnerable Code: The problematic part of the function is the loop where checks are made and where a revert is performed if any check fails:

```
for (uint256 i; i < _subdomainOwners.length; ) {
    if (_subdomainOwners[i] == address(0)) revert ZeroAddress();
    if (bytes(_subdomainNames[i]).length == 0) revert EmptyString();
    ...
}
```

In this code snippet, if any owner address is zero or if the length of any subdomain name is zero, the entire function call is reverted.

Recommendation: It is suggested to implement a more efficient way to skip over invalid entries and continue processing the remaining ones. Alternatively, these checks could be made before entering the loop, thus identifying and addressing issues before any registrations are attempted.

DISCLAIMER

The smart contracts provided by the client for audit purposes have been thoroughly analyzed in compliance with the global best practices to date w.r.t cybersecurity vulnerabilities and issues in smart contract code, the details of which are enclosed in this report.

This report is not an endorsement or indictment of the project or team, and they do not in any way guarantee the security of the particular object in context. This report is not considered and should not be interpreted as an influence on the potential economics of the token, its sale, or any other aspect of the project.

Crypto assets/tokens are the results of emerging blockchain technology in the domain of decentralized finance, and they carry with them high levels of technical risk and uncertainty. No report provides any warranty or representation to any third-Party in any respect, including regarding the bug-free nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service, or another asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and is not a guarantee as to the absolute security of the project.

Smart contracts are deployed and executed on a blockchain. The platform, its programming language, and other software related to the smart contract can have vulnerabilities that can lead to hacks. The scope of our review is limited to a review of the Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer or any other areas beyond Solidity that could present security risks.

This audit cannot be considered a sufficient assessment regarding the utility and safety of the code, bug-free status, or any other statements of the contract. While we have done our best in conducting the analysis and producing this report, it is important to note that you should not rely on this report only - we recommend proceeding with several independent audits and a public bug bounty program to ensure the security of smart contracts.