Spacebar Staking Contract Security Audit

: AO-Labs - Spacebar on Blast (Staking Contract)

Mar 22, 2024

Revision 1.0

ChainLight@Theori

Table of Contents

Executive Summary

Starting on Mar 13, 2024, ChainLight of Theori audited the Spacebar's staking contract for three days. The staking contract allows users to deposit their Blast ETH to boost Blast points while earning native yield. In the audit, we primarily considered the issues/impacts listed below.

- Theft or permanent freeze of staked funds
- Denial of Service
- Incorrect usage of Blast API's

As a result, we identified the issues listed below.

- Total: 5
- Low: 2 (Denial of Service via service hook, etc.)
- Informational: 3 (Minor issues including defense-in-depth suggestions.)

Audit Overview

Scope

Name	Spacebar Staking Contract Security Audit
Target / Version	• Git Repository (ao-labs/blast-staking-contract): commit `d91b9c97e8397b6a63962078cb20428765afc8cb` ~ `44ae5ab94f6c8dfd0cb3c98c002d2736e42f405a`
Application Type	Smart contracts
Lang. / Platforms	Smart contracts [Solidity]

Code Revision

N/A

Severity Categories

Severity	Description
Critical	The attack cost is low (not requiring much time or effort to succeed in the actual attack), and the vulnerability causes a high-impact issue. (e.g., Effect on service availability, Attacker taking financial gain)
High	An attacker can succeed in an attack which clearly causes problems in the service's operation. Even when the attack cost is high, the severity of the issue is considered "high" if the impact of the attack is remarkably high.
Medium	An attacker may perform an unintended action in the service, and the action may impact service operation. However, there are some restrictions for the actual attack to succeed.
Low	An attacker can perform an unintended action in the service, but the action does not cause significant impact or the success rate of the attack is remarkably low.
Informational	Any informational findings that do not directly impact the user or the protocol.
Note	Neutral information about the target that is not directly related to the project's safety and security.

Status Categories

Status	Description
Confirm	ChainLight reported the issue to the vendor, and they confirm that they received.
Reported	ChainLight reported the issue to the vendor.
Patched	The vendor resolved the issue.
Acknowledged	The vendor acknowledged the potential risk, but they will resolve it later.
WIP	The vendor is working on the patch.
Won't Fix	The vendor acknowledged the potential risk, but they decided to accept the risk.

Finding Breakdown by Severity

Category	Count	Findings
Critical	0	• N/A
High	0	• N/A
Medium	0	• N/A
Low	2	• `AOETHSTAKING-001` • `AOETHSTAKING-002`
Informational	3	• `AOETHSTAKING-003` • `AOETHSTAKING-004` • `AOETHSTAKING-005`
Note	0	• N/A

Findings

Summary

#	ID	Title	Severity	Status
1	AOETHSTAKING-001	Assembly call should be used in calls to hook functions in `EthStakeRegistry`	Low	WIP
2	AOETHSTAKING-002	`EthStakingContract.withdraw()` should use `call()` to transfer ETH	Low	WIP
3	AOETHSTAKING-003	`ReentrancyGuard` should be applied to `EthStakeRegistry.stake()/unstake()`	Informational	WIP
4	AOETHSTAKING-004	`_disableInitializers()` should be called in `EthStakingContract.constructor()`	Informational	WIP
5	AOETHSTAKING-005	Minor suggestions	Informational	WIP

#1 `AOETHSTAKING-001` Assembly call should be used in calls to hook functions in `EthStakeRegistry`

ID	Summary	Severity
`AOETHSTAKING-001`	If a service contract has a malicious hook function, the caller (`EthStakeRegistry`) may revert despite its exception handling.	Low

Description

In the `EthStakeRegistry` contract, a try-catch statement is used to catch all exceptions occurring in the service contract's hook function, allowing normal operation regardless of errors. However, various tricks exist to force the transaction to fail despite using a try-catch, such as reverting the internal transaction with a huge revert reason, returning malformed data, etc.

It may not be a critical problem in the case of `stake()`, but if a service's upgrade key is compromised and the attacker upgrades the contract to one with a malicious hook, causing `unstake()` to fail, funds will be frozen.

Impact

Low

Since the `EthStakingContract` is where the assets are stored, users will expect their funds to be accessible even if a service has issues as long as `EthStakingContract` is functional. However, this issue allows services to freeze funds arbitrarily.

Recommendation

An assembly call should be used, and return data should be copied only in a limited length or not copied at all. For the event emitting purpose, only the success of the hook should be checked; the return value should be ignored.

Remediation

WIP

It is patched as recommended using `ExcessivelySafeCall` library. However, the new version has not yet been deployed.

#2 `AOETHSTAKING-002` `EthStakingContract.withdraw()` should use `call()` to transfer ETH

ID	Summary	Severity
`AOETHSTAKING-002`	`EthStakingContract.withdraw()` should send ETH by using `call()` instead of `transfer()` to avoid a potential permanent freeze of funds staked by smart contracts.	Low

Description

As `transfer()` sends ETH with fixed gas fees, it may fail due to insufficient gas in `receive()` or `fallback()` when the receiver is a smart contract. Since the `withdraw()` accepts the destination address as an argument, withdrawal is still possible by specifying an EOA address. However, if a contract that cannot change the target address made a deposit, ETH may be permanently frozen.

Impact

Low

This is not much of a problem for contracts that can send arbitrary transactions, such as smart contract wallets. However, funds may be permanently frozen for contracts that are not upgradeable and have not mitigated this issue in advance.

Recommendation

In `EthStakingContract.withdraw()`, send ETH by `call()` instead of `transfer()`.

Remediation

WIP

It is patched as recommended. However, the new version has not yet been deployed.

#3 AOETHSTAKING-003 ReentrancyGuard should be applied to EthStakeRegistry.stake()/unstake()

ID	Summary	Severity
AOETHSTAKING-003	Since the stake() and unstake() may be re-entered either through hooks or ETH transfer via call(), the ReentrancyGuard modifier should be applied.	Informational

Description

In the EthStakeRegistry contract, stake() and unstake() call the hook of service before and after interaction with EthStakingContract. stake() and unstake() may be re-entered in the hook depending on its implementation. ETH transfer via call() also allows users to re-enter these functions. (If Issue 002 is fixed.)

Impact

Informational

Re-entering does not currently impact EthStakeRegistry. However, unexpected side effects may occur in each service.

Recommendation

Apply ReentrancyGuard to EthStakeRegistry.stake() and EthStakeRegistry.unstake().

Remediation

WIP

It is patched as recommended. However, the new version has not yet been deployed.

#4 `AOETHSTAKING-004` `_disableInitializers()` should be called in `EthStakingContract.constructor()`

ID	Summary	Severity
`AOETHSTAKING-004`	`EthStakingContract` should call `_disableInitializers()` in the `constructor()` to prevent potential DoS.	Informational

Description

`EthStakeRegistry.constructor()` creates an implementation contract `EthStakingContract`, but does not call `init()` or disable the initializer by other means.

It is not exploitable in the current code but may become a critical issue for future `EthStakingContract` versions. For instance, if `delegatecall()` to an arbitrary address is possible because of the exposed initializer function on a blockchain network where EIP-6780 is not applied, the implementation contract can be destroyed. In this case, clone contracts that refer to the implementation contract will be bricked. Therefore, it should be mitigated.

Impact

Informational

Recommendation

Add a call to `_disableInitializers()` to `EthStakingContract.constructor()`.

Remediation

WIP

It is patched as recommended. However, the new version has not yet been deployed.

#5 AOETHSTAKING-005 Minor suggestions

ID	Summary	Severity
AOETHSTAKING-005	The description includes suggestions for preventing incorrect settings caused by operational mistakes and improving code maturity.	Informational

Description

- In `EthStakeRegistry`, contract addresses (`blast`, `blastPoints`) are passed as constructor arguments. However, these addresses can be defined as constants to limit misconfiguration and improve readability.
- `payable` keyword has been added to the `constructor()` of `EthStakeRegistry`, but since this contract is not designed to hold ETH, it should be removed.

Impact

Informational

Recommendation

Apply the suggestions in the description above.

Remediation

WIP

All suggestions except (1) have been patched as recommended. However, the new version has not yet been deployed.

Revision History

Version	Date	Description
1.0	Mar 22, 2024	Initial version

Theori, Inc. (“We”) is acting solely for the client and is not responsible to any other party. Deliverables are valid for and should be used solely in connection with the purpose for which they were prepared as set out in our engagement agreement. You should not refer to or use our name or advice for any other purpose. The information (where appropriate) has not been verified. No representation or warranty is given as to accuracy, completeness or correctness of information in the Deliverables, any document, or any other information made available. Deliverables are for the internal use of the client and may not be used or relied upon by any person or entity other than the client. Deliverables are confidential and are not to be provided, without our authorization (preferably written), to entities or representatives of entities (including employees) that are not the client, including affiliates or representatives of affiliates of the client.