Venice Security Audit

Report Version 1.0

January 9, 2025

Conducted by **Hunter Security**

Table of Contents

1 About Hunter Security

Hunter Security is an industry-leading smart contract security auditing firm. Having conducted over 100 security audits protecting over $1B of TVL, our team always strives to deliver top-notch security services to the best DeFi protocols. For security audit inquiries, you can reach out on Telegram or Twitter at *@georgehntr*.

2 Disclaimer

Audits are a time-, resource-, and expertise-bound effort where trained experts evaluate smart contracts using a combination of automated and manual techniques to identify as many vulnerabilities as possible. Audits can reveal the presence of vulnerabilities, but cannot guarantee their absence.

3 Risk classification

Severity	Impact: High	Impact: Medium	Impact: Low
Likelihood: High	High	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

3.1 Impact

- **High** - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- **Medium** - involves a small loss of funds or affects a core functionality of the protocol.
- **Low** - encompasses any unexpected behavior that is non-critical.

3.2 Likelihood

- **High** - a direct attack vector; the cost is relatively low compared to the potential loss of funds.
- **Medium** - only a conditionally incentivized attack vector, with a moderate likelihood.
- **Low** - involves too many or unlikely assumptions; offers little to no incentive.

3.3 Actions required by severity level

- **High** - client **must** fix the issue.
- **Medium** - client **should** fix the issue.
- **Low** - client **could** fix the issue.

4 Executive summary

Overview

Project Name	Venice
Repository	https://github.com/shafu0x/venice
Commit hash	e3fd314e97c84c5c4484c40d2deb2ca55636a227
Resolution	-
Methods	Manual review & testing

Scope

src/Oracle.sol
src/Staking.sol
src/Venice.sol
script/Deploy.s.sol

Issues Found

High risk	0
Medium risk	0
Low risk	0
Informational	13

5 Findings

5.1 Informational

5.1.1 Typographical mistakes, non-critical issues or centralization vulnerabilities

Severity: Informational

Files: src/*

Description: The contracts contain one or more typographical mistakes, non-critical issues or centralization vulnerabilities. In an effort to keep the report size reasonable, we enumerate these below:

1. Redundant storage variable *updatedAt* in the Oracle contract.
2. The *initialize* method could be marked as *external* rather than *public*.
3. Rewards intended for the first token that is minted to *address(this)* upon construction cannot be claimed.
4. The *amount <= balanceOf(msg.sender)* check in *initiateUnstake* is redundant as the same would be performed when burning the passed amount of tokens.
5. It is generally considered a bad practise allowing users to execute actions on behalf of others without their permission as is the case in *stake*. Consider reducing the attack surface by letting users deposit tokens only for themselves.
6. Redundant *totalSupply() == 0* check in *_updateGlobalReward()* as the total supply can never be 0 after construction.
7. The *treasury* address should not be settable to *address(0)* as this would block the entire protocol functionality.
8. The *treasury* address should not be settable to *address(this)* as some rewards will become stuck that way.
9. Consider checking in *stake* that the passed *recipient* address is not *address(this)*.
10. Consider adding lower and upper limit in the *setEmissionRate* setter function to ensure values such as 0 or such causing an overflow cannot be set.
11. Consider adding an upper limit in the *setCooldownDuration* setter function to ensure no value can be set so that it keeps users' funds locked for too long.
12. The *cooldownDuration* is used in *finalizeUnstake* be having it added to the *cooldownStart*. However, if it has changed after the user has initiated their unstake, it could have an unexpected value. Consider storing a *cooldownEnd* rather than *cooldownStart*.
13. Consider implementing a timelock mechanism for more than *cooldownDuration* seconds to ensure no actions or implementation updates are possible before users have the chance to withdraw their funds.

Recommendation: Consider fixing the above typographical mistakes, non-critical issues or centralization vulnerabilities.

Resolution: Acknowledged.