Kernel Protocol Security Review

Report Version 1.0

June 14, 2024

Conducted by:

George Hunter, Independent Security Researcher

Table of Contents

1 About George Hunter

George Hunter is a leading smart contract security researcher and the founder of Hunter Security. Having conducted over 100 security reviews and reported tens of live smart contract security vulnerabilities protecting over $1B of TVL, he always strives to deliver top-quality security services to DeFi protocols. For security audit inquiries, you can reach out to him 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

George Hunter was engaged by Kernel Protocol (previously Vector Reserve) to review their smart contract protocol as part of an ongoing retainer partnership.

Overview

Project Name	Kernel Protocol
Repository	https://github.com/vectorreserve/kernel-contracts
Commit hash	31813ad912d8e1784f4beba9bd91d882e1327cf8
Resolution	cfe2efea9f93c46fed85e9098444de7db903f158
Methods	Manual review & testing

Scope

contracts/Kernel.sol
contracts/Migration.sol
contracts/Treasury.sol
contracts/kUSD.sol
contracts/krETH.sol
contracts/ksETH.sol
contracts/rates/*

Issues Found

High risk	0
Medium risk	0
Low risk	1
Informational	1

5 Findings

5.1 Low

5.1.1 Re-entrancy risk in burnAndRedeem

Severity: Low

Context: Treasury.sol#L95-L107

Description: The *Treasury.burnAndRedeem* function does not fully adhere to the Checks-Effects-Interactions pattern due to the external call (ERC20 transfer) made inside the *for* loop. In case that any of the *redeemableTokens* implements a callback/hook function, a malicious recipient may reenter into the function. After inspecting various possible scenarios, there seems to be no specific attack vector. However, due to caching the *_total* supply and corresponding *percent* before the *for* loop, it is advisory to apply a re-entrancy guard as an additional safety measurement.

Recommendation: Consider inheriting OpenZeppelin's *ReentrancyGuard* and applying the *nonReentrant* modifier to *Treasury.burnAndRedeem*.

Resolution: Resolved.

5.2 Informational

5.2.1 Typographical mistakes, non-critical issues and code-style suggestions

Severity: Informational

Context: .

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

1. Consider implementing a single token contract and deploying it using different initialization variables instead of reusing the same code for *krETH*, *ksETH* and *kUSD* in 3 different files.
2. All ERC20 token names are "Kernel Restaked ETH" despite the tokens being different.
3. Consider using the latest Solidity pragma version,
4. Consider inheriting OpenZeppelin's *Ownable2Step* instead of *Ownable*.
5. *Treasury.setRedeemableTokens* should not allow for duplicate elements.
6. *Treasury.addApprovedSender* and *Treasury.removeApprovedSender* can be called with addresses that are already approved/removed;
7. Consider whether *transferFromTreasury* should allow withdrawing redeemable tokens.
8. Consider using the *1e18* notation instead of *10 ** 18*.
9. Missing SPDX license identifier on several contract files.
10. Consider inheriting the *IKERN* interface in *Kernel*.
11. No need to check for maximum supply in *Kernel.mint* since max migrated amount check is already present in the migration contract.
12. The *approvedTokens* is never used on-chain and may lead to Out-Of-Gas exception if too many tokens have been added. Consider either removing it or using a different structure such as OpenZeppelin's *EnumerableSet*.

13. Not using safe approve (OpenZeppelin's *forceApprove*) in *deposit*.
14. Consider using *depositAndGimmie* instead of calling the functions separately.
15. Mint and redeem fees should not be applied when the caller is the fee recipient itself.
16. The _require(token.deposited >= *toSend)* check is unnecessary due to the default underflow checks performed on integer subtraction.
17. *RateProviderUpdated* event not emitted when a token is removed.
18. *redeem* not using safe transfer.

Recommendation: Consider fixing the above typographical mistakes, non-critical issues and code-style suggestions.

Resolution: Resolved.