QuakeCash Mitigation Audit Report

Reviewed by:

- ParthMandale

Prepared For: QuakeCash Protocol

Review Date(s): 11/8/25 - 14/8/25

About Security Researcher

Audit is led by a `ParthMandale` a experienced security researchers with over 1.5 years in the smart contract security space, with a proven track record of consistently uncovering vulnerabilities (100+ HM) with multiple Top ranks in Public Audit Contests and completing multiple private audits and collaborative audits engagements.

Protocol Summary

Quake Cash is a decentralized, gamified yield protocol where users deposit stablecoins to earn high yield, per-second interest, with withdrawals unlockable after a short delay and risks.

Commit: quake-cash-contracts - @ 7e40c8733061f0d65a4f2806fd80bf1b4f464a64

Repo:

- QuakeCash.sol
- QuakeYield.sol

Deployment Chain(s):

- Base

Summary of Findings

Identifier	Title	Severity	Mitigated
[L-01]	Cannot revert boosted referral code back to default offset	Low	Fixed
[L-02]	Boost index 0 unintentionally grants a +1 referral bonus	Low	Acknowledged
[L-03]	Token decimals hard-coded to 6 causes wrong normalization for 18-dec tokens(DAI)	Low	Acknowledged
[I-01]	Redundant DEFAULT_ADMIN_ROLE declaration in Roles library	Informational	Acknowledged
[I-02]	Stablecoin address is immutable and cannot be updated post-deploy	Informational	Acknowledged
[I-03]	Rounding down in calculations causes minor wei loss for users	Informational	Acknowledged
[I-04]	Unused internal getReferrer(address) reduces clarity and utility	Informational	Fixed

Findings

[L-01] - Cannot revert boosted referral code back to default offset

Description

`_setBoostRate` forbids setting index == 1. If a code is boosted to a higher offset (e.g., +2), then later it cannot be set back to the default referral offset 1 using this function. This makes boosts effectively irreversible through `_setBoostRate` and conflicts with the default initialization where new codes start at 1.

Recommendation

function `_setDelayTimeSeconds` check

```
- require(index != 1, "Boost rate must be greater than default offset");
+ require(index >= 1, "Boost rate must be greater than default offset");
```

[L-02] - Boost index 0 unintentionally grants a +1 referral bonus

Description

`_setBoostRate` allows setting index = 0 (it only rejects index == 1). However, `_computeReferralOffset` assigns a minimum offset of `1` for any non-disabled referralCode, regardless of a stored boost of 0. This means setting a code’s boost to 0 (intending “no boost”, if not done through `_disableReferralRate`) will still grant a +1 boost and pay referrer rewards.

```
    function _setBoostRate(uint256 code, uint8 index) public
onlyRole(Roles.OPERATOR_ROLE) {
        require(code < nextCode, "Code does not exist");
        require(index != 1, "Cannot set boost rate to default referral rate offset");
        referralCodeRateBoostIndex[code] = index;
        emit BoostRateSet(code, index);
    }

    function _computeReferralOffset(uint256 rateSetIndex, uint16 baseRateIndex,
uint256 referralCode) internal returns (uint16 referralOffset) {

...existing code...

@> referralOffset = 1;

...existing code...
```

Recommendation

function `_setDelayTimeSeconds` check

```
- require(index != 1, "Boost rate must be greater than default offset");
+ require(index >= 1, "Boost rate must be greater than default offset");
```

[L-03] - Token decimals hard-coded to 6 causes wrong normalization for 18-dec tokens (DAI)

Description

The contract assumes a 6-decimal stablecoin via `TOKEN_DECIMALS = 1e6` and `WAD_TOKEN_QUOTIENT = 1e12`. `normalizeTokenAmount` multiplies by 1e12 unconditionally, which is correct for 6-dec tokens (USDC) but wrong for 18-dec tokens (DAI). With DAI, `normalizeTokenAmount(amount)` becomes amount * 1e12, producing 1e30-scale values and cascading incorrect math across deposits, indices, and limits.

Recommendation

Either don't use DAI stablecoin or fetch the decimals of tokens in dynamic way and adjust normalization/denormalization.

[I-01] - Redundant DEFAULT_ADMIN_ROLE declaration in Roles library

Description

The contract imports `AccessControl` which already defines `DEFAULT_ADMIN_ROLE = 0x00`. Declaring it again in the Roles library is redundant and unnecessary. The contract should use the existing `DEFAULT_ADMIN_ROLE` from OpenZeppelin's AccessControl instead of importing it from Roles. This simplifies the code and removes duplication.

Recommendation

- Remove `DEFAULT_ADMIN_ROLE` from the Roles library
- Use the built-in `DEFAULT_ADMIN_ROLE` from AccessControl
- Update the `constructor` to use `_grantRole(DEFAULT_ADMIN_ROLE, adminAddress);`

[I-02] - Stablecoin address is immutable and cannot be updated post-deploy

Description

The contract fixes the stablecoin at deployment via `stableToken` (declared immutable) and provides no admin function to change it. If the stable coin needs to be needs to be changed for some reason then the protocol must redeploy and migrate users. This is acceptable as a design choice, but limits operational flexibility.

Recommendation

```
IERC20 public stableToken; // Not immutable

function setStableToken(address _newToken) external onlyRole(Roles.OPERATOR_ROLE) {
```

```
    require(_newToken != address(0), "Invalid address");
    stableToken = IERC20(_newToken);
}
```

[I-03] - Rounding down in calculations causes minor wei loss for users

Description

Due to floor operations in scaling/denormalization and fixed-point math, users may receive slightly less than expected. For example, a 100e6 deposit with 30% target yield returns 129.999948e6 instead of 130e6. This is a minor precision loss (few wei) from rounding down during conversions between different precision formats (RAY, WAD, token decimals).

[I-04] - Unused internal getReferrer(address) reduces clarity and utility

Description

The internal function `getReferrer(address account)` is unused and duplicates logic already derivable via `referralCodes[account]` -> `codeToReferrer[code]`, while an external overload `getReferrer(uint256 code)` already exists. To avoid dead code and developer confusion, either remove the unused internal function, or make it callable for consumers by changing visibility to public/external.

Disclaimer

The team does not provide guarantees relating to the absolute security of the project as the ability to detect all potential vulnerabilities without the ability to formally prove correctness is near impossible. Usage of all smart contract software is at the respective users’ sole risk and is the users’ responsibility.