SUPREMACY

Smart Contract Security Audit Report

Prepared for SpinUp

Prepared by Supremacy

October 09, 2025

Contents

1 Introduction

Given the opportunity to review the design document and related codebase of the SpinUp, we outline in the report our systematic approach to evaluate potential security issues in the smart contract(s) implementation, and provide additional suggestions or recommendations for improvement. Our results show that the given version of smart contracts can be further improved due to the presence of several issues related to either security or performance. This document outlines our audit results.

1.1 About Client

SpinUp is a DeFi ecosystem built on HyperEVM, bringing together liquid staking, a Meme Launchpad, and a Meme DEX, fueling the perpetual evolution of the meme economy. Designed for the next generation of on-chain markets, SpinUp connects liquidity, creativity, and performance under one unified system.

Item	Description
Client	SpinUp
Type	Smart Contract
Languages	Solidity
Platform	EVM-compatible

1.2 Audit Scope

In the following, we show the Git repository of reviewed file and the commit hash used in this security audit:

Version	Repository	Commit Hash
1	Hyperpie	ba87f502f42df10a9275435a36ae78e6da61c2cf
2	Hyperpie	392ba80072507b9c51e56ea876da4d09903ce540
3	Hyperpie	3a0891c58890832b9b0c723ae52728fea576fab2
4	Hyperpie	ee9971758acbad1fdc9a3c5977ce803e0c08a0c4
5	Hyperpie	e609e58665f390a05beeb8933e8a74b9114235a5
6	Hyperpie	e8f30164fc4a6f4565ad18d6c6ad3ec5e62e9a73
7	Hyperpie	815f7526ac0cd44de55670aaf17332a98957b230
8	Hyperpie	82f07c2093c16a850189b92cfef530452997f63a
9	Hyperpie	0f8400ab3495d843ca859255b32399c6a0d15755

1.3 Changelogs

Version	Date	Description
0.1	March 02, 2025	Initial Draft
1.0	March 22, 2025	Final Release
1.1	June 24, 2025	Post-Final Release #1
1.2	July 01, 2025	Post-Final Release #2
1.3	July 08, 2025	Post-Final Release #3
1.4	August 17, 2025	Post-Final Release #4

1.5	August 23, 2025	Post-Final Release #5
1.6	October 09, 2025	Post-Final Release #6

1.4 About Us

Supremacy is a leading blockchain security firm, composed of industry hackers and academic researchers, provide top-notch security solutions through our technology precipitation and innovative research.

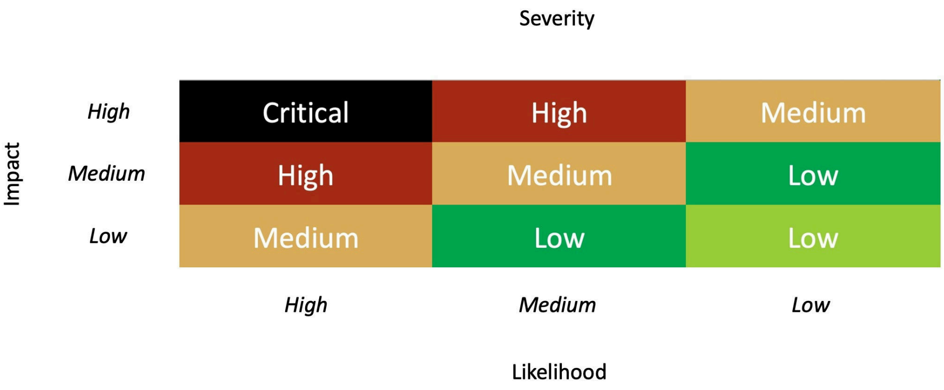
We are reachable at X (https://x.com/SupremacyHQ), or Email (contact@supremacy.email).

1.5 Terminology

For the purpose of this assessment, we adopt the following terminology. To classify the severity of our findings, we determine the likelihood and impact (according to the CVSS risk rating methodology).

- Likelihood represents the likelihood of a finding to be triggered or exploited in practice
- Impact specifies the technical and business-related consequences of a finding
- Severity is derived based on the likelihood and the impact

We categorize the findings into four distinct categories, depending on their severity. These severities are derived from the likelihood and the impact using the following table, following a standard risk assessment procedure.



Severity

Impact

	High	Medium	Low
High	Critical	High	Medium
Medium	High	Medium	Low
Low	Medium	Low	Low

Likelihood

As seen in the table above, findings that have both a high likelihood and a high impact are classified as critical. Intuitively, such findings are likely to be triggered and cause significant disruption. Overall, the severity correlates with the associated risk. However, every finding's risk should always be closely checked, regardless of severity.

2 Findings

The table below summarizes the findings of the audit, including status and severity details.

ID	Severity	Description	Status
1	Low	Insufficient Approval Check	Confirmed
2	Low	Withdrawal Buffer Management	Confirmed
3	Low	Unbounded Fee Percentages	Fixed
4	Informational	Potential Exchange Rate Manipulation	Acknowledged
5	Informational	Lack of Minimum Deposit Check	Fixed
6	Informational	Follow Check-Effects-Interactions Pattern	Acknowledged
7	Informational	Potential loss of precision	Undetermined
8	Informational	Lack of event records	Undetermined

2.1 Low

1. Insufficient Approval Check [Low]

Severity: Low Likelihood: Low Impact: Low

Status: Confirmed

Description

The `queueWithdrawal()` function uses `safeTransferFrom` to move `mHYPE` but doesn’t explicitly check if the user has approved the contract to spend the required amount. If the user hasn’t approved enough `mHYPE`, the transaction will revert unnecessarily, wasting gas and degrading user experience.

```
    /*//////////////////////////////////////////////////////////////
                            EXTERNAL FUNCTIONS
    //////////////////////////////////////////////////////////////*/
    function queueWithdrawal(uint256 mHYPEAmount) external whenNotPaused nonReentrant {
        if (mHYPEAmount == 0) revert InvalidAmount();

        address mHYPE = hyperpieConfig.getAddress(HyperpieConstants.MHYPE_TOKEN);
        IERC20 mHYPEToken = IERC20(mHYPE);
        if (mHYPEToken.balanceOf(msg.sender) < mHYPEAmount) {
            revert InsufficientBalance();
        }

        // Transfer mHYPE tokens first
        mHYPEToken.safeTransferFrom(msg.sender, address(this), mHYPEAmount);

        uint256 hypeAmount = (mHYPEAmount * ImHYPE(mHYPE).exchangeRateToUnderlying()) / 1 ether;
        uint256 availableToWithdraw = getAvailableToWithdraw();

        // Increment the withdrawRequestId
        withdrawRequestId++;

        WithdrawalRequest memory request = WithdrawalRequest({
            owner: msg.sender,
            requestId: withdrawRequestId,
            mHYPEAmount: mHYPEAmount,
            hypeAmount: hypeAmount,
            requestTime: block.timestamp
        });

        bool queued = false;
        if (hypeAmount > availableToWithdraw) {
            // Queue the withdrawal
            claimReserve += availableToWithdraw;
            withdrawQueue.queuedWithdrawFilled += availableToWithdraw;
            withdrawQueue.queuedWithdrawToFill += hypeAmount;

            bytes32 withdrawHash = keccak256(abi.encode(request, msg.sender));
            withdrawQueued[withdrawHash].queued = true;
            withdrawQueued[withdrawHash].fillAt = withdrawQueue.queuedWithdrawToFill;
            queued = true;
    } else {
      // For instant withdrawals from buffer
      claimReserve += hypeAmount;
    }

    withdrawRequests[msg.sender].push(request);

    emit WithdrawalQueued(msg.sender, withdrawRequestId, mHYPEAmount,
hypeAmount, queued, availableToWithdraw);
  }
```

HyperpieWithdrawManager.sol

Recommendation

Consider add an explicit approval check.

```
    // Transfer mHYPE tokens first
    if (mHYPEToken.allowance(msg.sender, address(this)) < mHYPEAmount) {
     revert InsufficientAllowance();
    }

    mHYPEToken.safeTransferFrom(msg.sender, address(this), mHYPEAmount);
```

HyperpieWithdrawManager.sol

2. Withdrawal Buffer Management [Low]

Severity: Low Likelihood: Low Impact: Low

Status: Confirmed

Description

In stakeHype(), the contract transfers HYPE to the withdrawManager to cover a deficit without verifying the transfer's success beyond a basic success check. There's no assurance that the withdrawManager can handle or process the funds correctly. If the withdrawManager is misconfigured, compromised, or fails to receive funds (e.g., due to a revert), the HYPE could be lost or stuck.

```
  /*//////////////////////////////////////////////////////////////
              WRITE FUNCTIONS
  //////////////////////////////////////////////////////////////*/
  /// @notice Deposits HYPE and mints mHype, note the current implementation
is consider only for native
  /// token deposits
  /// @param minMHYPETokenAmount Minimum amount of mHype to accept
  /// @param referral Address of the referrer
  function stakeHype(uint256 minMHYPETokenAmount, address referral) external
payable whenNotPaused nonReentrant {
    if (msg.value == 0 || msg.value < minAmountToDeposit) {
      revert InvalidAmountToDeposit();
    }

    uint256 mHYPEAmount = getMHYPEToMint(msg.value);
    if (mHYPEAmount < minMHYPETokenAmount) {
      revert MinimumAmountToReceiveNotMet();
```

```
        }

        // Mint mHype - this will set the stake time
        address mHYPE = hyperpieConfig.getAddress(HyperpieConstants.MHYPE_TOKEN);
        IMintableERC20(mHYPE).mint(msg.sender, mHYPEAmount);

        // Check withdraw buffer deficit and fill if needed
        address withdrawManagerAddr = hyperpieConfig.getAddress(HyperpieConstants.HYPERPIE_WITHDRAW_MANAGER);
        IHyperpieWithdrawManager withdrawManager = IHyperpieWithdrawManager(withdrawManagerAddr);
        uint256 withdrawDeficit = withdrawManager.getWithdrawDeficit();

        if (withdrawDeficit > 0) {
            uint256 amountToBuffer = withdrawDeficit > msg.value ? msg.value : withdrawDeficit;
            // Send HYPE directly to the withdraw manager to fill the buffer
            (bool success,) = withdrawManagerAddr.call{ value: amountToBuffer }("");
            if (!success) revert TransferFailed();
        }

        emit AssetDeposited(msg.sender, HyperpieConstants.PLATFORM_TOKEN_ADDRESS, msg.value, mHYPEAmount, referral);
    }
```

HyperpieStaking.sol

Recommendation

Always validate the `withdrawManagerAddr` balance after transferring funds.

```
        if (withdrawDeficit > 0) {
            uint256 amountToBuffer = withdrawDeficit > msg.value ? msg.value : withdrawDeficit;
            // Send HYPE directly to the withdraw manager to fill the buffer
            uint256 balanceBefore = withdrawManagerAddr.balance;
            (bool success,) = withdrawManagerAddr.call{ value: amountToBuffer }("");
            require(success && withdrawManagerAddr.balance >= balanceBefore + amountToBuffer);

```

HyperpieStaking.sol

3. Unbounded Fee Percentages [Low]

`Severity: Low` `Likelihood: Low` `Impact: Low`

`Status: Fixed`

Description

The `PoolFees::collectFee()` function retrieves fee percentages from `hyperpieConfig` without upper bounds or sanity checks (except for the sum exceeding `_amount`). If `hyperpieConfig` returns excessively high values (e.g., `FEE_TEAM_COLLECTOR > DENOMINATOR`), the contract could: Revert due to `InvalidFees` if the sum exceeds _`amount`. Misbehave if `hyperpieConfig` is compromised, allowing an malicious actor to drain funds by setting a single fee to 100% or more.

Recommendation

Revise the code logic accordingly.

Feedback: Fixed in `90f08cf`.

2.2 Informational

4. Potential Exchange Rate Manipulation [Informational]

`Status: Acknowledged`

Description

The `getMHYPEToMint()` function calculates `mHYPE` to mint based on an exchange rate from `ImHYPE(mHYPE).exchangeRateToUnderlying()`. If this rate is manipulable (e.g., by an external oracle) or not updated correctly, it could distort the `mHYPE` issuance. Users could receive more or fewer `mHYPE` tokens than expected, leading to unexpected results.

```
    /*//////////////////////////////////////////////////////////////
                            VIEW FUNCTIONS
    //////////////////////////////////////////////////////////////*/
    /// @notice Calculates the amount of mHype to mint for HYPE deposit
    /// @param amount Amount of HYPE being deposited
    /// @return mHYPEAmount Amount of mHype to mint
    function getMHYPEToMint(uint256 amount) public view returns (uint256 mHYPEAmount) {
        address mHYPE = hyperpieConfig.getAddress(HyperpieConstants.MHYPE_TOKEN);
        uint256 rate = ImHYPE(mHYPE).exchangeRateToUnderlying();
        return (amount * 1 ether) / rate;
    }
```

HyperpieStaking.sol

Recommendation

Consider add bounds checking to prevent extreme rate values.

5. Lack of Minimum Deposit Check [Informational]

`Status: Fixed`

Description

While `stakeHype` checks `msg.value` against `minAmountToDeposit`, there's no validation in `setMinAmountToDeposit` to prevent setting it to zero. A zero minimum allows spam deposits, potentially clogging the contract with low-value transactions and affecting performance.

```
    /// @notice Sets the minimum amount required for deposits
    /// @param _minAmountToDeposit New minimum deposit amount
    function setMinAmountToDeposit(uint256 _minAmountToDeposit) external onlyDefaultAdmin {
        minAmountToDeposit = _minAmountToDeposit;
        emit MinAmountToDepositUpdated(_minAmountToDeposit);
    }
```

HyperpieStaking.sol

Recommendation

Consider add a check in `setMinAmountToDeposit()`.

Feedback: Fixed in `392ba80`.

6. Follow Check-Effects-Interactions Pattern [Informational]

`Status: Acknowledged`

Description

In the `HyperpieStaking::stakeHype()`, the minting of `mHYPE` does not follow the Check-Effects-Interactions Pattern.

```
    /*//////////////////////////////////////////////////////////////
                            WRITE FUNCTIONS
    //////////////////////////////////////////////////////////////*/
    /// @notice Deposits HYPE and mints mHype, note the current implementation is consider only for native
    /// token deposits
    /// @param minMHYPETokenAmount Minimum amount of mHype to accept
    /// @param referral Address of the referrer
    function stakeHype(uint256 minMHYPETokenAmount, address referral) external payable whenNotPaused nonReentrant {
        if (msg.value == 0 || msg.value < minAmountToDeposit) {
            revert InvalidAmountToDeposit();
        }

        uint256 mHYPEAmount = getMHYPEToMint(msg.value);
        if (mHYPEAmount < minMHYPETokenAmount) {
            revert MinimumAmountToReceiveNotMet();
        }

        // Mint mHype - this will set the stake time
        address mHYPE = hyperpieConfig.getAddress(HyperpieConstants.MHYPE_TOKEN);
        IMintableERC20(mHYPE).mint(msg.sender, mHYPEAmount);

        // Check withdraw buffer deficit and fill if needed
        address withdrawManagerAddr = hyperpieConfig.getAddress(HyperpieConstants.HYPERPIE_WITHDRAW_MANAGER);
        IHyperpieWithdrawManager withdrawManager = IHyperpieWithdrawManager(withdrawManagerAddr);
        uint256 withdrawDeficit = withdrawManager.getWithdrawDeficit();

        if (withdrawDeficit > 0) {
            uint256 amountToBuffer = withdrawDeficit > msg.value ? msg.value : withdrawDeficit;
            // Send HYPE directly to the withdraw manager to fill the buffer
            (bool success,) = withdrawManagerAddr.call{ value: amountToBuffer }("");
            if (!success) revert TransferFailed();
        }

        emit AssetDeposited(msg.sender, HyperpieConstants.PLATFORM_TOKEN_ADDRESS, msg.value, mHYPEAmount, referral);
    }
```

HyperpieStaking.sol

Recommendation

Revise the code logic accordingly.

7. Potential loss of precision [Informational]

`Status: Undetermined`

Description

The `_getMHYPEToInflate` internal function calculates the amount of `mHYPE` tokens to mint for fee accrual using integer division `(numerator / denominator)`. Since integer division rounds down, this can lead to precision loss, especially when `feeValue` or `totalSupply` are small. This may result in slightly lower fees than intended, affecting the protocol's revenue.

Recommendation

Revise the code logic accordingly.

8. Lack of event records [Informational]

`Status: Undetermined`

Description

The `accrueFee()` function in the `HyperpieStaking` contract updates the `lastWeeklyFeeAccrualTimestamp` state variable to track the timestamp of the last fee accrual, ensuring fees are accrued only once per week. However, no event is emitted to log this state change. This omission reduces the contract's transparency and hinders off-chain monitoring tools or users from tracking fee accrual timing effectively.

Recommendation

Revise the code logic accordingly.

3 Disclaimer

This security audit report does not constitute investment advice or a personal recommendation. It does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Any entity should not rely on this report in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This security audit report is not an endorsement of any particular project or team, and the report does not guarantee the security of any particular project. This audit does not give any warranties on discovering all security issues of the smart contracts, i.e., the evaluation result does not guarantee the nonexistence of any further findings of security issues, also cannot make guarantees about any additional code added to the assessed project after the audit version. As one audit-based assessment cannot be considered comprehensive, we always recommend proceeding with independent audits and a public bug bounty program to ensure the security of smart contract(s). Unless explicitly specified, the security of the language itself (e.g., the solidity language), the underlying compiling toolchain and the computing infrastructure are out of the scope.