freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

HyprBurnTier

30/08/2025

TOKEN OVERVIEW

Fees

- Buy fees: N/A
- Sell fees: N/A

Fees privileges

- Not available

Ownership

- Owned

Minting

- Mint function not detected

Max Tx Amount / Max Wallet Amount

- Not available

Blacklist

- Blacklist function not detected

Other privileges

- Not available

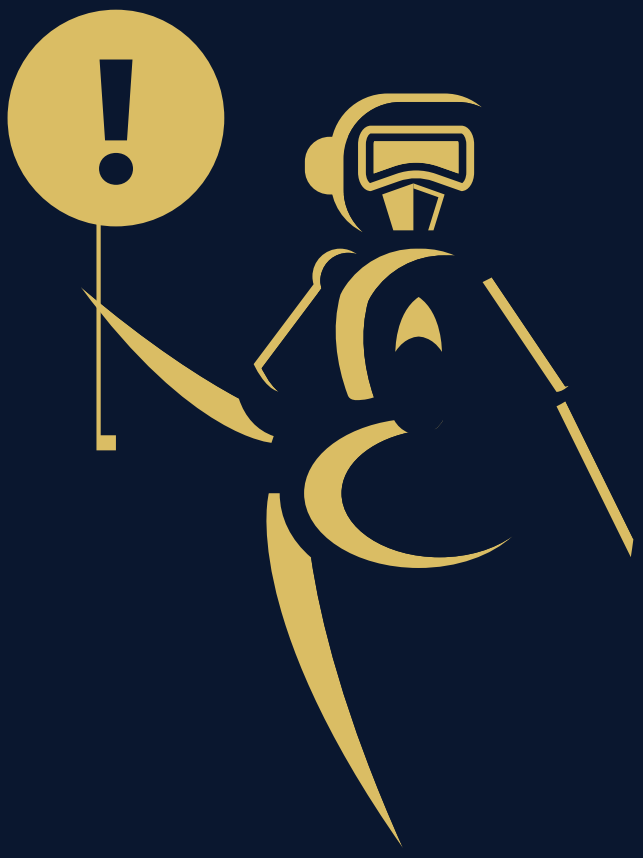
TABLE OF CONTENTS

DISCLAIMER

The information provided on this analysis document is only for general information and should not be used as a reason to invest.

FreshCoins Team will take no payment for manipulating the results of this audit.

The score and the result will stay on this project page information on our website https://freshcoins.io
FreshCoins Team does not guarantees that a project will not sell off team supply, or any other scam strategy (RUG or Honeypot etc)



INTRODUCTION

FreshCoins (Consultant) was contracted by HYPR (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0xe34713C563dce2FA2963dF8863e01b958DBBF68F

Network: Sepolia (Testnet)

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 30/05/2025

WEBSITE DIAGNOSTIC

https://hypr.fund/

0-49 50-89 90-100

Performance | Accessibility | Best Practices | SEO | Progressive Web App

Socials

X (Twitter)

https://x.com/hyprfund

Telegram

https://t.me/hyprfund

AUDIT OVERVIEW

88

Security Score

94

Static Scan
Automatic scanning for common vulnerabilities

92

ERC Scan
Automatic checks for ERC's conformance

0 High

2 Medium

0 Low

4 Optimizations

0 Informational

No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Low
3	Front running	Low
4	Timestamp dependence	Passed
5	Integer Overflow and Underflow	Passed
6	Reverted DoS	Passed
7	DoS with block gas limit	Passed
8	Methods execution permissions	Passed
9	Exchange rate impact	Passed
10	Malicious Event	Passed
11	Scoping and Declarations	Passed
12	Uninitialized storage pointers	Passed
13	Design Logic	Passed
14	Safe Zeppelin module	Passed

OWNER PRIVILEGES & FINDINGS

No Reentrancy Protection

The burn and burnFor functions call token.transferFrom (via _safeTransferFrom) before updating state in updateTier. Non-standard tokens (e.g., ERC777) with callbacks could trigger reentrancy, potentially manipulating totalBurned or currentTier before the state is finalized.

```
function burn(uint256 amount) public {
    require(amount > 0, "amount=0");
    _requireAllowance(msg.sender, amount);
    _safeTransferFrom(msg.sender, deadAddress, amount); // External call before state update
    updateTier(amount, msg.sender);
    emit Burned(msg.sender, amount, burnMode);
}

function burnFor(uint256 amount, address destinationAccount) public {
    require(amount > 0, "amount=0");
    _requireAllowance(msg.sender, amount);
    _safeTransferFrom(msg.sender, deadAddress, amount); // External call before state update
    updateTier(amount, destinationAccount);
    emit BurnedFor(msg.sender, destinationAccount, amount, burnMode);
}

function _safeTransferFrom(address from, address to, uint256 amount) internal {
    bool ok = token.transferFrom(from, to, amount);
    require(ok, "transferFrom failed");
}
```

Add a reentrancy guard (e.g., OpenZeppelin's ReentrancyGuard) to burn and burnFor.

No Restriction on Setting Thresholds to Zero

The setTierThresholdsRaw and setTierThresholdsTokens functions allow setting both thresholds to 0 (as long as hypr >= supr). This would make all addresses (even those with zero burns) qualify as Hypr tier, since totalBurned starts at 0 and comparisons would always pass for b >= 0. This could trivialize the tier system unintentionally

```
function setTierThresholdsRaw(uint256 newSuprRaw, uint256 newHyprRaw) external onlyOwner {
    require(newHyprRaw >= newSuprRaw, "hypr < supr");
    suprThresholdRaw = newSuprRaw;
    hyprThresholdRaw = newHyprRaw;
    emit TierThresholdsUpdated(newSuprRaw, newHyprRaw);
}
```

```
function setTierThresholdsTokens(uint256 newSuprTokens, uint256 newHyprTokens) external onlyOwner {
    uint8 dec = _tokenDecimals();
    uint256 suprRaw = newSuprTokens * (10 ** dec);
    uint256 hyprRaw = newHyprTokens * (10 ** dec);
    require(hyprRaw >= suprRaw, "hypr < supr");
    suprThresholdRaw = suprRaw;
    hyprThresholdRaw = hyprRaw;
    emit TierThresholdsUpdated(suprRaw, hyprRaw);
}
```

Add a require check to ensure newSuprRaw > 0 (or make it optional via config). Alternatively, document this edge case and its implications.

Centralization Risk Due to Owner Privileges

The owner has broad control (e.g., changing burn mode, thresholds, dead address), which is typical for Ownable contracts but introduces risks if the owner is compromised or acts maliciously. For example, lowering thresholds could devalue existing burns, or switching modes could break functionality. No timelocks or multi-sig are enforced.

Consider adding a timelock for critical owner functions or migrating to a DAO/multi-sig ownership post-deployment. Renounce ownership if no further changes are needed. This is a general best practice for production contracts.

Lack of Reentrancy Protection

While the contract's functions (e.g., burn, burnFrom) do not make external calls after state changes that could enable reentrancy, if the token has non-standard behavior (e.g., ERC777-style hooks), it could theoretically allow reentry during transferFrom or burnFrom. Standard ERC20 tokens are safe, but this assumes compliance.

Add the ReentrancyGuard modifier from OpenZeppelin to critical functions like burn for defense-in-depth, especially if supporting non-standard tokens. Test with tokens that have hooks.

Gas Optimization Opportunity in updateTier

The updateTier function uses multiple if-else statements to determine the tier, which can be optimized to reduce gas usage by minimizing comparisons and state writes, especially since tier updates are frequent.

```
function updateTier(uint256 amount, address account) private {
  totalBurned[account] += amount;
  if (totalBurned[account] >= hyprThresholdRaw) currentTier[account] = Tier.Hypr;
  else if (totalBurned[account] >= suprThresholdRaw) currentTier[account] = Tier.Supr;
  else currentTier[account] = Tier.Commonr;
  emit TierUpdated(account, currentTier[account]);
}
```

Optimize the logic to use a single comparison and only update state/emit events if the tier changes

- **No Bulk Burn Functionality**

  Users can only burn in single transactions via burn or burnWithPermit. For large or multiple burns, this increases gas costs and user friction, though it's not a security issue.

  Add a burnBatch function accepting an array of amounts if frequent bulk operations are expected. This is an optimization suggestion.

Workflow

The HyprBurnTier contract is a Solidity smart contract (version ^0.8.20) designed to manage token burning for a specified ERC20 token, assigning users to one of three tiers (Commonr, Supr, Hypr) based on the cumulative amount of tokens they burn. It supports burning tokens by transferring them to a dead address and allows the contract owner to configure tier thresholds.

Recommendation:

The team should carefully manage the private keys of the owner's account. We strongly recommend a powerful security mechanism that will prevent a single user from accessing the contract admin functions. The risk can be prevented by temporarily locking the contract or renouncing ownership.

CONCLUSION AND ANALYSIS

Smart Contracts within the scope were manually reviewed and analyzed with static tools.

Audit report overview contains all found security vulnerabilities and other issues in the reviewed code.

Found no HIGH issues during the first review.

TOKEN DETAILS

Details

Buy fees:	N/A
Sell fees:	N/A
Max TX:	N/A
Max Sell:	N/A

Honeypot Risk

Ownership:	Owned
Blacklist:	Not detected
Modify Max TX:	Not detected
Modify Max Sell:	Not detected
Disable Trading:	Not detected

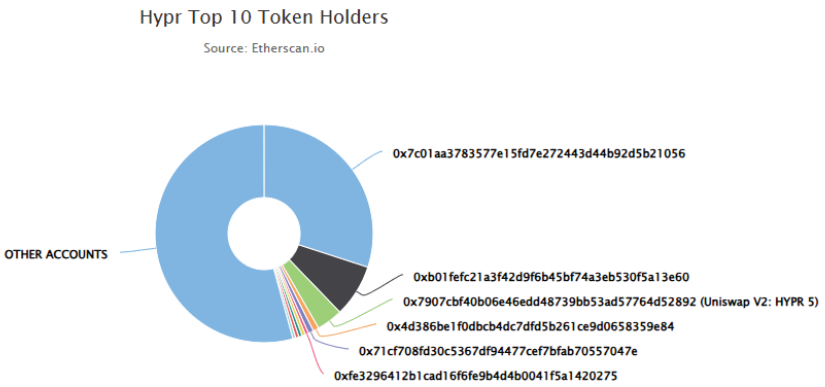
Rug Pull Risk

Liquidity:	N/A
Holders:	Clean

HYPR TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS

The top 10 holders collectively own 45.75% (457,476,373.73 Tokens) of Hypr

Token Total Supply: 1,000,000,000.00 Token | Total Token Holders: 1,891



(A total of 457,476,373.73 tokens held by the top 10 accounts from the total supply of 1,000,000,000.00 token)

Rank	Address	Quantity (Token)	Percentage
1	0x7C01AA37...2d5b21056	300,000,000	30.0000%
2	0xb01fEFC2...0f5a13E60	78,000,000	7.8000%
3	Uniswap V2: HYPR 5	38,826,230.305356	3.8826%
4	0x4D386bE1...658359e84	8,579,554.426610385	0.8580%
5	0x71Cf708f...70557047E	7,900,000	0.7900%
6	0xfe329641...5A1420275	5,000,000	0.5000%
7	0x0852628e...ce86937a4	5,000,000	0.5000%
8	0xFFF03526...719b8cfe2	5,000,000	0.5000%
9	0x52CDd240...3F5AeB44B	4,970,000	0.4970%
10	0xaa2572e7...a405710ad	4,200,589	0.4201%

TECHNICAL DISCLAIMER

Smart contracts are deployed and executed on the blockchain platform. The platform, its programming language, and other software related to the smart contract can have its vulnerabilities that can lead to hacks. The audit can't guarantee the explicit security of the audited project / smart contract.