freshcoins

SMART CONTRACT CODE REVIEW AND SECURITY ANALYSIS REPORT

HyprDropDore

NFT Smart Contract

08/09/2025

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

1 DISCLAIMER

2 INTRODUCTION

3 WEBSITE + SOCIALS

4-5 AUDIT OVERVIEW

6-9 OWNER PRIVILEGES & FINDINGS

10 CONCLUSION AND ANALYSIS

11 TOKEN DETAILS

12 HYPR TOKEN ANALYTICS &
TOP 10 TOKEN HOLDERS

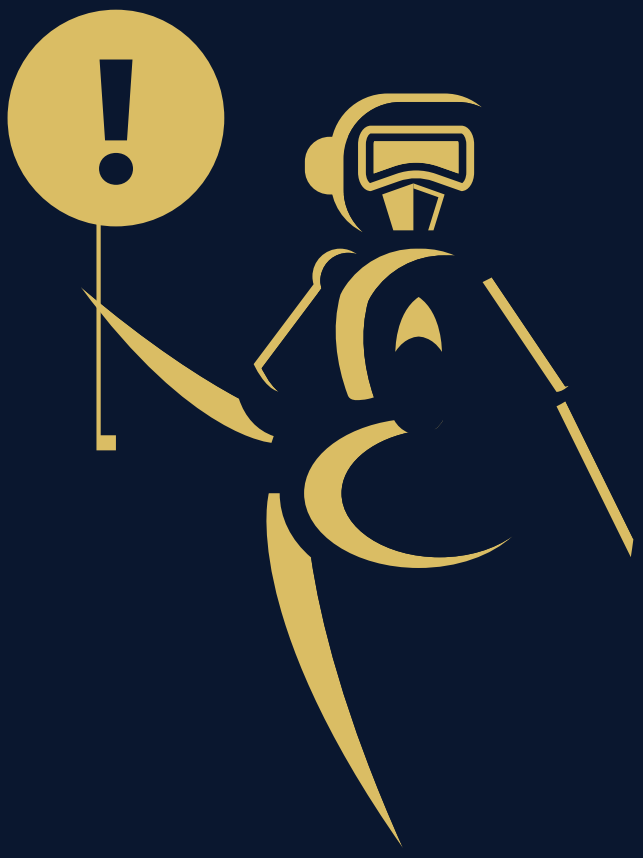
13 TECHNICAL DISCLAIMER

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 HyprDropDore (Customer) to conduct a Smart Contract Code Review and Security Analysis.

0x3Ce64bc08EAd93636541048DDa0D459af998d3B9

Network: Sepolia (Testnet)

This report presents the findings of the security assessment of Customer's smart contract and its code review conducted on 09/08/2025

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

72

Security Score

92

Static Scan
Automatic scanning for common vulnerabilities

88

ERC Scan
Automatic checks for ERC's conformance

- 0 High
- 4 Medium
- 0 Low
- 3 Optimizations
- 0 Informational

No.	Issue description	Checking Status
1	Compiler Errors / Warnings	Passed
2	Reentrancy and Cross-function	Passed
3	Front running	Passed
4	Timestamp dependence	Low
5	Integer Overflow and Underflow	Passed
6	Reverted DoS	Low
7	DoS with block gas limit	Low
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	Low
14	Safe Zeppelin module	Passed

OWNER PRIVILEGES & FINDINGS

Inconsistent Phase Duration Logic

The PHASE1_TOTAL_DURATION is set to 9999 seconds, which is inconsistent with the sum of HYPR_ONLY_DURATION (15 minutes = 900 seconds) and HYPR_SUPR_DURATION (15 minutes = 900 seconds). This implies the ALL_TIERS phase starts after 9999 seconds (about 2.78 hours), not after 30 minutes as suggested by the phase durations. Additionally, the phase1MaxRaise applies to both HYPR_ONLY and HYPR_SUPR phases, but the code allows ALL_TIERS investments to exceed phase1MaxRaise without clear documentation, which could confuse users or developers.

```
// Timing configuration (in seconds)
uint256 public constant HYPR_ONLY_DURATION = 15 * 60;     // 15 minutes
uint256 public constant HYPR_SUPR_DURATION = 15 * 60;     // 15 minutes
uint256 public constant PHASE1_TOTAL_DURATION = 9999;
```

Clarify the intended duration of Phase 1 in the code and documentation. If Phase 1 is meant to be 30 minutes, set PHASE1_TOTAL_DURATION = HYPR_ONLY_DURATION + HYPR_SUPR_DURATION (1800 seconds).

Lack of Input Validation for Tier Strings

The contract relies on the IHyprBurn.getUserTier function to return tier strings (Hypr, Supr, Commonr). However, there's no validation to handle unexpected or invalid tier strings. If getUserTier returns an unrecognized string (e.g., due to a misconfiguration or upgrade in the IHyprBurn contract), the contract's tier-based logic (isTierEligible, tierMaxInvestment, etc.) could behave unpredictably, potentially allowing ineligible users to invest or causing errors in allocation tracking.

Add a validation function to ensure the tier string is one of the expected values (Hypr, Supr, Commonr).

No Emergency Withdrawal Mechanism During Active Drop

The withdrawUSDT function can only be called after the drop has ended (dropConfig.ended = true). If an issue occurs during the drop (e.g., a bug, exploit, or external contract failure), the owner cannot withdraw funds to mitigate risks until the drop is manually ended. This could trap funds in the contract during an emergency.

```
function emergencyWithdraw(address to) external onlyOwner whenPaused {
    uint256 balance = usdtToken.balanceOf(address(this));
    require(balance > 0, "No USDT to withdraw");
    require(usdtToken.transfer(to, balance), "Transfer failed");
}
```

Add an emergency withdrawal function (e.g., emergencyWithdraw) that allows the owner to withdraw USDT even if the drop is active, possibly with additional safeguards (e.g., pausing the contract first).

No Check for USDT Allowance

The invest function calls usdtToken.transferFrom without explicitly checking if the user has approved sufficient USDT allowance for the contract. If the allowance is insufficient, the transaction will fail with a generic error from the USDT contract, which may confuse users.

```
require(
  usdtToken.allowance(msg.sender, address(this)) >= amount,
  "Insufficient USDT allowance"
);
require(
  usdtToken.transferFrom(msg.sender, address(this), amount),
  "USDT transfer failed"
);
```

Add a check for the USDT allowance before calling transferFrom.

Hardcoded NFT Name and Symbol

The NFT name ("DORE by LeTAO") and symbol ("DORE") are hardcoded in the constructor. If the project needs to rebrand or deploy a similar contract for a different NFT collection, the code must be modified. This reduces reusability and increases maintenance effort.

```
constructor(
  address _hyprBurnContract,
  address _usdtToken,
  string memory _name,
  string memory _symbol
) ERC721(_name, _symbol) Ownable(msg.sender) {
  // ... existing constructor logic
}
```

Allow the NFT name and symbol to be set during deployment via constructor parameters.

Potential Gas Limit Issues with Large Investments

The invest function loops through the number of NFTs to mint (nftCount = amount / dropConfig.nftPrice). If a user invests a large amount (e.g., enough to mint hundreds of NFTs), the loop could consume excessive gas, potentially hitting the block gas limit and causing the transaction to fail. This is particularly risky if gas prices are high or the network is congested.

```
uint256 constant MAX_NFTS_PER_TX = 50;
require(nftCount <= MAX_NFTS_PER_TX, "Exceeds max NFTs per transaction");
```

Add a maximum limit on nftCount per transaction (e.g., 50 NFTs) to prevent gas limit issues.

No Validation for Zero Address in Withdraw

The withdrawUSDT function allows the owner to specify a to address but doesn't check if it's the zero address (address(0)). Sending USDT to the zero address would burn the funds, making them unrecoverable.

```
function withdrawUSDT(address to) external onlyOwner {
    require(to != address(0), "Cannot withdraw to zero address");
    require(dropConfig.ended, "Drop must be ended");
    uint256 balance = usdtToken.balanceOf(address(this));
    require(balance > 0, "No USDT to withdraw");
    require(usdtToken.transfer(to, balance), "Transfer failed");
}
```

Add a check to prevent withdrawal to the zero address.

Workflow

The HyprDropDore smart contract is an ERC-721 NFT drop platform that allows users to invest USDT to mint NFTs in a tiered, phased system (Hypr, Supr, Commonr tiers). It integrates with an external IHyprBurn contract to verify user tiers and enforces investment limits per tier and phase, with a total raise cap and a Phase 1 cap. Users invest USDT, receive NFTs based on the amount (divided by NFT price), and the contract tracks investments, mints NFTs with metadata, and emits events. The owner can configure, activate, pause, or end the drop, and withdraw USDT after completion. Reentrancy protection and pausability ensure security

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

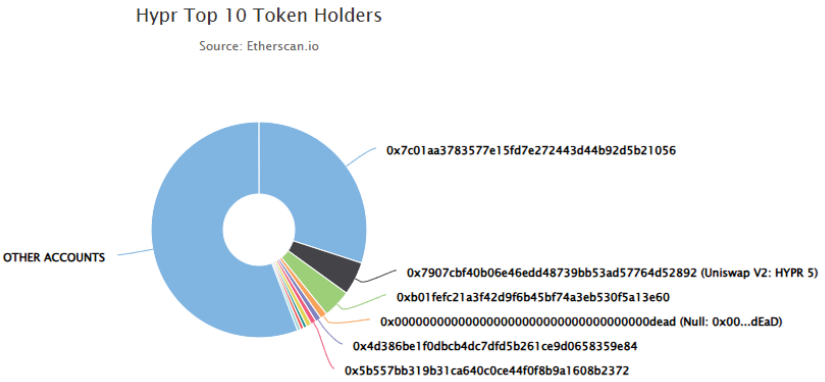
Rug Pull Risk

Liquidity:	N/A
Holders:	Clean

HYPR TOKEN ANALYTICS & TOP 10 TOKEN HOLDERS

The top 10 holders collectively own 44.20% (441,972,069.98 Tokens) of Hypr

Token Total Supply: 1,000,000,000.00 Token | Total Token Holders: 1,907



(A total of 441,972,069.98 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	Uniswap V2: HYPR 5	49,779,662.205303967	4.9780%
3	0xb01fEFC2...0f5a13E60	42,698,990.33	4.2699%
4	Null: 0x00...dEaD	10,463,259	1.0463%
5	0x4D386bE1...658359e84	8,579,543.444998304	0.8580%
6	0x5B557Bb3...1608B2372	8,000,615.0001	0.8001%
7	0x71Cf708f...70557047E	7,450,000	0.7450%
8	0xfe329641...5A1420275	5,000,000	0.5000%
9	0xFFF03526...719b8cfe2	5,000,000	0.5000%
10	0x7d56675a...Ac916D8BF	5,000,000	0.5000%

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.