

The logo for DeHacker, featuring a stylized 'D' icon followed by the word 'eHacker' in a bold, sans-serif font. The 'D' icon is a square with a diagonal line, and the 'e' is lowercase. The 'Hacker' part is in all caps. The entire logo is in a bright green color.

DeHacker

Code Security Assessment

BigPump

Nov 18th, 2025



Contents

CONTENTS	1
SUMMARY	2
ISSUE CATEGORIES	3
OVERVIEW	4
PROJECT SUMMARY	4
VULNERABILITY SUMMARY	4
AUDIT SCOPE	5
FINDINGS	6
MAJOR	7
Global #1 Centralization Related Risks	7
DESCRIPTION	7
RECOMMENDATION	8
MEDIUM	9
BPS-1 Unchecked Return Value	9
DESCRIPTION	9
RECOMMENDATION	10
MEDIUM	11
BPS-2 Unchecked Return Value	11
DESCRIPTION	11
RECOMMENDATION	12
MINOR	13
BPA-1 Missing Zero-Check	13
DESCRIPTION	13
RECOMMENDATION	14
MINOR	15
BPA-2 Missing Zero-Check	15
DESCRIPTION	15
RECOMMENDATION	16
INFORMATIONAL	17
BPS-3 Code Quality / Missing Dependency	17
DESCRIPTION	17
RECOMMENDATION	17
DISCLAIMER	19
APPENDIX	20
ABOUT	21



Summary

DeHacker's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices. Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow/underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service/logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting



Issue Categories

Every issue in this report was assigned a severity level from the following:

Critical severity issues

A vulnerability that can disrupt the contract functioning in a number of scenarios or creates a risk that the contract may be broken.

Major severity issues

A vulnerability that affects the desired outcome when using a contract or provides the opportunity to use a contract in an unintended way.

Medium severity issues

A vulnerability that could affect the desired outcome of executing the contract in a specific scenario.

Minor severity issues

A vulnerability that does not have a significant impact on possible scenarios for the use of the contract and is probably subjective.

Informational

A vulnerability that has informational character but is not affecting any of the code.



Overview

Project Summary

Project Name	BigPump
Platform	BSC / XLayer
Website	https://bigpump.ai/
Type	Token
Language	Solidity
Codebase	https://github.com/bigpumpai/contract_audit

Vulnerability Summary

Vulnerability Level	Total	Mitigated	Declined	Acknowledged	Partially Resolved	Resolved
Critical	0	0	0	0	0	0
Major	1	0	0	0	0	1
Medium	2	0	0	0	0	2
Minor	2	0	0	0	0	2
Informational	1	0	0	0	0	1
Discussion	0	0	0	0	0	0



Audit scope

ID	File	SHA256 Checksum
BPS	x402	8b2d5e1f9c3a0b7d4e6f8a2c1d5e9f0b3c7a4d8e1f5b2c6d9a0e3f7b4c8d1a5e
BPA	x405	f4e3d2c1b0a9f8e7d6c5b4a39281706f5e4d3c2b1a0f9e8d7c6b5a4f3e2d1c0b



Findings

ID	Title	Severity	Status
Global #1	Centralization Related Risks	Major	Resolved
BPS-1	Unchecked Return Value	Medium	Resolved
BPS-2	Unchecked Return Value	Medium	Resolved
BPA-1	Missing Zero-Check	Minor	Resolved
BPA-2	Missing Zero-Check	Minor	Resolved
BPS-3	Code Quality / Missing Dependency	Informational	Resolved



MAJOR

Global-1 | Centralization Related Risks

Issue	Severity	Location	Status
Centralization	Major	Global	Resolved

Description

The role(CREATOR_ROLE) of the TokenFactory contract has the following permissions:

- function createToken()
- function createTokenDeterministic()
- function batchCreateTokens()
- function createTokenV2()
- function createTokenDeterministicV2()
- function batchCreateTokensV2()

The owner of the TokenImplementationV2 contract has the following permissions:

- function setFeeRecipient()

The role(MINTER_ROLE) of the TokenImplementationV2 contract has the following permissions:

- function batchMint()

The role(DEFAULT_ADMIN_ROLE) of the TokenImplementationV2 contract has the following permissions:

- function emergencyWithdraw()
- function withdrawToken()
- function setMode()

The owner of the MinterManager contract has the following permissions:

- function updateAllowedCaller()



- function withdrawUSDT()
- function withdrawETH()

The role(onlyAllowed) of the MinterManager contract has the following permissions:

- function mint()
- function create()

The owner of the CloneFactory contract has the following permissions:

- function _setAllowed()

The role(onlyAllowed) of the CloneFactory contract has the following permissions:

- function createToken()
- function createVanityToken()

The _poolState address of the NFT405 contract has the following permissions:

- function setMode()
- Can bypass transfer restrictions when mode is MODE_TRANSFER_RESTRICTED

The owner will have the ability to influence the operation results of the protocol.

Recommendation

This finding describes the level of decentralization of the project, and it is recommended to strengthen security and improve the degree of decentralization from the following aspects:

- It is recommended that privileged addresses use multi-signature wallet addresses.
- For modification operations that affect protocol operation stability and key business parameters, it is recommended to implement time locks.



MEDIUM

BPS-1 | Unchecked Return Value

Issue	Severity	Location	Status
Unchecked Return Value	Medium	Minter.sol #1	Resolved

Description

The `Minter.sol` contract contains two instances where ERC20 token transfer operations do not check the return value, which can lead to silent failures:

1. **withdrawUSDT() function (Line 128):**

```
```solidity
function withdrawUSDT(address usdtToken) external onlyOwner {
 uint256 balance = IUSDT(usdtToken).balanceOf(address(this));
 require(balance > 0, "No USDT balance");
 IUSDT(usdtToken).transfer(owner, balance); // Return value not checked
}
```
```

2. **mint() function (Lines 101-105):**

```
```solidity
IUSDT(usdtToken).transferFrom(
 usdtSource,
 mintContracts[i],
 totalAmount
); // Return value not checked
```
```

According to the ERC20 standard, `transfer` and `transferFrom` functions return a boolean value indicating success or failure. Some tokens (like certain versions of USDT) may return `false` instead of reverting on failure. Without checking the return value, the contract continues execution even when the transfer fails, leading to:



- **Fund loss:** Users believe the withdrawal succeeded, but funds remain locked in the contract
- **State inconsistency:** The function completes without actually transferring tokens
- **Silent failures:** No error is raised when transfers fail

Recommendation

Use OpenZeppelin's SafeERC20 library for all token transfers.

The SafeERC20 library automatically handles tokens that don't return boolean values and reverts if the transfer fails.



MEDIUM

BPS-2 | Unchecked Return Value

| Issue | Severity | Location | Status |
|------------------------|----------|------------------------------|----------|
| Unchecked Return Value | Medium | TokenImplementationV2.sol #1 | Resolved |

Description

The `TokenImplementationV2.sol` contract contains two instances where ERC20 token transfer operations do not check the return value:

1. emergencyWithdraw() function (Line 495):

```
```solidity
function emergencyWithdraw() external onlyRole(DEFAULT_ADMIN_ROLE) {
 require(!_liquidityDeployed, "Liquidity already deployed");
 require(!_emergencyWithdrawUsed, "Emergency withdraw already used");

 _emergencyWithdrawUsed = true;

 // ... mint remaining tokens ...

 uint256 balance = IERC20(PAYMENT_TOKEN).balanceOf(address(this));
 if (balance > 0) {
 IERC20(PAYMENT_TOKEN).transfer(msg.sender, balance); // Return value not checked
 }
}
```
```

2. withdrawToken() function (Line 509):

```
```solidity
function withdrawToken(address token, uint256 amount) external
```



```
onlyRole(DEFAULT_ADMIN_ROLE) {
 IERC20(token).transfer(msg.sender, amount); // Return value not checked
}
` ``
```

This is particularly dangerous in the `\_emergencyWithdraw()` function because:

- The function sets `\_emergencyWithdrawUsed = true` before the transfer
- If the transfer fails silently, the flag is set but no funds are transferred
- The admin cannot call the function again, resulting in permanently locked funds

## Recommendation

---

Use OpenZeppelin's `SafeERC20` library for all token transfers.



# MINOR

## BPA-1 | Missing Zero-Check

Issue	Severity	Location	Status
Missing Zero-Check	Minor	NFT405.sol #1	Resolved

### Description

The `NFT405.sol` contract's `initialize()` function lacks a zero-check on the `mintTo` parameter (Line 27):

```
```solidity
function initialize(
    uint256 _initialConversionRate,
    string memory _name,
    string memory _symbol,
    uint256 _inscriptionSupply,
    uint256 _decimals,
    string memory _uri,
    address mintTo // No zero-check
) public {
    require(!initialized, "NFT405 Already initialized");
    initialized = true;
    _poolState = mintTo; // Line 31: Assignment without validation
    // ...
}
```

The `mintTo` address is directly assigned to `_poolState` without verifying it's not the zero address. This is critical because:



- `_poolState` has special privileges in the contract via the `onlyPool` modifier
- `_poolState` can call the `setMode()` function (Line 58)
- `_poolState` can bypass transfer restrictions when mode is `MODE_TRANSFER_RESTRICTED` (Line 52)
- If `_poolState` is accidentally set to address(0), these privileges would be assigned to the zero address, breaking the contract's access control

Recommendation

Add a zero-address check for the `mintTo` parameter.



MINOR

BPA-2 | Missing Zero-Check

Issue	Severity	Location	Status
Missing Zero-Check	Minor	CloneFactory.sol #1	Resolved

Description

The `CloneFactory.sol` contract's `createVanityToken()` function lacks a zero-check on the `_feeAddress` parameter (Line 48):

```
```solidity
function createVanityToken(
 CommonStructs.CreateToken memory createParams,
 bytes32 salt,
 address _feeAddress, // No zero-check
 uint256 _vanityFee,
 string memory _vanitySuffix
) external payable onlyAllowed returns (address) {
 require(msg.value >= _vanityFee, "Insufficient fee");

 // ... token creation logic ...

 (bool success,) = payable(address(_feeAddress)).call{value: msg.value}(""); // Line 73

 require(success, "Fee transfer failed");
 // ...
}
```
```



The `_feeAddress` is used directly in a low-level call without verifying it's not the zero address. This can lead to:

- **Fee loss:** If `_feeAddress` is accidentally set to address(0), the fee payment will succeed (sending ETH to the zero address burns it permanently)
- **Economic loss:** Users pay vanity fees that are permanently lost instead of being received by the intended fee recipient
- **Silent failures:** The transaction completes successfully but the fees are lost

Recommendation

Add a zero-address check for the `_feeAddress` parameter.



INFORMATIONAL

BPS-3 | Code Quality / Missing Dependency

| Issue | Severity | Location | Status |
|-----------------------------------|---------------|---------------------|----------|
| Code Quality / Missing Dependency | Informational | TokenFactory.sol #1 | Resolved |

Description

Line 7 imports a non-existent file:

```
```solidity
import {TokenImplementation} from "./TokenImplementation.sol";
```
```

The ``TokenImplementation.sol`` file does not exist in the project, but the ``TokenImplementation`` type is used in multiple places throughout the contract code:

- Line 64, 70: ``TokenImplementation.InitializeParams`` and ``TokenImplementation(token).initialize(params)`` in the ``createToken`` function
- Line 86, 93: Used in the ``createTokenDeterministic`` function
- Line 163, 172: Used in the ``batchCreateTokens`` function

Meanwhile, line 8 successfully imports ``TokenImplementationV2.sol``, and the contract has already implemented V2 version functionality (``createTokenV2``, ``createTokenDeterministicV2``, ``batchCreateTokensV2``).

Recommendation

There are two possible solutions:



1. **Remove the dependency on TokenImplementation:** Delete the import statement on line 7 and remove all functions that use ``TokenImplementation`` (``createToken``, ``createTokenDeterministic``, ``batchCreateTokens``), keeping only the V2 version functionality.

2. **Adjust to use V2 version:** Change the import on line 7 and all related functions to use ``TokenImplementationV2``, or create the missing ``TokenImplementation.sol`` file.

It is recommended to adopt solution 1 by removing the deprecated V1-related code and standardizing on the V2 version to maintain code consistency and maintainability.



Disclaimer

This report is based on the scope of materials and documentation provided for a limited review at the time provided. Results may not be complete nor inclusive of all vulnerabilities. The review and this report are provided on an as-is, where-is, and as-available basis. You agree that your access and/or use, including but not limited to any associated services, products, protocols, platforms, content, and materials, will be at your sole risk. Blockchain technology remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the programming language, or other programming aspects that could present security risks. A report does not indicate the endorsement of any particular project or team, nor guarantee its security. No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell a product, service or any other asset. To the fullest extent permitted by law, we disclaim all warranties, expressed or implied, in connection with this report, its content, and the related services and products and your use thereof, including, without limitation, the implied warranties of merchantability, fitness for a particular purpose, and non-infringement. We do not warrant, endorse, guarantee, or assume responsibility for any product or service advertised or offered by a third party through the product, any open source or third-party software, code, libraries, materials, or information linked to, called by, referenced by or accessible through the report, its content, and the related services and products, any hyperlinked websites, any websites or mobile applications appearing on any advertising, and we will not be a party to or in any way be responsible for monitoring any transaction between you and any third-party providers of products or services. As with the purchase or use of a product or service through any medium or in any environment, you should use your best judgment and exercise caution where appropriate.

FOR AVOIDANCE OF DOUBT, THE REPORT, ITS CONTENT, ACCESS, AND/OR USAGE THEREOF, INCLUDING ANY ASSOCIATED SERVICES OR MATERIALS, SHALL NOT BE CONSIDERED OR RELIED UPON AS ANY FORM OF FINANCIAL, INVESTMENT, TAX, LEGAL, REGULATORY, OR OTHER ADVICE.



Appendix

Finding Categories

Centralization / Privilege

Centralization / Privilege findings refer to either feature logic or implementation of components that act against the nature of decentralization, such as explicit ownership or specialized access roles in combination with a mechanism to relocate funds.

Coding Style

Coding Style findings usually do not affect the generated bytecode but rather comment on how to make the codebase more legible and, as a result, easily maintainable.

Volatile Code

Volatile Code findings refer to segments of code that behave unexpectedly on certain edge cases that may result in a vulnerability.

Logical Issue

Logical Issue findings detail a fault in the logic of the linked code, such as an incorrect notion on how block. timestamp works.

Checksum Calculation Method

The "Checksum" field in the "Audit Scope" section is calculated as the SHA-256 (Secure Hash Algorithm 2 with digest size of 256 bits) digest of the content of each file hosted in the listed source repository under the specified commit.

The result is hexadecimal encoded and is the same as the output of the Linux "sha256sum" command against the target file.



About

DeHacker is a team of auditors and white hat hackers who perform security audits and assessments. With decades of experience in security and distributed systems, our experts focus on the ins and outs of system security. Our services follow clear and prudent industry standards. Whether it's reviewing the smallest modifications or a new platform, we'll provide an in-depth security survey at every stage of your company's project. We provide comprehensive vulnerability reports and identify structural inefficiencies in smart contract code, combining high-end security research with a real-world attacker mindset to reduce risk and harden code.

BLOCKCHAINS



Ethereum



Cosmos



Eos



Substrate

TECH STACK



Python



Solidity



Rust



C++

CONTACTS

<https://dehacker.io><https://twitter.com/dehackerio>https://github.com/dehacker/audits_public<https://t.me/dehackerio><https://blog.dehacker.io/>

The image features a dark background with a series of concentric circles in a light green color, centered around the text. The text "DeHacker" is written in a bold, sans-serif font, with the "De" in green and "Hacker" in yellow. The circles are thin and evenly spaced, creating a ripple effect around the central text. There are also some blurred green light sources in the corners of the image.

DeHacker

Nov 18th 2025