Wakehacker Report

Contract address	0x680d3113caf77b61b510f332d5ef4cf5b41a761d
Chain ID	56
Report version	1.0
Last edit	December 12, 2025

"Static analysis or stay rekt"

Document Revisions

1.0	Wakehacker Report - 0x680d3113caf77b61b510f332d5ef4cf5b41a761d on chain 56	December 12, 2025

Contents

Overview

This report was generated using Wakehacker, an automated vulnerability analysis tool. Wakehacker utilizes Wake with additional detectors to perform comprehensive AI and static analysis.

To identify potential vulnerabilities and issues in smart contracts Wake framework utilizes:

- Code structure and patterns
- Control flow graph
- Data flow graph
- Common vulnerability patterns
- Contract interactions

The findings presented in this report are based on automated analysis optimized for precision, aiming for a low false-positive rate. The detection is not optimized for recall—it doesn't target finding all issues (which come at the cost of a high false-positive rate). This code review should be complemented with additional manual code review for a complete security assessment.

Disclaimer

The best effort has been put into finding known vulnerabilities in the system, however automated findings shouldn't be considered as a complete list of all existing issues. The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Finding Classification

Each finding is classified by two independent ratings: *Impact* and *Confidence*.

Impact

Measuring the potential consequences of the issue on the system.

- **Critical** - Code that activates the issue directly enables theft of significant assets with minimal preconditions.
- **High** - Code that activates the issue will lead to undefined or catastrophic consequences for the system.
- **Medium** - Code that activates the issue will result in consequences of serious substance.
- **Low** - Code that activates the issue will have outcomes on the system that are either recoverable or don't jeopardize its regular functioning.
- **Warning** - The issue represents a potential security concern in the code structure or logic that could become problematic with code modifications.
- **Info** - The issue relates to code quality practices that may affect security. Examples include insufficient logging for critical operations or inconsistent error handling patterns.

Confidence

Indicating the probability that the identified issue is a valid security concern.

- **High** - The analysis has identified a pattern that strongly indicates the presence of the issue.
- **Medium** - Evidence suggests the issue exists, but manual verification is recommended.
- **Low** - Potential indicators of the issue have been detected, but there is a significant possibility of false positives.

Executive Summary

Revision 1.0

The audit of DeHub's smart contract at address 0x680d3113caf77b61b510f332d5ef4cf5b41a761d on BSC (Chain ID 56) identified a total of 20 detections across 2 unique detector categories. The analysis revealed 19 high confidence detections and 1 high impact detection, with no critical issues found. The protocol, which focuses on decentralized gaming and streaming through shared computing power infrastructure, demonstrates a generally moderate risk profile with most findings falling into lower severity categories that warrant attention but do not pose immediate critical threats to the system's security or functionality.

Findings

Summary by Impact

2	1	2	0	0	15	20
Critical	High	Medium	Low	Warning	Info	Total

Complete List

Finding Title	Impact	Reported	Status
C1: Snapshot corruption via post-mutation updates in transferBulk	Critical	1.0	Reported
C2: Snapshot corruption via post-mutation updates in transferFromBulk	Critical	1.0	Reported
H1: Public initializeImplementation enables proxy takeover and implementation ownership; can block primary initializer	High	1.0	Reported
M1: Bulk transfer bypasses _beforeTokenTransfer and mis-records snapshot values	Medium	1.0	Reported
M2: Bulk transferFrom bypasses _beforeTokenTransfer and mis-records snapshot values	Medium	1.0	Reported
I1: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported

I2: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I3: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I4: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I5: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I6: pause() documentation claims to block burns, but burns are allowed while paused	Info	1.0	Reported
I7: Burn operations bypass pause via temporary unpause in _beforeTokenTransfer (comment mismatch)	Info	1.0	Reported
I8: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I9: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I10: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported

I11: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I12: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I13: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I14: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported
I15: Function is declared as public but could be external since it has no internal references	Info	1.0	Reported

C1: Snapshot corruption via post-mutation updates in transferBulk

Impact	Critical	Confidence	High
Target	contracts/DeHubTokenV3.sol	Type	Logic error

Description

The bulk transfer implementation updates snapshot data after mutating balances, which violates the ERC20Snapshot invariant that snapshots should record pre-change values for the current snapshot id.

In OpenZeppelin's ERC20SnapshotUpgradeable, snapshot entries for accounts are recorded in the `_beforeTokenTransfer` hook before any balance mutations occur. This ensures that on the first transfer after a snapshot is created, the snapshot captures the balance that existed at the moment of the snapshot.

In `transferBulk`, the code subtracts from the sender's balance and adds to recipients' balances first, and only then calls `_updateAccountSnapshot` for the sender and each recipient. Because `_updateAccountSnapshot` computes the current value via `balanceOf(account)` and stores it for the latest snapshot id if no entry exists yet, this ordering records the post-change balances for the current snapshot id whenever this is the first transfer affecting those accounts after the snapshot.

Effectively, this allows the first bulk transfer after a snapshot to make it appear as if recipients held the transferred tokens at snapshot time (and that the sender held fewer), corrupting any governance voting or distribution logic that relies on `balanceOfAt(snapshotId)`.

Affected components:

- Function: `transferBulk`
- State variables: `_balances[…]` (from ERC20Upgradeable storage), account snapshots maintained by ERC20SnapshotUpgradeable through `_accountBalanceSnapshots` (updated via `_updateAccountSnapshot`).

Listing 1. Code snippet from contracts/DeHubTokenV3.sol

```
136 _balances[sender] -= total;
137 _updateAccountSnapshot(sender);
138
139 for (uint256 i = 0; i < length; ++i) {
140   address recipient = recipients[i];
141   require(recipient != address(0), "ERC20: transfer to the zero address");
142
143   // Note: _beforeTokenTransfer isn't called here
144   // This function emulates what it would do (paused and snapshot)
145
146   _balances[recipient] += amounts[i];
147
148   _updateAccountSnapshot(recipient);
149
150   emit Transfer(sender, recipient, amounts[i]);
151 }
```

Acknowledgement

> The snapshot was only used during the migration phase and is no longer active. During the migration, all trading activities were paused to prevent any potential issues.
>
> — DeHub Team Response

Go back to Findings Summary

C2: Snapshot corruption via post-mutation updates in transferFromBulk

Impact	Critical	Confidence	High
Target	contracts/DeHubTokenV3.sol	Type	Logic error

Description

The `transferFromBulk` function mirrors the same snapshot ordering flaw observed in `transferBulk`. It updates balances first and only then calls `_updateAccountSnapshot` for the affected accounts.

In OpenZeppelin's ERC20Snapshot model, account snapshot entries for the current snapshot id are written before balances are mutated via the `_beforeTokenTransfer` hook. This guarantees that the first transfer after a snapshot records the pre-change balance for that account. Here, because `_updateAccountSnapshot` is invoked after mutations, the first transfer affecting an account following a snapshot will store the post-change balance as the snapshot value, corrupting `balanceOfAt(snapshotId)` results.

With `transferFromBulk`, any operator with sufficient allowance can perform this manipulation for an arbitrary `sender`, inflating recipients' snapshot balances for the current snapshot id.

Affected components:

- Function: `transferFromBulk`
- State variables: `_balances[…]` (ERC20Upgradeable storage), and account snapshots updated by ERC20SnapshotUpgradeable through `_updateAccountSnapshot`.

Listing 2. Code snippet from contracts/DeHubTokenV3.sol

```
189 _balances[sender] -= total;
190 _updateAccountSnapshot(sender);
191
192 for (uint256 i = 0; i < length; ++i) {
193   address recipient = recipients[i];
194   require(recipient != address(0), "ERC20: transfer to the zero address");
195
196   // Note: _beforeTokenTransfer isn't called here
197   // This function emulates what it would do (paused and snapshot)
198
199   _balances[recipient] += amounts[i];
200
201   _updateAccountSnapshot(recipient);
202
203   emit Transfer(sender, recipient, amounts[i]);
204 }
```

Acknowledgement

> The snapshot was only used during the migration phase and is no longer active. During the migration, all trading activities were paused to prevent any potential issues.
>
> — DeHub Team Response

Go back to Findings Summary

H1: Public initializeImplementation enables proxy takeover and implementation ownership; can block primary initializer

Impact	High	Confidence	Medium
Target	contracts/DeHubTokenV3.sol	Type	Access control

Description

The contract exposes a publicly callable initializer, `initializeImplementation()`, on an upgradeable ERC20. When deployed behind a proxy, if this function is callable before the intended `initialize(name, symbol)` is executed, any external account can call it through the proxy. Doing so will set `owner = msg.sender` (via `__Ownable_init()`), call `_pause()`, and consume the initializer modifier, which permanently prevents `initialize(name, symbol)` from running.

This allows an arbitrary account to take ownership and brick the proper initialization path. It also allows the new owner to invoke any `onlyOwner` functions (e.g., `mint`, `burn`, `pause/unpause`) through the proxy.

Listing 3. Code snippet from contracts/DeHubTokenV3.sol

```
34 // Call this on the implementation contract (not the proxy)
35 function initializeImplementation() public initializer {
36   __Ownable_init();
37   _pause();
38 }
```

Acknowledgement

The issue was mitigated by proper initialization immediately after deployment.

Go back to Findings Summary

M1: Bulk transfer bypasses _beforeTokenTransfer and mis-records snapshot values

Impact	Medium	Confidence	High
Target	contracts/DeHubTokenV3.sol	Type	Logic error

Description

The bulk transfer path intentionally bypasses the standard ERC20 transfer flow (and thus the `_beforeTokenTransfer` hook). It directly mutates `_balances` and then calls account-level snapshot updates. While a top-level `paused()` check is replicated, the design omits any additional logic centralized in `_beforeTokenTransfer` (e.g., future blacklists, transfer limits, or fee/taxing) and already diverges from OpenZeppelin snapshot semantics by recording post-change balances at snapshot boundaries instead of pre-change values.

In OpenZeppelin, snapshots are recorded inside `_beforeTokenTransfer` which runs before balances are modified. This ensures the snapshot captures the balance "as of" the snapshot id creation. In `transferBulk`, the sender's balance is decremented before its snapshot update, and recipients' balances are incremented before their snapshot updates. As a result, at the first transfer after a snapshot id is created, the snapshot for those accounts will store post-transfer values, not their balances at the snapshot time. This can distort governance voting power or dividend distributions that rely on accurate historical balances.

Affected elements:

- Function: `transferBulk`
- State: `_balances`, snapshots via `_updateAccountSnapshot`

Listing 4. Code snippet from contracts/DeHubTokenV3.sol

```
117 function transferBulk(
118   address[] calldata recipients,
119   uint256[] calldata amounts
120 ) external returns (bool) {
121   require(!paused(), "ERC20Pausable: token transfer while paused");
122   require(recipients.length == amounts.length, "Invalid arguments length");
123
124   address sender = _msgSender();
125
126   uint256 length = recipients.length;
127   uint256 total = 0;
128   for (uint256 i = 0; i < length; ++i) {
129     total += amounts[i];
130   }
131   require(
132     _balances[sender] >= total,
133     "ERC20: transfer amount exceeds balance"
134   );
135
136   _balances[sender] -= total;
137   _updateAccountSnapshot(sender);
138
139   for (uint256 i = 0; i < length; ++i) {
140     address recipient = recipients[i];
141     require(recipient != address(0), "ERC20: transfer to the zero address");
142
143     // Note: _beforeTokenTransfer isn't called here
144     // This function emulates what it would do (paused and snapshot)
145
146     _balances[recipient] += amounts[i];
147
148     _updateAccountSnapshot(recipient);
149
150     emit Transfer(sender, recipient, amounts[i]);
151   }
152
153   return true;
154 }
```

For contrast, OpenZeppelin records snapshots before balance mutation within the hook:

Listing 5. Code snippet from contracts/oz/ERC20SnapshotUpgradeable.sol

```
124 // Update balance and/or total supply snapshots before the values are
    modified. This is implemented
125 // in the _beforeTokenTransfer hook, which is executed for _mint, _burn, and
    _transfer operations.
126 function _beforeTokenTransfer(
127   address from,
128   address to,
129   uint256 amount
130 ) internal virtual override {
131   super._beforeTokenTransfer(from, to, amount);
132
133   if (from == address(0)) {
134     // mint
135     _updateAccountSnapshot(to);
136     _updateTotalSupplySnapshot();
137   } else if (to == address(0)) {
138     // burn
139     _updateAccountSnapshot(from);
140     _updateTotalSupplySnapshot();
141   } else {
142     // transfer
143     _updateAccountSnapshot(from);
144     _updateAccountSnapshot(to);
145   }
146 }
```

Acknowledgement

> The snapshot was only used during the migration phase and is no longer active. During the migration, all trading activities were paused to prevent any potential issues.
>
> — DeHub Team Response

Go back to Findings Summary

M2: Bulk transferFrom bypasses _beforeTokenTransfer and mis-records snapshot values

Impact	Medium	Confidence	High
Target	contracts/DeHubTokenV3.sol	Type	Logic error

Description

The bulk transfer-from path bypasses the standard ERC20 transfer flow and the `_beforeTokenTransfer` hook. It manually decrements allowance once for the total, updates the sender's balance directly, and updates snapshot values after balances mutate. As with `transferBulk`, this design omits any additional logic centralized in the hook and already records snapshot values post-change, diverging from OpenZeppelin's intended snapshot semantics.

Affected elements:

- Function: `transferFromBulk`
- State: `_balances`, `_allowances`, snapshots via `_updateAccountSnapshot`

Listing 6. Code snippet from contracts/DeHubTokenV3.sol

```
163 function transferFromBulk(
164   address sender,
165   address[] calldata recipients,
166   uint256[] calldata amounts
167 ) external returns (bool) {
168   require(!paused(), "ERC20Pausable: token transfer while paused");
169   require(recipients.length == amounts.length, "Invalid arguments length");
170
171   uint256 length = recipients.length;
172   uint256 total = 0;
173   for (uint256 i = 0; i < length; ++i) {
174     total += amounts[i];
```

```
175   }
176   require(
177     _balances[sender] >= total,
178     "ERC20: transfer amount exceeds balance"
179   );
180
181   // Ensure enough allowance
182   uint256 currentAllowance = _allowances[sender][_msgSender()];
183   require(
184     currentAllowance >= total,
185     "ERC20: transfer total exceeds allowance"
186   );
187   _approve(sender, _msgSender(), currentAllowance - total);
188
189   _balances[sender] -= total;
190   _updateAccountSnapshot(sender);
191
192   for (uint256 i = 0; i < length; ++i) {
193     address recipient = recipients[i];
194     require(recipient != address(0), "ERC20: transfer to the zero address");
195
196     // Note: _beforeTokenTransfer isn't called here
197     // This function emulates what it would do (paused and snapshot)
198
199     _balances[recipient] += amounts[i];
200
201     _updateAccountSnapshot(recipient);
202
203     emit Transfer(sender, recipient, amounts[i]);
204   }
205
206   return true;
207 }
```

For intended snapshot ordering, OpenZeppelin records snapshots within `_beforeTokenTransfer` before any balances are changed:

Listing 7. Code snippet from contracts/oz/ERC20SnapshotUpgradeable.sol

```
124 // Update balance and/or total supply snapshots before the values are
    modified. This is implemented
125 // in the _beforeTokenTransfer hook, which is executed for _mint, _burn, and
    _transfer operations.
```

```
126 function _beforeTokenTransfer(
127   address from,
128   address to,
129   uint256 amount
130 ) internal virtual override {
131   super._beforeTokenTransfer(from, to, amount);
132
133   if (from == address(0)) {
134     // mint
135     _updateAccountSnapshot(to);
136     _updateTotalSupplySnapshot();
137   } else if (to == address(0)) {
138     // burn
139     _updateAccountSnapshot(from);
140     _updateTotalSupplySnapshot();
141   } else {
142     // transfer
143     _updateAccountSnapshot(from);
144     _updateAccountSnapshot(to);
145   }
146 }
```

Acknowledgement

> The snapshot was only used during the migration phase and is no longer active. During the migration, all trading activities were paused to prevent any potential issues.
>
> — DeHub Team Response

Go back to Findings Summary

I1: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	DeHubTokenV3	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 8. Function is declared as public but could be external since it has no internal references in contracts/DeHubTokenV3.sol (L24-L32)

```
21 // Mapping which stores all addresses allowed to snapshot
22 mapping(address => bool) authorizedToSnapshot;
23
24 function initialize(
25   string memory name,
26   string memory symbol
27 ) public initializer {
```

Go back to Findings Summary

I2: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 9. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L154-L160)

```
151  *
152  * - `spender` cannot be the zero address.
153  */
154 function approve(
155   address spender,
156   uint256 amount
157 ) public virtual override returns (bool) {
```

Go back to Findings Summary

I3: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 10. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L175-L190)

```
172  * - the caller must have allowance for ``sender``'s tokens of at least
173  * `amount`.
174  */
175 function transferFrom(
176   address sender,
177   address recipient,
178   uint256 amount
```

Go back to Findings Summary

I4: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 11. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L204-L214)

```
201  *
202  * - `spender` cannot be the zero address.
203  */
204 function increaseAllowance(
205   address spender,
206   uint256 addedValue
207 ) public virtual returns (bool) {
```

Go back to Findings Summary

I5: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 12. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L230-L242)

```
227  * - `spender` must have allowance for the caller of at least
228  * `subtractedValue`.
229  */
230 function decreaseAllowance(
231     address spender,
232     uint256 subtractedValue
233 ) public virtual returns (bool) {
```

Go back to Findings Summary

I6: pause() documentation claims to block burns, but burns are allowed while paused

Impact	Info	Confidence	High
Target	contracts/DeHubTokenV3.sol	Type	Code quality

Description

The function documentation for `pause()` states that pausing prevents any token mint/transfer/burn operations. However, the contract overrides `_beforeTokenTransfer` such that burn operations (where `to == address(0)`) are allowed while paused: it temporarily unpauses, executes the hook, and then re-pauses if the contract was previously paused. This means owner-initiated burns will succeed during pause, contradicting the documentation and potentially misleading operators or off-chain tooling that rely on the stated semantics.

Affected elements:

- Function: `pause()` documentation
- Behavior: `_beforeTokenTransfer` allows burns while paused

Listing 13. Code snippet from contracts/DeHubTokenV3.sol

```
58 /**
59
60  * Pauses the token contract preventing any token mint/transfer/burn
   operations.
61  * Can only be called if the contract is unpaused.
62
63  */
64 function pause() external onlyOwner {
65   _pause();
66 }
```

The following override enables burns while paused by temporarily unpausing and re-pausing around burn operations:

Listing 14. Code snippet from contracts/DeHubTokenV3.sol

```
209 function _beforeTokenTransfer(
210   address from,
211   address to,
212   uint256 amount
213 )
214   internal
215   virtual
216   override(
217     ERC20PausableUpgradeable,
218     ERC20SnapshotUpgradeable,
219     ERC20Upgradeable
220   )
221 {
222   bool alreadyPaused = paused();
223   if (to == address(0)) {
224     if (alreadyPaused) {
225       _unpause();
226     }
227   }
228   super._beforeTokenTransfer(from, to, amount);
229   if (to == address(0)) {
230     if (alreadyPaused) {
231       _pause();
232     }
233   }
234 }
```

Go back to Findings Summary

I7: Burn operations bypass pause via temporary unpause in _beforeTokenTransfer (comment mismatch)

Impact	Info	Confidence	High
Target	contracts/DeHubTokenV3.sol	Type	Code quality

Description

The `_beforeTokenTransfer` override introduces a semantic exception for burns when the token is paused. If the transfer is a burn (`to == address(0)`) and the token is currently paused, the function temporarily unpauses the token, runs the parent hooks, and then re-pauses. This allows burns to succeed while paused.

This behavior contradicts the `pause()` function's documentation ("preventing any token mint/transfer/burn operations"). Although no external calls occur during this temporary unpause, the mismatch may confuse integrators or off-chain systems that expect burns to be blocked when paused.

Listing 15. Code snippet from contracts/DeHubTokenV3.sol

```
209 function _beforeTokenTransfer(
210   address from,
211   address to,
212   uint256 amount
213 )
214   internal
215   virtual
216   override(
217     ERC20PausableUpgradeable,
218     ERC20SnapshotUpgradeable,
219     ERC20Upgradeable
220   )
221 {
222   bool alreadyPaused = paused();
```

```
223   if (to == address(0)) {
224     if (alreadyPaused) {
225       _unpause();
226     }
227   }
228   super._beforeTokenTransfer(from, to, amount);
229   if (to == address(0)) {
230     if (alreadyPaused) {
231       _pause();
232     }
233   }
234 }
```

Go back to Findings Summary

I8: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	DeHubTokenV3	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 16. Function is declared as public but could be external since it has no internal references in contracts/DeHubTokenV3.sol (L35-L38)

```
32 }
33
34 // Call this on the implementation contract (not the proxy)
35 function initializeImplementation() public initializer {
36   __Ownable_init();
37   _pause();
38 }
```

Go back to Findings Summary

I9: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20SnapshotUpgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 17. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20SnapshotUpgradeable.sol (L98-L108)

```
95 /**
96  * @dev Retrieves the balance of `account` at the time `snapshotId` was
   created.
97  */
98 function balanceOfAt(
99   address account,
100   uint256 snapshotId
101 ) public view virtual returns (uint256) {
```

Go back to Findings Summary

I10: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20SnapshotUpgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 18. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20SnapshotUpgradeable.sol (L113-L122)

```
110 /**
111  * @dev Retrieves the total supply at the time `snapshotId` was created.
112  */
113 function totalSupplyAt(
114   uint256 snapshotId
115 ) public view virtual returns (uint256) {
116   (bool snapshotted, uint256 value) = _valueAt(
```

Go back to Findings Summary

I11: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 19. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L76-L78)

```
73 /**
74  * @dev Returns the name of the token.
75  */
76 function name() public view virtual returns (string memory) {
77   return _name;
78 }
```

Go back to Findings Summary

I12: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 20. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L84-L86)

```
81  * @dev Returns the symbol of the token, usually a shorter version of the
82  * name.
83  */
84 function symbol() public view virtual returns (string memory) {
85   return _symbol;
86 }
```

Go back to Findings Summary

I13: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 21. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L101-L103)

```
98  * no way affects any of the arithmetic of the contract, including
99  * {IERC20-balanceOf} and {IERC20-transfer}.
100  */
101 function decimals() public view virtual returns (uint8) {
102   return 18;
103 }
```

Go back to Findings Summary

I14: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 22. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L129-L135)

```
126  * - `recipient` cannot be the zero address.
127  * - the caller must have a balance of at least `amount`.
128  */
129 function transfer(
130   address recipient,
131   uint256 amount
132 ) public virtual override returns (bool) {
```

Go back to Findings Summary

I15: Function is declared as public but could be external since it has no internal references

Impact	Info	Confidence	High
Target	ERC20Upgradeable	Type	N/A

Description

> **Detector Description:** Detects functions declared as `public` that are never called internally within the contract. Such functions can often be marked as `external` to better reflect their intended usage and slightly optimize gas consumption for external calls.
>
> **Potential Problems:** `public` functions that are not used internally introduce unnecessary internal call selectors and may mislead readers into thinking they are intended for internal invocation. This slightly increases deployment size and external call gas costs. It can also make the contract interface less clear and more prone to misunderstandings during audits or maintenance.
>
> — Wake documentation

Code Snippet

Listing 23. Function is declared as public but could be external since it has no internal references in contracts/oz/ERC20Upgradeable.sol (L140-L145)

```
137 /**
138  * @dev See {IERC20-allowance}.
139  */
140 function allowance(
141   address owner,
142   address spender
143 ) public view virtual override returns (uint256) {
```

Go back to Findings Summary

”Static analysis or stay rekt”

https://wakehacker.ai