

EDAINToken.sol

Medium severity issues

- **Initial Token Distribution / Centralization**

In the constructor function, the deployer is minting tokens for his wallet address. All the initial supply are sent to the contract deployer when deploying the contract. This could be a centralization risk, the deployer can distribute those tokens without obtaining the consensus of the community.

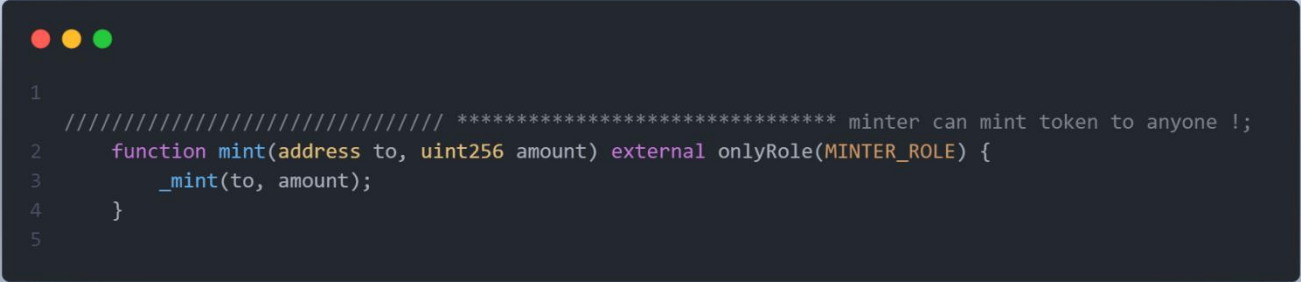
```
1
2  function initialize(uint256 initialMint) public initializer {
3      __ERC20_init("EDAIN", "EAI");
4      __ERC20Burnable_init();
5      __ERC20Snapshot_init();
6      __AccessControl_init();
7      __Pausable_init();
8      __EDAINstaking_init();
9      __ERC20Capped_init(47e7 * 10**decimals());
10
11     _setupRole(DEFAULT_ADMIN_ROLE, msg.sender);
12     _setupRole(SNAPSHOT_ROLE, msg.sender);
13     _setupRole(PAUSER_ROLE, msg.sender);
14     _setupRole(MINTER_ROLE, msg.sender);
15
16     //////////////////////////////////// ***** we are minting initial supply to
17     _mint(msg.sender, initialMint * 10**decimals());
18 }
```

✓ Recommendation

We recommend the team to be transparent regarding the initial token distribution process, and the team shall make enough efforts to restrict the access of the private key, We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage the specific account in this case.

- **MINTER_ROLE / Centralization**

In the mint function, the MINTER_ROLE has access to mint token for any address. This could be a centralization risk, the MINTER_ROLE can distribute those tokens without obtaining the consensus of the community.



```
1 /////////////////////////////////////////////////// ***** minter can mint token to anyone !;
2 function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE) {
3     _mint(to, amount);
4 }
5
```

- ✓ **Recommendation**

We also advise the client to adopt Multisig, Timelock, and/or DAO in the project to manage the specific account in this case.

Low severity issues

- **Unlocked Pragma Used**

Contracts should be deployed with the same compiler version and flags that they have been tested the most with. Locking the pragma helps ensure that contracts do not accidentally get deployed using, for example, the latest compiler which may have higher risks of undiscovered bugs. Contracts may also be deployed by others and the pragma indicates the compiler version intended by the original authors.

```
// bad
pragma solidity ^0.4.4;

// good
pragma solidity 0.4.4;
```

- ✓ **Recommendation**

Should lock pragmas to a specific compiler version.

- **Missing Emit Events**

There should always be events emitted in the sensitive functions.

- function mint(address to, uint256 amount) external onlyRole(MINTER_ROLE);
- function stake(uint256 amount) external;
- function withdrawStake(uint256 amount, uint256 stake_index) external;

- ✓ **Recommendation**

It is recommended emitting events for sensitive functions.