203 – Code Audit

Team Information

Team Name: ISEGYE_IDOL

Team Member: Eungchang Lee, Sojeong Kim, Mingyu Seong, Donghyun HA

Email Address: dfc-isegyeidol@googlegroups.com

Instructions

Description

The DFCC token is an ERC20 token that was developed by the DFC CLUB foundation. DFC CLUB announced in the white paper that DFCC would only issue up to 100 million tokens. In addition, DFC CLUB has stated that they can freeze a malicious account but only the token holder has the authority to transfer.

However, recently, an accident occurred in which DFCC was transferred from some users' addresses to a specific address. The users reported it to the DFC CLUB, saying they had never applied for a transfer. The foundation froze the address to minimize damage. A week after the freeze, the DFCC owner's address was suddenly changed to an unknown address, and then more than 10 billion DFCCs were issued indiscriminately. As a result, DFCC investors suffered huge losses. DFC CLUB claims the incidents were caused by hacking, but investors suspect the foundation's cheating.

A forensic investigator obtained the source code used for token development from the DFC CLUB foundation's server to investigate this case.

Target	Hash (MD5)
DFCC.sol	f277fcdc0f1fe6c0baee92511641f802

Questions

1) Verify through code analysis whether it is possible to transfer tokens to a specific address without the sender's authority. (100 points)

2) Verify that the total token supply is set correctly in the code. (100 points)

Teams must:

- Develop and document the step-by-step approach used to solve this problem to allow another examiner to replicate team actions and results.
- Specify all tools used in deriving the conclusion(s).

Tools used:

Name:	VSCode	Publisher:	Microsoft
Version:	1.68.1		
URL:	https://code.visualstudio.com/		

Step-by-step methodology:

1. Verify through code analysis whether it is possible to transfer tokens to a specific address without the sender's authority. (100 points)

DFCC.sol(Line 364)를 보면 transferFrom함수가 구현되어있다. 이 함수는 notFrozen(from) modifier를 통해 from인자가 frozen된 주소가 아니라면 수행되고, 함수 내부에서 require(msg.sender == owner); 구문을 통해 msg.sender가 owner인 경우에만 진행된다.

```
function transferFrom(address from, address to, uint256 value) public notFrozen(from) returns (bool) {
    require(msg.sender == owner);
    return super.transferFrom(from, to, value);
}
```

이후 super.transferFrom(from, to, value);로 상속하고 있는 컨트랙트의 transferFrom을 동일인자로 호출한다. 현재 DFCC가 상속하고 있는 컨트랙트는 ERC20Detailed, ERC20Pausable, ERC20Burnable으로 이중 tranferFrom을 구현한 컨트랙트는 ERC20Pausable이다.

```
295  contract ERC20Pausable is ERC20, Pausable {
296      function transfer(address to, uint256 value) public whenNotPaused returns (bool) {
297          return super.transfer(to, value);
298      }
299
300      function transferFrom(address from, address to, uint256 value) public whenNotPaused returns (bool) {
301          return super.transferFrom(from, to, value);
302      }
303  }
```

해당 함수는 whenNotPaused modifier를 사용하고 있으므로 _paused 변수가 false일때 동작한다. 동일하게 super.transferFrom(from, to, value);를 호출한다. ERC20Pausable이 상속하고 있는 ERC20을 보면 transferFrom 함수의 원형을 볼 수 있다.

```
function transferFrom(address from, address to, uint256 value) public returns (bool) {
    _balances[from] = _balances[from].sub(value);
    _transfer(from, to, value);
    emit Approval(from, msg.sender, _allowed[from][msg.sender]);
    return true;
}
```

첫번째 인자로 보내는 사람의 주소, 두번째 인자로 받는 사람의 주소, 세번째 인자로 보낼 수량을 입력받는다. _balances은 각 계정의 토큰 수량을 담고 있다.

즉 보내는 사람의 토큰 수량을 value만큼 감소시킨 뒤, _transfer(from, to, value);를 호출한다. 이후 이벤트를 발생시키고 true를 리턴한다.

```
function _transfer(address from, address to, uint256 value) internal {
    require(to != address(0));

    _balances[from] = _balances[from].sub(value);
    _balances[to] = _balances[to].add(value);
    emit Transfer(from, to, value);
}
```

require(to != address(0));로 to가 address(0)이 아니면 진행한다.

이후 방금전에 했던 보내는 사람의 토큰 수량을 value만큼 다시 감소시킨다. 그리고 받는 사람의 토큰 수량을 value만큼 증가시킨다. 이후 이벤트를 발생시킨다.

정리를 하자면 msg.sender가 owner라면 임의의 from의 토큰수량을 value의 두배 만큼 줄이고 to의 토큰수량을 value만큼 증가시킬 수 있다.

이를 이용하면 해당 컨트랙트의 owner는 토큰 소유자의 동의없이 토큰 전송을 할 수 있다.

2. Verify that the total token supply is set correctly in the code. (100 points)

```
constructor() ERC20Detailed("DFC CLUB", "DFCC", 18) public  {
    _mint(msg.sender, 100000000 * (10 ** 18));
}
```

해당 생성자를 통해 토큰의 이름 및 decimals 설정하고 있다. decimals는 18로 설정되어있다. 따

라서 토큰 1개는 컨트랙트상 수치로 10 ** 18개로 표시된다.

```
function _mint(address account, uint256 value) internal {
    require(account != address(0));

    _totalSupply = _totalSupply.add(value);
    _balances[account] = _balances[account].add(value);
    emit Transfer(address(0), account, value);
}
```

_mint함수는 ERC20에 구현되어있고, account가 address(0)이 아닌 경우에만 수행된다. 이후 총 공급량을 뜻하는 _totalSupply에 value만큼 더한다.

그리고 account의 토큰 수량 또한 value만큼 올려준다.

따라서 account에게 value만큼의 토큰을 발행하게된다.

그러므로 생성자에서는 _mint(msg.sender, 100000000 * (10 ** 18));를 통해 총 100000000개를 발행한 것을 볼 수 있다.

현재 발행량이 100000000개인 것은 맞으나 mint함수를 보면 추가적으로 더 발행이 가능한 것을 알 수 있다.

```
function mint(address to, uint256 value) public notFrozen(msg.sender) returns (bool) {
    require(msg.sender == owner);
    _mint(to, value);
}
```

아까와 마찬가지로 notFrozen(msg.sender) modifier를 사용해 msg.sender가 frozen된 상태가 아니라면 수행할 수 있고, require(msg.sender == owner);로 msg.sender가 owner인 경우에만 수행된다.

이후 _mint(to, value);로 to 주소에 value만큼의 토큰을 발행한다.

정리해보면 최초 공급량은 100000000개가 맞지만, 컨트랙트의 owner가 언제든 공급량을 늘릴 수 있다.