Secureum Bootcamp Epoch∞ - February RACE #4

February 9, 2022 / patrickd

Bootcamp Epoch 0 has finished, but Epoch Infinity has just begun!
This is a writeup of the Secureum Bootcamp Race 4 Quiz with solutions.
For fairness it was published after submissions to it were closed.

This is the first quiz of the Secureum Epoch Infinity. Note that it was given the name RACE 4 because several quizzes of the previous Epoch 0 were selected to represent the first RACEs (slot-4 -> RACE-0; slot-5 -> RACE-1; slot-7 -> RACE-2; slot-8 -> RACE-3).

This quiz had a strict time limit of 16 minutes for 8 questions, no pause.
Choose all and *only* correct answers.
Syntax highlighting was omitted since the original quiz did not have any either.

[**Note:** All 8 questions in this quiz are based on the InSecureum contract. This is the same contract you will see for all the 8 questions in this quiz. *InSecureum* is adapted from a widely used ERC20 contract.]

```
pragma solidity 0.8.10;

import "https://github.com/OpenZeppelin/openzeppelin-contracts/
import "https://github.com/OpenZeppelin/openzeppelin-contracts/
import "https://github.com/OpenZeppelin/openzeppelin-contracts/

contract InSecureum is Context, IERC20, IERC20Metadata {
```

```
mapping(address => uint256) private _balances;
mapping(address => mapping(address => uint256)) private _a
uint256 private _totalSupply;
string private _name;
string private _symbol;

constructor(string memory name_, string memory symbol_) {
    _name = name_;
    _symbol = symbol_;
}

function name() public view virtual override returns (stri
    return _name;
}

function symbol() public view virtual override returns (st
    return _symbol;
}

function decimals() public view virtual override returns (
    return 8;
}

function totalSupply() public view virtual override return
    return _totalSupply;
}

function balanceOf(address account) public view virtual ov
    return _balances[account];
}

function transfer(address recipient, uint256 amount) publi
    _transfer(_msgSender(), recipient, amount);
    return true;
}

function allowance(address owner, address spender) public
    return _allowances[owner][spender];
}

function approve(address spender, uint256 amount) public v
```

```
    _approve(_msgSender(), spender, amount);
    return true;
}

function transferFrom(
    address sender,
    address recipient,
    uint256 amount
) public virtual override returns (bool) {
    uint256 currentAllowance = _allowances[_msgSender()][se
    if (currentAllowance != type(uint256).max) {
        unchecked {
            _approve(sender, _msgSender(), currentAllowance
        }
    }
    _transfer(sender, recipient, amount);
    return false;
}

function increaseAllowance(address spender, uint256 addedVa
    _approve(_msgSender(), spender, _allowances[_msgSender(
    return true;
}

function decreaseAllowance(address spender, uint256 subtrac
    uint256 currentAllowance = _allowances[_msgSender()][sp
    require(currentAllowance > subtractedValue, "ERCH20: de
    _approve(_msgSender(), spender, currentAllowance - subt
    return true;
}

function _transfer(
    address sender,
    address recipient,
    uint256 amount
) internal virtual {
    require(sender != address(0), "ERC20: transfer from the
    require(recipient != address(0), "ERC20: transfer to th
    uint256 senderBalance = _balances[sender];
    require(senderBalance >= amount, "ERC20: transfer amoun
    unchecked {
```

```
            _balances[sender] = senderBalance - amount;
        }
        _balances[recipient] += amount;
        emit Transfer(sender, recipient, amount);
    }

    function _mint(address account, uint256 amount) external v
        _totalSupply += amount;
        _balances[account] = amount;
        emit Transfer(address(0), account, amount);
    }

    function _burn(address account, uint256 amount) internal v
        require(account != address(0), "ERC20: burn from zero a
        require(_balances[account] >= amount, "ERC20: burn amou
        unchecked {
            _balances[account] = _balances[account] - amount;
        }
        _totalSupply -= amount;
        emit Transfer(address(0), account, amount);
    }

    function _approve(
        address owner,
        address spender,
        uint256 amount
    ) internal virtual {
        require(spender != address(0), "ERC20: approve from the
        require(owner != address(0), "ERC20: approve to the zer
        _allowances[owner][spender] += amount;
        emit Approval(owner, spender, amount);
    }
}
```

> “`InSecureum` implements”

— 1 of 8

- [x] A. Atypical `decimals` value
- [x] B. Non-standard `decreaseAllowance` and `increaseAllowance`
- [ ] C. Non-standard `transfer`
- [ ] D. None of the above

▼ Solution

Correct is A, B. The decimals value follows the standard but it typically returns `18` (`8` is atypical), imitating the relationship between Ether and Wei. The `decreaseAllowance` and `increaseAllowance` functions were introduced in the OpenZeppelin ERC20 implementation to mitigate frontrunning issues of the standard `approve`, but they are not part of the ERC20 standard. The `transfer` function is part of the standard though.

> "In `InSecureum`"

— 2 of 8

- [x] A. `decimals()` can have `pure` state mutability instead of `view`
- [ ] B. `_burn()` can have `external` visibility instead of `internal`
- [x] C. `_mint()` should have `internal` visibility instead of `external`
- [ ] D. None of the above

▼ Solution

Correct is A, C. Since `decimals()` returns a constant hardcoded value without accessing storage other non-calldata information it can indeed be declared as `pure`. Generally, functions prefixed with underscores should be internal or should not have the prefix. Making `_burn()` external would currently allow anyone to burn anyone else's balance. And the fact that `_mint()` is currently external allows anyone to mint as many InSecureum tokens as they wish.

> "`InSecureum transferFrom()`"

— 3 of 8

- [x] A. Is susceptible to an integer underflow

- [x] B. Has an incorrect allowance check
- [x] C. Has an optimization indicative of unlimited approvals
- [ ] D. None of the above

▼ Solution

Correct is A, B, C The subtraction within the `unchecked` block effectively allows anyone to steal anyone else's full token balance since subtracting from an allowance of `0` will cause an integer underflow and the allowance value will wrap (`0 - 1 == type(uint256).max`). The fact that the function won't revert when subtracting from the allowance due to the `unchecked` block, can by itself be seen as an incorrect allowance check. The other check, skipping allowance subtraction when an "infinite approval" was given by setting the allowance to the maximum value of `uint256`, appears to be correct. Since the special handling for unlimited approvals prevents unnecessary storage updates, it is indicative of an optimization.

> “In `InSecureum`”

— 4 of 8

- [ ] A. `increaseAllowance` is susceptible to an integer overflow
- [ ] B. `decreaseAllowance` is susceptible to an integer overflow
- [x] C. `decreaseAllowance` does not allow reducing allowance to zero
- [x] D. `decreaseAllowance` can be optimised with `unchecked{}`

▼ Solution

Correct is C, D. Neither function make use of the `unchecked` block which would allow integer overflows to happen in this version of solidity. The `decreaseAllowance` function does indeed not allow reducing the allowance to zero since the requirement enforces that the `subtractedValue` must always be smaller than `currentAllowance`. It would be better to use `>=` here to allow allowance reductions to zero. That requirement does make solidity's own integer underflow check for `currentAllowance - subtractedValue` redundant, so it could indeed be optimised with `unchecked{}`.

> “`InSecureum _transfer()`”

— 5 of 8

- [ ] A. Is missing a zero-address validation
- [ ] B. Is susceptible to an integer overflow
- [ ] C. Is susceptible to an integer underflow
- [x] D. None of the above

▼ Solution

Correct is D. All of the addresses `_transfer()` uses are checked to make sure they're not zero-addresses. Neither integer overflows nor underflows are possible with this solidity version without the use of `unchecked{}`.

> “`InSecureum _mint()`”
>
> — 6 of 8

- [x] A. Is missing a zero-address validation
- [ ] B. Has an incorrect event emission
- [x] C. Has an incorrect update of account balance
- [ ] D. None of the above

▼ Solution

Correct is A, C. The `_mint` function is currently not ensuring that the receiving address is non-zero. The event emission appears to be correctly following IERC20: `event Transfer(address indexed from, address indexed to, uint256 value);`. This mint implementation overwrites the accounts current balance instead of adding to it.

> “`InSecureum _burn()`”
>
> — 7 of 8

- [ ] A. Is missing a zero-address validation
- [x] B. Has an incorrect event emission
- [ ] C. Has an incorrect update of account balance
- [ ] D. None of the above

▼ Solution

Correct is B. It correctly applies zero-address validation on the account to burn from. The event permission is incorrect, `from` and `to` need to be switched around to follow the IERC20 interface: `event Transfer(address indexed from, address indexed to, uint256 value);`. The balance update is correct and although an `unchecked` block is used, no underflow can happen thanks to the requirement before.

> “`InSecureum _approve()`”
>
> — 8 of 8

- [ ] A. Is missing a zero-address validation
- [x] B. Has incorrect error messages
- [x] C. Has an incorrect update of allowance
- [ ] D. None of the above

▼ Solution

Correct is B, C. Although no zero-address validation is missing the error messages have been confused with each other. The update of allowances is currently incorrect since it only adds the amount to the current allowance instead of setting it to the amount overwriting the old value.

In Blockchain Tags Ethereum, Secureum Bootcamp

← Secureum Bootcamp CARE: Sushi's … Fuzzing complex Projects with Echidna:…