Secureum Bootcamp Epoch∞ - March RACE #5

March 8, 2022 / patrickd

This is a write-up of the Secureum Bootcamp Race 5 Quiz of Epoch Infinity with solutions.
For fairness it was published after submissions to it were closed.

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 well-known contract.]

```
pragma solidity ^0.8.0;

import "https://github.com/OpenZeppelin/openzeppelin-contracts
import "https://github.com/OpenZeppelin/openzeppelin-contracts
import "https://github.com/OpenZeppelin/openzeppelin-contracts
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, ERC165, IERC1155, IERC1155Meta

    mapping(uint256 => mapping(address => uint256)) private _b
    mapping(address => mapping(address => bool)) private _oper
    string private _uri;

    constructor(string memory uri_) {
        _setURI(uri_);
    }

    function supportsInterface(bytes4 interfaceId) public view
        return
            interfaceId == type(IERC1155).interfaceId ||
            interfaceId == type(IERC1155MetadataURI).interface
            super.supportsInterface(interfaceId);
    }

    function uri(uint256) public view virtual override returns
        return _uri;
    }

    function balanceOf(address account, uint256 id) public vie
        require(account != address(0), "ERC1155: balance query
        return _balances[id][account];
    }

    function balanceOfBatch(address[] memory accounts, uint256
        public
        view
        virtual
        override
        returns (uint256[] memory)
    {
        uint256[] memory batchBalances = new uint256[](account
        for (uint256 i = 0; i < accounts.length; ++i) {
            batchBalances[i] = balanceOf(accounts[i], ids[i]);
        }
        return batchBalances;
    }
```

```
function setApprovalForAll(address operator, bool approve
    _setApprovalForAll(_msgSender(), operator, approved);
}

function isApprovedForAll(address account, address operat
    return _operatorApprovals[account][operator];
}

function safeTransferFrom(
    address from,
    address to,
    uint256 id,
    uint256 amount,
    bytes memory data
) public virtual override {
    require(
        from == _msgSender() || isApprovedForAll(from, _m
        "ERC1155: caller is not owner nor approved"
    );
    _safeTransferFrom(from, to, id, amount, data);
}

function safeBatchTransferFrom(
    address from,
    address to,
    uint256[] memory ids,
    uint256[] memory amounts,
    bytes memory data
) public virtual override {
    require(
        from == _msgSender() || isApprovedForAll(from, _m
        "ERC1155: transfer caller is not owner nor approve
    );
    _safeBatchTransferFrom(from, to, ids, amounts, data);
}

function _safeTransferFrom(
    address from,
    address to,
    uint256 id,
```

```
        uint256 amount,
        bytes memory data
    ) public virtual {
        address operator = _msgSender();
        uint256 fromBalance = _balances[id][from];
        unchecked {
            fromBalance = fromBalance - amount;
        }
        _balances[id][from] = fromBalance;
        _balances[id][to] += amount;

        emit TransferSingle(operator, from, to, id, amount);
        _doSafeTransferAcceptanceCheck(operator, from, to, id,
    }

    function _safeBatchTransferFrom(
        address from,
        address to,
        uint256[] memory ids,
        uint256[] memory amounts,
        bytes memory data
    ) internal virtual {
        require(to != address(0), "ERC1155: transfer to the ze
        address operator = _msgSender();
        for (uint256 i = 0; i < ids.length; ++i) {
            uint256 id = ids[i];
            uint256 amount = amounts[i];
            uint256 fromBalance = _balances[id][from];
            fromBalance = fromBalance - amount;
            _balances[id][to] += amount;
        }
        emit TransferBatch(operator, from, to, ids, amounts);
        _doSafeBatchTransferAcceptanceCheck(operator, from, to
    }

    function _setURI(string memory newuri) internal virtual {
        _uri = newuri;
    }

    function _mint(
```

```
        address to,
        uint256 id,
        uint256 amount,
        bytes memory data
    ) internal virtual {
        require(to != address(0), "ERC1155: mint to the zero a
        address operator = _msgSender();
        _balances[id][to] += amount;
        emit TransferSingle(operator, address(0), to, id, amou
        _doSafeTransferAcceptanceCheck(operator, address(0), t
    }

    function _mintBatch(
        address to,
        uint256[] memory ids,
        uint256[] memory amounts,
        bytes memory data
    ) internal virtual {
        address operator = _msgSender();
        require(operator != address(0), "ERC1155: mint from th
        for (uint256 i = 0; i < ids.length; i++) {
            _balances[ids[i]][to] += amounts[i];
        }
        emit TransferBatch(operator, address(0), to, amounts,
        _doSafeBatchTransferAcceptanceCheck(operator, address(
    }

    function _burn(
        address from,
        uint256 id,
        uint256 amount
    ) internal virtual {
        require(from != address(0), "ERC1155: burn from the ze
        address operator = _msgSender();
        uint256 fromBalance = _balances[id][from];
        _balances[id][from] = fromBalance - amount;
        emit TransferSingle(operator, from, address(0), id, am
    }

    function _burnBatch(
```

```
        address from,
        uint256[] memory ids,
        uint256[] memory amounts
    ) internal virtual {
        require(from != address(0), "ERC1155: burn from the ze
        address operator = _msgSender();
        for (uint256 i = 0; i < ids.length; i++) {
            uint256 id = ids[i];
            uint256 amount = amounts[i];
            uint256 fromBalance = _balances[id][from];
            require(fromBalance >= amount, "ERC1155: burn amou
            unchecked {
                _balances[id][from] = fromBalance - amount;
            }
        }
        emit TransferBatch(operator, from, address(0), ids, ar
    }

    function _setApprovalForAll(
        address owner,
        address operator,
        bool approved
    ) internal virtual {
        require(owner != operator, "ERC1155: setting approval
        _operatorApprovals[owner][operator] = approved;
        emit ApprovalForAll(owner, operator, approved);
    }

    function _doSafeTransferAcceptanceCheck(
        address operator,
        address from,
        address to,
        uint256 id,
        uint256 amount,
        bytes memory data
    ) private {
        if (isContract(to)) {
            try IERC1155Receiver(to).onERC1155Received(operato
                if (response == IERC1155Receiver.onERC1155Rece
                    revert("ERC1155: ERC1155Receiver rejected
```

```
                }
            } catch Error(string memory reason) {
                revert(reason);
            } catch {
                revert("ERC1155: transfer to non ERC1155Recei
            }
        }
    }

    function _doSafeBatchTransferAcceptanceCheck(
        address operator,
        address from,
        address to,
        uint256[] memory ids,
        uint256[] memory amounts,
        bytes memory data
    ) private {
        if (isContract(to)) {
            try IERC1155Receiver(to).onERC1155BatchReceived(op
                bytes4 response
            ) {
                if (response != IERC1155Receiver.onERC1155Bat
                    revert("ERC1155: ERC1155Receiver rejected
                }
            } catch Error(string memory reason) {
                revert(reason);
            } catch {
                revert("ERC1155: transfer to non ERC1155Recei
            }
        }
    }

    function isContract(address account) internal view returns
        return account.code.length == 0;
    }
}
```

> “InSecureum balanceOf()”
>
> — 1 of 8

- [ ] A. May be optimised by caching state variable in local variable
- [ ] B. May be optimised by changing state mutability from `view` to `pure`
- [ ] C. May be optimised by changing its visibility to `external`
- [x] D. None of the above

▼ Solution

Correct is D.

Since the `_balances` state variable is only accessed once and immediately returned, caching doesn't make sense.

State mutability can't be changed to `pure` since the function accesses a state variable, that requires at least `view`.

It can't be changed to `external` because it is currently being called internally by the `balanceOfBatch()` function.

> “In InSecureum, array lengths mismatch check is missing in”
>
> — 2 of 8

- [x] A. `balanceOfBatch()`
- [x] B. `_safeBatchTransferFrom()`
- [x] C. `_mintBatch()`
- [x] D. `_burnBatch()`

▼ Solution

Correct is A, B, C, D.

The public function `balanceOfBatch()` receives a list of `accounts` and a list of `ids`, both of which items get passed on to `balanceOf(accounts[i], ids[i]);`. To ensure that neither array is accessed out-of-bounds, it should be checked whether both lists are of the same length.

Neither the internal function `_safeBatchTransferFrom()` nor its public caller function `safeBatchTransferFrom()` check the length of passed `ids` and `amounts`. Therefore the check is missing.

The internal functions `_mintBatch()` and `_burnBatch()` are currently never called, but a contract extending InSecureum might. It would make sense to check the lengths of passed `ids` and `amounts` in them, so that public functions calling them do not need to remember to do so.

> "The security concern(s) with InSecureum _safeTransferFrom() is/are"
>
> — 3 of 8

- [x] A. Incorrect visibility
- [x] B. Susceptible to an integer underflow
- [x] C. Missing zero-address validation
- [ ] D. None of the above

▼ Solution

Correct is A, B, C.

It is prefixed with an underscore, which is usually an indication of an `internal` visibility, and it's also called by a similarly named public `safeTransferFrom()` function that applies more input validation before calling it. This validation ensures that the sender actually has approval for the transfer of funds, which would be bypassed by this function being public. It should instead be `internal` allowing an inheriting contract to internally call it.

The new `fromBalance` is calculated within an `unchecked{}` block, bypassing integer underflow prevention measures of Solidity version 0.8.0^. Since the `fromBalance` isn't checked for whether there's a sufficient balance for a transfer, this effectively allows sending unlimited amounts to the specified recipient.

Neither `safeTransferFrom()` nor `_safeTransferFrom()` are checking whether the `to` address is non-zero, making it possible to accidentally burn tokens.

> “The security concern(s) with InSecureum _safeBatchTransferFrom() is/are”
>
> — 4 of 8

- [x] A. Missing array lengths mismatch check
- [ ] B. Susceptibility to an integer underflow
- [x] C. Incorrect balance update
- [ ] D. None of the above

▼ Solution

Correct is A, C.

The fact that the array lengths mismatch check is missing has already been determined in Question #2.

There's no usage of an `unchecked{}` block, therefore an integer underflow cannot happen with this Solidity version.

The new value of `fromBalance` is calculated but it's never actually updated in storage. This effectively allows sending the same tokens unlimited amount of times.

> “The security concern(s) with InSecureum _mintBatch() is/are”
>
> — 5 of 8

- [x] A. Missing array lengths mismatch check
- [x] B. Incorrect event emission
- [x] C. Allows burning of tokens
- [ ] D. None of the above

▼ Solution

Correct is A, B, C.

The fact that the array lengths mismatch check is missing has already been determined in Question #2.

Comparing the emission of the `TransferBatch` event to other occurrences, it appears that `ids` and `amounts` have been accidentally swapped.

The zero-address check incorrectly ensures that the sender is non-zero (which would never be possible anyway) instead of ensuring that the receiving account is non-zero. This effectively allows minting to the zero-address, burning all minted tokens immediately.

> “The security concern(s) with InSecureum _burn() is/are”
>
> — 6 of 8

- [ ] A. Missing zero-address validation
- [ ] B. Susceptibility to an integer underflow
- [ ] C. Incorrect balance update
- [x] D. None of the above

▼ Solution

Correct is D.

The zero-address validation exists and is correctly checking the value of `from`.

There's no usage of an `unchecked{}` block, therefore an integer underflow cannot happen with this Solidity version.

The balance appears to be correctly updated after subtraction.

> “The security concern(s) with InSecureum _doSafeTransferAcceptanceCheck() is/are”
>
> — 7 of 8

- [ ] A. `isContract` check on incorrect address
- [x] B. Incorrect check on return value
- [x] C. Call to incorrect `isContract` implementation

- [ ] D. None of the above

▼ Solution

Correct is B, C.

The `isContract()` function is correctly called on `to`, which is the receiving address that is potentially a contract that this function is supposed to check support of ERC1155, before tokens are sent to it, since they'd otherwise be stuck in a contract not supporting this standard.

Comparing `_doSafeTransferAcceptanceCheck()` and `_doSafeBatchTransferAcceptanceCheck()` shows a clear discrepancy when checking the return value, with the batch function's implementation correctly checking support for the ERC1155 standard. This function is in fact currently doing the opposite, ensuring that tokens are only sent to contracts that do NOT support it.

The `isContract()` function currently returns `true` if the passed address is in fact NOT a contract (has a code length of 0). It should instead return true only when the address has a code length larger than 0, showing that there's currently a contract residing at `account`.

> “The security concern(s) with InSecureum isContract() implementation is/are”
>
> — 8 of 8

- [ ] A. Incorrect visibility
- [x] B. Incorrect operator in the comparison
- [ ] C. Unnecessary because Ethereum only has Contract accounts
- [ ] D. None of the above

▼ Solution

Correct is B.

A visibility of `internal` allowing inheriting contracts to use it appears appropriate.

The comparison should indeed be "bigger-than-zero" instead of "equals-zero", for the reasons explained for the previous question.

Ethereum not only has Contract accounts but also EOA (Externally Owned Accounts), which do not have any contract code but an off-chain public-private keypair instead.

In Blockchain Tags Ethereum, Secureum Bootcamp

← EVM Puzzles – Second Wind Damn Vulnerable DeFi V2 - #10 Free rid…