TECHFUND Audit Score

Smart contract Audit Report

BOSOTOKYO Audit

Completed on 2022-07-22

Score	POSITIVE	
Risk level	Critical	0
	High	0
	Medium	1
	Low	0
	Note	3

Risk level detail

Overall Risk Severity

Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

The tester arrives at the likelihood and impact estimates, they can now combine them to get a final severity rating for this risk. Note that if they have good business impact information, they should use that instead of the technical impact information.

https://owasp.org/www-community/OWASP_Risk_Rating_Methodology

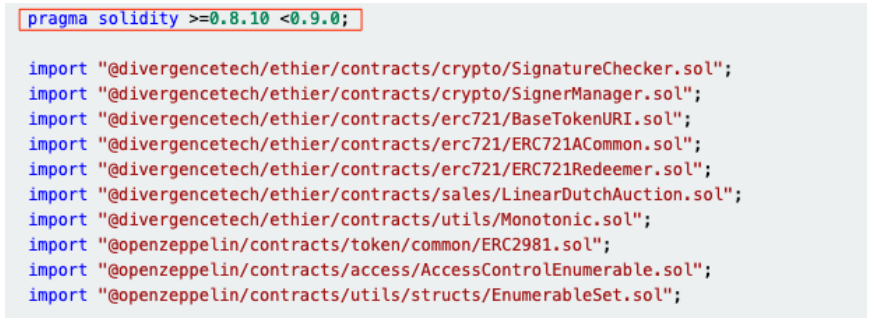
Vulnerability Review

Vulnerability Review	Number of warnings
Compiler Version	1
State Variable Default Visibility	0
Upper-bound Check Array	1
Unbounded Array	1
Integer Overflow / Underflow	0
Parity Multisig Bug	0
Callstack Depth Attack	0
Production Node modules security	0
Development Node modules security	0
Re-Entrancy	1
Double Withdrawal	0

Compiler Version

1

Solidity version used is inconsistent & not fixed. It is highly recommended to update the Solidity version to the latest. Using an old version prevents access to new Solidity security checks. We also recommend avoiding complex pragma statements and using a fixed version of Solidity upon understanding the version specifications.



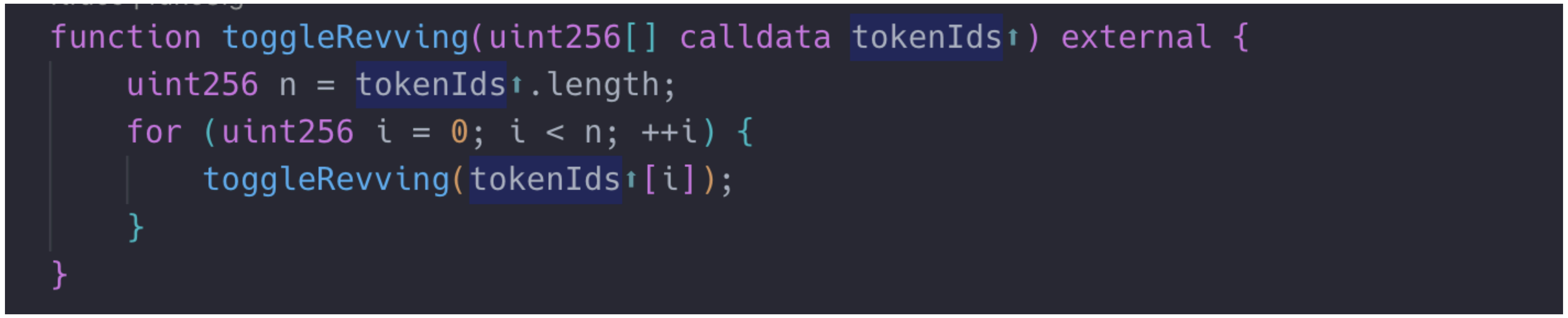
```
pragma solidity >=0.8.10 <0.9.0;

import "@divergencetech/ethier/contracts/crypto/SignatureChecker.sol";
import "@divergencetech/ethier/contracts/crypto/SignerManager.sol";
import "@divergencetech/ethier/contracts/erc721/BaseTokenURI.sol";
import "@divergencetech/ethier/contracts/erc721/ERC721ACommon.sol";
import "@divergencetech/ethier/contracts/erc721/ERC721Redeemer.sol";
import "@divergencetech/ethier/contracts/sales/LinearDutchAuction.sol";
import "@divergencetech/ethier/contracts/utils/Monotonic.sol";
import "@openzeppelin/contracts/token/common/ERC2981.sol";
import "@openzeppelin/contracts/access/AccessControlEnumerable.sol";
import "@openzeppelin/contracts/utils/structs/EnumerableSet.sol";
```

Unbounded Array

1

Loops that do not have a fixed number of iterations, have to be used carefully. Due to the block gas limit, transactions can only consume a certain amount of gas. The number of iterations in a loop can grow beyond the block gas limit which can cause the complete contract to be stalled at a certain point.

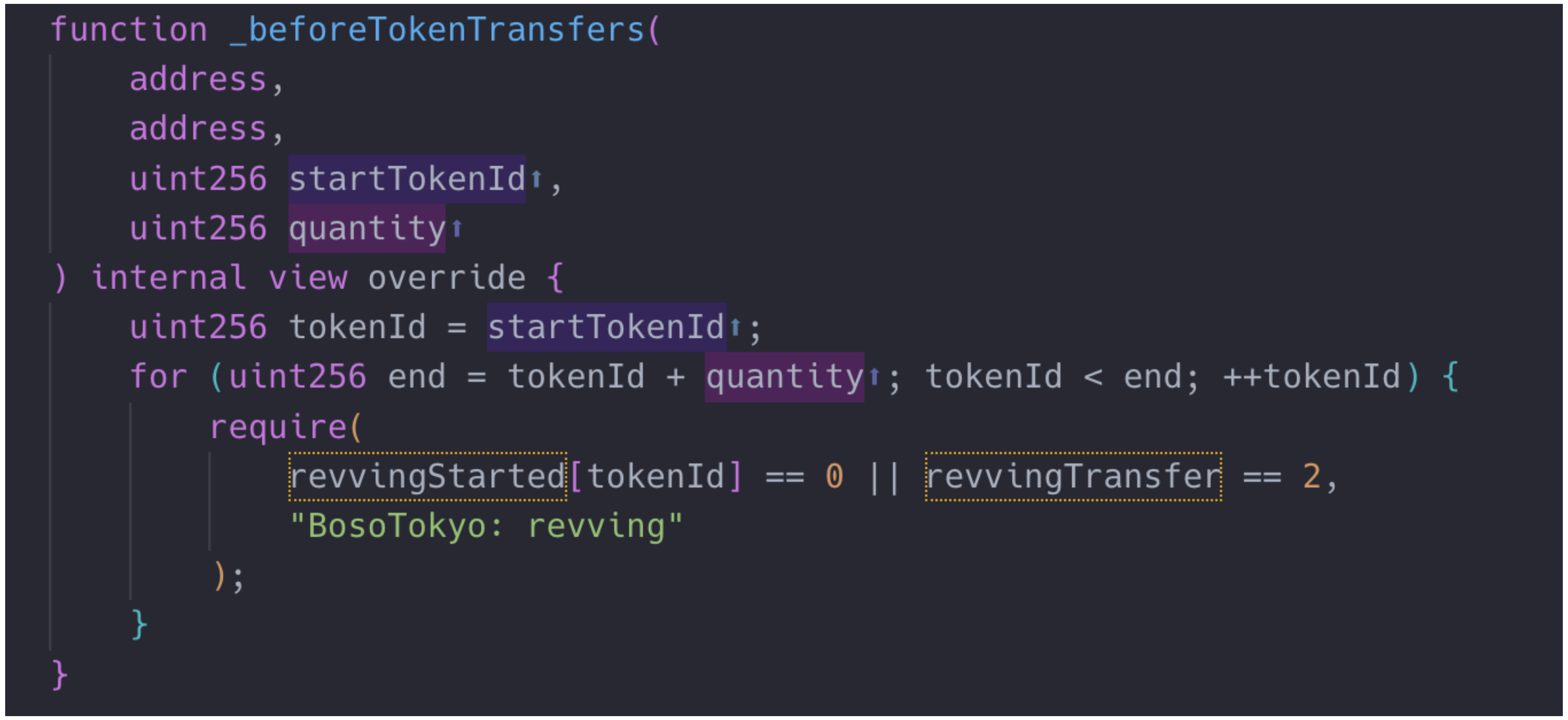


```
function toggleRevving(uint256[] calldata tokenIds) external {
    uint256 n = tokenIds.length;
    for (uint256 i = 0; i < n; ++i) {
        toggleRevving(tokenIds[i]);
    }
}
```

Upper-Bound Check Array

1

Although it is not an issue, but it is suggested that since “tokenId + quantity” can never be greater than total number of Tokens, it should be checked before starting the loop.
(Add check that tokenId + quantity <= Total Tokens before starting the loop to throw error early in the process)



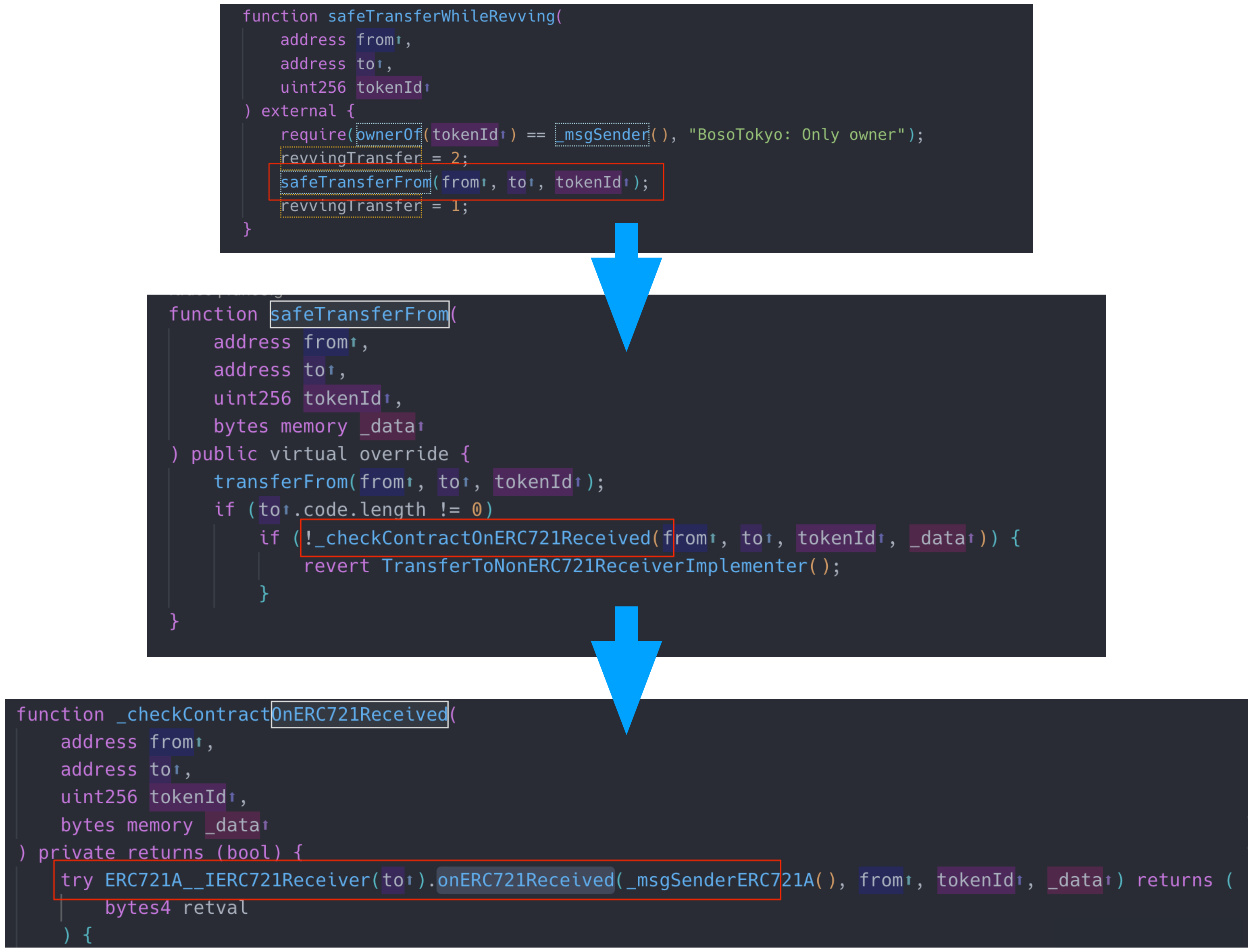
```
function _beforeTokenTransfers(
    address,
    address,
    uint256 startTokenId,
    uint256 quantity
) internal view override {
    uint256 tokenId = startTokenId;
    for (uint256 end = tokenId + quantity; tokenId < end; ++tokenId) {
        require(
            revvingStarted[tokenId] == 0 || revvingTransfer == 2,
            "BosoTokyo: revving"
        );
    }
}
```

Re-Entry

1

A reentrancy attack can occur when you create a function that makes an external call to another untrusted contract before it resolves any effects. If the attacker can control the untrusted contract, they can make a recursive call back to the original function, repeating interactions that would have otherwise not run after the effects were resolved.

For safety, **safeTransferFrom** function performs a callback to the recipient of the token to check that they're willing to accept the transfer. However, we're the recipient of the token, which means we just got a callback at which point we can do whatever we like. Which opens up the possibility of re-entry.



```
function safeTransferWhileRevving(
    address from,
    address to,
    uint256 tokenId
) external {
    require(ownerOf(tokenId) == _msgSender(), "BosoTokyo: Only owner");
    revvingTransfer = 2;
    safeTransferFrom(from, to, tokenId);
    revvingTransfer = 1;
}
```

```
function safeTransferFrom(
    address from,
    address to,
    uint256 tokenId,
    bytes memory _data
) public virtual override {
    transferFrom(from, to, tokenId);
    if (to.code.length != 0)
        if (!_checkContractOnERC721Received(from, to, tokenId, _data)) {
            revert TransferToNonERC721ReceiverImplementer();
        }
}
```

```
function _checkContractOnERC721Received(
    address from,
    address to,
    uint256 tokenId,
    bytes memory _data
) private returns (bool) {
    try ERC721A__IERC721Receiver(to).onERC721Received(_msgSenderERC721A(), from, tokenId, _data) returns (
        bytes4 retval
    ) {
```