MythX

REPORT 61B257769AE4C8001892007B

Created	Thu Dec 09 2021 19:22:30 GMT+0000 (Coordinated Universal Time)
Number of analyses	4
User	61a16d951fd393e3bd1a2ca0

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
48e4ed6e-5761-4bcc-a296-2b251a35c058	contracts/MarebitsLocker.sol	0
9683d006-f58c-46b5-a006-87e10dc26c2d	contracts/MarebitsLockerAccount.sol	4
f7190e6c-10db-4c2c-9937-30d7dcdc1f5e	contracts/MarebitsLockerToken.sol	2
cd0119b6-d36f-4fa5-865b-a31fbdd3e78a	contracts/MarebitsVault.sol	3

Analysis `48e4ed6e-5761-4bcc-a296-2b251a35c058` MythX

Started	Thu Dec 09 2021 19:23:35 GMT+0000 (Coordinated Universal Time)
Finished	Thu Dec 09 2021 20:08:41 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`Contracts/MarebitsLocker.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	0

ISSUES

Analysis 9683d006-f58c-46b5-a006-87e10dc26c2d MythX

Started	Thu Dec 09 2021 19:23:45 GMT+0000 (Coordinated Universal Time)
Finished	Thu Dec 09 2021 20:09:12 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/MarebitsLockerAccount.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	4

ISSUES

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

`contracts/RecoverableTokens.sol`

Locations

```
* @param tokenId to recover
*/
function __recoverERC721(IERC721 token, uint256 tokenId) internal { token.safeTransferFrom(address(this), payable(owner()), tokenId); }

/// @inheritdoc IRecoverableTokens
```

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

`contracts/RecoverableTokens.sol`

Locations

```
function __recoverERC1155(IERC1155 token, uint256 tokenId) internal {
uint256 balance = token.balanceOf(address(this), tokenId);
token.safeTransferFrom(address(this), payable(owner()), tokenId, balance, "");
}

```

LOW

SWC-113

Multiple calls are executed in the same transaction.

This call is executed following another call within the same transaction. It is possible that the call never gets executed if a prior call fails permanently. This might be caused intentionally by a malicious callee. If possible, refactor the code such that each transaction only executes one external call or make sure that all callees can be trusted (i.e. they're part of your own codebase).

Source file

`contracts/RecoverableTokens.sol`

Locations

```
function __recoverERC1155(IERC1155 token, uint256 tokenId) internal {
uint256 balance = token.balanceOf(address(this), tokenId);
token.safeTransferFrom(address(this), payable(owner()), tokenId, balance, "");
}

```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

contracts/RecoverableTokens.sol

Locations

```
*/
function __recoverERC1155(IERC1155 token, uint256 tokenId) internal {
uint256 balance = token.balanceOf(address(this), tokenId);
token.safeTransferFrom(address(this), payable(owner()), tokenId, balance, "");
}
```

Source file

contracts/MarebitsLockerAccount.sol

Locations

```
* @author Twifag
*/
contract MarebitsLockerAccount is Recoverable, KnowsBestPony, IMarebitsLockerAccount {
using Account for mapping(uint256 => Account.Info);
using Account for Account.Info;

/** @dev the maximum value for time (uint64) */
uint64 private constant MAXIMUM_TIME = (2 << 63) - 1;

/** @dev Stores all the accounts; `accountId` => {IMarebitsLockerAccount.Account} */
mapping(uint256 => Account.Info) private _accounts;

/**
* @dev The account must exist
* @param accountId (also `tokenId`) representing the locked account
*/
modifier accountExists(uint256 accountId) {
if (!_accounts[accountId].isDefined()) {
revert NonexistentAccount(accountId);
}
_;
}

/// @inheritdoc IMarebitsLockerAccount
function __createAccount(uint256 accountId, uint256 amount, address tokenContract, uint256 tokenId, Token.Type tokenType, uint64 unlockTime) public onlyOwner {
_accounts.create(accountId, amount, tokenContract, tokenId, tokenType, unlockTime);
}

/// @inheritdoc IMarebitsLockerAccount
function __burn(uint256 accountId) external onlyOwner accountExists(accountId) { _accounts[accountId].burn(); }

// /// @inheritdoc IMarebitsLockerAccount
function __redeem(uint256 accountId) external onlyOwner accountExists(accountId) { _accounts[accountId].redeem(); }

// /// @inheritdoc IMarebitsLockerAccount
// function __setAmount(uint256 accountId, uint256 amount) external onlyOwner accountExists(accountId) { _accounts[accountId].setAmount(amount); }

// /// @inheritdoc IMarebitsLockerAccount
function __setUnlockTime(uint256 accountId, uint64 unlockTime) external onlyOwner accountExists(accountId) { _accounts[accountId].setUnlockTime(unlockTime); }

/// @inheritdoc IMarebitsLockerAccount
function getAccount(uint256 accountId) external view accountExists(accountId) returns (Account.Info memory) { return _accounts[accountId]; }

// /// @inheritdoc IMarebitsLockerAccount
// function getAmount(uint256 accountId) external view accountExists(accountId) returns (uint256) { return _accounts[accountId].amount; }
```

```solidity

// /// @inheritdoc IMarebitsLockerAccount
// function getTokenContract(uint256 accountId) external view accountExists(accountId) returns (address) { return _accounts[accountId].tokenContract; }

// /// @inheritdoc IMarebitsLockerAccount
// function getTokenType(uint256 accountId) external view accountExists(accountId) returns (Token.Type) { return _accounts[accountId].tokenType; }

// /// @inheritdoc IMarebitsLockerAccount
// function getTokenId(uint256 accountId) external view accountExists(accountId) returns (uint256) { return _accounts[accountId].tokenId; }

// /// @inheritdoc IMarebitsLockerAccount
// function getUnlockTime(uint256 accountId) external view accountExists(accountId) returns (uint64) { return _accounts[accountId].unlockTime; }

// /// @inheritdoc IMarebitsLockerAccount
// function hasAccount(uint256 accountId) external view returns (bool) { return _accounts[accountId].isDefined(); }

// /// @inheritdoc IMarebitsLockerAccount
// function isUnlocked(uint256 accountId) external view accountExists(accountId) returns (bool) { return _accounts[accountId].isUnlocked(); }

/**
* @dev Implementation of the {IERC165} interface.
* @inheritdoc ERC165
*/
function supportsInterface(bytes4 interfaceId) public view virtual override(IERC165, Recoverable) returns (bool) {
return interfaceId == type(IMarebitsLockerAccount).interfaceId ||
interfaceId == type(KnowsBestPony).interfaceId ||
interfaceId == type(Recoverable).interfaceId ||
super.supportsInterface(interfaceId);
}
}
```

Analysis `f7190e6c-10db-4c2c-9937-30d7dcdc1f5e` MythX

Started	Thu Dec 09 2021 19:23:45 GMT+0000 (Coordinated Universal Time)
Finished	Thu Dec 09 2021 20:09:06 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/MarebitsLockerToken.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	2

ISSUES

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

contracts/RecoverableTokens.sol

Locations

```
* @param tokenId to recover
*/
function __recoverERC721(IERC721 token, uint256 tokenId) internal { token.safeTransferFrom(address(this), payable(owner()), tokenId); }

/// @inheritdoc IRecoverableTokens
```

Source file

node_modules/@openzeppelin/contracts/token/ERC721/ERC721.sol

Locations

```
*/
function getApproved(uint256 tokenId) public view virtual override returns (address) {
require(_exists(tokenId), "ERC721: approved query for nonexistent token");

return _tokenApprovals[tokenId];
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

contracts/RecoverableTokens.sol

Locations

```
*/
function __recoverERC1155(IERC1155 token, uint256 tokenId) internal {
uint256 balance = token.balanceOf(address(this), tokenId);
token.safeTransferFrom(address(this), payable(owner()), tokenId, balance, "");
}
```

Source file

contracts/MarebitsLockerToken.sol

Locations

```
* @author Twifag
*/
contract MarebitsLockerToken is Recoverable, ERC721Enumerable, KnowsBestPony, IMarebitsLockerToken {
using Strings for uint256;

/** @dev The base URI for computing the {ERC721#tokenURI}, roundabout way of overriding {ERC721#_baseURI} */
string private __baseURI;

/** @dev Optional mapping for image URIs */
// mapping(uint256 => string) private _imageURIs;

/** @dev uint256 to keep track of the tokens as they are created */
uint256 private _tokenIdTracker;

/** @dev Require that the caller is the same address as the owner of this contract's owner */
modifier onlyLockerOwner() {
address lockerOwner = IOwnable(owner()).owner();

if (lockerOwner != _msgSender()) {
revert NotLockerOwner(_msgSender(), lockerOwner);
}
_;
}

/**
* @param name of this token
* @param symbol of this token
* @param baseURI initially set for this token
*/
constructor(string memory name, string memory symbol, string memory baseURI) ERC721(name, symbol) { __baseURI = baseURI; }

/// @inheritdoc IMarebitsLockerToken
function __burn(uint256 tokenId) external onlyOwner { _burn(tokenId); }

/// @inheritdoc IMarebitsLockerToken
function __exists(uint256 tokenId) external view onlyOwner returns (bool) { return _exists(tokenId); }

/// @inheritdoc IMarebitsLockerToken
function __getNextTokenId() external onlyOwner returns (uint256 tokenId) { return _tokenIdTracker++; }

/// @inheritdoc IMarebitsLockerToken
function __issueToken(address payable owner, uint256 tokenId) external onlyOwner { _safeMint(owner, tokenId); }

/// @inheritdoc IMarebitsLockerToken
function __setBaseURI(string calldata baseURI) external onlyLockerOwner { __baseURI = baseURI; }

/** @return string the `__baseURI` */
function _baseURI() internal view override returns (string memory) { return __baseURI; }

/**
* @param path of the URI
* @param suffix of the URI
* @return string the generated URI by combining `__baseURI`, `path`, and `suffix` */
function _generateURI(string memory path, string memory suffix) private view returns (string memory) { return string(abi.encodePacked(_baseURI(), path, suffix)); }

/**
* @dev Marks the {IMarebitsLockerAccount.Account} as being burned and frees up storage, see {ERC721Enumerable}
* @inheritdoc ERC721
*/
function _burn(uint256 tokenId) internal override {
IMarebitsLocker(owner()).__burn(tokenId);
super._burn(tokenId);
}

/// @inheritdoc ERC721
function _mint(address to, uint256 tokenId) internal override {
emit URI(tokenURI(tokenId), tokenId);
super._mint(to, tokenId);
}

/// @inheritdoc IMarebitsLockerToken
function burn(uint256 tokenId) external {
if (!_isApprovedOrOwner(_msgSender(), tokenId)) {
revert NotApprovedOrOwner(tokenId);
}
_burn(tokenId);
}

/// @inheritdoc IMarebitsLockerToken
// function imageURI(uint256 tokenId) external view returns (string memory) {return _generateURI(tokenId.toString(), ".svg"); }

/**
* @dev Implementation of the {IERC165} interface.
* @inheritdoc ERC165
*/
function supportsInterface(bytes4 interfaceId) public view virtual override(IERC165, ERC721Enumerable, Recoverable) returns (bool) {
return interfaceId == type(IMarebitsLockerToken).interfaceId ||
interfaceId == type(KnowsBestPony).interfaceId ||
interfaceId == type(Recoverable).interfaceId ||
interfaceId == type(IERC721Metadata).interfaceId ||
ERC721Enumerable.supportsInterface(interfaceId) ||
Recoverable.supportsInterface(interfaceId);
}

/// @inheritdoc ERC721
function tokenURI(uint256 tokenId) public view override(ERC721, IERC721Metadata) returns (string memory tokenUri) { tokenUri = _generateURI(tokenId.toString(), ".json"); }

/// @inheritdoc IMarebitsLockerToken
// function uri(uint256 tokenId) external view returns (string memory) { return tokenURI(tokenId); }
}
```

Analysis cd0119b6-d36f-4fa5-865b-a31fbdd3e78a

MythX

Started	Thu Dec 09 2021 21:55:22 GMT+0000 (Coordinated Universal Time)
Finished	Thu Dec 09 2021 22:40:25 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/MarebitsVault.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	3

ISSUES

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/MarebitsVault.sol

Locations

```
* @param amount of tokens to be transferred
*/
function _transferERC1155(IERC1155 token, address payable to, uint256 tokenId, uint256 amount) private { token.safeTransferFrom(address(this), to, tokenId, amount, ""); }

/**
```

LOW

SWC-107

A call to a user-supplied address is executed.

An external message call to an address specified by the caller is executed. Note that the callee account might contain arbitrary code and could re-enter any function within this contract. Reentering the contract in an intermediate state may lead to unexpected behaviour. Make sure that no state modifications are executed after this call and/or reentrancy guards are in place.

Source file

contracts/MarebitsVault.sol

Locations

```
* @param tokenId of the token to be transferred
*/
function _transferERC721(IERC721 token, address payable to, uint256 tokenId) private { token.safeTransferFrom(address(this), to, tokenId); }

/**
```

LOW

SWC-123

Requirement violation.

A requirement was violated in a nested call and the call was reverted as a result. Make sure valid inputs are provided to the nested call (for instance, via passed arguments).

Source file

`contracts/MarebitsVault.sol`

Locations

```
* @param tokenId of the token to be transferred
*/
function _transferERC721(IERC721 token, address payable to, uint256 tokenId) private { token.safeTransferFrom(address(this), to, tokenId); }

/**
```

Source file

`contracts/MarebitsVault.sol`

Locations

```
* @author Twifag
*/
contract MarebitsVault is ERC1155Holder, ERC721Holder, KnowsBestPony, RecoverableEther, IMarebitsVault {
using SafeERC20 for IERC20;

/// @inheritdoc IMarebitsVault
function __transfer(Token.Type tokenType, address tokenContract, address payable to, uint256 tokenId, uint256 amount) external onlyOwner {
if (tokenType == Token.Type.ERC1155) {
_transferERC1155(IERC1155(tokenContract), to, tokenId, amount);
} else if (tokenType == Token.Type.ERC20) {
_transferERC20(IERC20(tokenContract), to, amount);
} else if (tokenType == Token.Type.ERC721) {
_transferERC721(IERC721(tokenContract), to, tokenId);
}
}

/**
* @dev Transfers ERC1155 tokens out of this contract and to the original owner
* @param token to be transferred
* @param to wallet address
* @param tokenId of the token to be transferred
* @param amount of tokens to be transferred
*/
function _transferERC1155(IERC1155 token, address payable to, uint256 tokenId, uint256 amount) private { token.safeTransferFrom(address(this), to, tokenId, amount, ""); }

/**
* @dev Transfers ERC20 tokens out of this contract and to the original owner
* @param token to be transferred
* @param to wallet address
* @param amount of tokens to be transferred
*/
function _transferERC20(IERC20 token, address payable to, uint256 amount) private { token.safeTransfer(to, amount); }

/**
* @dev Transfers ERC721 tokens out of this contract and to the original owner
* @param token to be transferred
* @param to wallet address
* @param tokenId of the token to be transferred
*/
function _transferERC721(IERC721 token, address payable to, uint256 tokenId) private { token.safeTransferFrom(address(this), to, tokenId); }

/**
* @dev Implementation of the {IERC165} interface.
* @inheritdoc ERC165
*/
function supportsInterface(bytes4 interfaceId) public view virtual override(ERC1155Receiver, IERC165, RecoverableEther) returns (bool) {
return interfaceId == type(IERC721Receiver).interfaceId ||
interfaceId == type(KnowsBestPony).interfaceId ||
interfaceId == type(RecoverableEther).interfaceId ||
ERC1155Receiver.supportsInterface(interfaceId) ||
RecoverableEther.supportsInterface(interfaceId);
}
}
```