MythX

REPORT 6029650E2BFEB700194AE5C3

Created	Sun Feb 14 2021 17:59:42 GMT+0000 (Coordinated Universal Time)
Number of analyses	1
User	bl33234679@gmail.com

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
bdba23fb-f24c-4a27-bd39-e39e9bc8c09a	TokenFactory.sol	8

Analysis bdba23fb-f24c-4a27-bd39-e39e9bc8c09a MythX

Started	Sun Feb 14 2021 17:59:51 GMT+0000 (Coordinated Universal Time)
Finished	Sun Feb 14 2021 18:45:44 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	TokenFactory.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	1	7

ISSUES

MEDIUM

Multiple calls are executed in the same transaction.

SWC-113

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

ERC1155ERC721WithAdapter.sol

Locations

```
109 | {
110 | address adapter = _createClone(template);
111 | ERC20Adapter(adapter).initialize(_tokenId);
112 | _adapters[_tokenId] = adapter;
113 | emit NewAdapter(_tokenId, adapter);
```

LOW

Function visibility is not set.

SWC-100

The function definition of "null" lacks a visibility specifier. Note that the compiler assumes "public" visibility by default. Function visibility should always be specified explicitly to assure correctness of the code and improve readability.

Source file

TokenFactory.sol

Locations

```
14 | BaseRelayRecipient
15 | {
16 | constructor (address _trustedForwarder) {
17 | trustedForwarder = _trustedForwarder;
18 | }
19 |
20 | /// @notice Query if a contract implements an interface
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

ERC1155ERC721WithAdapter.sol

Locations

```
110 address adapter = _createClone(template);
111 ERC20Adapter(adapter).initialize(_tokenId);
112 _adapters[_tokenId] = adapter;
113 emit NewAdapter(_tokenId, adapter);
114 }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

ERC1155ERC721WithAdapter.sol

Locations

```
108 internal
109 {
110 address adapter = _createClone(template);
111 ERC20Adapter(adapter).initialize(_tokenId);
112 _adapters[_tokenId] = adapter;
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

ERC1155ERC721.sol

Locations

```
685 internal
686 {
687 _recordingBalances[_recordingOperator][_tokenId] += _supply;
688 _recordingOperators[_tokenId] = _recordingOperator;
689 emit RecordingTransferSingle(_msgSender(), address(0), _recordingOperator, _tokenId, _supply);
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

ERC1155ERC721.sol

Locations

```
686 {
687 _recordingBalances[_recordingOperator][_tokenId] += _supply;
688 _recordingOperators[_tokenId] = _recordingOperator;
689 emit RecordingTransferSingle(_msgSender(), address(0), _recordingOperator, _tokenId, _supply);
690 }
```

LOW

SWC-107

Read of persistent state following external call.

The contract account state is accessed after an external call. To prevent reentrancy issues, consider accessing the state only before the call, especially if the callee is untrusted. Alternatively, a reentrancy lock can be used to prevent untrusted callees from re-entering the contract in an intermediate state.

Source file

GSN/BaseRelayRecipient.sol

Locations

```
18
19 function isTrustedForwarder(address forwarder) public override view returns(bool) {
20 return forwarder == trustedForwarder;
21 }
```

LOW

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).

SWC-123

Source file

ERC1155ERC721.sol

Locations

```
762 returns (bool)
763 {
764 return (IERC1155TokenReceiver(_to).onERC1155BatchReceived(_operator, _from, _tokenIds, _values, _data)
765 == ERC1155_BATCH_ACCEPTED);
766 }
```

Source file

TokenFactory.sol

Locations

```
8  import "./GSN/BaseRelayRecipient.sol";
9
10 contract TokenFactory is
11 ITokenFactory,
12 ERC1155ERC721Metadata,
13 ERC1155ERC721WithAdapter,
14 BaseRelayRecipient
15 {
16 constructor (address _trustedForwarder) {
17 trustedForwarder = _trustedForwarder;
18 }
19
20 /// @notice Query if a contract implements an interface
21 /// @param _interfaceId The interface identifier, as specified in ERC-165
22 /// @dev Interface identification is specified in ERC-165. This function
23 /// uses less than 30,000 gas.
24 /// @return `true` if the contract implements `_interfaceId`,
25 /// `false` otherwise
26 function supportsInterface(bytes4 _interfaceId)
27 public
28 pure
29 override(ERC1155ERC721Metadata, ERC1155ERC721)
30 returns (bool)
31 {
32 return super.supportsInterface(_interfaceId);
33 }
34
35 /// @notice Queries accumulated holding time for a given owner and token
36 /// @dev It throws if it's not a need-time token. The way how holding time is
37 /// calculated is by suming up (token amount) * (holding time in second)
38 /// @param _owner Address to be queried
39 /// @param _tokenId Token ID of the token to be queried
40 /// @return Holding time
41 function holdingTimeOf(
42 address _owner,
43 uint256 _tokenId
44 )
45 external
46 view
47 override
48 returns (uint256)
49 {
50 require(_tokenId & NEED_TIME > 0, "Doesn't support this token");
51
52 return _holdingTime[_owner][_tokenId] + _calcHoldingTime(_owner, _tokenId);
```

```
}

/// @notice Queries accumulated holding time for a given owner and recording token
/// @dev It throws if it's not a need-time token. The way how holding time is
/// calculated is by suming up (token amount) * (holding time in second)
/// @dev It returns zero if it doesn't have a corresponding recording token
/// @param _owner Address to be queried
/// @param _tokenId Token ID of the token to be queried
/// @return Holding time
function recordingHoldingTimeOf(
address _owner,
uint256 _tokenId
)
external
view
override
returns (uint256)
{
return _recordingHoldingTime[_owner][_tokenId] + _calcRecordingHoldingTime(_owner, _tokenId);
}

/// @notice Create a token without setting uri
/// @param _supply The amount of token to create
/// @param _receiver Address that receives minted token
/// @param _settingOperator Address that can perform setTimeInterval
/// and set ERC20 Attribute
/// @param _needTime Set to `true` if need to query holding time for token
/// @param _erc20 Set to `true` to create a erc20 adapter for token
/// @return Token ID
function createToken(
uint256 _supply,
address _receiver,
address _settingOperator,
bool _needTime,
bool _erc20
)
public
override
returns (uint256)
{
uint256 tokenId = _mint(_supply, _receiver, _settingOperator, _needTime, "");
if (_erc20)
_createAdapter(tokenId);
return tokenId;
}

/// @notice Create a token with uri
/// @param _supply The amount of token to create
/// @param _receiver Address that receives minted token
/// @param _settingOperator Address that can perform setTimeInterval
/// and set ERC20 Attribute
/// @param _needTime Set to `true` if need to query holding time for token
/// @param _uri URI that points to token metadata
/// @param _erc20 Set to `true` to create a erc20 adapter for token
/// @return Token ID
function createToken(
uint256 _supply,
address _receiver,
address _settingOperator,
bool _needTime,
string calldata _uri,
bool _erc20
)
```

```
external
override
returns (uint256)
{
uint256 tokenId = createToken(_supply, _receiver, _settingOperator, _needTime, _erc20);
if (_erc20)
_createAdapter(tokenId);
_setTokenURI(tokenId, _uri);
return tokenId;
}

/// @notice Create both normal token and recording token without setting uri
/// @dev Recording token shares the same token ID with normal token
/// @param _supply The amount of token to create
/// @param _receiver Address that receives minted token
/// @param _settingOperator Address that can perform setTimeInterval
/// and set ERC20 Attribute
/// @param _needTime Set to `true` if need to query holding time for token
/// @param _recordingOperator Address that can manage recording token
/// @param _erc20 Set to `true` to create a erc20 adapter for token
/// @return Token ID
function createTokenWithRecording(
uint256 _supply,
address _receiver,
address _settingOperator,
bool _needTime,
address _recordingOperator,
bool _erc20
)
public
override
returns (uint256)
{
uint256 tokenId = createToken(_supply, _receiver, _settingOperator, _needTime, _erc20);
if (_erc20)
_createAdapter(tokenId);
_mintCopy(tokenId, _supply, _recordingOperator);
return tokenId;
}

/// @notice Create both normal token and recording token with uri
/// @dev Recording token shares the same token ID with normal token
/// @param _supply The amount of token to create
/// @param _receiver Address that receives minted token
/// @param _settingOperator Address that can perform setTimeInterval
/// and set ERC20 Attribute
/// @param _needTime Set to `true` if need to query holding time for token
/// @param _recordingOperator Address that can manage recording token
/// @param _uri URI that points to token metadata
/// @param _erc20 Set to `true` to create a erc20 adapter for token
/// @return Token ID
function createTokenWithRecording(
uint256 _supply,
address _receiver,
address _settingOperator,
bool _needTime,
address _recordingOperator,
string calldata _uri,
bool _erc20
)
external
override
returns (uint256)
```

```
{
uint256 tokenId = createToken(_supply, _receiver, _settingOperator, _needTime, _erc20);
if (_erc20)
_createAdapter(tokenId);
_mintCopy(tokenId, _supply, _recordingOperator);
_setTokenURI(tokenId, _uri);
return 0;
}

/// @notice Set starting time and ending time for token holding time calculation
/// @dev Starting time must be greater than time at the moment
/// @dev To save gas cost, here use uint128 to store time
/// @param _startTime Starting time in unix time format
/// @param _endTime Ending time in unix time format
function setTimeInterval(
uint256 _tokenId,
uint128 _startTime,
uint128 _endTime
)
external
override
{
require(_msgSender() == _settingOperators[_tokenId], "Not authorized");
require(_startTime >= block.timestamp, "Time smaller than now");
require(_endTime > _startTime, "End greater than start");
require(_timeInterval[_tokenId] == 0, "Already set");

_setTime(_tokenId, _startTime, _endTime);
}

/// @notice Set erc20 token attribute
/// @dev Throws if `msg.sender` is not authorized setting operator
/// @param _tokenId Corresponding token ID with erc20 adapter
/// @param _name Name of the token
/// @param _symbol Symbol of the token
/// @param _decimals Number of decimals to use
function setERC20Attribute(
uint256 _tokenId,
string memory _name,
string memory _symbol,
uint8 _decimals
)
external
override
{
require(_msgSender() == _settingOperators[_tokenId], "Not authorized");
require(_adapters[_tokenId] != address(0), "No adapter found");

_setERC20Attribute(_tokenId, _name, _symbol, _decimals);
}

function _transferFrom(
address _from,
address _to,
uint256 _tokenId,
uint256 _value
)
internal
override(ERC1155ERC721, ERC1155ERC721WithAdapter)
{
super._transferFrom(_from, _to, _tokenId, _value);
}

```

```
function versionRecipient()
external
override
virtual
view
returns (string memory)
{
return "2.1.0";
}

function _msgSender()
internal
override(Context, BaseRelayRecipient)
view
returns (address payable)
{
return BaseRelayRecipient._msgSender();
}

function _msgData()
internal
override(Context, BaseRelayRecipient)
view
returns (bytes memory)
{
return BaseRelayRecipient._msgData();
}
}
```