Snag Solutions

Security Review

16th September, 2023

Review Conducted by:
Lyuboslav Bardenski, Smart Contract Security Researcher

Disclaimer

This smart contract security review does not bring any additional responsibilities for the group of people who have conducted it. The goal of the smart contract security review is to find possible risks and vulnerabilities, and provide useful recommendations in terms of security and optimization.

Severity classification

Severity	**Impact: High**	**Impact: Medium**	**Impact: Low**
Likelihood: High	Critical	High	Medium
Likelihood: Medium	High	Medium	Low
Likelihood: Low	Medium	Low	Low

Impact
- High - leads to a significant loss of assets in the protocol or significantly harms a group of users.
- Medium - only a small amount of funds can be lost or a functionality of the protocol is affected.
- Low - any kind of unexpected behavior that's not so critical.

Likelihood
- High - direct attack vector; the cost is relatively low to the amount of funds that can be lost.
- Medium - only conditionally incentivized attack vector, but still relatively likely.
- Low - too many or too unlikely assumptions, provides little or no incentive.

Actions required by severity level
- Critical - client must fix the issue.
- High - client must fix the issue.
- Medium - client should fix the issue.
- Low - client could fix the issue.

Security Assessment Summary

Name	Snag-Solutions
Repository	https://github.com/Snag-Solutions/contracts
Review Commit Hash	d7bc906bc18478298573045409b3332ee33f5e93
Resolution Commit Hash	-

Scope

The following smart contracts were in scope of the audit:

- *contracts/SnagMarketHybridV1.sol*

Critical	1
High	0
Medium	0
Low	2
Informational	6

Detailed Findings

This section provides a detailed description of the vulnerabilities identified within the Snag Solutions smart contracts.

Each vulnerability has a severity classification which is determined from the likelihood and impact of each issue by the matrix given in the Appendix: Severity Classification.

A number of additional properties of the contracts, including gas optimisations, are also described in this section and are labeled as “informational”.

Each vulnerability is also assigned a status:

• Open: the issue has not been addressed by the project team.

• Resolved: the issue was acknowledged by the project team and updates to the affected contract(s) have been made to mitigate the related risk.

• Partially Resolved: the issue was acknowledged by the project team but not the entire vulnerability was fixed.

Findings Summary

ID	Description	Severity
C-01	purchaseItems() can be frontrun and signature can be used by malicious actor	CRITICAL
L-01	publicKey can be set to zero address	LOW
L-2	Specify concrete compiler version	LOW
I-01	Redundant usage of SafeMath	INFORMATIONAL
I-02	Pack Item struct to optimize storage usage	INFORMATIONAL
I-03	Rename currentAssumedOwner to owner in Item struct	INFORMATIONAL
I-04	Pack into one require statement both Items length validations	INFORMATIONAL
I-05	tokenOwner is not needed to be checked that it is the _msgSender()	INFORMATIONAL
I-06	tokenOwner is not needed to be checked that it is the currentAssumedOwner	INFORMATIONAL

C-01 purchaseItems() can be frontran and signature can be used by malicious actor

SEVERITY | CRITICAL
LIKELIHOOD | HIGH
IMPACT | HIGH

Description

The function purchaseItems() uses typed structured data which only includes the ***orderIds*** argument whereas for the execution of the function the ***callTypes*** and the ***Items*** are needed. This creates a scenario for easy exploitation by a malicious actor.

Poc

1. Alice follows the transactions related to purchaseItems() function from the mempool.
2. The publicKey executes a transaction with a signature which only includes the ***orderIds*** parameter.
3. Alice sees the transaction, extracts the signature and initiates a new transaction with CallType other than ACCEPT_OFFER and particular Item parameters..

   As a result Alice can transfer to another address each tokenId which has been approved for the contract.

Recommendations

There are three possible solutions for this vulnerability:

1. Follow ***EIP712*** standard and include all of the function arguments in typed data structure. All the domain parameters must be included, as also a nonce must be added in order to guarantee that signature won't be used again.

   In order to recover the signature use the correct function from the OpenZeppelin's ***ECDSA*** library.

2. Remove signature verification and create access control modifier in the purchaseItems() function to be only callable by the ***publicKey***.

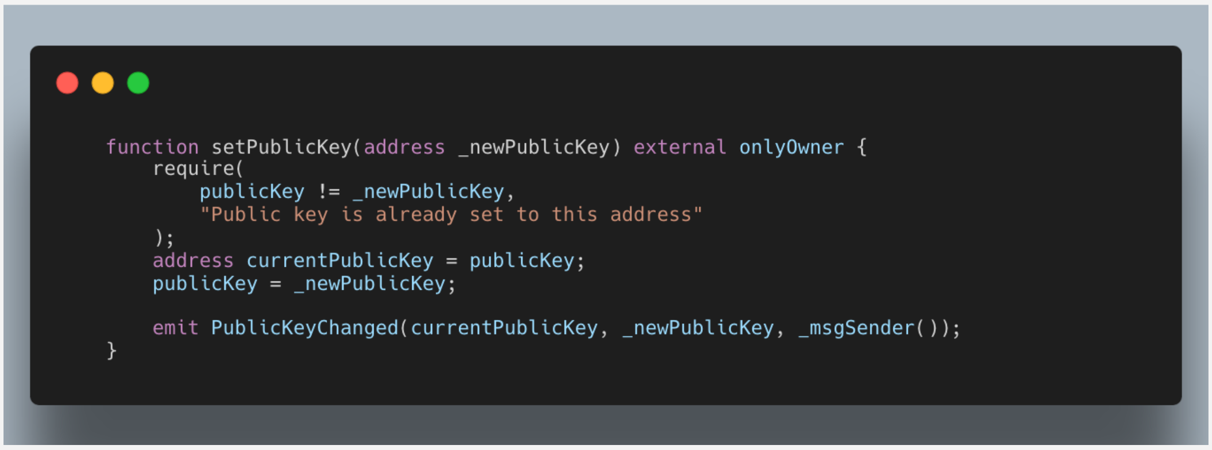
3. Add access control modifier in order to make the function only executable by the ***publicKey***.

L-01 publicKey can be set to zero address

SEVERITY | LOW
Likelihood | LOW
Impact | LOW

Description

setPublicKey() allows setting zero address to be set as publicKey which will make the ***purchaseItems()*** unusable.


```
function setPublicKey(address _newPublicKey) external onlyOwner {
    require(
        publicKey != _newPublicKey,
        "Public key is already set to this address"
    );
    address currentPublicKey = publicKey;
    publicKey = _newPublicKey;

    emit PublicKeyChanged(currentPublicKey, _newPublicKey, _msgSender());
}
```

Recommendations

Create validation to check if _newPublicKey is zero address.

L-02 Specify concrete compiler version

SEVERITY | LOW
Likelihood | LOW
Impact | LOW

Description

Some Solidity compiler versions are vulnerable to bugs and it is a good practice to use the last version. For example Solidity versions 0.8.13 and 0.8.14 are vulnerable to an optimizer bug related to inline assembly. Solidity 0.8.15 has been released with a fix. Due to the simplicity of the contract and the fact that 0.8.0 version and above are relatively stable, the severity is classified as Low.

Recommendations

Use a specific Solidity compiler version.

I-01 Redundant usage of SafeMath

Description

SafeMath is not needed to be imported as it is by default provided from Solidity compiler versions from 0.8.0.

Recommendations

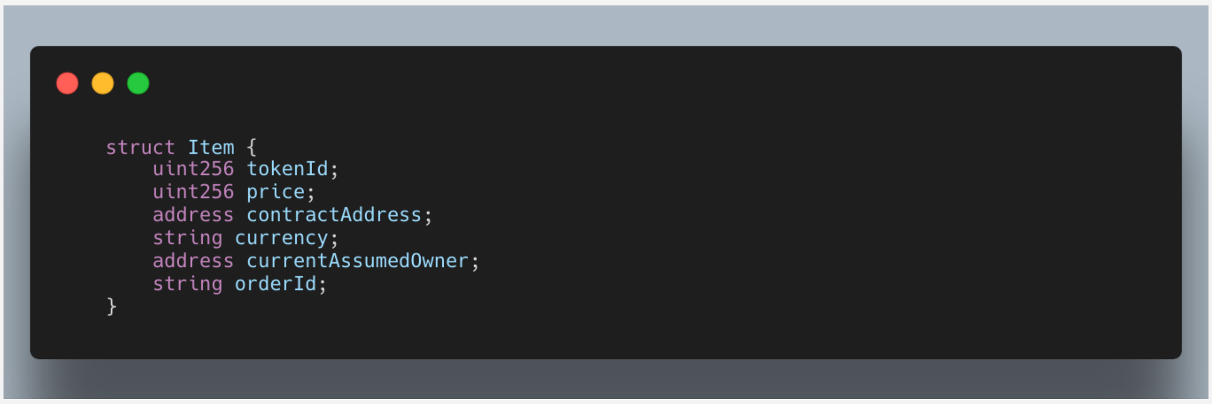
Remove the import and usage of SafeMath.

I-02 Pack Item struct to optimize storage usage

Description

It is possible to store as many variables into one storage slot, as long as the combined storage requirement is equal to or less than the size of one storage slot, which is 32 bytes. For example, one bool variable takes up one byte. A uint8 is one byte as well, uint16 is two bytes, uint32 four bytes, and so on.

Recommendations



```
struct Item {
    uint256 tokenId;
    uint256 price;
    address contractAddress;
    string currency;
    address currentAssumedOwner;
    string orderId;
}
```

I-03 Rename currentAssumedOwner to owner in Item struct

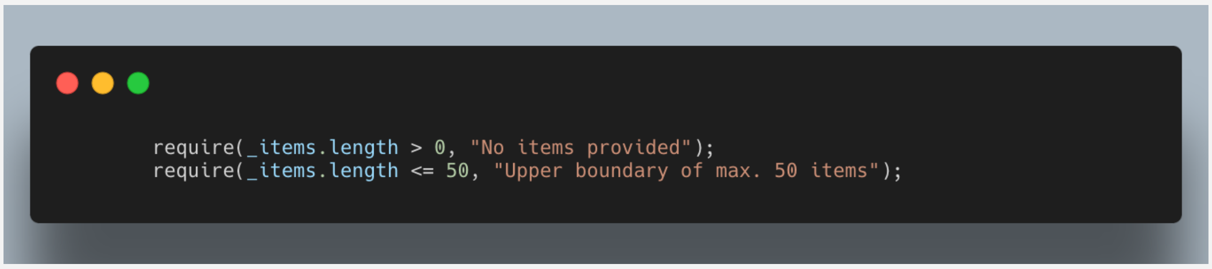
Description

Renaming currentAssumedOwner in Item struct to owner will be better in terms of readability.

I-04 Pack into one require statement both Items length validations

Description

Pack the following require statements into one in order to save from gas.



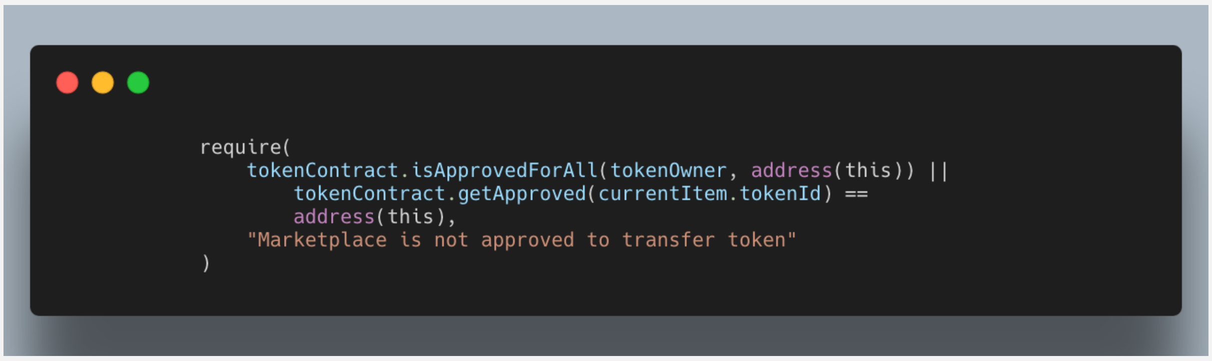
```
require(_items.length > 0, "No items provided");
require(_items.length <= 50, "Upper boundary of max. 50 items");
```

I-05 tokenOwner is not needed to be checked that it is the _msgSender()

Description

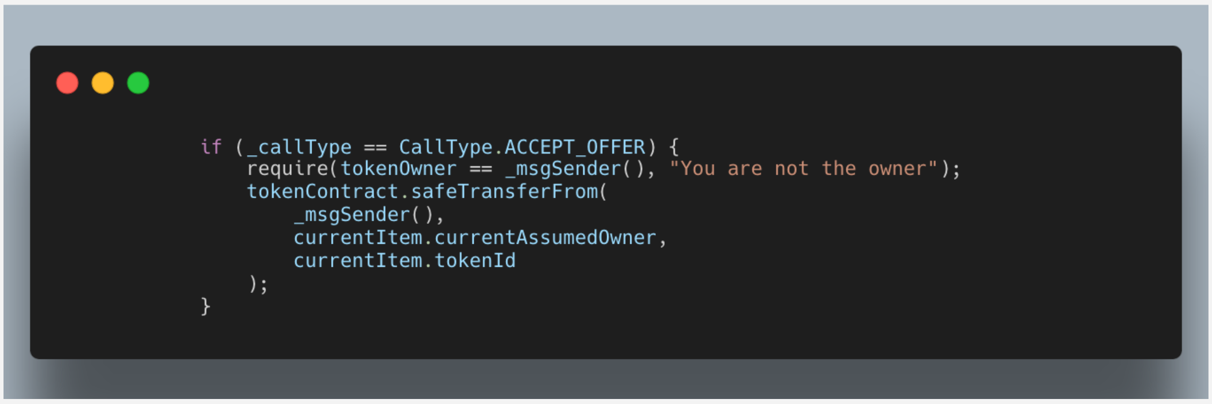
The check on line 113 in purchaseItems() function doesn't add additional security for the contract as the transfer will effectively fail if the _msgSender() is not the owner .

lines 105-110



```
require(
    tokenContract.isApprovedForAll(tokenOwner, address(this)) ||
        tokenContract.getApproved(currentItem.tokenId) ==
        address(this),
    "Marketplace is not approved to transfer token"
)
```

lines 112-119

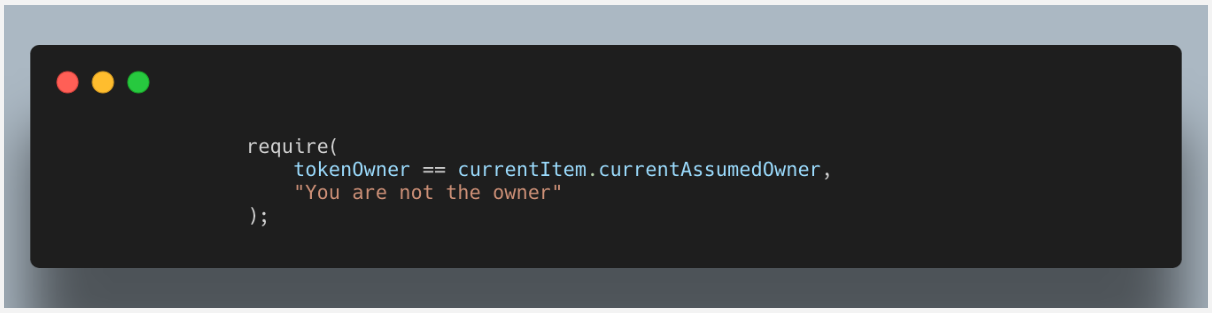


```
if (_callType == CallType.ACCEPT_OFFER) {
    require(tokenOwner == _msgSender(), "You are not the owner");
    tokenContract.safeTransferFrom(
        _msgSender(),
        currentItem.currentAssumedOwner,
        currentItem.tokenId
    );
}
```

I-06 tokenOwner is not needed to be checked that it is the currentAssumedOwner

Description

The check on line 121 in purchaseItems() function is unneeded as the transfer will effectively fail if the currentAssumedOwner is not the owner .



```
    require(
        tokenOwner == currentItem.currentAssumedOwner,
        "You are not the owner"
    );
```