

SMART CONTRACT AUDIT REPORT

For

ERC1155NonFungible (Order #FO711A5C5)

Prepared By: Yogesh Padsala

Prepared For: digipeso.exchange

Prepared on: 19/08/2018

Table of Content

1. Disclaimer
2. Overview of the audit
3. Attacks made to the contract
4. Good things in smart contract
5. Critical vulnerabilities found in the contract
6. Medium vulnerabilities found in the contract
7. Low severity vulnerabilities found in the contract
8. Summary of the audit

1. Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

2. Overview of the audit

The project has 1 file HadaFinal1155.sol. It contains approx 530 lines of Solidity code. All the functions and state variables are well commented using the natspec documentation, but that does not create any vulnerability.

3. Attacks made to the contract

In order to check for the security of the contract, we tested several attacks in order to make sure that the contract is secure and follows best practices.

3.1: Over and under flows

An overflow happens when the limit of the type variable uint256, 2^{256} , is exceeded. What happens is that the value resets to zero instead of incrementing more. On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0. For example, if you subtract $0 - 1$ the result will be $= 2^{256}$ instead of -1. This is quite dangerous.

This contract **does not** check for overflows and underflows by using OpenZeppelin's SafeMath to mitigate this attack, however, we noticed some place where it can be improved (discussed below).

3.2: Short address attack

If the token contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeros to the transaction until the address is complete.

Although this contract **is not vulnerable** to this attack, but there are some point where users can mess themselves due to this (Please see below). It is highly recommended to call functions after checking validity of the address.

3.3: Visibility & Delegatecall

It is also known as, The Parity Hack, which occurs while misuse of Delegatecall.

No such issues found in this smart contract and visibility also properly addressed. There are some places where there is no visibility defined. Smart Contract will assume “Public” visibility if there is no visibility defined. It is good practice to explicitly define the visibility, but again, the contract is not prone to any vulnerability due to this in this case.

3.4: Reentrancy / TheDAO hack

Reentrancy occurs in this case: any interaction from a contract (A) with another contract (B) and any transfer of Ether hands over control to that contract (B). This makes it possible for B to call back into A before this interaction is completed.

Use of “require” function in this smart contract **mitigated** this vulnerability.

3.5: Forcing ether to a contract

While implementing “selfdestruct” in smart contract, it sends all the ether to the target address. Now, if the target address is a contract address, then the fallback function of target contract does not get called. And thus Hacker can bypass the “Required” conditions. Here, the Smart Contract’s balance has never been used as guard, which **mitigated** this vulnerability.

4. Good things in smart contract

4.1 transferFrom Function:-

```
137
138 function transferFrom(address _from, address _to, uint256 _id, uint256 _value) external {
139     if(_from != msg.sender) {
140         require(allowances[_id][_from][msg.sender] >= _value);
141         allowances[_id][_from][msg.sender] = allowances[_id][_from][msg.sender].sub(_value);
142     }
143
144     items[_id].balances[_from] = items[_id].balances[_from].sub(_value);
145     items[_id].balances[_to] = _value.add(items[_id].balances[_to]);
146
147     emit Transfer(msg.sender, _from, _to, _id, _value);
148 }
149
```

- Here you are checking address parameter “_from” is not the same person who is calling the function and then processing the allowance.

4.2 Approve Function

```
154 function approve(address _spender, uint256 _id, uint256 _currentValue, uint256 _value) external {
155     // if the allowance isn't 0, it can only be updated to 0 to prevent an allowance change immediately after
156     require(_value == 0 || allowances[_id][msg.sender][_spender] == _currentValue);
157     allowances[_id][msg.sender][_spender] = _value;
158     emit Approval(msg.sender, _spender, _id, _currentValue, _value);
159 }
160
```

- Allowance either must be zero, or it must be the currentValue to proceed to setup new allowance.

5. Critical vulnerabilities found in the contract

=> No critical vulnerabilities found

6. Medium vulnerabilities found in the contract

=> No Medium vulnerabilities found

7. Low severity vulnerabilities found

7.1 Issues in approve function

=> This is not a big issue and your contract is **not susceptible** to any risks because you are checking balance and allowance in every function.

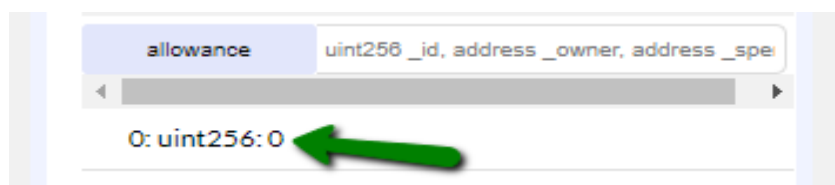
=>But it is good practice to check balance in approve function.

=>So here, negative value also gets accepted in this function for allowance. So allowance of any user goes wrong.

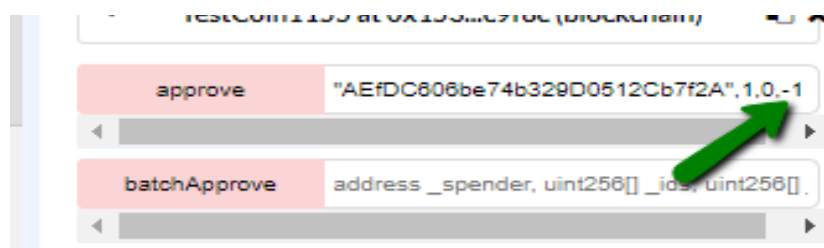
```
154
155
156 function approve(address _spender, uint256 _id, uint256 _currentValue, uint256 _value) external {
157     // if the allowance isn't 0, it can only be updated to 0 to prevent an allowance change immediately af
158     require(_value == 0 || allowances[_id][msg.sender][_spender] == _currentValue);
159     allowances[_id][msg.sender][_spender] = _value;
160     emit Approval(msg.sender, _spender, _id, _currentValue, _value);
161 }
```

- **Underflow and overflow issue**

- Allowance before approve



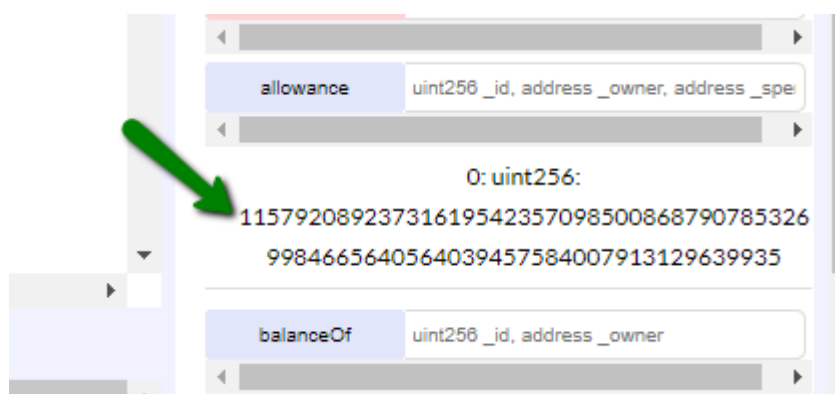
- Calling approve with negative value.



- Transaction hash

- <https://rinkeby.etherscan.io/tx/0x9135ac085c15dd784a27808e3afe589bf4713e75aa55dab144a9e881b00e9be3>

- Allowance after approve



Solution:-

```
74 function approve(address _spender, uint256 _value) returns (bool success) {  
75     allowed[msg.sender][_spender] = _value;  
76     Approval(msg.sender, _spender, _value);  
77     return true;  
78 }  
79
```

- In approve function you have to put one condition.

require(_value <= items[_id].balances[msg.sender]);

- By this way, user only approves the amount which he has in the balance.

7.2: Compiler version not fixed

=> In this file you have put "pragma solidity ^0.4.24;" which is not good way to define compiler version.

=> Solidity source files indicate the versions of the compiler they can be compiled with.

pragma solidity ^0.4.24; // bad: compiles w 0.4.24 and above

pragma solidity 0.4.24; // good : compiles w 0.4.24 only

=> If you put (^) symbol then you are able to get compiler version 0.4.24 and above. But if you don't use (^) symbol then you are able to use only 0.4.24 version. And if there is some changes come in compiler and you use old version then some issue may come at deploy time.

7.3 Short address Possibility

=> This is not a big issue in the solidity, because nowadays security is increased in new solidity version. But it is good practice to check for the short addresses.

=> In some functions you are not checking value of address parameter

- **function:-batchtransferFrom('_from','_to')**

```

185
186 ▾ function batchTransferFrom(address _from, address _to, uint256[] _ids, uint256[] _values) external {
187     uint256 _id;
188     uint256 _value;
189
190 ▾     if(_from == msg.sender) {
191 ▾         for (uint256 i = 0; i < _ids.length; ++i) {
192             _id = _ids[i];
193             _value = _values[i];
194
195             items[_id].balances[_from] = items[_id].balances[_from].sub(_value);
196             items[_id].balances[_to] = _value.add(items[_id].balances[_to]);
197
198             emit Transfer(msg.sender, _from, _to, _id, _value);
199         }
200     }
201 ▾     else {

```

- **Solution:-**

- require(_from != address(0));
- require(_to != address(0));

- **Function:-approve('_spender')**

```

155 ▾ function approve(address _spender, uint256 _id, uint256 _currentValue, uint256 _value) external {
156     // if the allowance isn't 0, it can only be updated to 0 to prevent an allowance change immediate
157     require(_value == 0 || allowances[_id][msg.sender][_spender] == _currentValue);
158     allowances[_id][msg.sender][_spender] = _value;
159     emit Approval(msg.sender, _spender, _id, _currentValue, _value);
160 }
161

```

- **Solution:-**

require(_spender != address(0));

- **Function:- batchApprove('_spender')**

```

220 ▾ function batchApprove(address _spender, uint256[] _ids, uint256[] _currentValues, uint256[] _values) exter
221     uint256 _id;
222     uint256 _value;
223
224 ▾     for (uint256 i = 0; i < _ids.length; ++i) {
225         _id = _ids[i];
226         _value = _values[i];
227
228         require(_value == 0 || allowances[_id][msg.sender][_spender] == _currentValues[i]);
229         allowances[_id][msg.sender][_spender] = _value;
230         emit Approval(msg.sender, _spender, _id, _currentValues[i], _value);
231     }
232 }

```

- **Solution:-**

require(_spender != address(0));

- **Function:- batchtransfer('_to')**

```
236 ▾ function batchTransfer(address _to, uint256[] _ids, uint256[] _values) external {
237     uint256 _id;
238     uint256 _value;
239
240 ▾     for (uint256 i = 0; i < _ids.length; ++i) {
241         _id = _ids[i];
242         _value = _values[i];
243
244         items[_id].balances[msg.sender] = items[_id].balances[msg.sender].sub(_value);
245         items[_id].balances[_to] = _value.add(items[_id].balances[_to]);
246
247         emit Transfer(msg.sender, msg.sender, _to, _id, _value);
248     }
249 }
```

- **Solution:-**

require(_to != address(0));

- **Function:- Transfer('_to')**

```
171
172 ▾ function transfer(address _to, uint256 _id, uint256 _value) external {
173     // Not needed. SafeMath will do the same check on .sub(_value)
174     //require(_value <= items[_id].balances[msg.sender]);
175     items[_id].balances[msg.sender] = items[_id].balances[msg.sender].sub(_value);
176     items[_id].balances[_to] = _value.add(items[_id].balances[_to]);
177     emit Transfer(msg.sender, msg.sender, _to, _id, _value);
178 }
```

- **Solution:-**

require(_to != address(0));

- **Function:- TransferFrom('_from','_to')**

```
405
406 ▾ function transferFrom(address _from, address _to, uint256[] _ids, uint256[] _values) external {
407
408     uint256 _id;
409     uint256 _value;
410
411 ▾     for (uint256 i = 0; i < _ids.length; ++i) {
412         _id = _ids[i];
413         _value = _values[i];
414
415 ▾         if (isNonFungible(_id)) {
```

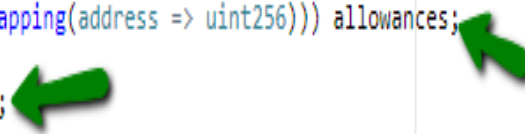
- **Solution:-**

require(_to != address(0));

7.4 Implicit visibility level

=> This is again not a big issue in the solidity. Because if you do not put any visibility, then it will automatically take “public”. But it is good practice to specify visibility at every variables and functions.

```
127 mapping (uint256 => string) public symbols;  
128 mapping (uint256 => mapping(address => mapping(address => uint256))) allowances;  
129 mapping (uint256 => Items) public items;  
130 mapping (uint256 => string) metadataURIs;
```



- For #128
 - mapping (uint256 => mapping(address => mapping(address => uint256))) public allowances;
- For #130
 - mapping (uint256 => string) public metadataURIs;

```
315  
316 mapping (uint256 => address) nfiOwners;  
317
```

- For #316
 - mapping (uint256 => address) public nfiOwners;

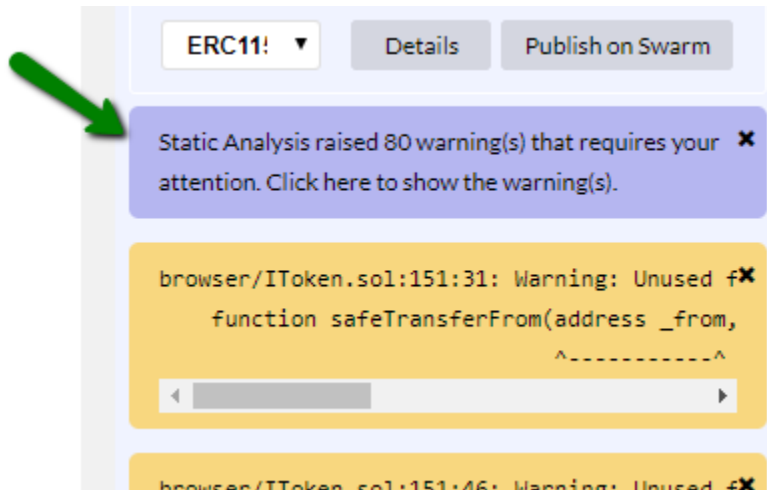
```
460 uint256 nonce;  
461
```

- For #460
 - uint256 public nonce;

8. Summary of the Audit

Overall the code is well commented, and performs good data validations.

The compiler also displayed 80 warnings:



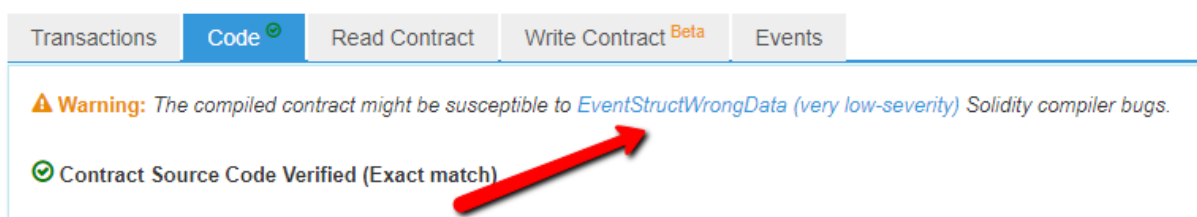
Now, we checked those warnings are due to their static analysis, which includes like gas errors and all. So, it is important to supply correct gas values while calling various functions.

Those warnings (in purple colors) can be safely ignored as should be taken care while calling the smart contract functions.

However, the warning in orange division should be fixed. Although they will not create any risks, but it is good practice to resolve them.

We also noticed one warning popping up on Etherscan:

<https://rinkeby.etherscan.io/address/0x9060380fe557e2b440bce9ba35409fc9bb1b4e18#code>



Which is basically says the if we used Struct in event, then it will not work as it should. Now, we checked we have used Struct at one place and it will not be used in event, so again we can safely ignore this warning.

Please try to check the address and value of token externally before sending to the solidity code.

Our final recommendation would be to pay more attention to the visibility of the functions , hardcoded address and mapping since it's quite important to define who's supposed to executed the functions and to follow best practices regarding the use of assert, require etc. (which you are doing ;)).