Seascape - Moonscape

Smart Contract Security Audit

Prepared by: **Halborn**

Date of Engagement: **February 4th, 2022 - March 25th, 2022**

Visit: **Halborn.com**

DOCUMENT REVISION HISTORY 4
CONTACTS 4
1 EXECUTIVE OVERVIEW 5
1.1 INTRODUCTION 6
1.2 AUDIT SUMMARY 6
1.3 TEST APPROACH & METHODOLOGY 6
RISK METHODOLOGY 7
1.4 SCOPE 9
2 ASSESSMENT SUMMARY & FINDINGS OVERVIEW 10
3 FINDINGS & TECH DETAILS 11
3.1 (HAL-01) UNTRUSTED TOKENS NOT CHECKING BEFORE AND AFTER BALANCE - MEDIUM 13
Code Location 13
Risk Level 14
Recommendation 15
Remediation Plan 15
3.2 (HAL-02) POSSIBLE SIGNATURE RE-USAGE - MEDIUM 16
Code Location 16
Risk Level 19
Recommendation 19
Remediation Plan 20
3.3 (HAL-03) MULTIPLE ACTIVE SESSIONS - LOW 21
PoC 21
Code Location 21
Risk Level 22

Recommendation 22
Remediation Plan 22
3.4 (HAL-04) MISSING PARAMETERS VALIDATION - LOW 23
Code Location 23
Risk Level 23
Recommendation 23
Remediation Plan 24
3.5 (HAL-05) UNUSED MINTED VARIABLE - LOW 25
Code Location 25
Risk Level 26
Recommendation 26
Remediation Plan 26
3.6 (HAL-06) STORAGE USAGE ON READ ONLY FLOW - INFORMATIONAL 27
Code Location 27
Recommendation 28
Remediation Plan 28
3.7 (HAL-07) UNUSED PARAMETERS - INFORMATIONAL 29
Code Location 29
Risk Level 30
Recommendation 30
Remediation Plan 30
4 MANUAL TESTING 31
4.1 CityNFT and RoverNFT 32
4.2 Moonscape 32
5 CALL GRAPH AND INHERITANCE 33
Call Graph 35

5.1 Inheritance 36

DOCUMENT REVISION HISTORY

VERSION	MODIFICATION	DATE	AUTHOR
0.1	Document Creation	02/18/2022	Ferran Celades
0.2	Draft Review	02/18/2022	Gabi Urrutia
0.3	Document updates	03/25/2022	Ferran Celades
0.4	Document updates Review	03/28/2022	Ferran Celades
1.0	Remediation Plan	04/04/2022	Ferran Celades
1.1	Remediation Plan Review	04/04/2022	Gabi Urrutia

CONTACTS

CONTACT	COMPANY	EMAIL
Rob Behnke	Halborn	Rob.Behnke@halborn.com
Steven Walbroehl	Halborn	Steven.Walbroehl@halborn.com
Gabi Urrutia	Halborn	Gabi.Urrutia@halborn.com
Ferran Celades	Halborn	Ferran.Celades@halborn.com

EXECUTIVE OVERVIEW

1.1 INTRODUCTION

engaged Halborn to conduct a security audit on their smart contracts beginning on February 4th, 2022 and ending on March 25th, 2022 . The security assessment was scoped to the smart contracts provided to the Halborn team.

1.2 AUDIT SUMMARY

The team at Halborn was provided seven weeks for the engagement and assigned a full-time security engineer to audit the security of the smart contract. The security engineer is a blockchain and smart-contract security expert with advanced penetration testing, smart-contract hacking, and deep knowledge of multiple blockchain protocols.

The purpose of this audit is to:

- Ensure that smart contract functions operate as intended
- Identify potential security issues with the smart contracts

In summary, Halborn identified some security risks that were mostly addressed by the team.

1.3 TEST APPROACH & METHODOLOGY

Halborn performed a combination of manual and automated security testing to balance efficiency, timeliness, practicality, and accuracy in regard to the scope of this audit. While manual testing is recommended to uncover flaws in logic, process, and implementation; automated testing techniques help enhance coverage of the bridge code and can quickly identify items that do not follow security best practices. The following phases and associated tools were used throughout the term of the audit:

- Research into architecture and purpose
- Smart contract manual code review and walkthrough
- Graphing out functionality and contract logic/connectivity/functions (solgraph)
- Manual assessment of use and safety for the critical Solidity variables and functions in scope to identify any arithmetic related vulnerability classes
- Manual testing by custom scripts
- Static Analysis of security for scoped contract, and imported functions. (Slither)
- Testnet deployment (Brownie, Remix IDE)

RISK METHODOLOGY:

Vulnerabilities or issues observed by Halborn are ranked based on the risk assessment methodology by measuring the **LIKELIHOOD** of a security incident and the **IMPACT** should an incident occur. This framework works for communicating the characteristics and impacts of technology vulnerabilities. The quantitative model ensures repeatable and accurate measurement while enabling users to see the underlying vulnerability characteristics that were used to generate the Risk scores. For every vulnerability, a risk level will be calculated on a scale of 5 to 1 with 5 being the highest likelihood or impact.

RISK SCALE - LIKELIHOOD

5 - Almost certain an incident will occur.
4 - High probability of an incident occurring.
3 - Potential of a security incident in the long term.
2 - Low probability of an incident occurring.
1 - Very unlikely issue will cause an incident.

RISK SCALE - IMPACT

5 - May cause devastating and unrecoverable impact or loss.
4 - May cause a significant level of impact or loss.
3 - May cause a partial impact or loss to many.
2 - May cause temporary impact or loss.

1 - May cause minimal or un-noticeable impact.

The risk level is then calculated using a sum of these two values, creating a value of 10 to 1 with 10 being the highest level of security risk.

10 - CRITICAL
9 - 8 - HIGH
7 - 6 - MEDIUM
5 - 4 - LOW
3 - 1 - VERY LOW AND INFORMATIONAL

1.4 SCOPE

The security assessment was scoped to the following Seastarinteractive/moonscape-smartcontracts

- /contracts/game/MoonscapeGame.sol
- /contracts/nfts/CityNft.sol
- /contracts/nfts/RoverNft.sol

Commit ID: 7f01159e97c132b6bbad5d5511768461d2d480c9

- /contracts/defi/MoonscapeDefi.sol
- /contracts/defi/Stake.sol
- /contracts/beta/MoonscapeBeta.sol

Commit ID: ba85acb2bf7b893d338b9150c2273305e5303eb7

2. ASSESSMENT SUMMARY & FINDINGS OVERVIEW

CRITICAL	HIGH	MEDIUM	LOW	INFORMATIONAL
0	0	2	3	2

LIKELIHOOD

IMPACT

(HAL-02)	(HAL-01)			
	(HAL-03)			
		(HAL-04)		
(HAL-06) (HAL-07)		(HAL-05)		

EXECUTIVE OVERVIEW

SECURITY ANALYSIS	RISK LEVEL	REMEDIATION DATE
UNTRUSTED TOKENS NOT CHECKING BEFORE AND AFTER BALANCE	Medium	SOLVED - 04/04/2022
POSSIBLE SIGNATURE RE-USAGE	Medium	SOLVED - 03/31/2022
MULTIPLE ACTIVE SESSIONS	Low	RISK ACCEPTED
MISSING PARAMETERS VALIDATION	Low	SOLVED - 04/04/2022
UNUSED MINTED VARIABLE	Low	SOLVED - 04/06/2022
STORAGE USAGE ON READ ONLY FLOW	Informational	SOLVED - 03/31/2022
UNUSED PARAMETERS	Informational	SOLVED - 04/04/2022

FINDINGS & TECH DETAILS

3.1 (HAL-01) UNTRUSTED TOKENS NOT CHECKING BEFORE AND AFTER BALANCE - MEDIUM

Under MoonscapeDefi, if untrusted tokens are used as reward tokens, they could be manipulated in such a way that the transfer never happens. The code does not check the result of the low-level call and does not check the before and after balance.

The system allows adding untrusted tokens by calling addTokenStaking. This function does not check any signatures or use any whitelisting system for token addresses.

Code Location:

Listing 1: contracts/defi/MoonscapeDefi.sol (Lines 304,306)

```
298     function _safeTransfer(address _token, address _to, uint256
 ↳ _amount) internal {
299         if (_token != address(0)) {
300             IERC20 _rewardToken = IERC20(_token);
301
302             uint256 _balance = _rewardToken.balanceOf(address(this
 ↳ ));
303             if (_amount > _balance) {
304                 _rewardToken.transfer(_to, _balance);
305             } else {
306                 _rewardToken.transfer(_to, _amount);
307             }
308         } else {
309             uint256 _balance = address(this).balance;
310             if (_amount > _balance) {
311                 payable(_to).transfer(_balance);
312             } else {
313                 payable(_to).transfer(_amount);
314             }
315         }
316     }
```

Listing 2: contracts/defi/MoonscapeDefi.sol (Line 108)

```
94      function addTokenStaking(uint _sessionId, address stakeAddress
↳ , uint rewardPool, address rewardToken, uint _burn) external {
95          bytes32 key = keccak256(abi.encodePacked(_sessionId,
↳ stakeAddress, rewardToken));
96
97          require(!addedStakings[key], "DUPLICATE_STAKING");
98
99          addedStakings[key] = true;
100         bool burn = true;
101
102         if (_burn == 1) {
103             burn = true;
104         } else {
105             burn = false;
106         }
107
108         tokenStakings[++stakeId] = TokenStaking(_sessionId,
↳ stakeAddress, rewardPool, rewardToken, burn);
109
110         bytes32 stakeKey = stakeKeyOf(sessionId, stakeId);
111
112         keyToId[stakeKey] = stakeId;
113
114         Session storage session = sessions[_sessionId];
115
116         newStakePeriod(
117             stakeKey,
118             session.startTime,
119             session.endTime,
120             rewardPool
121         );
122
123         emit AddStaking(_sessionId, stakeId);
124     }
```

Risk Level:

Likelihood - 2
Impact - 5

Recommendation:

It is recommended to always check that the balance difference before and after is the same amount requested. This will prevent untrusted tokens from manipulating balances and allow the system to ensure integrity.

Remediation Plan:

SOLVED: The code does check the post balance and compares it to the sum of the pre-balance and the requested amount.

3.2 (HAL-02) POSSIBLE SIGNATURE RE-USAGE - MEDIUM

In MoonscapeDefi, if a user ever had the same signature parameters as another user in the system, such as the same _buildingId or any other present in the parameters, they could be reusing their signature to get NFT, bonus or add a staking on your behalf.

Code Location:

Listing 3: contracts/defi/MoonscapeDefi.sol (Line 141)

```
127     function stakeToken(uint _stakeId, uint _cityId, uint
 ↳ _buildingId, uint _amount, uint8 v, bytes32[2] calldata sig)
 ↳ external {
128         TokenStaking storage tokenStaking = tokenStakings[_stakeId
 ↳ ];
129 
130         // todo
131         // validate the session id
132         bytes32 stakeKey = stakeKeyOf(tokenStaking.sessionId,
 ↳ _stakeId);
133 
134         require(isActive(stakeKey), "session not active");
135 
136         //validate stake id
137         require(_stakeId <= stakeId,"do not have this stakeId");
138 
139         {
140         bytes memory prefix     = "\x19Ethereum Signed Message:\
 ↳ n32";
141         bytes32 message         = keccak256(abi.encodePacked(
 ↳ _stakeId, tokenStaking.sessionId, _cityId, _buildingId));
142         bytes32 hash            = keccak256(abi.encodePacked(
 ↳ prefix, message));
143         address recover         = ecrecover(hash, v, sig[0], sig
 ↳ [1]);
144 
145         require(recover == owner(), "Verification failed about
 ↳ stakeToken");
146         }
```

```
147
148         deposit(stakeKey, msg.sender, _amount);
149
150         IERC20 token = IERC20(tokenStaking.stakeToken);
151
152         require(token.balanceOf(msg.sender) >= _amount, "Not
 ↳ enough token to stake");
153         // uint preBalance = token.balanceOf(address(this));
154
155         token.safeTransferFrom(msg.sender, address(this), _amount)
 ↳ ;
156         // _amount = token.balanceOf(address(this)) - preBalance;
157         emit StakeToken(msg.sender, tokenStaking.sessionId,
 ↳ _stakeId, _cityId, _buildingId, _amount);
158     }
```

Listing 4: contracts/defi/MoonscapeDefi.sol (Line 200)

```
180     function stakeNft(uint _stakeId, uint _cityId, uint
 ↳ _buildingId, uint _scapeNftId, uint _power, uint8 _v, bytes32[2]
 ↳ calldata sig) external {
181         TokenStaking storage tokenStaking = tokenStakings[_stakeId
 ↳ ];
182
183         // validate the session id
184         bytes32 stakeKey = stakeKeyOf(tokenStaking.sessionId,
 ↳ _stakeId);
185
186         Player storage player = playerParams[stakeKey][msg.sender
 ↳ ];
187
188         require(player.nftId <= 0 && player.power <= 0, "already
 ↳ stake nft");
189
190         require(isActive(stakeKey), "session not active");
191
192         //validate stake id
193         require(_stakeId <= stakeId,"do not have this stakeId");
194
195         IERC721 nft = IERC721(tokenStaking.stakeToken);
196         require(nft.ownerOf(_scapeNftId) == msg.sender, "not owned
 ↳ ");
197
198         {
```

```
199         bytes memory prefix     = "\x19Ethereum Signed Message:\
 ↳ n32";
200         bytes32 message         = keccak256(abi.encodePacked(
 ↳ tokenStaking.sessionId, _stakeId, _cityId, _buildingId,
 ↳ _scapeNftId, _power));
201         bytes32 hash            = keccak256(abi.encodePacked(
 ↳ prefix, message));
202         address recover         = ecrecover(hash, _v, sig[0], sig
 ↳ [1]);
203
204         require(recover == owner(), "Verification failed about
 ↳ stakeNft");
205         }
206
207         nft.safeTransferFrom(msg.sender, address(this),
 ↳ _scapeNftId);
208
209         deposit(stakeKey, msg.sender, _power);
210
211         player.nftId = _scapeNftId;
212         player.power = _power;
213
214         emit StakeNft(msg.sender, tokenStaking.sessionId, _stakeId
 ↳ , _cityId, _buildingId, _scapeNftId, _power);
215     }
```

Listing 5: contracts/defi/MoonscapeDefi.sol (Line 268)

```
255     function getBonus(uint _stakeId, uint _cityId, uint
 ↳ _buildingId, uint _bonusPercent, uint8 _v, bytes32[2] calldata sig
 ↳ ) external {
256         TokenStaking storage tokenStaking = tokenStakings[_stakeId
 ↳ ];
257         Session storage session = sessions[tokenStaking.sessionId
 ↳ ];
258         bytes32 stakeKey = stakeKeyOf(tokenStaking.sessionId,
 ↳ _stakeId);
259
260         require(block.timestamp > session.endTime, "it has to be
 ↳ after the session");
261
262         Player storage player = playerParams[stakeKey][msg.sender
 ↳ ];
263
```

```
264         require(player.receiveBonus == true, "already rewarded");
265
266         {
267         bytes memory prefix     = "\x19Ethereum Signed Message:\
 ↳ n32";
268         bytes32 message         = keccak256(abi.encodePacked(
 ↳ _stakeId, tokenStaking.sessionId, _cityId, _buildingId,
 ↳ _bonusPercent));
269         bytes32 hash            = keccak256(abi.encodePacked(
 ↳ prefix, message));
270         address recover         = ecrecover(hash, _v, sig[0], sig
 ↳ [1]);
271
272         require(recover == owner(), "Verification failed about
 ↳ getBonus");
273         }
274
275         uint256 _totalreward = claimable(stakeKey, msg.sender) +
 ↳ player.claimedAmound;
276         uint256 _totalBonus  = _totalreward.mul(scaler).mul(
 ↳ _bonusPercent).div(100).div(scaler);
277
278         require(_totalBonus > 0, "totalBonus must > 0");
279         _safeTransfer(tokenStaking.rewardToken, msg.sender,
 ↳ _totalBonus);
280
281         player.receiveBonus = true;
282     }
```

Risk Level:

Likelihood - 1
Impact - 5

Recommendation:

It is recommended to have a unique value in the signatures that always prevents the reuse of signatures. In this case, it is recommended to use a nonce value that would be stored on an account basis or to use `msg.sender` in the signature verification. This will prevent any other

user with the same signature parameters from reusing previous signatures.

Remediation Plan:

SOLVED: Code is not checking for `msg.sender` in signatures.

3.3 (HAL-03) MULTIPLE ACTIVE SESSIONS - LOW

In the MoonscapeDefi contract you can add multiple sessions at the same time, this will increment the sessionId and both sessions will be set active at the same time.

PoC:

```
>>> moon = MoonscapeDefi.deploy({'from':a[0]})
Transaction sent: 0xe06ff1e6e1e624dae5f25a60e851d9315a58975565d60f741b6b28f62565521e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 17
  MoonscapeDefi.constructor confirmed   Block: 18   Gas used: 2768410 (23.07%)
  MoonscapeDefi deployed at: 0xFbD588c72B438faD4Cf7cD879c8F730Faa213Da0

>>> moon.startSession(chain.time() + 1, chain.time() + 10)
Transaction sent: 0x27c9b101c5f9a28edc3b8213deab75c6a78cc1099bf29b9565919d79fcdfa580
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 18
  MoonscapeDefi.startSession confirmed   Block: 19   Gas used: 107910 (0.90%)

<Transaction '0x27c9b101c5f9a28edc3b8213deab75c6a78cc1099bf29b9565919d79fcdfa580'>
>>> moon.startSession(chain.time()+10, chain.time() + 20)
Transaction sent: 0x7f903e68f6fcc2e4f50f3473b5fd174ad5450fd8c2988d3700909e746167a797
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 19
  MoonscapeDefi.startSession confirmed   Block: 20   Gas used: 92910 (0.77%)

<Transaction '0x7f903e68f6fcc2e4f50f3473b5fd174ad5450fd8c2988d3700909e746167a797'>
>>> moon.sessions(1)
(1647623370, 1647623379, True)
>>> moon.sessions(2)
(1647623382, 1647623392, True)
>>>
```

Figure 1: Demo showing how two sessions can be active at the same time

Code Location:

Listing 6: contracts/defi/MoonscapeDefi.sol (Line 61)

```
61      function startSession(uint _startTime, uint _endTime) external
↳ {
62          require(validSessionTime(_startTime, _endTime), "
↳ INVALID_SESSION_TIME");
63
64          sessionId++;
65
66          sessions[sessionId] = Session(_startTime, _endTime, true);
67
```

```
68          emit StartSession(sessionId, _startTime, _endTime);
69      }
```

Listing 7: contracts/defi/MoonscapeDefi.sol (Line 330)

```
330     function validSessionTime(uint _startTime, uint _endTime)
 ↳ public view returns(bool) {
331         Session storage session = sessions[sessionId];
332 
333         if (_startTime > session.endTime && _startTime >= block.
 ↳ timestamp && _startTime < _endTime) {
334             return true;
335         }
336 
337         return false;
338     }
```

Risk Level:

Likelihood - 2
Impact - 3

Recommendation:

It is recommended to check already active sessions by validating the start and end times.

Remediation Plan:

RISK ACCEPTED: The code is protected with the onlyOwner modifier, so only the owner can add new sessions. Plus, the client states that: “This is done on purpose because if we by any chance make a mistake, we would need to redeploy everything. But with multiple active sessions, we can just open another session and change the active session in the client”

3.4 (HAL-04) MISSING PARAMETERS VALIDATION - LOW

The constructor in MoonscapeGame does not check the validity of the parameters, and there is no setter that can change the values. This would cause a contract lockout or the need to redeploy.

Code Location:

Listing 8: game/MoonscapeGame.sol (Lines 65-70)

```
57     constructor(
58         address _mscpToken,
59         address _cityNft,
60         address _roverNft,
61         address _scapeNft,
62         address _verifier,
63         address _feeTo
64     ) public {
65         MSCP = _mscpToken;
66         cityNft = _cityNft;
67         roverNft = _roverNft;
68         scapeNft = _scapeNft;
69         verifier = _verifier;
70         feeTo = _feeTo;
71     }
```

Risk Level:

Likelihood - 3
Impact - 2

Recommendation:

The constructor in MoonscapeGame should check if the tokens are valid and if the addresses are non-zero.

Remediation Plan:

SOLVED: The code is now checking if the parameters are not zero.

3.5 (HAL-05) UNUSED MINTED VARIABLE - LOW

The mint function under the CityNft and RoverNft contracts does not check or set the internal minted mapping. The minted variable is never used and should be used or removed from the system.

Code Location:

Listing 9: nfts/CityNft.sol (Line 26)

```
25     function mint(uint _tokenId, uint8 _category, address _to)
↳ external returns(bool) {
26         if (!minters[msg.sender] || minted[_tokenId] || _to ==
↳ address(0) || _category > 8) {
27             return false;
28         }
29
30         categoryOf[_tokenId] = _category;
31
32         _safeMint(_to, _tokenId);
33
34         emit Minted(_to, _tokenId, _category, block.timestamp);
35         return true;
36     }
```

Listing 10: nfts/RoverNft.sol (Line 26)

```
25     function mint(uint _tokenId, uint8 _type, address _to)
↳ external returns(bool) {
26         if (!minters[msg.sender] || minted[_tokenId] || _to ==
↳ address(0) || _type > 8) {
27             return false;
28         }
29
30         typeOf[_tokenId] = _type;
31
32         _safeMint(_to, _tokenId);
33
34         emit Minted(_to, _tokenId, _type, block.timestamp);
```

```
35          return true;
36      }
```

Risk Level:

Likelihood - 3
Impact - 1

Recommendation:

Although the _safeMint function will take care of checking that the _tokenId has not already been minted, minted was introduced to check for the same thing, doubling the functionality. The minted variable is not used and should be removed. The variable could be replaced using !_exists(tokenId) from the ERC721 contract.

Remediation Plan:

SOLVED: Variables were removed from the parameters and the contract storage.

3.6 (HAL-06) STORAGE USAGE ON READ ONLY FLOW - INFORMATIONAL

The addTokenStaking function does use storage for the session. However, the object is only used to fetch the values and is never used to store them. It would be nice to change the keyword from storage to memory.

Code Location:

Listing 11: contracts/defi/MoonscapeDefi.sol (Line 114)

```
94 function addTokenStaking(uint _sessionId, address stakeAddress,
↳ uint rewardPool, address rewardToken, uint _burn) external {
95         bytes32 key = keccak256(abi.encodePacked(_sessionId,
↳ stakeAddress, rewardToken));
96
97         require(!addedStakings[key], "DUPLICATE_STAKING");
98
99         addedStakings[key] = true;
100        bool burn = true;
101
102        if (_burn == 1) {
103            burn = true;
104        } else {
105            burn = false;
106        }
107
108        tokenStakings[++stakeId] = TokenStaking(_sessionId,
↳ stakeAddress, rewardPool, rewardToken, burn);
109
110        bytes32 stakeKey = stakeKeyOf(sessionId, stakeId);
111
112        keyToId[stakeKey] = stakeId;
113
114        Session storage session = sessions[_sessionId];
115
116        newStakePeriod(
117            stakeKey,
118            session.startTime,
119            session.endTime,
120            rewardPool
```

```
121         );
122
123         emit AddStaking(_sessionId, stakeId);
124     }
```

Listing 12: contracts/defi/MoonscapeDefi.sol (Line 331)

```
330 function validSessionTime(uint _startTime, uint _endTime) public
 ↳ view returns(bool) {
331         Session storage session = sessions[sessionId];
332
333         if (_startTime > session.endTime && _startTime >= block.
 ↳ timestamp && _startTime < _endTime) {
334             return true;
335         }
336
337         return false;
338     }
```

Recommendation:

It is recommended to use memory access instead of storage if no state changes are being made.

Remediation Plan:

SOLVED: The memory keyword is now used instead of storage.

3.7 (HAL-07) UNUSED PARAMETERS - INFORMATIONAL

The mintRover and mintCity functions in MoonscapeGame does not use the _amount parameter for any storage state operation. It has no effect on the code other than signature verification.

Code Location:

Listing 13: game/MoonscapeGame.sol (Lines 248,252)

```
248     function mintRover(uint _id, uint8 _type, uint  _amount, uint8
 ↳  _v, bytes32 _r, bytes32 _s) external {
249         {   // avoid stack too deep
250         // investor, project verification
251     bytes memory prefix     = "\x19Ethereum Signed Message:\n32";
252     bytes32 message         = keccak256(abi.encodePacked(msg.
 ↳ sender, address(this), roverNft, _id, _amount, _type));
253     bytes32 hash            = keccak256(abi.encodePacked(prefix,
 ↳ message));
254     address recover         = ecrecover(hash, _v, _r, _s);
255
256     require(recover == verifier, "sig");
257         }
258
259         CityNft nft = CityNft(roverNft);
260         require(nft.mint(_id, _type, msg.sender), "Failed to mint
 ↳ rover");
261
262         emit MintRover(msg.sender, _amount, _id, _type);
263     }
```

Listing 14: game/MoonscapeGame.sol (Lines 151,155)

```
151     function mintCity(uint _id, uint8 _category, uint _amount,
 ↳ uint8 _v, bytes32 _r, bytes32 _s) external {
152         {   // avoid stack too deep
153         // investor, project verification
154     bytes memory prefix     = "\x19Ethereum Signed Message:\n32";
```

```
155     bytes32 message          = keccak256(abi.encodePacked(msg.
 ↳ sender, address(this), cityNft, _id, _amount, _category));
156     bytes32 hash             = keccak256(abi.encodePacked(prefix,
 ↳ message));
157     address recover          = ecrecover(hash, _v, _r, _s);
158 
159     require(recover == verifier, "sig");
160         }
161 
162         CityNft nft = CityNft(cityNft);
163         require(nft.mint(_id, _category, msg.sender), "Failed to
 ↳ mint city");
164 
165         emit MintCity(msg.sender, _amount, _id, _category);
166     }
```

Risk Level:

Likelihood - 1
Impact - 1

Recommendation:

All parameters given to a function should have an effect on the state of the code. The _amount parameter is used for signature verification, but is not used for any contract state operations.

Remediation Plan:

SOLVED: Removed _amount variable from function arguments.

MANUAL TESTING

4.1 CityNFT and RoverNFT

Checking for mint duplication:

```
>>> cnft.mint(0, 1, a[0])
Transaction sent: 0x6da81d29109268feb69c3700bf24fbda1ae914a28a359b84505a0e8d7581bc77
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  CityNft.mint confirmed   Block: 2   Gas used: 158464 (1.32%)

<Transaction '0x6da81d29109268feb69c3700bf24fbda1ae914a28a359b84505a0e8d7581bc77'>
>>> cnft.mint(0, 1, a[0])
Transaction sent: 0x9d679b2ded4d56fbbf511cd78a86ec41fad73ffd87893ef346dc83b0b1dcc9e9
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 2
  CityNft.mint confirmed (ERC721: token already minted)   Block: 3   Gas used: 26863 (0.22%)

<Transaction '0x9d679b2ded4d56fbbf511cd78a86ec41fad73ffd87893ef346dc83b0b1dcc9e9'>
>>>
```

Trying to mint without being a minter

```
>>> cnft.minters(a[0])
True
>>> cnft.mint(5, 8, a[0])
Transaction sent: 0x6be27df6fca2a138d1ce895a561ab6f747cd3f3f74fb91664f067bfeec232eb5
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 12
  CityNft.mint confirmed   Block: 13   Gas used: 166876 (1.39%)

<Transaction '0x6be27df6fca2a138d1ce895a561ab6f747cd3f3f74fb91664f067bfeec232eb5'>
>>> cnft.categoryOf(5)
8
>>> cnft.unsetMinter(a[0])
Transaction sent: 0x2f56f0a55ac8c5ef8545f3e6355a72cc6211d7b6b4e9e39204b1efa90f1d14c3
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 13
  CityNft.unsetMinter confirmed   Block: 14   Gas used: 15518 (0.13%)

<Transaction '0x2f56f0a55ac8c5ef8545f3e6355a72cc6211d7b6b4e9e39204b1efa90f1d14c3'>
>>> cnft.mint(6, 8, a[0])
Transaction sent: 0x761e0900faa64b50b862a15ae4c364ddb74300ed6866cf61472a97bc683d1f5e
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 14
  CityNft.mint confirmed   Block: 15   Gas used: 23072 (0.19%)

<Transaction '0x761e0900faa64b50b862a15ae4c364ddb74300ed6866cf61472a97bc683d1f5e'>
>>> cnft.categoryOf(6)
0
>>> cnft.minters(a[0])
False
>>>
```

4.2 Moonscape

Import a city not owned by the importer:

MANUAL TESTING

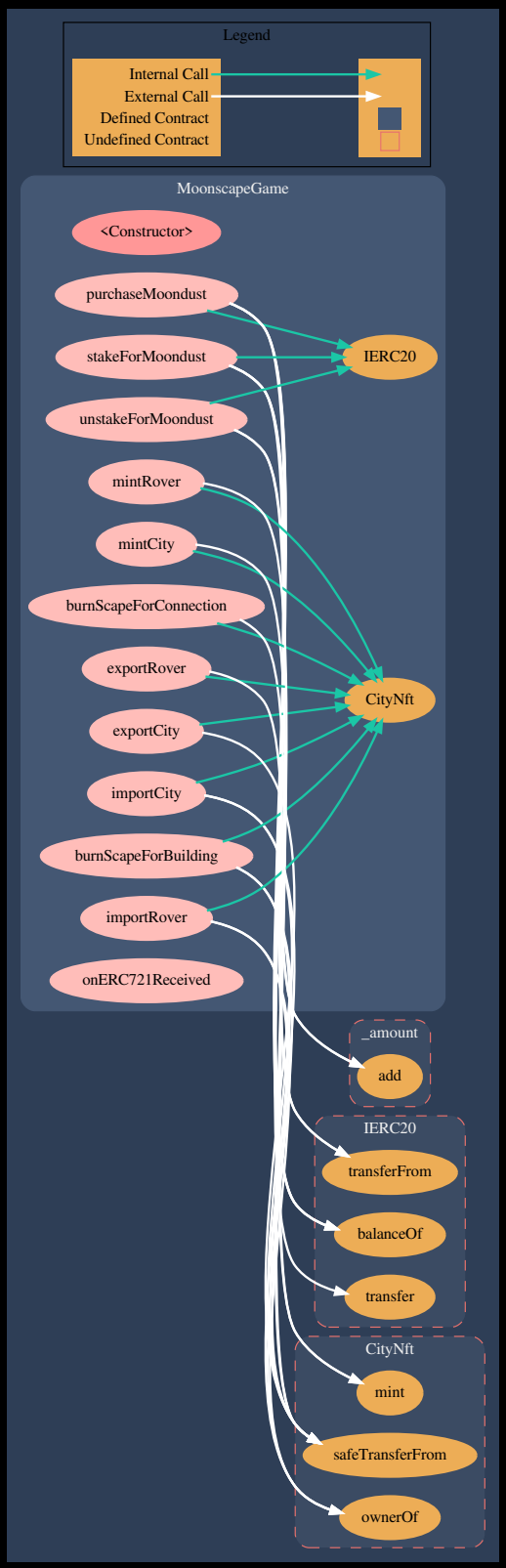
```
>>> moon.importCity(1, {'from':a[1]})
Transaction sent: 0xa4cfa02668e0971adf4969cdabc8b30ec559a5bc879bb2d7894fed15a21cf688
  Gas price: 0.0 gwei   Gas limit: 12000000   Nonce: 1
  MoonscapeGame.importCity confirmed (Not city owner)   Block: 40   Gas used: 27164 (0.23%)

<Transaction '0xa4cfa02668e0971adf4969cdabc8b30ec559a5bc879bb2d7894fed15a21cf688'>
>>>
```

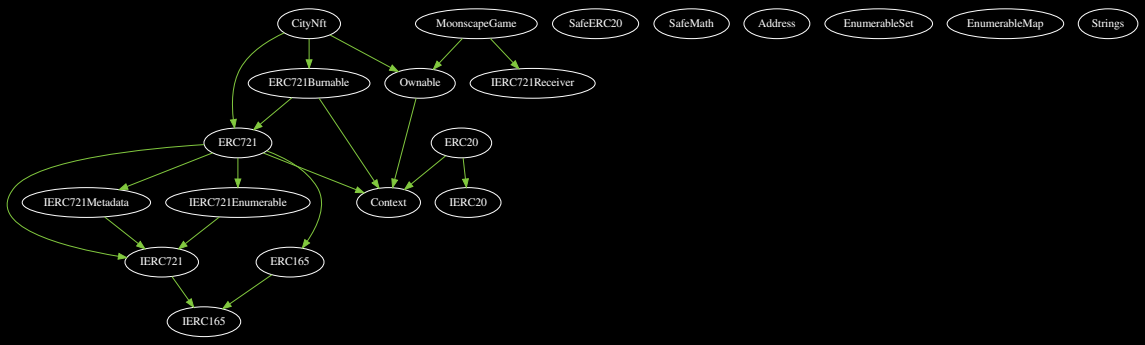
CALL GRAPH AND INHERITANCE

CALL GRAPH AND INHERITANCE

Call Graph:



5.1 Inheritance



THANK YOU FOR CHOOSING

// HALBORN