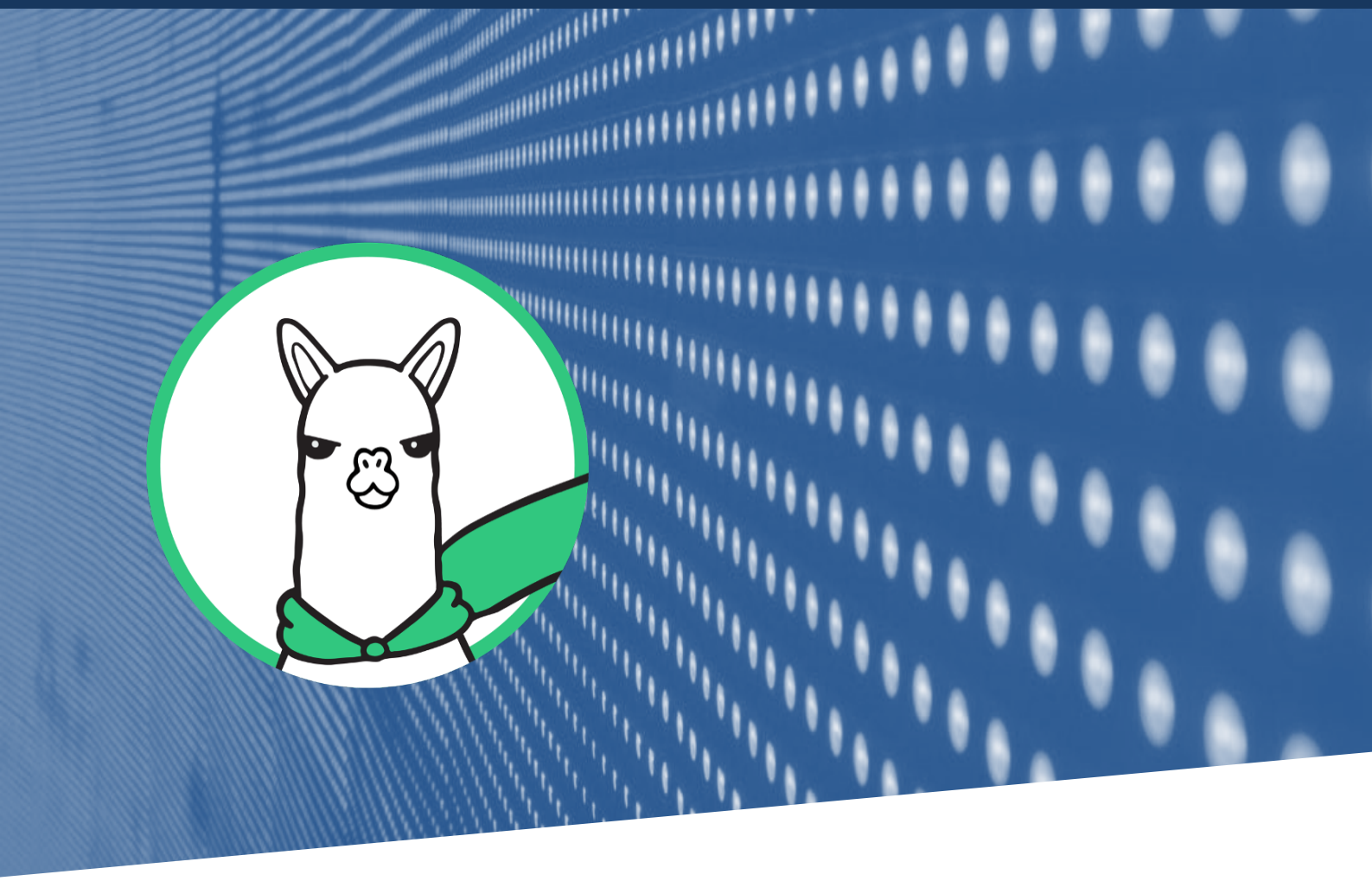
Grazing Range

Smart Contract Security Review

Alpaca Finance



Full Report

5 May 2021

Public

ValiX

Table of Contents

Inherent Limitations

Our review was conducted over a limited period and was performed on the smart contract at a single point in time. As such, the scope was limited to current known risks during the work period. The review may not yield any risks, which does not indicate that the smart contract has no vulnerability exposure. We performed the review with our best effort, and we do not guarantee a hundred percent coverage of the underlying risk existing in the ecosystem.

Disclaimer

Due to the engagement type, we warrant that it shall be performed in good faith and with reasonable skill and care, and in accordance with generally accepted professional standards.

The procedures performed do not constitute an audit, a review, or a compilation of the client's security capability or any part thereof, nor an examination of management's assertions concerning the effectiveness of the client's internal control systems, nor an examination of compliance with laws, regulations, or other matters. Accordingly, our performance of the procedures will not result in the expression of an opinion, or any other form of assurance, on the client's security capability or any part thereof, nor an opinion or any other form of assurance on the client's internal control systems or its compliance with laws, regulations, or other matters.

Executive Summary

Overview

Valix conducted a smart contract review for the Alpaca Finance Protocol to discover issues and vulnerabilities in the source code. Its scope is limited to the Grazing Range smart contract, and to some aspects of the smart contract itself. Security best practices strongly recommend that Alpaca conduct a full security audit of the on-chain and off-chain components of their infrastructure, and the interaction between the two. A comprehensive examination has been performed, utilizing Valix's Formal Verification Platform, Static Analysis, and Manual Review techniques.

Scope of Work

The security review conducted doesn't replace a full security audit of the overall Alpaca Finance Protocol. Its scope is limited to the Grazing Range smart contract, and to some aspects of the smart contract itself.

The security review covers the components of the Alpaca Finance Protocol at this specific state:

Item	Description
Components	▪ Alpaca Finance Grazing Range smart contract ▪ Imported associated smart contracts ▪ Compilation and testing environment
GitHub Repository	https://github.com/alpaca-finance/bsc-alpaca-contract
Commit	`fcd2f3cba29f94049a573fd6464eaa628aaeaf84`
Reassessment Commit	`a2f6250b68094fcac720e29cb676e2c0b2673b83`
Files	`GrazingRange.sol`

Remark: Security best practices strongly recommend that Alpaca conduct a full security audit of the on-chain and off-chain components of their infrastructure.

Review Result Summary

Initially, Valix was able to identify 8 issues which are categorized from the “High” to “Note” risk level respectively in the given timeframe of the assessment. Below is the breakdown of the vulnerabilities found and their associated risk rating for each assessment conducted. On the reassessment, all issues were fixed.

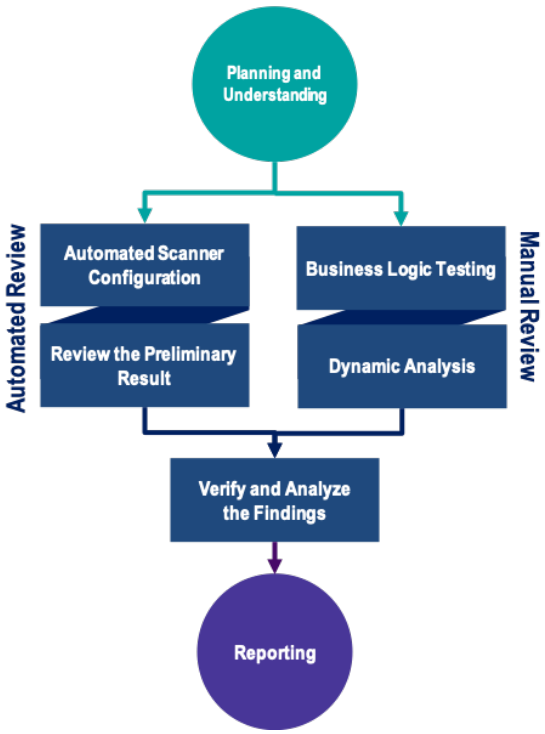
Target	Assessment				Reassessment			
	H	M	L	N	H	M	L	N
Grazing Range Smart Contract	**2**	**3**	**1**	**2**	**0**	**0**	**0**	**0**

Note: Risk Rating H = High, M = Medium, L = Low, N = Note

Review Detail

Methodology

The smart contract security audit methodology is based on Smart Contract Weakness Classification and Test Cases (SWC Registry), and Common Weakness Enumeration (CWE). Manual and automated review approaches can be mixed and matched including business logic analysis in terms of malicious doers' perspective. Using automated scanning tools to navigate or find offending software patterns in the codebase along with a purely manual or semi-automated approach, where the analyst primarily relies on one's knowledge, will be performed in order to eliminate the false-positive results.



Area	Description
Planning and Understanding	• *Determine scope of testing and understand application purpose and workflows* • *Identify key risk areas, including technical and business risks* • *Determine approach – which sections to review within the resource constraints and review method – automated, manual or mixed*
Automated Review	• *Adjust automated source code review tools to inspect the code for known unsafe coding patterns* • *Verify output of the tool in order to eliminate false positive result, and if necessary, adjust and re-run the code review tool*
Manual Review	• *Testing for business logic flaws requires thinking in unconventional methods* • *Identify unsafe coding behavior via static code analysis*
Reporting	• *Analyze the root cause of the flaws* • *Recommend coding process improvements*

Risk Level Classification

For prioritization of the vulnerabilities, we have adopted the scheme by four distinct levels for risk: High, Medium, Low, and Note. The risk level definitions are presented in table.

Risk Level	Definition
High	*The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.*
Medium	*The code implementation does not match the specification under certain conditions, or it could affect the security standard by loss of access control.*
Low	*The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.*
Note	*Findings in this category are informational and may be further improved by following best practices and guidelines.*

The risk value for each threat is calculated as the product of impact and likelihood values, illustrated in a two-dimensional matrix. The shading of the matrix visualizes the different risk levels. Based on the acceptance criteria, the risk level “High” is decided to be unacceptable. Any threat obtaining this risk level must be treated in order to have its risk reduced to an acceptable level.

Impact \ Likelihood	Low	Medium	High
High	Medium	High	High
Medium	Low	Medium	High
Low	Note	Low	Medium

Review Findings

Review Findings Summary

The summary of findings with the risks associated are shown in the table below:

No.	Findings	Risk Level	Reassessment
1	Unprotected Token Withdrawal (SWC-105)	High	Fixed
2	State Variable Not Updated (CWE-840)	High	Fixed
3	Reentrancy (SWC-107)	Medium	Fixed
4	Incorrect Pending Reward Calculation (CWE-840)	Medium	Fixed
5	Incorrect Reward Per Block Display (CWE-840)	Medium	Fixed
6	Usage of Outdated Solidity Compiler Version (SWC-102)	Low	Fixed
7	Duplicated Imports	Note	Fixed
8	Public Function That Could Be Declared External	Note	Fixed

Detailed Findings and Recommendations

1. Unprotected Token Withdrawal (SWC-105)

Risk Level	High	Likelihood	Medium
		Impact	High
Affected	*GrazingRange.sol*		
Location	*emergencyRewardWithdraw() L:276*		

Observation and Implication

The only condition checked in the `emergencyRewardWithdraw()` function was the amount to be withdrawn compared to the balance of the `rewardToken` this contract is holding. This allows the whole balance of the `rewardToken` in the contract to be withdrawn by the contract owner to any arbitrary address.

Recommendations

Modify the function and limit the withdraw amount to be less than or equal to the pending reward of the contract.

Evidence of Finding

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L276

```
contracts > 6 > token > GrazingRange.sol > ...
274
275     // @notice Withdraw reward. EMERGENCY ONLY.
        ftrace | funcSig
276     function emergencyRewardWithdraw(uint256 _campaignID⬆, uint256 _amount⬆, address _beneficiary⬆) external onlyOwner {
277         CampaignInfo storage campaign = campaignInfo[_campaignID⬆];
278         require(_amount⬆ < campaign.rewardToken.balanceOf(address(this)), "GrazingRange::emergencyRewardWithdraw::not enough token");
279         campaign.rewardToken.safeTransfer(_beneficiary⬆, _amount⬆);
280     }
281 }
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L316

bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...

```
314
315       // @notice Withdraw reward. EMERGENCY ONLY.
          ftrace | funcSig
316       function emergencyRewardWithdraw(uint256 _campaignID⬆, uint256 _amount⬆, address _beneficiary⬆) external onlyOwner nonReentrant {
317           CampaignInfo storage campaign = campaignInfo[_campaignID⬆];
318           uint256 currentStakingPendingReward = _pendingReward(_campaignID⬆, campaign.totalStaked, 0);
319           require(currentStakingPendingReward.add(_amount⬆) <= campaign.totalRewards, "GrazingRange::emergencyRewardWithdraw::not enough reward token");
320           campaign.totalRewards = campaign.totalRewards.sub(_amount⬆);
321           campaign.rewardToken.safeTransfer(_beneficiary⬆, _amount⬆);
322       }
323   }
```

2. State Variable Not Updated (CWE-840)

Risk Level	High	Likelihood	High
		Impact	Medium
Affected	*GrazingRange.sol*		
Location	*emergencydWithdraw() L:266*		

Observation and Implication

The `campaign.totalStaked` variable was not updated in the `emergencyWithdraw()` function. This causes all reward calculations after this function call to be incorrect, as the `campaign.totalStaked` would be more than the actual amount staked in the contract.

Recommendations

Deduct `campaign.totalStaked` by the amount withdrawn.

Evidence of Finding

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L266

bsc-alpaca-contract > contracts > 6 > token > GrazingRange.sol > ...

```
264
265     // @notice Withdraw without caring about rewards. EMERGENCY ONLY.
        ftrace | funcSig
266     function emergencyWithdraw(uint256 _campaignID) external {
267         CampaignInfo storage campaign = campaignInfo[_campaignID];
268         UserInfo storage user = userInfo[_campaignID][msg.sender];
269         campaign.stakingToken.safeTransfer(address(msg.sender), user.amount);
270         user.amount = 0;
271         user.rewardDebt = 0;
272         emit EmergencyWithdraw(msg.sender, user.amount, _campaignID);
273     }
274
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L308

bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...

```
302
303     // @notice Withdraw without caring about rewards. EMERGENCY ONLY.
        ftrace | funcSig
304     function emergencyWithdraw(uint256 _campaignID⬆) external nonReentrant {
305         CampaignInfo storage campaign = campaignInfo[_campaignID⬆];
306         UserInfo storage user = userInfo[_campaignID⬆][msg.sender];
307         uint256 _amount = user.amount;
308         campaign.totalStaked = campaign.totalStaked.sub(_amount);
309         user.amount = 0;
310         user.rewardDebt = 0;
311         campaign.stakingToken.safeTransfer(address(msg.sender), _amount);
312         emit EmergencyWithdraw(msg.sender, _amount, _campaignID⬆);
313     }
314
```

3. Reentrancy (SWC-107)

Risk Level	Medium	Likelihood	Low
		Impact	High
Affected	*GrazingRange.sol*		
Location	*emergencyWithdraw() L:269*		

Observation and Implication

External contract was called in the `emergencyWithdraw()` function without a reentrancy guard. Furthermore, the function was not following the "Checks-Effects-Interactions" pattern, so it could potentially be abused if the contract of `stakingToken` is written maliciously.

Recommendations

Add `nonReentrant` modifier from OpenZeppelin's `ReentrancyGuard.sol` to the affected function.

Furthermore, follow the "Checks-Effects-Interactions" in the function with external contract calling whenever possible.

Reference:
- https://docs.openzeppelin.com/contracts/2.x/api/utils#ReentrancyGuard
- https://docs.soliditylang.org/en/v0.6.12/security-considerations.html?highlight=security#re-entrancy

Evidence of Finding

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L269

contracts > 6 > token > GrazingRange.sol > ...

```
264
265     // @notice Withdraw without caring about rewards. EMERGENCY ONLY.
        ftrace | funcSig
266     function emergencyWithdraw(uint256 _campaignID⬆) external {
267         CampaignInfo storage campaign = campaignInfo[_campaignID⬆];
268         UserInfo storage user = userInfo[_campaignID⬆][msg.sender];
269         campaign.stakingToken.safeTransfer(address(msg.sender), user.amount);
270         user.amount = 0;
271         user.rewardDebt = 0;
272         emit EmergencyWithdraw(msg.sender, user.amount, _campaignID⬆);
273     }
274
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L304

bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...

```
302
303     // @notice Withdraw without caring about rewards. EMERGENCY ONLY.
        ftrace | funcSig
304     function emergencyWithdraw(uint256 _campaignID) external nonReentrant {
305         CampaignInfo storage campaign = campaignInfo[_campaignID];
306         UserInfo storage user = userInfo[_campaignID][msg.sender];
307         uint256 _amount = user.amount;
308         campaign.totalStaked = campaign.totalStaked.sub(_amount);
309         user.amount = 0;
310         user.rewardDebt = 0;
311         campaign.stakingToken.safeTransfer(address(msg.sender), _amount);
312         emit EmergencyWithdraw(msg.sender, _amount, _campaignID);
313     }
314
```

4. Incorrect Pending Reward Calculation (CWE-840)

Risk Level	Medium	Likelihood	Medium
		Impact	Medium
Affected	*GrazingRange.sol*		
Location	*pendingReward() L:163*		

Observation and Implication

The value of `campaign.lastRewardBlock` in `pendingReward()` function was not updated, causing the `multiplier` of some reward periods to be calculated repeatedly when the `updateCampaign()` function has never been called during the current reward period. This would cause incorrect display for the reward value.

For example:

If the `campaign.lastRewardBlock` is before `rewardInfo[0].endBlock`, and the current `block.number` is after `rewardInfo[1].endBlock` and before `rewardInfo[2].endBlock`.

Current multiplier calculation:

```
rewardInfo[0]: campaign.lastRewardBlock -> rewardInfo[0].endBlock
rewardInfo[1]: campaign.lastRewardBlock -> rewardInfo[1].endBlock
rewardInfo[2]: campaign.lastRewardBlock -> block.number
```

What the multiplier should be:

```
rewardInfo[0]: campaign.lastRewardBlock -> rewardInfo[0].endBlock
rewardInfo[1]: rewardInfo[0].endBlock -> rewardInfo[1].endBlock
rewardInfo[2]: rewardInfo[1].endBlock -> block.number
```

Recommendations

Update the starting block of `multiplier` calculation in each loop of `rewardInfo` entry.

Evidence of Finding

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L163

contracts > 6 > token > GrazingRange.sol > ...

```
155
156     // @notice View function to see pending Reward on frontend.
        ftrace | funcSig
157     function pendingReward(uint256 _campaignID⬆, address _user⬆) external view returns (uint256) {
158         CampaignInfo memory campaign = campaignInfo[_campaignID⬆];
159         UserInfo memory user = userInfo[_campaignID⬆][_user⬆];
160         RewardInfo[] memory rewardInfo = campaignRewardInfo[_campaignID⬆];
161         uint256 accRewardPerShare = campaign.accRewardPerShare;
162         if (block.number > campaign.lastRewardBlock && campaign.totalStaked != 0) {
163             for (uint256 i = 0; i < rewardInfo.length; ++i) {
164                 uint256 multiplier = getMultiplier(campaign.lastRewardBlock, block.number, rewardInfo[i].endBlock);
165                 if (multiplier == 0) continue;
166                 uint256 reward = multiplier.mul(rewardInfo[i].rewardPerBlock);
167                 accRewardPerShare = accRewardPerShare.add(reward.mul(1e12).div(campaign.totalStaked));
168             }
169         }
170         return user.amount.mul(accRewardPerShare).div(1e12).sub(user.rewardDebt);
171     }
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L184

bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...

```
182
183     // @notice View function to see pending Reward on frontend.
        ftrace | funcSig
184     function pendingReward(uint256 _campaignID⬆, address _user⬆) external view returns (uint256) {
185         return _pendingReward(_campaignID⬆, userInfo[_campaignID⬆][_user⬆].amount, userInfo[_campaignID⬆][_user⬆].rewardDebt);
186     }
187
        ftrace | funcSig
188     function _pendingReward(uint256 _campaignID⬆, uint256 _amount⬆, uint256 _rewardDebt⬆) internal view returns (uint256) {
189         CampaignInfo memory campaign = campaignInfo[_campaignID⬆];
190         RewardInfo[] memory rewardInfo = campaignRewardInfo[_campaignID⬆];
191         uint256 accRewardPerShare = campaign.accRewardPerShare;
192         if (block.number > campaign.lastRewardBlock && campaign.totalStaked != 0) {
193             uint256 cursor = campaign.lastRewardBlock;
194             for (uint256 i = 0; i < rewardInfo.length; ++i) {
195                 uint256 multiplier = getMultiplier(cursor, block.number, rewardInfo[i].endBlock);
196                 if (multiplier == 0) continue;
197                 cursor = rewardInfo[i].endBlock;
198                 accRewardPerShare = accRewardPerShare.add(multiplier.mul(rewardInfo[i].rewardPerBlock).mul(1e12).div(campaign.totalStaked));
199             }
200         }
201         return _amount⬆.mul(accRewardPerShare).div(1e12).sub(_rewardDebt⬆);
202     }
203
```

5. Incorrect Reward Per Block Display (CWE-840)

Risk Level	Medium	Likelihood	Medium
		Impact	Medium
Affected	*GrazingRange.sol*		
Location	*_rewardPerBlockOf() L:141*		

Observation and Implication

Incorrect logic found at the return statement of `_rewardPerBlockOf()` function, causing the reward to be displayed as more than 0, even after the reward period has ended.

Recommendations

Change the return value to 0 for the case when the reward period has ended.

Evidence of Finding

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L141

contracts > 6 > token > GrazingRange.sol > GrazingRange

```
129
        ftrace | funcSig
130     function _rewardPerBlockOf(uint256 _campaignID⬆, uint256 _blockNumber⬆) internal view returns (uint256) {
131         RewardInfo[] memory rewardInfo = campaignRewardInfo[_campaignID⬆];
132         uint256 len = rewardInfo.length;
133         if (len == 0) {
134             return 0;
135         }
136         for (uint256 i = 0; i < len; ++i) {
137             if (_blockNumber⬆ <= rewardInfo[i].endBlock) return rewardInfo[i].rewardPerBlock;
138         }
139         // @dev when couldn't find any reward info, it means that timestamp exceed endblock
140         // so return the latest reward info
141         return rewardInfo[len-1].rewardPerBlock;
142     }
143
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L168

bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...

```
156
        ftrace | funcSig
157     function _rewardPerBlockOf(uint256 _campaignID, uint256 _blockNumber) internal view returns (uint256) {
158         RewardInfo[] memory rewardInfo = campaignRewardInfo[_campaignID];
159         uint256 len = rewardInfo.length;
160         if (len == 0) {
161             return 0;
162         }
163         for (uint256 i = 0; i < len; ++i) {
164             if (_blockNumber <= rewardInfo[i].endBlock) return rewardInfo[i].rewardPerBlock;
165         }
166         // @dev when couldn't find any reward info, it means that timestamp exceed endblock
167         // so return 0
168         return 0;
169     }
170
```

6. Usage of Outdated Solidity Compiler Version (SWC-102)

Risk Level	Low	Likelihood	Low
		Impact	Medium
Affected	*GrazingRange.sol*		
Location	*L:1*		

Observation and Implication

The compiler version specified in the contract was 0.6.6, which is outdated.

Outdated Solidity compiler may have some bugs which is publicly known, that could potentially be used to perform attacks on the smart contract.

Recommendations

Consider changing the compiler version to latest stable version.

The current lastest stable release (as of April 2021) of Solidity with the major 0.6 is version 0.6.12.

Reference:
https://github.com/ethereum/solidity/releases

Evidence of Finding

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L1

```
contracts > 6 > token > GrazingRange.sol > GrazingRange

    report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml | abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
1   pragma solidity 0.6.6;
2
    abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
3   import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol";
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L1

```
bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...
    report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml
1   pragma solidity 0.6.12;
```

7. Duplicated Imports

Risk Level	Note	Likelihood	Low
		Impact	Low
Affected	*GrazingRange.sol*		
Location	*L:6*		

Observation and Implication

`@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol` is imported twice.

Recommendations

Remove the duplicated import.

Evidence of Finding

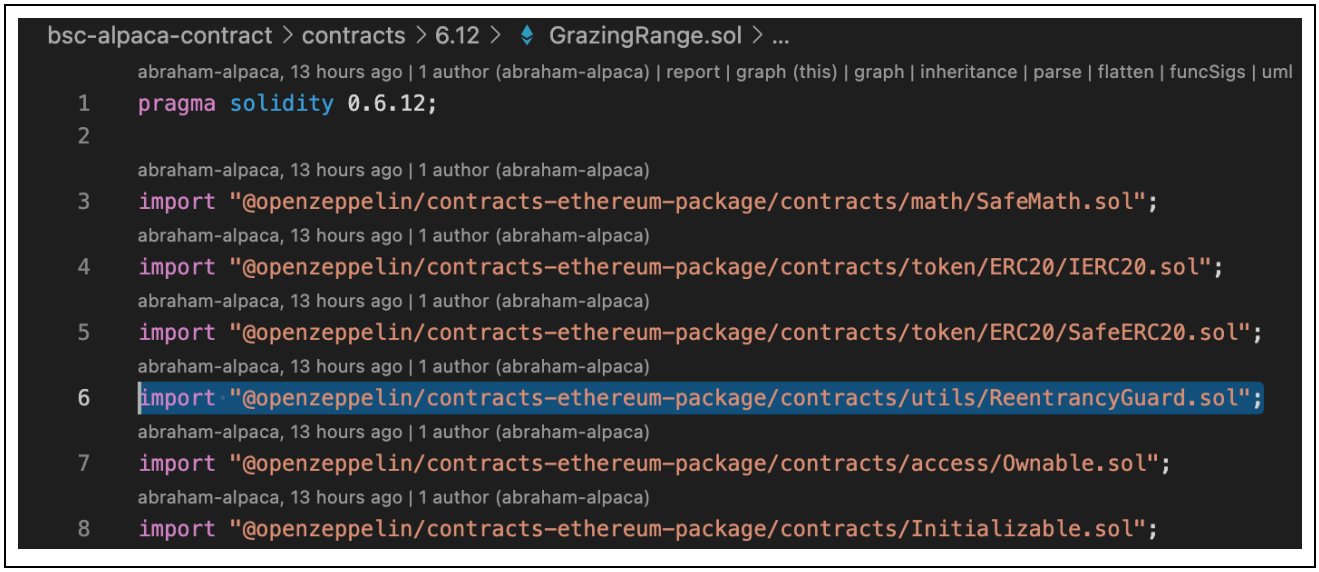
https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L6

```
contracts > 6 > token > GrazingRange.sol > ...
      report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml | abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and
1     pragma solidity 0.6.6;
2
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
3     import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol";
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
4     import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol";
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
5     import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol";
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
6     import "@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol";
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
7     import "@openzeppelin/contracts-ethereum-package/contracts/access/Ownable.sol";
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
8     import "@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol";
      abraham-alpaca, 3 days ago | 2 authors (abraham-alpaca and others)
9     import "@openzeppelin/contracts-ethereum-package/contracts/Initializable.sol";
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L6



```
bsc-alpaca-contract > contracts > 6.12 > GrazingRange.sol > ...
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca) | report | graph (this) | graph | inheritance | parse | flatten | funcSigs | uml
1  pragma solidity 0.6.12;
2
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca)
3  import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol";
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca)
4  import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol";
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca)
5  import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol";
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca)
6  import "@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol";
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca)
7  import "@openzeppelin/contracts-ethereum-package/contracts/access/Ownable.sol";
abraham-alpaca, 13 hours ago | 1 author (abraham-alpaca)
8  import "@openzeppelin/contracts-ethereum-package/contracts/Initializable.sol";
```

8. Public Function That Could Be Declared External

Risk Level	Note	Likelihood	Low
		Impact	Low
Affected	*GrazingRange.sol*		
Location	*initialize() L:69*		

Observation and Implication

`public` functions that are never called by the contract should be declared `external` to save gas.

Recommendations

Use the `external` attribute for functions never called from the contract.

Evidence of Finding

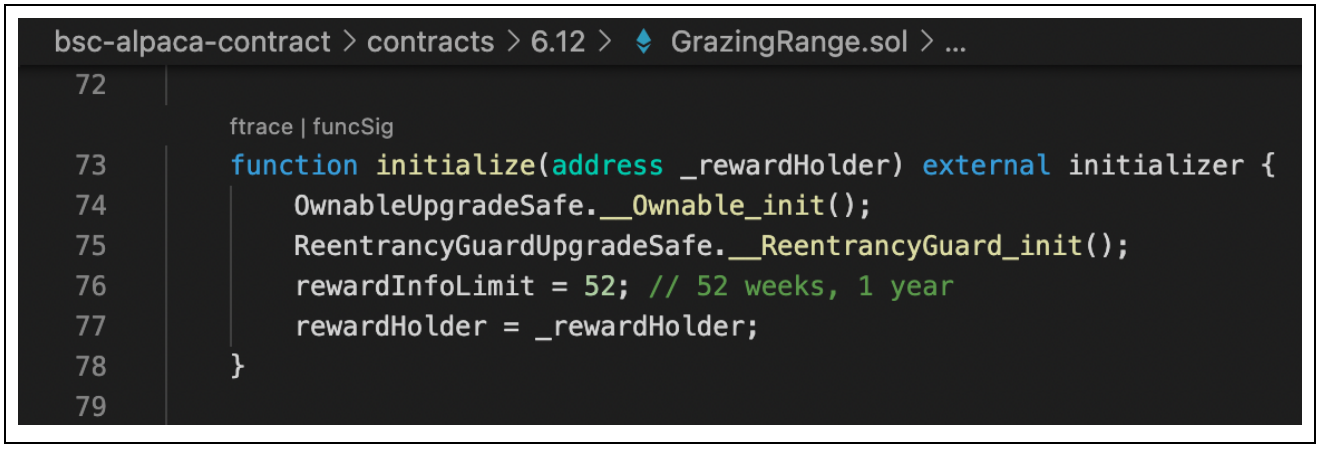
https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L69

```
contracts > 6 > token > GrazingRange.sol > ...
68
        ftrace | funcSig
69      function initialize() public initializer {
70          OwnableUpgradeSafe.__Ownable_init();
71          ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init();
72          rewardInfoLimit = 8;
73      }
74
```

Reassessment Result

The issue was fixed.

https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L73




ValiX