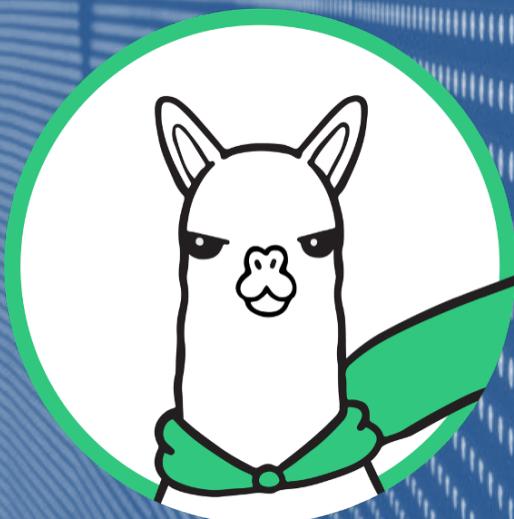


Grazing Range

Smart Contract Security Review

Alpaca Finance



Full Report

5 May 2021

Confidential

valix

Table of Contents

Executive Summary	4
Overview.....	4
Scope of Work	4
Review Result Summary	5
Review Detail	6
Methodology	6
Risk Level Classification.....	7
Review Findings	8
Review Findings Summary	8
Detailed Findings and Recommendations	9

Confidentiality Statement

Our reports are intended solely for the information and use of management and customers of **Alpaca Finance** (“**Alpaca**”) and should not be used for any other purpose. **Alpaca** may not provide access to or copies of this report to any third party without the prior written consent of **Valix Consulting Company Limited** (“**Valix**”).

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	<ul style="list-style-type: none">▪ 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.

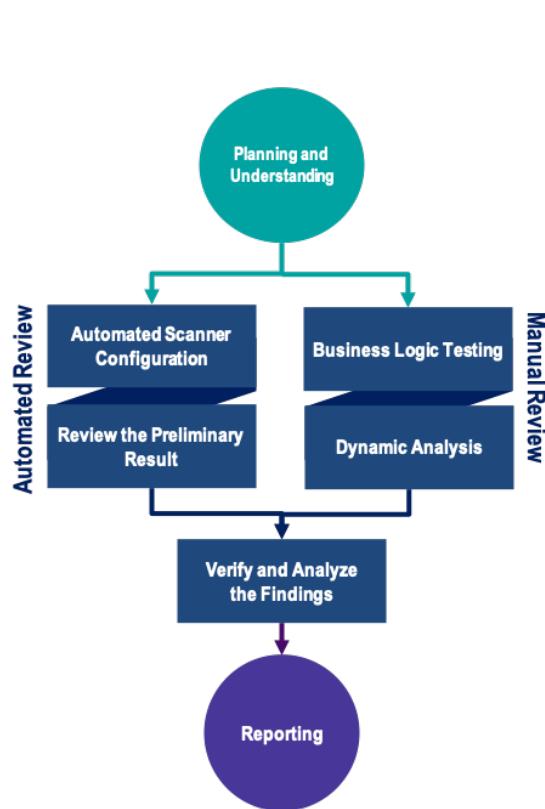
Target	Assessment				Reassessment			
	H	M	L	N	H	M	L	N
Grazing Range Smart Contract	2	3	1	2	0	0	0	0

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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> Testing for business logic flaws requires thinking in unconventional methods Identify unsafe coding behavior via static code analysis
Reporting	<ul style="list-style-type: none"> 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	<i>The code implementation does not match the specification, or it could result in the loss of funds for contract owner or users.</i>
Medium	<i>The code implementation does not match the specification under certain conditions, or it could affect the security standard by loss of access control.</i>
Low	<i>The code implementation does not follow best practices, or use suboptimal design patterns, which may lead to security vulnerabilities further down the line.</i>
Note	<i>Findings in this category are informational and may be further improved by following best practices and guidelines.</i>

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
Affected	<i>GrazingRange.sol</i>		
Location	<i>emergencyRewardWithdraw()</i> 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.
276     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.
316     ftrace|funcSig
317     function emergencyRewardWithdraw(uint256 _campaignID↑, uint256 _amount↑, address _beneficiary↑) external onlyOwner nonReentrant {
318         CampaignInfo storage campaign = campaignInfo[_campaignID↑];
319         uint256 currentStakingPendingReward = _pendingReward(_campaignID↑, campaign.totalStaked, 0);
320         require(currentStakingPendingReward.add(_amount↑) <= campaign.totalRewards, "GrazingRange::emergencyRewardWithdraw::not enough reward token");
321         campaign.totalRewards = campaign.totalRewards.sub(_amount↑);
322         campaign.rewardToken.safeTransfer(_beneficiary↑, _amount↑);
323     }
```

2. State Variable Not Updated (CWE-840)						
Risk Level	High	Likelihood	High			
		Impact	Medium			
Affected	<i>GrazingRange.sol</i>					
Location	<i>emergencydWithdraw() L:266</i>					
Observation and Implication						
The <code>campaign.totalStaked</code> variable was not updated in the <code>emergencyWithdraw()</code> function. This causes all reward calculations after this function call to be incorrect, as the <code>campaign.totalStaked</code> would be more than the actual amount staked in the contract.						
Recommendations						
Deduct <code>campaign.totalStaked</code> by the amount withdrawn.						
Evidence of Finding						
https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L266 <pre>bsc-alpaca-contract > contracts > 6 > token > ♦ GrazingRange.sol > ... 264 265 // @notice Withdraw without caring about rewards. EMERGENCY ONLY. 266 ftrace funcSig 267 function emergencyWithdraw(uint256 _campaignID) external { 268 CampaignInfo storage campaign = campaignInfo[_campaignID]; 269 UserInfo storage user = userInfo[_campaignID][msg.sender]; 270 campaign.stakingToken.safeTransfer(address(msg.sender), user.amount); 271 user.amount = 0; 272 user.rewardDebt = 0; 273 emit EmergencyWithdraw(msg.sender, user.amount, _campaignID); 274 }</pre>						
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.
304     ftrace | funcSig
305     function emergencyWithdraw(uint256 _campaignID↑) external nonReentrant {
306         CampaignInfo storage campaign = [campaignInfo][_campaignID↑];
307         UserInfo storage user = [userInfo][_campaignID↑][msg.sender];
308         uint256 _amount = user.amount;
309         campaign.totalStaked = campaign.totalStaked.sub(_amount);
310         user.amount = 0;
311         user.rewardDebt = 0;
312         campaign.stakingToken.safeTransfer(address(msg.sender), _amount);
313         emit EmergencyWithdraw(msg.sender, _amount, _campaignID↑);
314     }
```

3. Reentrancy (SWC-107)						
Risk Level	Medium	Likelihood	Low			
		Impact	High			
Affected	<i>GrazingRange.sol</i>					
Location	<i>emergencyWithdraw() L:269</i>					
Observation and Implication						
External contract was called in the <i>emergencyWithdraw()</i> 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 <i>stakingToken</i> 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:						
<ul style="list-style-type: none"> - 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 <pre> contracts > 6 > token > ♦ GrazingRange.sol > ... 264 265 // @notice Withdraw without caring about rewards. EMERGENCY ONLY. 266 ftrace funcSig 267 function emergencyWithdraw(uint256 _campaignID↑) external { 268 CampaignInfo storage campaign = campaignInfo[_campaignID↑]; 269 UserInfo storage user = userInfo[_campaignID↑][msg.sender]; 270 campaign.stakingToken.safeTransfer(address(msg.sender), user.amount); 271 user.amount = 0; 272 user.rewardDebt = 0; 273 emit EmergencyWithdraw(msg.sender, user.amount, _campaignID↑); 274 } </pre>						

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.
304     ftrace | funcSig
305     function emergencyWithdraw(uint256 _campaignID↑) external nonReentrant {
306         CampaignInfo storage campaign = [campaignInfo][_campaignID↑];
307         UserInfo storage user = [userInfo][_campaignID↑][msg.sender];
308         uint256 _amount = user.amount;
309         campaign.totalStaked = campaign.totalStaked.sub(_amount);
310         user.amount = 0;
311         user.rewardDebt = 0;
312         campaign.stakingToken.safeTransfer(address(msg.sender), _amount);
313         emit EmergencyWithdraw(msg.sender, _amount, _campaignID↑);
314     }
```

4. Incorrect Pending Reward Calculation (CWE-840)						
Risk Level	Medium	Likelihood	Medium			
Affected	<i>GrazingRange.sol</i>					
Location	<i>pendingReward() L:163</i>					
Observation and Implication						
The value of <code>campaign.lastRewardBlock</code> in <code>pendingReward()</code> function was not updated, causing the multiplier of some reward periods to be calculated repeatedly when the <code>updateCampaign()</code> function has never been called during the current reward period. This would cause incorrect display for the reward value.						
For example:						
If the <code>campaign.lastRewardBlock</code> is before <code>rewardInfo[0].endBlock</code> , and the current <code>block.number</code> is after <code>rewardInfo[1].endBlock</code> and before <code>rewardInfo[2].endBlock</code> .						
Current multiplier calculation:						
<pre>rewardInfo[0]: campaign.lastRewardBlock -> rewardInfo[0].endBlock rewardInfo[1]: campaign.lastRewardBlock -> rewardInfo[1].endBlock rewardInfo[2]: campaign.lastRewardBlock -> block.number</pre>						
What the multiplier should be:						
<pre>rewardInfo[0]: campaign.lastRewardBlock -> rewardInfo[0].endBlock rewardInfo[1]: rewardInfo[0].endBlock -> rewardInfo[1].endBlock rewardInfo[2]: rewardInfo[1].endBlock -> block.number</pre>						
Recommendations						
Update the starting block of multiplier calculation in each loop of <code>rewardInfo</code> 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
Affected	<i>GrazingRange.sol</i>		
Location	<i>_rewardPerBlockOf() L:141</i>		

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

Reassessment Result

The issue was fixed.

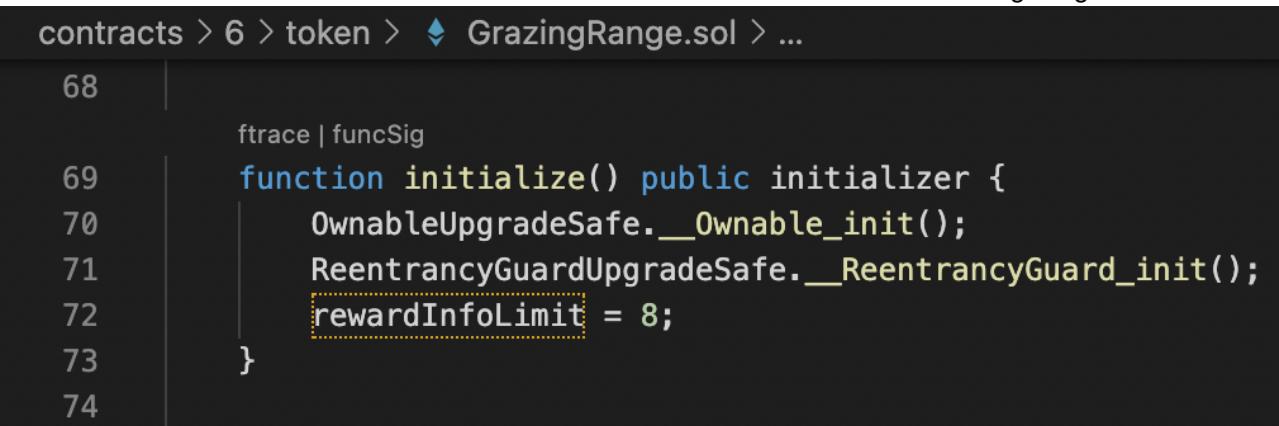
<https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L168>

```
bsc-alpaca-contract > contracts > 6.12 > ♦ GrazingRange.sol > ...
156
157     ftrace | funcSig
158     function _rewardPerBlockOf(uint256 _campaignID↑, uint256 _blockNumber↑) internal view returns (uint256) {
159         RewardInfo[] memory rewardInfo = [campaignRewardInfo[_campaignID↑]];
160         uint256 len = rewardInfo.length;
161         if (len == 0) {
162             return 0;
163         }
164         for (uint256 i = 0; i < len; ++i) {
165             if (_blockNumber↑ <= rewardInfo[i].endBlock) return rewardInfo[i].rewardPerBlock;
166         }
167         // @dev when couldn't find any reward info, it means that timestamp exceed endblock
168         // so return 0
169         return 0;
170     }
```

6. Usage of Outdated Solidity Compiler Version (SWC-102)						
Risk Level	Low	Likelihood	Low			
	Impact	Medium				
Affected	<i>GrazingRange.sol</i>					
Location	L:1					
Observation and Implication						
<p>The compiler version specified in the contract was 0.6.6, which is outdated.</p> <p>Outdated Solidity compiler may have some bugs which is publicly known, that could potentially be used to perform attacks on the smart contract.</p>						
Recommendations						
<p>Consider changing the compiler version to latest stable version.</p> <p>The current lastest stable release (as of April 2021) of Solidity with the major 0.6 is version 0.6.12.</p>						
Reference:						
https://github.com/ethereum/solidity/releases						
Evidence of Finding						
<p>https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L1</p> <pre>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 3 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 4 import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol";</pre>						
Reassessment Result						
<p>The issue was fixed.</p> <p>https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L1</p> <pre>bsc-alpaca-contract > contracts > 6.12 > ♦ GrazingRange.sol > ... report graph (this) graph inheritance parse flatten funcSigs uml 1 pragma solidity 0.6.12;</pre>						

7. Duplicated Imports						
Risk Level	Note	Likelihood	Low			
		Impact	Low			
Affected	<i>GrazingRange.sol</i>					
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						
<p>https://github.com/alpaca-finance/bsc-alpaca-contract/blob/fcd2f3cba29f94049a573fd6464eaa628aaeaf84/contracts/6/token/GrazingRange.sol#L6</p> <pre> contracts > 6 > token > ♦ GrazingRange.sol > ... 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 3 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 4 import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol"; 5 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 6 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol"; 7 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 8 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol"; 9 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 10 import "@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol"; 11 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 12 import "@openzeppelin/contracts-ethereum-package/contracts/access/Ownable.sol"; 13 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 14 import "@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol"; 15 abraham-alpaca, 3 days ago 2 authors (abraham-alpaca and others) 16 import "@openzeppelin/contracts-ethereum-package/contracts/Initializable.sol"; </pre>						
Reassessment Result						
The issue was fixed.						
<p>https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L6</p>						

```
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
3 import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol";
4 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/IERC20.sol";
5 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol";
6 import "@openzeppelin/contracts-ethereum-package/contracts/utils/ReentrancyGuard.sol";
7 import "@openzeppelin/contracts-ethereum-package/contracts/access/Ownable.sol";
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	<i>GrazingRange.sol</i>					
Location	<i>initialize() L:69</i>					
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  <pre> contracts > 6 > token > ⚡ GrazingRange.sol > ... 68 69 ftrace funcSig 70 function initialize() public initializer { 71 OwnableUpgradeSafe.__Ownable_init(); 72 ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init(); 73 rewardInfoLimit = 8; 74 } </pre>						
Reassessment Result						
The issue was fixed.						
https://github.com/alpaca-finance/bsc-alpaca-contract/blob/a2f6250b68094fcac720e29cb676e2c0b2673b83/contracts/6.12/GrazingRange.sol#L73						

```
bsc-alpaca-contract > contracts > 6.12 > ♦ GrazingRange.sol > ...
72
72     ftrace | funcSig
73     function initialize(address _rewardHolder) external initializer {
74         OwnableUpgradeSafe.__Ownable_init();
75         ReentrancyGuardUpgradeSafe.__ReentrancyGuard_init();
76         rewardInfoLimit = 52; // 52 weeks, 1 year
77         rewardHolder = _rewardHolder;
78     }
79 }
```

The background of the image features a subtle, abstract pattern of blue and white. It consists of two main components: a series of thin, light blue diagonal lines that create a sense of motion and depth, and a larger, more organic shape made of white, rounded, oval-like dots that resembles a stylized wave or a network of connections.

valix