MythX

REPORT 5FBB93F8A54FA30019454BDF

Created	Mon Nov 23 2020 10:50:32 GMT+0000 (Coordinated Universal Time)
Number of analyses	21
User	team@akropolis.io

REPORT SUMMARY

Analyses ID	Main source file	Detected vulnerabilities
24ab70b2-d4ac-48d1-a163-c5b797e45378	contracts/core/ModuleNames.sol	1
a27fa560-642e-4ab8-835b-d8831de804b6	@openzeppelin/contracts-ethereum-package/contracts/access/Roles.sol	1
c6cd8a9f-b543-4bbf-9f81-fefddf6e9074	@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol	1
14d554b0-223f-4f84-abfe-c6e983d36ff8	@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol	1
1fd20b2e-2294-4b88-a73e-7c63e598c021	contracts/modules/staking/StakingPoolBase.sol	2
a163c432-54df-4002-a3f5-bc3a0db41b1c	@openzeppelin/upgrades/contracts/Initializable.sol	1
1bacf621-f9f0-4689-95b7-17f1378834ff	contracts/utils/AddressMap.sol	1
401fbddb-0a3d-46a7-a078-7dc4de47d1e4	contracts/modules/staking/StakingPool.sol	1
edfe839e-5c20-4788-b6cc-a70111e61bec	@openzeppelin/contracts-ethereum-package/contracts/access/roles/CapperRole.sol	1
71c53fea-cc55-4c75-954b-e7acd9ef6163	@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol	1
42320149-f103-4bd3-add7-69ddce4c6727	contracts/utils/AddressList.sol	1
0a86260d-59f1-4114-bfb7-f065e0857a5e	@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol	1
d6ec20d3-64df-46dc-9902-c2d5f4dad739	contracts/core/Pool.sol	2
dd1c8ad4-fde9-426a-a79f-b7c341584d1d	contracts/common/Module.sol	1
49534af9-5c22-4617-87a4-a963e2aa4d83	contracts/common/Base.sol	1
70dbe553-eb16-496b-980e-f0e98a600722	contracts/utils/CalcUtils.sol	1
b39e5be9-d766-423c-a436-a03cf0c0df18	contracts/test/FreeERC20.sol	1
34713680-01a8-47ac-8009-9025aed89b89	contracts/modules/reward/RewardManagerRole.sol	1

b19aad51-72a8-4e5d-ab4d-9668a2b5a2cb	@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol	1
f126eb4b-8caf-4bde-b654-ad770888ce64	@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol	1
3a8a4089-dcb9-411e-9930-0e74094edb72	contracts/modules/reward/RewardVestingModule.sol	1

Analysis 24ab70b2-d4ac-48d1-a163-c5b797e45378 MythX

Started	Mon Nov 23 2020 10:58:21 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:30 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Core/ModuleNames.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/core/ModuleNames.sol

Locations

```
1  pragma solidity ^0.5.12;
2
3  /**
```

Analysis a27fa560-642e-4ab8-835b-d8831de804b6 MythX

Started	Mon Nov 23 2020 10:58:22 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:31 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`@Openzeppelin/Contracts-Ethereum-Package/Contracts/Access/Roles.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`@openzeppelin/contracts-ethereum-package/contracts/access/Roles.sol`

Locations

```
1  pragma solidity ^0.5.0;
2
3  /**
```

Analysis c6cd8a9f-b543-4bbf-9f81-fefddf6e9074 MythX

Started	Mon Nov 23 2020 10:58:22 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:31 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	@Openzeppelin/Contracts-Ethereum-Package/Contracts/Token/ERC20/ERC20Burnable.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Burnable.sol

Locations

```
1  pragma solidity ^0.5.0;
2
3  import "@openzeppelin/upgrades/contracts/Initializable.sol";
```

Analysis 14d554b0-223f-4f84-abfe-c6e983d36ff8 MythX

Started	Mon Nov 23 2020 10:58:22 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:31 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	@Openzeppelin/Contracts-Ethereum-Package/Contracts/Math/SafeMath.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN **Arithmetic operation "+" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol
Locations

```
25 */
26 function add(uint256 a, uint256 b) internal pure returns (uint256) {
27 uint256 c = a + b;
28 require(c >= a, "SafeMath: addition overflow");
```

UNKNOWN **Arithmetic operation "-" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol
Locations

```
57 function sub(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
58 require(b <= a, errorMessage);
59 uint256 c = a - b;
60
61 return c;
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol

Locations

```
79  }
80
81  uint256 c = a * b;
82  require(c / a == b, "SafeMath: multiplication overflow");
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol

Locations

```
80
81  uint256 c = a * b;
82  require(c / a == b, "SafeMath: multiplication overflow");
83
84  return c;
```

UNKNOWN Arithmetic operation "/" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol

Locations

```
116 // Solidity only automatically asserts when dividing by 0
117 require(b > 0, errorMessage);
118 uint256 c = a / b;
119 // assert(a == b * c + a % b); // There is no case in which this doesn't hold
```

UNKNOWN Arithmetic operation "%" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol

Locations

```
152 function mod(uint256 a, uint256 b, string memory errorMessage) internal pure returns (uint256) {
153 require(b != 0, errorMessage);
154 return a % b;
155 }
156 }
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol`

Locations

```
pragma solidity ^0.5.0;

/**
```

Analysis 1fd20b2e-2294-4b88-a73e-7c63e598c021 MythX

Started	Mon Nov 23 2020 10:58:32 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:54 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Modules/Staking/StakingPoolBase.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	2

ISSUES

UNKNOWN **Arithmetic operation "-" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
contracts/modules/staking/StakingPoolBase.sol
Locations

```
109
110 if (stakingCapEnabled && !(vipUserEnabled && isVipUser[_msgSender()])) {
111 require((stakingCap > totalStaked() && (stakingCap-totalStaked() >= stake)), "StakingModule: stake exeeds staking cap");
112 }
```

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
contracts/modules/staking/StakingPoolBase.sol
Locations

```
202 function setUserCap(address[] memory users, uint256[] memory caps) public onlyCapper {
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
(,actualAmounts,) = getPersonalStakes(_address);
uint256 totalStake;
for(uint256 i=0; i <actualAmounts.length; i++) {
totalStake = totalStake.add(actualAmounts[i]);
}
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 personalStakeIndex = stakeHolders[_msgSender()].personalStakeIndex;

for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data); withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
361 */
362 function totalScoresFor(address _address) public view returns (uint256) {
363 return stakeHolders[_address].totalStakedFor.mul(coeffScore).div(10**18);
364 }
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
405 StakeContract storage stakeContract = stakeHolders[_address];
406
407 uint256 arraySize = stakeContract.personalStakes.length - stakeContract.personalStakeIndex;
408 uint256[] memory unlockedTimestamps = new uint256[](arraySize); uint256[] memory unlockedTimestamps = new uint256[](arraySize);
409 uint256[] memory actualAmounts = new uint256[](arraySize);
410 address[] memory stakedFor = new address[](arraySize);
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
410 address[] memory stakedFor = new address[](arraySize);
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp; unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
492
493 personalStake.actualAmount = 0;
494 stakeHolders[_msgSender()].personalStakeIndex++;
495
496 totalStakedAmount = totalStakedAmount.sub(_amount);
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
1 pragma solidity ^0.5.12;
2
3 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol";
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "stakingToken" is internal. Other possible visibility settings are public and private.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

// Token used for staking
ERC20 stakingToken;

// The default duration of stake lock-in (in seconds)
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
for(uint256 i=0; i < users.length; i++) {
userCap[users[i]] = caps[i];
emit UserCapChanged(users[i], caps[i]);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
for(uint256 i=0; i < users.length; i++) {
userCap[users[i]] = caps[i];
emit UserCapChanged(users[i], caps[i]);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
269 uint256 totalStake;
270 for(uint256 i=0; i <actualAmounts.length; i++) {
271 totalStake = totalStake.add(actualAmounts[i]);
272 }
273 return totalStake;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
}
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
470 internal isUserCapEnabledForUnStakeFor(_amount)
471 {
472 Stake storage personalStake = stakeHolders[_msgSender()].personalStakes[stakeHolders[_msgSender()].personalStakeIndex];
473
474 // Check that the current stake has unlocked & matches the unstake amount
475 require(
```

Analysis a163c432-54df-4002-a3f5-bc3a0db41b1c MythX

Started	Mon Nov 23 2020 10:58:32 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:41 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`@Openzeppelin/Upgrades/Contracts/Initializable.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""=0.4.24<0.7.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`@openzeppelin/upgrades/contracts/Initializable.sol`

Locations

```
1 pragma solidity >=0.4.24 <0.7.0;
2
```

Analysis 1bacf621-f9f0-4689-95b7-17f1378834ff MythX

Started	Mon Nov 23 2020 10:58:32 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:41 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`Contracts/Utils/AddressMap.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/utils/AddressList.sol`

Locations

```
55 }
56 _data.isContain[_item] = true;
57 ++_data.length;
58 }
```

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/utils/AddressList.sol`

Locations

```
95 }
96 _data.isContain[_item] = true;
97 ++_data.length;
98 }
```

UNKNOWN

SWC-101

Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/utils/AddressList.sol`

Locations

```

_data.isContain[_item] = false;
--_data.length;
}
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/utils/AddressMap.sol`

Locations

```
pragma solidity ^0.5.12;

import "./AddressList.sol";
```

Analysis 401fbddb-0a3d-46a7-a078-7dc4de47d1e4 MythX

Started	Mon Nov 23 2020 10:58:32 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:52 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Modules/Staking/StakingPool.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN Arithmetic operation "++" discovered

SWC-101

This plugin produces issues to support false positive discovery within MythX.

Source file
contracts/modules/staking/StakingPool.sol
Locations

```
54
55 function isRegisteredRewardToken(address token) public view returns(bool) {
56 for(uint256 i=0; i<registeredRewardTokens.length; i++){
57 if(token == registeredRewardTokens[i]) return true;
58 }
```

UNKNOWN Arithmetic operation "++" discovered

SWC-101

This plugin produces issues to support false positive discovery within MythX.

Source file
contracts/modules/staking/StakingPool.sol
Locations

```
85 UserRewardInfo storage uri = userRewards[user];
86 uint256 reward;
87 for(uint256 i=uri.nextDistribution[token]; i < rd.distributions.length; i++) {
88 RewardDistribution storage rdistr = rd.distributions[i];
89 uint256 r = shares.mul(rdistr.amount).div(rdistr.totalShares);
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/modules/staking/StakingPool.sol`

Locations

```
95  function _withdrawRewards(address user) internal returns(uint256[] memory rwrds) {
96  rwrds = new uint256[](registeredRewardTokens.length);
97  for(uint256 i=0; i<registeredRewardTokens.length; i++){
98  rwrds[i] = _withdrawRewards(user, registeredRewardTokens[i]);
99  }
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/modules/staking/StakingPool.sol`

Locations

```
130 function _claimRewardsFromVesting() internal {
131 rewardVesting.claimRewards();
132 for(uint256 i=0; i < registeredRewardTokens.length; i++){
133 address rt = registeredRewardTokens[i];
134 uint256 expectedBalance = rewards[rt].unclaimed;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
109

110 if (stakingCapEnabled && !(vipUserEnabled && isVipUser[_msgSender()])) {
111 require((stakingCap > totalStaked() && (stakingCap-totalStaked() >= stake)), "StakingModule: stake exeeds staking cap");
112 }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
202 function setUserCap(address[] memory users, uint256[] memory caps) public onlyCapper {
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
268 (,actualAmounts,) = getPersonalStakes(_address);
269 uint256 totalStake;
270 for(uint256 i=0; i <actualAmounts.length; i++) {
271 totalStake = totalStake.add(actualAmounts[i]);
272 }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
335 uint256 personalStakeIndex = stakeHolders[_msgSender()].personalStakeIndex;
336
337 for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {
338
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
```

UNKNOWN

SWC-101

Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
338
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
340 unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
341 withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data); withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
342 }
343 }
```

UNKNOWN

SWC-101

Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
361 */
362 function totalScoresFor(address _address) public view returns (uint256) {
363 return stakeHolders[_address].totalStakedFor.mul(coeffScore).div(10**18);
364 }
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
405 StakeContract storage stakeContract = stakeHolders[_address];
406
407 uint256 arraySize = stakeContract.personalStakes.length - stakeContract.personalStakeIndex;
408 uint256[] memory unlockedTimestamps = new uint256[](arraySize); uint256[] memory unlockedTimestamps = new uint256[](arraySize);
409 uint256[] memory actualAmounts = new uint256[](arraySize);
410 address[] memory stakedFor = new address[](arraySize);
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
410 address[] memory stakedFor = new address[](arraySize);
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp; unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
492
493 personalStake.actualAmount = 0;
494 stakeHolders[_msgSender()].personalStakeIndex++;
495
496 totalStakedAmount = totalStakedAmount.sub(_amount);
```

UNKNOWN

Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
Module.initialize(_pool);
RewardManagerRole.initialize(_msgSender());
defaultEpochLength = 7*24*60*60;
}
```

UNKNOWN

Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
Module.initialize(_pool);
RewardManagerRole.initialize(_msgSender());
defaultEpochLength = 7*24*60*60;
}
```

UNKNOWN

Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
Module.initialize(_pool);
RewardManagerRole.initialize(_msgSender());
defaultEpochLength = 7*24*60*60;
}
```

UNKNOWN

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
epochStart = 0;
}else {
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
RewardInfo storage ri = r.rewardInfo[token];
require(ri.epochs.length > 0, "RewardVesting: protocol or token not registered");
return ri.epochs.length-1;
}
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
//require(r.tokens.length > 0, "RewardVesting: call only from registered protocols allowed");
if(r.tokens.length == 0) return; //This allows claims from protocols which are not yet registered without reverting
for(uint256 i=0; i < r.tokens.length; i++){
_claimRewards(protocol, r.tokens[i]);
}
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
require(epochsLength > 0, "RewardVesting: protocol or token not registered");

Epoch storage lastEpoch = ri.epochs[epochsLength-1];
uint256 previousClaim = ri.lastClaim;
if(previousClaim == lastEpoch.end) return; // Nothing to claim yet
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
120 uint256 i;
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
120 uint256 i;
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
128 }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
132 if(ep.end > block.timestamp) {
133 //Half-claim
134 uint256 epStart = ri.epochs[i-1].end;
135 uint256 claimStart = (previousClaim > epStart)?previousClaim:epStart;
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
137 claimAmount = claimAmount.add(epochClaim);
138 i--;
139 }
140 //Claim rest
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
139 }
140 //Claim rest
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
145 if(previousClaim > epStart) {
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
162 uint256 epochsLength = ri.epochs.length;
163 require(epochsLength > 0, "RewardVesting: protocol or token not registered");
164 uint256 prevEpochEnd = ri.epochs[epochsLength-1].end;
165 require(epochEnd > prevEpochEnd, "RewardVesting: new epoch should end after previous");
166 ri.epochs.push(Epoch({
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
(protocols.length == amounts.length),
"RewardVesting: array lengths do not match");
for(uint256 i=0; i<protocols.length; i++) {
_addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
}
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
if(epoch == 0) epoch = epochsLength; // creating a new epoch
if (epoch == epochsLength) {
uint256 epochEnd = ri.epochs[epochsLength-1].end.add(defaultEpochLength);
if(epochEnd < block.timestamp) epochEnd = block.timestamp; //This generally should not happen, but just in case - we generate only one epoch since previous end
ri.epochs.push(Epoch({
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
epochStart = 0;
}else {
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
RewardInfo storage ri = r.rewardInfo[token];
require(ri.epochs.length > 0, "RewardVesting: protocol or token not registered");
return ri.epochs.length-1;
}
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
require(epochsLength > 0, "RewardVesting: protocol or token not registered");

Epoch storage lastEpoch = ri.epochs[epochsLength-1];
uint256 previousClaim = ri.lastClaim;
if(previousClaim == lastEpoch.end) return; // Nothing to claim yet
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
uint256 i;
// Searching for last claimable epoch
for(i = epochsLength-1; i > 0; i--) {
ep = ri.epochs[i];
if(ep.end < block.timestamp) { // We've found last fully-finished epoch
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
132 if(ep.end > block.timestamp) {
133 //Half-claim
134 uint256 epStart = ri.epochs[i-1].end;
135 uint256 claimStart = (previousClaim > epStart)?previousClaim:epStart;
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
145 if(previousClaim > epStart) {
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
162 uint256 epochsLength = ri.epochs.length;
163 require(epochsLength > 0, "RewardVesting: protocol or token not registered");
164 uint256 prevEpochEnd = ri.epochs[epochsLength-1].end;
165 require(epochEnd > prevEpochEnd, "RewardVesting: new epoch should end after previous");
166 ri.epochs.push(Epoch({
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
200 if(epoch == 0) epoch = epochsLength; // creating a new epoch
201 if (epoch == epochsLength) {
202 uint256 epochEnd = ri.epochs[epochsLength-1].end.add(defaultEpochLength);
203 if(epochEnd < block.timestamp) epochEnd = block.timestamp; //This generally should not happen, but just in case - we generate only one epoch since previous end
204 ri.epochs.push(Epoch({
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/modules/staking/StakingPool.sol

Locations

```
1 pragma solidity ^0.5.12;
2
3 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol";
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPool.sol

Locations

```
55 function isRegisteredRewardToken(address token) public view returns(bool) {
56 for(uint256 i=0; i<registeredRewardTokens.length; i++){
57 if(token == registeredRewardTokens[i]) return true;
58 }
59 return false;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPool.sol`

Locations

```
uint256 reward;
for(uint256 i=uri.nextDistribution[token]; i < rd.distributions.length; i++) {
RewardDistribution storage rdistr = rd.distributions[i];
uint256 r = shares.mul(rdistr.amount).div(rdistr.totalShares);
reward = reward.add(r);
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPool.sol`

Locations

```
rwrds = new uint256[](registeredRewardTokens.length);
for(uint256 i=0; i<registeredRewardTokens.length; i++){
rwrds[i] = _withdrawRewards(user, registeredRewardTokens[i]);
}
return rwrds;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPool.sol`

Locations

```
rwrds = new uint256[](registeredRewardTokens.length);
for(uint256 i=0; i<registeredRewardTokens.length; i++){
rwrds[i] = _withdrawRewards(user, registeredRewardTokens[i]);
}
return rwrds;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPool.sol`

Locations

```
rewardVesting.claimRewards();
for(uint256 i=0; i < registeredRewardTokens.length; i++){
address rt = registeredRewardTokens[i];
uint256 expectedBalance = rewards[rt].unclaimed;
if(rt == address(stakingToken)){
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
for(uint256 i=0; i < users.length; i++) {
userCap[users[i]] = caps[i];
emit UserCapChanged(users[i], caps[i]);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
for(uint256 i=0; i < users.length; i++) {
userCap[users[i]] = caps[i];
emit UserCapChanged(users[i], caps[i]);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
269 uint256 totalStake;
270 for(uint256 i=0; i <actualAmounts.length; i++) {
271 totalStake = totalStake.add(actualAmounts[i]);
272 }
273 return totalStake;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN Out of bounds array access

SWC-110

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN Out of bounds array access

SWC-110

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN Out of bounds array access

SWC-110

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
internal isUserCapEnabledForUnStakeFor(_amount)
{
Stake storage personalStake = stakeHolders[_msgSender()].personalStakes[stakeHolders[_msgSender()].personalStakeIndex];

// Check that the current stake has unlocked & matches the unstake amount
require(
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
epochStart = 0;
}else {
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
rewardAmount = ri.epochs[epoch].amount;
return (epochStart, epochEnd, rewardAmount);
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
75 }
76 epochEnd = ri.epochs[epoch].end;
77 rewardAmount = ri.epochs[epoch].amount;
78 return (epochStart, epochEnd, rewardAmount);
79 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
92 if(r.tokens.length == 0) return; //This allows claims from protocols which are not yet registered without reverting
93 for(uint256 i=0; i < r.tokens.length; i++){
94 _claimRewards(protocol, r.tokens[i]);
95 }
96 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
106 require(epochsLength > 0, "RewardVesting: protocol or token not registered");
107
108 Epoch storage lastEpoch = ri.epochs[epochsLength-1];
109 uint256 previousClaim = ri.lastClaim;
110 if(previousClaim == lastEpoch.end) return; // Nothing to claim yet
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
117
118 uint256 claimAmount;
119 Epoch storage ep = ri.epochs[0];
120 uint256 i;
121 // Searching for last claimable epoch
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
128 }
129 break;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
132 if(ep.end > block.timestamp) {
133 //Half-claim
134 uint256 epStart = ri.epochs[i-1].end;
135 uint256 claimStart = (previousClaim > epStart)?previousClaim:epStart;
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
140 //Claim rest
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
145 if(previousClaim > epStart) {
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
uint256 epochsLength = ri.epochs.length;
require(epochsLength > 0, "RewardVesting: protocol or token not registered");
uint256 prevEpochEnd = ri.epochs[epochsLength-1].end;
require(epochEnd > prevEpochEnd, "RewardVesting: new epoch should end after previous");
ri.epochs.push(Epoch({
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
"RewardVesting: array lengths do not match");
for(uint256 i=0; i<protocols.length; i++) {
_addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
}
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
"RewardVesting: array lengths do not match");
for(uint256 i=0; i<protocols.length; i++) {
_addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
}
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
182 "RewardVesting: array lengths do not match");
183 for(uint256 i=0; i<protocols.length; i++) {
184 _addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
185 }
186 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
182 "RewardVesting: array lengths do not match");
183 for(uint256 i=0; i<protocols.length; i++) {
184 _addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
185 }
186 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
200 if(epoch == 0) epoch = epochsLength; // creating a new epoch
201 if (epoch == epochsLength) {
202 uint256 epochEnd = ri.epochs[epochsLength-1].end.add(defaultEpochLength);
203 if(epochEnd < block.timestamp) epochEnd = block.timestamp; //This generally should not happen, but just in case - we generate only one epoch since previous end
204 ri.epochs.push(Epoch({
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
208 } else {
209 require(epochsLength > epoch, "RewardVesting: epoch is too high");
210 Epoch storage ep = ri.epochs[epoch];
211 require(ep.end > block.timestamp, "RewardVesting: epoch already finished");
212 ep.amount = ep.amount.add(amount);
```

Analysis edfe839e-5c20-4788-b6cc-a70111e61bec MythX

Started	Mon Nov 23 2020 10:58:32 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:43 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`@Openzeppelin/Contracts-Ethereum-Package/Contracts/Access/Roles/CapperRole.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`@openzeppelin/contracts-ethereum-package/contracts/access/roles/CapperRole.sol`

Locations

```
1  pragma solidity ^0.5.0;
2
3  import "@openzeppelin/upgrades/contracts/Initializable.sol";
```

Analysis 71c53fea-cc55-4c75-954b-e7acd9ef6163 MythX

Started	Mon Nov 23 2020 10:58:42 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:51 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	@Openzeppelin/Contracts-Ethereum-Package/Contracts/Token/ERC20/SafeERC20.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/SafeERC20.sol

Locations

```
1  pragma solidity ^0.5.0;
2
3  import "./IERC20.sol";
```

Analysis 42320149-f103-4bd3-add7-69ddce4c6727 MythX

Started	Mon Nov 23 2020 10:58:42 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:52 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Utils/AddressList.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/utils/AddressList.sol

Locations

```
55 }
56 _data.isContain[_item] = true;
57 ++_data.length;
58 }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/utils/AddressList.sol

Locations

```
95 }
96 _data.isContain[_item] = true;
97 ++_data.length;
98 }
```

UNKNOWN

SWC-101

Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

`contracts/utils/AddressList.sol`

Locations

```
122
123 _data.isContain[_item] = false;
124 --_data.length;
125 }
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`contracts/utils/AddressList.sol`

Locations

```
1 pragma solidity ^0.5.12;
2 /**
3 * @dev Double linked list with address items
```

Analysis `0a86260d-59f1-4114-bfb7-f065e0857a5e` MythX

Started	Mon Nov 23 2020 10:58:42 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:52 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`@Openzeppelin/Contracts-Ethereum-Package/Contracts/Utils/Address.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.5"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

`@openzeppelin/contracts-ethereum-package/contracts/utils/Address.sol`

Locations

```
1  pragma solidity ^0.5.5;
2
3  /**
```

Analysis d6ec20d3-64df-46dc-9902-c2d5f4dad739 MythX

Started	Mon Nov 23 2020 10:58:42 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:57 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Core/Pool.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	2

ISSUES

UNKNOWN **Arithmetic operation "-" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
contracts/modules/staking/StakingPoolBase.sol
Locations

```
109
110 if (stakingCapEnabled && !(vipUserEnabled && isVipUser[_msgSender()])) {
111 require((stakingCap > totalStaked() && (stakingCap-totalStaked() >= stake)), "StakingModule: stake exeeds staking cap");
112 }
```

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
contracts/modules/staking/StakingPoolBase.sol
Locations

```
202 function setUserCap(address[] memory users, uint256[] memory caps) public onlyCapper {
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
(,actualAmounts,) = getPersonalStakes(_address);
uint256 totalStake;
for(uint256 i=0; i <actualAmounts.length; i++) {
totalStake = totalStake.add(actualAmounts[i]);
}
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 personalStakeIndex = stakeHolders[_msgSender()].personalStakeIndex;

for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data); withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
361 */
362 function totalScoresFor(address _address) public view returns (uint256) {
363 return stakeHolders[_address].totalStakedFor.mul(coeffScore).div(10**18);
364 }
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
405 StakeContract storage stakeContract = stakeHolders[_address];
406
407 uint256 arraySize = stakeContract.personalStakes.length - stakeContract.personalStakeIndex;
408 uint256[] memory unlockedTimestamps = new uint256[](arraySize); uint256[] memory unlockedTimestamps = new uint256[](arraySize);
409 uint256[] memory actualAmounts = new uint256[](arraySize);
410 address[] memory stakedFor = new address[](arraySize);
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
410 address[] memory stakedFor = new address[](arraySize);
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp; unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
492
493 personalStake.actualAmount = 0;
494 stakeHolders[_msgSender()].personalStakeIndex++;
495
496 totalStakedAmount = totalStakedAmount.sub(_amount);
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/core/Pool.sol

Locations

```
1 pragma solidity ^0.5.12;
2
3 import "../common/Base.sol";
```

LOW

SWC-108

State variable visibility is not set.

It is best practice to set the visibility of state variables explicitly. The default visibility for "modules" is internal. Other possible visibility settings are public and private.

Source file

`contracts/core/Pool.sol`

Locations

```

/* Modules map */
AddressMap.Data modules;

using AddressList for AddressList.Data;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
for(uint256 i=0; i < users.length; i++) {
userCap[users[i]] = caps[i];
emit UserCapChanged(users[i], caps[i]);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
require(users.length == caps.length, "SavingsModule: arrays length not match");
for(uint256 i=0; i < users.length; i++) {
userCap[users[i]] = caps[i];
emit UserCapChanged(users[i], caps[i]);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
269 uint256 totalStake;
270 for(uint256 i=0; i <actualAmounts.length; i++) {
271 totalStake = totalStake.add(actualAmounts[i]);
272 }
273 return totalStake;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
}
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
470 internal isUserCapEnabledForUnStakeFor(_amount)
471 {
472 Stake storage personalStake = stakeHolders[_msgSender()].personalStakes[stakeHolders[_msgSender()].personalStakeIndex];
473
474 // Check that the current stake has unlocked & matches the unstake amount
475 require(
```

Analysis dd1c8ad4-fde9-426a-a79f-b7c341584d1d MythX

Started	Mon Nov 23 2020 10:58:52 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:44:00 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Common/Module.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN Arithmetic operation "-" discovered

SWC-101

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
109
110 if (stakingCapEnabled && !(vipUserEnabled && isVipUser[_msgSender()])) {
111 require((stakingCap > totalStaked() && (stakingCap-totalStaked() >= stake)), "StakingModule: stake exeeds staking cap");
112 }
```

UNKNOWN Arithmetic operation "++" discovered

SWC-101

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
202 function setUserCap(address[] memory users, uint256[] memory caps) public onlyCapper {
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
```

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
268 (,actualAmounts,) = getPersonalStakes(_address);
269 uint256 totalStake;
270 for(uint256 i=0; i <actualAmounts.length; i++) {
271 totalStake = totalStake.add(actualAmounts[i]);
272 }
```

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
335 uint256 personalStakeIndex = stakeHolders[_msgSender()].personalStakeIndex;
336
337 for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {
338
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
```

UNKNOWN **Arithmetic operation "+" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
338
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
340 unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
341 withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data); withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
342 }
343 }
```

UNKNOWN

SWC-101

Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
361 */
362 function totalScoresFor(address _address) public view returns (uint256) {
363 return stakeHolders[_address].totalStakedFor.mul(coeffScore).div(10**18);
364 }
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
405 StakeContract storage stakeContract = stakeHolders[_address];
406
407 uint256 arraySize = stakeContract.personalStakes.length - stakeContract.personalStakeIndex;
408 uint256[] memory unlockedTimestamps = new uint256[](arraySize); uint256[] memory unlockedTimestamps = new uint256[](arraySize);
409 uint256[] memory actualAmounts = new uint256[](arraySize);
410 address[] memory stakedFor = new address[](arraySize);
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
410 address[] memory stakedFor = new address[](arraySize);
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp; unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
492
493 personalStake.actualAmount = 0;
494 stakeHolders[_msgSender()].personalStakeIndex++;
495
496 totalStakedAmount = totalStakedAmount.sub(_amount);
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/common/Module.sol

Locations

```
1 pragma solidity ^0.5.12;
2
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 totalStake;
for(uint256 i=0; i <actualAmounts.length; i++) {
totalStake = totalStake.add(actualAmounts[i]);
}
return totalStake;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
340 unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
341 withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
342 }
343 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
417 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
470 internal isUserCapEnabledForUnStakeFor(_amount)
471 {
472 Stake storage personalStake = stakeHolders[_msgSender()].personalStakes[stakeHolders[_msgSender()].personalStakeIndex];
473
474 // Check that the current stake has unlocked & matches the unstake amount
475 require(
```

Analysis 49534af9-5c22-4617-87a4-a963e2aa4d83 MythX

Started	Mon Nov 23 2020 10:58:52 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:44:01 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Common/Base.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/common/Base.sol

Locations

```
1 pragma solidity ^0.5.12;
2
```

Analysis 70dbe553-eb16-496b-980e-f0e98a600722 MythX

Started	Mon Nov 23 2020 10:58:52 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:58 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	`Contracts/Utils/CalcUtils.Sol`

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN **Arithmetic operation "**" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
`contracts/utils/CalcUtils.sol`
Locations

```
12 return amount;
13 } else if (decimals > 18) {
14 return amount.div(uint256(10)**(decimals-18));
15 } else if (decimals < 18) {
16 return amount.mul(uint256(10)**(18 - decimals));
```

UNKNOWN **Arithmetic operation "-" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file
`contracts/utils/CalcUtils.sol`
Locations

```
12 return amount;
13 } else if (decimals > 18) {
14 return amount.div(uint256(10)**(decimals-18));
15 } else if (decimals < 18) {
16 return amount.mul(uint256(10)**(18 - decimals));
```

UNKNOWN Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/utils/CalcUtils.sol

Locations

```
14 return amount.div(uint256(10)**(decimals-18));
15 } else if (decimals < 18) {
16 return amount.mul(uint256(10)**(18 - decimals));
17 }
18 }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/utils/CalcUtils.sol

Locations

```
14 return amount.div(uint256(10)**(decimals-18));
15 } else if (decimals < 18) {
16 return amount.mul(uint256(10)**(18 - decimals));
17 }
18 }
```

UNKNOWN Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/utils/CalcUtils.sol

Locations

```
23 return amount;
24 } else if (decimals > 18) {
25 return amount.mul(uint256(10)**(decimals-18));
26 } else if (decimals < 18) {
27 return amount.div(uint256(10)**(18 - decimals));
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/utils/CalcUtils.sol

Locations

```
23 return amount;
24 } else if (decimals > 18) {
25 return amount.mul(uint256(10)**(decimals-18));
26 } else if (decimals < 18) {
27 return amount.div(uint256(10)**(18 - decimals));
```

UNKNOWN

SWC-101

Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/utils/CalcUtils.sol

Locations

```
25 return amount.mul(uint256(10)**(decimals-18));
26 } else if (decimals < 18) {
27 return amount.div(uint256(10)**(18 - decimals));
28 }
29 }
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/utils/CalcUtils.sol

Locations

```
25 return amount.mul(uint256(10)**(decimals-18));
26 } else if (decimals < 18) {
27 return amount.div(uint256(10)**(18 - decimals));
28 }
29 }
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/utils/CalcUtils.sol

Locations

```
1 pragma solidity ^0.5.12;
2
3 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol";
```

Analysis b39e5be9-d766-423c-a436-a03cf0c0df18 MythX

Started	Mon Nov 23 2020 10:58:52 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:44:23 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Test/FreeERC20.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
109
110 if (stakingCapEnabled && !(vipUserEnabled && isVipUser[_msgSender()])) {
111 require((stakingCap > totalStaked() && (stakingCap-totalStaked() >= stake)), "StakingModule: stake exeeds staking cap");
112 }
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
202 function setUserCap(address[] memory users, uint256[] memory caps) public onlyCapper {
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
(,actualAmounts,) = getPersonalStakes(_address);
uint256 totalStake;
for(uint256 i=0; i <actualAmounts.length; i++) {
totalStake = totalStake.add(actualAmounts[i]);
}
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 personalStakeIndex = stakeHolders[_msgSender()].personalStakeIndex;

for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
```

UNKNOWN Arithmetic operation "+" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```

if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data); withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
}
}
```

UNKNOWN

SWC-101

Arithmetic operation "**" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
361 */
362 function totalScoresFor(address _address) public view returns (uint256) {
363 return stakeHolders[_address].totalStakedFor.mul(coeffScore).div(10**18);
364 }
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
405 StakeContract storage stakeContract = stakeHolders[_address];
406
407 uint256 arraySize = stakeContract.personalStakes.length - stakeContract.personalStakeIndex;
408 uint256[] memory unlockedTimestamps = new uint256[](arraySize); uint256[] memory unlockedTimestamps = new uint256[](arraySize);
409 uint256[] memory actualAmounts = new uint256[](arraySize);
410 address[] memory stakedFor = new address[](arraySize);
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
410 address[] memory stakedFor = new address[](arraySize);
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
```

UNKNOWN

SWC-101

Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
411
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp; unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN

SWC-101

Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
492
493 personalStake.actualAmount = 0;
494 stakeHolders[_msgSender()].personalStakeIndex++;
495
496 totalStakedAmount = totalStakedAmount.sub(_amount);
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/test/FreeERC20.sol

Locations

```
1 pragma solidity ^0.5.0;
2
3 import "@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20Detailed.sol";
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
203 require(users.length == caps.length, "SavingsModule: arrays length not match");
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
204 for(uint256 i=0; i < users.length; i++) {
205 userCap[users[i]] = caps[i];
206 emit UserCapChanged(users[i], caps[i]);
207 }
208 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
269 uint256 totalStake;
270 for(uint256 i=0; i <actualAmounts.length; i++) {
271 totalStake = totalStake.add(actualAmounts[i]);
272 }
273 return totalStake;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
337 for(uint256 i=personalStakeIndex; i<stakeHolders[_msgSender()].personalStakes.length; i++) {
338
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
340 unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
341 withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
338
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
340 unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
341 withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
342 }
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
339 if (stakeHolders[_msgSender()].personalStakes[i].unlockedTimestamp <= block.timestamp) {
340 unstakeAllAmount = unstakeAllAmount+stakeHolders[_msgSender()].personalStakes[i].actualAmount;
341 withdrawStake(stakeHolders[_msgSender()].personalStakes[i].actualAmount, _data);
342 }
343 }
```

UNKNOWN **Out of bounds array access**

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

`contracts/modules/staking/StakingPoolBase.sol`

Locations

```
412 for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
413 uint256 index = i - stakeContract.personalStakeIndex;
414 unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
415 actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
416 stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
for (uint256 i = stakeContract.personalStakeIndex; i < stakeContract.personalStakes.length; i++) {
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
uint256 index = i - stakeContract.personalStakeIndex;
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
unlockedTimestamps[index] = stakeContract.personalStakes[i].unlockedTimestamp;
actualAmounts[index] = stakeContract.personalStakes[i].actualAmount;
stakedFor[index] = stakeContract.personalStakes[i].stakedFor;
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/staking/StakingPoolBase.sol

Locations

```
internal isUserCapEnabledForUnStakeFor(_amount)
{
Stake storage personalStake = stakeHolders[_msgSender()].personalStakes[stakeHolders[_msgSender()].personalStakeIndex];

// Check that the current stake has unlocked & matches the unstake amount
require(
```

Analysis 34713680-01a8-47ac-8009-9025aed89b89 MythX

Started	Mon Nov 23 2020 10:58:52 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:43:57 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Modules/Reward/RewardManagerRole.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/modules/reward/RewardManagerRole.sol

Locations

```
1  pragma solidity ^0.5.12;
2
3  import "@openzeppelin/upgrades/contracts/Initializable.sol";
```

Analysis b19aad51-72a8-4e5d-ab4d-9668a2b5a2cb MythX

Started	Mon Nov 23 2020 10:59:03 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:44:12 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	@Openzeppelin/Contracts-Ethereum-Package/Contracts/Ownership/Ownable.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

@openzeppelin/contracts-ethereum-package/contracts/ownership/Ownable.sol

Locations

```
1  pragma solidity ^0.5.0;
2
3  import "@openzeppelin/upgrades/contracts/Initializable.sol";
```

Analysis f126eb4b-8caf-4bde-b654-ad770888ce64 MythX

Started	Mon Nov 23 2020 10:59:03 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:44:13 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	@Openzeppelin/Contracts-Ethereum-Package/Contracts/Token/ERC20/ERC20.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.0"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

@openzeppelin/contracts-ethereum-package/contracts/token/ERC20/ERC20.sol

Locations

```
1  pragma solidity ^0.5.0;
2
3  import "@openzeppelin/upgrades/contracts/Initializable.sol";
```

Analysis 3a8a4089-dcb9-411e-9930-0e74094edb72 MythX

Started	Mon Nov 23 2020 10:59:03 GMT+0000 (Coordinated Universal Time)
Finished	Mon Nov 23 2020 11:44:44 GMT+0000 (Coordinated Universal Time)
Mode	Deep
Client Tool	Mythx-Cli-0.6.22
Main Source File	Contracts/Modules/Reward/RewardVestingModule.Sol

DETECTED VULNERABILITIES

HIGH	MEDIUM	LOW
0	0	1

ISSUES

UNKNOWN Arithmetic operation "*" discovered

SWC-101

This plugin produces issues to support false positive discovery within MythX.

Source file
contracts/modules/reward/RewardVestingModule.sol
Locations

```
37 Module.initialize(_pool);
38 RewardManagerRole.initialize(_msgSender());
39 defaultEpochLength = 7*24*60*60;
40 }
```

UNKNOWN Arithmetic operation "*" discovered

SWC-101

This plugin produces issues to support false positive discovery within MythX.

Source file
contracts/modules/reward/RewardVestingModule.sol
Locations

```
37 Module.initialize(_pool);
38 RewardManagerRole.initialize(_msgSender());
39 defaultEpochLength = 7*24*60*60;
40 }
```

UNKNOWN Arithmetic operation "*" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
Module.initialize(_pool);
RewardManagerRole.initialize(_msgSender());
defaultEpochLength = 7*24*60*60;
}
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
epochStart = 0;
}else {
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
RewardInfo storage ri = r.rewardInfo[token];
require(ri.epochs.length > 0, "RewardVesting: protocol or token not registered");
return ri.epochs.length-1;
}
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
//require(r.tokens.length > 0, "RewardVesting: call only from registered protocols allowed");
if(r.tokens.length == 0) return; //This allows claims from protocols which are not yet registered without reverting
for(uint256 i=0; i < r.tokens.length; i++){
_claimRewards(protocol, r.tokens[i]);
}
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
106 require(epochsLength > 0, "RewardVesting: protocol or token not registered");
107
108 Epoch storage lastEpoch = ri.epochs[epochsLength-1];
109 uint256 previousClaim = ri.lastClaim;
110 if(previousClaim == lastEpoch.end) return; // Nothing to claim yet
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
120 uint256 i;
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
120 uint256 i;
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
```

UNKNOWN Arithmetic operation "++" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
128 }
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
132 if(ep.end > block.timestamp) {
133 //Half-claim
134 uint256 epStart = ri.epochs[i-1].end;
135 uint256 claimStart = (previousClaim > epStart)?previousClaim:epStart;
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
137 claimAmount = claimAmount.add(epochClaim);
138 i--;
139 }
140 //Claim rest
```

UNKNOWN Arithmetic operation "--" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
139 }
140 //Claim rest
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
```

UNKNOWN Arithmetic operation "-" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
145 if(previousClaim > epStart) {
```

UNKNOWN **Arithmetic operation "-" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
uint256 epochsLength = ri.epochs.length;
require(epochsLength > 0, "RewardVesting: protocol or token not registered");
uint256 prevEpochEnd = ri.epochs[epochsLength-1].end;
require(epochEnd > prevEpochEnd, "RewardVesting: new epoch should end after previous");
ri.epochs.push(Epoch({
```

UNKNOWN **Arithmetic operation "++" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
(protocols.length == amounts.length),
"RewardVesting: array lengths do not match");
for(uint256 i=0; i<protocols.length; i++) {
_addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
}
```

UNKNOWN **Arithmetic operation "-" discovered**

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
if(epoch == 0) epoch = epochsLength; // creating a new epoch
if (epoch == epochsLength) {
uint256 epochEnd = ri.epochs[epochsLength-1].end.add(defaultEpochLength);
if(epochEnd < block.timestamp) epochEnd = block.timestamp; //This generally should not happen, but just in case - we generate only one epoch since previous end
ri.epochs.push(Epoch({
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
epochStart = 0;
}else {
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
RewardInfo storage ri = r.rewardInfo[token];
require(ri.epochs.length > 0, "RewardVesting: protocol or token not registered");
return ri.epochs.length-1;
}
```

UNKNOWN Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

SWC-101

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
require(epochsLength > 0, "RewardVesting: protocol or token not registered");

Epoch storage lastEpoch = ri.epochs[epochsLength-1];
uint256 previousClaim = ri.lastClaim;
if(previousClaim == lastEpoch.end) return; // Nothing to claim yet
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
120 uint256 i;
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
132 if(ep.end > block.timestamp) {
133 //Half-claim
134 uint256 epStart = ri.epochs[i-1].end;
135 uint256 claimStart = (previousClaim > epStart)?previousClaim:epStart;
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
145 if(previousClaim > epStart) {
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
162 uint256 epochsLength = ri.epochs.length;
163 require(epochsLength > 0, "RewardVesting: protocol or token not registered");
164 uint256 prevEpochEnd = ri.epochs[epochsLength-1].end;
165 require(epochEnd > prevEpochEnd, "RewardVesting: new epoch should end after previous");
166 ri.epochs.push(Epoch({
```

UNKNOWN

SWC-101

Compiler-rewritable "<uint> - 1" discovered

This plugin produces issues to support false positive discovery within MythX.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
200 if(epoch == 0) epoch = epochsLength; // creating a new epoch
201 if (epoch == epochsLength) {
202 uint256 epochEnd = ri.epochs[epochsLength-1].end.add(defaultEpochLength);
203 if(epochEnd < block.timestamp) epochEnd = block.timestamp; //This generally should not happen, but just in case - we generate only one epoch since previous end
204 ri.epochs.push(Epoch({
```

LOW

SWC-103

A floating pragma is set.

The current pragma Solidity directive is ""^0.5.12"". It is recommended to specify a fixed compiler version to ensure that the bytecode produced does not vary between builds. This is especially important if you rely on bytecode-level verification of the code.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
1 pragma solidity ^0.5.12;
2
3 import "@openzeppelin/contracts-ethereum-package/contracts/math/SafeMath.sol";
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
epochStart = 0;
}else {
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
epochStart = ri.epochs[epoch-1].end;
}
epochEnd = ri.epochs[epoch].end;
rewardAmount = ri.epochs[epoch].amount;
return (epochStart, epochEnd, rewardAmount);
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
}
epochEnd = ri.epochs[epoch].end;
rewardAmount = ri.epochs[epoch].amount;
return (epochStart, epochEnd, rewardAmount);
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
if(r.tokens.length == 0) return; //This allows claims from protocols which are not yet registered without reverting
for(uint256 i=0; i < r.tokens.length; i++){
_claimRewards(protocol, r.tokens[i]);
}
}
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
require(epochsLength > 0, "RewardVesting: protocol or token not registered");

Epoch storage lastEpoch = ri.epochs[epochsLength-1];
uint256 previousClaim = ri.lastClaim;
if(previousClaim == lastEpoch.end) return; // Nothing to claim yet
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```

uint256 claimAmount;
Epoch storage ep = ri.epochs[0];
uint256 i;
// Searching for last claimable epoch
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
121 // Searching for last claimable epoch
122 for(i = epochsLength-1; i > 0; i--) {
123 ep = ri.epochs[i];
124 if(ep.end < block.timestamp) { // We've found last fully-finished epoch
125 if(i < epochsLength-1) { // We have already started current epoch
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
125 if(i < epochsLength-1) { // We have already started current epoch
126 i++; // Go back to currently-running epoch
127 ep = ri.epochs[i];
128 }
129 break;
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
132 if(ep.end > block.timestamp) {
133 //Half-claim
134 uint256 epStart = ri.epochs[i-1].end;
135 uint256 claimStart = (previousClaim > epStart)?previousClaim:epStart;
136 uint256 epochClaim = ep.amount.mul(block.timestamp.sub(claimStart)).div(ep.end.sub(epStart));
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
140 //Claim rest
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
141 for(i; i > 0; i--) {
142 ep = ri.epochs[i];
143 uint256 epStart = ri.epochs[i-1].end;
144 if(ep.end > previousClaim) {
145 if(previousClaim > epStart) {
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
162 uint256 epochsLength = ri.epochs.length;
163 require(epochsLength > 0, "RewardVesting: protocol or token not registered");
164 uint256 prevEpochEnd = ri.epochs[epochsLength-1].end;
165 require(epochEnd > prevEpochEnd, "RewardVesting: new epoch should end after previous");
166 ri.epochs.push(Epoch({
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
182 "RewardVesting: array lengths do not match");
183 for(uint256 i=0; i<protocols.length; i++) {
184 _addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
185 }
186 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
182 "RewardVesting: array lengths do not match");
183 for(uint256 i=0; i<protocols.length; i++) {
184 _addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
185 }
186 }
```

UNKNOWN

SWC-110

Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

Source file

`contracts/modules/reward/RewardVestingModule.sol`

Locations

```
182 "RewardVesting: array lengths do not match");
183 for(uint256 i=0; i<protocols.length; i++) {
184 _addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
185 }
186 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
182 "RewardVesting: array lengths do not match");
183 for(uint256 i=0; i<protocols.length; i++) {
184 _addReward(protocols[i], tokens[i], epochs[i], amounts[i]);
185 }
186 }
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
200 if(epoch == 0) epoch = epochsLength; // creating a new epoch
201 if (epoch == epochsLength) {
202 uint256 epochEnd = ri.epochs[epochsLength-1].end.add(defaultEpochLength);
203 if(epochEnd < block.timestamp) epochEnd = block.timestamp; //This generally should not happen, but just in case - we generate only one epoch since previous end
204 ri.epochs.push(Epoch({
```

UNKNOWN Out of bounds array access

The index access expression can cause an exception in case of use of invalid array index value.

SWC-110

Source file

contracts/modules/reward/RewardVestingModule.sol

Locations

```
208 } else {
209 require(epochsLength > epoch, "RewardVesting: epoch is too high");
210 Epoch storage ep = ri.epochs[epoch];
211 require(ep.end > block.timestamp, "RewardVesting: epoch already finished");
212 ep.amount = ep.amount.add(amount);
```