ABDK CONSULTING

SMART CONTRACT AUDIT

Relevant

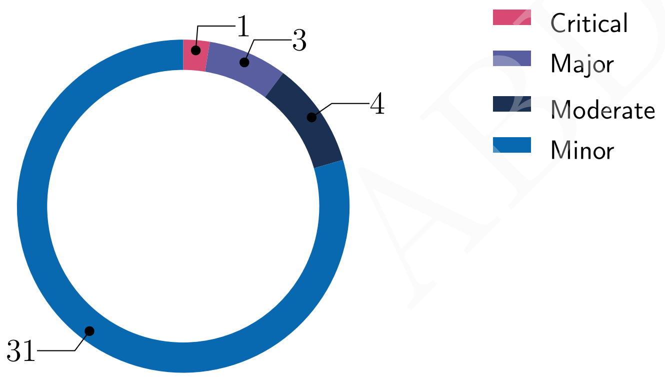
Solidity

abdk.consulting

SMART CONTRACT AUDIT CONCLUSION

by Mikhail Vladimirov and Dmitry Khovratovich
24th November 2021

We've been asked to review the 3 files in a Github repository. We found 1 critical, 3 major, and a few less important issues. All critical and major issues were fixed.



Findings

ID	Severity	Category	Status
CVF-1	Minor	Unclear behavior	Info
CVF-2	Minor	Unclear behavior	Info
CVF-3	Minor	Bad naming	Info
CVF-4	Minor	Documentation	Fixed
CVF-5	Minor	Procedural	Fixed
CVF-6	Minor	Bad datatype	Fixed
CVF-7	Minor	Procedural	Fixed
CVF-8	Major	Procedural	Fixed
CVF-9	Minor	Suboptimal	Fixed
CVF-10	Minor	Documentation	Fixed
CVF-11	Minor	Suboptimal	Fixed
CVF-12	Minor	Suboptimal	Fixed
CVF-13	Minor	Suboptimal	Fixed
CVF-14	Moderate	Suboptimal	Fixed
CVF-15	Minor	Procedural	Fixed
CVF-16	Major	Bad naming	Fixed
CVF-17	Minor	Procedural	Info
CVF-18	Minor	Procedural	Fixed
CVF-19	Minor	Procedural	Fixed
CVF-20	Minor	Suboptimal	Fixed
CVF-21	Minor	Suboptimal	Fixed
CVF-22	Minor	Procedural	Fixed
CVF-23	Minor	Suboptimal	Fixed
CVF-24	Minor	Procedural	Fixed
CVF-25	Minor	Bad datatype	Fixed
CVF-26	Minor	Bad datatype	Fixed
CVF-27	Minor	Procedural	Fixed

ID	Severity	Category	Status
CVF-28	Major	Procedural	Fixed
CVF-29	Critical	Flaw	Fixed
CVF-30	Minor	Suboptimal	Fixed
CVF-31	Moderate	Procedural	Fixed
CVF-32	Moderate	Suboptimal	Fixed
CVF-33	Minor	Suboptimal	Fixed
CVF-34	Minor	Suboptimal	Fixed
CVF-35	Minor	Suboptimal	Fixed
CVF-36	Minor	Suboptimal	Fixed
CVF-37	Minor	Unclear behavior	Info
CVF-38	Moderate	Flaw	Fixed
CVF-39	Minor	Procedural	Fixed

Contents

1 Document properties 7

2 Introduction 8
2.1 About ABDK 8
2.2 Disclaimer 8
2.3 Methodology 8

3 Detailed Results 10
3.1 CVF-1 10
3.2 CVF-2 10
3.3 CVF-3 10
3.4 CVF-4 11
3.5 CVF-5 11
3.6 CVF-6 11
3.7 CVF-7 12
3.8 CVF-8 13
3.9 CVF-9 13
3.10 CVF-10 14
3.11 CVF-11 14
3.12 CVF-12 14
3.13 CVF-13 15
3.14 CVF-14 15
3.15 CVF-15 15
3.16 CVF-16 16
3.17 CVF-17 16
3.18 CVF-18 17
3.19 CVF-19 17
3.20 CVF-20 17
3.21 CVF-21 18
3.22 CVF-22 18
3.23 CVF-23 18
3.24 CVF-24 19
3.25 CVF-25 19
3.26 CVF-26 19
3.27 CVF-27 20
3.28 CVF-28 20
3.29 CVF-29 21
3.30 CVF-30 21
3.31 CVF-31 22
3.32 CVF-32 22
3.33 CVF-33 22
3.34 CVF-34 23
3.35 CVF-35 23
3.36 CVF-36 24
3.37 CVF-37 24

3.38 CVF-38 24
3.39 CVF-39 25

1 Document properties

Version

Version	Date	Author	Description
0.1	November 23, 2021	D. Khovratovich	Initial Draft
0.2	November 23, 2021	D. Khovratovich	Minor revision
1.0	November 24, 2021	D. Khovratovich	Release

Contact

D. Khovratovich

khovratovich@gmail.com

2 Introduction

The following document provides the result of the audit performed by ABDK Consulting at the customer request. The audit goal is a general review of the smart contracts structure, critical/major bugs detection and issuing the general recommendations. We have reviewed the contracts at repository:

- RelevantTokenV3.sol
- sRel.sol
- Utils.sol

The fixes were provided in commit c175a52.

2.1 About ABDK

ABDK Consulting, established in 2016, is a leading service provider in the space of blockchain development and audit. It has contributed to numerous blockchain projects, and co-authored some widely known blockchain primitives like Poseidon hash function. The ABDK Audit Team, led by Mikhail Vladimirov and Dmitry Khovratovich, has conducted over 40 audits of blockchain projects in Solidity, Rust, Circom, C++, JavaScript, and other languages.

2.2 Disclaimer

Note that the performed audit represents current best practices and smart contract standards which are relevant at the date of publication. After fixing the indicated issues the smart contracts should be re-audited.

2.3 Methodology

The methodology is not a strict formal procedure, but rather a collection of methods and tactics that combined differently and tuned for every particular project, depending on the project structure and and used technologies, as well as on what the client is expecting from the audit. In current audit we use:

- **General Code Assessment**. The code is reviewed for clarity, consistency, style, and for whether it follows code best practices applicable to the particular programming language used. We check indentation, naming convention, commented code blocks, code duplication, confusing names, confusing, irrelevant, or missing comments etc. At this phase we also understand overall code structure.
- **Entity Usage Analysis**. Usages of various entities defined in the code are analysed. This includes both: internal usages from other parts of the code as well as potential external usages. We check that entities are defined in proper places and that their visibility scopes and access levels are relevant. At this phase we understand overall system architecture and how different parts of the code are related to each other.

- **Access Control Analysis**. For those entities, that could be accessed externally, access control measures are analysed. We check that access control is relevant and is done properly. At this phase we understand user roles and permissions, as well as what assets the system ought to protect.
- **Code Logic Analysis**. The code logic of particular functions is analysed for correctness and efficiency. We check that code actually does what it is supposed to do, that algorithms are optimal and correct, and that proper data types are used. We also check that external libraries used in the code are up to date and relevant to the tasks they solve in the code. At this phase we also understand data structures used and the purposes they are used for.

3 Detailed Results

3.1 CVF-1

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** RelevantTokenV3.sol

Description The compiler version requirement is very different from the other contracts. Is it intentional?
Client Comment This is intentional - we're keeping the same version as the original contract that is being upgraded. The contract is using OpenZeppelin libraries that cannot be updated and rely on the older version.

Listing 1:

```
pragma solidity ^0.5.2;
```

3.2 CVF-2

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** RelevantTokenV3.sol

Description This contract looks like an update for some other contract, but without the code of the previous version it is hard to tell whether the update is correct.
Client Comment We have exetensive tests to verify the update is correct.

Listing 2:

```
contract RelevantTokenV3 is Initializable, ERC20, Ownable {
```

3.3 CVF-3

- **Severity** Minor
- **Category** Bad naming
- **Status** Info
- **Source** RelevantTokenV3.sol

Recommendation Events are usually named via nouns, such as "Release" and "Claim".
Client Comment Leaving these as is to keep event names constistent with the previous version.

Listing 3:

```
event Released(uint256 amount, uint256 hoursSinceLast);
event Claimed(address indexed account, uint256 amount);
```

3.4 CVF-4

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Description The comment looks irrelevant and confusing.
Recommendation Consider removing it.

Listing 4:

```
uint256[101] private ______gap; // ERC20Minter gaps
```

3.5 CVF-5

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Description There is no access level specified for this mapping, so internal access will be used by default.
Recommendation Consider explicitly specifying an access level.
Client Comment Setting it to private

Listing 5:

```
mapping(address => uint256) nonces;
```

3.6 CVF-6

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Recommendation This should be a named constant
Client Comment This is beeing passed in as an intialization parameter

Listing 6:

```
inflation = 500;  // default init to 5% (can be adjusted later)
```

3.7 CVF-7

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Recommendation These functions should emit some events.

Listing 7:

```
function setInflation(uint256 _inflation) public onlyOwner {

function setAdmin(address _admin) public onlyOwner {

function updateAllocatedRewards(uint256 newAllocatedRewards) public onlyOwner {
```

3.8 CVF-8

- **Severity** Major
- **Status** Fixed
- **Category** Procedural
- **Source** RelevantTokenV3.sol

Description These functions don't ensure that the "initializedV3" flag is true.
Recommendation Consider adding such checks.
Client Comment Partial fix - adding the v3 intialization check to:
47 function setInflation(uint256 _inflation) public onlyOwner { 51 function setAdmin(address _admin) public onlyOwner { 99 function claimTokens(uint256 amount, bytes memory signature) public {
The following methods don't require initialization to function correctly, so we are conscously omitting the v3 intialization check:
56 function burn(uint256 amount) public onlyOwner { 61 function updateAllocatedRewards(uint256 newAllocatedRewards) public onlyOwner { 69 function vestAllocatedTokens(address vestingContract, uint256 amount) public onlyOwner {

Listing 8:

```
47 function setInflation(uint256 _inflation) public onlyOwner {

51 function setAdmin(address _admin) public onlyOwner {

56 function burn(uint256 amount) public onlyOwner {

61 function updateAllocatedRewards(uint256 newAllocatedRewards)
     ↪ public onlyOwner {

69 function vestAllocatedTokens(address vestingContract, uint256
     ↪ amount) public onlyOwner {

99 function claimTokens(uint256 amount, bytes memory signature)
     ↪ public {
```

3.9 CVF-9

- **Severity** Minor
- **Status** Fixed
- **Category** Suboptimal
- **Source** RelevantTokenV3.sol

Recommendation This could be simplified as: _burn (address (this), amount); require (balanceOf (address (this)) >= allocatedRewards);

Listing 9:

```
require(balanceOf(address(this)).sub(allocatedRewards) >= amount
  ↪ , "Relevant: cannot burn allocated tokens");
_burn(address(this), amount);
```

3.10 CVF-10

- **Severity** Minor
- **Category** Documentation
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Description The error message is inaccurate, as the owner may explicitly set the inflation rate to zero.
Recommendation Consider either fixing the error message, or adding a check into the "set-Inflation" function to forbid setting the inflation to zero.

Listing 10:

```
78 require(inflation > 0, "Relevant: inflation rate has not been
    ↪ set");
```

3.11 CVF-11

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Recommendation It would be more precise to use 1 second instead of 1 hour here.

Listing 11:

```
79 uint256 hoursSinceLast = (block.timestamp.sub(lastReward)).div(1
    ↪ hours);
80 require(hoursSinceLast/24 > 0, "Relevant: less than one day from
    ↪ last reward");

82 uint256 rewardAmount = totalSupply().mul(hoursSinceLast).mul(
    ↪ inflation).div(10000).div(365).div(24);
```

3.12 CVF-12

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Description The denominator 10000 * 365 * 24 could be precalculated.
Recommendation Using three divisions instead of one is waste of gas.

Listing 12:

```
82 uint256 rewardAmount = totalSupply().mul(hoursSinceLast).mul(
    ↪ inflation).div(10000).div(365).div(24);
```

3.13 CVF-13

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Recommendation Using a per-second inflation rate instead of annual inflation rate would allow simplifying this code, however, the inflation rate precision for per-second rate should be higher.
Client Comment This is now better with secondsSincleLastReward and INFLATION_DENOM constant

Listing 13:

```
uint256 rewardAmount = totalSupply().mul(hoursSinceLast).mul(inflation).div(10000).div(365).div(24);
```

3.14 CVF-14

- **Severity** Moderate
- **Category** Suboptimal
- **Status** Fixed
- **Source** RelevantTokenV3.sol

Description The hashed message doesn't include the contract name nor chain ID. This makes replay attacks possible in case of several contract instances.
Recommendation Consider including contract address and chain ID into the hashed message. Or even better, consider following the EIP-712 standard for signed message structure.

Listing 14:

```
bytes32 hash = keccak256(abi.encodePacked(amount, msg.sender, nonces[msg.sender]));
hash = keccak256(abi.encodePacked("\x19Ethereum Signed Message:\n32", hash));
```

3.15 CVF-15

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Utils.sol

Recommendation Should be "0̂.8.0" according to a common best practice unless there is something special about this particular version.

Listing 15:

```
pragma solidity ^0.8.2;
```

3.16 CVF-16

- **Severity** Major
- **Category** Bad naming
- **Status** Fixed
- **Source** Utils.sol

Description There is a terminology problem here. In traditional finance, "vested" securities means those securities that are already could be spent, while "unvested" are those that are still under lock. However here, the term "vested" is used for still locked tokens.
Recommendation Consider using the right terms.

Listing 16:

```
// note: we should be able to unlock all tokens (including vested tokens)

function transferVestedTokens(Vest storage self, Vest storage vestTo) internal {

function setVestedAmount(Vest storage self, uint shortAmnt, uint longAmnt) public {

function vested(Vest storage self) internal view returns (uint) {
```

3.17 CVF-17

- **Severity** Minor
- **Category** Procedural
- **Status** Info
- **Source** Utils.sol

Description This overwrites the fields of the passed structure.
Recommendation Consider checking that the structure fields are zero before overwriting them.
Client Comment This is intentional - we should be able to update the unlock amount/time if needed and will have a warning about overwriteing them the UI

Listing 17:

```
self.unlockAmnt = amount;
self.unlockTime = block.timestamp + lockTime;
```

3.18 CVF-18

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Utils.sol

Recommendation It would be more efficient to check that: 'self.shortAmount | self.longAmount == 0' as in this case Solidity will not perform overflow checks.

Listing 18:

```
require(self.shortAmnt + self.longAmnt == 0, "sRel Utils: this account already has vested tokens");
```

3.19 CVF-19

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Utils.sol

Recommendation It would be convenient to just return 0 instead of reverting.

Listing 19:

```
require(block.timestamp > vestBegin, "sRel Utils: Vesting has't started yet");

require(amount > 0, "sRel Utils: There are no vested tokens to claim");
```

3.20 CVF-20

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** Utils.sol

Recommendation This calculation should be performed only when block.timestamp < vestShort. In the opposite case, 'sAmnt' just equals 'shortAmnt'.

Listing 20:

```
uint sAmnt = shortAmnt * (currentTime - last) / (vestShort - last);
```

3.21 CVF-21

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** Utils.sol

Description Here, 'self.shortAmnt' is read from the storage once again.
Recommendation Consider using already read value like this: self.shortAmnt = shortAmnt - sAmnt;

Listing 21:

```
self.shortAmnt -= sAmnt;
```

3.22 CVF-22

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** Utils.sol

Recommendation This calculation should be performed only when block.timestamp < vestLong. In the opposite case, 'lAmnt'just equals 'longAmnt'.

Listing 22:

```
uint lAmnt = longAmnt * (currentTime - last) / (vestLong - last)
    ↪ ;
```

3.23 CVF-23

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** Utils.sol

Description Here, 'self.longAmnt' is read from the storage once again.
Recommendation Consider using already read value like this: self.longAmnt = longAmnt - lAmnt;

Listing 23:

```
self.longAmnt -= lAmnt;
```

3.24 CVF-24

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** sRel.sol

Recommendation Should be "0̂.8.0" according to a common best practice unless there is something special about this particular version.

Listing 24:

```
pragma solidity ^0.8.2;
```

3.25 CVF-25

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** sRel.sol

Recommendation The type of this variable should be "RelevantTokenV3" or "IERC20".

Listing 25:

```
address public immutable r3l; // RELEVANT TOKEN
```

3.26 CVF-26

- **Severity** Minor
- **Category** Bad datatype
- **Status** Fixed
- **Source** sRel.sol

Recommendation The type of the "_r3l" argument should be "RelevantTokenV3" or "IERC20".

Listing 26:

```
constructor(address _r3l, address _vestAdmin, uint _vestBegin,
    ↪ uint _vestShort, uint _vestLong)
```

3.27 CVF-27

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** sRel.sol

Description There are not range checks for the arguments.
Recommendation Consider adding appropriate checks. For example. consider adding a check that _vestBegin <= _vestShort <= _vestLong.

Listing 27:

```
constructor(address _r3l, address _vestAdmin, uint _vestBegin,
  ↪ uint _vestShort, uint _vestLong)
```

3.28 CVF-28

- **Severity** Major
- **Category** Procedural
- **Status** Fixed
- **Source** sRel.sol

Description There is a terminology problem here. In traditional finance, "vested" securities means those securities that are already could be spent, while "unvested" are those that are still under lock. However here, the term "vested" is used for still locked tokens and the term "unvested" is used for spendable token.
Recommendation Consider using the right terms.

Listing 28:

```
// only unlocked & unvested tokens can be transferred

    require(amount <= unvestedBalance, "sRel: you cannot
      ↪ transfer vested tokens");

// onwer can set amount of vested tokens manually

function setVestedAmount(address account, uint256 shortAmnt,
  ↪ uint256 longAmnt) onlyOwner external override(IsRel) {

function claimVestedRel() external override(IsRel) {

// transfer all vested tokens to a new address
function transferVestedTokens(address to) external override(
  ↪ IsRel) {

function vested(address account) external view override(IsRel)
  ↪ returns (uint) {
```

3.29 CVF-29

- **Severity** Critical
- **Category** Flaw
- **Status** Fixed
- **Source** sRel.sol

Recommendation It should be 'from' instead of 'msg.sender' here. Currently, a token holder may approve his tokens to his another address, and then transfer tokens using 'transferFrom' call to bypass vesting rules.
Client Comment I'm glad we got this contract audited!

Listing 29:

```
uint unvestedBalance = balanceOf(msg.sender) - vest[msg.sender].vested();
```

3.30 CVF-30

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** sRel.sol

Description The expression "unlocks[from]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 30:

```
    unlocks[from].useUnlocked(amount); // only unlocked tokens can be transferred
    emit lockUpdated(from, unlocks[from]);

unlocks[msg.sender].unlock(amount, lockPeriod);
emit lockUpdated(msg.sender, unlocks[msg.sender]);

unlocks[msg.sender].resetLock();
emit lockUpdated(msg.sender, unlocks[msg.sender]);
```

3.31 CVF-31

- **Severity** Moderate
- **Category** Procedural
- **Status** Fixed
- **Source** sRel.sol

Description The returned value is ignored here.
Recommendation Consider explicitly checking that the returned value is true.

Listing 31:

```
IERC20(r3l).transferFrom(msg.sender, address(this), amount);

IERC20(r3l).transfer(msg.sender, amount);

transfer(to, amount);
```

3.32 CVF-32

- **Severity** Moderate
- **Category** Suboptimal
- **Status** Fixed
- **Source** sRel.sol

Description The hashed message doesn't include the contract name nor chain ID. This makes replay attacks possible in case of several contract instances.
Recommendation Consider including contract address and chain ID into the hashed message. Or even better, consider following the EIP-712 standard for signed message structure.
Client Comment EIP-712 standard implemented.

Listing 32:

```
bytes32 hash = keccak256(abi.encodePacked(_shortAmount,
    ↪ _longAmount, msg.sender, vestNonce[msg.sender]));
```

3.33 CVF-33

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** sRel.sol

Description The "vestNonce[msg.sender]" value is read from the storage twice.
Recommendation Consider reading once and reusing.

Listing 33:

```
bytes32 hash = keccak256(abi.encodePacked(_shortAmount,
    ↪ _longAmount, msg.sender, vestNonce[msg.sender]));

vestNonce[msg.sender] += 1;
```

3.34 CVF-34

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** sRel.sol

Description The expression "vest[account]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 34:

```
vest[account].setVestedAmount(shortAmnt, longAmnt);

emit vestUpdated(account, msg.sender, vest[account]);
```

3.35 CVF-35

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** sRel.sol

Description The expression "shortAmount + longAmount" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 35:

```
require(totalSupply() + shortAmnt + longAmnt <= IERC20(r3l).
    ↪ balanceOf(address(this)), "sRel: Not enought REL in
    ↪ contract");
_mint(account, shortAmnt + longAmnt);
```

3.36 CVF-36

- **Severity** Minor
- **Category** Suboptimal
- **Status** Fixed
- **Source** sRel.sol

Description The expression "vest[msg.sender]" is calculated twice.
Recommendation Consider calculating once and reusing.

Listing 36:

```
uint amount = vest[msg.sender].updateVestedAmount(vestShort,
    ↪ vestLong, vestBegin);

emit vestUpdated(msg.sender, msg.sender, vest[msg.sender]);

uint amount = vest[msg.sender].vested();
vest[msg.sender].transferVestedTokens(vest[to]);

emit vestUpdated(msg.sender, msg.sender, vest[msg.sender]);
emit vestUpdated(to, msg.sender, vest[msg.sender]);
```

3.37 CVF-37

- **Severity** Minor
- **Category** Unclear behavior
- **Status** Info
- **Source** sRel.sol

Description This overwrites an existing unlock is any. If this fine?
Client Comment This is intentional - we will warn users about this via the UI.

Listing 37:

```
unlocks[msg.sender].unlock(amount, lockPeriod);
```

3.38 CVF-38

- **Severity** Moderate
- **Category** Flaw
- **Status** Fixed
- **Source** sRel.sol

Recommendation It should be 'vest[to]' (or its cached version) instead of 'vest[msg.sender]' here, as 'vest[msg.sender]' is already zeroed out.

Listing 38:

```
emit vestUpdated(to, msg.sender, vest[msg.sender]);
```

3.39 CVF-39

- **Severity** Minor
- **Category** Procedural
- **Status** Fixed
- **Source** sRel.sol

Description These events are logged even if nothing actually changed.

Listing 39:

```
emit lockPeriodUpdated(lockPeriod);

emit vestAdminUpdated(vestAdmin);
```