

Security Audit

of SMART VALOR's Smart Contracts

October 24, 2018



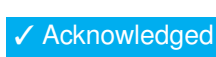









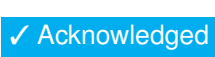
Produced for






by



Table Of Content

Foreword	1
Executive Summary	1
Audit Overview	2
1. Scope of the Audit	2
2. Depth of Audit	2
3. Terminology	2
Limitations	4
System Overview	5
1. Stake creation	5
2. Stake management	5
Best Practices in SMART VALOR's project	6
1. Hard Requirements	6
2. Soft Requirements	6
Security Issues	7
1. Possibility of creating a stake without paying tokens  	7
2. Potentially unsafe dependence on <code>block.timestamp</code>  	7
3. Users can create valueless stakes  	8
4. Locked tokens in <code>ValorStakeFactory</code>  	8
Trust Issues	9
1. Admin can invalidate membership  	9
Design Issues	10
1. Old compiler version  	10
2. No <code>Release</code> event present  	10

3.	No event emitted when factory gets terminated			10
4.	Zero day staking allowed			10
5.	createOnBehalf functionality missing			10
Recommendations / Suggestions				11
Disclaimer				13

Foreword

We first and foremost thank SMART VALOR for giving us the opportunity to audit their smart contracts. This document outlines our methodology, limitations, and results.

– ChainSecurity

Executive Summary

CHAINSECURITY has analyzed the SMART VALOR contracts carefully under different aspects, with a variety of tools for automated security analysis of Ethereum smart contracts and manual review.

Overall, we found that SMART VALOR employs good coding practices and has clean, well-maintained code. No critical security vulnerabilities were found during the audit, but nonetheless several issues and questions regarding the system design were raised. SMART VALOR reevaluated the interaction between smart contracts and off-chain part of the platform, which resulted in an improved and more secure system design.

Audit Overview

Scope of the Audit

The scope of the audit is limited to the following source code files. All of these source code files were received on October 11, 2018 and an updated version on October 24, 2018:

File	SHA-256 checksum
ValorStakeFactory.sol	ab11bb484235482c80eecaafa3606376e43e216c8f537c37b0e2cffbf59f2af23
ValorTimelock.sol	0cdc6b4df0bf07be56d56001719cf7d95d13e0fabe9e0a8954aeb2f2f8c2372b

Depth of Audit

The scope of the security audit conducted by CHAINSECURITY was restricted to:

- Scan the contracts listed above for generic security issues using automated systems and manually inspect the results.
- Manual audit of the contracts listed above for security issues.

Terminology





For the purpose of this audit, we adopt the following terminology. For security vulnerabilities, we specify the *likelihood*, *impact* and *severity* (inspired by the OWASP risk rating methodology¹).

Likelihood represents the likelihood of a security vulnerability to be encountered or exploited in the wild.










Impact specifies the technical and business related consequences of an exploit.

Severity is derived based on the likelihood and the impact calculated previously.

We categorize the findings into 4 distinct categories, depending on their severities:

-  Low: can be considered as less important
-  Medium: should be fixed
-  High: we strongly suggest to fix it before release
-  Critical: needs to be fixed before release

These severities are derived from the likelihood and the impact using the following table, following a standard approach in risk assessment.

LIKELIHOOD	IMPACT		
	High	Medium	Low
High			
Medium			
Low			

¹https://www.owasp.org/index.php/OWASP_Risk_Rating_Methodology

During the audit concerns might arise or tools might flag certain security issues. If our careful inspection reveals no security impact, we label it as **✓ No Issue**. If during the course of the audit process, an issue has been addressed technically, we label it as **✓ Fixed**, while if it has been addressed otherwise by improving documentation or further specification, we label it as **✓ Addressed**. Finally, if an issue is meant to be fixed in the future without immediate changes to the code, we label it as **✓ Acknowledged**.

Findings that are labelled as either **✓ Fixed** or **✓ Addressed** are resolved and therefore pose no security threat. Their severity is still listed, but just to give the reader a quick overview what kind of issues were found during the audit.

Limitations

Security auditing cannot uncover all existing vulnerabilities, and even an audit in which no vulnerabilities are found is not a guarantee for a secure smart contract. However, auditing allows to discover vulnerabilities that were overlooked during development and areas where additional security measures are necessary.

In most cases, applications are either fully protected against a certain type of attack, or they lack protection against it completely. Some of the issues may affect the entire smart contract application, while some lack protection only in certain areas. We therefore carry out a source code review trying to determine all locations that need to be fixed. Within the customer-determined timeframe, CHAINSECURITY has performed auditing in order to discover as many vulnerabilities as possible.

System Overview

The SMART VALOR system is a staking platform that allows users to stake a number of tokens in exchange for a certain type of membership. The functionality is decoupled into two smart contracts, each responsible for stake creation and stake management logic respectively.

Stake creation

Stakes are created using a `ValorStakeFactory` contract. At any point there is exactly one active stake factory deployed by SMART VALOR. Membership claims are accepted only if their associated stake is created by the legitimate factory.

Stakes are created as first class citizen contracts, meaning that the `ValorStakeFactory` deploys a new contract every time a stake is created. Each stake has an associated lock period, which prevents a user from releasing tokens too early. The lock period is not longer than 365 days.

It is planned that users will be able to create stakes through the use of a dedicated DApp even if they are not yet on-boarded. On-boarded users can access their user profile. Each user is allowed to have many stakes since every stake can grant different privileges or membership rights.

The active `ValorStakeFactory` contract always has an owner. The owner can:

- Pause the creation of new time lock staking contracts.
- Resume the creation of new time lock staking contracts, if previously paused.
- Destroy the currently active `ValorTokenFactory` contract in case a new upgraded version of the factory contract is deployed.

Stake management

Each stake is managed separately from others, through the use of its dedicated `ValorTimelock` contract. The contract implements the logic for releasing a stake.

A full or a partial release is only possible after the lock period associated with the stake has expired. Users are restricted to invoking a release only on their own stakes. When a release is executed, tokens are transferred back to the user who created the stake.

In case of emergencies such as a security breach, the owner of the stake contract is allowed to perform an emergency release. During an emergency all tokens are transferred back to the users, regardless of whether the lock period has passed or not.

Best Practices in SMART VALOR's project

Projects of good quality follow best practices. In doing so, they make audits more meaningful, by allowing efforts to be focused on subtle and project-specific issues rather than the fulfillment of general guidelines.

Avoiding code duplication is a good example of a good engineering practice which increases the potential of any security audit.

We now list a few points that should be enforced in any good project that aims to be deployed on the Ethereum blockchain. The corresponding box is ticked when SMART VALOR's project fitted the criterion when the audit started.

Hard Requirements

These requirements ensure that the SMART VALOR's project can be audited by CHAINSECURITY.

- ☒ The code is provided as a Git repository to allow the review of future code changes.
- ☒ Code duplication is minimal, or justified and documented.
- ☒ Libraries are properly referred to as package dependencies, including the specific version(s) that are compatible with SMART VALOR's project. No library file is mixed with SMART VALOR's own files.
- ☒ The code compiles with the latest Solidity compiler version. If SMART VALOR uses an older version, the reasons are documented.
- ☒ There are no compiler warnings, or warnings are documented.

Soft Requirements

Although these requirements are not as important as the previous ones, they still help to make the audit more valuable to SMART VALOR.

- ☒ There are migration scripts.
- ☒ There are tests.
- ☒ The tests are related to the migration scripts and a clear separation is made between the two.
- ☒ The tests are easy to run for CHAINSECURITY, using the documentation provided by SMART VALOR.
- ☒ The test coverage is available or can be obtained easily.
- ☒ The output of the build process (including possible flattened files) is not committed to the Git repository.
- ☒ The project only contains audit-related files, or, if not possible, a meaningful separation is made between modules that have to be audited and modules that CHAINSECURITY should assume correct and out of scope.
- ☒ There is no dead code.
- ☒ The code is well documented.
- ☒ The high-level specification is thorough and allow a quick understanding of the project without looking at the code.
- ☒ Both the code documentation and the high-level specification are up to date with respect to the code version CHAINSECURITY audits.
- ☒ There are no getter functions for public variables, or the reason why these getters are in the code is given.
- ☒ Function are grouped together according either to the Solidity guidelines², or to their functionality.

²<https://solidity.readthedocs.io/en/latest/style-guide.html#order-of-functions>

Security Issues

In the following, we discuss our investigation into security issues. Therefore, we highlight whenever we found specific issues but also mention what vulnerability classes do not appear, if relevant.

Possibility of creating a stake without paying tokens

A successful stake creation requires that tokens get transferred from the user's wallet to the stake contract deployed for this stake. This is done through the use of the token's `transferFrom` function which returns a boolean indicating the success of the transfer.

However, the `createStake` function blindly creates the stake without verifying the success of the token transfer. As a result, if something goes wrong during the transfer `ValorStakeFactory` will create a stake without having deducted any tokens from the user's wallet.

```
function createStake(uint256 lockPeriod, uint256 atStake)
public whenNotPaused {
    require(lockPeriod <= 365 * 86400); //being 1 day = 86400s
    address beneficiary = msg.sender;
    ValorTimelock stake = new ValorTimelock(token, beneficiary, owner,
        lockPeriod);
    token.transferFrom(msg.sender, address(stake), atStake);
    emit StakeCreated(address(stake), msg.sender, lockPeriod, atStake);
}
```

If `ValorStakeFactory` uses the `ValorToken` as its staking token, as is intended, this is unlikely to be problematic as its `transferFrom()` implementation uses `require` statements and `SafeMath` operations that will propagate any errors thrown during transfer.

However, should any other token be used, an attacker can exploit this and gain membership without paying any tokens.

Therefore, CHAINSECURITY strongly recommends checking the output of all external calls. The same applies for the calls to `token.transfer()` in `ValorTimelock`. One solution to this is enclosing the external call in a `require` statement.

```
require(token.transferFrom(msg.sender, address(stake), atStake));
```

Likelihood: Low

Impact: High

Fixed: SMART VALOR successfully fixed the issue by verifying the return values of all external calls.

Potentially unsafe dependence on `block.timestamp`

Function `partialRelease` uses `block.timestamp` to verify that a certain time has passed before sending tokens to a beneficiary. Although block manipulation is considered hard to perform, a malicious miner is able to move forward block timestamps by up to 900 seconds (15min) compared to the actual time. As a result, tokens for a beneficiary could be released earlier than they should be.

CHAINSECURITY notes that SMART VALOR and its users should be aware of this and adhere to the 15 seconds rule³.

Likelihood: Low

Impact: Low

Acknowledged: SMART VALOR acknowledges the issue and states that a drift of up to 15 minutes is considered acceptable against time locks of 12 months.

³<https://consensys.github.io/smart-contract-best-practices/recommendations/#the-15-second-rule>

Users can create valueless stakes

Anyone is able to call `createStake()` with `atStake = 0`. Therefore the number of stakes a malicious user can create is restricted only by the gas cost of executing the transaction. Depending on how stake tracking is handled off-chain, this may have some effect on the SMART VALOR backend.

CHAINSECURITY recommends to have input validation on the value of `atStake` to ensure that the stake is meaningful.

Likelihood: Low

Impact: Low

Fixed: SMART VALOR solved the issue by adding a minimum stake variable, which is checked every time a stake is created. The minimum stake value can be updated only by the owner.

Locked tokens in `ValorStakeFactory`

Users who are not yet-on-boarded on the SMART VALOR platform can still create stakes by directly executing transactions from their wallet. These users are more prone to making errors such as doing a token transfer instead of approving a withdrawal, which is a prerequisite for a call to `createStake`.

In such cases tokens would be permanently locked in the `ValorStakeFactory` contract since there is no functionality to handle them. Numerous historic cases have shown that accidental ERC20 transfers to token contracts occur frequently.

Likelihood: Low

Impact: Medium

Fixed: SMART VALOR fixed the issue by implementing a `withdraw` function, which transfers all tokens in the `ValorStakeFactory` contract to the admin wallet. Additional guidance and a DApp will be provided to limit the risk for untrained users.

Trust Issues

The issues described in this section are not security issues but describe functionality which is not fixed inside the smart contract and hence requires additional trust into SMART VALOR, including in SMART VALOR's ability to deal with such powers appropriately.

Admin can invalidate membership

The owner of a `ValorTimeLock` contract has the power to call `emergencyRelease` at anytime. As a result tokens are returned to the user who created the stake and their membership is immediately invalidated.

Although this functionality has a good use case such as being used after a security breach, it can still be executed without the consent of the user. This means that SMART VALOR can invalidate any user's membership at any point in time.

Addressed: SMART VALOR addresses the issue by managing membership revocation off-chain. Depending on the cause of emergency release, the membership can be maintained while the issue is resolved.

Design Issues

The points listed here are general recommendations about the design and style of SMART VALOR's project. They highlight possible ways for SMART VALOR to further improve the code.

Old compiler version

Without a documented reason, the newest compiler version should be used homogeneously. In particular, given the vulnerabilities of version 0.4.24⁴, SMART VALOR should enforce the new version 0.4.25 through the use of a corresponding `pragma`.

Fixed: SMART VALOR successfully updated the pragma to use the latest compiler version, which is 0.4.25.

No Release event present

ValorTimelock contract declares and emits an EmergencyRelease event, but no event is emitted when full or partial releases are made. Token releases comprise the primary functionality of the ValorTimelock contract and as such should have an appropriate event associated with them.

Acknowledged: SMART VALOR states that adding code for more event emissions will cause a bigger bytecode. For any relevant use cases the Transfer event can be monitored instead.

No event emitted when factory gets terminated

The specifications states that a FactoryTerminated event gets emitted when a factory is dismissed by SMART VALOR.

However, the ValorStakeFactory contract neither defines, nor emits such an event.

Fixed: SMART VALOR added a FactoryDismiss event that gets fired whenever a factory is dismissed.

Zero day staking allowed

Anyone is allowed to call the createStake function, with a value of 0 for the lockPeriod argument. Therefore a user can accidentally create meaningless stake contracts. This is not fatal as the user is able to immediately release their tokens, but can be an inconvenience.

SMART VALOR should consider having a minimum lockPeriod limit check:

```
require(lockPeriod > 0);
```

Fixed: SMART VALOR added a variable which stores the minimum staking period and is used for verification every time a stake gets created. The minimum staking period can be updated only by the owner.

createOnBehalf functionality missing

The requirements mention that:

```
Nobody, except SV with 'createOnBehalf', can create a stake with a beneficiary different from himself.
```

This implies that SMART VALOR is able to create stakes on behalf of a user. However, this functionality is missing in the current version of the codebase.

Acknowledged: SMART VALOR acknowledges that the specifications were not up to date.

⁴<https://etherscan.io/solcbuginfo?a=ExpExponentCleanup>

Recommendations / Suggestions

- ✓ Functions `createStake`, `dismiss`, `release` and `emergencyRelease` can have their visibility changed from `public` to `external` since they are not called from within the contracts.
- ✗ Once the owner is assigned in a `ValorTimeLock` contract there is no possible way of passing ownership to another address. SMART VALOR can consider if this might be desired.
- ✓ Function arguments are not named in a consistent manner. The convention in Solidity is to start argument names with an underscore.

```
constructor(address _tokenAddress, address _companyWallet)
```

```
constructor(ERC20 _token, address _beneficiary, address _admin, uint256  
_duration)
```

- ✓ The `createStake` function initializes a variable `beneficiary = msg.sender`. However, it is not used consistently since the rest of the function statements keep switching between `beneficiary` and `msg.sender` to refer to the same entity.
CHAINSECURITY recommends either removing the `beneficiary` variable completely or using it consistently for the rest of the function.
- ✓ The specification states that the `ValorStakeFactory` and `ValorTimeLock` contracts cannot accept Ether. Due to the lack of a `payable` function this indeed holds for regular transfers which use `send()` or `transfer()`.
However, there are ways of forcing a contract to accept Ether⁵. In the case of SMART VALOR a malicious user cannot gain any advantage from doing this, because both `ValorStakeFactory` and `ValorTimeLock` do not execute calculations based on the contract's balance. Nonetheless, CHAINSECURITY would like to raise awareness that this requirement may be violated.
- ✓ The `StakeCreated` event does not have any indexed parameters. In case emitted events need to be quickly searchable, for example within a DApp, SMART VALOR should consider marking key parameters with the `indexed` keyword.
- ✓ CHAINSECURITY suggests using `MultiSig` wallets to control the owner of `ValorTimeLock` contracts. The owner address is set in the constructor and once the contract is deployed, there is no way to change ownership.
- ✓ The statement `require(lockPeriod = 365 * 86400)` can be improved by using `TimeUnits` provided by Solidity:

```
require(lockPeriod <= 365 days)
```

- ✓ `ValorStakeFactory.sol` has mixed tabs and spaces on line 36. CHAINSECURITY recommends following the Solidity style guide, which states that spaces are the preferred indentation method⁶.
- ✓ The solidity guidelines recommend formatting `event` definitions in the following way⁷:

```
event StakeCreated(  
    address stake,  
    address beneficiary,  
    uint256 lockPeriod,  
    uint256 atStake  
);
```

⁵https://consensys.github.io/smart-contract-best-practices/known_attacks/#forcibly-sending-ether-to-a-contract

⁶<https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#tabs-or-spaces>

⁷<https://solidity.readthedocs.io/en/v0.4.25/style-guide.html#maximum-line-length>

Post-audit comment: SMART VALOR has fixed most of the issues above and is aware of all the implications of those points which were not addressed. Given this awareness, SMART VALOR has to perform no more code changes with regards to these recommendations.

Disclaimer

UPON REQUEST BY SMART VALOR, CHAINSECURITY LTD. AGREES MAKING THIS AUDIT REPORT PUBLIC. THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND CHAINSECURITY LTD. DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH CHAINSECURITY LTD..