SWISS TOKEN AUDIT

November 2020

BLOCKCHAIN CONSILIUM

Contents

Disclaimer

The audit makes no statements or warranties about utility of the code, safety of the code, suitability of the business model, regulatory regime for the business model, or any other statements about fitness of the contracts to purpose, or their bug free status. The audit documentation is for discussion purposes only.

THE CONTENT OF THIS AUDIT REPORT IS PROVIDED "AS IS", WITHOUT REPRESENTATIONS AND WARRANTIES OF ANY KIND, AND BLOCKCHAIN CONSILIUM DISCLAIMS ANY LIABILITY FOR DAMAGE ARISING OUT OF, OR IN CONNECTION WITH, THIS AUDIT REPORT. COPYRIGHT OF THIS REPORT REMAINS WITH BLOCKCHAIN CONSILIUM.

Purpose of the report

The Audits and the analysis described therein are created solely for Clients and published with their consent. The scope of our review is limited to a review of Solidity code and only the Solidity code we note as being within the scope of our review within this report. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond the Solidity programming language that could present security risks. Cryptographic tokens and smart contracts are emergent technologies and carry with them high levels of technical risk and uncertainty.

The Audits are not an endorsement or indictment of any particular project or team, and the Audits do not guarantee the security of any particular project. This Report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. No Report provides any warranty or representation to any Third-Party in any respect, including regarding the bugfree nature of code, the business model or proprietors of any such business model, and the legal compliance of any such business. No third party should rely on the Audits in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. This Report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project. There is no owed duty to any Third-Party by virtue of publishing these Audits.

Introduction

We first thank Swiss Finance for giving us the opportunity to audit their smart contract. This document outlines our methodology, audit details, and results.

Swiss Finance requested us to review their Swiss Token smart contract (At GitHub commit hash: `259466d40b318924541af6dfe8a70f6c350f3cb9`). Blockchain Consilium reviewed the system from a technical perspective looking for bugs, issues and vulnerabilities in their code base. The Audit is valid for SWISS.sol at `259466d40b318924541af6dfe8a70f6c350f3cb9` GitHub commit hash only. The audit is not valid for any other versions of the smart contracts. Read more below.

Audit Summary

Overall, the code is well commented and clear on what it is supposed to do for each function. The visibility and state mutability of all the functions are clearly specified, and there are no confusions.

GitHub Link:
https://github.com/swissfarming-finance/solidity/blob/259466d40b318924541af6dfe8a70f6c350f3cb9/SWISS.sol

Audit Result	**PASSED**
High Severity Issues Found	None
Moderate Severity Issues Found	None
Low Severity Issues Found	None

Overview

The project has one Solidity file for SWISS Token Smart Contract, the SWISS.sol file that contains about 772 lines of Solidity code. We manually reviewed each line of code in the smart contract. All the functions and state variables are well commented using the NatSpec documentation for the functions which is good to understand quickly how everything is supposed to work.

Methodology:

Blockchain Consilium manually reviewed the smart contract line-by-line, keeping in mind industry best practices and known attacks, looking for any potential issues and vulnerabilities, and areas where improvements are possible.

We also used automated tools like slither for analysis and reviewing the smart contract. The raw output of these tools is included in the Appendix. These tools often give false-positives, and any issues reported by them but not included in the issue list can be considered not valid.

Classification / Issue Types Definition:

1. **High Severity:** which presents a significant security vulnerability or failure of the contract across a range of scenarios, or which may result in loss of funds.
2. **Moderate Severity:** which affects the desired outcome of the contract execution or introduces a weakness that can be exploited. It may not result in loss of funds but breaks the functionality or produces unexpected behaviour.
3. **Low Severity:** which does not have a material impact on the contract execution and is likely to be subjective.

The smart contract is considered to pass the audit, as of the audit date, if no high severity or moderate severity issues are found.

Attacks & Issues considered while auditing

In order to check for the security of the contract, we reviewed each line of code in the smart contract considering several known Smart Contract Attacks & known issues.

- ### Overflows and underflows:
  An overflow happens when the limit of the type variable uint256 , 2 ** 256, is exceeded. What happens is that the value resets to zero instead of incrementing more.

  For instance, if we want to assign a value to a uint bigger than 2 ** 256 it will simple go to 0—this is dangerous.

  On the other hand, an underflow happens when you try to subtract 0 minus a number bigger than 0.For example, if you subtract 0 - 1 the result will be = 2 ** 256 instead of -1.

This is quite dangerous. This contract **DOES** check for overflows and underflows by using **OpenZeppelin's** *SafeMath*.

- **Reentrancy Attack:**
  One of the major dangers of calling external contracts is that they can take over the control flow, and make changes to your data that the calling function wasn't expecting. This class of bug can take many forms, and both of the major bugs that led to the DAO's collapse were bugs of this sort.

  This smart contract does not make any state changes after external calls and thus *is not found vulnerable* to re-entrancy attack.

- **Replay attack:**
  The replay attack consists of making a transaction on one blockchain like the original Ethereum's blockchain and then repeating it on another blockchain like the Ethereum's classic blockchain. The ether is transferred like a normal transaction from a blockchain to another. Though it's no longer a problem because since the version 1.5.3 of *Geth* and 1.4.4 of *Parity* both implement the attack protection EIP 155 by Vitalik Buterin.
  So the people that will use the contract depend on their own ability to be updated with those programs to keep themselves secure.

- **Short address attack:**
  This attack affects ERC20 tokens, was discovered by the Golem team and consists of the following:

  A user creates an Ethereum wallet with a trailing 0, which is not hard because it's only a digit. For instance: `0xiofa8d97756as7df5sd8f75g8675ds8gsdg0`
  Then he buys tokens by removing the last zero:
  Buy 1000 tokens from account `0xiofa8d97756as7df5sd8f75g8675ds8gsdg`. If the contract has enough amount of tokens and the buy function doesn't check the length of the address of the sender, the Ethereum's virtual machine will just add zeroes to the transaction until the address is complete.

  The virtual machine will return 256000 for each 1000 tokens bought. This is abug of the virtual machine.

  Here is a **fix for short address attacks**

```
modifier onlyPayloadSize(uint size) {
    assert(msg.data.length >= size + 4);
    _;
}
function transfer(address _to, uint256 _value) onlyPayloadSize(2 * 32) {
    // do stuff
}
```

Whether or not it is appropriate for token contracts to mitigate the short-address attack is a contentious issue among smart-contract developers. Many, including those behind the OpenZeppelin project, have explicitly chosen not to do so. Blockchain Consilium doesn't consider short address attack an issue of the smart contract at the token level.

This contract **does not** implement an `onlyPayloadSize(uint numwords)` modifier for `transfer`, `transferFrom`, `approve`, `increaseApproval`,
and `decreaseApproval` functions, it probably assumes that checks for short address attacks are handled at a higher layer (which generally are), and since the `onlyPayloadSize()` modifier started causing some bugs restricting the flexibility of the smart contracts, it's alright not to check for short address attacks at the Token Contract level to allow for some more flexibility for dAPP coding, but the checks for short address attacks must be done at some layer of coding (e.g. for buys and sells, the exchange can do it - almost all well-known exchanges check for short address attacks after the Golem Team discovered it), this contract *does not prevent short address attack*, so the *checks for short address attack must be done while buying or selling or coding a DAPP using SWISS where necessary*.

You can read more about the attack here: ERC20 Short Address Attacks.

- **Approval Double-spend:**
  Imagine that Alice approves Bob to spend 100 tokens. Later, Alice decides to approve Bob to spend 150 tokens instead. If Bob is monitoring pending transactions, then when he sees Alice's new approval he can attempt to quickly spend 100 tokens, racing to get his transaction mined in before Alice's new approval arrives. If his transaction beats Alice's, then he can spend another 150 tokens after Alice's transaction goes through.

  This issue is a consequence of the ERC20 standard, which specifies that `approve()` takes a replacement value but no prior value. Preventing the attack while complying with ERC20 involves some compromise: users should set the approval to zero, make sure Bob hasn't snuck in a spend, then set the new value. In general, this sort of attack is possible with functions which do not encode enough prior state; in this case Alice's baseline belief of Bob's outstanding spent token balance from the Bob allowance.

  It's possible for `approve()` to enforce this behaviour without API changes in the ERC20 specification:
  `if ((_value != 0) && (approved[msg.sender][_spender] != 0)) return false;`

However, this is just an attempt to modify user behaviour. If the user does attempt to change from one non-zero value to another, then the double spend can still happen, since the attacker will set the value to zero.

If desired, a nonstandard function can be added to minimize hassle for users. The issue can be fixed with minimal inconvenience by taking a change value rather than a replacement value:

```
function increaseApproval (address _spender, uint256 _addedValue)
returns (bool success) {
  uint oldValue = approved[msg.sender][_spender];
  approved[msg.sender][_spender] = safeAdd(oldValue, _addedValue);
  return true;
}
```

Even if this function is added, it's important to keep the original for compatibility with the ERC20 specification.

Likely impact of this bug is low for most situations. The contract does apply mitigation to short address attacks.

For more, see this discussion on GitHub:
https://github.com/ethereum/EIPs/issues/20#issuecomment263524729

- ## Accidental Token Loss

One more issue is when other ERC20 Tokens are transferred to the Swiss Token smart contract, traditionally there would be no way to take them out, though this is resolved by adding an admin accessible function which can be used to transfer out ERC20 tokens from this smart contract.

Issues Found

High Severity Issues

No high severity issues were found in the smart contract.

Moderate Severity Issues

No moderate severity issues were found in the smart contract.

Low Severity Issues

No low severity issues were found in the smart contract.

Informational Observations

The smart contract includes owner accessible functions to set uniswap pair address and set the transfer fees while transfer is made to or from the uniswap pair. This would make it charge predefined transfer fee % on every transfer to and from uniswap pair, which is generally the case while adding liquidity, removing liquidity, and while buying and selling.

Line by line comments

SWISS.sol

- Line 5:
  The compiler version is specified as 0.6.11, this means the code can be compiled with solidity compilers with versions equal to 0.6.11. The latest compiler version at the time of auditing is 0.7.4.

- Lines 13 to 32:
  OpenZeppelin's Context helper function is included to provide context about transaction sender and data.

- Lines 38 to 110:
  Interface of the ERC20 standard as defined in the EIP is included with required ERC20 function signatures.

- Lines 116 to 270:
  OpenZeppelin's SafeMath library is used for safe arithmetic operations to check for overflows and underflows.

- Lines 279 to 576:
  Implementation of the IERC20 interface is included, inheriting from Context and IERC20. This implements standard ERC20 functions, along with helper internal functions for minting and burning, minting function is not externally accessible – it is used while contract deployment to mint initial supply, and burning function is used in the ERC20Burnable Contract. This contract also implements mitigation for approval doublespend attacks.

- Lines 584 to 616:
  ERC20Burnable contract is included, inheriting from Context and ERC20 contracts. This makes the token Burnable and implements a helper burnFrom function to allow users with appropriate allowance from a spender to burn the spender's tokens.

- Lines 618 to 657:
  Ownable contract is included for basic access management.

- Lines 660 to 662:
  tokenRecipient interface is included to implement an approveAndCall function which will be useful for smart contract integrations.

- Lines 664 to 666:
  OldIERC20 interface with transfer method signature is included, this helps

contract admin to transfer Legacy ERC20 tokens from this smart contract if someone accidentally transfers Legacy ERC20 Tokens to this smart contracts.

- Lines 668 to 772:
  SwissToken contract is implemented inheriting from ERC20Burnable and Ownable contracts. This contract overrides transfer and transferFrom functions to allow for the 2% initially declared swiss fee and 1% initially declared desh fee to be deducted from the transferred tokens and sent to the initially declared fees_wallet_swiss and fees_wallet_decash while transferring to and from the set uniswap pair address.

  This contract includes owner accessible functions to modify swiss fee and desh fee, and set the respective wallet addresses, and a one-time owner only function to set the uniswap pair address.

  It also includes functions to allow admin to transfer out ERC20 Tokens (both modern and Legacy Tokens) from this smart contract if someone accidentally sends any tokens to this smart contract. And it includes a non-standard approveAndCall function to allow for easier smart contract integrations.

Appendix

Smart Contract Summary

- Contract Context
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)
- Contract IERC20
  - From IERC20
    - allowance(address,address) (external)
    - approve(address,uint256) (external)
    - balanceOf(address) (external)
    - totalSupply() (external)
    - transfer(address,uint256) (external)
    - transferFrom(address,address,uint256) (external)
- Contract SafeMath (Most derived contract)
  - From SafeMath
    - add(uint256,uint256) (internal)
    - div(uint256,uint256) (internal)
    - div(uint256,uint256,string) (internal)
    - mod(uint256,uint256) (internal)
    - mod(uint256,uint256,string) (internal)
    - mul(uint256,uint256) (internal)
    - sub(uint256,uint256) (internal)
    - sub(uint256,uint256,string) (internal)
- Contract ERC20
  - From Context
    - _msgData() (internal)
    - _msgSender() (internal)
  - From ERC20
    - _approve(address,address,uint256) (internal)
    - _beforeTokenTransfer(address,address,uint256) (internal)

        - _burn(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _setupDecimals(uint8) (internal)
        - _transfer(address,address,uint256) (internal)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - constructor(string,string) (public)
        - decimals() (public)
        - decreaseAllowance(address,uint256) (public)
        - increaseAllowance(address,uint256) (public)
        - name() (public)
        - symbol() (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)

- Contract ERC20Burnable
    - From ERC20
        - _approve(address,address,uint256) (internal)
        - _beforeTokenTransfer(address,address,uint256) (internal)
        - _burn(address,uint256) (internal)
        - _mint(address,uint256) (internal)
        - _setupDecimals(uint8) (internal)
        - _transfer(address,address,uint256) (internal)
        - allowance(address,address) (public)
        - approve(address,uint256) (public)
        - balanceOf(address) (public)
        - constructor(string,string) (public)
        - decimals() (public)
        - decreaseAllowance(address,uint256) (public)
        - increaseAllowance(address,uint256) (public)
        - name() (public)
        - symbol() (public)
        - totalSupply() (public)
        - transfer(address,uint256) (public)
        - transferFrom(address,address,uint256) (public)

- From Context
  - _msgData() (internal)
  - _msgSender() (internal)
- From ERC20Burnable
  - burn(uint256) (public)
  - burnFrom(address,uint256) (public)

- Contract Ownable
  - From Ownable
    - constructor() (public)
    - transferOwnership(address) (public)
- Contract tokenRecipient (Most derived contract)
  - From tokenRecipient
    - receiveApproval(address,uint256,bytes) (external)
- Contract OldIERC20 (Most derived contract)
  - From OldIERC20
    - transfer(address,uint256) (external)
- Contract SwissToken (Most derived contract)
  - From Ownable
    - constructor() (public)
    - transferOwnership(address) (public)
  - From ERC20Burnable
    - burn(uint256) (public)
    - burnFrom(address,uint256) (public)
  - From ERC20
    - _approve(address,address,uint256) (internal)
    - _beforeTokenTransfer(address,address,uint256) (internal)
    - _burn(address,uint256) (internal)
    - _mint(address,uint256) (internal)
    - _setupDecimals(uint8) (internal)
    - _transfer(address,address,uint256) (internal)
    - allowance(address,address) (public)
    - approve(address,uint256) (public)
    - balanceOf(address) (public)
    - constructor(string,string) (public)

        - decimals() (public)
        - decreaseAllowance(address,uint256) (public)
        - increaseAllowance(address,uint256) (public)
        - name() (public)
        - symbol() (public)
        - totalSupply() (public)
    - From Context
        - _msgData() (internal)
        - _msgSender() (internal)
    - From SwissToken
        - _transferWithFee(address,address,uint256) (internal)
        - approveAndCall(address,uint256,bytes) (external)
        - constructor(string,string,uint256) (public)
        - setDecashFeeWallet(address) (public)
        - setDeshFeePercentX100(uint256) (public)
        - setSwissFeePercentX100(uint256) (public)
        - setSwissFeeWallet(address) (public)
        - setUniswapPairAddress(address) (public)
        - transfer(address,uint256) (public)
        - transferAnyERC20Token(address,address,uint256) (public)
        - transferAnyOldERC20Token(address,address,uint256) (public)
        - transferFrom(address,address,uint256) (public)

Slither Results

```
> slither SWISS.sol

INFO:Detectors:
OldIERC20 (swiss-token.sol#660-662) has incorrect ERC20 function
interface:OldIERC20.transfer(address,uint256) (swiss-token.sol#661)
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#incorrect-erc20-interface
INFO:Detectors:
SwissToken.transferAnyERC20Token(address,address,uint256) (swiss-token.sol#762-
764) ignores return value by IERC20(_tokenAddress).transfer(_to,_amount) (swiss-
token.sol#763)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-
return
INFO:Detectors:
ERC20.constructor(string,string).name (swiss-token.sol#321) shadows:
        - ERC20.name() (swiss-token.sol#330-332) (function)
ERC20.constructor(string,string).symbol (swiss-token.sol#321) shadows:
        - ERC20.symbol() (swiss-token.sol#338-340) (function)
SwissToken.constructor(string,string,uint256).name (swiss-token.sol#674) shadows:
        - ERC20.name() (swiss-token.sol#330-332) (function)
SwissToken.constructor(string,string,uint256).symbol (swiss-token.sol#674)
shadows:
        - ERC20.symbol() (swiss-token.sol#338-340) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-
variable-shadowing
INFO:Detectors:
Contract tokenRecipient (swiss-token.sol#656-658) is not in CapWords
Parameter SwissToken.setSwissFeePercentX100(uint256)._swissFeePercentX100 (swiss-
token.sol#679) is not in mixedCase
Parameter SwissToken.setDeshFeePercentX100(uint256)._deshFeePercentX100 (swiss-
token.sol#682) is not in mixedCase
Parameter SwissToken.setSwissFeeWallet(address)._fees_wallet_swiss (swiss-
token.sol#685) is not in mixedCase
Parameter SwissToken.setDecashFeeWallet(address)._fees_wallet_decash (swiss-
token.sol#688) is not in mixedCase
Parameter SwissToken.approveAndCall(address,uint256,bytes)._spender (swiss-
token.sol#750) is not in mixedCase
Parameter SwissToken.approveAndCall(address,uint256,bytes)._value (swiss-
token.sol#750) is not in mixedCase
Parameter SwissToken.approveAndCall(address,uint256,bytes)._extraData (swiss-
token.sol#750) is not in mixedCase
Parameter SwissToken.transferAnyERC20Token(address,address,uint256)._tokenAddress
(swiss-token.sol#762) is not in mixedCase
Parameter SwissToken.transferAnyERC20Token(address,address,uint256)._to (swiss-
token.sol#762) is not in mixedCase
Parameter SwissToken.transferAnyERC20Token(address,address,uint256)._amount
(swiss-token.sol#762) is not in mixedCase
Parameter
SwissToken.transferAnyOldERC20Token(address,address,uint256)._tokenAddress (swiss-
token.sol#765) is not in mixedCase
Parameter SwissToken.transferAnyOldERC20Token(address,address,uint256)._to (swiss-
token.sol#765) is not in mixedCase
Parameter SwissToken.transferAnyOldERC20Token(address,address,uint256)._amount
(swiss-token.sol#765) is not in mixedCase
Variable SwissToken.fees_wallet_swiss (swiss-token.sol#666) is not in mixedCase
Variable SwissToken.fees_wallet_decash (swiss-token.sol#667) is not in mixedCase
Variable SwissToken.uniswap_pair_address (swiss-token.sol#672) is not in mixedCase
Reference: https://github.com/crytic/slither/wiki/Detector-
Documentation#conformity-to-solidity-naming-conventions
INFO:Detectors:
name() should be declared external:
        - ERC20.name() (swiss-token.sol#330-332)
symbol() should be declared external:
        - ERC20.symbol() (swiss-token.sol#338-340)
decimals() should be declared external:
        - ERC20.decimals() (swiss-token.sol#355-357)
totalSupply() should be declared external:
        - ERC20.totalSupply() (swiss-token.sol#362-364)
balanceOf(address) should be declared external:
        - ERC20.balanceOf(address) (swiss-token.sol#369-371)
transfer(address,uint256) should be declared external:
        - ERC20.transfer(address,uint256) (swiss-token.sol#381-384)
        - SwissToken.transfer(address,uint256) (swiss-token.sol#705-714)
transferFrom(address,address,uint256) should be declared external:
        - SwissToken.transferFrom(address,address,uint256) (swiss-token.sol#729-
737)
        - ERC20.transferFrom(address,address,uint256) (swiss-token.sol#418-422)
increaseAllowance(address,uint256) should be declared external:
        - ERC20.increaseAllowance(address,uint256) (swiss-token.sol#436-439)
decreaseAllowance(address,uint256) should be declared external:
        - ERC20.decreaseAllowance(address,uint256) (swiss-token.sol#455-458)
burn(uint256) should be declared external:
        - ERC20Burnable.burn(uint256) (swiss-token.sol#591-593)
burnFrom(address,uint256) should be declared external:
        - ERC20Burnable.burnFrom(address,uint256) (swiss-token.sol#606-611)
transferOwnership(address) should be declared external:
        - Ownable.transferOwnership(address) (swiss-token.sol#648-652)
setSwissFeePercentX100(uint256) should be declared external:
        - SwissToken.setSwissFeePercentX100(uint256) (swiss-token.sol#679-681)
setDeshFeePercentX100(uint256) should be declared external:
        - SwissToken.setDeshFeePercentX100(uint256) (swiss-token.sol#682-684)
setSwissFeeWallet(address) should be declared external:
        - SwissToken.setSwissFeeWallet(address) (swiss-token.sol#685-687)
setDecashFeeWallet(address) should be declared external:
        - SwissToken.setDecashFeeWallet(address) (swiss-token.sol#688-690)
setUniswapPairAddress(address) should be declared external:
        - SwissToken.setUniswapPairAddress(address) (swiss-token.sol#692-695)
transferAnyERC20Token(address,address,uint256) should be declared external:
        - SwissToken.transferAnyERC20Token(address,address,uint256) (swiss-
token.sol#762-764)
transferAnyOldERC20Token(address,address,uint256) should be declared external:
        - SwissToken.transferAnyOldERC20Token(address,address,uint256) (swiss-
token.sol#765-767)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-
function-that-could-be-declared-external
INFO:Slither:swiss-token.sol analyzed (9 contracts with 46 detectors), 42
result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and
Github integration
```