2021

GCAC Audit

GLOBAL CANNABIS APPLICATIONS CORPORATION ERC-20
SMART CONTRACT TOKEN AUDIT 19/4/2021. VERSION 1.2

AUDITED BY HORIZON GLOBEX

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 Horizon Globex disclaims any liability for damage arising out of, or in connection with, this audit report. Copyright of this report remains with Horizon Globex.

Actors

- **Global Cannabis Applications Corporation:** "GCAC" a public company listed on the CSE under the ticker APP.CN, who contracted Abbey Technology to create a marketing awareness campaign. Abbey Technology decided to market to the Uniswap demographic using a GCAC ERC-20 token.
- **Abbey Technology:** The marketing technology service provider company implementing and deploying the GCAC token.
- **Horizon Globex**: 3rd Party, independent, technology company. Experts in Ethereum smart contract development, requested to perform an external audit of the GCAC token.

Introduction

Horizon Globex was requested to review the GCAC Token implementation, auditing for code vulnerabilities and bugs. Under consideration is GCACToken.sol, commit https://github.com/abbey-ch/GCAC/commit/7b048593d57e52d28770811d180c874a6d967b8a available from the Abbey Technology public GITHub repository at https://github.com/abbey-ch/GCAC .

The token uses Open Zepplin ERC-20 libraries as part of the implementation. These are not considered within the scope of the code review as they have been extensively reviewed by other providers.

The GCAC smart contract is deployed to Ethereum at this address
https://etherscan.io/address/0xc0ba6eee30932c18e6cd19f433fe84186500148a#code

Audit Summary

Type	#Issues	Status
Requirements	3	Fixed
Security	2	Fixed
Static	39	No code changes required. False positive Slither recommendations.

Overview

Methodology

Each of the extended requirements of the smart contract are audited for a corresponding implementation.

Static analysis tooling is applied to the implementation to highlight security risks. The static analysis tools slither and surya are executed on the GCACToken.sol source code.

Taxonomy

- **Requirement**: Code prevents requirements being met.
- **Security**: Code creates a security attack vector.
- **Static**: Code has potential for a bug as revealed by static analysis tools.

Smart Contract

The GCAC token is created with the intention of subsequently listing as a Uniswap pair for secondary trading.

Actors

- **Liquidity Provider**: Abbey Technology is the token holder who will seed liquidity for the corresponding Uniswap token post-deployment with 100,000 GCAC tokens and 5 ETH. The liquidity provider address is constrained by the smart contract such that it may only swap its tokens on Uniswap and may only do so after it has announced its intention to redeem liquidity from the pool at a future date, basically, an early warning system for the Uniswap community.
- **Treasury**: A holding address that is constrained by the smart contract such that it may only swap its tokens on Uniswap and may only do so after it has announced its intention to swap some of its tokens at a future date, basically, an early warning system for the Uniswap community.
- **Buyback Trader**: An address that, when used, will cause any tokens contained inside it to only be burned, in other words, these tokens may never leave this address.

Additional Requirements

In addition to the standard feature set provided by an ERC-20 implementation, the following extra requirements are implemented in the GCAC token. The general purpose of these requirements is to mitigate market manipulation and provide Uniswap community comfort:

1. Do not allow Abbey, the Uniswap liquidity provider, to remove liquidity without advance notice.
2. Do not allow Treasury to transfer tokens to anywhere other than the Uniswap pair. This is to prevent the treasury from profiting at trading venues other than Uniswap.
3. Treasury can swap her tokens into the pair, but only if X days' notice is explicitly given to the ERC-20 contract for a future date and amount. If any of the amount is subsequently transferred after the notice period, then the notice is reset and must be given again for a transfer to occur.
4. Only the Buyback Trader is permitted to burn tokens.

Issues Identified

Requirements

Fixed ✓

Notice was hard coded to 4 months exactly, rather than a variable number of days as described by requirement #3.

```
- function giveNotice(uint256 quantity) public onlyOwner {
-     notice = Notice(quantity, block.timestamp + 123 days);
- }
```

```
+ function giveNotice(uint256 amount, uint256 numDays) public onlyOwner {
+     notices[msg.sender] = Notice(amount, block.timestamp + (numDays * 1 days));
+ }
```

Fixed ✓

The give notice features were written in such a way that only the owner of the contract would ever be able to give notice. This would prevent the Treasury or Liquidity Provider from ever giving notice as per requirements #1 and #3.

```
- function giveNotice(uint256 amount, uint256 numDays) public onlyOwner {
-     notices[msg.sender] = Notice(amount, block.timestamp + (numDays * 1 days));
- }
```

```
+ function giveNotice(address who, uint256 amount, uint256 numDays) public onlyOwner {
+     notices[who] = Notice(amount, block.timestamp + (numDays * 1 days));
+ }
```

Fixed ✓

The Buyback Trader address was being prevented from burning tokens as per requirement #4

```
- function burn(uint256 amount) public onlyOwner {
-     _burn(_msgSender(), amount);
- }
```

```
+ function burn(address who, uint256 amount) public onlyOwner {
+     _burn(who, amount);
+ }
```

Replace with a new modifier, only allowing the Buyback address to burn tokens:

```
function burn() public onlyBuyback {
    _burn(buyback, balanceOf(buyback));
}
```

Security

Fixed ✓

It was possible to give notice for more tokens than your hold when executing requirement #3

```
function giveNotice(address who, uint256 amount, uint256 numDays) public onlyOwner {
    notices[who] = Notice(amount, block.timestamp + (numDays * 1 days));
}
```

```
function giveNotice(address who, uint256 amount, uint256 numDays) public onlyOwner {
+     require(who != address(0), "Can't give notice for the null address.");
+     require(amount <= balanceOf(who), "Can't give notice for more tokens than owned.")
+
      notices[who] = Notice(amount, block.timestamp + (numDays * 1 days));
  }
```

Fixed ✓

Assumed incorrectly that the caller of transfer will always be the “from” wallet of the transfer as described by _msgSender(), however, there may have been an earlier “approve” where another source calls transfer on another address’s behalf.

Also, the “require” on notice period could be circumvented due to an erroneous conditional.

```
- function _transfer(address sender, address recipient, uint256 amount) internal override {
-     require(recipient != owner, "Liquidity Pool Creator cannot receive tokens.");
-     require(_msgSender() != buyback, "Buyback cannot transfer tokens, you must burn them.");
-     if(sender == treasury) require(_msgSender() == router, "Treasury account tokens can only be moved by the Uniswap
-     Notice memory notice = notices[sender];
-     require(notice.releaseDate != 0 || block.timestamp >= notice.releaseDate, "Notice period has not yet expired.");
-     require(notice.amount == 0 || amount <= notice.amount, "Treasury can't transfer more tokens than given notice for
-     require(notice.amount == 0 || notice.tokenType == TokenType.GCAC, "The notice given for this user is the wrong to
```

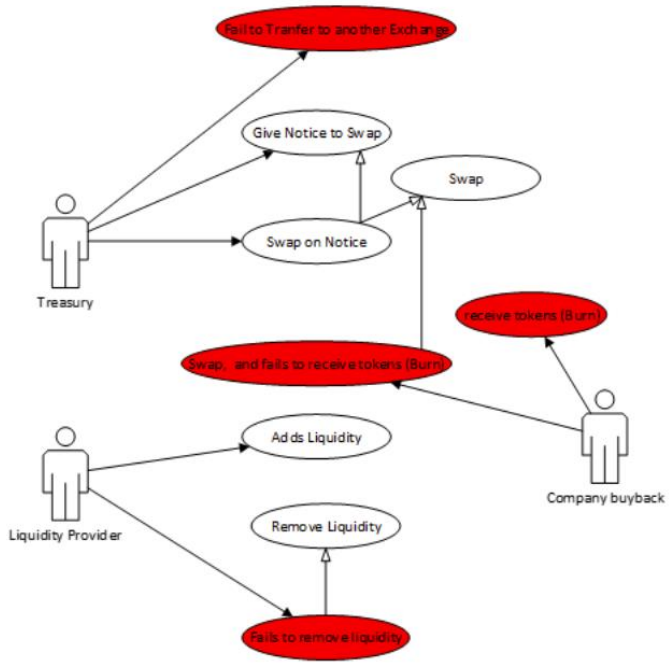
```
+ function _transfer(address sender, address recipient, uint256 amount) internal override {
+     require(recipient != owner, "Liquidity Pool Creator cannot receive tokens.");
+     require(sender != buyback, "Buyback cannot transfer tokens, it can only burn.");
+     if(sender == treasury) {
+         require(_msgSender() == router, "Treasury account tokens can only be moved by the Uniswap Router.");
+         require(noticeTreasury.releaseDate != 0 && block.timestamp >= noticeTreasury.releaseDate, "Notice period has
+         require(amount <= noticeTreasury.amount, "Treasury can't transfer more tokens than given notice for.");
+         require(noticeTreasury.tokenType == TokenType.GCAC, "The notice given for this user is the wrong token type."
+
+         // Clear the remaining notice balance, this prevents giving notice on all tokens and
+         // trickling them out.
+         noticeTreasury = Notice(0, 0, TokenType.Unknown);
+     }
```

Static

No code changes were required because of static analyses. Full report available in the appendix.

Appendix

UML



Unit Testing

```
Contract: GCAC
    √ [Negative] Non-owner can't use setBuyback (1122ms)
    √ [Negative] Non-owner can't use setTreasury (256ms)
    √ [Negative] Non-owner can't use setRouter (193ms)
    √ [Negative] Non-owner can't use setPairAddress (179ms)
    √ [Negative] Non-owner can't use giveNotice (191ms)
    √ [Negative] Can't use giveNotice until the Pair Contract is set (208ms)
    √ [Negative] Non-owner can't use transferLiquidityTokens (193ms)
    √ Set buyback as owner (317ms)
    √ [Negative] Set buyback as owner a second time (283ms)
    √ Set treasury as owner (365ms)
    √ [Negative] Set treasury as owner a second time (331ms)
    √ Set Uniswap Router as owner (445ms)
    √ [Negative] Set Uniswap Router as owner a second time (290ms)
    √ Transfer tokens to treasury as owner (317ms)
    √ [Negative] Attempt to send tokens direct from treasury (213ms)
    √ Transfer tokens to buyback as owner, Uniswap prevents us from auto burning them.
(489ms)
    √ [Negative] Attempt to send tokens from the buyback account (194ms)
    √ [Negative] Attempt to move buyback tokens with transferFrom (363ms)
    √ [Negative] Cannot move Liquidity Tokens if Pair Contract Address is not set. (239ms)
    √ Set the Pair Contract Address as Owner (256ms)
    √ [Negative] Set Pair Contract as owner a second time (337ms)

  Contract: GCAC
    √ Setup Contract for Tests (587ms)
    √ Transfer tokens to treasury as owner (407ms)
    √ [Negative] Attempt to send tokens to the owner (122ms)
    √ [Negative] Attempt to send tokens direct from treasury (193ms)
```

```
    √ Transfer tokens to buyback as owner, Uniswap prevents us from auto burning them.
(397ms)
    √ [Negative] Attempt to send tokens from the buyback account (207ms)
    √ [Negative] Attempt to move buyback tokens with transferFrom (285ms)
    √ [Negative] Cannot move Liquidity Tokens if Pair Contract Address is not set. (239ms)
    √ Set the Pair Contract Address as Owner (288ms)
    √ [Negative] Set Pair Contract as owner a second time (286ms)

  Contract: GCAC
    √ Setup Contract for Tests (1294ms)
    √ [Negative] Attempt to give notice by a non-owner address (179ms)
    √ [Negative] Attempt to give notice for the null address (193ms)
    √ [Negative] Attempt to give notice for a non-restricted address (195ms)
    √ [Negative] Attempt to give notice for more GCAC tokens than owned (249ms)
    √ [Negative] Attempt to give notice for more Liquidity Tokens than owned (283ms)
    √ Overwrite an expired Treasury GCAC notice (832ms)
    √ [Negative] Attempt to overwrite an existing Treasury GCAC notice (543ms)
    √ Overwrite an expired Liquidity Token notice (1056ms)
    √ [Negative] Attempt to overwrite an existing Treasury GCAC notice (642ms)

  Contract: GCAC
    √ Setup Contract for Tests (1434ms)
    √ Create a zero second notice for Treasury GCAC Tokens and spend some of it (1470ms)
    √ [Negative] Attempt to swap when no Treasury notice is given (653ms)
    √ [Negative] Attempt to spend more tokens than notice was given for (959ms)
    √ [Negative] Attempt to spend more GCAC tokens than notice was given for (937ms)
    √ [Negative] Attempt to spend GCAC tokens before notice expires (576ms)

  Contract: GCAC
    √ Setup Contract for Tests (982ms)
    √ [Negative] Attempt to give notice for more Liquidity Tokens than owned (506ms)
    √ [Negative] Attempt to transfer Liquidity Tokens where notice was not created (370ms)
    √ Set zero second Liquidity Token notice and transfer some of it (2662ms)
    √ Set zero second Liquidity Token notice and transfer all of it (2248ms)
    √ [Negative] Attempt to spend more Liquidity Tokens than owned (948ms)

  Contract: GCAC
    √ Setup Contract for Tests (349ms)
    √ [Negative] Attempt to burn tokens as non-buyback account (284ms)
    √ Burn buyback tokens (1029ms)

  Contract: GCAC
    √ Setup Contract for Tests (2197ms)
    √ [Negative] Attempt to swap when no Treasury notice is given (1005ms)
    √ [Negative] Attempt to spend GCAC tokens before notice expires (1975ms)
    √ Create a zero second notice for Treasury GCAC Tokens and spend some of it (9521ms)
    √ [Negative] Attempt to swap when Treasury notice was used and should revert to zero/empty
again (1149ms)

  61 passing (1m)
```

Slither Output

```
INFO:Detectors:
Reentrancy in GCACToken.transferLiquidityTokens(address,uint256) (GCACToken.sol#252-264):
        External calls:
        - pair.transfer(to,amount) (GCACToken.sol#260)
        State variables written after the call(s):
        - noticeLiquidity = Notice(0,0,TokenType.Unknown) (GCACToken.sol#263)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#reentrancy-
vulnerabilities-1
INFO:Detectors:
GCACToken.transferLiquidityTokens(address,uint256) (GCACToken.sol#252-264) ignores return
value by pair.transfer(to,amount) (GCACToken.sol#260)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#unused-return
INFO:Detectors:
GCACToken.constructor(uint256,string,string).name (GCACToken.sol#154) shadows:
        - ERC20.name() (@openzeppelin/contracts/token/ERC20/ERC20.sol#60-62) (function)
        - IERC20Metadata.name()
(@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#14) (function)
```

```
GCACToken.constructor(uint256,string,string).symbol (GCACToken.sol#154) shadows:
        - ERC20.symbol() (@openzeppelin/contracts/token/ERC20/ERC20.sol#68-70) (function)
        - IERC20Metadata.symbol()
(@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#19) (function)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#local-variable-
shadowing
INFO:Detectors:
GCACToken.setBuyback(address).who (GCACToken.sol#162) lacks a zero-check on :
                - buyback = who (GCACToken.sol#164)
GCACToken.setTreasury(address).who (GCACToken.sol#170) lacks a zero-check on :
                - treasury = who (GCACToken.sol#172)
GCACToken.setRouter(address).who (GCACToken.sol#178) lacks a zero-check on :
                - router = who (GCACToken.sol#180)
GCACToken.setPairAddress(address).who (GCACToken.sol#186) lacks a zero-check on :
                - pairAddress = who (GCACToken.sol#188)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#missing-zero-address-
validation
INFO:Detectors:
GCACToken.giveNotice(address,uint256,uint256) (GCACToken.sol#199-222) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(noticeTreasury.releaseDate == 0 || block.timestamp >=
noticeTreasury.releaseDate,Cannot overwrite an active existing notice.) (GCACToken.sol#208)
        - require(bool,string)(noticeLiquidity.releaseDate == 0 || block.timestamp >=
noticeLiquidity.releaseDate,Cannot overwrite an active existing notice.) (GCACToken.sol#214)
GCACToken._transfer(address,address,uint256) (GCACToken.sol#230-245) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(noticeTreasury.releaseDate != 0 && block.timestamp >=
noticeTreasury.releaseDate,Notice period has not been set or has not expired.)
(GCACToken.sol#235)
        - require(bool,string)(amount <= noticeTreasury.amount,Treasury can't transfer more
tokens than given notice for.) (GCACToken.sol#236)
        - require(bool,string)(noticeTreasury.tokenType == TokenType.GCAC,The notice given for
this user is the wrong token type.) (GCACToken.sol#237)
GCACToken.transferLiquidityTokens(address,uint256) (GCACToken.sol#252-264) uses timestamp for
comparisons
        Dangerous comparisons:
        - require(bool,string)(noticeLiquidity.releaseDate != 0 && block.timestamp >=
noticeLiquidity.releaseDate,Notice period has not been set or has not expired.)
(GCACToken.sol#255)
        - require(bool,string)(amount <= noticeLiquidity.amount,Insufficient Liquidity Token
balance.) (GCACToken.sol#256)
        - require(bool,string)(noticeLiquidity.tokenType == TokenType.LiquidityPool,The notice
given for this user is the wrong token type.) (GCACToken.sol#257)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#block-timestamp
INFO:Detectors:
Different versions of Solidity is used in :
        - Version used: ['0.8.3', '>=0.4.22<0.9.0', '^0.8.0']
        - ^0.8.0 (@openzeppelin/contracts/utils/Context.sol#3)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC20/ERC20.sol#3)
        - 0.8.3 (GCACToken.sol#68)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC20/IERC20.sol#3)
        - ^0.8.0 (@openzeppelin/contracts/token/ERC20/extensions/IERC20Metadata.sol#3)
        - >=0.4.22<0.9.0 (Migrations.sol#2)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#different-pragma-
directives-are-used
Pragma version0.8.3 (GCACToken.sol#68) necessitates a version too recent to be trusted.
Consider deploying with 0.6.12/0.7.6
Pragma version^0.8.0 (@openzeppelin/contracts/token/ERC20/IERC20.sol#3) necessitates a version
too recent to be trusted. Consider deploying with 0.6.12/0.7.6
setBuyback(address) should be declared external:
        - GCACToken.setBuyback(address) (GCACToken.sol#162-165)
setTreasury(address) should be declared external:
        - GCACToken.setTreasury(address) (GCACToken.sol#170-173)
setRouter(address) should be declared external:
        - GCACToken.setRouter(address) (GCACToken.sol#178-181)
setPairAddress(address) should be declared external:
        - GCACToken.setPairAddress(address) (GCACToken.sol#186-189)
giveNotice(address,uint256,uint256) should be declared external:
        - GCACToken.giveNotice(address,uint256,uint256) (GCACToken.sol#199-222)
transferLiquidityTokens(address,uint256) should be declared external:
        - GCACToken.transferLiquidityTokens(address,uint256) (GCACToken.sol#252-264)
burn() should be declared external:
        - GCACToken.burn() (GCACToken.sol#270-272)
setCompleted(uint256) should be declared external:
```

```
        - Migrations.setCompleted(uint256) (Migrations.sol#16-18)
Reference: https://github.com/crytic/slither/wiki/Detector-Documentation#public-function-that-could-be-declared-external
INFO:Slither:. analyzed (6 contracts with 72 detectors), 39 result(s) found
INFO:Slither:Use https://crytic.io/ to get access to additional detectors and Github integration
```