Security Audit Report

ACoconut BTC（acBTC）

Phase One: AC Genesis Liquidity Mining

SECBIT

Oct 7, 2020

1. Introduction

ACoconut BTC (acBTC) is a synthetic BTC ERC20 Token on Ethereum. acBTC integrates native BTC and ERC-20 BTC, with the swap, lending, and yield generating applications into one highly secure, efficient, and usable standard. SECBIT Labs conducted an audit from Sep 24th to Oct 7th, 2020, including an analysis of Smart Contracts in 3 areas: **code bugs**, **logic flaws**, and **risk assessment**. The audit results show that ACoconut BTC (acBTC) Phase One has no critical security risks. The SECBIT team has some tips on logical implementation, potential risks, and code revising (see part 4 for details).

Type	Description	Level	Status
Logical Implementation	4.3.1 In ACoconut/ACoconutBTC contract， the sensitive operation of `setMinter()` does not add events.	Info	Fixed
Logical Implementation	4.3.2 In ACoconutBTC contract, the implementation of `burn()` function is in doubt.	Low	Fixed
Code Revising	4.3.3 In StrategyACoconutBTC contract, `reserveRecipient` variable is not visible.	Info	Fixed
Logical Implementation	4.3.4 In StrategyACoconutBTC contract, the permissions of `withdraw()` and `withdrawAll()` are in doubt.	Info	Discussed
Logical Implementation	4.3.5 StrategyCurveRenBTC::harvest() function has no permission control and may form arbitrage opportunities.	Medium	Fixed
Code Revising	4.3.6 ACoconut contract only needs to inherit ERC20Capped contract, and _beforeTokenTransfer() function does not need to be overloaded.	Info	Fixed

2. Project Information

This part describes the necessary information and code structure.

2.1 Basic information

The basic information about acBTC is shown below:

- Project website
  - https://acbtc.fi
- Design details of the protocol
  - https://docs.acbtc.fi
- Smart contract code
  - https://github.com/nutsfinance/acBTC, commit c04cb435b70f4194586ba20666abfc57fd020ce4

2.2 Contract List

The following content shows the main contracts included in acBTC project:

Contract	Description
Account.sol	Trading Account Contract
AccountFactory.sol	Trading account generation contract
ACoconut.sol	AC Token Contract
ACoconutBTC.sol	acBTC Token contract
ACoconutSwap.sol	Implementation of acSwap
ACoconutSwapProxy.sol	The proxy contract for the deployment of acSwap
ACoconutVault.sol	Implementation of acVault
CurveRenCrvMigrator.sol	RenCRV migration contract
StrategyACoconutBTC.sol	acBTC earning strategy contract
StakingApplication.sol	Implementation of Staking Application
AdminUpgradeabilityProxy.sol	Upgradeable proxy contract with authorization
BaseUpgradeabilityProxy.sol	Upgradeable proxy contract
Initializable.sol	Initialization control contract
Proxy.sol	Proxy contract
Controller.sol	Vault controller contract
RewardedVault.sol	Rewarded vault contract
StrategyCurveRenBTC.sol	RenCRV earning strategy contract
Vault.sol	Vault contract

Note: ACoconutSwap and CurveRenCrvMigrator contracts are still under development and testing, which are not included in the audit scope of acBTC Phase One.

3. Code Analysis

This part describes code assessment details, including two items: "role classification" and "functional analysis".

3.1 Role Classification

There are several key roles in the protocol, namely Governance Account, Minter, Wallet Account, and Common Account.

- Governance Account
  - Description Contract governor
  - Authority
    - Set the minter of AC Token or acBTC Token
    - Set the basic parameters of the contract
  - Method of Authorization The creator of the contract, or authorized by the transferring of governance account
- Minter
  - Description The accounts minting AC Token or acBTC Token
  - Authority
    - Mint
  - Method of Authorization Authorized by governance account
- Wallet Account
  - Description The account used to interact with the acBTC contract
  - Authority
    - Hierarchical authority management (owner, admin, operator)
    - Owner account can grant or revoke admin accounts
    - Admin account can grant or revoke operator accounts
    - Operator account can transfer ETH and ERC20 Token in the wallet account
    - Operator account can approve ERC20 Token in the wallet account
    - Operator account can transfer the ERC20 Token of owner account when approved by the owner account
    - Operator account can invoke delegate calls
  - Method of Authorization
    - Owner account is the creator of the contract or authorized by the

            transferring of the owner account
        - Admin account is authorized by the owner account
        - Operator account is authorized by the admin account
- Common Account
    - Description The accounts holding AC Token or acBTC Token
    - Authority
        - Transfer tokens in its account
        - Approve other accounts to transfer
        - AC Token holders can participate in the governance votes
    - Method of Authorization AC Token or acBTC holders

3.2 Functional Analysis

acBTC is a synthetic BTC ERC20 Token contract. We can divide the critical functions of the acBTC contract into several parts:

- acBTC

acBTC is a synthetic ERC20 BTC token backed by a basket of ERC20 BTC tokens. It's built on top of Curve's StableSwap algorithm with integrated saving and swap applications.

- acVault

acVault is a renCrv vault that earns yields in renCrv before migration and becomes an acBTC vault that makes yields in acBTC after migration.

- acSwap

acSwap is a decentralized exchange for ERC20 BTC tokens. It manages a basket of ERC20 BTC tokens, including WBTC and renBTC, and bootstraps the value of acBTC.

Note: acSwap function is still under development and testing, which is not included in the audit scope of acBTC Phase One.

4. Audit Detail

This part describes the process and detailed results of the audit and demonstrates the problems and potential risks.

4.1 Audit Process

The audit strictly followed the audit specification of SECBIT Labs. We analyzed the project from code bug, logical implementation, and potential risks. The process consists of four steps:

- Fully analysis of code line by line.
- Evaluation of vulnerabilities and potential risks revealed in the source code.
- Communication on assessment and confirmation.
- Audit report writing.

4.2 Audit Result

After scanning with adelaide, sf-checker, and badmsg.sender (internal version) developed by SECBIT Labs and open source tools including Mythril, Slither, SmartCheck, and Securify, the auditing team performed a manual assessment. The team inspected the contract line by line, and the result could be categorized into twenty-one types:

Number	Classification	Result
1	Normal functioning of features defined by the contract	✓
2	No obvious bug (e.g., overflow, underflow)	✓
3	Pass Solidity compiler check with no potential error	✓
4	Pass common tools check with no obvious vulnerability	✓
5	No obvious gas-consuming operation	✓
6	Meet with ERC20	✓
7	No risk in low-level call (call, delegatecall, callcode) and in-line assembly	✓
8	No deprecated or outdated usage	✓
9	Explicit implementation, visibility, variable type and Solidity version number	✓
10	No redundant code	✓
11	No potential risk manipulated by timestamp and network environment	✓
12	Explicit business logic	✓

13	Implementation consistent with annotation and other info	✓
14	No hidden code about any logic that is not mentioned in design	✓
15	No ambiguous logic	✓
16	No risk threatening the developing team	✓
17	No risk threatening exchanges, wallets, and DApps	✓
18	No risk threatening token holders	✓
19	No privilege on managing others' balances	✓
20	No unnecessary minting method	✓
21	Correct managing hierarchy	✓

4.3 Issues

4.3.1 In ACoconut/ACoconutBTC contract，the sensitive operation of `setMinter()` does not add events.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Operation auditability	Fixed

Description

The sensitive operation of `setMinter()` in the ACoconut/ACoconutBTC contract does not add events. Users cannot easily check the minter status of the contract.

```
// ACoconut.sol/ACoconutBTC.sol
function setMinter(address _user, bool _allowed) public {
    require(msg.sender == governance, "not governance");
    minters[_user] = _allowed;
}
```

Suggestion

It is recommended to add events to `setMinter()` in ACoconut/ACoconutBTC contract and facilitate the community to audit minter permissions.

Status

It is fixed according to the suggestion in 6bf5964.

4.3.2 In ACoconutBTC contract, the implementation of `burn()` function is in doubt.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Low	Permission control	Fixed

Description

In the `burn()` function of the ACoconutBTC contract, the minter account can perform any burn operation, but the reason is not explained.

```
// ACoconutBTC.sol
function burn(address _user, uint256 _amount) public {
    require(minters[msg.sender], "not minter");
    _burn(_user, _amount);
}
```

Suggestion

It is recommended to add a description of whether it is a need.

Status

A description has been added as suggested in 6bf5964.

4.3.3 In StrategyACoconutBTC contract, `reserveRecipient` variable is not visible.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Variable visibility	Fixed

Description

In StrategyACoconutBTC contract, `reserveRecipient` is a private variable with a setter function but no getter function. Because `reserveRecipient` involves transfer operations, the absence of a getter function may easily cause misunderstandings.

Suggestion

It is recommended to add a getter function of `reserveRecipient` to the contract.

Status

It is fixed according to the suggestion in 6bf5964.

4.3.4 In StrategyACoconutBTC contract, the permissions of `withdraw()` and `withdrawAll()` are in doubt.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Info	Permission control	Discussed

Description

In StrategyACoconutBTC contract, the permissions of `withdraw()` and `withdrawAll()` functions are inconsistent with the functions of the same name in StrategyCurveRenBTC contract.

Suggestion

It is recommended to clarify the permission design of the `withdraw()` and `withdrawAll()` functions in the contract.

Status

The issue has been discussed. The contract is designed so and remains the current status.

4.3.5 StrategyCurveRenBTC::harvest() function has no permission control and may form arbitrage opportunities.

Risk Type	Risk Level	Impact	Status
Logical Implementation	Medium	Permission control	Fixed

Description

The `harvest()` function in StrategyCurveRenBTC contract has no permission control. It may be manipulated to invoke other DEX for trading, thereby forming arbitrage opportunities for others and affecting earnings.

Suggestion

Add permission control to `harvest()` function.

Status

It is fixed according to the suggestion in c959dc2.

4.3.6 ACoconut contract only needs to inherit ERC20Capped contract, and _beforeTokenTransfer() function does not need to be overloaded.

Risk Type	Risk Level	Impact	Status
Code Revising	Info	Redundant code	Fixed

Description

The ACoconut contract's implementation inherits two contracts, ERC20 and ERC20Capped, but the two contracts themselves have an inheritance relationship. Therefore, the ACoconut contract only needs to inherit the ERC20Capped contract when it is implemented.

```
// ACoconut.sol
contract ACoconut is ERC20, ERC20Capped {

    ...

    function _beforeTokenTransfer(address from, address to, uint256
amount) internal override(ERC20, ERC20Capped) {
        ERC20Capped._beforeTokenTransfer(from, to, amount);
    }

    ...
}
```

Suggestion

It is recommended to only inherit the ERC20Capped contract in implementation.

Status

Fixed according to the suggestion in a0dec81.

5. Conclusion

After auditing and analyzing the contract code of it, SECBIT Labs found some issues to optimize and proposed corresponding suggestions, which have been shown above. For the issues found in this report, the acBTC developers have them all fixed in the latest version of the code. SECBIT Labs holds that the acBTC project has high code quality, detailed documentation, and complete test cases. acBTC Phase One fully implements the initial distribution of AC Token, especially ACoconutVault can simultaneously mine CRV and AC Token and provides a reserved interface for future migration to acBTC. In addition, ACoconutSwap and CurveRenCrvMigrator contracts are undergoing final testing and verification. They will be the main audit targets of acBTC Phase Two.

Disclaimer

APPENDIX

Vulnerability/Risk Level Classification

Level	Description
High	Severely damage the contract's integrity and allow attackers to steal ethers and tokens, or lock ethers inside the contract.
Medium	Damage contract's security under given conditions and cause impairment of benefit for stakeholders.
Low	Cause no actual impairment to contract.
Info	Relevant to practice or rationality could possibly bring risks.

SECBIT Labs is devoted to constructing a common-consensus, reliable and ordered blockchain economic entity.

http://www.secbit.io

audit@secbit.io

@secbit_io