Biggest Lab, Co. Ltd

Cloudfloat AUD ERC-20 Contract Audit

October 31st, 2022

Initial audit is based on GitHub repo at: https://github.com/CloudfloatAU/cloud-aud/tree/coding-style

```
commit ac3a78f92181ce4e98f23456b0ed65e75e269b3f (HEAD -> coding-style, origin/coding-style)
```

Final audit is based on: https://github.com/CloudfloatAU/cloud-aud/tree/main

```
commit e31ea76b44317cf15930a9f468fa198228033f08 (HEAD -> main, origin/main, origin/HEAD)
```

Contract deployed at: https://polygonscan.com/address/0xb1ef313e3119e13f827e14d7c90e03180ac828ed#code

BACKGROUND:

Cloudfloat has engaged Biggest Lab to conduct a code audit for their cloud-aud ERC-20 interest bearing token. This is a combined functional audit - meaning the contract is fit for purpose; and security audit – meaning it cannot be exploited to be used against its intended purpose. Most code audits follow no standard processes and use overloaded ambiguous terms to describe how “secure” the contract is. There being no objective standard definition in the DeFi space for “secure” or terms like “high” vs. “low” risk – or even “risk”, it is no surprise therefore that a lot of the major exploits in the sector were made against audited contracts with insufficient methodologies.

Biggest Lab has been actively working on remedying this serious oversight by the cryptoledger community by working with ISECOM.org who created the Open Source Security Testing Methodology Manual (OSSTMM) in 2001. OSSTMM is the foundational methodology that the NIST 800.53 & OWASP standards are based upon and has been adopted as a security standard for every branch of the US military, NATO, the Vatican and CERN, amongst others. Biggest Lab’s founder & CTO, Benjamin Scherrey, has been a security consultant since 1996 having been a special consultant to Internet Security Systems and the CTO of a security firm founded by a former Deputy Attorney General of the United States responsible for Computer Crime and Anti-Terrorism. Since encountering the OSSTMM methodology around 2004, it has been the process model Mr. Scherrey has adapted and applied to all of his security related efforts.

The collaboration between ISECOM.org and Biggest Lab to incorporate standards for decentralized code bases (on-chain contracts) and the supporting off-chain systems that integrate with them is intended to be incorporated into the forthcoming OSSTMM v. 4. While this audit does not have the scope necessary for a full security evaluation, it does introduce a lot of the concepts, terminology, and

specialized capabilities necessary to help create an objective definition of security in the domain of decentralized systems.

SCOPE OF AUDIT:

The scope of this audit is limited strictly to the Smart Contract code in the afore-mentioned source code repos, specifically the token.vy code, and the associated unit tests that exercise the contract's capabilities. Figure 1, below, shows the "Smart Contracts" scope in its greater context in which is operates.

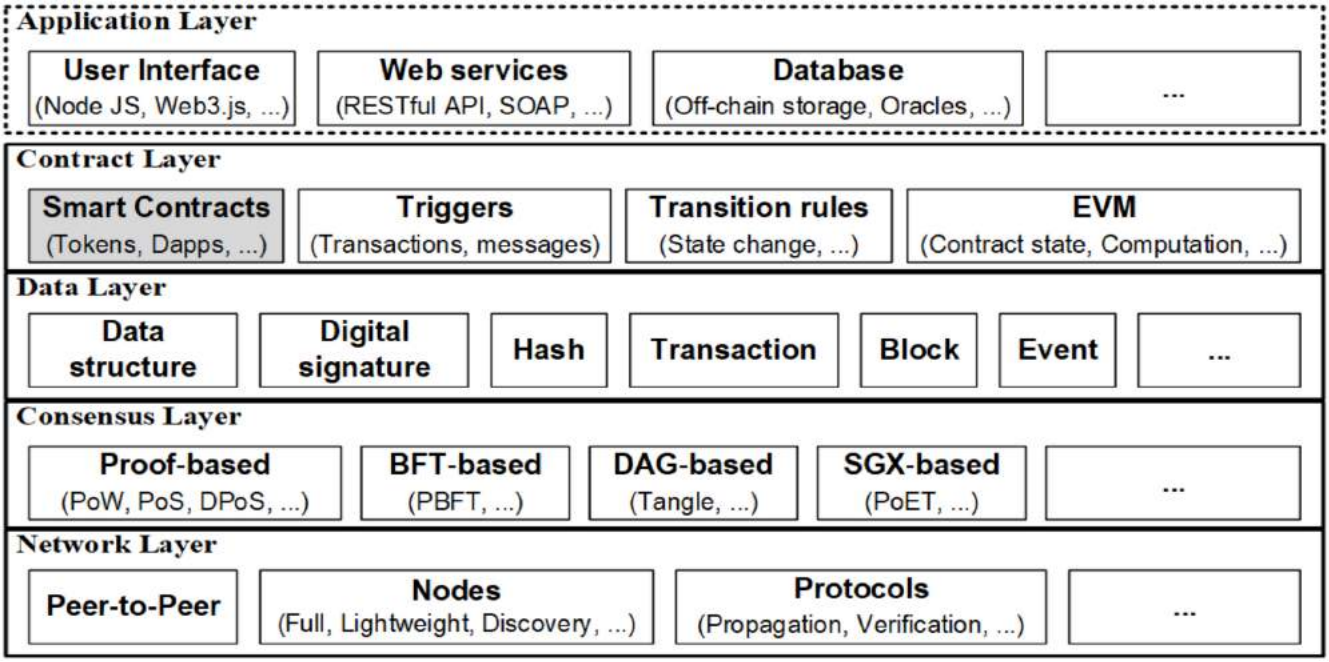

Figure 1: Architecture of the Ethereum block chain in layers from TokenHook paper (Rahimian, Clark)

Not included in the scope of this audit are the non-functional aspects of a system which include the capacity, performance, availability, and correctness of the environment and tools which provide the context in which this contract operates. Non-functional components for this contract code include the

Biggest Lab, Co. Ltd

Cloudfloat AUD ERC-20 Contract Audit

October 31st, 2022

Vyper compiler, the node software that implements the EVM executing the code, and the services or components that define and implement the decentralized consensus protocol which ultimately provides for finality of operations against the contract. Those are considered trusted and correct components for the scope of this audit.

METHODOLOGY:

Following an adapted OSSTMM protocol, the audit first sought to identify & categorize the various stakeholders (both friendly & unfriendly) and the operations each might perform as it interacts with the SUT (system under test). The items were then categorized according to OSSTMM definitions (further defined at the end of this document).

We then duplicated the SUT environment to reflect its operational deployment and conducted both automated test script executions and manual code inspection following models developed by Michael E. Fagan and Barry Boehm. The initial findings were shared with Cloudfloat who then made corrections and the process was applied again on the resulting code base.

OPSEC:

Visibility:	Contract Data { totalSupply, balanceOf, allowance }
Access:	__init__()
	name()
	symbol()
	decimals()
	transfer(receiver: address, amount: uint256) → bool
	batchTransfer(payments: DynArray[Payment, MAX_PAYMENTS], min_gas_remaining: uint256 = MIN_GAS_REMAINING) → unit256
	transferFrom(sender: address, receiver: address, amount: uint256) → uint256
	approve(spender: address, amount: uint256) → bool

	burn(amount: uint256) → bool
	mint(receiver: address, amount: uint256) → bool
	transferOwnership(target: address) → bool
	transferMinter(target: address) → bool
	Gas Consumption Limits enforced by Node protocol implementation (partial scope)
Trust:	Node Code (out of scope – assume trusted)
	Consensus Model (out of scope – assume trusted)
	Finality Assumptions (out of scope – assume trusted)

CONTROLS Class A:

Authentication:	msg.sender
	allowance for transferFrom method
Indemnification:	n/a
Resilience:	chain rollback
	tx revertion on fail
Subjugation:	node code/protocol
Continuity:	decentralized nodes

CONTROLS Class B:

Non-repudiation:	(not applicable with the strict existing definition of OSSTMM)
Confidentiality:	(not in scope for this audit)
Privacy:	(under review as to how to apply OSSTMM model to DeFi contract)
Integrity:	on-chain event logs

Biggest Lab, Co. Ltd

Cloudfloat AUD ERC-20 Contract Audit

October 31st, 2022

Alarm: off-chain monitoring agents

LIMITATIONS:

Vulnerability: transferFrom front running *(necessary for ERC-20 compliance - accepted)*

mint to ZERO_ADDRESS *(fixed)*

Weakness: minting process management *(indefinite number of minters - fixed)*

owner can be minter *(single point of failure/key control process - accepted)*

no event when minter changes *(fixed)*

no event when initial owner/minter is established *(during init - accepted)*

Concern: non-owner can burn tokens *(as designed - accepted)*

Anomaly: transferFrom does not distinguish between no authorization vs. inadequate authorization to transfer *(no-impact – accepted)*

FINDINGS:

The selection of the Vyper programming language over the more common Solidity language significantly reduced the attack surface available to those intent on exploiting this ERC-20 contract. Many common exploits in smart contracts written in Solidity are structurally impossible to express in Vyper. This reduced the size of the automated test coverage tooling and the time required to inspect the code manually thus reducing the immediate cost of this audit and ongoing maintenance costs that Cloudfloat may incur as it supports and augments the system going forward.

The first round of tests discovered 2 Vulnerabilities, 4 Weaknesses, 1 Concern, and 1 Anomaly. These are detailed below.

After Cloudfloat acted on the initial findings and clarified some of the use case specifications, the resulting code left only 1 vulnerability (inherent in the need for ERC-20 compliance therefore deemed acceptable), 2 weaknesses (deemed acceptable and mitigated by key management outside of this audit scope) and no unresolved Concerns or Anomalies.

Biggest Lab, Co. Ltd

Cloudfloat AUD ERC-20 Contract Audit

October 31st, 2022

ERC-20 Compliance

The Ethereum ERC-20 Standard specifies 6 required functions, 3 optional functions, and 2 events for interoperability with decentralized and web3 applications. The CAUD token complies with all aspects of this standard with two minor **No Risk** exceptions:

1. The naming convention for parameter names is different than the standard uses i.e. sender instead of _from, receiver instead of _to, amount instead of _value, and spender instead of _spender. *(outcome: no impact – accepted)*
2. **Anomaly** → The transferFrom method does not distinguish between the sender having no authorization to transfer tokens on behalf of the owner and the sender simply having an insufficient authorization to send the requested quantity of tokens on behalf of the owner. *(outcome: no impact - accepted)*

Neither of these exceptions has a practical impact on the contract's fit for purpose or security. The limited transferFrom expressiveness only makes certain use cases that one might want to test for redundant but does not change the functional result of any transferFrom transaction request. Otherwise the code handles all the exceptional behavior described in the standard correctly.

Vulnerability → transfer_From Front Running : There is a known potential front running exploit inherent in the ERC-20 standard that this contract is vulnerable to related to the 'approval' and 'transferFrom' mechanics. If Alice approves Bob's contract for X tokens and then later decides to change the approval to Y tokens, Bob may see the approval change in the mempool and front run that change by first calling 'transferFrom' for quantity X then, after Alice's approval for quantity Y tokens comes through, Bob can make a second call to 'transferFrom' for quantity Y tokens allowing Bob to transact X+Y quantity of tokens of Alice's rather than her intended quantity Y tokens. Services like Flashbots.net make exploiting such decentralized race conditions fairly easy to execute for bad actors. There is no completely "standard" ERC-20 method for fixing this vulnerability in the contract but there are discussions of heuristics and quasi-standard methods of handling it. Regular ERC-20 holders are recommended to adopt a practice of always making an 'approve' request with a value of 0 and then waiting for that tx to complete without a 'transferFrom' request being executed before then calling 'approve' again with the new desired limit. As in all things decentralized, once permission to access

tokens is granted, there's no way to change one's mind and deterministically remove that permission before the permission is used. *(outcome: necessary for ERC-20 compliance - accepted)*

Extended Token Capabilities

The CAUD token has an extended form of the 'transfer' method for multiple dynamic batched payments called 'batchTransfer' that provides significant gas efficiencies over multiple calls to 'transfer'. Methods with dynamic behavior are often susceptible to gas-exhaustion attacks resulting in denial of service. CAUD's 'batchTransfer' implementation mitigates this by performing gas accounting inside the method itself and trades off the potential of only completing partial batches and having to resubmit unprocessed payments in return for maximized used of available gas by the calling client.

Consideration of Intent: There exists no corresponding 'batchTransferFrom' method in the CAUD contract thus limiting batch transfer transactions strictly to being called by the token holder. *(outcome: as designed - accepted)*

Practical Token Capabilities

Although not part of the standard itself, various additional capabilities are either required or often present for ERC-20 tokens.

Minting Tokens

The minting semantics of the CAUD contract are currently as follows:

- The owner may always mint new CAUD.
- The owner may give unlimited additional addresses minting capabilities.
- The owner may remove existing minters' (other than the owner) minting capabilities.
- Minters that are not the owner may not add or remove minting capabilities.
- Changing ownership (see below) also removes minting capabilities that may have been additionally assigned (redundantly) to the owner address.

Weakness → minting process management : There can be an unlimited number of addresses with minting capabilities and there is no direct way to get a list of who can mint. As minting capabilities are stored in a map of addresses => boolean capability indicators, maps cannot be iterated, and adding or removing minting capabilities generates no event (see below), it can be difficult to track who can and

cannot mint tokens. Reconsider changing to a single minter with the explicit role, keeping a clear separation of concerns by not having the owner automatically (or ever) be able to mint, or have a fixed list with a small maximum number of minters that can easily be iterated. The nature of the CAUD token is that as new assets backing the value of the token are received by the company, new CAUD may be frequently minted (or burned when removed). This means CAUD will invoke minting operations more frequently and adhoc than most ERC-20 tokens. Having a controls and strict limits as to how many people can even potentially have minting capabilities is critical for reducing this obvious and attractive attack surface. Recommend that the owner NOT be able to mint; there be only one address who may mint (which may be set by owner); and that this minter is preferably another smart contract that requires multiple authorizations to invoke the mint function. *(outcome: fixed)*

Weakness → no event when minter changes: When new minting capabilities are added or removed there is no event generated. It is a strong best practice that any time contract state data is updated an event with the new state should be generated. This provides critical support for external contract monitoring for unexpected activities to be easily implemented. Recommend that the full extent addresses having minting capabilities be discoverable via events published on the block chain, including initialization of the contract with the initial minter (if appropriate). *(outcome: fixed)*

Vulnerability → mint to ZERO_ADDRESS: Minters are not prohibited from minting to the ZERO_ADDRESS. If this happens the total supply calculations are irreversibly broken. *(outcome: fixed)*

Burning Tokens

The CAUD contract prohibits intentional or accidental transfer of tokens to the ZERO_ADDRESS which effectively burns tokens. In order to burn tokens and remove them permanently from the supply a call to ‘burn’ must be explicitly made. Burn semantics are presently as follows:

- Anyone who owns tokens may burn them.
- Being approved to spend tokens does not give approval to burn tokens.

Concern → non-owner can burn tokens: What are the intended use cases for burning tokens? Is it expected that normal holders burn their own holdings or should only the owner (or another role) have authority to burn tokens held by that authority? Recommend that burn use cases be explicitly documented. *(Outcome: as designed - accepted)*

Handling ETH Transfers

The CAUD token contract prevents the transfer of ETH directly to it due to the default nature of the Vyper language and the fact that no __default__ method exists and none of the existing methods have been marked as @payable. This is best practice for contracts that don't directly manage ETH.

Pausing Contracts

The CAUD token contract has no governance capability for pausing activities in case of unexpected behavior or compromise of accounts with control or access to the contract. Having pause capabilities has both positive and negative aspects. It can potentially mitigate losses if a critical user is compromised or becomes a bad actor but it also increases trust risk for the holders of the asset. For a simple contract like an ERC-20 with clear separation of concerns, pausing is probably undesirable especially if the asset is to be used in other defi services. However adding pause capabilities could be one potential mitigating factor for the risk exposure of the current minting semantics described above. Recommend to fix the minting semantics correctly and not have pause capabilities.

Contract Ownership

The current CAUD 'owner' may transfer ownership to another address (but not ZERO_ADDRESS) by calling the 'transferOwnership' method which generates an 'OwnershipTransfer' event. This removes the current owner from being both the owner and, if applicable, explicitly a minter under the current behavior.

Weakness → owner can be minter: Recommend that minting and ownership be completely isolated per the recommendations about minting above and that the 'transferOwnership' method be updated to help enforce this separation of concerns by not allowing transfer of ownership to an address that is presently already a minter. *(outcome: managed by out of scope process - accepted)*

Biggest Lab, Co. Ltd
Cloudfloat AUD ERC-20 Contract Audit

October 31st, 2022

OSSTMM Background & Terms[1]:

OpSec is a combination of separation and controls. Under OpSec, for a threat to be effective, it must interact either directly or indirectly with the asset. To separate the threat from the asset is to avoid a possible interaction. Therefore it is possible to have total (100%) security if the threat and the asset are completely separated from each other. Otherwise what you have is safety of the asset which is provided by the controls you put on the asset or the degree to which you lessen the impact of the threat.

To have true safety of the assets different types of controls are required. However, controls also may increase the number of interactions within the scope which means more controls are not necessarily better. Therefore it is recommended to use different types of operational controls rather than just more controls. More controls of the same type of operational controls do not provide a defense in depth as access through one is often access through all of that type. This is why it is so important to be able to categorize controls by what they do in operations to be certain of the level of protection provided by them.

To better understand how OpSec can work within an operational environment, it must be reduced to its elements. These elements allow one to quantify the Attack Surface, which is the lack of specific separations and functional controls that exist for that Vector, the direction of the interaction. The reductionist approach resolves to us needing to see security and safety in a new way, one that allows for them to exist independent of risk and fully capable of creating Perfect Security, the exact balance of

1 The following content taken from OSSTMM3 manual p. 20-22. https://www.isecom.org/OSSTMM.3.pdf Copyright 2010, ISECOM. Used with permission.

security and controls with operations and limitations. However, to see security in a new way requires new terminology as well.

Term	Definition
Attack Surface	The lack of specific separations and functional controls that exist for that vector.
Attack Vector	A sub-scope of a vector created in order to approach the security testing of a complex scope in an organized manner. It is based on the divide and conquer algorithm design paradigm that consists in recursively breaking down a problem into two or more sub-problems of the same (or related) type, until these become simple enough to be solved directly.
Controls	Impact and loss reduction controls. The assurance that the physical and information assets as well as the channels themselves are protected from various types of invalid interactions as defined by the channel. For example, insuring the store in the case of fire is a control that does not prevent the inventory from getting damaged or stolen but will pay out equivalent value for the loss. Ten controls have been defined. The first five controls are Class A and control interactions. The five Class B controls are relevant to controlling procedures.
Limitations	This is the current state of perceived and known limits for channels, operations, and controls as verified within the audit. Limitation types are classified by how they interact with security and safety on an operational level. Therefore, opinions as to impact, availability in the wild, difficulty to perform, and complexity are not used to classify them. For example, an old rusted lock used to secure the gates of the store at closing time has an imposed security limitation providing a fraction of the protection strength necessary to delay or withstand an attack. Determining that the lock is old and weak through visual verification is referred to as an identified limitation. Determining it is old and weak by breaking it using 100 kg of force when a successful deterrent requires 1000 kg of force shows a verified limitation. One of its limitations is then classified based on the consequence of the operational action, which in this case is Access.
Operations	Operations are the lack of security one must have to be interactive, useful, public, open, or available. For example, limiting how a person buys goods

	or services from a store over a particular channel, such as one door for going in and out, is a method of security within the store's operations.
Perfect Security	The exact balance of security and controls with operations and limitations.
Porosity	All interactive points, operations, which are categorized as a Visibility, Access, or Trust.
Safety	A form of protection where the threat or its effects are controlled. In order to be safe, the controls must be in place to assure the threat itself or the effects of the threat are minimized to an acceptable level by the asset owner or manager. This manual covers safety as "controls" which are the means to mitigate attacks in an operational or live environment.
Security	A form of protection where a separation is created between the assets and the threat. This includes but is not limited to the elimination of either the asset or the threat. In order to be secure, the asset is removed from the threat or the threat is removed from the asset. This manual covers security from an operational perspective, verifying security measures in an operating or live environment.
RAV	The rav *(risk assessment values)* is a scale measurement of an attack surface, the amount of uncontrolled interactions with a target, which is calculated by the quantitative balance between porosity, limitations, and controls. In this scale, 100 rav (also sometimes shown as 100% rav) is perfect balance and anything less is too few controls and therefore a greater attack surface. More than 100 rav shows more controls than are necessary which itself may be a problem as controls often add interactions within a scope as well as complexity and maintenance issues.
Target	That within the scope that you are attacking, which is comprised of the asset and any protections the asset may have.
Vector	The direction of an interaction.
Vulnerability	One classification of Limitation where a person or process can access, deny access to others, or hide itself or assets within the scope.