CryptoTask ICO Contract Audit

November, 2017

New Alchemy

Introduction

During November of 2017, CryptoTask engaged New Alchemy to audit the smart contracts they created to conduct their ICO. The engagement was technical in nature and focused on identifying security flaws within the source code and design of the contracts.

CryptoTask provided New Alchemy with access to their GitHub repository and a brief explanation of the intended behavior of the contracts. No official documentation on the ICO or details of how it will be presented to potential investors were provided.

Files Audited

The code is at the following GitHub repository:

https://github.com/vkajic/cryptotask/

New Alchemy evaluated commit hash `ed137bec6d198aa19c74028a71fea38834533b54`

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 bugfree status. The audit documentation is for discussion purposes only.

Executive Summary

The audit identified a variety of issues with minor impacts. The most significant findings were either non-exploitable or the result of a trade-off in the usability of the contract. A majority of the code was standard and copied from widely-used and reviewed contracts. As a result, many of the findings pertained to custom portions of code that related to locking an account's tokens and the contract defining the behavior of the Crowdsale itself.

General Discussion

Smart contracts will often contain functionality for the owners to update the logic of contracts post-deployment in order to respond to attacks and discovered bugs. This can lead to a security trade off. The owners, given the ability to respond to flaws in the contract, will also be able to alter the contract's logic in a way that the contract will behave differently than what the investors originally agreed to. CryptoTask chose not to include this functionality and, as a result, cannot update contracts if issues are discovered after the contracts have been deployed.

Minor Issues

Code pattern susceptible to re-entrance:

The `checkGoalReached` function has a code pattern that would result in a viable re-entrance attack if more gas were to be forwarded in the `transfer` function. This function checks that the `crowdsaleClosed` variable is set to `false` and that the `stage` variable is set to one before sending funds to the vaults using the `transfer` function in `forward`. After the transfer goes through, the variables are updated to reflect that the stage has moved on and the crowdsale has been closed. Since the `transfer` function can trigger the vaults' fallback functions, a malicious contract at one of the vault addresses could re-enter the `checkGoalReached` function upon the transfer in order to trick the contract into sending the funds again. The only thing stopping this attack from being viable is the fact that the `transfer` function will only supply 2300 gwei to the fallback function, which is not enough gas to re-enter the function and trigger this vulnerability. CryptoTask should update the `stage` and `crowdsaleClosed` variables prior to sending funds.

If Solidity were to change the behavior of the `transfer` function or if, in future development, CryptoTask were to use a different function to transfer funds or pass more gas to `transfer`, then the vault owners would be able to deplete the contract of its funds prematurely.

Lack of short address attack protections

Users who wish to transfer CTS tokens may be vulnerable to a short address attack if they do not verify the length of the address that they send to the `transfer`, `approve`, or `transferFrom` functions. Due to the way that the EVM works, if a user is coerced into sending an address that is not long enough, the remaining bytes will be filled with the beginning bytes of the next argument and the next argument will be shifted over by the number of missing bytes. This means that an attacker can generate a wallet address ending with "00" and submit their address without the "00" at the end to another user in the hopes that they will fill in the short address and shift the value of the transfer over by the number of bytes missing from the address. As a result, users, particularly automated ones such as exchanges, may be tricked into transferring a much larger number of tokens than they intended. The length of the input data should be checked on all function calls where this attack vector may be relevant.

CryptoTask is aware of this issue and has chosen to not implement protections against this attack, citing the usability tradeoff described in a blog post[1]. After reviewing the blog post, New Alchemy still suggests that some protections be implemented by ensuring a minimum payload length. The concerns in the blog post relating to the payload being potentially padded by another contract are not an issue because longer payloads would be allowed.

Additionally, the case in which a payload may be smaller than the required length does not exist in CryptoTask's existing codebase because those functions are never being called by a subcontract. As a result of these limitations, many token implementations have decided that protections against this sort of attack should be implemented at the level of the exchange. Therefore, many exchanges will already be aware of this issue. These protections should be implemented regardless, because

[1]https://blog.coinfabrik.com/smart-contract-short-address-attack-mitigation-failure/

additional protections can be created at the level of this contract without significant impact on usability

Additionally, the case in which a payload may be smaller than the required length because it is being called by a subcontract function with fewer arguments does not exist in the CryptoTask's existing codebase. Many token implementations have decided that protections against this sort of attack should be implemented at the level of the exchange as a result of these limitations, and so many exchanges will already be aware of this issue. Regardless, since additional protections can be created at the level of this contract without significant impact on usability, these protections should be implemented.

Two-step owner transfers

Ownership transfers are currently a single step process in which the owner simply provides the address of the new owner. This sort of behavior can be dangerous because a mistake when entering the address could result in a contract without an owner. The preferred behavior is to require the new owner to approve the transfer of ownership to finalize the change. Due to the fact that the owner only has the privileges to end the presale early, having an ownerless contract would be fairly insignificant.

Minimum purchase size

Users cannot purchase less than 100 tokens in a single transaction during the ICO or less than 200 tokens during the presale period. This requirement is not mentioned in any of the limited documentation provided to New Alchemy but CryptoTask claims that this will be known to investors.

Double-spend attack

The `StandardToken` contract is susceptible to the double-spend attack described in https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729. In this attack, an attacker attempts to spend their allowance between the time when the victim user checks the attacker's allowance and when the victim's transaction (that adjusts the attacker's allowance) is mined. This means that the potential exists for attackers to spend the sum of the existing allowance and the desired allowance every time a user approves an allowance for them. As an alternative to this functionality, CryptoTask has provided the `increaseApproval` and `decreaseApproval` functions, which cannot be taken advantage of in this way. In order to remain compliant with the ERC20 standard, the contract still must expose the `approve` function.

As a way of effectively mitigating the potential for a double-spend attack on the `approve` function, contracts often require that investors set their allowance to zero before changing it to another value. CryptoTask apparently realizes this, and has provided documentation suggesting that users set their allowance to zero before setting it to the desired value. This behavior, however, cannot be enforced by the contract itself because it must remain backwards-compatible to meet the official ERC20 token standard[2]. As a result, users are able to call the `approve` function in an unsafe way.

CryptoTask has done all that they can to prevent this attack while still remaining compliant with the ERC20 token standard.

Line by line comments

BasicToken.sol

Line 21: `transfer`

A pattern in which functions should only execute if the tokens of a particular address are not locked occurs frequently in this codebase. CryptoTask should consider creating a modifier in the `BasicToken` contract, which will require that an address passed as an argument is not locked. This refactoring would simplify the code and make it simpler to audit the codebase by observing what modifiers are applied to a function at a glance rather than looking into the body of the function to see what logic exists and whether it was implemented correctly each time. It should be noted that this seems to be non-trivial to implement at first due to the fact that functions sometimes need to check multiple addresses. However, it actually would not be very difficult as arguments can be passed to modifiers and the same modifier can be invoked multiple times with different arguments.

Crowdsale.sol

Line 44 `FundTransfer` event:

Consider indexing this event by the address of the backer for easier filtering.

Line 71 fallback function:

The `throw` command is deprecated and should be replaced with the `revert` function.

Line 123(`closePresale` function), line 191 (`countVotes` function):

On these lines, the pattern `now.sub(startTime) < duration` could be more readable as `now < startTime.add(duration)` as it more intuitively reflects the psuedocode “if now is less than `duration` minutes past the `startTime`”.

[2]https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md#approve

SafeMath.sol

Lines 9(`mul` function), 15(`div` function), 22(`sub` function), 27(`add` function):

These functions could be marked as `pure` as they do not access storage in any way.