eMusic Audit

EMU Token Smart Contracts

April 15, 2019

Prepared For:

Nir Peled | Triplay

nirp@triplay-inc.com

Prepared By:

Alex Towle | Authio

alext@authio.org

Contents

1. **Summary** 2
2. **Scope** 3
3. **System Overview** 3
4. **Findings** 5
5. **Recommendations** 7

1. Summary

This audit consisted of a review of eMusic EMU token. The EMU token is a token that prevents users from transferring their tokens to accounts that are not designated as “allowed receivers” by the owner of the contract. After these restrictions are lifted, the token functions as a standard ERC20 token. We found their implementation to contain no serious security issues other than two common security issues for ERC20 tokens. This contract is susceptible to the standard ERC20 racing condition, although it does implement the functions `increaseApproval` and `decreaseApproval`, which Authio recommends be used instead of `approve` as it mitigates the issues to an extent. Additionally, the token supply and the whitelists of the token are governed by a single address, so the contract is susceptible to vulnerabilities if there is not a key management strategy in place.

eMusic has produced a great implementation, and upon completion of the mitigations outlined in the Findings and Recommendations sections, the token will be ready to be deployed to the mainnet.

Disclaimer: This document reflects the understanding of security flaws and vulnerabilities as they are known to the Authio team, and as they relate to the reviewed project. This document makes no statements on the viability of the project, or the safety of its contracts. This audit is not intended to represent investment advice and should not be taken as such.

2. Scope

The scope of this audit consisted of the `contracts` directory of the following commit of the eMusic smart contract repository: https://github.com/emusic-eng/eMusicSmartContracts/tree/9feff3487759abaa4188d7554f3d8dd24f8dc8a9.

3. System Overview

3.1 Records

The EMU token follows the ERC20 standard and therefore maintains the following public records:

- The boolean representing whether or not transfers are locked
- The duration that regular addresses must wait when transfers are being locked
- The token balances of owner addresses.
- The allowance of spender address on behalf of owner addresses
- The total supply of tokens.

The contract additionally maintains the following records:

- **Owner address**
    - This address is given the total supply of the token and several permissions at construction.
- **Allowed sender list**
    - Addresses in this whitelist can transfer tokens without waiting.
- **Receiver whitelist**
    - Addresses in this whitelist can receive tokens without waiting.
- **Unlock dates**
    - This mapping encodes the unlock dates of addresses. These unlock dates determine when regular users become able to make transfers to addresses that are not allowed receivers.

3.2 Deployment

When the token is deployed, the sender of the initial transaction is made the `owner` address, is added to the `sender` whitelist, and is minted the total supply of tokens of the contract. Additionally, `_lockDuration` will be set to 40 days.

3.3 Configuration

Users can use several wallets to buy goods and services on emusic.com. These wallet addresses should be added to the receiver whitelist so that addresses that receive tokens can immediately transfer funds to the eMusic wallets. This ensures that users will be able to purchase things from eMusic without needing to wait to transfer funds.

3.4 Disbursement

Since the owner of the token is on the sender whitelist after the **Deployment** phase, the owner can transfer tokens immediately. Using this ability, the owner adds several tokens to the sender whitelist. One of these addresses may be a wallet that can airdrop tokens to other Ethereum addresses.

3.5 Usage

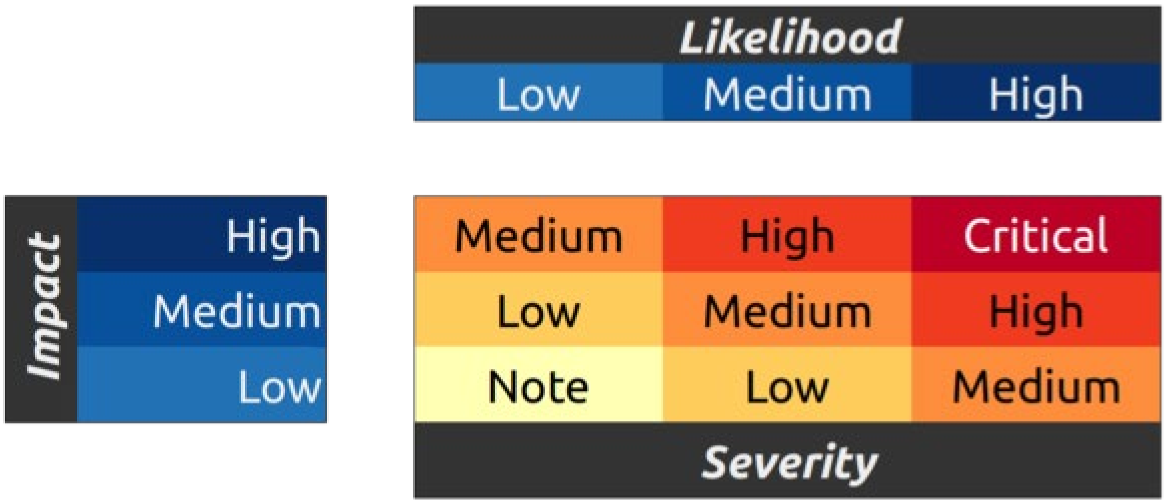
Users that receive tokens from an address on the allowed sender list must wait for 40 days before transferring their tokens. The owner can change the unlock date of a user to an arbitrary value, so the 40-day limit is not the only possible wait time. Once a user's unlock date has passed, the user can transfer tokens to any address and the recipient's unlock date is set to the block timestamp of the transfer.

The owner has the ability to unlock transfers, which allows any address that possesses tokens to transfer tokens at any time and to any address. Once transfers are unlocked, the owner cannot relock them.

4. Findings

The findings section of this report is for explicit security problems with quantifiable possible effects. For this reason, each finding is ranked using a variant of the OWASP Risk Rating which rates each vulnerability based on how much damage it can do versus the likelihood of the damage occurring.

The following chart details the classification:



		Likelihood		
		Low	Medium	High
Impact	High	Medium	High	Critical
	Medium	Low	Medium	High
	Low	Note	Low	Medium
			Severity	

4.1 Medium Severity:

4.1.1 ERC20 Racing Condition: Since this contract adheres to the ERC20 standard, it is susceptible to a well-known racing condition that involves the functions `approve` and `transfer`. If the approval is being changed from a value of `N` to `M`, then a malicious spender can take `N + M` tokens. Suppose a token owner has already approved a spender to spend a value of `N` tokens. This exploit works as followed:

1. The token owner broadcasts a transaction `T1` to the network that changes the approval value of the sender to `M`.
2. The malicious spender notices `T1` and broadcasts a transaction `T2` to the network to `transferFrom` `N` tokens from the owner's address to the

spender's address. The spender sets the gas price of `T2` to a value that is significantly higher than the gas price of `T1`.

3. There is a nonzero probability that `T2` is executed before `T1`. If this occurs, then `N` tokens will be transferred to the spender and the spender's approval will be set to `M`.
4. If step 3 was successful, the spender sender broadcasts a final transaction `T3` to the network with a high gas price before the owner notices. `T3` will `transferFrom` `M` tokens from the owner's address to the sender's address, thereby increasing the total tokens transferred to the spender to `N + M` if the transaction succeeds. `T3` will succeed as long as the owner does not notice this transaction and broadcast a transaction `T4` to change the spender's approval value to `V`, where `V < M`.

The OpenZeppelin `ERC20` contract used by the `Lockable` contract implements the functions `increaseApproval` and `decreaseApproval` which a commonly recommended mitigation for the racing condition. These functions limit the profitability of exploiting the racing conditions, but the remaining existence of the `approve` function means that this vulnerability still has the potential to be exploited.

Reference for the `transferFrom` racing condition:
https://docs.google.com/document/d/1YLPtQxZu1UAvO9cZ1O2RPXBbT0mooh4DYKjA_jp-RLM/edit#heading=h.teplhora0j44

4.1.2 Single Key Whitelist and Token Supply Control: The `owner` address of the EMU token controls the sender and receiver whitelists and also starts by possessing the total supply of tokens in the contract. This centralization is a convenient way to distribute tokens initially, and it is currently cryptographically infeasible to find the `owner`'s private key using the public key; however, practically speaking there should be a key management strategy in place to prevent the private keys from being discovered by another party. Exploits involving the theft of private keys are not unprecedented, and there have been cases where keys were stored or transmitted in such a way as to expose them. In fact, at the ETHDenver hackathon, the WhiteBlock team managed to get private keys of BuffiDai wallets that were transmitted over a public network.

The likelihood of issues arising from the centralized control of the token supply and whitelists can be reduced by constructing a key management strategy that:

1. Outlines how the keys will be stored. Some options for key storage include hardware wallets, cold wallets, and paper wallets among others.
2. Provides a roadmap for how the keys will be used, where they will be used, and who will use them.

4.2 Notes

ERC20 Compatibility: The function `DECIMALS` in the EMU contract breaks the extended ERC20 Standard.

We recommend that the function name be changed to `decimals` so that the EMU token can be fully ERC20 compatible. This change will ensure that contracts that use the `decimals` function when interacting with ERC20 contracts will be able to integrate with the EMU token.

5. Recommendations

The following recommendations do not present findings of security risks that the EMU token's stakeholders should know about. Instead, these recommendations serve to improve the gas efficiency of the contract, to inform users about the limitations of some of the constructs used in the contract, and to suggest changes that we feel will improve the quality of the audited codebase.

Getters with Modifiers: The getters involving information from the sender and receiver whitelists (`allowedReceiverAddresses`, `allowedSenderAddresses`, `myUnlockDate`, and `unlockDateOf`) are all modified by `onlyOwner`, `whenLockingTransfers`, or both. While these modifiers will limit the ability of other smart contracts to call these functions, non-contract Ethereum accounts will still be able to access this information as Ethereum is a completely public blockchain. In fact, there is even a `web3` function returns the value in the storage of an Ethereum account at an arbitrary value. With this in mind, while these modifiers can make accessing this information more inconvenient on-chain, they cannot prevent the data from being submitted to a smart contract by a human user or an oracle service.

We recommend that the getters' modifiers be removed as this will make the Lockable contracts code easier to reason about. This change is a low priority change as removing the modifiers will only slightly improve the deployment cost of the contract and the modifiers will likely not present a serious problem for users, but it is still worth mentioning the limitations of the modifiers' ability to restrict the flow of the state of the smart contract off-chain and potentially back on-chain.

Constants: The values in the EMU contract `name`, `symbol`, and `LOCK_DURATION_IN_DAYS` can all be made constants as they are never set outside of the constructor.

We recommend that these values be changed to constants will reduce the gas price of using these values slightly.