Cryptocurrency Token: An Overview

Mahesh Shirole, Maneesh Darisi, and Sunil Bhirud

Abstract With the advent of blockchain, the trustless transactions between cross-border parties becomes easy. The profusion of eclectic tokens coming into the cryptocurrency world, one primary requirement is to enable robust and secure exchange between different varieties of tokens. With the rise of ICOs into the cryptocurrency market, the funding of research-based projects has become meteoric in nature. The dearth of knowledge about these different varieties of tokens is blocking amateur developers to unlock the true potential of the blockchain technology. There are myriad tokens, which are either fungible or non-fungible tokens that are used to represent different types of assets and research projects. This paper mainly focuses to provide an overview of tokens and comparative analysis for token standards available in the present cryptocurrency world.

Keywords Blockchain · Cryptocurrencies · Token · ICO

1 Introduction

Cryptocurrencies are digital currencies that are still in their embryonic stage and have been gaining a lot of attention worldwide. Cryptocurrency is neither government-issued nor government-regulated currency. Cryptocurrency can be used as a medium of exchange and to perform monetary transactions, in the same way as the printable currency/bills can be used. Although the number of blockchain use cases is augmenting as the time progresses, the lack of understanding of this new technology and tokens is restricting the use of cryptocurrencies widely. The first ever cryptocur-

M. Shirole · M. Darisi (✉) · S. Bhirud
CE & IT Department, Veermata Jijabai Technological Institute, Mumbai 400019, India
e-mail: mdarisi_b15@it.vjti.ac.in

M. Shirole
e-mail: mrshirole@it.vjti.ac.in

S. Bhirud
e-mail: sgbhirud@ce.vjti.ac.in

D. Patel et al. (eds.), *IC-BCT 2019*, Blockchain Technologies,
https://doi.org/10.1007/978-981-15-4542-9_12

rency bitcoin [13] was created by Satoshi Nakamoto introduced in the year 2009. Subsequently, the development of blockchain technology has lead to the emergence of Ethereum [1]. Ever since advent of these cryptocurrencies in the market, it has attracted several potent investors. Roughly, there are about 2116 cryptocurrencies in the cryptoworld and keep augmenting regularly. These cryptocurrencies have a market capital of $119,068,338,608 [9] by the end of 2018. They have changed the way how cross-border transactions take place.

Initial coin offering (ICO) is the means of crowdfunding and thus leads to the creation of a new cryptocurrency. ICOs sell tokens/coins to the different stakeholders in exchange of legal tender or cryptocurrencies. It uses several cryptocurrency protocols to a create token on the top of the existing blockchain. ICOs have exponentially increased in the last two years with launch of ERC20 open standard token protocol. In last year 2018, total 686 new ICOs are launched, with a peak of 144 in May, raising total of $21,498,711,596 fund [2].

DApps used to define their own token standard and implementation for their private currency. With the launch of ERC20 token standard describing the rules and standards for cryptocurrency tokens, all the DApps and start-ups are using ERC20 token standard. It helped to uniformly understand tokens, its format and a way to interact with tokens. As of January 16, 2019, a total of 162,906 total token contracts are found on Etherscan [9].

In this paper, token standards are explored on different aspects such as standard, attack vectors, use cases of the attacks, improvements to ERC20 standard and other secure token proposals. This paper is organized as follows: Sect. 2 discusses the cryptocurrency tokens. Section 3 discusses token attacks and their possible solutions. A brief comparison of tokens is given in Sect. 4. Finally, Sect. 5 concludes the paper.

2 Cryptocurrency Tokens

Cryptocurrencies can be classified based on their: (a) **usage**: fungible and non-fungible, (b) **type of implementation**: currency token (payment), utility token and security token. Fungibility is one of the essential characteristics of the currency. It can be used to represent anything that is interchangeable in the real world. *Fungible tokens* possess currency-like properties rather than unique and valuable assets. These tokens are interchangeable, uniform across platforms and are divisible into smaller units. All fungible tokens are based on ERC20 standard. *Non-fungible tokens* enable people and organizations to think beyond the cryptocurrencies, such as IDs and certificates. Non-fungible tokens are uniquely identifiable during trade, i.e., interaction and circulation. These tokens are non-interchangeable, unique in nature and non-divisible into smaller units. ERC721 standard is also used for non-fungible tokens on Ethereum blockchain.

Inherently, every currency token is associated with its own unique blockchain. Blockchain platforms serve as a medium for payment of goods and services. Currency tokens are used to perform monetary transactions using digital currencies rather than fiat currencies. Bitcoin and ether are paragons of cryptocurrencies and have their own blockchain platform.

2.1 Utility Tokens

Utility tokens are digital assets, which are built to support the structure of investor's payment mechanism. Utility tokens are handled using DApps. The proliferation of these tokens is enabling blockchain to affect various spectrums of the industry. Utility tokens are provided by businesses; the holder of the utility token gets access to different functions provided by businesses for trading tokens. Most of the tokens available on the blockchain are utility tokens.

ERC20. ERC20 is an open standard protocol that defines the software interface to implement tokens in Ethereum ecosystem. With ERC20 token standard protocol specification, all ICO developers now implement their own code with same method names and their arguments. Thus, improving the interoperability of different tokens implemented by different developers. This reduced the complexity of implementation of tokens, uniform interface and increased the rate of liquidity of the different tokens. ERC20 token standard interface includes six methods: $totalSupply$, $balanceOf$, $transfer$, $transferFrom$, $approve$ and $allowance$ and two events: $Transfer$ and $Approval$. In short, the standard facilitates to share, transfer, exchange and trade tokens seamlessly through cryptoworld.

ERC223. ERC223 [6] has suggested an improvement to ERC20 token standard. ERC20 suffers token losses due to $transfer$ function in the contract, which does not support token receiving and handling mechanism. The total loss estimated as on 27 Dec 2017 is \$3,000,000 [9]. ERC223 token standard is backward compatible with ERC20 as it uses same interface. However, it requires contracts to implement the $tokenFallBack$ function.

$$function\ tokenFallback(address\ \_from, uint\ \_value, bytes\ \_data)$$

ERC223 is applicable to new contracts rather than old deployed contracts.

ERC777. ERC777 [8] token standard allows a new way to interact with contracts with the help of ERC820. ERC777 is backward compatible with earlier token standards and thus mitigates the problem of ERC223 to modify the contract. ERC777 takes support of ERC820 standard, which is a contract registry that will verify whether a contract is token compatible or not. In case if ERC777 token contract is not registered or compatible, then the transaction will raise an exception, thus preventing loss of tokens. The only problem with ERC777 is that it uses a central registry for smart contracts lookup.

ERC721. ERC721 [7] is a token standard that defines an interface to allow non-fungible tokens to be managed, owned and traded by a smart contract. It defines functions: $name$, $symbol$, $totalSupply$, $balanceOf$, $ownerOf$, $approve$, $takeOwnership$, $tokenOfOwnerByIndex$, $transfer$ and $tokenMetadata$. It defines two events: $Transfer$ and $Approval$. The $takeOwnership$ and $transfer$ functions define how the contract will handle token ownership and how ownership can be transferred. The function $tokenMetadata$ makes token non-fungible by its unique set of attributes. ERC721 does not mandate implementation of a token metadata or restrict addition of extra functions.

2.2 Security Tokens

Security tokens need to follow myriad federation rules and regulations; hence, they are complex. They are the type of assets, which assure positive ROI on their holding. Such returns are guaranteed by the platform itself or the company, which had launched the security token. A security token shares many of the characteristics of both fungible and non-fungible tokens.

ERC1400. ERC1400 [3] is a simple restricted token standard developed for corporate governance and banking considering securities laws. It is fully open source ensuring security, quality and interoperability of tokens. ERC1400 tokens are partially fungible. One ERC1400 token issued by one entity may not be exchangeable with another, because these tokens have different properties and a group of owners. ERC1400 is an umbrella standard that incorporates ERC1594 [5] (core functionality), ERC1410 [4] (partially fungible tokens), ERC1643 (document management) and ERC1644 (controller token operation) with some additional constraints to ensure these standards interoperate in a consistent manner.

R-Token. R-token [12] is a permissioned token allowing token transfer to occur only if they are approved by an on-chain regulator service. R-tokens extends ERC-20 tokens for regulated securities. Regulator services can be configured to meet security regulations such as KYC policies and anti-money laundering.

3 Token Attacks

Blockchain is immutable; hence, the contracts which are deployed on blockchain are immutable. Majority of contracts are written in solidity language; therefore, one must understand what are the common attacks in solidity. These attacks have caused huge losses due to loopholes in the contracts.

3.1 Approve Attack

Although 90% of the tokens present in the crypto world are compliant with the ERC20 standards, one of the significant flaws in its interface is in the definition of the $approve$ and $transferFrom$ methods. These methods can be exploited to withdraw

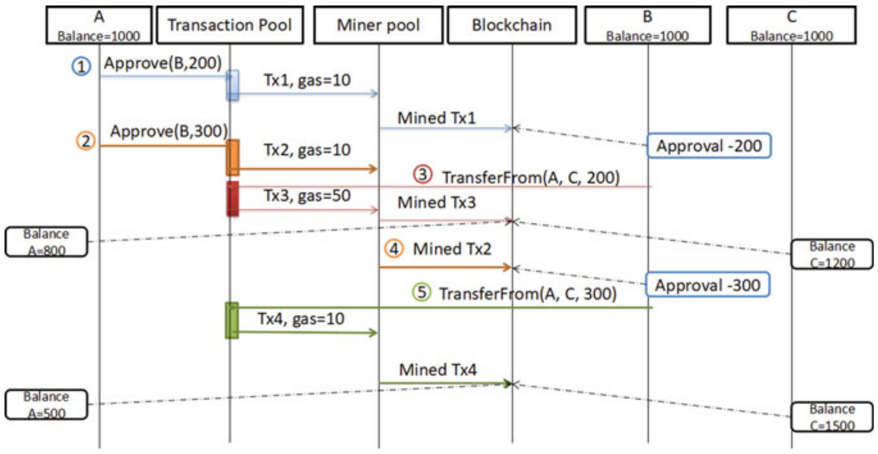


Fig. 1 Illustration of approve transfer attack using sequence diagram

tokens more than given allowance if approvals are given in succession. Approve function permits the given address to withdraw token amount from message sender token balance up to the specified value in single or multiple withdraws. Consider a scenario wherein there are two parties/accounts A and B, where account B also controls account C. It is assumed that all accounts initial balance is 1000 ether and the balance of account A is greater than or equal to $X + Y$ ether, i.e., 200+300 in Fig. 1. The illustration of the attack vector use case scenario is as follows referring Fig. 1.

STEP 1. Account A gives approval of X ether from his wallet to account B.

STEP 2. Account A decides to change the amount of approval from X ether to Y ether for account B and sends Y as the argument in the $Approve$ function.

STEP 3. Account B notices A's second transaction and before it gets mined, B sends the allowance of X ether that it had approved from A to C using $transferFrom$ function with high gas value to prioritize transfer.

STEP 4. If B's $transferFrom$ call gets executed before A's approve transaction, then B will get the ability to transfer another Y tokens to C.

STEP 5. A's attempt to change the allowance from X to Y has lead B to transfer $X + Y$ to C, whereas A never wanted to transfer $X + Y$ tokens to B. **Solution.** A simple change to the present interface may assist in making the ERC20 less error prone. Convert the approve function to a three-argument function from a two-argument function to prevent the above attack.

```
function approve(address spenderAddresss, uint256 currentValueOfAllowance,
uint256 ChangedValueOfAllowanceValue) returns (bool success)
```

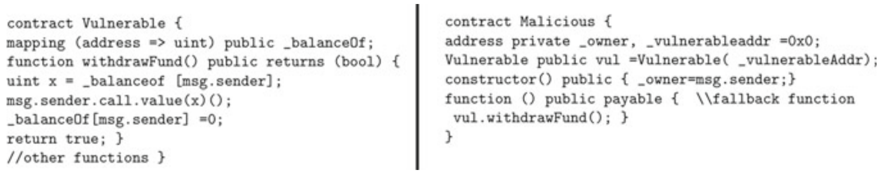
If current allowance for $spenderAddress$ is equal to $currentValueOfAllowance$, then overwrite it with $ChangedValueOfAllowanceValue$ and return true, otherwise return false. Accordingly, events $Transfer$ and $Approve$ can be changed.

3.2 Overflow Exploit Attack

The classical arithmetic integer overflow problem can be exploited to transfer large amount of tokens. This bug was noticed by PeckShield on April 22nd 2018. He found fallacious transaction which involved two large BEC token transactions. The reason was arithmetic overflow in statements like
$balances[msg.sender] += msg.value;$
Solution. OpenZeppelin [11] library has provided a transparent solution to counter this attack vector. OpenZeppelin has provided a solidity file known as $SafeMath.sol$, which provides several checks to avoid any overflow during basic arithmetic operations.

3.3 Reentrancy Attack

A non-recursive function of a contract should not enter in the same function before termination of the function. In reentrancy attack, an attacker tries to reenter the calling function of a contract with the fallback mechanism in solidity. In Ethereum, *call* function is used to transfer a value/data or to execute a function of same contract or another contract. The *call* function starts code execution of function and spends available gas for execution. It makes code vulnerable to reentrancy attack. As there is no gas limit for the call function, the fallback function in *call* function can run as long as it exhausts all gas allocated for that function or balance of the account. To understand this attack, consider example as discussed in [10]. Consider a wallet contract, named $Vulnerable$, shown in the following code in Fig. 2. Its statement $msg.sender.call.value(x)()$ will cause reentrancy attack, as it invokes the $message.sender's$ fallback function in order to send his balance of 'x' wei to the sender. A malicious user contract named $Malicious$ is as shown in the code in Fig. 2, where the contract address is initialized with $\_owner$ and another address is vulnerable wallet's address. When the malicious user withdraws balance from $Vulnerable$



```
contract Vulnerable {
mapping (address => uint) public _balanceOf;
function withdrawFund() public returns (bool) {
uint x = _balanceof [msg.sender];
msg.sender.call.value(x)();
_balanceOf[msg.sender] =0;
return true; }
//other functions }
```

```
contract Malicious {
address private _owner, _vulnerableaddr =0x0;
Vulnerable public vul =Vulnerable( _vulnerableAddr);
constructor() public { _owner=msg.sender;}
function () public payable {  \\fallback function
 vul.withdrawFund(); }
}
```

Fig. 2 A sample code to illustrate reenterancy attack

wallet using *Malicious* contract, then wallet calls users contract fallback function and in turn calls fallback function which executes *withdrawFund* function of the wallet contract repeatedly until it fails due to gas limit or balance.
Solution. To mitigate reenterancy attack avoid using *call* function. The *transfer*() and *send*() are safe against reentrancy attack since they limit code execution to 2300 gas units.

4 Utility Tokens Comparison

In this section, a comparative analysis of the utility tokens is presented based on fungibility, backward compatibility, token sale delegation and contract registration. The comparative analysis is shown in Table 1.

Fallacious transactions to anonymous addresses can result in the loss of ether; this issue is resolved by ERC223 (assembly code), ERC777 and ERC721 using a centralized registry ERC 820. ERC777 and ERC721 are backward compatible with the ERC20 standard and thus provide a easy mechanism for upgrading the presently deployed contracts. The major aim of using upgraded token standard is to avoid the loopholes in the present contract and to abate the present gas consumption rate. Except ERC721, other token standards are fungible.

Table 1 Comparison of utility tokens

	ERC20	ERC223	ERC777	ERC721
Fungible	Yes	Yes	Yes	No
Verify contract address	No	Yes	Yes	Yes
Back compatibility with ERC20	–	No	Yes	Yes
Delegate token sale	Yes	Yes	Yes	Yes
Usage ERC820 for contract registration	No	No	Yes	Yes

5 Conclusion

Sharp increase in the number of ICOs in the last two years had lead to the standardization of tokens. Cryptocurrency tokens are evolving and growing community effort to set interoperable standards. This paper discusses a list of representative token standards, their possible attacks and possible solution for the same. Possible loss due to faulty and incompatible token implementations can be decreased using appropriate mechanisms as discussed in this paper.

References

1. Buterin V (2013) Ethereum white paper: a next generation smart contract & decentralized application platform. https://doi.org/10.5663/aps.v1i1.10138
2. Coinschedule: https://www.coinschedule.com/stats.html. Last accessed Jan 2019
3. ERC1400: https://github.com/ethereum/eips/issues/1400
4. ERC1410: https://github.com/ethereum/eips/issues/1410
5. ERC1594: https://github.com/ethereum/eips/issues/1594
6. ERC223: https://github.com/ethereum/eips/issues/223
7. ERC721: Non-fungible token standard. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721
8. ERC777: https://eips.ethereum.org/eips/eip-777
9. Etherscan: Ethereum Blockchain Explorer. https://etherscan.io/
10. Li X, Jiang P, Chen T, Luo X, Wen Q (2017) A survey on the security of blockchain systems. https://doi.org/10.1016/j.future.2017.08.020
11. OpenZeppelin: https://github.com/openzeppelin/openzeppelin-solidity (2018)
12. Remeika, B., Amano, A., Sacks, D.: The Regulated Token (R-Token) Standard (2018). 10.1007/BF02162388
13. Satoshi, N.: Bitcoin: A Peer-to-Peer Electronic cash system. Bitcoin (2008)