Report number	EBS-AQ-2021-110301
Total pages	A total of 14 pages

EBSecurity

Smart contract security audit report

Eastern Blockchain Security Technology Testing Center

2021

Contract name	bitstore—token
Compiler version	Solidity ^0.8.2
Total number of lines of contract code	13
File hash (SHA256)	a4c65421d174fcd5146bbb32646e9eecd74a43e2fbd0301faf4ea14282fda38d
Audit start time	2021.11.02
Audit end time	2021.11.03
Audit team	Eastern Blockchain Security Technology Testing Center
Audit results	Pass

Disclaimer

This audit is only conducted on the audit items specified in this report and the scope of audit types given in the result table. Other unknown security vulnerabilities are not within the scope of this audit, and it makes no representation or warranty as to the regulatory regime of its business model or the applicability of any other relevant application. Eastern Blockchain Security Technology Testing Center(EBS) only issues this report based on existing or occurring attacks or vulnerabilities before the issuance of this report, and does not assume any responsibility for new attacks or vulnerabilities that exist or occur in the future. The security audit analysis and other content made in this report are only based on the documents and materials provided by the information provider to EBS as of the issuance of this report. If the information is not provided or the information provided is inconsistent with the actual situation or the documents and materials provided are in this report if any changes occur after the issuance, EBS will not bear any responsibility for the losses and adverse effects caused thereby.

The final interpretation right of this statement belongs to Eastern Blockchain Security Technology Testing Center.

Tel: (86) + 13574843710

Mail: chenxin@eastsec.org.cn

1. Audit overview

Code auditing works by analyzing the source code of the current contract, understanding the contract structure, checking the relationship between its various modules and functions, authorization verification, etc. from the contract structure; checking its code security and design risk from the security perspective. Under the circumstance of clarifying the current security status and requirements, it is of great significance to the construction of contract security norms. The source code audit work uses certain programming specifications and standards to review the source code of the contract in terms of structure, vulnerability, and defects, so as to find the security flaws in the current smart contract and the normative flaws of the code.

- **Audit purpose**

This source code audit work is to review the source code of each module of the current system to check the code security and design risks that may be caused by the programming of the code.

- **Audit scope**

According to the code given by the user, the contract code is checked for vulnerabilities, defects, and structure. Determine key inspection modules and important logic, find possible risks and provide feasible solutions.

- **Audit method**

The security of the contract code is checked through white box (code audit). The method used in white box testing is tool review + manual confirmation + manual code extraction inspection. According to the design purpose, the target code security and design risk are checked as well as Structural problems.

This code audit is divided into three stages:

1). Information Collection

In this stage, the code auditor will analyze the structure and functional modules of the audit contract, and collect contract-related information in the block explorer.

2). Contract code standards audit

In this stage, code auditors mainly conduct compliance checks on the contract source code through manual audits and using code specification checking tools , with the purpose of improving the code quality and making it more in line with the requirements of the coding specification.

3). Contract Code vulnerability audit

This stage, the code auditors mainly through artificial audits and the use of contract vulnerability checks for vulnerabilities and security flaws contract of source code tool inspection, the purpose is to find loopholes in the code .

2. Audit conclusion

The use of manual review of the way bitstore—token code specifications, safety intelligent contracts Liang aspects of a comprehensive security audit. After auditing, the bitstore—token smart contract passed all inspection items, and the audit result was passed.

3. Audit items and results

Audit categories	Audit subcategory	Audit results
Contract code specification audit	Compiler version security audit	Pass
	Redundant code audit	Pass
	Constructor out of control audit	Pass
	Inconsistent audit of function description and function code	Pass
	require/assert usage audit	Pass
	Gas consumption control audit	Pass
	Audit the use of the fallback function	Pass
	Visibility specification audit	Pass
	Deprecated item audit	Pass
Contract code	Pseudo-random number generation audit	Pass

vulnerability audit	Low-level function call risk audit	Pass
	Uninitialized storage pointer	Pass
	External data source risk audit	Pass
	Re-entry attack audit	Pass
	Transaction sequence relies on audit	Pass
	Authority audit	Pass
	Floating point and precision audit	Pass
	Arithmetic overflow audit	Pass
	Dangerous function usage audit	Pass
	No address is empty judgment audit	Pass
	Short address attack audit	Pass
	Self-destruct attack audit	Pass
	Balance judgment audit	Pass
	Variable coverage audit	Pass
	Global variable use security audit	Pass
	Return value security audit	Pass
	Denial of service attack audit	Pass

4. Audit details

4.1 Contract code standards audit

- **Compiler version security audit**

Canonical	Compiler version security

name	
Risk description	The lower version of the compiler may cause its own compilation problems and various known security problems .
Audit results	Pass

- Redundant code audit

Canonical name	Redundant code
Risk description	Redundant code in smart contracts (such as assigned variables that are not used, etc.) will reduce the readability of the code, and may consume additional storage space and execution time , and even consume more gas to deploy the contract .
Audit results	Pass

- Constructor out of control audit

Canonical name	Constructor out of control
Risk description	Constructors are special functions that often perform key authority tasks when initializing contracts. In S olidity v0.4.22 before constructor is defined as a function of the same name where the contract. Therefore, if the contract name changes during the development process, but the constructor function name does not change, it will become a normal callable function. This can (and has) caused some contracts to be hacked .
Audit results	Pass

- The function description is inconsistent with the function code audit

Canonical name	The function description is inconsistent with the function code
Risk description	Inconsistent function code and comments or inconsistent function code and requirements can easily confuse others .
Audit results	Pass

- require/assert use audit

Canonical	require/assert use

name	
Risk description	Solidity uses state recovery exceptions to handle errors. This mechanism will undo all changes made to the state in the current call (and all its sub-calls) and flag the caller of errors. The functions assert and require can be used to check conditions and throw exceptions when the conditions are not met. The assert function can only be used to test for internal errors and check non-variables. The require function is used to confirm the validity of conditions, such as whether input variables or contract state variables meet the conditions, or to verify the return value of an external contract call .
Audit results	Pass

- Gas consumption control audit

Canonical name	gas consumption control
Risk description	When the gas is insufficient, the code execution will throw an out of gas exception and undo all state changes , causing the function to fail to execute all the time .
Audit results	Pass

- Audit the use of the fallback function

Canonical name	Use of the fallback function
Risk description	In a smart contract, there can be a function that has no function name, no parameters, and no return value, that is, a fallback function. A contract is not defined fallback function, if the receiving ether, will trigger an exception, and the return of ether (Solidity v0.4.0 start). Therefore, in order for the contract to receive ether, it must implement a fallback function. This function will be triggered in the following situations: 1 , if calling this contract, there is no match on any one function. Then, the default fallback function will be called. 2 , when the contract is received ether (without any other data), the function will be executed. Note that the execution of the fallback function will consume gas.
Audit results	Pass

- Visibility standard audit

Canonical name	Visibility Specification

Risk description	public : State variables are of public type by default, can be inherited, and can be called externally and internally internal : When the internal attribute is added to the state variable, it can still be inherited. The internal attribute can only be called by methods in the contract, and cannot be called directly from the outside. external : The state variable does not have an external attribute, but the function does. When the external attribute is added to the function, it means that the contract can only be called externally, not internally. If you want the contract to be called internally, you need to use the following this. function .
Audit results	Pass

- Deprecated item audit

Canonical name	Deprecated items
Risk description	Solidity in some keywords have been deprecated new version of the compiler, such as throw, years, etc., may exist risks .
Audit results	Pass

4.2 Contract code vulnerability audit

- Pseudo-random number generation audit

Canonical name	Pseudo-random number generation
Risk description	Intelligent contracts may use the random number, the S when at olidity common factor is generated as a random block with a block of information, but such use is unsafe, block information can be controlled or miners attacker in the transaction Obtained, this kind of random number is predictable or collidable to a certain extent. A typical example is that the airdrop random number of fomo3d can be collided.
Audit results	Pass

- Low- level function call risk

Canonical name	Low-level function call
Risk description	In Solidity, there are many methods to perform external calls. The transfer of Ether to an external account is usually done through the transfer() method. However, the send() function can also be used, and for more external calls, the CALL opcode can be used directly in Solidity. The call() and send() functions return a boolean value to indicate

	whether the call succeeded or failed. Therefore, these functions have a simple warning that if the external call fails (initialization of call() or send() fails, instead of call() or send() returning false), the transaction executing these functions will not be rolled back . When the return value is not checked, a common trap occurs, and the developer expects a rollback operation.
Audit results	Pass

- **Uninitialized storage pointer audit**

Canonical name	Uninitialized storage pointer
Risk description	EVM not only uses storage to store, but also uses memory to store. Uninitialized local storage variables may point to other accidental storage variables in the contract, leading to intentional (that is, the developer deliberately put them there for attack) or unintentional vulnerabilities.
Audit results	Pass

- **Risk audit of external data sources**

Canonical name	External data source risk
Risk description	In a state where all blockchain nodes execute the same piece of code in parallel and completely independently, to ensure the robustness and reliability of the execution, the code of the smart contract itself must have a very high degree of certainty. This requires that the code of the smart contract must produce exactly the same results on each execution node. However, when a smart contract calls an external web service or database, it greatly increases the probability that the same piece of smart contract code will produce different results when it is executed repeatedly and independently. Because this will cause the blockchain nodes to be unable to reach a reliable consensus, such a design will cause chaos in the system.
Audit results	Pass

- **Re-entry attack audit**

Canonical name	Reentry attack
Risk description	The vulnerability is mainly because the smart contract calls an unknown contract address. The attacker can carefully construct a smart contract and add malicious code to the callback function. When the smart contract sends Ether to the malicious contract

	address, the malicious code on the contract will be triggered. When the logical sequence of calling the call.value() function to send ETH is wrong, there is a risk of reentry attack. In The DAO incident, hackers used this kind of attack, which eventually led to the hard fork of Ethereum.
Audit results	Pass

- **The transaction sequence relies on audit**

Canonical name	Transaction order dependence
Risk description	The execution of the smart contract will produce different results according to the current transaction processing sequence. In the process of transaction packaging execution, when faced with transactions of the same difficulty, miners often choose the priority packaging with high gas cost, so users can specify a higher gas cost to make their own transactions be packaged and executed first.
Audit results	Pass

- **Permission audit**

Canonical name	Authority
Risk description	Intelligent contract in the transfer of assets, mint, self-destruction, change owner and other high level operations , restrictions on the rights to do a function call to avoid security problems caused by leaks permission.
Audit results	通过

- **Floating point and precision audit**

Canonical name	Floating point and precision
Risk description	Since there is no fixed decimal point type in Solidity, developers need to use standard integer data types to implement their own types. In this process, developers may encounter some pitfalls.
Audit results	Pass

- **Arithmetic overflow audit**

Canonical name	Arithmetic overflow

Risk description	Overflow/underflow exists in many programming languages. In the Ethereum virtual machine, the maximum type of uint is 256 bits. Overflow will occur if it exceeds this range. Overflow conditions can lead to incorrect results, especially if the possibility is not anticipated, which may affect the reliability and safety of the program.
Audit results	Pass

- Dangerous function usage audit

Canonical name	Use of dangerous functions
Risk description	When a smart contract uses DELEGATECALL, it will call the code that exists in other smart contracts, but will maintain the current context. Although this feature is convenient for developers to use, it increases the difficulty of designing a secure code base. Attackers will Use the feature of keeping the context unchanged to modify the content of the original context to attack. The function of call is similar to delegatecall. If used improperly, it will cause call injection vulnerabilities. For example, if the parameters of call are controllable, you can control this contract to perform unauthorized operations or call dangerous functions of other contracts.
Audit results	Pass

- No address is empty judgment audit

Canonical name	No address is empty judgm
Risk description	Failure to judge whether the address is empty leads to abnormal contract operations, which in turn causes various unknown errors to be thrown and the smart contract function cannot be completed.
Audit results	Pass

- Short address attack audit

Canonical name	Short address attack
Risk description	When calling the transfer method to withdraw coins, if the user is allowed to enter a short address, this is usually because the exchange has not done any processing, such as not verifying whether the length of the address entered by the user is legal. If an Ethereum address is as follows, notice that the ending is 0: 0x123456789012345678901234567890123456780 0 When we omit the following 00, EVM will get 00 from the high bit of the next

	parameter to supplement. At this time, the token quantity parameter will actually be 1 byte less, that is, the token quantity is shifted by one byte to the left, making The contract sends out a lot of tokens.
Audit results	Pass

- **Self-destruct attack audit**

Canonical name	Self-destruct attack
Risk description	Contract risk self-destruction, S existence of self-destruction interfaces olidity contract, can destroy the current contract, and the contract of the current balance forcibly sent to the specified address.
Audit results	Pass

- **Balance judgment audit**

Canonical name	Balance judgment
Risk description	Insufficient balance will result in false transfers
Audit results	Pass

- **Variable coverage audit**

Canonical name	Variable coverage
Risk description	There are complex variable types in smart contracts, such as structures, dynamic arrays, etc. If they are used improperly, they may overwrite the values of existing state variables and cause abnormal contract execution logic.
Audit results	Pass

- **Security audit of the use of global variables**

Canonical name	Safe use of global variables
Risk description	For example, tx.origin is a global variable of Solidity that traverses the entire call stack and returns the address of the account that originally sent the call (or transaction). Using this variable for authentication makes the contract vulnerable to attacks like phishing. Because tx.origin represents the address of the initial creator of

	the transaction, if you use tx.origin for permission judgment, errors may occur; in addition, if the contract needs to determine whether the caller is the contract address, you need to use tx.origin instead of extcodesize. If someone wants to refuse the external contract to call the current contract, they can implement a requirement (tx.origin == msg.sender) to achieve this requirement.
Audit results	Pass

- **Return value security audit**

Canonical name	Return value safety
Risk description	In Solidity, there are methods such as transfer(), send(), call.value(), transfer transfer failed transactions will be rolled back, and send and call.value transfer failed will return false, if the return is not correctly judged, then Unexpected logic may be executed; in addition, in the implementation of TRC20 Token's transfer/transferFrom function, it is also necessary to avoid transfer failure and return false, so as to avoid false recharge loopholes.
Audit results	Pass

- **Denial of service attack audit**

Canonical name	Denial of service attack
Risk description	A denial of service attack, or Denial of Service, can make the target unable to provide normal services. Such problems also exist in the smart contract, because the contract can not change the nature of intelligence, such attacks may make a contract for a short time or even never be able to return to normal working condition , cause there are many reasons for denial of service contract intelligence, including Malicious revert when acting as a transaction receiver, accidental execution of SELFDESTRUCT instructions , code design flaws leading to gas exhaustion , unexpected lead- outs , etc.
Audit results	Pass