September 25, 2020

Audit Report

The EVAI token contract.

TABLE OF CONTENTS

● 1. DISCLAIMER

This audit report presents the findings of a security review of the smart contracts under scope of the audit. The audit does not give any *warranties on the security of the code*. Using specially designed tools for debugging and using automated tests to verify minor changes and fixes.

I always recommend proceeding to several independent audits and a public bug bounty program to ensure the security of the smart contracts.

2. INTRODUCTION

✓ 2.1 Audit request

EVAI.io will leverage the current industry standards via the Ethereum blockchain for issuing custom digital assets and smart contracts.By conforming to the ERC20 token interface, EVAI will be compatible with existing Ethereum infrastructure, such as wallets and exchanges.

More information can be found on https://www.evai.io/.

✓ 2.2 In scope

This document is a security audit report performed by danbogd https://github.com/danbogd, where EVAI smart contract https://ropsten.etherscan.io/address/0x21921bc278d750b9a487585faa7758ec106616d6#code have been reviewed.

- Evaitoken.sol

3. METHODOLOGY

3.1 My review methodology

My smart contract review methodology involves automated and manual audit techniques.
The applications are subjected to a round of dynamic analysis using tools like linters, program profilers and source code security scanners (publicly available automated Solidity analysis tools such as Remix, Oyente, Solidity static code analyzer SmartCheck). The contracts have their source code manually inspected for security vulnerabilities. This type of analysis has the ability to detect issues that are missed by automated scanners and static analyzers, as it can discover edge-cases and business logic-related problems. Special attention is directed towards to the ability of an owner to manipulate contract.

✓ 3.2 Classification of detected issues

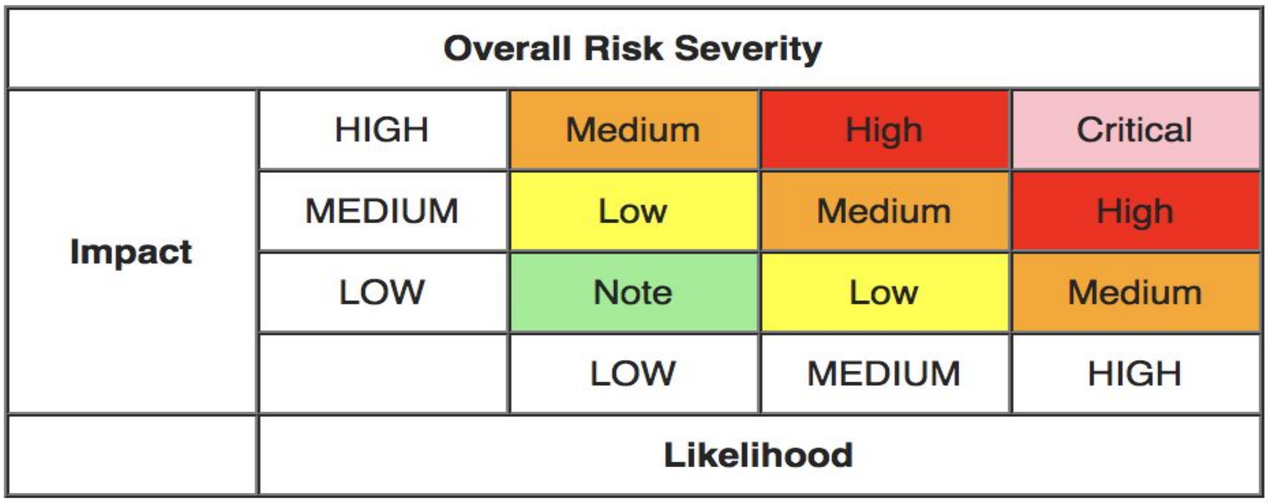
High - vulnerability can be exploited at any time and cause a loss of customers funds or a complete breach of contract operability. (Example: Parity Multisig hack, a user has exploited a vulnerability and violated the operability of the whole system of smart-contracts (Parity Multisigs). This could be performed regardless of external conditions at any time.)

Medium - vulnerability can be exploited in some specific circumstances and cause a loss of customers funds or a breach of operability of smart-contract (or smart-contract system). (Example: ERC20 bug, a user can exploit a bug (or "undocumented opportunity") of transfer function and occasionally burn his tokens. A user can not violate someone else's funds or cause a complete breach of the whole contract operability. However, this leads to millions of dollars losses for Ethereum ecosystem and token developers.)

Low - vulnerability can not cause a loss of customers funds or a breach of contracts operability. However it can cause any kind of problems or inconveniences. (Example: Permanent owners of multisig contracts, owners are permanent, thus if it will be necessary to remove a "misbehaving" owner from the owners list then it will require to redeploy the whole contract and transfer funds to a new one.)

Owner privileges - the ability of an owner to manipulate contract, may be risky for investors.
Note - other code flaws, not security-related issues.

The severity is calculated according to the OWASP risk rating model based on Impact and Likelihood:



Overall Risk Severity				
Impact	HIGH	Medium	High	Critical
	MEDIUM	Low	Medium	High
	LOW	Note	Low	Medium
		LOW	MEDIUM	HIGH
	Likelihood			

4. SUMMARY OF FINDINGS

✓ In total, **2** issues were reported including:

✗ - 0 high severity issues.

● - 0 medium severity issues.

● - 0 low severity issues.

● - 2 notes.

● - 0 owner privileges (optional).

5. AUDIT FINDING

✓ 5.1 HIGH SEVERTY ISSUES

✖ No high severity issues were identified in smart contract.

✓ 5.2 MEDIUM SEVERTY ISSUE

- No high severity issues were identified in smart contract.

✓ 5.3 LOW SEVERTY ISSUES

- No low severity issues were identified in smart contract..

5.4 NOTES

5.4.1 Known vulnerabilities of ERC-20 token

Description

Lack of transaction handling mechanism issue. This is a very common issue and it already caused millions of dollars losses for lots of token users! More information ____ here.

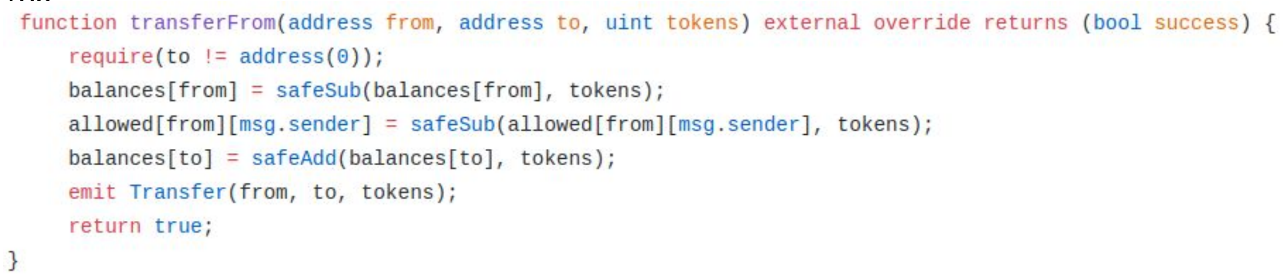
5.4.2 No checking for zero address.

Description

You may add checking for zero address for **from** in **transferFrom** function like in OpenZeppelin implementation https://docs.openzeppelin.com/contracts/3.x/api/token/erc20 where requirements: - sender and recipient cannot be the zero address. But this is not required by the EIP.

Code snippet

Line 109



```
function transferFrom(address from, address to, uint tokens) external override returns (bool success) {
    require(to != address(0));
    balances[from] = safeSub(balances[from], tokens);
    allowed[from][msg.sender] = safeSub(allowed[from][msg.sender], tokens);
    balances[to] = safeAdd(balances[to], tokens);
    emit Transfer(from, to, tokens);
    return true;
}
```

6. CONCLUSION

In the end, I should mention the high code quality of the project and the pleasant UI of https://www.evai.io/ webpage. The audited smart contract is safe to deploy. No any severity issues were found during the
audit.