Audit of 888 DAO

A report of findings by

Genji Sakamoto

February 21th, 2020

Table of Contents

Executive Summary

This audit report has been written to discover issues and vulnerabilities in the 888 DAO smart contracts.

This process included a line by line analysis of the in-scope contracts, optimization analysis, analysis of key functionalities and limiters, and reference against intended functionality.

Audited smart contracts

- 888-token.sol
- 888-vault.sol

Audit Method

- Static analysis based on source.
- Dynamic analysis by testing deployed ones on BSC main net.

Audit Focus

- Contract logic.
- Vulnerabilities for common and uncommon attacks
- Gas optimization
- Validation for variable limiters
- Transparency for all users.

Conclusion

While auditing the smart contracts for 888 DAO project, I realized that the idea is very interesting that could attract many users in near future once deployed.

The logic is some tricky and some complex, but the whole flow of contracts meet the specification perfectly, except there are some issues I recommend to fix before deployment.

Type of Issues

Title	Description	Issues	SWC ID
Integer Overflow and Underflow	An overflow/underflow happens when an arithmetic operation reaches the maximum or minimum size of a type.	0	SWC-101
Function Incorrectness	Function implementation does not meet the specification, leading to intentional or unintentional vulnerabilities.	0	
Buffer Overflow	An attacker is able to write to arbitrary storage locations of a contract if array of out bound happens.	0	SWC-124
Reentrancy	A malicious contract can call back into the calling contract before the first invocation of the function is finished.	0	SWC-107
Transaction Order Dependence	A race condition vulnerability occurs when code depends on the order of the transactions submitted to it.	0	SWC-114
Timestamp Dependence	Timestamp can be influenced by minors to some degree.	0	SWC-116
Insecure Compiler Version	Using a fixed outdated compiler version or floating pragma can be problematic, if there are publicly disclosed bugs and issues that affect the current compiler version used.	0	SWC-102 SWC-103
Insecure Randomness	Block attributes are insecure to generate random numbers, as they can	0	SWC-120

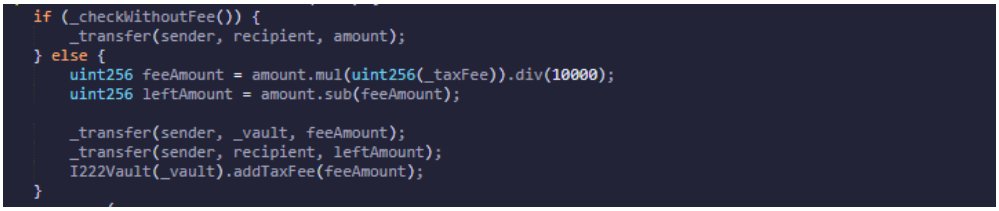
	be influenced by minors to some degree.		
“tx.origin” for authorization	“tx.origin” should not be used for authorization. Use “msg.sender” instead.	0	SWC-115
Delegate call to Untrusted Calling	Calling into untrusted contracts is very dangerous, the target and arguments provided must be sanitized.	0	SWC-112
State Variable Default Visibility	Labeling the visibility explicitly makes it easier to catch incorrect assumptions about who can access the variable.	0	SWC-108
Function Default Visibility	Functions are public by default. A malicious user is able to make unauthorized or unintended state changes if a developer forgot to set the visibility.	0	SWC-100
Uninitialized Variables	Uninitialized local storage variables can point to other unexpected storage variables in the contract.	0	SWC-109
Assertion Failure	The assert() function is meant to assert invariants. Properly functioning code should never reach a failing assert statement.	0	SWC-110
Deprecated Solidity Features	Several functions and operators in Solidity are deprecated and should not be used as best practice.	0	SWC-111
Unused Variables	Unused variables reduce code quality.	0	

Findings

888-token.sol

Code optimization

The token contract source is perfect. I just recommend to move the same code of transfer and transferFrom function to _transfer internal function. But it is not must.



```
if (_checkWithoutFee()) {
    _transfer(sender, recipient, amount);
} else {
    uint256 feeAmount = amount.mul(uint256(_taxFee)).div(10000);
    uint256 leftAmount = amount.sub(feeAmount);

    _transfer(sender, _vault, feeAmount);
    _transfer(sender, recipient, leftAmount);
    I222Vault(_vault).addTaxFee(feeAmount);
}
```

888-vault.sol

The source of vault contract is perfect and works seamlessly.

There are just some small optimization issues but it doesn’t affect to main flow.

Transparency and Security

- There is not any mint function that somebody would ever use for an irregular purpose.