MADE FOR GAMERS

SMART CONTRACT AUDIT

ZOKYO.

July, 5th 2022 | v. 1.0

PASS

Zokyo's Security Team has concluded that this smart contract passes security qualifications to be listed on digital asset exchanges.

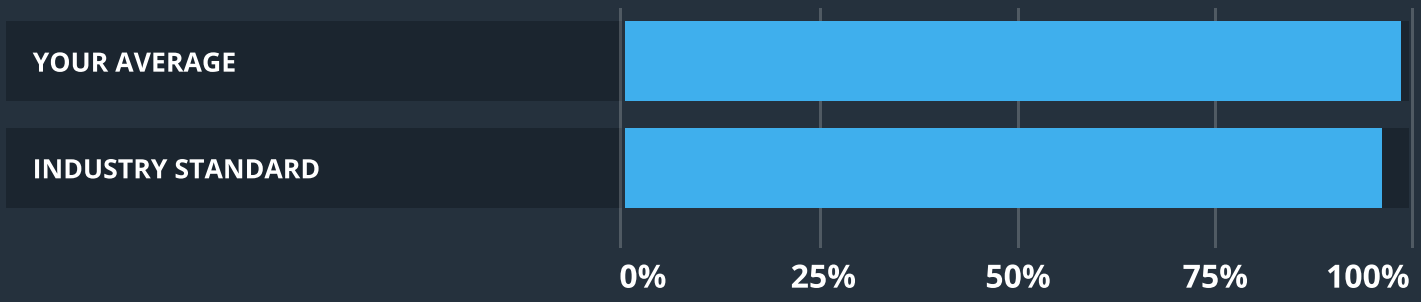
TECHNICAL SUMMARY

This document outlines the overall security of the Made for gamers smart contracts, evaluated by Zokyo's Blockchain Security team.

The scope of this audit was to analyze and document the Made for gamers smart contract codebase for quality, security, and correctness.

Contract Status

Testable Code



The testable code is 100%, which is above the industry standard of 95%.

It should be noted that this audit is not an endorsement of the reliability or effectiveness of the contract, rather limited to an assessment of the logic and implementation. In order to ensure a secure contract that's able to withstand the Ethereum network's fast-paced and rapidly changing environment, we at Zokyo recommend that the Made for gamers team put in place a bug bounty program to encourage further and active analysis of the smart contract.

TABLE OF CONTENTS

AUDITING STRATEGY AND TECHNIQUES APPLIED

The Smart contract's source code was taken from the Made for gamers repository.

Repository - https://github.com/Exponential-Games/expo-token

Last audited commit: ce5733ea2bb7e5832b535cf531060ca11f3d8739

Within the scope of this audit Zokyo auditors have reviewed the following contract(s):

- Expo.sol
- ExpoV0.sol

Throughout the review process, care was taken to ensure that the contract:

- Implements and adheres to existing standards appropriately and effectively;
- Documentation and code comments match logic and behavior;
- Distributes tokens in a manner that matches calculations;
- Follows best practices in efficient use of resources, without unnecessary waste;
- Uses methods safe from reentrance attacks;
- Is not affected by the latest vulnerabilities;
- Whether the code meets best practices in code readability, etc.

Zokyo's Security Team has followed best practices and industry-standard techniques to verify the implementation of Made for gamers smart contracts. To do so, the code is reviewed line-by-line by our smart contract developers, documenting any issues as they are discovered. Part of this work includes writing a unit test suite using the Truffle testing framework. In summary, our strategies consist largely of manual collaboration between multiple team members at each stage of the review:

1	Due diligence in assessing the overall code quality of the codebase.	3	Testing contract logic against common and uncommon attack vectors.
2	Cross-comparison with other, similar smart contracts by industry leaders.	4	Thorough, manual review of the codebase, line-by-line.

EXECUTIVE SUMMARY

Zokyo Security team has conducted the audit for the Expo token from the Exponential-Games protocol by Made for Gamers team.

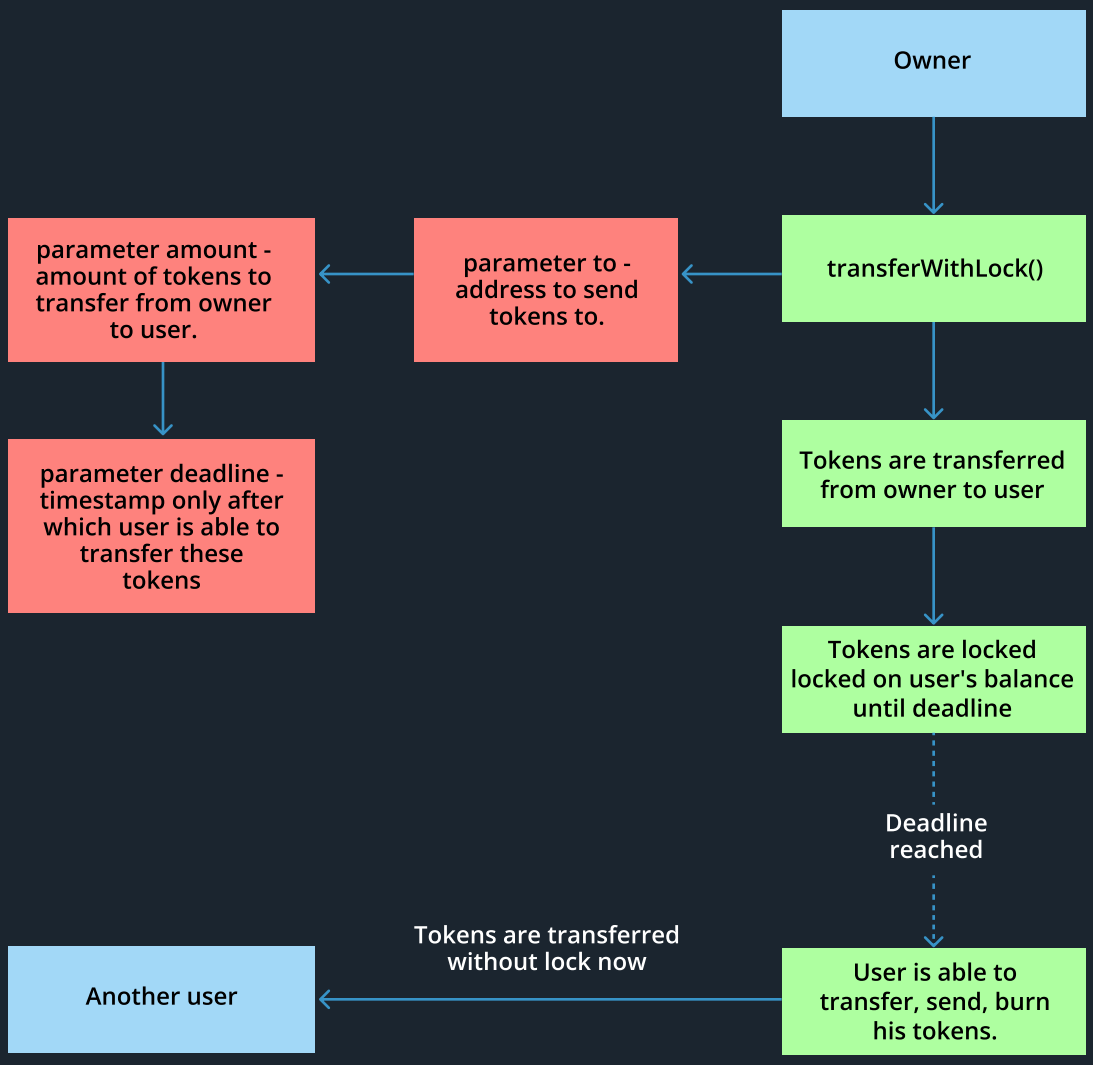
Auditors has carefully checked the implementation of ERC777 token. Contract has high level of code quality and documentation. Token has limited burn functionality (which leaves minimum supply token cannot be burnt under) additionally checked by both auditors team and Made for Gamers team. Also, it is worth to mention, that the token has upgradeable functionality.

Though, there were few critical issues discovered during the audit: for the incorrect initialization of the upgraded contract and for the incorrect implementation of lock functionality. Both issues were successfully resolved by Made for Gamers team.

Also, contracts set has impressive native unit-test coverage which completes the documentation prepared for the token.

PROTOCOL OVERVIEW

Protocol inherits standard ERC777 token. There is a maximum supply of tokens in the contract. Owner is able to mint to any address, however he is not able to mint more tokens than maximum supply.
Owner is able to transfer his tokens to other users with lock period. This lock period means, that user cannot transfer this tokens before deadline of the lock.



STRUCTURE AND ORGANIZATION OF DOCUMENT

For ease of navigation, sections are arranged from most critical to least critical. Issues are tagged “Resolved” or “Unresolved” depending on whether they have been fixed or addressed. Issues tagged “Verified” contain unclear or suspicious functionality that either needs explanation from the Customer’s side or it is an issue that the Customer disregards as an issue. Furthermore, the severity of each issue is written as assessed by the risk of exploitation or other unexpected or otherwise unsafe behavior:

Critical

The issue affects the contract in such a way that funds may be lost, allocated incorrectly, or otherwise result in a significant loss.

High

The issue affects the ability of the contract to compile or operate in a significant way.

Medium

The issue affects the ability of the contract to operate in a way that doesn’t significantly hinder its behavior.

Low

The issue has minimal impact on the contract’s ability to operate.

Informational

The issue has no impact on the contract’s ability to operate.

COMPLETE ANALYSIS

CRITICAL | RESOLVED

Anyone is able to upgrade implementation of the contract.

EXPO.sol and EXPOV0.sol: function _authorizeUpgrade().
Function _authorizeUpgrade() is necessary in order to authorize that upgrading implementation is valid, thus this function should always be implemented and check that msg.sender of upgradeTo() function is valid(Either owner, admin or authorized user).

Recommendation:
Validate msg.sender in _authorizeUpgrade(). For example, onlyOwner modifier from OZ OwnableUpgradeable can be used.

Post-audit:
OnlyOwner modifier is used and scenario was also covered with additional unit-tests.

CRITICAL | RESOLVED

Lock period can be avoided with other ERC777 functions.

Expo.sol.
Contract overrides transfer functions, such as transfer(), send(), burn() in order to check lock period. However, there are few ERC777 standard functions which are not overridden and thus, can be used to avoid lock period on transfer. The following ERC777 functions perform transfer and burn and don't validate lock period: operatorSend(), operatorBurn(), transferFrom().

Recommendation:
Override these functions as well to validate lock period in them.

Post-audit:
Internal functions _send() ans _burn() were overridden to check lock period.

MEDIUM | RESOLVED

Iteration through all locks, including expired locks.

Expo.sol: functions transfer(), send(), burn(), transferWithLock().
Functions iterate through all user's locks to verify that he doesn't transfer more than locked until some period of time. Iteration is performed through expired locks as well, performing unnecessary actions and increasing gas spendings.
Also, in case there are a lot of locks for the user, iterating through all of them might consume more gas than allowed per transaction, preventing user from transferring his tokens. Issue is marked as medium, since only the owner can create locks for user.

Recommendation:
Remove locks, which have expired after successful transfer.

INFORMATIONAL | RESOLVED

Owner is able to destroy the contract.

Expo.sol: function destroySmartContract().
Owner has the ability to destroy the contract with the following function, destroying all users' balances as well. In case the owner key is compromised or stolen, the contract can be destroyed forever with all users' balances. Verify the necessity of this function.
This issue is connected to crucial smart-contract logic, thus need to be mentioned in the report, and it needs to be verified by the team.

Post-audit:
The team has removed the selfdestruct functionality

INFORMATIONAL | RESOLVED

Use Solidity literal.

EXPO.sol and EXPOV0.sol: function initialize(), line 31.
Currently, “maxSupply” is assigned to a value 270000000000000000000000000000. Such notation is difficult to be read and may lead to accuracy mistakes, when some zeros might be missed.

Recommendation:
Since both contracts inherit OZ ERC777 standard token, which has 18 decimals, ether literal can be used to increase code readability. For example: 2_700_000_000_000 ether.

INFORMATIONAL | RESOLVED

Burnable token.

EXPO.sol and EXPOV0.sol inherit the OZ ERC777 standard token which has a default burn function. Though, the default burnable functionality allows every user to burn their own tokens. Thus in case of big distributions or any hack for a big amount, there is an ability for the malicious user to burn enough tokens to affect the economy of the protocol. Thus verify the usage of the default burn functionality.
Burn functionality itself is not a security issue, but it crucial for the protocol, thus it needs to be verified and reflected in the report.

Post-audit:
The Customer team decided to leave the burn functionality, but set a minimum total supply that it cannot go below

	Expo.sol	ExpoV0.sol
Re-entrancy	Pass	Pass
Access Management Hierarchy	Pass	Pass
Arithmetic Over/Under Flows	Pass	Pass
Unexpected Ether	Pass	Pass
Delegatecall	Pass	Pass
Default Public Visibility	Pass	Pass
Hidden Malicious Code	Pass	Pass
Entropy Illusion (Lack of Randomness)	Pass	Pass
External Contract Referencing	Pass	Pass
Short Address/ Parameter Attack	Pass	Pass
Unchecked CALL Return Values	Pass	Pass
Race Conditions / Front Running	Pass	Pass
General Denial Of Service (DOS)	Pass	Pass
Uninitialized Storage Pointers	Pass	Pass
Floating Points and Precision	Pass	Pass
Tx.Origin Authentication	Pass	Pass
Signatures Replay	Pass	Pass
Pool Asset Security (backdoors in the underlying ERC-20)	Pass	Pass

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Made for gamers team

As part of our work assisting Made for gamers team in verifying the correctness of their contract code, our team has checked the complete set of unit tests prepared by the Made for gamers team.

It needs to be mentioned, that the original code has a significant original coverage with testing scenarios provided by the Made for gamers team. All of them were also carefully checked by the auditors' team.

Contract: EXPO
- ✓ Should return the name of token (67ms)
- ✓ Should return the symbol of token (49ms)
- ✓ Should return the token supply after initial minting of token (68ms)
- ✓ Should return the granularity of token
- ✓ Should return the balanceOf of token of specified address (104ms)
- ✓ Should send tokens to recipient (189ms)
- ✓ Should burn tokens from caller (126ms)
- ✓ Should mint tokens to recipient (208ms)
- ✓ Should not be able to mint after max supply (220ms)
- ✓ Should return max supply

Contract: EXPO-Upgradability
- ✓ Should upgrade token contract (1134ms)

Contract: EXPO-ACL
- ✓ Should be able to find owner of token (42ms)
- ✓ Should be able to transfer ownership (111ms)

Token locking functionality with transfer
- ✓ Should send unlocked tokens (133ms)
- ✓ Should not send locked tokens before deadline (187ms)
- ✓ Should not transfer locked tokens before deadline with existing deadline (347ms)
- ✓ Should send locked tokens after deadline with existing deadline (558ms)
- ✓ Should send locked tokens after deadline (434ms)

Token locking functionality with send
- ✓ Should send unlocked tokens (131ms)
- ✓ Should not send locked tokens before deadline (198ms)
- ✓ Should not send locked tokens before deadline with existing deadline (342ms)
- ✓ Should send locked tokens after deadline with existing deadline (549ms)

✓ Should send locked tokens after deadline (416ms)

Token locking functionality with burn

✓ Should burn unlocked tokens (116ms)
✓ Should not burn locked tokens before deadline (181ms)
✓ Should not burn locked tokens before deadline with existing deadline (283ms)
✓ Should burn locked tokens after deadline with existing deadline (435ms)
✓ Should burn locked tokens after deadline (338ms)

Token locking functionality with mint with initial supply 0

✓ "Should mint tokens

Contract: EXPO

✓ Should return the name of token
✓ Should return the symbol of token
✓ Should return the token supply after initial minting of token
✓ Should return the granularity of token
✓ Should return the balanceOf of token of specified address (53ms)
✓ Should send tokens to recipient (107ms)
✓ Should burn tokens from caller (85ms)
✓ Should mint tokens to recipient (119ms)
✓ Should not be able to mint after max supply (55ms)
✓ Should not be able to mint after max supply (55ms)

Contract: EXPO-Upgradability

✓ Should upgrade token contract (515ms)

Contract: EXPO-ACL

✓ Should be able to find owner of token
✓ Should be able to transfer ownership (73ms)

42 passing (14s)

FILE	% STMTS	% BRANCH	% FUNCS
Expo.sol	100	96.43	100
ExpoV0.sol	100	100	100
All files	**100**	**96.67**	**100**

CODE COVERAGE AND TEST RESULTS FOR ALL FILES

Tests written by Zokyo Secured team

As part of our work assisting Made for gamers in verifying the correctness of their contract code, our team was responsible for writing integration tests using the Truffle testing framework.

Tests were based on the functionality of the code, as well as a review of the Made for gamers contract requirements for details about issuance amounts and how the system handles these.

Contract: EXPO

- ✓ Should return the name of token
- ✓ Should return the symbol of token
- ✓ Should return the token supply after initial minting of token
- ✓ Should return the granularity of token
- ✓ Should return the balanceOf of token of specified address
- ✓ Should send tokens to recipient (42ms)
- ✓ Should burn tokens from caller
- ✓ Should mint tokens to recipient
- ✓ Should not be able to mint after max supply
- ✓ Should return max supply
- ✓ User shouldn't be able to transfer more than he owns
- ✓ Anyone other than owner should not be able to kill smart contract
- ✓ Owner should be able to kill smart contract

Contract: EXPO-Upgradability

- ✓ Should upgrade token contract (410ms)
- ✓ Should be able to find owner of token
- ✓ Should be able to transfer ownership
- ✓ Should send unlocked tokens
- ✓ Should not send locked tokens before deadline (42ms)
- ✓ Should not transfer locked tokens before deadline with existing deadline (90ms)
- ✓ Should send locked tokens after deadline with existing deadline (106ms)
- ✓ Should send locked tokens after deadline (99ms)
- ✓ Should send unlocked tokens
- ✓ Should not send locked tokens before deadline (44ms)
- ✓ Should not send locked tokens before deadline with existing deadline (71ms)
- ✓ Should send locked tokens after deadline with existing deadline (124ms)
- ✓ Should send locked tokens after deadline (79ms)
- ✓ Should burn unlocked tokens

✓ Should not burn locked tokens before deadline
✓ Should not burn locked tokens before deadline with existing deadline (82ms)
✓ Should burn locked tokens after deadline with existing deadline (110ms)
✓ Should burn locked tokens after deadline (92ms)
✓ Should mint tokens

Contract: EXPO

✓ Should return the name of token
✓ Should return the symbol of token
✓ Should return the token supply after initial minting of token
✓ Should return the granularity of token
✓ Should return the balanceOf of token of specified address
✓ Should send tokens to recipient (43ms)
✓ Should burn tokens from caller
✓ Should mint tokens to recipient
✓ Should not be able to mint after max supply
✓ Should return max supply
✓ Should upgrade token contract (322ms)

Contract: EXPO-ACL

✓ Should be able to find owner of token
✓ Should be able to transfer ownership

Contract: EXPO

✓ Owner should mint tokens to himself and to user (98ms)
✓ Owner should not mint more then max supply tokens (67ms)
✓ Owner should destroy contract
✓ Owner should transfer with lock
✓ Owner should burn tokens

50 passing (6s)

FILE	% STMTS	% BRANCH	% FUNCS
Expo.sol	100	100	100
ExpoV0.sol	100	100	100
All files	**100**	**100**	**100**

We are grateful to have been given the opportunity to work with the Made for gamers team.

The statements made in this document should not be interpreted as investment or legal advice, nor should its authors be held accountable for decisions made based on them.

Zokyo's Security Team recommends that the Made for gamers team put in place a bug bounty program to encourage further analysis of the smart contract by third parties.

ZOKYO.