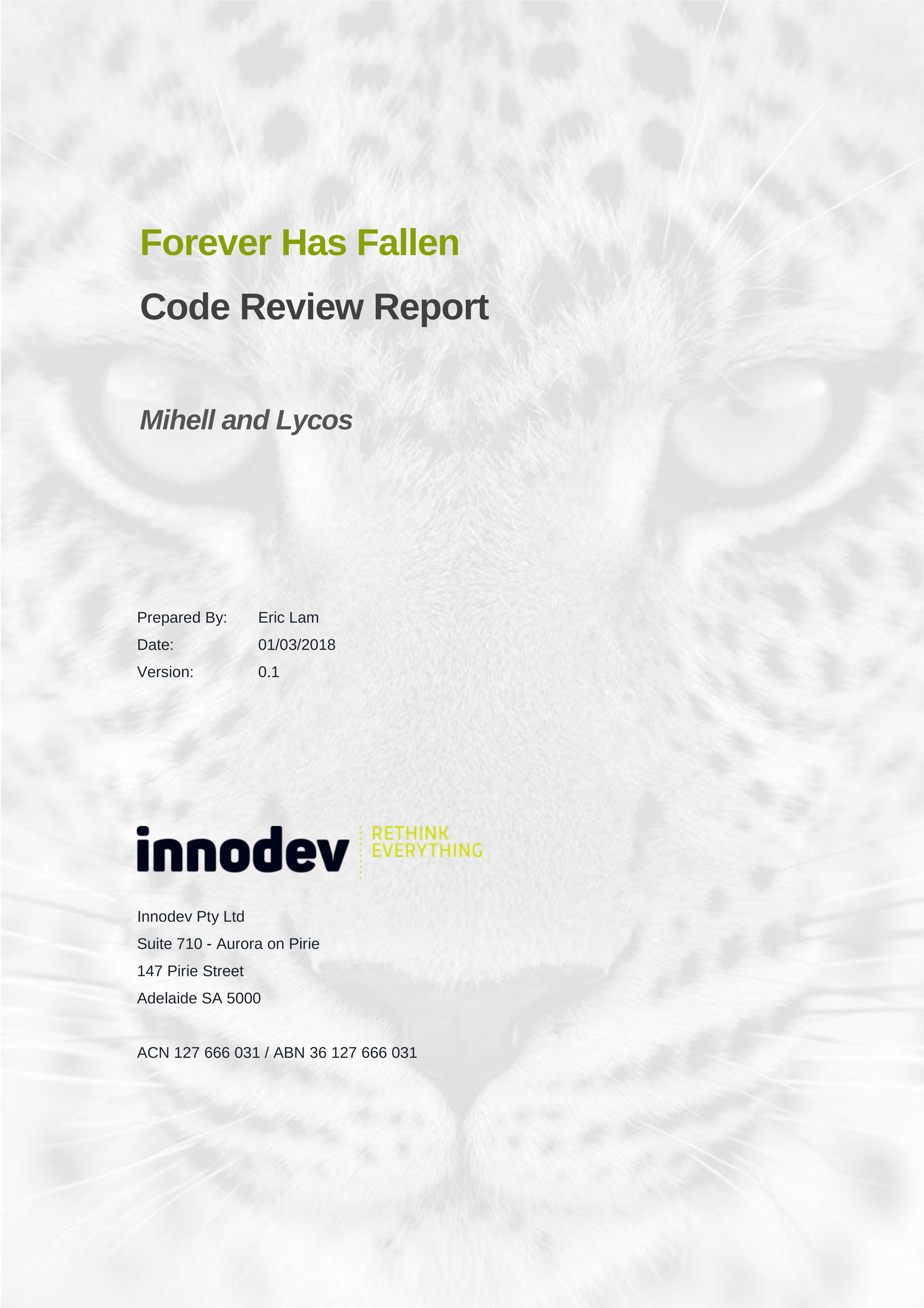


Forever Has Fallen

Code Review Report

Mihell and Lycos

Prepared By:	Eric Lam
Date:	01/03/2018
Version:	0.1

innodev RETHINK EVERYTHING

Innodev Pty Ltd
Suite 710 - Aurora on Pirie
147 Pirie Street
Adelaide SA 5000

ACN 127 666 031 / ABN 36 127 666 031

The digital transformation of an organisation and the transition to a digital core presents one of the greatest challenges to a successful company – digital maturity is one of the greatest signifiers of success. Our speed and agility combined with deep expertise in digital systems and thought leadership, position us at the forefront in the transformation of business and the integration of digital systems. ***Rethink Everything.***

Table of Contents

1. Overview

1.1. Background

Forever has Fallen is a new game utilising blockchain ledger capabilities and smart contracts to provide the tools and framework to create a gaming experience. The initial step is to create an Initial Coin Offering (ICO) to raise funds for the game.

The ICO smart contract is written by UseTech, a software company in Russia with experience in writing smart contracts.

Innodev was engaged to provide a secondary source of checks for the smart contract. This consisted of a code review on the smart contract written by UseTech.

1.2. Code Review Objectives

The aim of the code review is to do a examination of the source code and unit tests, limited testing was performed. The intention is to find mistakes overlooked in the original software development and check the code does what is expected.

1.3. Purpose

The purpose of this document is to detail:

1. The findings of the source code and its relevance to the intended requirements.

2. Requirements Check

2.1. Token Minting Check.

Lists the tokens, the amount assigned to each wallet on initial deployment.

Token Wallet	Amount
Token supply quantity	1 Billion
Maximum ICO token cap	350 Million
Community Reserve	450 Million
Team	170 Million
Advisors	4.8 Million
Bounty	17.6 Million
Administrative	7.6 Million
Players Reserve	0, unsold tokens for 350 Million ICO cap to be transferred here at end of ICO.

2.2. Dates

As coded into the smart contract.

The vesting period defines the period in which the tokens cannot be transferred after receiving from that wallet.

For example a vesting period of, 1/1/2020 on the team tokens means: A wallet which has been assigned 100 tokens from the "Team" wallet cannot transfer those 100 tokens to someone else until the end of the vesting period (1/1/2020).

Type	Timestamp	GMT time	Adelaide Time (daylight saving ends 1st April)
ICO sale start date	1525777200	Tuesday, May 8, 2018 11:00:00 AM	Tuesday May 8, 2018 8:30:00 PM
ICO sale end date	1529406000	Tuesday, June 19, 2018 11:00:00 AM	Tuesday June 19, 2018 08:30:00 PM
Team vesting end date	1592564400	Friday, June 19, 2020 11:00:00 AM	Friday June 19, 2020 08:30:00 PM
Bounty vesting end date	1529406000	Tuesday, June 19, 2018 11:00:00 AM	Tuesday June 19, 2018 08:30:00 PM

2.3. Token price

Price of each Forever Coin during the ICO is coded into contract.

1 Ether will buy 10 000 Forever Coins (Ten Thousand).

Upon purchase of Forever Coins, the wallet of the buyer is assigned the Forever Coin straight away, that is, it does not wait till the end of the ICO to distribute the Forever Coins to the buyers wallets.

2.4. ICO Minimum/Maximum Caps

There is no minimum amount of Forever Coin a buyer needs to purchase, that is, a buyer can purchase 1 Wei (the smallest denomination of Ether) worth of Forever Coin.

There is a maximum cap of 350 million Forever Coins in the ICO. When reached, the ICO contract will not allow any more to be purchased. Any extra Ether will be refunded to the buyer.

2.5. Refund Procedures

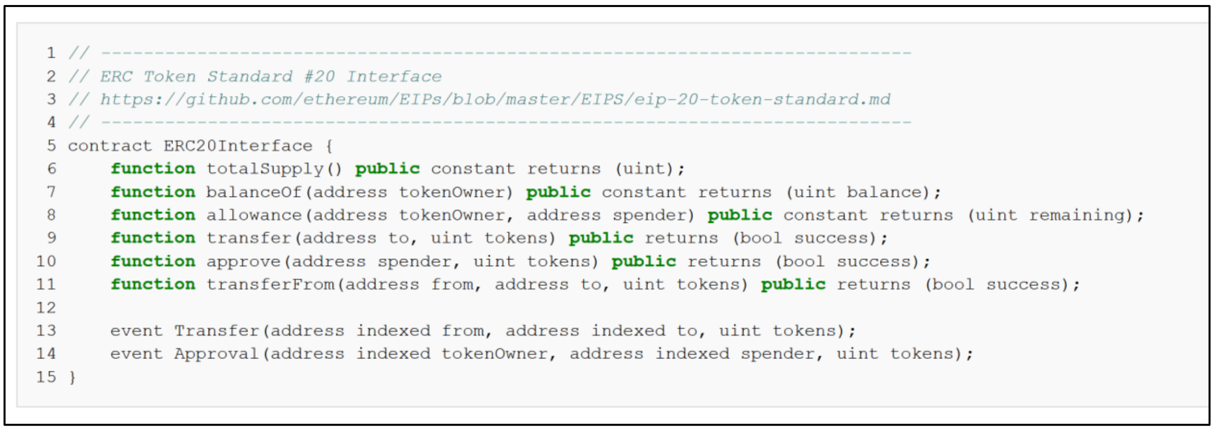
There is no facility to refund Ether to Forever Coin buyers.

2.6. ERC20 Token Type

The Forever Coin is implemented as an ERC20 standard type.

Ensuring Forever Coin is a ERC20 standard allows it to be re-used by other applications, for example wallets and decentralized exchanges.

To adhere to the ERC20 interface, Forever Coin must implement a specific list of interfaces listed here: https://theethereum.wiki/w/index.php/ERC20_Token_Standard



```
// ----------------------------------------------------------------------------
// ERC Token Standard #20 Interface
// https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20-token-standard.md
// ----------------------------------------------------------------------------
contract ERC20Interface {
    function totalSupply() public constant returns (uint);
    function balanceOf(address tokenOwner) public constant returns (uint balance);
    function allowance(address tokenOwner, address spender) public constant returns (uint remaining);
    function transfer(address to, uint tokens) public returns (bool success);
    function approve(address spender, uint tokens) public returns (bool success);
    function transferFrom(address from, address to, uint tokens) public returns (bool success);

    event Transfer(address indexed from, address indexed to, uint tokens);
    event Approval(address indexed tokenOwner, address indexed spender, uint tokens);
}
```

2.7. Token Size

The maximum number of Forever Coins someone can buy is the ICO maximum cap

The minimum number of Forever Coins someone can buy is 1 Wei = 0.000 000 000 000 01 (10^{-14}) forever token

3. Code Review

Innodev has reviewed the smart contract code and unit tests. During the code review Innodev has communicated with UseTech about the code, the correspondence is provided in accompanying word documents. Innodev is satisfied with the responses and the code in general.

3.1. Analysis Tools

As part of the code review Innodev ran Mythril over the smart contract. Mythril detects a range of security issues, including integer underflows, owner-overwrite-to-Ether-withdrawal, and others. However, the analysis will not detect business logic issues and is not equivalent to formal verification.

Analysis was completed successfully and no issues where found.

3.2. Review against Known Attacks

Innodev reviewed against the known attacks listed here: https://consensys.github.io/smart-contract-best-practices/known_attacks/. The outcomes are listed below.

3.2.1. Integer Overflow

An overflow/underflow happens when an arithmetic operation reach the maximum or minimum size of the type. The standard way of handling these issues is to use a common library from Zepplin called SafeMaths.

Innodev has noted that this library is not used in the code and raised it with UseTech. UseTech have responded with: The arithmetic operations in the code are not complex enough to require the use of this library and there are additional overheads in using the library.

Innodev has reviewed all mathematical operations and found an overflow issue if someone sends too much Ether to the ICO, however, this is not possible as there is not enough Ether in the world to do so.

3.2.2. Race Conditions

Checking for:

1) Use send() instead of call.value()() if possible.
2) Complete internal work first before calling external functions (or functions which call external functions).

FHFToken.sol does not call any external functions or send().

FHFTokenCrowdsale.sol processPayment contains the external function call to FHFToken.transfer() and a send(). No state change occurs after calling the external functions.

No issues found here.

3.2.3. Timestamp Dependence

There is no timestamp dependence on critical functions of the contract. That is, there are no timestamp dependencies on the transfer functions.

No issues found here.

3.2.4. DOS with (Unexpected) revert

There is a revert function call when refund cannot be sent back to the buyer. This is not a denial of service (DOS) as the logic does not block another address from transferring Ether to the contract.

No issues found here.

3.2.5. DOS with Block Gas Limit

Does not apply, there are no loops.

No issues found here.

3.2.6. Forcibly Sending Ether to a Contract

Only a consideration if important logic is placed in the fallback function or making calculations based on a contracts balance. Neither applies in this case.

No issues found here.

3.3. Short Address Attack

Short Address attack can occur when the addresses of either the transferrer or transferee has an address that is shorter than 32 bytes.

The solution to this is to perform this size check in the contract or the exchange must support it. The only option open under our control is to check it in the contract, which is implemented.

However, it must be noted that this prevents users with Multi-Signature wallets to buy Forever Coin.

3.4. Other

Noted that a assert() is used instead of require() to check msg.value > 0. It is recommended in solidity to always use require instead of assert where possible.

This is not a security issue and only occurs when the user tries to buy Forever Coin with zero Ether.

innodev RETHINK EVERYTHING

Innodev Pty Ltd
Suite 710 - Aurora on Pirie
147 Pirie Street
Adelaide SA 5000

Phone: +61 8 7421 5901
Website: innodev.com.au