SecurityInnovation®
THE SOFTWARE SECURITY COMPANY

UBIshares Security Audit Code Review

Document Name:	Code Review
Date:	Jan. 21, 2021
Customer Contact:	Bensam Packiaraj <bensampackiaraj@gmail.com>
Author:	Mohd Javeed <mjaveed@securityinnovation.com> Ben Stewart <bstewart@securityinnovation.com>
Project Manager:	Garrett Jaynes <gjaynes@securityinnovation.com>

SecurityInnovation®
THE SOFTWARE SECURITY COMPANY
BOSTON | SEATTLE | PUNE
187 Ballardvale St., Suite A195 ● Wilmington, MA 01887 ● Ph: +1.978.694.1008
getsecure@securityinnovation.com ● www.securityinnovation.com

Introduction

Security Innovation performed a code review of various components developed by UBIshares. Security Innovation conducted this review over the course of 7 engineering days. This report summarizes the testing that was completed and the issues that were uncovered.

UBIshares has developed a universal basic income application which is a smart contract based platform developed using Xamarin and .NET technologies, built on the EOSIO blockchain software and powered by UBI tokens.

The platform provides an ecosystem of services through an easy-to-use interface, connecting the users while allowing full transparency and traceability of transactions. Individuals, SMEs, and enterprises are the target users for the services provided by the UBI platform. To simplify the transaction process and increase UBI token adoption, the UBI App will integrate several services in one single platform including transfers, transaction history for tracking bonuses, and additional features. In addition, UBIshares supports a live auction which is provided through their current web interface and allows users to invest in the current auction round for a specified pool amount.

The code review and generated test cases focused primarily on a blend of manual code review and static analysis. The following components were prioritized as part of this engagement:

- Auction Functionality
  - Auction contract
  - UBIAuctions contract
  - Auction.js
  - Global.asax.cs
- Staking Functionality
  - UBIToken contract
  - Utility and dependency contracts
  - Defi.js
  - Defistaking.aspx.cs
- Android Mobile Application
  - Flacon APK
  - Hawk APK
  - Panacea APK

While investigating, the following assumptions were made:

- The facial identification/authentication mechanism is out of scope
- The iOS application is out of scope
- The UBIWeb Application functionality is largely out of scope (except for the aforementioned blockchain components)
- The overall system was under active development and testing focused on the source code provided at the beginning of the assessment
- Current version of Open Zeppelin libraries were used for the utility functions

Overall, the system was found to be moderately secure with four issues identified. These issues ranged in severity from low to medium. The most severe issue identified presents a configuration that could result in funds being locked within a contract. Separately an issue that was identified greatly increases the attack surface of a potential attacker by using software components with known public exploits which could be easily leveraged by a malicious actor. Also, the Android application allows backups which can lead to forms of information disclosure which can put user data at risk. These issues and others are highlighted briefly below:

- A total of 4 security issues were identified:
  - PR1 - Floating Pragma Version
  - PR2 - Contracts Could Lock Ether
  - PR3 - Software Components with Known Vulnerabilities
  - PR4 - Android Application Allows Backups

Blockchain Component Description

This section describes the blockchain components in further detail. This was one of the factors used to generate relevant test cases.

Auction Functionality

Auction Contract

The Auction contract defines the following functions:

- update()
    - Manual function to adjust currentPrice based on $dropPrice \times dropLevel$
- bid(address _bidder, uint256 _amount)
    - Manual function to bid within a live auction, track max bid, and bid history
- calcSupply()
    - Internal function to calculate $currentSupply$ based on $(totalAmount)/(currentPrice)$
- complete()
    - Manual function which ends an auction, calculates price/value, and tracks bidders
- status()
    - View function that returns auction data such as endTime, currentPrice, totalSupply, etc
- ended()
    - View function that returns uint256 value related to auction status
- startInt()
    - View function that returns startTime and dropInterval values
- allBidders()
    - View function that returns the addresses of bidders
- tokensOf(address _bidder)
    - View function that returns the tokens of a specific bidder
- bidOf(address _bidder)
    - View function that returns the amount related to a specific bidder
- sumAuc()
    - View function that returns the total amount of the auction, number of bidders, the max bid, and other data related to the auction
- sumAdd(address _bidder)
    - View function that returns the total amount, bids, price, etc from an auction for a specific bidder

UBIAuctions Contract

The UBIAuctions Contract defines the following functions:

- status()
    - View function that returns the status of an auction
- update()

- Manual, owner function to update price of an auction
- ended()
    - View function that returns the end value of an auction
- bid()
    - Payable function to submit a bid
- create()
    - Manual, owner function to create new auction
- complete()
    - Manual, owner function to complete or process an auction
- setConfig(uint256[] memory _config)
    - Manual function to create account configuration
- startInt()
    - View function that returns Auction startInt()
- allbidders()
    - View function that returns Auction allBidders()
- getConfig()
    - View function that returns an auction configuration
- sumAuc(uint256 index)
    - View function that returns Auction sumAuc()
- sumAdd(uint256 index, address _bidder)
    - View function that returns Auction sumAdd()
- tokensOf(address _bidder)
    - View function that returns Auction tokensOf()
- bidOf(address _bidder)
    - View function that returns Auction bidOf()

Auction.js

This file loads all the data related to an auction on an ongoing basis and is also responsible for placing the bid (Place Bid function of the smart contract) when the user interacts with that functionality through the front-end interface. This functionality validates the bid by checking to ensure Metamask is installed then validates the user is signed in. Finally, the Bid function on the contract is called which passes the bid amount as a parameter and calls the Metamask extension for the transaction execution.

Global.asax.cs

This file is important for running a timer and triggering certain blockchain functions of the Auction smart contract. The timer performs the following tasks:

- Check for the time interval (defined in the Auction smart contract) when the price will be updated. If the interval equals the current time, then the timer calls the UpdatePrice smart contract function and the price gets updated for that round of the auction and the new price is set to the value already defined in the contract.
- Checks if the auction has ended because of the round time completing or because the

complete pool is sold out. If the auction has ended, the timer executes the "complete" function, sends tokens to all the bidders, and then starts a new auction round.

- Lastly, it reads the status of the current auction round from the blockchain and writes them to a text file which is accessed by Auction.js.

Staking Functionality

UBIToken Contract

The UBIToken contract defines the following functions:

- createStake(uint256 _stake)
  - Manual function to burn value and create stake
- removeStake(uint256 _stake)
  - Manual function to mint value and remove stake
- stakeOf(address _stakeholder)
  - View function that returns the stakes of a specific stakeholder
- totalStakes()
  - View function that returns all the stakes of a specific stakeholder
- isStakeholder(address _address)
  - View function that returns value if address is stakeholder
- addStakeholder(address _stakeholder)
  - Manual function to add individual stakeholder
- removeStakeholder(address _stakeholder)
  - Manual function to remove a specific stakeholder
- totalStakeHolders()
  - View function that returns the number of stakeholders
- rewardOf(address _stakeholder)
  - View function that returns the rewards of a specific stakeholder
- totalRewards()
  - View function that returns total rewards of all stakeholders
- calculateReward(address _stakeholder)
  - View function that returns reward amount based on a fixed value
- distributeRewards()
  - Manual, owner function to distribute rewards to stakeholders
- withdrawReward(address _stakeholder)
  - Manual function to transfer stakeholder reward

Utility Contracts (OpenZeppelin)

The SafeMath contract defines the following functions:

- Addition, subtraction, multiplication, division, and modulus

The Address contract defines the following functions:

- isContract, sendValue, functionCall, and functionCallWithValue

The ERC20 contract defines the following functions:

- name, symbol, decimals, totalSupply, balanceOf, transfer, allowance, approve, transferFrom, increaseAllowance, decreaseAllowance, _transfer, _mint, _burn, _approve, _setupDecimals, and _beforeTokenTransfer

The Ownable contract defines the following functions:

- owner, renounceOwnership, and transferOwnership

Defi.js

This file loads the staking contract and aggregates all the staking information from the blockchain, displays the data via the web interface, and then draws the relevant graphs. If the user selects a Metamask address as the current address to perform transactions, then Defi.js makes sure that Metamask is installed and that the selected address is the same address which is valid.

In addition, it is also responsible for adding and removing the stakes (the createstake and removestake function of the staking smart contract). Once the staking value are validated, the stake amount is passed a parameter and the Metamask extension is used to execute the transactions.

DefiStaking.aspx.cs

If the user has elected to use a native UBIshares address for executing the transactions, then the code for adding and removing a stake is executed by receiving the required parameters (account address, private key, stake amount) from this file. The transaction gets executed and the private key is immediately cleared from the page. This file also contains the WithdrawReward function which calls the web API (through UBI Web) and makes the request to transfer the rewards to the specific account of a user.

Problem Report Summary

A total of 4 issues were identified. This section describes, at a high level, each of the problems discovered. The problem report summaries are sorted by problem report ID. The format of the problem report table is as follows:

- Problem Report ID
- Component in which the issue was discovered
- Problem Report severity
- Problem Report title

PR #	Component	Severity	Title
1	Smart Contracts	Low	Floating Pragma Version
2	Smart Contracts	Medium	Contracts Could Lock Ether
3	UBI Web	Medium	Software Components with Known Vulnerabilities
4	Android Application	Medium	Android Allows Backup

Problem Reports

Problem Report 1 - Floating Pragma Version

Severity	Target	Line Number
Low	Auction & Token contracts	1

Description

To lower the potential risk of undiscovered bugs, smart contracts should be deployed with the compiler version that has already been thoroughly tested. By locking in the pragma, it helps to ensure that contracts do not get deployed with a different version which could potentially increase the chances for bugs or other unexpected consequences that the original authors may not have originally considered.

The current version is specified at a minimum of v0.6.0 and is not locked in:

`pragma solidity ^0.6.0;`

Remediation

Lock Pragmas to Specific Compiler Version - Locking smart contracts to a specific version helps guarantee deployment occurs consistently with the original authors intentions and limits the risk of unknown bugs in latter versions.

myContract.sol

```
// less secure
pragma solidity ^0.6.0;

// more secure
pragma solidity 0.6.0;
```

Use Latest Compiler Build - It is recommended that UBIshares use the latest available update of the solidity compiler, which is version 0.8.1. Older versions such as 0.6.0 have known issues and must be avoided. For further information on the latest build: https://github.com/ethereum/solidity/releases

Problem Report 2 - Contracts Could Lock Ether

Severity	Target	Line Number
Medium	UBIAuctions	176

Description

The UBIAuctions contract makes uses of a bid() function, which is a payable function, and handles the functionality to support bidding/tracking on a specific auction. However, upon review, it was noticed that this contract does not appear to support a withdrawal function. A specific withdrawal function provides the ability to extract funds sent to the contract. In the event that funds are accidentally sent to the contract, it might be impossible to retrieve said funds and they will be lost.

UBIAuctions

```
...

 function bid() public payable{
   if (msg.value < 10000000000000000)
     return;

   auctions[auctions.length -1].bid(msg.sender, msg.value);
 }
```

Remediation

Support a withdraw function or remove the payable attribute It is recommended that either a withdraw function is added to recover funds or that the payable attribute is removed.

Problem Report 3 - Software Components With Known Vulnerabilities

Severity	Target	Line Number
Medium	UBI Web	N/A

Description

UBI Web uses a version of jQuery which contains known security vulnerabilities. Exploitation of these vulnerabilities could lead to a loss of confidential data or be used as a pivot point by an attacker to gain further control of the system.

Problem Details

UBI Web implements the following package with known Cross-Site Scripting vulnerabilities:

- jQuery v3.4.1

jQuery v3.4.1 was released in 2019 and has known vulnerabilities in the CVE database relating to Cross-Site Scripting. For further details of these vulnerabilities, see:

- https://github.com/advisories/GHSA-gxr4-xjj5-5px2 (affects v3.4.1)
- https://github.com/advisories/GHSA-jpcq-cgw6-v4j6 (affects v3.4.1)

Based on an initial review, the system under test does not seem to be directly exploitable to these vulnerabilities. However, due to time constraints, this investigation and exploit development effort was time boxed.

Remediation

Update jQuery to version 3.5.1. The package should be updated to the most recent patched version which fixes the known CVEs.

Monitor Updates and Security Advisories. There should be a defined process and responsible party for staying aware of any updates released for the Security Audit application and any security advisories made public.

- Periodically run tools that verify if the modules being used by the application have known vulnerabilities.
- If the tools do identify some issues, verify if a patch has been released by the vendor.

- Proceed to test if the patch breaks any existing functionality and then deploy it to all hosts running that version of the affected module.

Problem Report 4 - Android Application Allows Debug

Severity	Target	Line Number
Medium	Falcon APK	N/A

Description

The Falcon mobile application allows users to back up all application data. An attacker with access to the user's device or application with elevated permissions is able to back up all user data locally or remotely. This could result in the exposure of sensitive user information or modification of the application data before it is restored.

Problem Details

The Falcon APK enables the Android `allowBackup=true` in the *AndroidManifest.xml* . This allows users to back up all application data that it stores internally in the Android Sandbox. An attacker who is able to back up the application will have access to all application data stored on the device which could include sensitive data about the user, account information, or potential keys and access credentials. Also, on older versions of Android, it was possible to take a backup, modify it, then re-deploy it which could lead to different forms of tampering.

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
android:versionCode="24" android:versionName="2.4" android:installLocation="auto"
android:compileSdkVersion="29" android:compileSdkVersionCodename="10"
package="com.ubifalcon.ubi" platformBuildVersionCode="29" platformBuildVersionName="10">
 <uses-sdk android:minSdkVersion="21" android:targetSdkVersion="29"/>
 <uses-permission android:name="android.permission.CAMERA"/>
 <uses-permission android:name="android.permission.INTERNET"/>
 <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"/>
 <application android:label="UBI Falcon"
  android:icon="@mipmap/icon"
  android:name="android.app.Application"
  android:allowBackup="true"
  android:largeHeap="true"
  android:extractNativeLibs="true"
  android:usesCleartextTraffic="true"
```

Remediation

Most apps maintain the entire user state remotely and do not rely on local data. There are few scenarios where users genuinely need to backup app data. For this reason, the inconvenience created by disallowing backups is often quite low, while the benefit of additional defense in depth is high.

The UBIshares team should review its internal build settings and policies to ensure that every app built is packaged with the `allow_backup=false` setting.

The following resource provides additional documentation and information pertaining to the Android manifest file and all available parameters.

- https://developer.android.com/guide/topics/manifest/manifest-intro

Observations

Over the course of testing, there were a total of 4 problems identified. In addition, there were several other observations made. While not severe enough to warrant problem reports, Security Innovation recommends that UBIshares investigate these in addition to the security-related issues that were found.

Observation 1: Time.now Deprecated

While reviewing the Auction contract, it was noticed that to get the current block timestamp, the now syntax is used:

Auction

```
aucVal["startTime"] = now;
```

Using "now" has been deprecated and as of Solidity version 0.7.0, the use of now has been removed. It is recommended that block.timestamp is used as an alternative. For additional information, please consult the documentation:

- https://docs.soliditylang.org/en/develop/units-and-global-variables.html?highlight=block.timestamp#block-and-transaction-properties

Observation 2: SPDX license identifier

While reviewing the contracts related to the Auction functionality, it was noticed that the SPDX license identifier was not provided in the source file. This was applied for the Token contract and it is recommended that the Auction contracts also include a license identifier. This is important for assigning the specific license that a contract uses. Failure to include will also result in compiler warnings. Further information documented below:

- https://docs.soliditylang.org/en/v0.6.8/layout-of-source-files.html?highlight=spdx#spdx-license-identifier

Observation 3: Long Literals

While reviewing the contract code associated with the token and auction contracts, it was noticed that long literals composed of many digits were used. Literals with many digits can be difficult to read and review. This could lead to them being used incorrectly which might cause serious implementation issues. It is recommended that a suffix is used or that scientific notation is applied to make these areas of the code more readable.

UBIAuction

```
...

constructor() public{
  config["maxSupply"] = 1000;
  config["basePrice"] = 10000000000000000;
```

UBIToken

```
...
```

```
constructor () public ERC20("UBI Token", "UBI") {
    _mint(msg.sender, 1000000000 * 10 ...);
}
```

Further Information:

- https://docs.soliditylang.org/en/latest/types.html#rational-and-integer-literals

Observation 4: Public functions could be declared external (Optimization)

Public functions that are never called by the contract should be declared external to save gas. Below lists those functions that were flagged and it is recommended that the development team review each function and assign the appropriate visibility.

Auction Contract:

- Auction.update()
- Auction.bid(address,uint256)
- Auction.complete()
- Auction.status()
- Auction.ended()
- Auction.startInt()
- Auction.allbidders()
- Auction.tokensOf(address)
- Auction.bidOf(address)
- Auction.sumAuc()
- Auction.sumAdd(address)

UBIAuctions Contract:

- UBIAuctions.status()
- UBIAuctions.update()
- UBIAuctions.ended()
- UBIAuctions.bid()
- UBIAuctions.create()
- UBIAuctions.complete()
- UBIAuctions.setConfig(uint256[])
- UBIAuctions.startInt()
- UBIAuctions.allbidders()
- UBIAuctions.getConfig()
- UBIAuctions.sumAuc(uint256)
- UBIAuctions.sumAdd(uint256,address)
- UBIAuctions.tokensOf(address)
- UBIAuctions.bidOf(address)

UBIToken Contract

- UBIToken.createStake(uint256)
- UBIToken.removeStake(uint256)
- UBIToken.stakeOf(address)
- UBIToken.totalStakes()
- UBIToken.totalStakeHolders()
- UBIToken.rewardOf(address)
- UBIToken.totalRewards()
- UBIToken.distributeRewards()
- UBIToken.withdrawReward(address)

Observation 5: Deprecated SafeMath Operations

While reviewing the Library contracts provided, it was noticed that certain SafeMath operations have been deprecated by OpenZeppelin.

The operations that have been deprecated:

- sub(uint256 a, uint256 b, string memory errorMessage)
- div(uint256 a, uint256 b, string memory errorMessage)
- mod(uint256 a, uint256 b, string memory errorMessage)

According to the documentation, these functions have been deprecated because it requires allocating memory for the error message unnecessarily. For custom revert reasons use tryDiv.

- https://github.com/OpenZeppelin/openzeppelin-contracts/blob/c34211417cec0d2bdf68410ade9bce1d962ccaf0/contracts/math/SafeMath.sol#L151

Also, it was noticed that the implemented mod function (Within the utilized SafeMath contract L41) makes use of a != 0 operator opposed to a > 0 operator. If you are not expecting negative values, this should be replaced with the latter alternative.

Observation 6: Plaintext traffic is Enabled For App

The Android application uses plaintext network traffic. When an app communicates with servers using a plaintext network traffic, such as HTTP, it could raise a risk of eavesdropping and tampering of content. Third parties can inject unauthorized data or leak information about the users. The solution to fix this is just change the `useCleartextTraffic` attribute in manifest file to `false`.

```
<application
 android:usesCleartextTraffic="false"
```

Observation 7: Information Leakage via Shared clipboard

The Android application allows users to copy from and paste to text fields. The copied content is stored in the shared clipboard/pasteboard by Android.

Attackers can steal this data if they have physical access to the user's phone or over the network using any administration tools running on the device. The code review revealed that the Falcon Mobile application does not disable copy/paste on any of the available fields. Since the copied content is in a shared buffer, other applications of the device can read it.

The best solution to save data from being leaked via the copy/paste feature is to clean-up the copied content after use or to disable the copy-paste feature on sensitive fields. The copied buffer can be cleared after a timeout of 15 to 30 seconds.

Observation 8: Application is signed with v1 signature scheme, making it vulnerable to Janus Vulnerability:

The Android applications are self-signed, signature verification is important when updating Android applications. When the user downloads an update of an application, the Android runtime compares its signature with the signature of the original version. If the signatures match, the Android runtime proceeds to install the update. The updated application inherits the permissions of the original application. Attackers can, therefore, use the Janus vulnerability (a new flaw discovered in Android apps that is capable of modifying the Android apps code without making any alteration in their signatures) to mislead the update process and get unverified code with powerful permissions installed on the devices of unsuspecting users.

The Janus vulnerability affects recent Android devices (Android 5.0 and newer). Applications that have been signed with APK signature scheme v2 and that are running on devices supporting the latest signature scheme (Android 7.0 and newer) are protected against the vulnerability. Unlike scheme v1, this scheme v2 considers all bytes in the APK file. Older versions of applications and newer applications running on older devices remain susceptible. Developers should at least always apply signature scheme v2.

For more information, refer to the following:

- https://github.com/jcrutchvt10/Janus

Observation 9: Division Before Multiplication

While reviewing the smart contracts, it was noticed that the order of operations handling certain calculations within various functions, do not follow recommended best practices. Within Solidity, primarily earlier versions prior to 0.4.0, it was possible for division to return a truncated value. After version 0.4.0, the return value is converted into a rational number mitigating the issue. Although this does not appear to be a direct security concern, based on the target version of solidity, it is important to mention if contracts are compiled with older version. And if that is the case, it is recommended to move multiplication operations before division operations. Further information about the return types can be found in the documentation:

- https://docs.soliditylang.org/en/v0.8.1/types.html

The following code demonstrates this:

Auction

```
...
constructor(uint256[] memory _aucVal) public{
...
```

```
  aucVal["reservePrice"] = _aucVal[1] - ((_aucVal[2] / _aucVal[3]) * _aucVal[4]);
...

function complete() public{
...
  uint256 price = aucVal["avgPr"] / bidders.length;
  ...
  if ((tokens[bidder] * price) < amounts[bidder])
...
```

UBIAuctions

```
...
function create() public{
...
  aucVal[4] = (aucVal[1] / 100) * config["dropPercent"];
...

function getConfig() public view returns(uint256, ...
...
  initPrice = config["basePrice"] / 100;
  ...
  initPrice = initPrice * config["incrPercent"] * config["incrLevel"];
  ...
  uint256 dropPrice = (initPrice / 100) * config["dropPercent"];
...
```

Executed Test Cases

The following table shows the breakdown of executed test cases, including any problem reports relevant to that item, and gives a brief summary of the methodology used to check that item and any other observations.

Column descriptions are as follows:

- Id - An identifier for quick test case reference
- Title - A title describing the test case
- Description - A short description of the test case and why it was performed
- Outcome - Either 'Pass' or a reference to the Problem Report Number

Smart Contract Test Cases

Test ID	Test Title	Test Description	Outcome
1	Assert Violation	(SWC-110) As a best practice, assertion failures should never occur on live code.	**PASS**
2	Audited Dependencies	Verify that all dependencies have been audited, such as OpenZeppelin's contracts.	**OBS5**
3	BlockHash for Current/Future Block	Check that the blockhash(block.now) or greater is not used since it returns 0.	**PASS**
4	BlockHash for Old Block	Check that the blockhash of a block 256+ blocks ago is not used since it returns 0.	**PASS**
5	Business Logic Attacks	Confirm that the application's business logic or functionality cannot be used in unintended ways for malicious purposes.	**PASS**
6	Comments	Confirm dangerous areas of code are commented as such.	**PASS**
7	Compiler	(SWC-102) Check that the contract was compiled with the most recent solidity compiler.	**PR1**
8	Constructor Declaration	(SWC-118) Check that contracts written in Solidity v0.4.22 or greater use the constructor declaration, and that older ones do not contain a typo in the function name.	**PASS**
9	Cryptography Strength	Verify that all cryptographic methods and libraries used by the application use appropriate defaults and follow industry best standards.	**PASS**
10	Data in Code	Confirm that no sensitive information (including embedded encryption keys/application codes, etc.) is contained in client code.	**PASS**
11	Delegatecall Storage Slot Matching	(SWC-124) Check that implementations that utilize delegatecall correctly maintain storage slot ordering.	**PASS**

Test ID	Test Title	Test Description	Outcome
12	Deprecated Syntax	Confirm that code syntax adheres to current best practices.	**OBS1 OBS3 OBS9**
13	Denial of Service - External Contracts	(SWC-113) Check that an operation does not depend on successfully calling another contract.	**PASS**
14	Function Visibility	(SWC-100) Check that all functions are appropriately labeled as Public, Private, External, or Internal.	**OBS4**
15	Integer Overflow and Underflow	(SWC-101) Check that the safe math library is used and that arithmetic operations are done with .sub, .add, .mul, .div.	**PASS**
16	Known Vulnerabilities	Confirm that any software with disclosed version information is not vulnerable to known security vulnerabilities.	**PR3**
17	Missing Withdraw Functionality	Confirm that all payable contracts support a withdraw functionality to prevent funds being locked.	**PR2**
18	Missing Revert Messages	Confirm that all reverts, requires, asserts, etc. include string explanations, as enabled in solidity 0.4.22.	**PASS**
19	Optimizations	Check that the contract is sufficiently optimized to not waste gas.	**OBS4**
20	Randomness	(SWC-120) Check that a source of Randomness is not predictable.	**PASS**
21	Re-entrency	(SWC-107) Check that .call() is avoided when possible, and that value sends happen at the end of the function.	**PASS**
22	Safemath Calculation Stored	Ensure that safemath operations store the result in a variable.	**PASS**
23	SPDX Identifier	Confirm all contracts have the necessary SPDX identifiers included.	**OBS2**
24	Timestamp Operations	(SWC-116) Check that block.timestamp is not checked for precision in any trusted operations.	**PASS**
25	Timing Attacks in ERC20 - Approve	(SWC-114) Confirm that a contract does not unsafely set the 'approve' value of an ERC20 token such that it can be double spent.	**PASS**
26	Tx.origin Authentication	(SWC-115) Check that tx.origin is not used for authorization in a contract.	**PASS**
27	Unchecked Return Values	(SWC-104) Check that the return values of .call(...), .callcode(...), .send(...), and .delegetecall(...), and are always validated.	**PASS**
28	Unexpected Balance	Check that a contract with a balance that skips the fallback (selfdestruct or precomputed address) does not affect the business logic.	**PR2**

Test ID	Test Title	Test Description	Outcome
29	Unexpected Consequence of Uninitialized Memory	Confirm that an uninitialized variable defaulting to 0 does not mean anything (such as a 0 index in an array).	PASS
30	Unprotected Self Destruct	(SWC-106) Confirm that selfdestruct cannot be called by an unauthorized user.	PASS
31	Unsafe Assembly	(SWC-127) Verify that all uses of direct assembly are safe.	PASS
32	Unsafe Delegatecall	(SWC-112) Check that delegatecall is only used against trusted contracts.	PASS
33	Use of Deprecated Functions	(SWC-111) Confirm that no deprecated functions are in use, including: suicide(...), block.blockhash(...), sha3(...), callcode(...), throw, msg.gas, and constant.	PASS
34	Variable Visibility	(SWC-108) Check that all contract variables are appropriately labeled as Public, Private, External, or Internal.	PASS

Android APK Test Cases:

Test ID	Test Title	Test Description	Outcome
1	Sensitive Data in Code	Confirm that no sensitive information (including embedded encryption keys/application codes, etc.) is contained in the mobile code and accessible via decompiling (e.g. using apktools).	PASS
2	Android Clipboard	Confirm that sensitive fields do not allow copy-paste functionality.	Observation 7
3	Application can be debugged	Ensure that a debugger cannot be attached to the application and the code debugged step by step.	PASS
4	Android Backup Vulnerability	Ensure that backups are not enabled for the application. If backup is enabled, make sure it does not reveal sensitive information - or can not be exploited to perform malicious operations.	PR4
5	Plaintext Secrets	Confirm that the mobile application does not store usernames, passwords, encryption keys, challenge questions and answers, application codes, etc in plaintext.	PASS
6	Vulnerable Activity Components	Ensure that the activities in the application are not public and not directly callable by any other malicious APK. Verify that activities cannot be called when the screen is locked by the user.	PASS

Test ID	Test Title	Test Description	Outcome
7	Intent Sniffing	Identify if the application is using intents to send sensitive information to other applications. If it is, ensure that explicit intents are used or permissions are set correctly.	**PASS**
8	Flawed Broadcast Receivers	Ensure that broadcast receivers that listen for specific intents have permissions set so that they accept intents only from specific applications.	**PASS**
9	Insecure Content Provider Access	Ensure that content providers are not public and not callable directly by any other malicious APK. Also ensure that the provider is secure against SQL injection and path traversal.	**PASS**
10	Cleartext Traffic	Confirm that the mobile application does not communicate to server using cleartext traffic such as HTTP.	**Observation 6**
11	Hard coded credentials in source	Ensure that no sensitive information is being stored in binaries or source code.	**PASS**
12	Misuse of App permissions	Confirm that the permissions define in android manifest cannot be exploitable.	**PASS**
13	Code Signing	Ensure that application is not signed using the vulnerable schemes.	**Observation 8**
14	Abusing URL schemes	Identify URL schemes through source code or apk file.	**PASS**

Tools

While testing Security Audit, the following tools were employed:

Tool	Description	Link
Remix	Solidity IDE	https://remix.ethereum.org
Slither	Static Analyzer for Solidity	https://github.com/crytic/slither
apktool	APK reverse engineering and disassembly tool	https://ibotpeaches.github.io/Apktool/
jadx	Dex to Java decompiling tool	https://github.com/skylot/jadx

Next Steps

This section contains our recommendations for areas that may benefit from additional testing. For each section, we describe why it is important to test these sections, either more thoroughly or for the first time.

Retest After Remediation - A retest of the system is recommended to be performed when the problems found as a result of this test have been remediated. This validates the remediation put in place and ensures that other vulnerabilities have not been introduced in the course of remediation.

Full Test - The focus of this test was on a subset of smart contracts, supporting blockchain components, and the current version of the Falcon Android APK. Security Innovation recommends a full test of the UBIshares system and all integrations. Because of the number of features, it is important that all of these elements are analyzed and receive security testing to ensure their added functionality and other interactions do not put the rest of the system at risk. Each one of these modules/apps adds to the overall attack surface of the application and, as such, should undergo security testing.