Geg Finance Smart Contract
Preliminary Audit Report

Project Synopsis

Project Name	**Geg Finance**
Platform	Ethereum, Solidity
Github Repo	Not Provided
Deployed Contract	Not Deployed
Total Duration	5 days
Timeline of Audit	**7th June 2021 to 12th june 2021**

Contract Details

Total Contract(s)	5
Name of Contract(s)	Bank.sol, GErc20.sol, GEther.sol, OracleV1.sol, Queue.sol
Language	Solidity
Commit Hash	**Null**

Contract Vulnerabilities Synopsis

Issues	Open Issues	Closed Issues
Critical Severity	2	0
Medium Severity	3	0
Low Severity	7	0
Information	4	0
Total Found	16	0

Detailed Results

The contract has gone through several stages of the audit procedure that includes structural analysis, automated testing, manual code review etc.

All the issues have been explained and discussed in detail below. Along with the explanation of the issue found during the audit, the recommended way to overcome the issue or improve the code quality has also been mentioned.

A. Contract Name: Bankable

High Severity Issues

A.1 Floating Pragma Issue. Compiler version is not fixed

Contracts - Bank.sol, Gerc20.sol, Gether.sol, OracleV1.sol, Queue.sol

SWC Reference : SWC 103 – Floating Pragma

Description:

The pragma solidity version of the above mentioned contracts have not been fixed to a specific solidity version.

Keeping in mind the fact that different solidity versions include various new changes and modifications that might not be compatible with the older versions, it is always considered as a better practice to lock the Solidity version of a contract to a specific version.

For instance, as per the Solidity version 0.7, *derived contracts no longer inherit libraries using declarations for types (e.g. using SafeMath for uint). Instead, such declarations must be repeated in every derived contract that wishes to use the library for a type.*

However, the GErc20.sol and Gether.sol contracts don't include the SafeMath declarations for **uint256** type despite the fact that they use the SafeMath functions. This will lead to Compilation errors while compiling with Solidity version 0.7.

Recommendation:

Pragma solidity version must be locked to a specific version to ensure that the contract doesn't use any outdated version and it does follow all the rules of the specific version being used.

Medium Severity Issues

A.2 Multiplication is being performed on the result of Division

Line no - 268

Explanation:

During the automated testing of the **Bankable** contract, it was found that the ***_applyFine*** function in the contract is performing multiplication on the result of a Division.
Integer Divisions in Solidity might truncate. Moreover, this performing division before multiplication might lead to loss of precision.

The following functions involve division before multiplication in the mentioned lines:

- ***_applyFine*** *at 268*

```
268                 _amount = _amount.div(100).mul(100 - fine);
```

Recommendation:

Solidity doesn't encourage arithmetic operations that involve division before multiplication. Therefore the above-mentioned function should be checked once and redesigned if they do not lead to expected results.

Low Severity Issues

A.3 External Visibility should be preferred

Explanation:

Those functions that are never called throughout the contract should be marked as ***external*** visibility instead of ***public*** visibility.
This will effectively result in Gas Optimization as well.

Therefore, the following function must be marked as **external** within the contract:

- ***accrueInterestAll()***
- ***SetAutoRenewal()***
- ***setOracle()***

Recommendation:

If the PUBLIC visibility of the above-mentioned functions is not intended, then the EXTERNAL Visibility keyword should be preferred.

A.4 Return Value of an External Call is never used Effectively

Line no -378

Explanation:

The external calls made in the above-mentioned line do return a boolean value that indicates whether or not the external call made was successful.

These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.

However, the ***_accrueInterest function*** in the **Bankable** contract never uses these return values.

```
378         tokenContract.transferFrom(owner(), client, amount);
379
380         return true;
381     }
```

Recommendation:

Effective use of all the return values from external calls must be ensured within the contract.

Informational

A.5 Coding Style Issues in the Contract

Explanation:

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

```
Function Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle) (flat/flat_bank.sol#1649-1674) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._name (flat/flat_bank.sol#1650) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._symbol (flat/flat_bank.sol#1651) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._term (flat/flat_bank.sol#1652) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._interest (flat/flat_bank.sol#1653) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._fine (flat/flat_bank.sol#1654) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._tokenContract (flat/flat_bank.sol#1655) is not in mixedCase
Parameter Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._currencyOracle (flat/flat_bank.sol#1656) is not in mixedCase
```

During the automated testing, it was found that the **Bankable** contract had quite a few code style issues.

Recommendation:

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

B. Contract Name: GErc20

High Severity Issues

B.1 Contract completely locks Ether and fails to provide a way to Unlock it

Line no: 54, 131-134,

Description:

The **GErc20 contract** includes a few functions with the **payable** keyword. It indicates that the contract allows the transfer of **ETHER** into the contract.

```
131     function donate(uint256 _amount) external payable {
132         _getValue(_amount);
133         _payout();
134     }
```

However, during the audit procedure, it was found that the contract fails to provide a way to withdraw the locked ETHER in the contract.

This could lead to a very undesirable situation where any Ether sent to the contract will be completely lost and unrecoverable.

Moreover, a similar warning was found while conducting the automated testing of the contract.

```
Contract locking ether found in :
        Contract GErc20 (flat/flat_GErc20.sol#1861-2077) has payable functions:
         - GErc20.receive() (flat/flat_GErc20.sol#1900)
         - GErc20.donate(uint256) (flat/flat_GErc20.sol#1977-1980)
        But does not have a function to withdraw the ether
```

Recommendation:

The contract should either remove the **payable keyword** from the **above-mentioned functions** or include a function that allows the withdrawal of the locked ETHER in the contract.

Low Severity Issues

B.2 Return Value of an External Call is never used Effectively

Line no

Explanation:

The external calls made in the above-mentioned line do return a boolean value that indicates whether or not the external call made was successful.

These boolean return values can be used in the function as a check to ensure that the further execution of the function is only allowed if the external is successfully made.

However, the following function in the ***GErc20*** contract never uses these return values:

- ***makeWithdrawal at Line 109***
- ***withdraw at Line 122***
- ***_getValue at Line 152***
- ***_payout at Line 228***

Recommendation:

Effective use of all the return values from external calls must be ensured within the contract.

B.3 Absence of Zero Address Validation

Line no- 138

Explanation:

The **GErc20** Contract includes a function that updates an imperative address in the contract, i.e, ***underlying***.

However, during the automated testing of the contact it was found that no Zero Address Validation is implemented on the following functions while updating the address state variables of the contract:

- ***setUnderlying at Line 138***

Recommendation:

A **require** statement should be included in such functions to ensure no zero address is passed in the arguments.

B.4 Comparison to boolean Constant

Line no: 73,76

Explanation:

Boolean constants can directly be used in conditional statements or require statements.

Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require** statements.

- ***makeWithdrawal()***

```
73          require(_deposit.active == true, "Deposit is closed.");
74          _isMsgSender(_deposit.client);
75          // require(_deposit.client == msg.sender, "Owner missmatch.");
76          require(_deposit.claimed == false, "Deposit is already claimed.");
```

Recommendation:

The equality to boolean constants must be removed from the above-mentioned line. This enhances code readability and saves gas.

Informational

B.5 Coding Style Issues in the Contract

Explanation:

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the **GErc20** contract had quite a few code style issues.

```
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._name (flat/flat_GErc20.sol#1875) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._symbol (flat/flat_GErc20.sol#1876) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._term (flat/flat_GErc20.sol#1877) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._interest (flat/flat_GErc20.sol#1878) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._fine (flat/flat_GErc20.sol#1879) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._tokenContract (flat/flat_GErc20.sol#1880) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._currencyOracle (flat/flat_GErc20.sol#1881) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._underlying (flat/flat_GErc20.sol#1882) is not in mixedCase
Parameter GErc20.deposit(uint256,bool)._amount (flat/flat_GErc20.sol#1909) is not in mixedCase
```

Recommendation:

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

C. Contract Name: GEther

Medium Severity Issues

C.1 Internal function is never used within the contract

Line no - 139

Explanation:

The GEther contract includes an internal function, **getValue (***Line 139***),** that is never used within the contract.

A similar function is already available in the Bank.sol contract which performs an exact similar task and is being inherited by the Gether.sol contract already.

```
139     function _getValue(uint256) internal override returns (uint256) {
140         return msg.value;
```

Recommedation:

If the above-mentioned function doesn't hold any signifcance, it should be removed from the contract.

Low Severity Issues

C.2 Comparison to boolean Constant

Line no: 73,76

Explanation:

Boolean constants can directly be used in conditional statements or require statements.

Therefore, it's not considered a better practice to explicitly use **TRUE or FALSE** in the **require** statements.

- ***makeWithdrawal()***

```
70      function makeWithdrawal(uint256 id) external {
71          _hasDeposit(id);
72
73          Deposit storage _deposit = deposits[id];
74          require(_deposit.active == true, "Deposit is closed.");
75          _isMsgSender(_deposit.client);
76          require(_deposit.claimed == false, "Deposit is already claimed.");
77
```

Recommendation:

The equality to boolean constants must be removed from the above-mentioned line. This enhances code readability and saves gas.

D. Contract Name: OracleV1

Medium Severity Issues

D.1 Absence of Input Validations in _setRate function

Line no - 135-139

Description:

The **OracleV1** contract includes quite a few functions that update some crucial state variables of the contract.

However, no input validation is included in any of those functions.

Is this intended?

This might lead to an unwanted scenario where an invalid or wrong argument is passed to the function that could badly affect the expected behavior of the contract.

```
135     function _setRate(address _address↑, uint256 _amount↑) internal {
136         rates[_address↑] = _amount↑;
137         updated[_address↑] = block.timestamp;
138     }
139 }
```

Recommendation:

Arguments passed to such functions must be validated before being used to update the State variable.

Low Severity Issues

D.2 Absence of Zero Address Validation

Line no- *118*

Explanation:

The **Oraclev1** Contract includes a function that updates an imperative address in the contract, i.e, ***baseToken***.

However, during the automated testing of the contact it was found that no Zero Address Validation is implemented on the following functions while updating the address state variables of the contract:

- ***setToken at Line 118***

```
118     function setToken(address _address↑) external onlyOwner {
119         baseToken = _address↑;
```

Recommendation:

A **require** statement should be included in such functions to ensure no zero address is passed in the arguments.

Informational

D.3 Commented code Issue

Line no- 37-42

Explanation

The OracleV1 contract includes quite a few commented codes regarding the contract.

This badly affects the readability of the code.

```
37      // /**
38      //  * @dev Set ETH<->GEG exchange rate from signed message
39      //  * It will be used to allow other users set exchange rate if they were allowed by owner
40      //  */
41      // function setEthRateSigned(bytes32 hash, bytes memory signature)  external{
42      // }
```

Recommendation:

If these instances of code are not required in the current version of the contract, then the commented codes must be removed before deployment.

D.4 Coding Style Issues in the Contract

Explanation:

Code readability of a Smart Contract is largely influenced by the Coding Style issues and in some specific scenarios may lead to bugs in the future.

During the automated testing, it was found that the **OracleV1** contract had quite a few code style issues.

```
Parameter OracleV1.convert(address,uint256)._amount (flat/flat_oracleV1.sol#516) is not in mixedCase
Parameter OracleV1.setRate(address,uint256)._address (flat/flat_oracleV1.sol#535) is not in mixedCase
Parameter OracleV1.setRate(address,uint256)._amount (flat/flat_oracleV1.sol#535) is not in mixedCase
Parameter OracleV1.setRateSigned(address,uint256,uint256,bytes)._address (flat/flat_oracleV1.sol#553) is not in mixedCase
Parameter OracleV1.setRateSigned(address,uint256,uint256,bytes)._amount (flat/flat_oracleV1.sol#554) is not in mixedCase
Parameter OracleV1.setRateSigned(address,uint256,uint256,bytes)._timestamp (flat/flat_oracleV1.sol#555) is not in mixedCase
Parameter OracleV1.setRateSigned(address,uint256,uint256,bytes)._sig (flat/flat_oracleV1.sol#556) is not in mixedCase
Parameter OracleV1.setToken(address)._address (flat/flat_oracleV1.sol#584) is not in mixedCase
```

Recommendation:

Therefore, it is recommended to fix the issues like naming convention, indentation, and code layout issues in a smart contract.

E. Contract Name: Queue

No Issues Found

Automated Test Results

```
Contract locking ether found in :
        Contract GErc20 (flat/flat_GErc20.sol#1861-2077) has payable functions:
         - GErc20.receive() (flat/flat_GErc20.sol#1900)
         - GErc20.donate(uint256) (flat/flat_GErc20.sol#1977-1980)
        But does not have a function to withdraw the ether
```

```
GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._name (flat/flat_GErc20.sol#1875) shadows:
        - ERC20Upgradeable._name (flat/flat_GErc20.sol#522) (state variable)
GErc20.initialize(string,string,uint256,uint256,uint256,GEG,Oracle,address)._symbol (flat/flat_GErc20.sol#1876) shadows:
        - ERC20Upgradeable._symbol (flat/flat_GErc20.sol#523) (state variable)
```

```
GErc20._payout() (flat/flat_GErc20.sol#2045-2076) has costly operations inside a loop:
        - claimValue = claimValue.sub(amount) (flat/flat_GErc20.sol#2064)
GErc20._payout() (flat/flat_GErc20.sol#2045-2076) has costly operations inside a loop:
        - depositValue = depositValue.sub(amount) (flat/flat_GErc20.sol#2068)
GErc20._payout() (flat/flat_GErc20.sol#2045-2076) has costly operations inside a loop:
        - depositIndex = depositIndex.sub(1) (flat/flat_GErc20.sol#2069)
```

```
Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._name (flat/flat_bank.sol#1650) shadows:
        - ERC20Upgradeable._name (flat/flat_bank.sol#449) (state variable)
Bankable.__init(string,string,uint256,uint256,uint256,GEG,Oracle)._symbol (flat/flat_bank.sol#1651) shadows:
        - ERC20Upgradeable._symbol (flat/flat_bank.sol#450) (state variable)
```

```
Reentrancy in Bankable.accrueInterestOneWithRates(uint256,uint256,uint256,bytes) (flat/flat_bank.sol#1562-1572):
        External calls:
        - success = currencyOracle.setRateSigned(underlying,_amount,_ts,_sig) (flat/flat_bank.sol#1568-1569)
        - _accrueInterest(_id) (flat/flat_bank.sol#1571)
                - tokenContract.transferFrom(owner(),client,amount) (flat/flat_bank.sol#1862)
        Event emitted after the call(s):
        - LogAccruedInterest(client,_id,amount) (flat/flat_bank.sol#1861)
```