

GEG FINANCE

Smart Contract Audit

Final Report



June 24, 2021

Short Executive Summary:

While conducting the final audit of Geg Finance smart contracts, it was observed that the team closed High, Medium, and Low severity issues.

The Geg Finance team has fixed the issues based on the auditor's recommendation.

Introduction	3
About GEG FINANCE	3
About ImmuneBytes	3
Documentation Details	3
Audit Process & Methodology	4
Audit Details	4
Audit Goals	5
Security Level References	5
Contract Name: Bankable	6
High Severity Issues	6
Medium Severity Issues	7
Low Severity Issues	7
Informational	8
Contract Name: GErc20	9
Low Severity Issues	9
Informational	11
Contract Name: GEther	11
Medium Severity Issues	11
Low Severity Issues	12
Contract Name: OracleV1	12
Medium Severity Issues	12
Low Severity Issues	13
Contract Name: Queue	14
Test Cases	15
Automated Audit	19
Concluding Remarks	20
Disclaimer	20

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Introduction

1. About GEG FINANCE

GEG.FINANCE is a unique DeFi project with a brand new approach that allows its clients to earn up to 28% APY on financing the construction of the photovoltaic parks of a strong and already operational solar business. Token sale participants have access to products with up to 58% APY.

About the group:

The project is backed by Green Energy Group (GEG), an independent EU renewable energy production, supply, operation, and maintenance company. GEG.FINANCE offers a unique opportunity for participants of the token sale to start earning immediately upon entering the project via making a deposit for a period of 9 months or more.

2. About ImmuneBytes

ImmuneBytes is a security start-up to provide professional services in the blockchain space. The team has hands-on experience in conducting smart contract audits, penetration testing, and security consulting. ImmuneBytes's security auditors have worked on various A-league projects and have a great understanding of DeFi projects like AAVE, Compound, 0x Protocol, Uniswap, dydx.

The team has been able to secure 15+ blockchain projects by providing security services on different frameworks. ImmuneBytes team helps start-up with a detailed analysis of the system ensuring security and managing the overall project.

Visit <http://immunebytes.com/> to know more about the services.

Documentation Details

For the purpose of audit, the Geg.Finance team has provided documents as follows:

1. PW Final (2).pdf
2. GEG_2pager.pdf

Audit Process & Methodology

ImmuneBytes team has performed thorough testing of the project starting with analyzing the code design patterns in which we reviewed the smart contract architecture to ensure it is structured and safe use of third-party smart contracts and libraries.

Our team then performed a formal line-by-line inspection of the Smart Contract in order to find any potential issues like Signature Replay Attacks, Unchecked External Calls, External Contract Referencing, Variable Shadowing, Race conditions, Transaction-ordering dependence, timestamp dependence, DoS attacks, and others.

In the Unit testing phase, we run unit tests written by the developer in order to verify the functions work as intended. In Automated Testing, we tested the Smart Contract with our in-house developed tools to identify vulnerabilities and security flaws.

The code was audited by a team of independent auditors which includes -

1. Testing the functionality of the Smart Contract to determine proper logic has been followed throughout.
2. Analyzing the complexity of the code by thorough, manual review of the code, line-by-line.
3. Deploying the code on testnet using multiple clients to run live tests.
4. Analyzing failure preparations to check how the Smart Contract performs in case of bugs and vulnerabilities.
5. Checking whether all the libraries used in the code are on the latest version.
6. Analyzing the security of the on-chain data.

Audit Details

- Project Name: GEG FINANCE
- Languages: Solidity(Smart contract)
- Github commit hash for audit: [ac0c8a481c795488f4af7f71d672818e6209bc43](#)
- Github commit hash for audit: [062cc6268e4533549cc38b264d62c17c3874a6aa](#)
- Platforms and Tools: Remix IDE, Truffle, Truffle Team, Ganache, Solhint, VScode, Contract Library, Slither, SmartCheck

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Audit Goals

The focus of the audit was to verify that the smart contract system is secure, resilient, and working according to its specifications. The audit activities can be grouped into the following three categories:

1. Security: Identifying security-related issues within each contract and within the system of contracts.
2. Sound Architecture: Evaluation of the architecture of this system through the lens of established smart contract best practices and general software best practices.
3. Code Correctness and Quality: A full review of the contract source code. The primary areas of focus include:
 - a. Correctness
 - b. Readability
 - c. Sections of code with high complexity
 - d. Quantity and quality of test coverage

Security Level References

Every issue in this report was assigned a severity level from the following:

High severity issues will bring problems and should be fixed.

Medium severity issues could potentially bring problems and should eventually be fixed.

Low severity issues are minor details and warnings that can remain unfixed but would be better fixed at some point in the future.

Issues	<u>High</u>	<u>Medium</u>	<u>Low</u>
Open	-	-	-
Closed	1	3	7

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Contract Name: Bankable

High Severity Issues

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](#).

Medium Severity Issues

1. Multiplication is being performed on the result of Division

Line no - 268

Description:

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/Bankable/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

Low Severity Issues

1. External Visibility should be preferred

Description:

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](#).

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

Line no -378

Description:

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](#).

Informational

1. Coding Style Issues in the Contract**Description:**

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,GE6,Oracle) (flat/flat_bank.sol#1649-1674) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,Oracle)._name (flat/flat_bank.sol#1650) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,Oracle)._symbol (flat/flat_bank.sol#1651) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,Oracle)._term (flat/flat_bank.sol#1652) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,Oracle)._interest (flat/flat_bank.sol#1653) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,Oracle)._fine (flat/flat_bank.sol#1654) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,Oracle)._tokenContract (flat/flat_bank.sol#1655) is not in mixedCase
Parameter Bankable._init(string,string,uint256,uint256,uint256,GE6,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.

Contract Name: GErc20

Low Severity Issues

1. Return Value of an External Call is never used Effectively

Description:

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/GErc20/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

2. Absence of Zero Address Validation

Line no- 138

Description:

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

However, during the automated testing of the contract 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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/GErc20/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

3. Comparison to boolean Constant

Line no: 73,76

Description:

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 mismatch.");
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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/GErc20/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

Informational

1. Coding Style Issues in the Contract

Description:

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,GE6,Oracle,address)._name (flat/flat_GErc20.sol#1875) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,Oracle,address)._symbol (flat/flat_GErc20.sol#1876) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,Oracle,address)._term (flat/flat_GErc20.sol#1877) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,Oracle,address)._interest (flat/flat_GErc20.sol#1878) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,Oracle,address)._fine (flat/flat_GErc20.sol#1879) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,Oracle,address)._tokenContract (flat/flat_GErc20.sol#1880) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,Oracle,address)._currencyOracle (flat/flat_GErc20.sol#1881) is not in mixedCase
Parameter GErc20.initialize(string,string,uint256,uint256,uint256,GE6,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.

Contract Name: Gether

Medium Severity Issues

1. Internal function is never used within the contract

Line no - 139

Description:

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;
```

Recommendation:

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

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/062cc6268e4533549cc38b264d62c17c3874a6aa).

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Low Severity Issues

1. Comparison to boolean Constant

Line no: 73,76

Description:

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/GegFinance/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

Contract Name: OracleV1

Medium Severity Issues

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.

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/GegFinance/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

Low Severity Issues

1. Absence of Zero Address Validation

Line no- 118

Description:

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

However, during the automated testing of the contract 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.

Amended (Jun 24th 2021): Issue fixed by the **Geg Finance** team and is no longer present in commit [062cc6268e4533549cc38b264d62c17c3874a6aa](https://github.com/GegFinance/GegFinance/commit/062cc6268e4533549cc38b264d62c17c3874a6aa).

Informational

1. Commented code Issue

Line no- 37-42

Description:

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.

2. Coding Style Issues in the Contract

Description:

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.

Contract Name: Queue

No Issues Found

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Test Cases

```
Bank(Exchange)
  ✓ should call ETH oracle (433ms)
  ✓ should change Oracle (723ms)
  ✓ should prevent change Oracle by non-admin (42ms)

gERC20::Exchange::V3
  ✓ should call ETH oracle (177ms)
  ✓ should change Oracle (245ms)
  ✓ should prevent change Oracle by non-admin

gERC20::Interest::V3
  ✓ should emit Transfer log (94ms)
  ✓ should emit AccruedInterest log (93ms)
  ✓ should not emit AccruedInterest log if no interest (47ms)
  ✓ should transfer proper amount of tokens (1 year) (95ms)
  ✓ should not transfer proper amount of tokens (next day) (96ms)
  ✓ should transfer proper amount of tokens (2 days) (111ms)
  ✓ should transfer proper amount of tokens (14 hours) (98ms)
  ✓ should not transfer twice (142ms)
  ✓ should consider previous payouts (189ms)
  ✓ should transfer multiple accounts (127ms)
  ✓ should accrue interest after withdrawal (80ms)
  ✓ should not accrue if deposit is closed (100ms)
  ✓ should not allow zero underlying
renewal
  ✓ should not claim after expire (new) (81ms)
  ✓ should claim after expire (new) (81ms)
```

```
Bank::gERC20::v1
  owner only
    1) "before each" hook for "should have owner"
  deposits
    2) "before each" hook for "should start with 0 deposit"
  withdrawal
    3) "before each" hook for "should allow withdrawal only by owner"
  donate
    4) "before each" hook for "should increase balance"
  claims
    5) "before each" hook for "should start with no claims"

Bank::gETH::v1
  6) "before each" hook for "should deploy contract"

Bank(Interest)
  ✓ should emit Transfer log (118ms)
  ✓ should emit AccruedInterest log (109ms)
  ✓ should not emit AccruedInterest log if no interest (76ms)
  ✓ should not accrue if deposit is closed (136ms)

Bank::Interest::V2
  ✓ should emit Transfer log (97ms)
  ✓ should emit AccruedInterest log (71ms)
  ✓ should not emit AccruedInterest log if no interest (93ms)
  ✓ should transfer proper amount of tokens (1 year) (71ms)
  ✓ should not transfer proper amount of tokens (next day) (115ms)
  ✓ should transfer proper amount of tokens (2 days) (134ms)
  ✓ should transfer proper amount of tokens (14 hours) (127ms)
  ✓ should not transfer on same day (179ms)
  ✓ should consider previous payouts (174ms)
  ✓ should transfer multiple accounts (161ms)
  ✓ should accrue interest after withdrawal (104ms)
  ✓ should not accrue if deposit is closed (96ms)
renewal
  ✓ should not claim after expire (legacy) (115ms)
  ✓ should not claim after expire (new) (111ms)
  ✓ should claim after expire (new) (101ms)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
Signature
7) "before each" hook for "should accept valid message"

Oracle::v1
✓ should raise if no conversion rate
✓ should update and convert ETH

Oracle::v2
ETH
8) "before each" hook for "should raise if no conversion rate"

Bank (Proxy)
9) should deploy proxy
10) should set underlying

Bank::Deposits::Renewal
deposits
11) "before each" hook for "should change deposit autorenewal"

42 passing (23s)
11 failing

1) Bank::gERC20::v1
   owner only
   "before each" hook for "should have owner":
     HardhatError: HH701: There are multiple artifacts for contract "Gerc20", please use a fully qualified name.

Please replace Gerc20 for one of these options wherever you are trying to read its artifact:

contracts:Gerc20
contracts\Gerc20.sol:Gerc20
```

```
2) Bank::gERC20::v1
   deposits
   "before each" hook for "should start with 0 deposit":
     HardhatError: HH701: There are multiple artifacts for contract "Gerc20", please use a fully qualified name.

Please replace Gerc20 for one of these options wherever you are trying to read its artifact:

contracts:Gerc20
contracts\Gerc20.sol:Gerc20

   at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
   at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
   at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
   at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)

3) Bank::gERC20::v1
   withdrawal
   "before each" hook for "should allow withdrawal only by owner":
     HardhatError: HH701: There are multiple artifacts for contract "Gerc20", please use a fully qualified name.

Please replace Gerc20 for one of these options wherever you are trying to read its artifact:

contracts:Gerc20
contracts\Gerc20.sol:Gerc20

   at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
   at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
   at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
   at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.


```
4) Bank::gERC20::v1
  donate
  "before each" hook for "should increase balance":
  HardhatError: HH701: There are multiple artifacts for contract "Gerc20", please use a fully qualified name.
```

Please replace Gerc20 for one of these options wherever you are trying to read its artifact:

```
contracts:Gerc20
contracts\Gerc20.sol:Gerc20
```

```
at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

```
5) Bank::gERC20::v1
  claims
  "before each" hook for "should start with no claims":
  HardhatError: HH701: There are multiple artifacts for contract "Gerc20", please use a fully qualified name.
```

Please replace Gerc20 for one of these options wherever you are trying to read its artifact:

```
contracts:Gerc20
contracts\Gerc20.sol:Gerc20
```

```
at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

```
6) Bank::gETH::v1
  "before each" hook for "should deploy contract":
  HardhatError: HH701: There are multiple artifacts for contract "Gether", please use a fully qualified name.
```

Please replace Gether for one of these options wherever you are trying to read its artifact:

```
contracts:Gether
contracts\Gether.sol:Gether
```

```
at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

```
7) Signature
  "before each" hook for "should accept valid message":
  HardhatError: HH701: There are multiple artifacts for contract "Gerc20", please use a fully qualified name.
```

Please replace Gerc20 for one of these options wherever you are trying to read its artifact:

```
contracts:Gerc20
contracts\Gerc20.sol:Gerc20
```

```
at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

```
8) Oracle::v2
  "before each" hook for "should raise if no conversion rate":
  HardhatError: HH701: There are multiple artifacts for contract "ERC20", please use a fully qualified name.
```

Please replace ERC20 for one of these options wherever you are trying to read its artifact:

```
@openzeppelin\contracts\token\ERC20\ERC20.sol:ERC20
contracts:ERC20
```

```
at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

```
9) Bank (Proxy)
  should deploy proxy:
Error: Validations log not found. Recompile with `hardhat compile --force`
  at Object.readValidations (node_modules\@openzeppelin\hardhat-upgrades\src\validations.ts:30:13)
  at Proxy.deployProxy (node_modules\@openzeppelin\hardhat-upgrades\src\deploy-proxy.ts:39:25)

10) Bank (Proxy)
  should set underlying:
Error: Validations log not found. Recompile with `hardhat compile --force`
  at Object.readValidations (node_modules\@openzeppelin\hardhat-upgrades\src\validations.ts:30:13)
  at Proxy.deployProxy (node_modules\@openzeppelin\hardhat-upgrades\src\deploy-proxy.ts:39:25)

11) Bank::Deposits::Renewal
  deposits
    "before each" hook for "should change deposit autorenewal":
HardhatError: HH701: There are multiple artifacts for contract "GErc20", please use a fully qualified name.

Please replace GErc20 for one of these options wherever you are trying to read its artifact:

contracts:GErc20
contracts\GErc20.sol:GErc20

  at Artifacts._getArtifactPathFromFiles (node_modules\hardhat\src\internal\artifacts.ts:401:13)
  at Artifacts._getArtifactPath (node_modules\hardhat\src\internal\artifacts.ts:316:17)
  at Artifacts.readArtifact (node_modules\hardhat\src\internal\artifacts.ts:49:26)
  at getContractFactoryByName (node_modules\@nomiclabs\hardhat-ethers\src\helpers.ts:100:20)
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Automated Audit

```

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

- ERC20Upgradeable._symbol (flat/flat_GErc20.sol#523) (state variable)
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)
  
```

This audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Concluding Remarks

While conducting the final audit of Geg Finance smart contracts, it was observed that the team closed High, Medium, and Low severity issues. Resolving the areas of recommendations are up to the team's discretion. The recommendations given will improve the operations of the smart contract.

Disclaimer

ImmuneBytes's audit does not provide a security or correctness guarantee of the audited smart contract. Securing smart contracts is a multistep process, therefore running a bug bounty program as a complement to this audit is strongly recommended.

Our team does not endorse the Geg Finance platform or its product nor this audit is investment advice.

Notes:

- Please make sure contracts deployed on the mainnet are the ones audited.
- Check for the code refactor by the team on critical issues.

ImmuneBytes Pvt Ltd.