0.7

Gravity Finance Process Quality Review

Score: 72%

Overview

This is a Gravity Finance Process Quality Review completed on August 31st 2021. It was performed using the Process Review process (version 0.7.3) and is documented here. The review was performed by Nic of DeFiSafety. Check out our Telegram.

The final score of the review is **72%**, a **PASS**. The breakdown of the scoring is in Scoring Appendix. For our purposes, a pass is **70%.**

Summary of the Process

Very simply, the review looks for the following declarations from the developer's site. With these declarations, it is reasonable to trust the smart contracts.

- **Here are my smart contracts on the blockchain**
- **Here is the documentation that explains what my smart contracts do**
- **Here are the tests I ran to verify my smart contract**
- **Here are the audit(s) performed on my code by third party experts**
- **Here are the admin controls and strategies**

Disclaimer

views to change. All information is subject to possible correction. Information may quickly become unreliable for various reasons, including changes in market conditions or economic circumstances.

Chain

This section indicates the blockchain used by this protocol.

> ✓ **Chain:** Polygon

Guidance:

Ethereum
Binance Smart Chain
Polygon
Avalanche
Terra

Code and Team

This section looks at the code deployed on the Mainnet that gets reviewed and its corresponding software repository. The document explaining these questions is here. This review will answer the following questions:

1) Are the executing code addresses readily available? (%)
2) Is the code actively being used? (%)
3) Is there a public software repository? (Y/N)
4) Is there a development history visible? (%)
5) Is the team public (not anonymous)? (Y/N)

1) Are the executing code addresses readily available? (%)

> ✓ **Answer:** 100%

They are available at website https://inthenextversion.gitbook.io/gravity-finance/smart-contracts, as indicated in the Appendix.

Guidance:

100%	Clearly labelled and on website, docs or repo, quick to find
70%	Clearly labelled and on website, docs or repo but takes a bit of looking

40%	Addresses in mainnet.json, in discord or sub graph, etc
20%	Address found but labeling not clear or easy to find
0%	Executing addresses could not be found

2) Is the code actively being used? (%)

Answer: 100%

Activity is over 10 transactions a day on contract *TransparentUpgradeableProxy.sol* (Governance Contracts), as indicated in the Appendix.

Guidance:

100%	More than 10 transactions a day
70%	More than 10 transactions a week
40%	More than 10 transactions a month
10%	Less than 10 transactions a month
0%	No activity

3) Is there a public software repository? (Y/N)

Answer: Yes

GitHub: https://github.com/inthenextversion.

Is there a public software repository with the code at a minimum, but also normally test and scripts. Even if the repository was created just to hold the files and has just 1 transaction, it gets a **"Yes"**. For teams with private repositories, this answer is **"No"**.

4) Is there a development history visible? (%)

Answer: 0%

With 6 commits and 1 branch, this is a very underdeveloped software repository.

This metric checks if the software repository demonstrates a strong steady history. This is normally demonstrated by commits, branches and releases in a software repository. A healthy history demonstrates a history of more than a month (at a minimum).

Guidance:

100%	Any one of 100+ commits, 10+branches
70%	Any one of 70+ commits, 7+branches

50%	Any one of 50+ commits, 5+branches
30%	Any one of 30+ commits, 3+branches
0%	Less than 2 branches or less than 30 commits

How to improve this score:

Continue to test and perform other verification activities after deployment, including routine maintenance updating to new releases of testing and deployment tools. A public development history indicates clearly to the public the level of continued investment and activity by the developers on the application. This gives a level of security and faith in the application.

5) Is the team public (not anonymous)? (Y/N)

⚠ **Answer:** No

For a **"Yes"** in this question, the real names of some team members must be public on the website or other documentation (LinkedIn, etc). If the team is anonymous, then this question is a **"No"**.

Documentation

This section looks at the software documentation. The document explaining these questions is here.

Required questions are;

6) Is there a whitepaper? (Y/N)
7) Are the basic software functions documented? (Y/N)
8) Does the software function documentation fully (100%) cover the deployed contracts? (%)
9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)
10) Is it possible to trace from software documentation to the implementation in
code (%)

6) Is there a whitepaper? (Y/N)

✓ **Answer:** Yes

Location: https://inthenextversion.gitbook.io/gravity-finance/.

7) Are the basic software functions documented? (Y/N)

✓ **Answer:** Yes

The basic software functions are documented at https://github.com/inthenextversion/audit-gravity-ctdsec-v1-core/blob/main/Gravity%20Finance%20Code%20Brief.pdf.

8) Does the software function documentation fully (100%) cover the deployed contracts? (%)

✓ **Answer:** 100%

All of the Gravity Finance software functions and contracts are documented at https://github.com/inthenextversion/audit-gravity-ctdsec-v1-core/blob/main/Gravity%20Finance%20Code%20Brief.pdf.

Guidance:

100%	All contracts and functions documented
80%	Only the major functions documented
79-1%	Estimate of the level of software documentation
0%	No software documentation

9) Are there sufficiently detailed comments for all functions within the deployed contract code (%)

! **Answer:** 29%

Code examples are in the Appendix. As per the SLOC, there is 29% commenting to code (CtC).

The Comments to Code (CtC) ratio is the primary metric for this score.

Guidance:

100%	CtC > 100 Useful comments consistently on all code
90-70%	CtC > 70 Useful comment on most code
60-20%	CtC > 20 Some useful commenting
0%	CtC < 20 No useful commenting

How to improve this score

This score can improve by adding comments to the deployed code such that it comprehensively covers the code. For guidance, refer to the SecurEth Software Requirements.

10) Is it possible to trace from software documentation to the implementation in code (%)

! **Answer:** 40%

The Gravity Finance documentation lists and describes all of their functions without providing traceability as to their implementations in the source code.

Guidance:

100%	Clear explicit traceability between code and documentation at a requirement level for all code
60%	Clear association between code and documents via non explicit traceability
40%	Documentation lists all the functions and describes their functions
0%	No connection between documentation and code

How to improve this score:

This score can improve by adding traceability from documentation to code such that it is clear where each outlined function is coded in the source code. For reference, check the SecurEth guidelines on traceability.

Testing

This section looks at the software testing available. It is explained in this document. This section answers the following questions;

11) Full test suite (Covers all the deployed code) (%)
12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)
13) Scripts and instructions to run the tests (Y/N)
14) Report of the results (%)
15) Formal Verification test done (%)
16) Stress Testing environment (%)

11) Is there a Full test suite? (%)

✓ **Answer:** 80%

Code examples are in the Appendix. As per the SLOC, there is 107% testing to code (TtC).

The Test to Code (TtC) ratio is the primary metric for this score.

Guidance:

100%	TtC > 120% Both unit and system test visible
80%	TtC > 80% Both unit and system test visible
40%	TtC < 80% Some tests visible
0%	No tests obvious

How to improve this score:

This score can improved by adding tests to fully cover the code. Document what is covered by traceability or test results in the software repository.

12) Code coverage (Covers all the deployed lines of code, or explains misses) (%)

ⓘ **Answer:** 50%

There is no evidence of any Gravity Finance code coverage. However, they do have a reasonably complete set of tests.

Guidance:

100%	Documented full coverage
99-51%	Value of test coverage from documented results
50%	No indication of code coverage but clearly there is a reasonably complete set of tests
30%	Some tests evident but not complete
0%	No test for coverage seen

How to improve this score:

This score can improved by adding tests that achieve full code coverage. A clear report and scripts in the software repository will guarantee a high score.

13) Scripts and instructions to run the tests (Y/N)

⚠ **Answer:** No

There is no test result report to be found in the Gravity Finance GitHub.

How to improve this score:

Add the scripts to the repository and ensure they work. Ask an outsider to create the environment and run the tests. Improve the scripts and docs based on their feedback.

14) Report of the results (%)

⚠ **Answer:** 0%

No test results report was found in any of the Gravity Finance documentation.

Guidance:

100%	Detailed test report as described below
70%	GitHub code coverage report visible
0%	No test report evident

How to improve this score

Add a report with the results. The test scripts should generate the report or elements of it.

15) Formal Verification test done (%)

> ⚠ **Answer:** 0%

No evidence of a Gravity Finance Formal Verification test was found in any of their documentation.

16) Stress Testing environment (%)

> ⚠ **Answer:** 0%

No evidence of any Gravity Finance testnet smart contract usage was found in any of their documentation

---

Security

This section looks at the 3rd party software audits done. It is explained in this document. This section answers the following questions;

17) Did 3rd Party audits take place? (%)
18) Is the bounty value acceptably high?

17) Did 3rd Party audits take place? (%)

> ✓ **Answer:** 100%

CTDSec published a Gravity Finance audit report on August 12th, which is after their mainnet launch in May 2021. However, this audit was started before the mainnet launch.

In addition, an Obelisk audit report was published on September 24th 2021. However, this audit had been going on for more than a month previous to their new LP farms' mainnet launch.

Note: Fix recommendations were implemented by the Gravity Finance team.

Guidance:

100%	Multiple Audits performed before deployment and results public and implemented or not required
90%	Single audit performed before deployment and results public and implemented

	or not required
70%	Audit(s) performed after deployment and no changes required. Audit report is public
50%	Audit(s) performed after deployment and changes needed but not implemented
20%	No audit performed
0%	Audit Performed after deployment, existence is public, report is not public and no improvements deployed OR smart contract address' not found, (where question 1 is 0%)

Deduct 25% if code is in a private repo and no note from auditors that audit is applicable to deployed code

18) Is the bounty value acceptably high (%)

Answer: 70%

Gravity Finance's Bug Bounty program rewards participating users with up to 100k worth of GFI tokens for the most critical of finds.

Guidance:

100%	Bounty is 10% TVL or at least $1M AND active program (see below)
90%	Bounty is 5% TVL or at least 500k AND active program
80%	Bounty is 5% TVL or at least 500k
70%	Bounty is 100k or over AND active program
60%	Bounty is 100k or over
50%	Bounty is 50k or over AND active program
40%	Bounty is 50k or over
20%	Bug bounty program bounty is less than 50k
0%	No bug bounty program offered

An active program means that a third party (such as Immunefi) is actively driving hackers to the site. An inactive program would be static mentions on the docs.

Access Controls

This section covers the documentation of special access controls for a DeFi protocol. The admin access controls are the contracts that allow updating contracts or coefficients in the protocol. Since these contracts can allow the protocol admins to "change the rules", complete disclosure of capabilities is vital for user's transparency. It is explained in this document. The questions this section asks are as follow;

19) Can a user clearly and quickly find the status of the admin controls?
20) Is the information clear and complete?
21) Is the information in non-technical terms that pertain to the investments?
22) Is there Pause Control documentation including records of tests?

19) Can a user clearly and quickly find the status of the access controls (%)

✓ **Answer:** 100%

Gravity Finance's access controls documentation is clearly labelled and easy to find at https://inthenextversion.gitbook.io/gravity-finance/audits-security/owner-priv-and-time-locks.

Guidance:

100%	Clearly labelled and on website, docs or repo, quick to find
70%	Clearly labelled and on website, docs or repo but takes a bit of looking
40%	Access control docs in multiple places and not well labelled
20%	Access control docs in multiple places and not labelled
0%	Admin Control information could not be found

20) Is the information clear and complete (%)

✓ **Answer:** 90%

a) All contracts are clearly labelled as upgradeable (or not) - 30% - Contracts are clearly labelled as being able to be "called", which is a term for saying that it can be upgraded.

b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) - Defined ownership and permissions of the contracts are listed at https://inthenextversion.gitbook.io/gravity-finance/audits-security/owner-priv-and-time-locks.

c) Capabilities for change in contracts are listed at https://inthenextversion.gitbook.io/gravity-finance/audits-security/owner-priv-and-time-locks.

Guidance:

All the contracts are immutable -- 100% OR

a) All contracts are clearly labelled as upgradeable (or not) -- 30% AND
b) The type of ownership is clearly indicated (OnlyOwner / MultiSig / Defined Roles) -- 30% AND
c) The capabilities for change in the contracts are described -- 30%

How to improve this score:

Create a document that covers the items described above. An example is enclosed.

21) Is the information in non-technical terms that pertain to the investments (%)

✓ **Answer:** 90%

Most of the descriptions at https://inthenextversion.gitbook.io/gravity-finance/audits-security/owner-priv-and-time-locks related to how and why their users' funds are safe, and do so in user friendly language.

Guidance:

100%	All the contracts are immutable
90%	Description relates to investments safety and updates in clear, complete non-software l language
30%	Description all in software specific language
0%	No admin control information could not be found

How to improve this score:

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

22) Is there Pause Control documentation including records of tests (%)

ⓘ **Answer:** 40%

Pause Control is briefly mentionned at the top of https://inthenextversion.gitbook.io/gravity-finance/audits-security/owner-priv-and-time-locks.

Guidance:

100%	All the contracts are immutable or no pause control needed and this is explained OR
100%	Pause control(s) are clearly documented and there is records of at least one test within 3 months
80%	Pause control(s) explained clearly but no evidence of regular tests
40%	Pause controls mentioned with no detail on capability or tests
0%	Pause control not documented or explained

How to improve this score**:**

Create a document that covers the items described above in plain language that investors can understand. An example is enclosed.

Appendices

Author Details

The author of this review is Rex of DeFi Safety.

Email : rex@defisafety.com Twitter : @defisafety

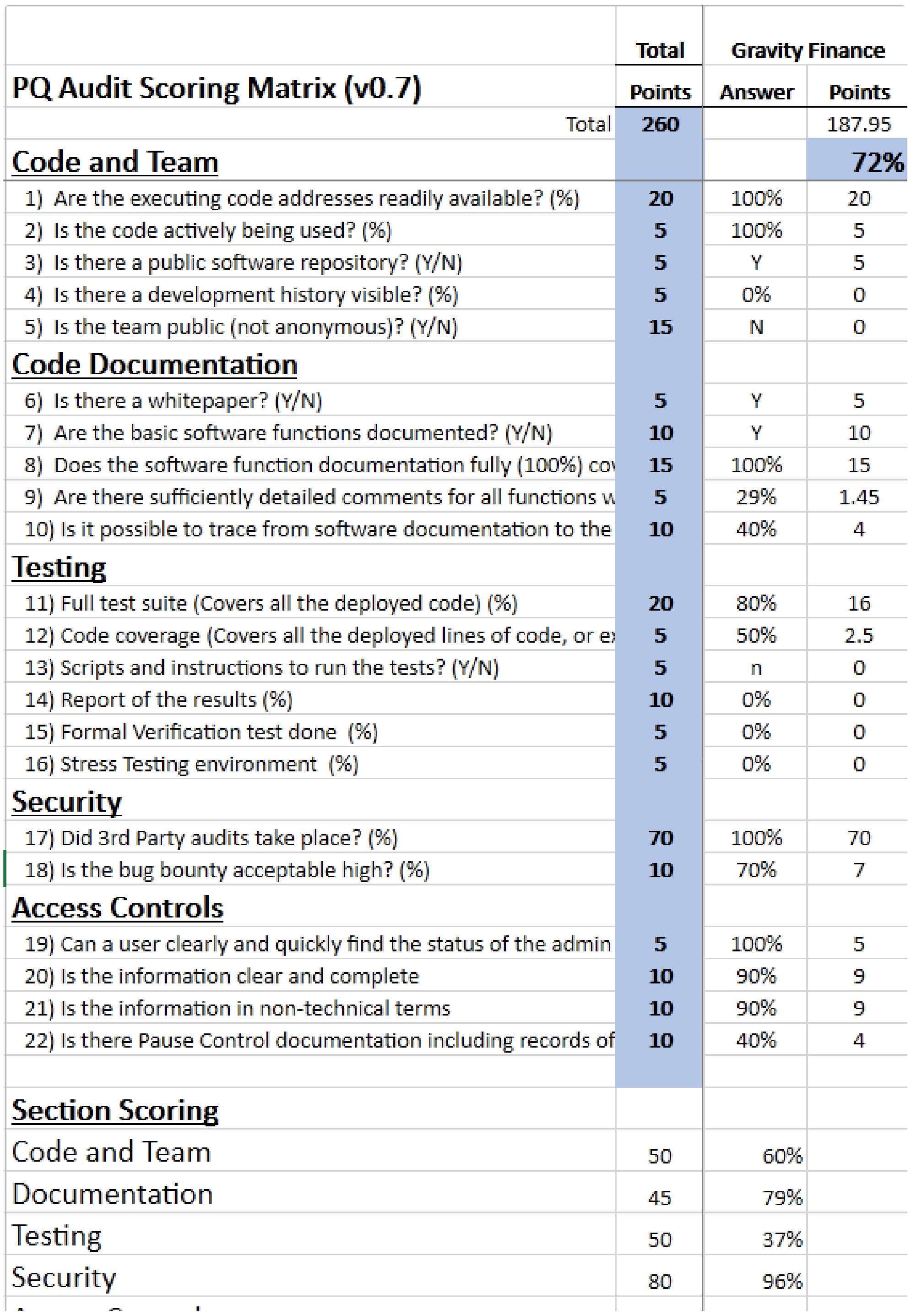
I started with Ethereum just before the DAO and that was a wonderful education. It showed the importance of code quality. The second Parity hack also showed the importance of good process. Here my aviation background offers some value. Aerospace knows how to make reliable code using quality processes.

I was coaxed to go to EthDenver 2018 and there I started SecuEth.org with Bryant and Roman. We created guidelines on good processes for blockchain code development. We got EthFoundation funding to assist in their development.

Process Quality Reviews are an extension of the SecurEth guidelines that will further increase the quality processes in Solidity and Vyper development.

DeFiSafety is my full time gig and we are working on funding vehicles for a permanent staff.

Scoring Appendix



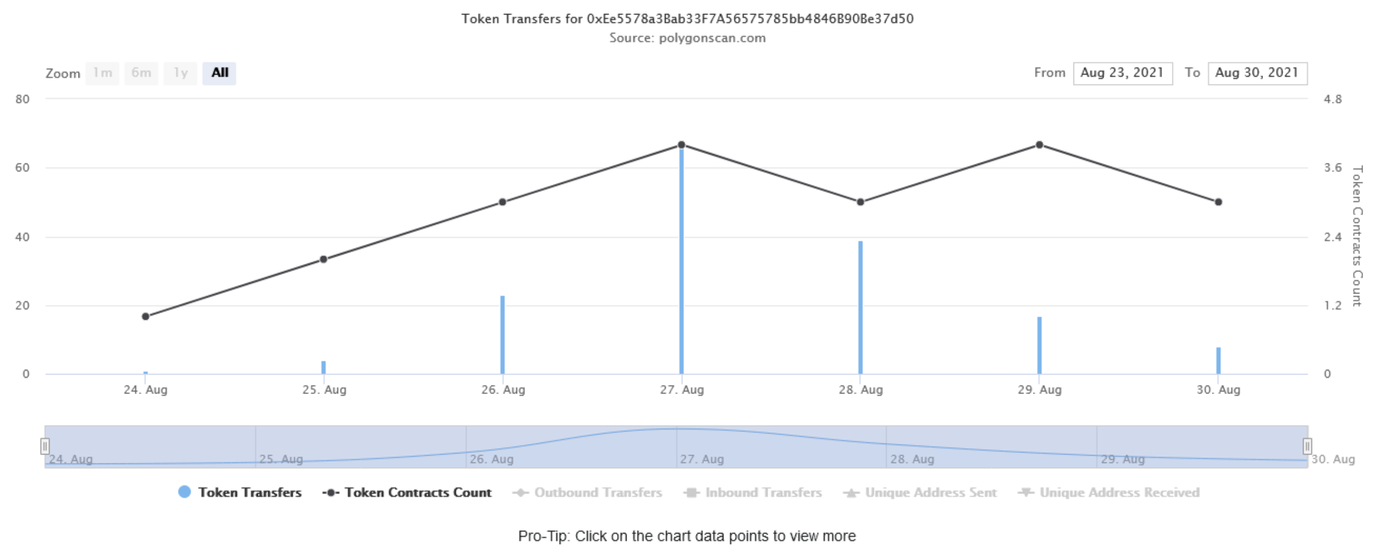
	Total	Gravity Finance	
PQ Audit Scoring Matrix (v0.7)	**Points**	**Answer**	**Points**
Total	**260**		187.95
Code and Team			**72%**
1) Are the executing code addresses readily available? (%)	**20**	100%	20
2) Is the code actively being used? (%)	**5**	100%	5
3) Is there a public software repository? (Y/N)	**5**	Y	5
4) Is there a development history visible? (%)	**5**	0%	0
5) Is the team public (not anonymous)? (Y/N)	**15**	N	0
Code Documentation			
6) Is there a whitepaper? (Y/N)	**5**	Y	5
7) Are the basic software functions documented? (Y/N)	**10**	Y	10
8) Does the software function documentation fully (100%) cov	**15**	100%	15
9) Are there sufficiently detailed comments for all functions w	**5**	29%	1.45
10) Is it possible to trace from software documentation to the	**10**	40%	4
Testing			
11) Full test suite (Covers all the deployed code) (%)	**20**	80%	16
12) Code coverage (Covers all the deployed lines of code, or ex	**5**	50%	2.5
13) Scripts and instructions to run the tests? (Y/N)	**5**	n	0
14) Report of the results (%)	**10**	0%	0
15) Formal Verification test done (%)	**5**	0%	0
16) Stress Testing environment (%)	**5**	0%	0
Security			
17) Did 3rd Party audits take place? (%)	**70**	100%	70
18) Is the bug bounty acceptable high? (%)	**10**	70%	7
Access Controls			
19) Can a user clearly and quickly find the status of the admin	**5**	100%	5
20) Is the information clear and complete	**10**	90%	9
21) Is the information in non-technical terms	**10**	90%	9
22) Is there Pause Control documentation including records of	**10**	40%	4
Section Scoring			
Code and Team	50	60%	
Documentation	45	79%	
Testing	50	37%	
Security	80	96%	
[illegible]			

Access Controls	35	77%

Executing Code Appendix

Name	Address
GFI Token	0x874e178A2f3f3F9d34db862453Cd756E7eAb0381
Governance	0xEe5578a3Bab33F7A56575785bb4846B90Be37d50
Exchange Factory	0x3ed75AfF4094d2Aaa38FaFCa64EF1C152ec1Cf20
Exchange Router	0x57dE98135e8287F163c59cA4fF45f1341b680248
Fee Manager	0x12e26ad5ce1ed4b51f6d2d12ac92765659d4e756
Earnings Manager	0x867A1CCE2Df35b26B9a50Ce5cbD5c1B603938E6F
Price Oracle	0x2e0DfCD5D693DdcE4f0E0c7472561f62B912b19a
Path Oracle	0x3E22044f743c35C9689F9ab76063942beBdF559D
Incinerator (buy & burn GFI)	0x4F6cCd1242323c23fEcf95b9C073C839db18649F
Farm Factory	0x41d8920282eEDCcfC2f857e5e40Aa560a65d762B
Farm Implementation	0xd5d3b955698831cc05ad1cb03ba2ba4ddfc2de1d

Code Used Appendix



Example Code Appendix

```
contract Governance is Initializable, OwnableUpgradeable {
    mapping(address => uint256) public feeBalance;
    address public tokenAddress;
    struct FeeLedger {
        uint256 totalFeeCollected_LastClaim;
        uint256 totalSupply_LastClaim;
        uint256 userBalance_LastClaim;
    }
    mapping(address => FeeLedger) public feeLedger;

    mapping(address => uint[3]) public tierLedger;
    uint[3] public Tiers;
    uint256 public totalFeeCollected;
    iGravityToken GFI;
    IERC20 WETH;
    IERC20 WBTC;

    /**
    * @dev emitted when Fees are deposited into the Governance contract
    * @param weth the amount of wETH deposited into the governance contract
    * @param wbtc the amount of wBTC deposited into the governance contract
    **/
    event FeeDeposited(uint weth, uint wbtc);

    /**
    * @dev emitted when a wETH fee is claimed
    * @param claimer the address that had it's fees claimed
    * @param recipient the address the fees were sent to
    * @param amount the amount of wETH sent to the recipient
    **/
    event FeeClaimed(address claimer, address recipient, uint amount);

    /**
    * @dev emitted when GFI is burned for wBTC
    * @param claimer the address burning GFI for wBTC
    * @param GFIamount the amount of GFI burned
    * @param WBTCamount the amount of wBTC sent to claimer
    **/
    event WbtcClaimed(address claimer, uint GFIamount, uint WBTCamount);

    /**
    * @dev used to ensure only token contract can call govAuth functions lines 233 -> 268
    **/
    modifier onlyToken() {
        require(msg.sender == tokenAddress, "Only the token contract can call this functio
        _;
    }

    function initialize(
        address GFI_ADDRESS,
        address WETH_ADDRESS,
        address WBTC_ADDRESS
    ) public initializer {
        __Ownable_init();
        tokenAddress = GFI_ADDRESS;
        GFI = iGravityToken(GFI_ADDRESS);
        WETH = IERC20(WETH_ADDRESS);
        WBTC = IERC20(WBTC_ADDRESS);
    }

    function updateTiers(uint tier3, uint tier2, uint tier1) external onlyOwner{
        require(tier3 > tier2 && tier2 > tier1, 'Gravity Finance: Invalid Tier assignments
        Tiers[0] = tier1;
        Tiers[1] = tier2;
        Tiers[2] = tier3;
    }

    /**
    * @dev internal function called when token contract calls govAuthTransfer or govAuthTra
    * Will update the recievers fee balance. This will not change the reward they would hav
    * rather it updates the fee ledger to refelct the new increased amount of GFI in their
    * @param _address the address of the address recieving GFI tokens
    * @param amount the amount of tokens the address is recieving
    * @return amount of wETH added to _address fee balance
    **/
    function _updateFeeReceiver(address _address, uint256 amount)
        internal
        returns (uint256)
    {
        uint256 supply;
        uint256 balance;

        //Pick the greatest supply and the lowest user balance
        uint256 currentBalance = GFI.balanceOf(_address) + amount; //Add the amount they a
        if (currentBalance > feeLedger[_address].userBalance_LastClaim) {
            balance = feeLedger[_address].userBalance_LastClaim;
        } else {
            balance = currentBalance;
        }

        uint256 currentSupply = GFI.totalSupply();
        if (currentSupply < feeLedger[_address].totalSupply_LastClaim) {
            supply = feeLedger[_address].totalSupply_LastClaim;
        } else {
            supply = currentSupply;
        }

        uint256 feeAllocation =
            ((totalFeeCollected -
                feeLedger[_address].totalFeeCollected_LastClaim) * balance) /
                supply;
        feeLedger[_address].totalFeeCollected_LastClaim = totalFeeCollected;
        feeLedger[_address].totalSupply_LastClaim = currentSupply;
        feeLedger[_address].userBalance_LastClaim = currentBalance;
        feeBalance[_address] = feeBalance[_address] + feeAllocation;
        return feeAllocation;
    }

    /**
    * @dev updates the fee ledger info for the specified address
    * This function can be used to update the fee ledger info for any address, and is used
    * @param _address the address you want to update the fee ledger info for
    * @return the amount of wETH added to _address feeBalance
    **/
    function updateFee(address _address) public returns (uint256) {
        require(GFI.balanceOf(_address) > 0, "_address has no GFI");
        uint256 supply;
        uint256 balance;

        //Pick the greatest supply and the lowest user balance
        uint256 currentBalance = GFI.balanceOf(_address);
        if (currentBalance > feeLedger[_address].userBalance_LastClaim) {
            balance = feeLedger[_address].userBalance_LastClaim;
        } else {
            balance = currentBalance;
        }

        uint256 currentSupply = GFI.totalSupply();
        if (currentSupply < feeLedger[_address].totalSupply_LastClaim) {
            supply = feeLedger[_address].totalSupply_LastClaim;
        } else {
            supply = currentSupply;
        }

        uint256 feeAllocation =
            ((totalFeeCollected -
                feeLedger[_address].totalFeeCollected_LastClaim) * balance) /
                supply;
        feeLedger[_address].totalFeeCollected_LastClaim = totalFeeCollected;
        feeLedger[_address].totalSupply_LastClaim = currentSupply;
        feeLedger[_address].userBalance_LastClaim = currentBalance;
        feeBalance[_address] = feeBalance[_address] + feeAllocation;
        return feeAllocation;
    }

    /**
    * @dev updates callers fee ledger, and pays out any fee owed to caller
    * @return the amount of wETH sent to caller
    **/
    function claimFee() public returns (uint256) {
        require(GFI.balanceOf(msg.sender) > 0, "User has no GFI");
        uint256 supply;
        uint256 balance;

        //Pick the greatest supply and the lowest user balance
        uint256 currentBalance = GFI.balanceOf(msg.sender);
        if (currentBalance > feeLedger[msg.sender].userBalance_LastClaim) {
            balance = feeLedger[msg.sender].userBalance_LastClaim;
        } else {
            balance = currentBalance;
        }
        }

        uint256 currentSupply = GFI.totalSupply();
        if (currentSupply < feeLedger[msg.sender].totalSupply_LastClaim) {
            supply = feeLedger[msg.sender].totalSupply_LastClaim;
        } else {
            supply = currentSupply;
        }

        uint256 feeAllocation =
            ((totalFeeCollected -
                feeLedger[msg.sender].totalFeeCollected_LastClaim) * balance) /
                supply;
        feeLedger[msg.sender].totalFeeCollected_LastClaim = totalFeeCollected;
        feeLedger[msg.sender].totalSupply_LastClaim = currentSupply;
        feeLedger[msg.sender].userBalance_LastClaim = currentBalance;
        //Add any extra fees they need to collect
        feeAllocation = feeAllocation + feeBalance[msg.sender];
        feeBalance[msg.sender] = 0;
        require(WETH.transfer(msg.sender, feeAllocation),"Failed to delegate wETH to calle
        emit FeeClaimed(msg.sender, msg.sender, feeAllocation);
        return feeAllocation;
    }

    /**
    * @dev updates callers fee ledger, and pays out any fee owed to caller to the reciever
    * @param reciever the address to send callers fee balance to
    * @return the amount of wETH sent to reciever
    **/
    function delegateFee(address reciever) public returns (uint256) {
        require(GFI.balanceOf(msg.sender) > 0, "User has no GFI");
        uint256 supply;
        uint256 balance;

        //Pick the greatest supply and the lowest user balance
        uint256 currentBalance = GFI.balanceOf(msg.sender);
        if (currentBalance > feeLedger[msg.sender].userBalance_LastClaim) {
            balance = feeLedger[msg.sender].userBalance_LastClaim;
        } else {
            balance = currentBalance;
        }

        uint256 currentSupply = GFI.totalSupply();
        if (currentSupply < feeLedger[msg.sender].totalSupply_LastClaim) {
            supply = feeLedger[msg.sender].totalSupply_LastClaim;
        } else {
            supply = currentSupply;
        }
```

SLOC Appendix

Solidity Contracts

Language	Files	Lines	Blanks	Comments	Code	Complex
Solidity	16	2852	330	567	1955	208

Comments to Code 567/1955 = 29%

Javascript Tests

Language	Files	Lines	Blanks	Comments	Code	Complex
JavaScript	9	2779	556	137	2086	35

Tests to Code 2086/1955 = 107%