Quantstamp Security Assessment Certificate

November 2nd 2020 — Quantstamp Verified

Reflexer Helper Contracts

This security assessment was prepared by Quantstamp, the leader in blockchain security

Executive Summary

Type	DAO
Auditors	Sebastian Banescu, Senior Research Engineer Jake Goh Si Yuan, Research Engineer Fayçal Lalidji, Security Auditor
Timeline	2020-08-21 through 2020-11-02
EVM	Muir Glacier
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	GEB Docs
Documentation Quality	Medium
Test Quality	Medium
Source Code	

Repository	Commit
geb-protocol-token-authority	0ba4673
geb-printing-permissions	96a8737
esm	3dfc0cc
geb-chainlink-median	0216237
geb-safe-manager	8c775b4
geb-proxy-actions	e0e2c51
ds-roles	424a441
geb-fsm-governance-interface	b503d55
geb-uniswap-median	60ce6fa
geb-deploy	dba8e28

Total Issues	**33** (16 Resolved)
High Risk Issues	0 (0 Resolved)
Medium Risk Issues	**7** (5 Resolved)
Low Risk Issues	**12** (8 Resolved)
Informational Risk Issues	**14** (3 Resolved)
Undetermined Risk Issues	0 (0 Resolved)

Documentation Issues Addressed

High Risk	The issue puts a large number of users' sensitive information at risk, or is reasonably likely to lead to catastrophic impact for client's reputation or serious financial implications for client and users.
Medium Risk	The issue puts a subset of users' sensitive information at risk, would be detrimental for the client's reputation if exploited, or is reasonably likely to lead to moderate financial impact.
Low Risk	The risk is relatively small and could not be exploited on a recurring basis, or is a risk that the client has indicated is low-impact in view of the client's business circumstances.
Informational	The issue does not post an immediate risk, but is relevant to security best practices or Defence in Depth.
Undetermined	The impact of the issue is uncertain.
Unresolved	Acknowledged the existence of the risk, and decided to accept it without engaging in special efforts to control it.
Acknowledged	The issue remains in the code but is a result of an intentional business or design decision. As such, it is supposed to be addressed outside the programmatic means, such as: 1) comments, documentation, README, FAQ; 2) business processes; 3) analyses showing that the issue shall have no negative consequences in practice (e.g., gas analysis, deployment settings).
Resolved	Adjusted program implementation, requirements or constraints to eliminate the risk.
Mitigated	Implemented actions to minimize the impact or likelihood of the risk.

Summary of Findings

Quantstamp has identified a set of 33 issues spanning from Medium to Informational security levels. Additionally, 3 findings were marked as Undetermined, because their severity depends on the interpretation of several factors by the development team. We strongly recommend that the development team addresses these findings, as well as the 25 issues included in the "Adherence to Specification", "Code Documentation" and "Adherence to Best Practices" section.

The `dapptools` suite (used to develop and run tests for all the repositories that were in scope for this audit), does not contain a feature that can generate a test coverage summary similar to `solidity-coverage`. Therefore, we were not able to measure the coverage for this project. However, we have noticed during our audit that the coverage is less than 100%. We strongly recommend increasing the number of tests to cover all branches in the code such that all the functionality is tested each time there is a new change to the code. This is the best way to detect functional code defects, which affect the business logic.

Note:

- The scope for the `geb-deploy` repository was restricted to only the `CollateralJoin6` contract from the `AdvancedTokenAdapters.sol` file.
- The scope for the `ds-roles` repository was restricted to only the `recursive_roles.sol` file.

Update: Quantstamp has determined that the majority of findings were resolved and/or acknowledged by Reflexer Labs. Only 5 issues remain unresolved, one of which (QSP-33) is newly added due to errors introduced during the fixes or due to miscommunication of the issues. We strongly recommend these issues be resolved before deployment on Mainnet.

Update 2: Quantstamp has performed a 2nd reaudit based on the responses given by the dev team. All issues have been either resolved or acknowledged for the commit hashes listed on the first page of this report, except for the 2nd sub-point of QSP-14.

ID	Description	Severity	Status
QSP-1	Inconsistent `CollateralLike` definitions	Medium	Acknowledged
QSP-2	Missing input validation	Medium	Mitigated
QSP-3	All accounts may be unauthorized	Medium	Fixed
QSP-4	Recursive authentification	Medium	Acknowledged
QSP-5	Uniswap pair modification blocked after the first time	Medium	Fixed
QSP-6	Incorrect converter price average	Medium	Fixed
QSP-7	ChainLink stale price	Medium	Mitigated
QSP-8	Failure to authorize new `ESMThresholdSetter`	Low	Fixed
QSP-9	Negative values for `collateralJoined` possible	Low	Fixed
QSP-10	`ESM.triggerThreshold` can be modified after shutdown	Low	Fixed
QSP-11	Window size and period size could be 0 for `UniswapPriceFeedMedianizer`	Low	Fixed
QSP-12	Off-by-one error	Low	Acknowledged
QSP-13	Access control is error prone	Low	Acknowledged
QSP-14	Integer Overflow / Underflow	Low	Mitigated
QSP-15	Division by zero	Low	Fixed
QSP-16	Expecting function to throw, where `try/catch` prevents this	Low	Acknowledged
QSP-17	Joined collateral added to wrong address balance	Low	Acknowledged
QSP-18	Converter price cumulative computation may diverge from real median	Low	Mitigated
QSP-19	Unlocked Pragma	Informational	Acknowledged
QSP-20	Only start event emitted for some system uncover calls	Informational	Acknowledged
QSP-21	Block Timestamp Manipulation	Informational	Acknowledged
QSP-22	Incompatibility with ERC20 standard	Informational	Acknowledged
QSP-23	User annoyance due to missing reward and reason	Informational	Acknowledged
QSP-24	`DSRecursiveRoles` authority	Informational	Fixed
QSP-25	ChainLink Medianizer does not compute median price	Informational	Acknowledged
QSP-26	Uniswap median computation may be inaccurate	Informational	Fixed
QSP-27	Token burning is not guaranteed to send to `address(0)`	Informational	Acknowledged
QSP-28	Expected events not emitted	Informational	Fixed
QSP-29	Potential precision loss	Informational	Acknowledged
QSP-30	Converter price cumulative update may skip periods	Low	Fixed
QSP-31	Improper handling of tokens with more than 18 decimals	Informational	Acknowledged
QSP-32	Improper handling of collateral	Informational	Acknowledged
QSP-33	Fixed `chainId` enables replay attacks	Informational	Acknowledged

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the repository for security-related issues, code quality, and adherence to specification and best practices.

Possible issues we looked for included (but are not limited to):

- Transaction-ordering dependence
- Timestamp dependence
- Mishandled exceptions and call stack limits
- Unsafe external calls
- Integer overflow / underflow
- Number rounding errors
- Reentrancy and cross-function vulnerabilities
- Denial of service / logical oversights
- Access control
- Centralization of power
- Business logic contradicting the specification
- Code clones, functionality duplication
- Gas usage
- Arbitrary token minting

Methodology

The Quantstamp auditing process follows a routine series of steps:

1. Code review that includes the following
   i. Review of the specifications, sources, and instructions provided to Quantstamp to make sure we understand the size, scope, and functionality of the smart contract.
   ii. Manual review of code, which is the process of reading source code line-by-line in an attempt to identify potential vulnerabilities.
   iii. Comparison to specification, which is the process of checking whether the code does what the specifications, sources, and instructions provided to Quantstamp describe.
2. Testing and automated analysis that includes the following:
   i. Test coverage analysis, which is the process of determining whether the test cases are actually covering the code and how much code is exercised when we run those test cases.
   ii. Symbolic execution, which is analyzing a program to determine what inputs cause each part of a program to execute.
3. Best practices review, which is a review of the smart contracts to improve efficiency, effectiveness, clarify, maintainability, security, and control based on the established industry and academic practices, recommendations, and research.
4. Specific, itemized, and actionable recommendations to help you take steps to secure your smart contracts.

Toolset

The notes below outline the setup and steps performed in the process of this audit.

Setup

Tool Setup:

- Slither v0.6.12

Steps taken to run the tools:

1. Installed the Slither tool: `pip install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Inconsistent `CollateralLike` definitions

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `GebProxyActions.sol`

Description: Several `abstract contract CollateralLike` definitions are given in the token adapter files. These definitions are inconsistent with respect to the functions they declare as `virtual` and hence, require to be implemented. The `GebProxyActions` contract defines yet another `abstract contract CollateralLike`, which requires a set of functions that does not match with any of the other aforementioned definitions. Moreover, `GebProxyActions` also casts addresses to this `CollateralLike` definition and uses those functions throughout the logic. This may lead to runtime transaction failures due to calling undefined functions in `GebProxyActions`. This would have a severe impact on all users of the system.

The following table indicates which `abstract contract CollateralLike` definitions require which functions (names on top row).

	allowance	approve	balanceOf	decimals	deposit	transfer	transferFrom	withdraw
BasicTokenAdapters.CollateralLike				✓		✓	✓	
AdvancedTokenAdapters.CollateralLike				✓		✓	✓	
AdvancedTokenAdapters.CollateralLike2	✓		✓	✓		✓	✓	

	allowance	approve	balanceOf	decimals	deposit	transfer	transferFrom	withdraw
AdvancedTokenAdapters.CollateralLike3						✓	✓	
AdvancedTokenAdapters.CollateralLike4			✓	✓		✓		
AdvancedTokenAdapters.CollateralLike5				✓		✓	✓	
GebProxyActions.CollateralLike		✓			✓	✓	✓	✓
GebSafeManager.CollateralLike						✓	✓	

Note that the last row of the previous table shows `GebSafeManager` which also defines `abstract contract CollateralLike`. However, `CollateralLike` does not seem to be used in `GebSafeManager`.

Recommendation: Align the function declarations of the `CollateralLike` definitions in the token adapter files, with the required functions used in `GebProxyActions`. Also consider writing a developer documentation page which clearly describes which functions contracts that implement `CollateralLike` should implement.
Update: The dev team has acknowledged the issue. However, they choose to keep the code as-is, because they (as well as MakerDAO) tested this code with a variety of tokens and have not discovered any issues so far.

QSP-2 Missing input validation

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `ProtocolTokenAuthority.sol`, `FsmGovernanceInterface.sol`, `UniswapPriceFeedMedianizer.sol`, `ChainlinkPriceFeedMedianizer.sol`

Description: Functions should always validate input parameters before using their values. One common mistake that could occur due to missing input validation is that funds are sent to the `0x0` address. The following instances of this issue were identified:

1. **[Acknowledged]** The `ProtocolTokenAuthority.setOwner` function does not check if the address of the `newOwner` is different from `address(0)`. This may not be a problem in the beginning while the `root` address is non-zero. However, it would leave the contract ownerless once the `root` is set to `address(0)`.
2. **[Acknowledged]** The `setFsm`, `setOwner` and `setAuthority` functions inside `FsmGovernanceInterface` do not validate their input parameter of type `address`, which could accidentally lead to setting these important state variables to `0x0`.
3. **[Fixed]** The `UniswapPriceFeedMedianizer.constructor` function does not check if the `converterFeed_` input parameter of type `address` is different from `0x0`. If the `converterFeed` state variable is `0x0` then it causes any call to `updateObservations` to revert without indicating the exact error. If `0x0` is a valid value for `converterFeed` we recommend wrapping the call to `converterFeed.getResultWithValidity()` inside `updateObservations` in a try-catch control structure.
4. **[Fixed]** The `UniswapPriceFeedMedianizer.constructor` function does not check if the `uniswapFactory_` input parameter of type `address` is different from `0x0`. If the `uniswapFactory` state variable is `0x0` then it prevents changing the values of the `targetToken` and `denominationToken` variables, via the `modifyParameters` function. Moreover, once set, the value of `uniswapFactory` cannot be modified by any other function.
5. **[Fixed]** The `ChainlinkPriceFeedMedianizer.constructor` function does not check if the `aggregator_` input parameter of type `address` is different from `0x0`. If the `chainlinkAggregator` state variable is `0x0` then it will cause the `updateResult` function to fail.

Recommendation: Add input parameter validation which checks that input parameters of type `address` are different from zero, in all functions where this is needed.
Update: The first 2 points indicated under this finding were acknowledged, while the following 3 points were fixed.

QSP-3 All accounts may be unauthorized

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `several contracts`

Description: Several contracts use a repeatable pattern for authorization, that is:

- Define a mapping called `authorizedAccounts`, which indicates which addresses are authorized and which not.
- Define two functions called `addAuthorization` and `removeAuthorization`, which either grant or deny access for any given address.
- Define a `isAuthorized` modifier, which checks if the given `msg.sender` is authorized and if so executes the associated function.

The `removeAuthorization` function does not prevent the only remaining authorized account from calling it, hence all addresses in `authorizedAccounts` would map to zero. This would permanently deny access to all functions that are protected by the `isAuthorized` modifier. An instance of this was found inside the `GebDeploy.sol` file in the `CoinJoinFactory.newCoinJoin` function, where the single authorized account calls `removeAuthorization` essentially denying access to all the `isAuthorized` methods of the `CoinJoin` contract, including the `disableContract()` function.

Recommendation:

1. Keep track of the number of authorized addresses using a state variable and if there is only one authorized address left, then revert that address call to `removeAuthorization` by adding a `require` statement at the beginning of that function.
2. Authorize `msg.sender` inside the `CoinJoinFactory.newCoinJoin` function, before calling `removeAuthorization` OR remove the call to `removeAuthorization`.

Update: Fixed using the 2nd recommendation.

QSP-4 Recursive authentification

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `recursive_roles.sol`, `delegate_roles.sol`

Description: Whenever `DSRecursiveRoles` calls the implemented `canCall` function in the `authority` contract address through the `isAuthorized` modifier, the code parameter will be set to its own address. Doing recursive calls will change the code parameter value following which authority is calling `canCall`, therefore possibly allowing denied functions for the `src` address to be called.

Recommendation: Remove the recursive calls to `canCall`
Update: In the process of fixing this issue (removing the recursive calls), a similar issue was introduced, because `msg.sender` is used as the 2nd parameter value for the call to `canCall` on L163. This is a delegated role, the initial `msg.sender` value will be different from the second caller (`msg.sender`) and since the functions are authorised by their contract address it will

incorrectly deny access.
Update 2: The `DSRecursiveRoles` contract (now called `DSDelegateRoles`) was amended and will no longer be used in a recursive manner. The dev team has acknowledged the residual risk of still using it recursively, but decided to keep the code as is, as they claim that the highlighted risk cannot materialize in such a setup. A note will be added to the contract to ensure onlookers and forkers are informed about the risk.

QSP-5 Uniswap pair modification blocked after the first time

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: The `modifyParameters` method allows modifying `targetToken` and `denominationToken` even after the `uniswapPair` is set on L224-L227 and L231-L234.

Initially when the medianizer contract is deployed both `targetToken` and `denominationToken` values are zero, after setting both of them using `modifyParameters`, `uniswapPair` will be set. If an authorized address calls `modifyParameters` after the `uniswapPair` is set, the conditions on L224 and L231 will always be false. However, `modifyParameters` will still modify `targetToken` or `denominationToken` values, meaning that the tokens pair will not match the `uniswapPair`.

Multiple issues can be raised:

- Wrong return token address value when sorting the token pair on L288, therefore getting a wrong cumulative price.
- A different `uniswapPair` than the expected one will be used if the `uniswapPair` is not updated correctly when modifying the token addresses.

Recommendation: Use only the constructor to set immutable state variables, otherwise, reimplement `modifyParameters`. Alternatively, ensure that the appropriate `uniswapPair` is updated.
Update: The `modifyParameters` function was reimplemented and can only set tokens and hence the `uniswapPair` only once.

QSP-6 Incorrect converter price average

Severity: *Medium Risk*

Status: Fixed

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: In `converterComputeAmountOut` on L379-L385, `priceAverage` is equal to the `converterPriceCumulative` divided by `subtract(uint(granularity), 1)`, however, `converterPriceCumulative` is the sum of the last n values returned by the `converterFeed` where n is equal to `granularity` and not to `granularity` minus `1` as implemented in `converterComputeAmountOut`.
This issue will raise the `priceAverage` with the following ratio `granularity / granularity - 1`.

Recommendation: Change the denominator of the division on L382 from `subtract(uint(granularity), 1)` to `granularity`.
Update: The recommendation has been implemented.

QSP-7 ChainLink stale price

Severity: *Medium Risk*

Status: Mitigated

File(s) affected: `ChainlinkPriceFeedMedianizer.sol`

Description: Chainlink price feeds contracts or Aggregators are updated by the projects that uses them by calling `requestRateUpdate` and transferring a certain amount of LINK to each node that will call back the Chainlink contract to set the token spot price. Both `latestAnswer` and `latestTimestamp` are public functions that can be called by anyone.
Currently, the `ChainlinkPriceFeedMedianizer` works by having a group of `Callers` call the method `updateResult` to get the `chainlinkAggregator.latestAnswer()`. The only validations related to this call is that `aggregatorPrice > 0` and `aggregatorTimestamp > 0 && aggregatorTimestamp >= linkAggregatorTimestamp`. The second half of the second expression, `aggregatorTimestamp >= linkAggregatorTimestamp` means that in the `aggregatorTimestamp == linkAggregatorTimestamp` case, this would still be a valid call.
Due to the fact that it is seen as a valid call :

1. **[Acknowledged]** The `callerReward` is still distributed, which means that `Callers` are rewarded for simply updating with a stale call, despite adding no new information to the system.
2. **[Fixed]** More perniciously, the `lastUpdateTime` is set to `now`. This may cause the appearance that the `medianPrice` is at its latest value relative to `now`, which is not true given that the `chainlinkAggregator` may be stale and lags behind the `lastUpdateTime` by an arbitrary amount. This may be manipulated by a malicious actor, given the fact that `read` and `getResultWithValidity` does not check for the `lastUpdateTime` nor the `linkAggregatorTimestamp`.

Recommendation: For the first issue, ensure that the `Caller` actually calls for a `requestRateUpdate`. For the second issue, ensure that validity of result includes the condition that `now`, `lastUpdateTime` and `linkAggregatorTimestamp` are not too far apart, and relate this condition to `periodSize`. Note that given the way Chainlink searches for the median of its results without regards to the staleness of it, price changes could potentially be erratic. `getResultWithValidity` should check the latest timestamps and define if the price is stale within a maximum accepted period and not use `medianPrice > 0` as validity check.
Update: Currently the `getResultWithValidity` function checks if the aggregator is stale. However, in both Uniswap medianizer implementations the `updateObservation` function will throw and not execute if the price is stale. Please note that this is an external factor to the project since the project team does not plan to update the Chainlink aggregator by themselves for now. If the aggregator is changed or abandoned for some reason the price will turn stale, and the medianizer contract will be stuck at a single median value. Our recommendation is to implement a safe mechanism as a fallback oracle to avoid such issues.

QSP-8 Failure to authorize new `ESMThresholdSetter`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ESM.sol`

Description: The code inside the `ESM.constructor` method shows that if a non-zero `thresholdSetter_` address is provided as an input parameter, then this address will be granted access by setting `authorizedAccounts[thresholdSetter_] = 1`. However, this is not the case inside the `ESM.modifyParameters` function, where the new `thresholdSetter` is not granted access and neither is the old `thresholdSetter` set to zero (denied access) in `authorization`.

Recommendation: Set the proper `authorization` or reconsider whether `thresholdSetter` should be receiving it after all.
Update: The authorization for `thresholdSetter` was removed.

QSP-9 Negative values for `collateralJoined` possible

Severity: *Low Risk*

Status: Fixed

File(s) affected: `AdvancedTokenAdapters.sol`

Description: The `collateralJoined` state variable of the `CollateralJoin6` contract is a mapping to signed integers, which allows for negative values. Moreover, both the `join` and `exit` functions in the same contract use a definition of the `addition` function which may return negative values and assign them to `collateralJoined[msg.sender]`.

Recommendation: Change the type of the `collateralJoined` state variable such that it is a mapping to unsigned integers and does not allow negative values.
Update: This issue is only partially solved, because the condition here should set `colatteralJoined` to zero in the else branch.
Update 2: The issue mentioned in the previous update has been addressed in commit `dba8e28ef289671502d5bbb8ad3fd68e52cbd561`.

QSP-10 `ESM.triggerThreshold` can be modified after shutdown

Severity: *Low Risk*

Status: Fixed

File(s) affected: `ESM.sol`

Description: There is no check inside the `ESM.modifyParameters` method that would prevent an authorized user from changing the `ESM.triggerThreshold` after `ESM.shutdown()` was called. Our assumption is that this should not be allowed, because the other `ESM.modifyParameters` method, which changes the `thresholdSetter` does contain such a check. We have also noticed that the missing check might be intentional due to the fact that `ESM.recomputeThreshold()` is called after `settled = 1` inside `ESM.shutdown()`. However, the order of these 2 instructions L142-143 should be swapped.

Recommendation: Add a `require` statement at the beginning of the `ESM.modifyParameters` method that would prevent an authorized user from changing the `ESM.triggerThreshold` after `ESM.shutdown()` was called. For example: `require(settled == 0, "esm/already-settled");`. Also swap the `ESM.recomputeThreshold()` and `settled = 1` statements on L142-143 inside `ESM.shutdown()`.
Update: The recommendation was implemented.

QSP-11 Window size and period size could be 0 for `UniswapPriceFeedMedianizer`

Severity: *Low Risk*

Status: Fixed

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: There is no check for whether the value of the `windowSize_` input parameter to the `UniswapPriceFeedMedianizer.constructor` function is higher than `0`. Therefore, it could be set to `0` by mistake, which would also set the value of the `periodSize` state variable to `0`. Note that there is no other function in the `UniswapPriceFeedMedianizer` which could change the values of these 2 state variables set in the constructor afterwards.

Recommendation: Add a `require` inside the `UniswapPriceFeedMedianizer.constructor` function that checks whether the value of the `windowSize_` input parameter is higher than `0`. Consider adding new functionality that would allow changing the value of this parameter.
Update: The recommendation was implemented.

QSP-12 Off-by-one error

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ESM.sol`

Description: The `require` statement at the beginning of the `ESM.constructor` function mandates that the `triggerThreshold_` for shutting down the system be strictly-less than the protocol token supply. This means that the `triggerThreshold_` can never be equal to the protocol token supply, which might be a valid threshold that the governance wants to set, because "all tokens are equal".

Note that setting the `triggerThreshold` state variable via the `ESM.modifyParameters` function is constrained in the same way: strictly-less than the protocol token supply.

Recommendation: Change the strictly-less comparison to less-than-or-equal in the first `require` statement from `ESM.constructor`. Also align the condition in the `require` statement on L115 in `ESM.modifyParameters` with the less-than-or-equal comparison.
Update: The development team indicated that it is intentionally off-by-one, because there must be remaining tokens so that holders vote to reimburse the address that sacrificed their funds and also vote to redeploy the system.

QSP-13 Access control is error prone

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `GebSafeManager.sol`, `ESM.sol`, `GebPrintingPermissions.sol`, `many other contracts`

Description: The following 2 state variables of the `GebSafeManager` contract are mappings to `uint`:

1. `safeCan` declared on L54-60;
2. `handlerCan` declared on L62-66. However, the values stored in these mappings are treated as binary flags, that is, if the value is equal to `1` then it is interpreted as `true` (as in the safe/handler can do something), otherwise, any other integer value is interpreted as `false` (as in the safe/handler cannot do something). This is only clear when checking the implementation of the `safeAllowed` and `handlerAllowed` modifiers. However, it may not be clear to the caller of the `allowSAFE` and `allowHandler` functions, who may accidentally use the wrong integer value for the `uint ok` parameter to any of those 2 functions. This would result in someone believing that they have allowed/disallowed a SAFE or a handler to do something, when in reality they have not.

A similar issue is also present in the `ESM`, `GebPrintingPermissions` and many other contracts, where `authorizedAccounts` is a mapping to `uint`. For each entry in this mapping `1` means access is granted and any other value means that access is not granted (according to the `isAuthorized` modifier). However, this issue is not as problematic as the aforementioned one, because the only functions that can control the values set to `authorizedAccounts` are `addAuthorization` and `removeAuthorization`, which do not assign input parameter values (or any values derived from input parameters) to `authorizedAccounts`.

Recommendation: Change the type of these mappings such that they map to `bool`. Adjust all functions and modifiers using them accordingly.
Update: The dev team has acknowledged this issue and chose to keep the code as-is, because this issue was inherited from the MakerDAO contracts, and the Reflexer dev team indicated that

the effort of fixing this issue is far larger than the outlined risks.

QSP-14 Integer Overflow / Underflow

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute.

```
    Integer overflow and underflow may cause many unexpected kinds of behavior and was the core reason for the `batchOverflow` attack. Here's an example with `uint8` variables, meaning unsigned integers with a range of
`0..255`.
    ```
    function under_over_flow() public {
        uint8 num_players = 0;
        num_players = num_players - 1; // 0 - 1 now equals 255!
        if (num_players == 255) {
            emit LogUnderflow();   // underflow occurred
        }
        uint8 jackpot = 255;
        jackpot = jackpot + 1;   // 255 + 1 now equals 0!
        if (jackpot == 0) {
            emit LogOverflow();   // overflow occurred
        }
    }
    ```
```

Several instances of integer underflow have been detected:

1. **[Fixed]** The `UniswapPriceFeedMedianizer` uses the block timestamp for multiple decisions including for determining the median price inside `getMedianPrice`. Here combined with the use of primitive integer subtraction (`-`), the use of `now` could lead to an integer underflow in the following statement: `uint timeElapsedSinceFirstUniObservation = now - firstUniswapObservation.timestamp;` This in turn would lead the `medianPrice` to not be updated.
2. **[Unresolved]** The `uniswapComputeAmountOut` function does not check whether the input parameter `priceCumulativeEnd` is greater-or-equal to `priceCumulativeStart`. This check is not performed at the call sites of this function either. Therefore, the result of the subtraction `priceCumulativeEnd - priceCumulativeStart` inside `uniswapComputeAmountOut` may underflow, leading to a much higher price average than in reality.
3. **[Fixed]** The `updateResult` function does not check if `uniswapObservations[observationIndex].timestamp` is less-or-equal than the current block timestamp. Therefore the subtraction `now - uniswapObservations[observationIndex].timestamp` could underflow, leading to an unnecessary update.

Recommendation: Use a safe subtraction function instead of primitive subtraction.
Update: The 1st and 3rd points were fixed, while the 2nd point was not addressed (code unchanged).

QSP-15 Division by zero

Severity: *Low Risk*

Status: Fixed

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: There are several instances of this issue:

1. The `uniswapComputeAmountOut` function does not check whether the input parameter `timeElapsed` is greater than zero. This check is not performed at the call sites of this function either. Therefore, the following division could result in a division-by-zero error `(priceCumulativeEnd - priceCumulativeStart) / timeElapsed` inside `uniswapComputeAmountOut` leading to a failure (without any error message) in `getMedianPrice`.
2. The `UniswapPriceFeedMedianizer.constructor` function does not check if the `converterFeedScalingFactor_` input parameter of type `uint256` is greater than zero. If the `converterFeedScalingFactor` state variable is `0` then it causes a division by zero in any call to `converterComputeAmountOut`. Moreover, once the value of this state variable is set in the constructor, it cannot be changed by any function.

Recommendation: Use a safe division function or check whether the operator of a division is greater than 0.
Update: The recommendation has been implemented.

QSP-16 Expecting function to throw, where `try/catch` prevents this

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ESM.sol`

Description: The `try/catch` statement allows to react on failed external calls. Calling `recomputeThreshold` on L143 in `shutdown` may actually fail to update the threshold and it will not throw because of the `try/catch` inside the `recomputeThreshold` function. This means that there is no way for the `shutdown` function to detect if the threshold was recomputed and react accordingly.

Recommendation: Change the way in which `recomputeThreshold` notifies the calling contract that the update failed, e.g. using different return values.
Update: During the re-audit, the auditors have realized the description of this issue was not correct. The description has now been updated. However, the title and recommendation have been kept the same.
Update 2: The dev team has acknowledged this issue and mentioned that it is intentional, as they would rather have the system shutdown without recomputing the threshold than revert the transaction if `recomputeThreshold` breaks.

QSP-17 Joined collateral added to wrong address balance

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `AdvancedTokenAdapters.sol`

Description: When joining, `wad` is added to `collateralJoined` for the `msg.sender` and the collateral is added for the `usr` address. However, when exiting `wad` is subtracted from `collateralJoined` for the `msg.sender` and the collateral is subtracted from the `msg.sender`. This means that the `usr` address will use an allowance that it will not be accounted for when joining.

Recommendation: When adding `wad` to `collateralJoined` in the `join` function, it should be done for the `usr` address not for the `msg.sender` address.
Update: The dev team has acknowledged this issue and has decided to keep this structure, especially because they want to use `GebSafeManager` to open Safes for other addresses (Safe handlers) that are not the manager contract itself.

QSP-18 Converter price cumulative computation may diverge from real median

Severity: *Low Risk*

Status: Mitigated

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: Calling the Uniswap oracle to compute the median price for a Uniswap pair once in a `periodSize` is correct since the oracle returns the cumulative price. However, doing the same with Chainlink aggregator can lead the generated average price to be different than the real market median price. This is mainly related with the fact that the Chainlink price feed returns the spot price rather than the cumulative price, as Uniswap does.

Please note that even if this description is related to the Chainlink median price, the target contract is the Uniswap medianizer. The average Chainlink price is processed in `updateObservations` on L436-L459 member of `UniswapPriceFeedMedianizer`.

Recommendation: To get a better approximation of the spot average price, the `granularity` used for Chainlink feeds should be higher than the Uniswap medianizer granularity. Also introducing time-weighted average similarly to Uniswap can reduce the error between the real and computed average since the samples from the feed are not guaranteed to be taken at equivalent periods.
Update: The dev team has created a new implementation of a Uniswap medianizer, which uses a time weighted average.

QSP-19 Unlocked Pragma

Severity: *Informational*

Status: Acknowledged

File(s) affected: `all contracts`

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.6.*`. The caret (`^`) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked."

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.
Update: The dev team has acknowledged this issue and will take care of it at a later time.

QSP-20 Only start event emitted for some system uncover calls

Severity: *Informational*

Status: Acknowledged

File(s) affected: `GebPrintingPermissions.sol`

Description: We assume that all system uncover (attempts) are expected to emit at least 2 events, namely:

1. `StartUncoverSystem` and
2. `EndUncoverSystem` or `AbandonUncoverSystem` depending on whether the uncover was carried out or not.

However, this assumption is broken by the scenario where the following `if`-statement inside the `startUncoverSystem` function is entered:

```
L191: if (now <= allowedSystems[accountingEngine].withdrawAddedRightsDeadline) {
            coveredSystems = subtract(coveredSystems, 1);
            usedAuctionHouses[allowedSystems[accountingEngine].previousDebtAuctionHouse] = 0;
            usedAuctionHouses[allowedSystems[accountingEngine].currentDebtAuctionHouse] = 0;
            revokeDebtAuctionHouses(accountingEngine);
          }
```

This will lead to a single event being emitted, because the code inside this `if`-statement also finalizes the uncovering of the system.

Recommendation: Either emit both events inside the `if`-statement and then `return` OR clearly document that there are cases where only the `StartUncoverSystem` event will be emitted and tell users of such events that the way to recognize these solo events is by checking if the penultimate event parameter (uncoverCooldownEnd) value is equal to zero.
Update: The dev team has acknowledged this issue and has decided to keep the logging as-is. The will create a section in the docs where more information will be given about this issue.

QSP-21 Block Timestamp Manipulation

Severity: *Informational*

Status: Acknowledged

File(s) affected: `OracleRelayer.sol`, `GebPrintingPermissions.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that miners individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes. If a smart contract relies on a timestamp, it must take this into account. There are several instances in the code base that use the block timestamp for making decisions:

1. `OracleRelayer` uses the value of the current block timestamp to compute the `_redemptionPrice` inside the `updateRedemptionPrice()` function. Redemption rate seems to be a per second rate. However, it is not clear why it needs to be per second and not per block.
2. `GebPrintingPermissions.startUncoverSystem`, `GebPrintingPermissions.endUncoverSystem` and `GebPrintingPermissions.proposeIndefinitePrintingPermissions` use the block timestamp for deciding if the `uncoverCooldownEnd`, `withdrawAddedRightsDeadline` and `revokeRightsDeadline` have passed for any given `accountingEngine`.
3. `ChainlinkPriceFeedMedianizer.getCallerReward()` uses the block timestamp for deciding if there should be any reward for the caller.
4. `UniswapPriceFeedMedianizer` uses the block timestamp for multiple decisions including for determining:
    . The median price inside `getMedianPrice`
    . The caller reward inside `getCallerReward`
    . The observation index in the array of Uniswap observations in `updateResult`

Timestamps can be manipulated by miners with up to 900 seconds, which could lead to significant differences in the results of the aforementioned functions.

Recommendation: Invest in user facing documentation that warns users about this possibility and acknowledge that a 900 second timestamp manipulation would be tolerable in any of these functions, that is it would not have a noticeable impact on the end-users.
Update: The dev team are aware of the possible manipulation and we will include it as a risk in the documentation.

QSP-22 Incompatibility with ERC20 standard

Severity: *Informational*

Status: Acknowledged

File(s) affected: `GebProxyActions.sol`, `AdvancedTokenAdapters.sol`

Description: Several deviations from the ERC20 standard have been identified in function signatures:

1. `CollateralLike.approve(address,uint256)` in `GebProxyActions.sol` on L21 is missing `returns (bool success)`
2. `CollateralLike.transfer(address,uint256)` in `GebProxyActions.sol` on L22 is missing `returns (bool success)`
3. `CollateralLike.transferFrom(address,address,uint256)` in `GebProxyActions.sol` on L23 is missing `returns (bool success)`
4. `DSTokenLike.approve(address,uint256)` in `GebProxyActions.sol` on L70 is missing `returns (bool success)`
5. `DSTokenLike.transfer(address,uint256)` in `GebProxyActions.sol` on L71 is missing `returns (bool success)`
6. `DSTokenLike.transferFrom(address,address,uint256)` in `GebProxyActions.sol` on L72 is missing `returns (bool success)`
7. `CollateralLike2.transfer(address,uint256)` in `AdvancedTokenAdapters.sol` on L117 is missing `returns (bool success)`
8. `CollateralLike2.transferFrom(address,address,uint256)` in `AdvancedTokenAdapters.sol` L118 is missing `returns (bool success)` These deviations may cause incompatibility with some smart contracts that assume a faithful implementation of the ERC20 standard.

Recommendation: Add the correct return values to all deviations listed above. Or add an explicit warning for users and developers such that they are aware of these deviations from the ERC20 standard.
Update: The dev team has acknowledged this issue and has decided to keep the code as it is, while adding a warning in the docs.

QSP-23 User annoyance due to missing reward and reason

Severity: *Informational*

Status: Acknowledged

File(s) affected: `UniswapPriceFeedMedianizer.sol`, `ChainlinkPriceFeedMedianizer.sol`

Description: If any of the conditions on the first 2 lines of the `rewardCaller` function are true, then the function simply returns without emitting any event or indicating why it failed to reward the caller. In case the treasury address is `address(0)`, the callers might get upset for wasting funds on gas without receiving any reward.

Recommendation: To avoid upsetting any caller (who is a benevolent actor wanting to help the system), the function should indicate the reason why no reward is given at this time, by emitting an event for each possible case where it cannot reward the caller. Moreover, we recommend keeping track of the rewards for each caller and awarding them later once the treasury is set and the same caller makes another call to `resultUpdate`. Alternatively, the governance could push any outstanding rewards after the treasury is set.
Update: The dev team has acknowledged this issue and considers addressing this issue later.

QSP-24 `DSRecursiveRoles` authority

Severity: *Informational*

Status: Fixed

File(s) affected: `recursive_roles.sol`

Description: Setting `authority` state variable in `DSRecursiveRoles` to its own contract address will create a call loop, since `isAuthorized` calls `canCall` when the `authority` address is non zero and `canCall` calls `isAuthorized`. If the `caller` address is not equal to the `root` address or if the called function is not public the function call will throw with an out of gas error.

Recommendation: Always use setter functions to modify state variable while implementing all necessary requirements to avoid unwanted variable values.
Update: Fixed by removing the recursive roles.

QSP-25 ChainLink Medianizer does not compute median price

Severity: *Informational*

Status: Acknowledged

File(s) affected: `ChainlinkPriceFeedMedianizer.sol`

Description: `ChainlinkPriceFeedMedianizer` does not compute the price median over a defined period of time, it rather simply saves the last value of the price feed when `updateResult` is called. Therefore, the semantics of the contract name is not reflected in the implementation, because there is no median price that is computed by this contract, which might be confusing for the users of this contract.

Recommendation: The contract can be bypassed by the Uniswap medianizer and rather call the Chainlink Aggregator contract directly. Alternatively, the contract can be renamed to indicate that it uses the last price and does not compute a median price.
Update: The dev team has acknowledged that the name is misleading and added that the contract is a proxy to the Chainlink contract and it does fetch a median price. Therefore, will keep this naming for now.

QSP-26 Uniswap median computation may be inaccurate

Severity: *Informational*

Status: Fixed

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: If in `getMedianPrice`, `timeElapsedSinceFirstUniObservation` does not satisfy the condition on L287, meaning that `updateResult` was not called during the first `periodSize` in the ongoing `granularity` rounds, `getMedianPrice` will simply return the previously saved value. Therefore, the return value will not be updated.

Recommendation: To mitigate this issue implement an in-house centralized oracle, that can be used as a fallback for such cases.
Update: Fixed in `UniswapConsecutiveSlotsPriceFeedMedianizer` by using arrays. New values are pushed without skipping a period size.

QSP-27 Token burning is not guaranteed to send to `address(0)`

Severity: *Informational*

Status: Acknowledged

File(s) affected: `ESM.sol`

Description: When burning the required threshold tokens, the transferred value is sent to the `tokenBurner` address. The `tokenBurner` is set in the constructor, which does not guarantee that the tokens are effectively burned as per the repository README.

Recommendation: Use the `burn` function if implemented. Otherwise, use a readable hard coded address.
Update: The dev team has indicated that burning will not transfer to `address(0)`, but to this contract that burns its tokens and reflects the change in `totalSupply`.

QSP-28 Expected events not emitted

Severity: *Informational*

Status: Fixed

File(s) affected: `ChainlinkPriceFeedMedianizer.sol`, `ESM.sol`

Description: There are some events which should be emitted, as per its canonical operations seen in other functions, but are not done so. This can be seen in :

1. `constructor` method in `ChainlinkPriceFeedMedianizer.sol` does not emit `ModifyParameters` for setting `maxRewardIncreaseDelay`, `periodSize` and `aggregator`.
2. `constructor` method in `ESM.sol` does not emit `AddAuthorization` for `thresholdSetter`

Recommendation: Emit the relevant events.
Update: The 1st issue has been fixed, while the 2nd issue is no longer applicable since `thresholdSetter` does not need to be added to the list of the `authorized` addresses.

QSP-29 Potential precision loss

Severity: *Informational*

Status: Acknowledged

File(s) affected: `multiple files`

Description: There are usages of `WAD` and `RAY` related arithmetic operations that uses the division by `WAD` and `RAY` to ensure that the value is correctly based at the end of it. However, there is no validation done for whether the parameters entering the arithmetic operations are of the correct base, and if they are valued lower than `WAD` or `RAY`, there could potentially be precision loss as division truncates in Solidity.

Recommendation: Make use of a simple validation to ensure that precision would not be lost, or document it such that users are aware of this issue.
Update: The dev team has acknowledged this issue. They will document and keep the current implementation.

QSP-30 Converter price cumulative update may skip periods

Severity: *Low Risk*

Status: Fixed

File(s) affected: `UniswapPriceFeedMedianizer.sol`

Description: The `updateResult` function calls `updateObservations` to update both the Uniswap cumulative price and the Chainlink cumulative price. However, if `updateResult` is not called during a specific `periodSize` , `converterPriceCumulative` will not be updated and more importantly `firstConverterFeedObservation.price` will not be subtracted from the `converterPriceCumulative`. Therefore it leaves an old price observation for a maximum period equal to `2 * windowSize` assuming that `updateResult` is called correctly in the next equivalent period.

Recommendation: Depending on final price estimation error tolerance, this issue should be addressed.
Update: The dev team has made a new implementation that does not skip slots. They will also combine the Uniswap median with this OSM alternative that bounds the magnitude of price changes between consecutive updates (e.g price now it $1, next price is $1.1 but the allowed deviation is 2% so the price introduced in the system is $1.02).

QSP-31 Improper handling of tokens with more than 18 decimals

Severity: *Informational*

Status: Acknowledged

File(s) affected: `GebProxyActions.sol`

Description: The `convertTo18` function declared in `GebProxyActions.sol` on L155-L162 assumes that all tokens have 18 decimals or less. If the protocol lists any collateral with token decimals value higher than 18 the subtraction inside the `convertTo18` will return a negative value therefore making the final result of the conversion equal to zero.

The same issue is applicable to L964 and L1007 in the same file.

Recommendation: Either check that no listed token will be allowed to have more than 18 decimals, or adjust the functions to cater to tokens with more than 18 decimals.
Update: The dev team has acknowledged this issue and indicated that it is up to governance to decide how they handle tokens with more than 18 decimals (e.g a separate `GebProxyActions`).
Update 2: The dev team has also added that: initially all tokens will have the default 18 decimal places. Only tokens with 18 decimals will be allowed, such as Ether and native tokens. However, the auditors would like to point out that USDC seems to be used and it only has 6 decimal places. However, tokens with less than 18 decimal places are not the subject of this issue, only those with more than 18 decimal places.

QSP-32 Improper handling of collateral

Severity: *Informational*

Status: Acknowledged

File(s) affected: `GebProxyActions.sol`

Description: Two main actions are required for a user to withdraw his staked collateral:

1. Internally transferring collateral from the safe to the proxy address using the `SAFEEngine.transferCollateral` function.
2. Withdrawing the collateral from `CollateralJoin` using the `exit` function.

However, in all the functions listed below the amount internally transferred can be higher than the amount finally withdrawn since the amount converted to 18 decimals is used in the internal transfer and the original amount is withdrawn using exit, possibly leaving part of the collateral in the proxy contract or throwing the transaction since the user balance will not be enough. Both operations should use the same unit since they both modify the same mapping (1, 2) in `SAFEEngine`.

Please note that this issue is only applicable for token with decimals value lower than 18. If the token has more than 18 decimals, then the previously described issue `Improper handling of tokens with more than 18 decimals` occurs.
Affected functions:

```
- `freeTokenCollateral` on L463
- `exitTokenCollateral` on L494
- `repayDebtAndFreeTokenCollateral` on L860
- `repayAllDebtAndFreeTokenCollateral` on L885
```

The severity of the issue can be defined only after auditing the proxy contract.

Recommendation: Use the same unit for both internal transfers of collateral and for exiting.
Update: The dev team has acknowledged this issue and indicated that governance can deploy a separate proxy action contract or handle cases like this one using specific `CollateralJoin` adapters for every collateral type.

QSP-33 Fixed `chainId` enables replay attacks

Severity: Informational

Status: Acknowledged

File(s) affected: `UniswapV2Pair.sol`

Description: The `chainId` used in the `DOMAIN_SEPARATOR` initialized in `UniswapV2Pair.constructor` uses a hardcoded value equal to `1`. This enables replay attacks on the Mainnet from transactions on the Testnet.

Recommendation: Replace hardcoded chain ID with actual `chainId` assembly instruction.
Update: The dev team acknowledged this issue and chose to keep the code as-is, because the contracts are only used locally and a fixed `chainId` is needed for compatibility with seth the command line tool.

Automated Analyses

Slither

Slither has output hundreds of findings, the vast majority of which we have filtered out as false positives. Some findings have been included under best practices.

Adherence to Specification

1. **[Fixed]** The documentation uses the acronym “CDP” in several places. We recommend replacing it with “SAFE”.

2. **[Fixed]** The following documentation section https://docs.reflexer.finance/system-contracts/token-module/token-adapters#2-contract-details, contains a constant name that does not appear in the implementation namely: `ONE - a 10^27 uint used for math in CoinJoin.` Moreover, the places where the constant `ONE` does appear in the implementation it seems to represent different values, e.g. `uint256 constant ONE = 1.00E18;` inside `geb-protocol-token-authority/src/geb/MockDebtAuctionHouse.sol` on L67.

3. **[Fixed]** Broken links on the main documentation page should be fixed:

    . On the homepage of the docs the whitepaper link is broken https://github.com/reflexer-labs/whitepapers/blob/master/rai.pdf

    . On the homepage of the docs the following broken link exists https://docs.reflexer.finance/system-overview/naming-transition

Code Documentation

We have identified the following issues in the code documentation and code comments:

1. Each repository’s `README.md` file should contain basic instructions on how to run the test suite and any prerequisites for running tests, e.g. installing dapp.tools, obtaining an infura key for mainnet, geth, etc. For each of these prerequisites also specify the versions supported/tested.

2. Every function should at least have a short description of its purpose, its input parameters and its return value(s) if any. Several functions in the code base do not have such comments which increases the effort of maintainability and the probability of human error when subsequent features are added/modified/removed.

3. **[Fixed]** The `@notice` comment of the `convertComputeAmountOut` function says that “Calculate the price of an amount of tokens using the convertor price feed. Used after the contract determines the amount of Uniswap pair denomination tokens for defaultAmountIn target tokens”. However, it does not use the value of the `defaultAmountIn` state variable, instead it uses an input parameter called `amountIn`. This comment should be updated to reflect the implementation.

4. **[Fixed]** `geb-chainlink-median/src/ChainlinkPriceFeedMedianizer.sol::L56` states that `Default to 8 for USD price feeds` and yet in `::L57` the `multiplier` is set to `10`.

5. **[Fixed]** `geb-uniswap-median/src/UniswapPriceFeedMedianizer.sol::L92` states that it is `Contract creation time` but the name of the variable is `updates` which is potentially misleading. Consider a more appropriate name like or comment.

6. `GebProxyActions.sol::L1050` states that `it will do the maximum COIN balance in the safeEngine` whereas functionally it chooses the minimum option in L1053.

7. **[Fixed]** Typo in `GebSafeManager.sol::L191` which states `Allow/disallow a usr address to quit to the the sender handler` has an extra `the`.

8. **[Fixed]** Typo in `GebSafeManager.sol::L275` which states `Frob the safe` -> `From the safe`.

9. **[Fixed]** Typo in `UniswapPriceFeedMedianizer.sol::L62` which states `Unisap pair` -> `Uniswap pair`

10. **[Fixed]** Typo in `GebProxyActions.sol::L156` which states `frob function` -> `from function`

11. Link in `README.md` file from `geb-safe-manager` repo is broken: "Check out the full documentation."

Adherence to Best Practices

1. Use of magic numbers should be replaced with the use of named constants, in order to avoid any ambiguity in what the magic number means and hence increase maintainability. Some examples of magic numbers in the code base:

    . L103, L211, L217, L295, L302, L633 and L700 in `geb-deploy/src/AdvancedTokenAdapters.sol` use `2 ** 255`.

2. Function names should clearly indicate what the purpose of the functions are whenever possible, and they should not be misleading. Some instances of misleading function names in the code base are:

    . The `allowSAFE` and `allowHandler` functions in `geb-cdp-manager/src/GebSafeManager.sol`, which can be used to set permissions for SAFEs and hanglers both ways, that is, allow and disallow. A more suggestive name for these functions would be something like: `setSAFEPermission` and `setHandlerPermission`.

3. SPDX license identifier not provided in a large number of source files. Before publishing the source code, consider adding a comment containing "SPDX-License-Identifier: " to each source file. Use "SPDX-License-Identifier: UNLICENSED" for non-open-source code. See https://spdx.org for more information.

4. L124 in `OSM.sol` contains an unused try/catch parameter. We recommend either using the parameter or remove/comment out the variable name.

5. The following deviations from Solidity naming conventions were found by slither:

    . Constant `ProtocolTokenAuthority.burn` is not in UPPER_CASE_WITH_UNDERSCORES.

    . Constant `ProtocolTokenAuthority.burnFrom` is not in UPPER_CASE_WITH_UNDERSCORES.

    . Constant `ProtocolTokenAuthority.mint` is not in UPPER_CASE_WITH_UNDERSCORES.

    . Function `DebtAuctionHouseLike.AUCTION_HOUSE_TYPE()` defined in `GebPrintingPermissions.sol` on L7 is not in mixedCase.

    . Function `Common.coinJoin_join` defined in `GebProxyActions.so` on L129 is not in mixedCase.

    . Function `GebProxyActions.ethJoin_join` defined in `GebProxyActions.sol` on L228 is not in mixedCase.

    . Function `GebProxyActions.tokenCollateralJoin_join` is not in mixedCase.

    . Function `DSRecursiveRoles.BITNOT` defined in `recursive_roles.sol` on L94 is not in mixedCase.

    . Variable `DSRecursiveRoles._root_users` defined in `recursive_roles.sol` on L24 is not in mixedCase

    . Variable `DSRecursiveRoles._user_roles` defined in `recursive_roles.sol` on L25 is not in mixedCase

    . Variable `DSRecursiveRoles._capability_roles` defined in `recursive_roles.sol` on L26 is not in mixedCase

    . Variable `DSRecursiveRoles._public_capabilities` defined in `recursive_roles.sol` on L27 is not in mixedCase

6. The `ChainlinkPriceFeedMedianizer.multiplier` and `ChainlinkPriceFeedMedianizer.symbol` state variables are initialized with a constant value and do not change during the lifetime of the contract. Therefore, they should be declared as `constant` and should be UPPER_CASE_WITH_UNDERSCORES.

7. **[Fixed]** Code should be reused and code clones should be avoided. For example the code in `GebProxyActions.openLockGNTGenerateDebtAndProtectSAFE` is copied from `GebProxyActions.openLockGNTAndGenerateDebt`. Therefore, `GebProxyActions.openLockGNTGenerateDebtAndProtectSAFE` should instead call the `GebProxyActions.openLockGNTAndGenerateDebt` function before calling `protectSAFE`.

8. Event parameters should be `indexed`, in order to allow easy filtering events that match specific values for those given parameters, thereby reducing the number of events that should be inspected. Out of the 287 events defined in the contracts that were in scope for this audit, only 36 events contained `indexed` parameters. For example, none of the parameters of the events from `GebSafeManager.sol` have been indexed.

9. Inconsistent use of `uint` vs `uint256`. Replace all usages of `uint` to `uint256` to increase maintainability of the code.

10. Unused code should be removed or it should be used where needed. We have found the following instances:

    . The internal `ESM.addition` function is not used anywhere.

11. **[Fixed]** Redundant instructions may be removed from the code. For example:

    . The call to the `revokeDebtAuctionHouses` function inside `GebPrintingPermissions.endUncoverSystem` will call `delete allowedSystems[accountingEngine]`, which sets all values inside the `allowedSystems[accountingEngine]` structure to their default values. Therefore, L242 `allowedSystems[accountingEngine].covered = false;` and L252 `delete allowedSystems[accountingEngine]` in `GebPrintingPermissions` are redundant and can be removed.

    . The call to `addition(now, addRightsCooldown)` is done twice inside of `GebPrintingPermissions.coverSystem`. In order to save gas and improve code maintainability we recommend storing the result of the call in a local variable and using it twice.

Test Results

Test Suite Results

The tests for the `geb-safe-manager` repo could not be executed, because of the following error:

```
Error: Undeclared identifier.
  --> src/GebSafeManager.t.sol:91:9:
   |
91 |         deployIndex(collateralAuctionType);
   |         ^^^^^^^^^^
```

All other tests in the test suite have passed.

However, we have identified the following issue during testing: The `solc` version (0.5.12) specified on L6 in `geb-protocol-token-authority/test.sh` is incorrect. This is indicated by the compilation errors output when running `test.sh`. The version should be changed to 0.6.7 or above.

Update: The issue inside `geb-protocol-token-authority/test.sh` has been fixed. However, the tests inside the `geb-safe-manager` repo could not be executed due to the error indicated above.

```
Running 18 tests for src/ProtocolTokenAuthority.t.sol:ProtocolTokenAuthorityTest
[PASS] testFailRemoveAuth() (gas: 8107)
[PASS] testRemoveOwnerByOwner() (gas: 46603)
[PASS] testFailSetRootAsOwner() (gas: 40093)
[PASS] testAddAuthOwner() (gas: 60646)
[PASS] testRemoveAuth() (gas: 37066)
[PASS] testFailMintNotAuthedNotRootNotOwner() (gas: 15111)
[PASS] testSetOwner() (gas: 39380)
[PASS] testSetRoot() (gas: 15655)
[PASS] testAddAuth() (gas: 35290)
[PASS] testMintAsAuthorizedAccount() (gas: 40010)
[PASS] testFailAddAuth() (gas: 8196)
[PASS] testMintAsRoot() (gas: 10748)
[PASS] testRemoveAuthOwner() (gas: 70630)
[PASS] testFailSetRoot() (gas: 11680)
[PASS] testFailSetOnwer() (gas: 12555)
[PASS] testMintAsOwner() (gas: 706702)
[PASS] testBurn() (gas: 48466)
[PASS] testRootCanCallAnything() (gas: 6872)

Running 38 tests for src/GebPrintingPermissions.t.sol:GebPrintingPermissionsTest
[PASS] testFailRemovePreviousHouseWhenNull() (gas: 199893)
[PASS] testFailCoverWithoutAuthorityPermission() (gas: 142943)
[PASS] testFailRemovePreviousHouseWhenOutstandingAuctions() (gas: 439757)
[PASS] testRemovePreviousAuctionHouse() (gas: 473544)
[PASS] testFailEndUncoverWithoutStarting() (gas: 439330)
[PASS] testFailStartUncoverWhenNotEnoughSystemsCovered() (gas: 207232)
[PASS] testFailStartUncoverWhilePreviousHouseHasOutstandingAuctions() (gas: 464038)
[PASS] testFailCoverSameAuctionHouseTwice() (gas: 335422)
[PASS] testAbandonUncover() (gas: 670653)
[PASS] testFailProposeIndefiniteWithZeroFreeze() (gas: 208808)
[PASS] testFailEndUncoverWhileCurrentHouseHasOutstandingAuctions() (gas: 439352)
[PASS] testFailEndUncoverWhenNotEnoughSystemsCovered() (gas: 439374)
[PASS] testFailEndUncoverWhilePreviousHouseHasOutstandingAuctions() (gas: 423661)
[PASS] testFailAbandonWithoutStarting() (gas: 225100)
[PASS] testFailCoverNonDebtAuctionHouse() (gas: 121464)
[PASS] testFailUpdateCurrentHouseWhenPreviousNotNull() (gas: 590367)
[PASS] testFailUpdateCurrentHouseNewHouseAlreadyUsed() (gas: 617191)
[PASS] testCoverUncoverImmediately() (gas: 277690)
[PASS] testFailUpdateCurrentHouseWhenNotCovered() (gas: 422630)
[PASS] testFailStartUncoverWhileCurrentHouseHasOutstandingAuctions() (gas: 251314)
[PASS] testFailProposeIndefiniteWithFreezeLowerThanCooldown() (gas: 233945)
[PASS] testFailUpdateCurrentHouseSameHouse() (gas: 202565)
[PASS] testCover() (gas: 216391)
[PASS] testModifyParameters() (gas: 97184)
[PASS] testEndUncover() (gas: 712278)
[PASS] testFailGiveUpAuthOwnership() (gas: 18221)
[PASS] testFailUncoverUncovered() (gas: 274032)
[PASS] testFailGiveUpAuthRoot() (gas: 41965)
[PASS] testFailProposeIndefiniteWhenNotCovered() (gas: 32251)
[PASS] testUpdateCurrentAuctionHouse() (gas: 446217)
[PASS] testProposeIndefinitePrintingPermissions() (gas: 225727)
[PASS] testFailUpdateCurrentHouseNewHouseNotDebtAuction() (gas: 332166)
[PASS] testFailEndUncoverBeforeCooldown() (gas: 464537)
[PASS] testGiveUpAuthRoot() (gas: 65037)
[PASS] testEndUncoverWhenDebtHousesUnauthed() (gas: 714141)
[PASS] testFailCoverAlreadyCovered() (gas: 198878)
[PASS] testFailRemovePreviousHouseWhenNotAuthedInProtocolAuth() (gas: 453204)
[PASS] testGiveUpAuthOwnership() (gas: 68618)

Running 12 tests for src/ESM.t.sol:ESMTest
[PASS] test_set_threshold() (gas: 1112221)
[PASS] testFail_set_high_threshold() (gas: 1102994)
[PASS] testFail_shutdown_with_less_than_threshold() (gas: 1154286)
[PASS] testFail_set_low_threshold() (gas: 1102971)
[PASS] testFail_construct_threshold_above_supply() (gas: 39436)
[PASS] test_constructor_with_threshold_setter() (gas: 1359439)
[PASS] testFail_construct_zero_threshold() (gas: 39392)
[PASS] test_shutdown_with_threshold_setter() (gas: 1455650)
[PASS] testFail_shutdown_twice() (gas: 1218406)
[PASS] test_modify_parameters_threshold_setter() (gas: 1554790)
[PASS] test_constructor_without_threshold_setter() (gas: 1109776)
[PASS] test_shutdown_no_threshold_setter() (gas: 1193256)

Running 12 tests for src/test/ChainlinkPriceFeedMedianizer.t.sol:ChainlinkPriceFeedMedianizerTest
[PASS] test_reward_other_multiple_times() (gas: 748265)
[PASS] testFail_null_timestamp() (gas: 32743)
[PASS] test_increased_reward_above_max_second_update() (gas: 236903)
[PASS] test_update_result_and_read() (gas: 206148)
[PASS] test_get_result_with_validity_when_stale() (gas: 185765)
[PASS] test_reward_caller_null_param_first_update() (gas: 170572)
[PASS] test_change_aggregator_address() (gas: 111080)
[PASS] testFail_new_timestamp_smaller_than_last() (gas: 180072)
[PASS] test_change_uint_params() (gas: 48841)
[PASS] test_reward_caller_other_first_update() (gas: 171636)
[PASS] testFail_read_when_stale() (gas: 184695)
[PASS] testFail_negative_price_feed() (gas: 55429)

Running 64 tests for src/test/GebProxyActions.t.sol:GebProxyActionsTest
[PASS] testLockTokenCollateralDGDAndGenerateDebt() (gas: 744893)
[PASS] testExitDGDAfterAuction() (gas: 2139899)
[PASS] testOpenLockTokenCollateralGNTAndGenerateDebtSafe() (gas: 1054505)
[PASS] testRepayAllDebtAfterTaxation() (gas: 1151685)
[PASS] testLockTokenCollateralGNTAndGenerateDebt() (gas: 1057044)
[PASS] testFailSafeLockETHOtherSAFEOwner() (gas: 369044)
[PASS] testLockTokenCollateralOtherSAFEOwner() (gas: 647410)
[PASS] testCoinSavingsAccountSimpleCase() (gas: 1375659)
[PASS] testWipeAndFreeTokenCollateralDGDAndGenerateDebt() (gas: 1042405)
[PASS] testLockTokenCollateralAndGenerateDebt() (gas: 797863)
[PASS] testGenerateDebt() (gas: 772421)
[PASS] testLockTokenCollateral() (gas: 546702)
[PASS] testExitTokenCollateralAfterAuction() (gas: 2212003)
[PASS] testTransferSAFEOwnership() (gas: 345809)
[PASS] testFailSafeRepayDebtAllOtherSAFEOwner() (gas: 896070)
[PASS] testRepayDebt() (gas: 984228)
[PASS] testSafeRepayDebt() (gas: 988331)
[PASS] testLockTokenCollateralGNT() (gas: 820971)
[PASS] testGiveSAFEToProxy() (gas: 1445356)
[PASS] testSafeLockTokenCollateral() (gas: 550794)
[PASS] testRepayDebtAfterTaxation() (gas: 1143963)
[PASS] testModifySAFECollateralization() (gas: 711185)
[PASS] testGlobalSettlement() (gas: 4175989)
[PASS] testApproveDenySAFEModification() (gas: 71094)
[PASS] testLockBackAfterCollateralAuction() (gas: 2112142)
[PASS] testSafeRepayAllDebt() (gas: 992223)
[PASS] testLockETHOtherSAFEOwner() (gas: 573952)
[PASS] testQuitSystem() (gas: 892790)
[PASS] testOpenLockTokenCollateralAndGenerateDebt() (gas: 773654)
[PASS] testRepayDebtAndFreeETH() (gas: 1057319)
[PASS] testOpenLockTokenCollateralGNTAndGenerateDebt() (gas: 1033640)
[PASS] testLockTokenCollateralDGD() (gas: 497462)
[PASS] testSafeLockETH() (gas: 477506)
[PASS] testExitETHAfterCollateralAuction() (gas: 2135751)
[PASS] testFreeTokenCollateralDGD() (gas: 635179)
[PASS] testEnterSystem() (gas: 694521)
[PASS] testFailSafeLockTokenCollateralOtherSAFEOwner() (gas: 430187)
[PASS] testAllowUrn() (gas: 72743)
[PASS] testGiveSAFEAllowedUser() (gas: 484363)
[PASS] testRepayAllDebtAndFreeETH() (gas: 1058473)
[PASS] testCreateSAFE() (gas: 245927)
[PASS] testFailSafeRepayDebtOtherSAFEOwner() (gas: 896111)
```

```
[PASS] testOpenLockETHAndGenerateDebt() (gas: 699331)
[PASS] testGiveSAFEToNewProxy() (gas: 1452861)
[PASS] testFailGiveSAFEToNewContractProxy() (gas: 366487)
[PASS] testLockETHAndGenerateDebt() (gas: 724872)
[PASS] testCoinSavingsAccountRounding2() (gas: 1349709)
[PASS] testFreeTokenCollateralGNT() (gas: 946063)
[PASS] testOpenLockTokenCollateralGNTAndGenerateDebtSafeTwice() (gas: 1538143)
[PASS] testRepayAllDebtAndFreeTokenCollateral() (gas: 1092376)
[PASS] testRepayDebtAll() (gas: 988119)
[PASS] testFreeETH() (gas: 641254)
[PASS] testTransfer() (gas: 139597)
[PASS] testFreeTokenCollateral() (gas: 675686)
[PASS] testRepayAllDebtAfterTaxation2() (gas: 5285346)
[PASS] testCoinSavingsAccountRounding() (gas: 1430818)
[PASS] testTransferCollateral() (gas: 442143)
[PASS] testPreventHigherCoinOnRepayDebt() (gas: 1124167)
[PASS] testRepayDebtAndFreeTokenCollateral() (gas: 1091446)
[PASS] testRepayDebtOtherSAFEOwner() (gas: 1077612)
[PASS] testGenerateDebtAfterCollectingTax() (gas: 951330)
[PASS] testLockETH() (gas: 473242)
[PASS] testMoveSAFE() (gas: 1074189)
[PASS] testCoinSavingsAccountWithdrawAll() (gas: 1326019)

Running 7 tests for src/test/delegate_roles.t.sol:DSDelegateRolesTest
[PASS] testFailSetAuthorityToSelf() (gas: 4988)
[PASS] testBasics() (gas: 161630)
[PASS] testRoot() (gas: 90999)
[PASS] testDelegateBasics() (gas: 141180)
[PASS] testDelegatePublicCapabilities() (gas: 75312)
[PASS] testDelegateRoot() (gas: 72619)
[PASS] testPublicCapabilities() (gas: 93713)

Running 3 tests for src/test/roles.t.sol:DSRolesTest
[PASS] testBasics() (gas: 124194)
[PASS] testRoot() (gas: 72997)
[PASS] testPublicCapabilities() (gas: 75756)

Running 12 tests for src/FsmGovernanceInterface.t.sol:FsmGovernanceInterfaceTest
[PASS] testStopOwner() (gas: 32870)
[PASS] testSetOwner() (gas: 10898)
[PASS] testFailSetFsm() (gas: 6653)
[PASS] testSetFsm() (gas: 29911)
[PASS] testSetAuthority() (gas: 10918)
[PASS] testFailSetOwner() (gas: 6531)
[PASS] testVerifySetup() (gas: 12857)
[PASS] testFailSetAuthority() (gas: 6552)
[PASS] testStopAuthorized() (gas: 40216)
[PASS] testFailCollateralTypeWithoutFsm() (gas: 12827)
[PASS] testFailStopNoAuthority() (gas: 14588)
[PASS] testFailStopCallerNotAuthorized() (gas: 167811)

Running 27 tests for src/test/UniswapBasicConverterAveragePriceFeedMedianizer.t.sol:UniswapConverterBasicAveragePriceFeedMedianizerTest
[PASS] test_read_after_passing_granularity() (gas: 14434587)
[PASS] test_update_treasury_reward_treasury() (gas: 603702)
[PASS] test_simulate_same_prices() (gas: 23196293)
[PASS] testFail_small_granularity() (gas: 64702)
[PASS] test_update_result() (gas: 692045)
[PASS] test_change_converter_feed() (gas: 21228)
[PASS] test_deep_liquidity_one_round_simulate_prices() (gas: 14763881)
[PASS] testFail_update_ETHRAI_again_immediately() (gas: 357020)
[PASS] test_update_treasury_throws() (gas: 1026116)
[PASS] testFail_update_result_USDC_converter_invalid_value() (gas: 123196)
[PASS] test_deep_liquidity_one_round_simulate_prices_erratic_delays() (gas: 12673161)
[PASS] testFail_read_raieth_before_passing_granularity() (gas: 348528)
[PASS] testFail_update_USDCRAI_again_immediately() (gas: 357157)
[PASS] test_correct_setup() (gas: 184518)
[PASS] test_update_result_converter_throws() (gas: 784323)
[PASS] testFail_negative_reward_increase() (gas: 84931)
[PASS] testFail_read_raiusdc_before_passing_granularity() (gas: 348667)
[PASS] test_thin_liquidity_multi_round_simulate_prices() (gas: 55847942)
[PASS] test_thin_liquidity_multi_round_simulate_prices_erratic_delays() (gas: 53240833)
[PASS] testFail_max_reward_smaller_than_base() (gas: 84920)
[PASS] testFail_window_not_evenly_divisible() (gas: 84811)
[PASS] test_get_result_after_passing_granularity() (gas: 14434609)
[PASS] test_simulate_same_prices_erratic_delays() (gas: 20524993)
[PASS] test_thin_liquidity_one_round_simulate_prices() (gas: 14668969)
[PASS] test_get_result_before_passing_granularity() (gas: 673751)
[PASS] test_simulate_denominator_token_positive_price_jump() (gas: 14886794)
[PASS] testFail_update_result_ETH_converter_invalid_value() (gas: 123078)

Running 30 tests for src/test/UniswapConsecutiveSlotsPriceFeedMedianizer.t.sol:UniswapConsecutiveSlotsPriceFeedMedianizerTest
[PASS] test_read_after_passing_granularity() (gas: 14348127)
[PASS] test_update_treasury_reward_treasury() (gas: 640810)
[PASS] test_simulate_same_prices() (gas: 28228086)
[PASS] testFail_small_granularity() (gas: 64664)
[PASS] test_update_result() (gas: 682779)
[PASS] test_change_converter_feed() (gas: 21336)
[PASS] test_deep_liquidity_one_round_simulate_prices() (gas: 14895943)
[PASS] testFail_update_ETHRAI_again_immediately() (gas: 353351)
[PASS] test_thin_liquidity_multi_round_simulate_prices_erratic_then_normal_delays() (gas: 147831311)
[PASS] test_update_treasury_throws() (gas: 1046570)
[PASS] testFail_update_result_USDC_converter_invalid_value() (gas: 120547)
[PASS] test_deep_liquidity_one_round_simulate_prices_erratic_delays() (gas: 14826589)
[PASS] testFail_read_raieth_before_passing_granularity() (gas: 324855)
[PASS] testFail_update_USDCRAI_again_immediately() (gas: 353510)
[PASS] test_correct_setup() (gas: 184566)
[PASS] test_update_result_converter_throws() (gas: 754314)
[PASS] testFail_negative_reward_increase() (gas: 84893)
[PASS] testFail_read_raiusdc_before_passing_granularity() (gas: 324972)
[PASS] test_thin_liquidity_multi_round_simulate_prices() (gas: 73749151)
[PASS] test_simulate_same_prices_erratic_then_normal_delays() (gas: 55326940)
[PASS] test_thin_liquidity_multi_round_simulate_prices_erratic_delays() (gas: 73870783)
[PASS] testFail_max_reward_smaller_than_base() (gas: 84873)
[PASS] testFail_window_not_evenly_divisible() (gas: 84861)
[PASS] test_get_result_after_passing_granularity() (gas: 14348127)
[PASS] test_simulate_same_prices_erratic_delays() (gas: 27935115)
[PASS] test_thin_liquidity_one_round_simulate_prices() (gas: 14580682)
[PASS] test_get_result_before_passing_granularity() (gas: 645690)
[PASS] test_simulate_denominator_token_positive_price_jump() (gas: 28344969)
[PASS] test_deep_liquidity_one_round_simulate_prices_erratic_then_normal_delays() (gas: 29618691)
[PASS] testFail_update_result_ETH_converter_invalid_value() (gas: 120451)

Running 11 tests for src/test/CollateralJoin.t.sol:CollateralJoin6Test
[PASS] test_set_positive_allowance_when_positive() (gas: 79321)
[PASS] test_set_positive_allowance_when_zero() (gas: 53059)
[PASS] test_exit_when_negative_collateral_joined() (gas: 257977)
[PASS] test_exit_when_positive_collateral_joined() (gas: 168433)
[PASS] test_join_when_negative_balance_positive_allowance() (gas: 345634)
[PASS] test_exit_when_zero_allowance() (gas: 174470)
[PASS] test_correct_setup() (gas: 27909)
[PASS] test_exit_when_positive_allowance() (gas: 164659)
[PASS] testFail_join_above_allowance_positive_balance() (gas: 141939)
[PASS] testFail_join_when_negative_balance_zero_allowance() (gas: 232455)
[PASS] testFail_join_when_zero_allowance() (gas: 6789)

Running 45 tests for src/test/GebDeploy.t.sol:GebDeployTest
[PASS] testFailModifySAFECollateralizationDust() (gas: 49472230)
[PASS] testModifySAFECollateralizationDrawCoinCollateralLimit() (gas: 49443175)
[PASS] testFailTransferSAFECollateralAndDebtUnsafeDst() (gas: 49402719)
[PASS] testLiquidateSAFE() (gas: 50141044)
[PASS] testDebtAuctionHouse() (gas: 51777970)
[PASS] testDeployIndex() (gas: 49170278)
[PASS] testModifySAFECollateralizationFromAnotherUser() (gas: 49449574)
[PASS] testFailMissingSAFEEngine() (gas: 4901)
[PASS] testModifySAFECollateralizationDrawCoinLimit() (gas: 49421450)
[PASS] testJoinETH() (gas: 49307923)
[PASS] testModifySAFECollateralizationDrawCoinCollateral() (gas: 49557318)
[PASS] testFailLiquidateSAFE() (gas: 49460370)
[PASS] testEnglishCollateralAuctionHouse() (gas: 51080055)
[PASS] testFailModifySAFECollateralizationFromAnotherUser() (gas: 49329422)
[PASS] testTransferSAFECollateralAndDebt() (gas: 49458578)
[PASS] testStableAuthFixedDiscountAuctionHouse() (gas: 52055051)
[PASS] testModifySAFECollateralizationDrawCoin() (gas: 49547115)
[PASS] testSetPauseAuthorityAndDelay() (gas: 49257824)
[PASS] testModifySAFECollateralizationPaybackDebt() (gas: 49752180)
[PASS] testIndexAuthFixedDiscountAuctionHouse() (gas: 50567472)
[PASS] testFailMissingTaxationAndAuctions() (gas: 8125664)
[PASS] testExitCollateral() (gas: 49361875)
[PASS] testFailTransferSAFECollateralAndDebtFromOtherUsr() (gas: 49405659)
[PASS] testFailMissingEnd() (gas: 19127617)
[PASS] testIndexAuthEnglishAuctionHouse() (gas: 49461885)
[PASS] testTransferSAFECollateralAndDebtFromOtherUsr() (gas: 49444281)
[PASS] testLiquidateSAFEPartial() (gas: 50141002)
[PASS] testFixedDiscountCollateralAuctionHouse() (gas: 51904330)
[PASS] testJoinCollateral() (gas: 49327128)
[PASS] testStableAuthEnglishAuctionHouse() (gas: 50949420)
[PASS] testFailTransferSAFECollateralAndDebtDustSrc() (gas: 49580202)
[PASS] testFailTransferSAFECollateralAndDebtUnsafeSrc() (gas: 49402739)
[PASS] testPreSettlementSurplusAuctionHouse() (gas: 50482715)
[PASS] testFailMissingLiquidator() (gas: 19126940)
[PASS] testFailModifySAFECollateralizationDrawCoinLimit() (gas: 49325101)
[PASS] testSetPauseDelay() (gas: 49241592)
[PASS] testFailTransferSAFECollateralAndDebt() (gas: 49402695)
[PASS] testSetPauseAuthority() (gas: 49225159)
[PASS] testFailTransferSAFECollateralAndDebtDustDst() (gas: 49578459)
[PASS] testFireESM() (gas: 49327581)
```

```
[PASS] testSettlementPartialLiquidation() (gas: 51990820)
[PASS] testFailModifySAFECollateralizationDrawCoinCollateralLimit() (gas: 49346847)
[PASS] testDeployStable() (gas: 50615588)
[PASS] testExitETH() (gas: 49334941)
[PASS] testSettlementFullLiquidation() (gas: 51990811)
```

Code Coverage

The `dapptools` suite (used to develop and run tests for all the repositories that were in scope for this audit), does not contain a feature that can generate a test coverage summary similar to `solidity-coverage`. Therefore, we were not able to measure the coverage for this project. However, we have noticed during our audit that the coverage is less than 100%. We strongly recommend increasing the number of tests to cover all branches in the code such that all the functionality is tested each time there is a new change to the code. This is the best way to detect functional code defects, which affect the business logic.

Appendix

File Signatures

The following are the SHA-256 hashes of the reviewed files. A file with a different SHA-256 hash has been modified, intentionally or otherwise, after the security review. You are cautioned that a different SHA-256 hash could be (but is not necessarily) an indication of a changed condition or potential vulnerability that was not within the scope of the review.

Contracts

```
05013e5cabd66aa0ec747684c177185f2cf47a9ef168051f6f1bfe6b7adb34c2 ./AdvancedTokenAdapters.sol
7b88dd3878ede58812a37e08c113cd035db943bb4337c68dbd05fb477137a3c1 ./UniswapV2Factory.sol
750172fe6b061e6fce0e8df68e4ea2046297f8924439480e449495f3e6613b8e ./UniswapV2Pair.sol
0c913a68831ab4bec072d72c797cda0f274ee8226f3ec9e9c9b0c3edcb6d4542 ./UniswapV2Router02.sol
283caf1f0db12878dbaa5fe54f3f3227efffaf59e8bf6f79741d63e541a79006 ./UniswapConsecutiveSlotsPriceFeedMedianizer.sol
5b103d3e304e783aef3bbdebd63d36fd6a66653d43cb1df8c045f41c6c971f4b ./UniswapConverterBasicAveragePriceFeedMedianizer.sol
5898539c70eacf49ddaca63d0021fb999f5cd5e4d716d308a5cbb59f6769f9c3 ./GebProxyActions.sol
6786ede09d038024bfd9180b2250c02ad2f60788c16f00fca74d0edaeb821a13 ./ProtocolTokenAuthority.sol
e3977d2bd082589fa0e6e1ddc4fd308c578870faf161759cef812b1ae2c4af40 ./GebPrintingPermissions.sol
70c99b2c79d46bdd34caac25e0508bfa97688250d8ee3d14cb2c6c02560b0219 ./FsmGovernanceInterface.sol
95031180ec7ce971bdc5f4aaa950979cb9a9531cd6f2b993b462341807f55178 ./GebDeploy.sol
57b079acbb45a39c374f51884c2644556fc941848a4f590ccdac2e772530879d ./GovActions.sol
bf7a3c7b2ee8fb64b02ddda6bb49a8a92594f8aa8697627c1cad4c130e67737b ./ChainlinkPriceFeedMedianizer.sol
1336746b76fedcd2ad48bd4475525744291ad72f011bb6d2c1a5a0e0604c3728 ./AggregatorInterface.sol
98c66eddb483764a41955dafd479a1eae370324351593c9a012194a2d756587d ./roles.sol
301a6bfdde0c8013778cc04d72d68555e870b4831d048f6ae9e5143fc84cc695 ./delegate_roles.sol
3e5394be78c48187c9afcc7028516cb7b955aa5a34852ee269549c6ec7e876ab ./GebSafeManager.sol
d15b073d4509dc4584d13c1e0c5b83417f7597415b278654d56a63ab1a483d21 ./GebSafeManager.t.sol
460693fbe61ee864126d561383c4f8b86daabee9ebf5b835935652fcecb326df ./GetSafes.sol
0487e488188b17ec87280a18a2bbc2e46ef83e03f72390e9201daa2c74225e30 ./ESM.t.sol
a32336d2a4933e18febc2623125c3b8141fcf781171f20209736370159b9f650 ./ESM.sol
```

Tests

```
527329bbbc00f74d7365a9273967bf62a1c90b796950873c752dedfdf76c61de ./delegate_roles.t.sol
be8b2d21d38245193bba856cb79fb0e899cffba1dfee3a78db45bbad818bb7ef ./roles.t.sol
0487e488188b17ec87280a18a2bbc2e46ef83e03f72390e9201daa2c74225e30 ./ESM.t.sol
d15b073d4509dc4584d13c1e0c5b83417f7597415b278654d56a63ab1a483d21 ./GebSafeManager.t.sol
443bc3a5526d8095dfbfe71f0886721db93b76c4c452a17a84961579d811994b ./ChainlinkPriceFeedMedianizer.t.sol
6c0d8c0d00430d42266909f9b1e3aa8b99a134f3360ddb0ccb2b718217310997 ./FsmGovernanceInterface.t.sol
9428243062665403c5c343cb75ad6a8c64400083ae48de96247eff5962b524cb ./GebPrintingPermissions.t.sol
963f5ad128f83b5ecd79292625192770077d8689810ab1d998345e890a488890 ./ProtocolTokenAuthority.t.sol
61d20b691ec0ddbfad9ecbbb4652e0bcdf5673be480cef71a28850250150c1c8 ./ProtocolTokenAuthorityHevm.t.sol
7b6e7cd4a2d78ea48066f5d37db5d53c8e64dd784bf0fa0052702830454bf69e ./GebProxyActions.t.sol
395b3ad614fa2f2d4db4b69e1f81f3963965a01f8cb994d2ffa9aed1d014efb6 ./tokens.sol
b9e26b0e59bb8ae1bd1bfe6bdb63c25d7ad811e46455141da179c2c449a0f487 ./UniswapBasicConverterAveragePriceFeedMedianizer.t.sol
8cd1693476b3740cdba22654a16ea31f1796ee4183882f14572865178ec765ef ./UniswapConsecutiveSlotsPriceFeedMedianizer.t.sol
8cd1693476b3740cdba22654a16ea31f1796ee4183882f14572865178ec765ef ./test/UniswapConsecutiveSlotsPriceFeedMedianizer.t.sol
b9e26b0e59bb8ae1bd1bfe6bdb63c25d7ad811e46455141da179c2c449a0f487 ./test/UniswapBasicConverterAveragePriceFeedMedianizer.t.sol
```

Changelog

- 2020-09-21 - Initial report based on the following commits: (1) https://github.com/reflexer-labs/geb-protocol-token-authority/commit/0ba46736563 4fdad12108719aae22a39066f818d, (2) https://github.com/reflexer-labs/geb-printing-permissions/commit/3889754b0c2ed0768ef5b6860e78f4e8845117e8, (3) https://github.com/reflexer-labs/esm/commit/0addd5f5e0e0b8b879a112cc2ec4f37ed1e5ebb1, (4) https://github.com/reflexer-labs/geb-chainlink-median/commit/a93377cf6e09eefce9e5c86ca14d811914ba8a40, (5) https://github.com/reflexer-labs/geb-safe-manager/commit/3ff7baca2d377c5b19fd948a86c519adc1e40070, (6) https://github.com/reflexer-labs/geb-proxy-actions/commit/e71ab42a769b3d93de068276635d546b593d5974, (7) https://github.com/reflexer-labs/ds-roles/commit/0c2c18808f3d7faacb32527253f17991583e31e7, (8) https://github.com/reflexer-labs/geb-fsm-governance-interface/commit/b503d55c7d375034033de530bff65731be15ebc2, (9) https://github.com/reflexer-labs/geb-uniswap-median/commit/35d6783ee342bdc047f2c53129064 6e561e644ef, (10) https://github.com/reflexer-labs/geb-deploy/blob/0da2b6d99af905beb2d5fc7134314dc9064cd78b
- 2020-10-16 - Updated report based on commits provided by dev team (see table on first page of the report for the commit hash corresponding to each repo)
- 2020-11-02 - Updated report based on response received from dev team. Only the `geb-depoly` repo commit hash was updated from `f59579611cdb02ef8b2e5456fd7c687b89d2052c` to `dba8e28ef289671502d5bbb8ad3fd68e52cbd561`

About Quantstamp

Quantstamp is a Y Combinator-backed company that helps to secure blockchain platforms at scale using computer-aided reasoning tools, with a mission to help boost the adoption of this exponentially growing technology.

With over 1000 Google scholar citations and numerous published papers, Quantstamp's team has decades of combined experience in formal verification, static analysis, and software verification. Quantstamp has also developed a protocol to help smart contract developers and projects worldwide to perform cost-effective smart contract security scans.

To date, Quantstamp has protected $5B in digital asset risk from hackers and assisted dozens of blockchain projects globally through its white glove security assessment services. As an evangelist of the blockchain ecosystem, Quantstamp assists core infrastructure projects and leading community initiatives such as the Ethereum Community Fund to expedite the adoption of blockchain technology.

Quantstamp's collaborations with leading academic institutions such as the National University of Singapore and MIT (Massachusetts Institute of Technology) reflect our commitment to research, development, and enabling world-class blockchain security.