Quantstamp Security Assessment Certificate

November 3rd 2022 — Quantstamp Verified

Hifi Governance

This audit report was prepared by Quantstamp, the leader in blockchain security.

Executive Summary

Type	Governance
Auditors	Roman Rohleder, Research Engineer Valerian Callens, Auditor Nikita Belenkov, Security Auditor Mostafa Yassin, Security Engineer
Timeline	2022-10-18 through 2022-11-03
EVM	Paris
Languages	Solidity
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	COMMITLOG
Documentation Quality	Low
Test Quality	High
Diff/Fork information	Contract `Hifi.sol` is forked off https://github.com/Uniswap/governance/blob/eabd8c71ad01f61fb54ed6945162021ee419998e/contracts/Uni.sol and the rest is forked off https://github.com/ampleforth/Forth/tree/e84984238700a6698e8f06a2a74a04f298f1aab82/, both of which are based on Compound's governance code.

Source Code

Repository	Commit
hifi-finance/hifi-governance	8b1f667 initial audit
hifi-finance/hifi-governance	7e96788 fixes

Total Issues	**24**	(3 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	**3**	(1 Resolved)
Low Risk Issues	**6**	(1 Resolved)
Informational Risk Issues	**9**	(1 Resolved)
Undetermined Risk Issues	**6**	(0 Resolved)

0 Unresolved
21 Acknowledged
3 Resolved

- High Risk
- Medium Risk
- Low Risk
- Informational
- Undetermined
- Unresolved
- Acknowledged
- Fixed
- Mitigated

Summary of Findings

Hifi Governance is a governance voting protocol, based on Compound Governance with additional changes such as interaction with other parts of the Hifi protocol.
During the audit 24 issues have been identified, ranging from undetermined to medium severity. Issues related to the way signatures are processed, issues related to the assumptions made about interactions with the network as well as issues intrinsic to contract-related vulnerabilities were found. Additionally, code documentation and best practices related issues were identified. While the test suite is of high quality, reaching nearly 100% statement coverage, the overall branch coverage is only at 73% and should be improved.
We highly recommend addressing all issues and improving the test suite before deployment.

Update: After the fix-review the developers fixed only three out of 24 issues and acknowledged the rest and we therefore want to point out that when deployed in this state multiple risks still remain.

ID	Description	Severity	Status
QSP-1	Signature Malleability	^ Medium	Acknowledged
QSP-2	Behavior of `target` Addresses Not Always Predictable	^ Medium	Acknowledged
QSP-3	Dynamic Total Supply Can Break the Static Voting Thresholds	^ Medium	Fixed
QSP-4	Implementation Can Be Changed via Proxy Selector Clashing	⌄ Low	Acknowledged
QSP-5	Human Error Could Block the Minting Process	⌄ Low	Acknowledged
QSP-6	Proposals Vulnerable to Sandwich Attacks	⌄ Low	Acknowledged
QSP-7	Delegations Made After the Beginning of Voting Are Not Accounted For.	⌄ Low	Acknowledged
QSP-8	Finality Limit Should Be Added	⌄ Low	Acknowledged
QSP-9	Return Value of `transferFrom()` Is Not Verified	⌄ Low	Fixed
QSP-10	Missing Input Validation	O Informational	Acknowledged
QSP-11	Privileged Roles and Ownership	O Informational	Acknowledged
QSP-12	Unnecessary Use of `add*()` and `sub*()` in Solidity `0.8.x`	O Informational	Fixed
QSP-13	`Timelock.queueTransaction()` and `Timelock.cancelTransaction()` Callable Multiple Times	O Informational	Acknowledged
QSP-14	Unlocked Pragma	O Informational	Acknowledged
QSP-15	A `Proposal` Can Be Canceled Several Times	O Informational	Acknowledged
QSP-16	Time Calculated Assuming Constant Block Frequency	O Informational	Acknowledged
QSP-17	Order of Evaluation of Expressions in Events Is Unexpected	O Informational	Acknowledged
QSP-18	Block Timestamp Manipulation	O Informational	Acknowledged
QSP-19	Arbitrary Transactions by Initial `Timelock.admin`	? Undetermined	Acknowledged
QSP-20	First Proposer Can Create Two Consecutive Proposals	? Undetermined	Acknowledged
QSP-21	No Hifi Received when Small Amount Swapped	? Undetermined	Acknowledged
QSP-22	Swaps Can Be Blocked if Contract MfT Is Paused	? Undetermined	Acknowledged
QSP-23	Potential Reentrancy	? Undetermined	Acknowledged
QSP-24	`ecrecover()` Can Return a Random Address Instead of 0 for an Invalid Signature	? Undetermined	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.

DISCLAIMER:
If the final commit hash provided by the client contains features that are not within the scope of the audit or an associated fix review, those features are excluded from consideration in this report.

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 v0.8.3

Steps taken to run the tools:

1. Install the Slither tool: `pip3 install slither-analyzer`
2. Run Slither from the project directory: `slither .`

Findings

QSP-1 Signature Malleability

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `Hifi.sol`, `GovernorBravoDelegate.sol`

Description: The given implementation of signature verification in `Hifi.permit()`, `Hifi.delegateBySig()` and - `GovernorBravoDelegate.castVoteBySig()` using `ecrecover` directly is prone to signature malleability.
This means that you might be able to create a second valid signature for the same data. This is no direct issue for the current implementation, but it is a good practice to protect against.

Recommendation: Consider using a secure wrapper like OpenZeppelin's ECDSA utility library, which performs additional security checks on the signature parameters.

QSP-2 Behavior of `target` Addresses Not Always Predictable

Severity: Medium Risk

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`

Description: A `Proposal` is composed of a sequence of transactions items, each being composed of four items:

1. `target`: the address of the target contract;
2. `value`: the number of native tokens to be sent;
3. `signature`: the first part of the hash representing the function of `target` to be called;
4. `calldata`: the values to be used as parameters when calling the function matching the `signature`;

There is a delay between the moment a `Proposal` is issued via the function `propose()` and the moment it is executed. Even if smart contracts are usually considered immutable, the result of a given transaction can actually differ depending on several factors, including:

- the block including the transaction;
- the position of the transaction within that block;
- the code of the target contract;
- the current state of the contract;

Based on this information, it becomes possible for a malicious proposer to modify the behavior of the underlying transactions of his `Proposal` between the issuance, the voting period, and the execution. Here are some examples, where the `target` address can be:

- a proxy contract with an upgradeable implementation contract;
- a metamorphic contract (https://blog.openzeppelin.com/the-state-of-smart-contract-upgrades/#metamorphic-contracts);
- an EOA that is a precomputed address where a contract can be deployed using the instruction `CREATE2`. This contract could have a function `fallback()` to react if a transfer of native tokens happens;

The impact is hard to assess, but we could imagine reentrancy attacks that execute more than once when the proposal is executed.

Recommendation: Different options could reduce the risks described above:

1. Double-check the `targets` of a given `Proposal` once it has been issued. If any address looks suspicious, make an official announcement and ask the community to vote against it.
2. Alternatively, consider adding a whitelist or a blacklist of addresses that can be considered as `targets`. Only a multi-sig of trusted users (team members and community members) could be authorized to update these lists.
3. Add a mechanism to prevent reentrancies in the function `execute()`.

QSP-3 Dynamic Total Supply Can Break the Static Voting Thresholds

Severity: Medium Risk

Status: Fixed

File(s) affected: `GovernorBravoDelegate.sol`

Description: The value of the voting thresholds `MIN_PROPOSAL_THRESHOLD`, `MAX_PROPOSAL_THRESHOLD` and `quorumVotes` is hardcoded and constant.
The correct values should be set carefully and by taking into account the total supply of the governance token. Indeed, the settings should in the same time:

- keep the governance participation accessible to a maximum of people;
- limit the number of users who can create a `Proposal`;

As the token HIFI can be minted and burnt, its total supply can change. It implies that the voting thresholds could become more or less restrictive.
In extreme cases, the governance could be blocked if, for instance, its total supply goes lower than the state variable `quorumVotes`.

Recommendation: Consider defining ratios taking into account the total supply of the token HIFI. Exploring the ratios used by other protocols following the `GovernorBravo` could help.
Consider also the option to make these thresholds mutable, or dynamically based on the current total supply.

Update: Fixed in commit 7e96788 by making `MIN_PROPOSAL_THRESHOLD` and `MAX_PROPOSAL_THRESHOLD` dependant on the current total supply of the HiFi token, as suggested.

QSP-4 Implementation Can Be Changed via Proxy Selector Clashing

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GovernorBravoDelegator.sol`, `GovernorBravoDelegate.sol`

Description: The project uses a transparent proxy pattern. Calls are made via the proxy address and an identifier called the selector is used to define the function to execute. Then, the function `fallback()` is executed and makes a `delegatecall` to the implementation contract only if no function with the given selector can be found in the proxy contract. However, the same selector can represent different functions signatures. That issue is called "proxy selector clashing".
The selector of the function `GovernorBravoDelegator._setImplementation(address implementation_)` is `0xBB913F41` (details here: https://solidity-by-example.org/function-selector/).
If in a future upgrade, a `function XXX(type param_p)` identified by the same selector `0xBB913F41` and one parameter is added to the implementation contract, and if someone wants to call it via the proxy, the function `fallback()` would not be executed because that selector can be found in the current proxy contract. If that happens, the function _setImplementation() would be executed and the address of the implementation contract would instantly be changed to the value of `param_p`.
The severity of that issue is [Low], mainly due to a limited likelihood.

Recommendation: Consider double-checking the underlying selector of any function added to the implementation contract when an upgrade is done. If a selector clash is found, slightly rename the function.

QSP-5 Human Error Could Block the Minting Process

Severity: Low Risk

Status: Acknowledged

File(s) affected: `Hifi.sol`

Description: The address `Hifi.minter` has the right to mint tokens HIFI. It can be modified via the function `setMinter()`. However, if an erroneous value is used as the new `minter`, the mint privileges will be transferred to the new incorrect address and it will not be possible anymore to mint tokens.

Recommendation: Consider adding an extra step to update `Hifi.minter`. An option could be to use `Ownable2Step` designed by OpenZeppelin:
https://github.com/OpenZeppelin/openzeppelin-contracts/blob/master/contracts/access/Ownable2Step.sol

QSP-6 Proposals Vulnerable to Sandwich Attacks

Severity: Low Risk

Status: Acknowledged

File(s) affected: `Timelock.sol`, `GovernorBravoDelegate.sol`

Description: A `Proposal` is a sequence of transactions that should be executed on-chain. It can have eight different statuses: `Pending`, `Active`, `Defeated`, `Succeeded`, `Queued`, `Executed`, `Canceled`, `Expired`. The lifecycle of a `Proposal P` is the following:

1. `P` is created by someone who has more voting powers than the `proposalThreshold`. Its status is `Pending`;
2. Votes can be cast during a `votingPeriod`, once the `votingDelay` has passed. Its status is `Active`;
3. At the end of the `votingPeriod`, its status is either `Succeeded` or `Defeated`, depending on the number of votes obtained;
4. If `P` has the status `Succeeded`, it can be queued by anyone;
5. If `P` has the status `Queued`, it can be executed by anyone. But only during a defined period of time. If we consider `q` the moment it was queued, its execution period is between `q + Timelock.delay` and `q + Timelock.GRACE_PERIOD`. That process will revert if at least one transaction with the same exact parameters has already been queued in the same block;
6. If `P` has the status `Queued` and was not executed before the end of the `GRACE_PERIOD`, it gets the status `Expired`;
7. At every moment, `P` can get canceled, except if is has the status `Executed`;

Based on that process, an attacker who disagrees with a given `Proposal P1` made by another proposer could quickly propose a `Proposal P2` that looks legit, but contains at least one transaction `T` also in `P1`. Then, he could prevent any execution of `P1` by using a sandwich attack that leverages the issue related to proposal multi-cancellations (see QSP-15. Exploit described below).
The severity of that issue is [Low]:

- In terms of impact, it could be High if `P1` is time-sensitive;
- In terms of likelihood, it is assessed as Low for the following reasons:
  . we should assume that both `P1` and `P2` get enough votes For to reach the status `Succeeded`;
  . it requires an advanced level of expertise to monitor the mempool and successfully perform a sandwich attack;
  . the attacker would need enough voting power to propose `P2`;
  . the fact that `P2` has a malicious purpose should remain undetected;
  . the attack would work once, but `P1` could be proposed again, and queued this time using Flashbots to prevent a sandwich attack.

Exploit Scenario: 1. A User issues a `Proposal P1`. Let `Tc` be the most basic transaction of `P1`.

1. The Attacker quickly detects that `P1` goes against his interests. He quickly issues a `Proposal P2` that seems legit, but that contains `Tc`.
2. Assumption: both `P1` and `P2` are accepted by the community;
3. The Attacker monitors the mempool to detect any transaction `Tq` that queues `P1`;
4. When `Tq` is detected, the Attacker front-runs `Tq` (paying more gas than `Tq`) with a transaction `Tq'` that queues `P2` and cancels it right after. Within the same block, the Attacker back-runs `Tq` (paying less gas than `Tq`) by sending several transactions `Tq''` that cancel `P2`, using the issue regarding proposal multi-cancellation described below;
5. If `Tq'`, `Tq` and `Tq''` are executed in the same order within the same block, the hash representing `Tc` in the mapping `Timelock.queuedTransactions` will be set to `false` by `Tq''`;
6. As `P1` can be executed only if the hash of each of its transactions is `true` in `Timelock.queuedTransactions`, any further execution of `P1` will fail;
7. After some time, `P1` will finally get the status `Expired`;

Recommendation: Consider fixing the vulnerability related to proposal multi-cancellation (QSP-15).

QSP-7 Delegations Made After the Beginning of Voting Are Not Accounted For.

Severity: Low Risk

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`

Description: In `GovernorBravoDelegate.castVoteInternal()`, the votes are casted on a proposal. The vote amount is calculated via the following function:
`hifi.getPriorVotes(voter, proposal.startBlock)`.
This means that any changes to delegations made after the proposal voting started are not accounted for.

Recommendation: Verify if this is intentional behaviour of the protocol.

QSP-8 Finality Limit Should Be Added

Severity: Low Risk

Status: Acknowledged

File(s) affected: `Hifi.sol`

Description: In `Hifi.getPriorVotes()`, a require statement checks that the block has been finalized before any decision is made based on the vote state in that block.
In this case, finality is assumed to be the next block after the current one, but that is not the case with the shift to Proof of Stake.

Recommendation: It is best to wait for at least 32 blocks or 1 epoch so that the said block becomes `justified`. `justified` or `safe` blocks are unlikely to be reorganized. If one wants to be extra safe, one can wait for 64 blocks or 2 epochs, so the block becomes `finalized`.

Update: Acknowledged by the developers with:

```
In a propose / execute scheme there can only be 3 types of reorgs :
- reorg doesn't revert any of propose / execute
(no issues)
- reorg reverts execute
- reorg reverts propose & execute
Reverting execute or propose + execute is just an
inconvenience, since the proposal / execution could
be redone another time, and all the original txs would
be "refunded" ( as they would have never happened
in the new blockchain state ) so there is no loss of
value and no possibility for double proposal / double
voting or double execution.
There could be a loss of opportunity, but we cannot
control reorgs anyways...
We believe this is an acceptable risk.
```

QSP-9 Return Value of `transferFrom()` Is Not Verified

Severity: Low Risk

Status: Fixed

File(s) affected: `Hifi.sol`

Description: In `Hifi.swap()` a transfer of MFT tokens occurs and `mft.transferFrom(msg.sender, address(1), mftAmount)` is called. MFT token implementation states that the successful transfer would return `True`. This should be verified.

Recommendation: Verify the return value of the `transferFrom()` function.

Update: Fixed in commit 91cd4bb by checking the return value by wrapping the transfer function inside a `require()` statement, as suggested.

QSP-10 Missing Input Validation

Severity: Informational

Status: Acknowledged

File(s) affected: `GovernorBravoDelegator.sol`, `Hifi.sol`

Related Issue(s): SWC-123

Description: It is important to validate inputs, even if they only come from trusted addresses, to avoid human error. Specifically, in the following functions arguments of type `address` may be initialized with value `0x0`:

1. `GovernorBravoDelegator.constructor()`: Should check that `implementation_` is a contract.
2. `Hifi.setMinter()`: Should check that the minter is not the same.
3. `Hifi.mint()`: Should check that `rawAmount` is not zero.
4. `GovernorBravoDelegator._setImplementation()`: Should check that `implementation_` is a contract.

Recommendation: We recommend adding the relevant checks.

QSP-11 Privileged Roles and Ownership

Severity: Informational

Status: Acknowledged

File(s) affected: `Hifi.sol`, `GovernorBravoDelegator.sol`, `GovernorBravoDelegate.sol`, `TimeLock.sol`

Description: Certain contracts have state variables, e.g. `owner`, which provide certain addresses with privileged roles. Such roles may pose a risk to end-users.
The `Hifi.sol` contract contains the following privileged roles:

- `minter`, as initialized during the `constructor()` execution to `minter_`:
  . Assign the `minter` role to arbitrary address by calling `setMinter()`.
  . Renounce the role (**and thereby preventing any future calls to the followingly listed funcitons!**) by calling `setMinter()` with a known uncontrollable address (i.e. `address(0)`).
  . Mint new Hifi tokens to an arbitrary address by calling `mint()`.

The `GovernorBravoDelegate.sol` contract contains the following privileged roles:

- `admin`, as initialized during the `constructor()` execution of `GovernorBravoDelegator.sol` to `msg.sender`:
  . Initialize the contract once with critical protocol variables by calling `initialize()`.
  . Change the voting delay (within the bounds of `MIN_VOTING_DELAY` and `MAX_VOTING_DELAY`) by calling `_setVotingDelay()`.
  . Change the voting period (within the bounds of `MIN_VOTING_PERIOD` and `MAX_VOTING_PERIOD`) by calling `_setVotingPeriod()`.

-
  - Change the proposal threshold (within the bounds of `MIN_PROPOSAL_THRESHOLD` and `MAX_PROPOSAL_THRESHOLD`) by calling `_setProposalThreshold()`.
  - Propose a new address to be the `admin` role (which has to be accepted by calling `_acceptAdmin()`) by calling `_setPendingAdmin()`.

The `GovernorBravoDelegator.sol` contract contains the following privileged roles:

- `admin`, as initialized during the `constructor()` execution to `msg.sender`:
  - Modify/Update the `GovernorBravoDelegate.sol` implementation code by calling `_setImplementation()`.

The `Timelock.sol` contract contains the following privileged roles:

- `admin`, as initialized during the `constructor()` execution to `admin_`:
  - Queue a transaction by calling `queueTransaction()`.
  - Unqueue a transaction by calling `cancelTransaction()`.
  - Execute a queued transaction by calling `executeTransaction()`.
- Functions that can be called only through the timelock itself (`executeTransaction()`) by the `admin`:
  - Change the timelock delay (within the bounds of `MINIMUM_DELAY` and `MAXIMUM_DELAY`) by calling `setDelay()`.
  - Propose a new address to be the `admin` role (which has to be accepted by calling `acceptAdmin()`) by calling `setPendingAdmin()`.

Recommendation: Clarify the impact of these privileged actions to the end-users via publicly facing documentation.

QSP-12 Unnecessary Use of `add*()` and `sub*()` in Solidity `0.8.x`

Severity: Informational

Status: Fixed

File(s) affected: `Hifi.sol`, `GovernorBravoDelegate.sol`

Description: Solidity `0.8.x` has a built-in mechanism for dealing with overflows and underflows. There is no need to use the `add*()` and `sub*()` functions in `Hifi.sol` and `GovernorBravoDelegate.sol` (it only increases gas usage).

Recommendation: We recommend against using said functions in Solidity `0.8.x`.

Update: Fixed in commit 36e0a24 by replacing calls to `add()` and `sub()` functions with their corresponding primitive operations, as suggested.

QSP-13 `Timelock.queueTransaction()` and `Timelock.cancelTransaction()` Callable Multiple Times

Severity: Informational

Status: Acknowledged

File(s) affected: `Timelock.sol`

Description: Functions `Timelock.queueTransaction()` and `Timelock.cancelTransaction()` are callable multiple times, disregarding their current state. As these functions emit events, third-party systems listening for these events may behave in unexpected ways.

Recommendation: Consider adding checks such that `queueTransaction()` may only be called when `queuedTransactions[txHash] == false` and `cancelTransaction()` only when `queuedTransactions[txHash] == true`.

QSP-14 Unlocked Pragma

Severity: Informational

Status: Acknowledged

File(s) affected: `all`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.8.*`. 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, we recommend to remove the caret to lock the file onto a specific Solidity version.

QSP-15 A `Proposal` Can Be Canceled Several Times

Severity: Informational

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`

Description: A `Proposal` can be canceled via the function `cancel()` in two situations:

- by the proposer;
- by anyone if the proposer lost his right to propose (vote power now lower than the `proposalThreshold`); Once the function is called, the hash of all the transactions that are part of the `Proposal` will be removed from the mapping `Timelock.queuedTransactions`.

This issue alone has no direct impact. However, combined with the issue related to sandwich attacks (QSP-6), it could let an attacker disrupt the governance process to prevent another `Proposal` to get executed.

Recommendation: Consider adding a require statement that will prevent a `Proposal` that is `Canceled` from being cancelled again.

QSP-16 Time Calculated Assuming Constant Block Frequency

Severity: Informational

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`, `GovernorBravoInterfaces.sol`

Description: The range limits for the parameters of the governance logic (`MIN_VOTING_PERIOD`, `MAX_VOTING_PERIOD`, `MIN_VOTING_DELAY`, `MAX_VOTING_DELAY`) are assigned to constant values and measured in numbers of blocks. Inline comments state that they match a given number of days. However, there is no guarantee that the block frequency will remain constant. If the average block frequency is modified, it could impact the governance process. It will still be possible to modify the values of `votingDelay` and `votingPeriod`, but only within the range limits that cannot be updated.

Recommendation: Consider making it possible to modify the state variables `MIN_VOTING_PERIOD`, `MAX_VOTING_PERIOD`, `MIN_VOTING_DELAY`, `MAX_VOTING_DELAY`. However, this should be done taking into account the risks of storage collisions that can happen when using a Proxy pattern. Alternatively, consider at least improving the readability of these values with an inline comment detailing the unit of measure and how it is obtained from the underlying expected time duration.

QSP-17 Order of Evaluation of Expressions in Events Is Unexpected

Severity: Informational

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`

Description: In `GovernorBravoDelegate.castVoteBySig()`, `GovernorBravoDelegate.castVote()` and `GovernorBravoDelegate.castVoteWithReason()` the vote casting is done via a function being called from an event.
In Solidity inside events, the order of the evaluation of arguments is somewhat unexpected. This is not an issue as long as one doesn't call 2 functions that rely on each other's data from the same event. Hence this is a warning for future iterations of the protocol.

> In an event, the indexed parameters are first evaluated in right-to-left order, and then the non-indexed parameters are evaluated left-to-right. In the following example, the arguments are evaluated in the order `f(c) -> f(a) -> f(b) -> f(d)`

```
Event Hello(uint256 indexed a, uint256 b, uint256 indexed d, uint256 e);
```

```
emit Hello(f(a), f(b), f(c), f(d))
```

> https://medium.com/chainsecurity/beware-of-undefined-behavior-underhanded-solidity-contest-winner-22-42c6a52e2a8

Recommendation: Take this into account in future protocol development.

QSP-18 Block Timestamp Manipulation

Severity: Informational

Status: Acknowledged

File(s) affected: `Timelock.sol`

Description: Projects may rely on block timestamps for various purposes. However, it's important to realize that validators individually set the timestamp of a block, and attackers may be able to manipulate timestamps for their own purposes.

Recommendation: Consider the protocol design level and write tests showing timestamp manipulation does not pose a problem.

QSP-19 Arbitrary Transactions by Initial `Timelock.admin`

Severity: Undetermined

Status: Acknowledged

File(s) affected: `Timelock.sol`

Description: The initial `admin` of the contract `Timelock` is set during the deployment. He has the right to:

- queue arbitrary transactions with the function `queueTransaction()`;
- execute queued transactions with the function `executeTransaction()`, after a `MINIMUM_DELAY` of 2 days once the transaction has been queued;
- cancel queued transactions with the function `cancelTransaction()`;

The severity of that issue is [Undetermined]:

- In terms of impact, actions such as allowing an arbitrary address to transfer tokens from the contract `Timelock` can be considered.
- In terms of likelihood, it is unlikely that the `admin` will act maliciously if it is controlled by the team. Also, there is a minimum delay of 2 days before a queued transaction can be executed. Finally, these transactions would be logged and their effect could hopefully be reversed.

Recommendation: Consider keeping that issue in mind when deploying the contracts. Also, double-check the first transactions performed by the contract `Timelock` to detect any suspect transaction.

QSP-20 First Proposer Can Create Two Consecutive Proposals

Severity: Undetermined

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`

Description: The mapping `latestProposalIds` stores for each proposer the id of his last `Proposal`. In Solidity, the default value of mappings fields is 0.
Also, a mechanism in the function `propose()` prevents users from having more than one live `Proposal` per proposer (`Active` or `Pending`). It is only activated when the id of the last `Proposal` stored by `msg.sender` is not zero.
But, as the ids of proposals start at 0, the first proposer will be able to create the `Proposal #0` and also the `Proposal #1` because the mechanism described above will not be activated.
The severity is assessed as [Undetermined], because it was not possible to find a concrete impact.

Recommendation: A way to solve that issue is to modify the condition in the function `propose()` by: `if (latestProposalId != 0 || (proposalCount == 1 && proposals[0].proposer == msg.sender ))`

QSP-21 No Hifi Received when Small Amount Swapped

Severity: Undetermined

Status: Acknowledged

File(s) affected: `Hifi.sol`

Description: The function `swap()` uses an integer division to calculate how much HIFI will be given in exchange for MFT. As the `swapRatio` is 100:1, each user sending less than 100 wei of MFT will get 0 HIFI.

Recommendation: Consider this issue. If this is a problem, add a require statement to block swaps when the amount is strictly lower than 100 wei. Else, add a disclaimer when the swap feature will be officially announced.

QSP-22 Swaps Can Be Blocked if Contract MfT Is Paused

Severity: Undetermined

Status: Acknowledged

File(s) affected: `Hifi.sol`

Description: It is possible to unilaterally swap MFT with HIFI via the function `swap()`. However, the contract MFT can be paused by an address `owner` which has a non-zero value at the time of audit. If the contract is paused, transfers of MFT will fail. In such a situation, swaps would fail as well.

Recommendation: Consider keeping in mind that possibility once the swap feature is launched.

QSP-23 Potential Reentrancy

Severity: Undetermined

Status: Acknowledged

File(s) affected: `Timelock.sol`

Description: In `Timelock.executeTransaction()` a `target.call{ value: value }(callData)` is made that can trigger a reentrant transaction.

Recommendation: Add OpenZeppelin's ReentrancyGuard contract modifiers to the function.

QSP-24 `ecrecover()` Can Return a Random Address Instead of 0 for an Invalid Signature

Severity: Undetermined

Status: Acknowledged

File(s) affected: `GovernorBravoDelegate.sol`

Description: In `GovernorBravoDelegate.castVoteBySig()`, the signatory is worked out from the following function `ecrecover(digest, v, r, s)`.
Due to the nature of `ecrecover()`, if the signature is not valid, it does not always return 0 but can return a valid ethereum address. This could lead to an issue as the signatory is used as the person casting votes. Hence you can cast votes on someone else's behalf.

Exploit Scenario: 1. Work out a combination of v, r, s and digest that produce a valid ethereum address from `ecrecover()`.

1. Send HiFi tokens to that address.
2. Cast a vote on behalf of the address you do not control, potentially against the holder's wishes.

Recommendation: Add `signatory` as one of the parameters in the signature and pass it as a parameter into the function. Then verify that those signatories match before a vote is cast.

Code Documentation

1. The following typographical errors have been noted:
    1. `Hifi.sol#L144`: `src` -> `msg.sender`.
    2. `Hifi.sol#L204`: `overflows` -> `underflows`.
    3. `Hifi.sol#L212`: `spends` -> `spender`.
    4. `Hifi.sol#L289`: `approve` -> `transferFrom`.
    5. `Hifi.sol#L440` and `L447`: `_moveVotes` -> `_moveDelegates`.
    6. `Timelock.sol#L45`: `setDelay` -> `constructor`.

2. Comment in `Hifi.sol#L124` states `amount exceeds totalSupply`. However, it should rather be something like `new totalSupply exceeds 96 bits`.
3. Comment in `GovernorBravoDelegate.sol#L29` states `600,000 = 4% of Hifi`. However, this seems to be a copy-and-paste error from the forked Forth codebase and should rather be `600,000 = 0,06% of Hifi`, considering the increased supply of `1_000_000_000e18` Hifi Tokens, compared to the `15_000_000e18` Forth tokens.
4. Consider improving the readability of the code documentation by systematically using the NatSpec format (https://docs.soliditylang.org/en/develop/natspec-format.html). For example, the functions of the contract `Timelock` are not documented.
5. In `Hifi.swap()`, a transfer is made to `address(1)` instead of `address(0)`, which is not a standard behavior. Even if the reason is explained in `COMMITLOG.md`, consider explaining it directly in the function `swap()` to improve the readability of the code.

Adherence to Best Practices

1. To facilitate logging it is recommended to index address parameters within events. Therefore the `indexed` keyword should be added to the (other) address parameters in
   1. `GovernorBravoEvent.NewImplementation()`,
   2. `GovernorBravoEvent.NewPendingAdmin()`,
   3. `GovernorBravoEvent.NewAdmin()`,
   4. `Hifi.MinterChanged()`,
2. For readability and consistency, code should adhere to the commonly accepted code style guide. In this regard, consider the following instances, deviating from said guide:
   1. Constants in contracts `Hifi.sol` and `GovernorBravoDelegate.sol` are not adhering to be named in all capital letters with underscores separating words.
3. To improve readability and lower the risk of introducing errors when making code changes, it is advised to not use magic constants throughout code, but instead declare them once (as constant and commented) and use these constant variables instead. Following instances should therefore be changed accordingly:
   1. `Hifi.sol#L472`: `2**32` (Consider changing the check from `n < 2**32` to `n <= type(uint32).max`).
   2. `Hifi.sol#L477`: `2**96` (Consider changing the check from `n < 2**96` to `n <= type(uint96).max`).
   3. `GovernorBravoDelegate.castVoteInternal()`: `0`, `1`, `2` (Consider using more descriptive enums).
4. `Timelock.getBlockTimestamp()`: using directly the value of `block.timestamp` instead of the function `getBlockTimestamp()` would reduce the code complexity.
5. `Timelock`: it is recommended to place the state variables of a contract before the events. Consider following the order of layout recommended in the style guide of Solidity (https://docs.soliditylang.org/en/v0.8.17/style-guide.html#order-of-layout).
6. `Timelock.constructor()`, `Hifi.transferFrom()`, `Hifi._moveDelegates()`: Incorrect error messages in require statements. Consider fixing these error messages to avoid confusion during debugging.
7. `GovernorBravoDelegate`: reading the value of the array's length in the loops of the functions `propose()`, `queue()` and `execute()` could be done only once by caching it using a local uint256 variable. Doing so would optimize the runtime gas costs.
8. `Timelock.queueTransaction()`: the function shares its name with the event `QueueTransaction`, which could be ambiguous. For more readability, consider changing the names of the events `CancelTransaction`, `ExecuteTransaction`, `QueueTransaction` with `TransactionCancelled`, `TransactionExecuted`, `TransactionQueued`.
9. It is good practice to add the contract version number to the EIP-712 signature.
10. In both functions `permit()` and `delegateBySig()`, the signatory's nonce is used and incremented right after, using the operator `++`. But all happens in the same line. This subtlety of the language might no be caught by all readers. Consider separating the usage and the incrementation of the nonce to improve the readability of the code.

Test Results

Test Suite Results

Of the provided 83 tests all were successfully executed.

```
Unit tests
  GovernorBravo
    castVote
      We must revert if:
         ✓ There does not exist a proposal with matching proposal id where the current block number is between the proposal's start block (exclusive) and end block (inclusive) (50ms)
         ✓ Such proposal already has an entry in its voters set matching the sender (497ms)
      Otherwise
         ✓ we add the sender to the proposal's voters set (229ms)
        and we take the balance returned by GetPriorVotes for the given sender and the proposal's start block, which may be zero,
           ✓ and we add that ForVotes (831ms)
           ✓ or AgainstVotes corresponding to the caller's support flag. (604ms)
          castVoteBySig
             ✓ reverts if the signatory is invalid (410ms)
    propose
      simple initialization
         ✓ ID is set to a globally unique identifier
         ✓ Proposer is set to the sender
         ✓ Start block is set to the current block number plus vote delay
         ✓ End block is set to the current block number plus the sum of vote delay and vote period
         ✓ ForVotes and AgainstVotes are initialized to zero
         ✓ Executed and Canceled flags are initialized to false
         ✓ ETA is initialized to zero
         ✓ Targets, Values, Signatures, Calldatas are set according to parameters
         ✓ This function returns the id of the newly created proposal. # proposalId(n) = succ(proposalId(n-1)) (113ms)
         ✓ emits log with id and description (395ms)
        This function must revert if
           ✓ the length of the values, signatures or calldatas arrays are not the same length, (63ms)
           ✓ or if that length is zero or greater than Max Operations.
          Additionally, if there exists a pending or active proposal from the same proposer, we must revert.
             ✓ reverts with pending
             ✓ reverts with active
    queue
      overlapping actions
         ✓ reverts on queueing overlapping actions in same proposal (598ms)
    state
       ✓ Invalid for proposal not found
       ✓ Pending
       ✓ Active
       ✓ Canceled (586ms)
       ✓ Defeated
       ✓ Succeeded (513ms)
       ✓ Queued (739ms)
       ✓ Expired (770ms)
       ✓ Executed (989ms)
    _setVotingDelay
      msg.sender != admin
         ✓ reverts
      msg.sender == admin
        newVotingDelay >= MIN_VOTING_DELAY && newVotingDelay <= MAX_VOTING_DELAY
           ✓ succeeds (52ms)
        Otherwise
           ✓ reverts
      _setVotingPeriod
        msg.sender != admin
           ✓ reverts
        msg.sender == admin
           ✓ succeeds (53ms)
      _setProposalThreshold
        msg.sender != admin
           ✓ reverts
        msg.sender == admin
          newProposalThreshold >= MIN_PROPOSAL_THRESHOLD && newProposalThreshold <= MAX_PROPOSAL_THRESHOLD
             ✓ succeeds (54ms)
          Otherwise
             ✓ reverts
      _setPendingAdmin
        msg.sender != admin
           ✓ reverts
        msg.sender == admin
           ✓ succeeds (53ms)
      _acceptAdmin
        msg.sender != pendingAdmin
           ✓ reverts
        msg.sender == pendingAdmin
          msg.sender != address(0)
             ✓ succeeds (57ms)
          msg.sender == address(0)
             ✓ reverts (456ms)

  Unit tests
    Hifi
      metadata
         ✓ has given name
         ✓ has given symbol
      balanceOf
         ✓ grants to initial account
      burn
         ✓ burn 0
         ✓ burn non-zero
         ✓ burn > totalSupply
         ✓ burn > balance
      burnFrom
         ✓ burn 0
         ✓ burn non-zero
         ✓ burn > approval
         ✓ burn > totalSupply
         ✓ burn > balance
         ✓ Zero Address
      numCheckpoints
         ✓ returns the number of checkpoints for a delegate (102ms)
         ✓ does not add more than one checkpoint in a block (93ms)
      setMinter
        msg.sender == minter
           ✓ succeeds
        msg.sender != minter
           ✓ reverts
      mint
        dst == address(0)
           ✓ reverts
        dst != address(0)
          msg.sender == minter
             ✓ succeeds
          msg.sender != minter
             ✓ reverts
      delegate
         ✓ nested delegation (96ms)
      delegateBySig
         ✓ succeeds (44ms)
      permit
         ✓ succeeds (47ms)
      swap
        mftAmount == 0
           ✓ reverts
        mftAmount != 0
          mftAmount > user balance
             ✓ reverts (574ms)
          mftAmount <= user balance
            mftAmount > allowance
               ✓ reverts
            mftAmount <= allowance
               ✓ succeeds (196ms)
      getCurrentVotes
        nCheckpoints > 0
           ✓ succeeds
        nCheckpoints == 0
           ✓ succeeds
      getPriorVotes
         ✓ reverts if block number >= current block
         ✓ returns 0 if there are no checkpoints
         ✓ returns the latest block if >= last checkpoint block
         ✓ returns zero if < first checkpoint block (39ms)
         ✓ generally returns the voting balance at the appropriate checkpoint (202ms)

  Unit tests
    Timelock
      setDelay
        msg.sender != address(this)
           ✓ reverts
        msg.sender == address(this)
          delay_ >= MINIMUM_DELAY
            delay_ <= MAXIMUM_DELAY
               ✓ succeeds
            Otherwise
               ✓ reverts
          Otherwise
             ✓ reverts
      setPendingAdmin
        msg.sender != address(this)
           ✓ reverts
        msg.sender == address(this)
           ✓ succeeds


  83 passing (26s)
```

Code Coverage

While the statement coverage reaches 98,2%, the branch coverage is only 72,83%. We highly recommend adding further tests to increase both coverage and in particular the branch coverage to at least 90% or ideally higher.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	98.2	72.83	100	98.18	
GovernorBravoDelegate.sol	97.89	75	100	98.53	349,376
GovernorBravoDelegator.sol	100	50	100	93.33	103
GovernorBravoInterfaces.sol	100	100	100	100	
Hifi.sol	98.55	76.67	100	98.55	154,232
Timelock.sol	97.56	62.5	100	97.56	131
All files	**98.2**	**72.83**	**100**	**98.18**	

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

```
1d58074d13bd65b80074834853092d6d6135783dcefbf27030255127c53c8aa6 ./contracts/GovernorBravoInterfaces.sol
d74fe811f3c6b44e6283b42403b4df4140b2c3d1a1a3d37057fa9726587201c5 ./contracts/Hifi.sol
ec32766fc4a82fc031f523142b1b888dfda3e110aa4fd0de3f8dabcb19493e6a ./contracts/Timelock.sol
572ac660e9b57a4c23ed782097df14377003896747dd067d17db3aab5ee36c50 ./contracts/GovernorBravoDelegator.sol
32e5faedb09c11a848b5d837643fc80766684de0e450eea32d1f36dbf6787dc3 ./contracts/GovernorBravoDelegate.sol
```

Tests

```
1fa2207cd77ef6f378c6b3711371bda91bcfd8b140918a55aebce18f70ee055f ./test/types.ts
5bde305bce139fe3ecfd1e989b26594f2d3a48976fa9439a7a674b5ee9110cde ./test/governorBravo/GovernorBravo.ts
36197487132afc21c5877f5373975383e9d42a8dcbd3f6ab6d22ebadaed30697 ./test/governorBravo/GovernorBravo.fixture.ts
0de8a6b984bf51c101ecba6d78ca9237af4f6d080b0163ec2cb79b94278ef865 ./test/governorBravo/GovernorBravo.behavior.ts
45e332a143fcde6dba8ad6775adda0589c1b08cbd11a72cf6680e1b50d146002 ./test/timelock/Timelock.ts
352b1d11834637601b27908b1df965d741799864759392481d5efdd05c9dc546 ./test/timelock/Timelock.fixture.ts
af12bb200c48cc7556106462725af802548ab393266939c8975423cead6937a8 ./test/timelock/Timelock.behavior.ts
4043bdbd32e260ca398f2cad00c2fea26bdcb9b8e3cc65f09a05a168d55c160f ./test/hifi/Hifi.fixture.ts
479a933e14109236bb2274aa177cda095dc9249937addc470312068d9b35c3ad ./test/hifi/Hifi.ts
f13290c96a739601a49d098341b690bd72667ac5e36cbcf64c8e7ff4f72f5af7 ./test/hifi/Hifi.behavior.ts
```

Changelog

- 2022-11-03 - Initial report

About Quantstamp

Quantstamp is a global leader in blockchain security backed by Pantera, Softbank, and Commonwealth among other preeminent investors. Founded in 2017, Quantstamp's mission is to securely onboard the next billion users to Web3 through its white glove security and risk assessment services.

The team consists of web3 thought leaders hailing from top organizations including Microsoft, AWS, BMW, Meta, and the Ethereum Foundation. Many of the auditors hold PhDs or advanced computer science degrees, with decades of combined experience in formal verification, static analysis, blockchain audits, penetration testing, and original leading-edge research.

To date, Quantstamp has performed more than 250 audits and secured over $200 billion in digital asset risk from hackers. In addition to providing an array of security services, Quantstamp facilitates the adoption of blockchain technology through strategic investments within the ecosystem and acting as a trusted advisor to help projects scale.

Quantstamp's collaborations and partnerships showcase our commitment to world-class research, development and security. We're honored to work with some of the top names in the industry and proud to secure the future of web3.

Notable Collaborations & Customers:

- Blockchains: Ethereum 2.0, Near, Flow, Avalanche, Solana, Cardano, Binance Smart Chain, Hedera Hashgraph, Tezos
- DeFi: Curve, Compound, Aave, Maker, Lido, Polygon, Arbitrum, SushiSwap
- NFT: OpenSea, Parallel, Dapper Labs, Decentraland, Sandbox, Axie Infinity, Illuvium, NBA Top Shot, Zora
- Academic institutions: National University of Singapore, MIT