Quantstamp Security Assessment Certificate

July 2nd 2020 — Quantstamp Verified

Arcadeum

This smart contract audit was prepared by Quantstamp, the protocol for securing smart contracts.

Executive Summary

Type	Smart Contract Wallet
Auditors	Martin Derka, Senior Research Engineer Sung-Shine Lee, Research Engineer Alex Murashkin, Senior Software Engineer
Timeline	2020-05-26 through 2020-06-05
EVM	Muir Glacier
Languages	Solidity, Javascript
Methods	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification	None
Documentation Quality	Medium
Test Quality	High

Source Code

Repository	Commit
wallet-contracts	cd5ee37

Changelog

- 2020-06-05 - Initial report
- 2020-06-15 - Update based on `ad4e3d191`. Only changes related to QSP-1 -- QSP-19 were reviewed in the `cd5ee37...ad4e3d191` diff.
- 2020-06-19 - Acknowledgement of QSP-5, QSP-6 and QSP-8 based on mutual communication.
- 2020-06-26 - Acknowledgement of QSP-9, QSP-10, QSP-11, QSP-15 and QSP-17 based on changes in `a64bc78` and `bb52a66`.
- 2020-06-30 - Update for PR87 and PR88.

Total Issues	**20** (4 Resolved)
High Risk Issues	**5** (1 Resolved)
Medium Risk Issues	**4** (0 Resolved)
Low Risk Issues	**3** (0 Resolved)
Informational Risk Issues	**7** (3 Resolved)
Undetermined Risk Issues	**1** (0 Resolved)

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

The code of Arcadeum is elegant and clear. The clarity is mildly impacted by the use of multiple inheritance and overrides as outlined in this report. With some minor exceptions, it respects the best practices and appears to be properly tested. Quantstamp discovered vulnerabilities and potentially exploitable features with severities ranging from informational to high. We highly recommend addressing our findings before deploying the platform for the public use. We also recommend developing a comprehensive documentation that explains the properties of the platform at the level comprehensible to both technical and non-technical user audience. Also, the users should be aware that the Arcadeum wallet does not provide any recovery mechanism for the case when user credentials become lost or otherwise unavailable.
Update: Within the fixing period, the Arcadeum team fixed some and acknowledged the rest of the findings outlined in this report. The only outstanding finding is QSP-19 waiting for documentation to be published by Arcadeum. The best practices recommendations were not addressed. As an additional work, the Arcadeum team requested the audit of PR87 and PR88 (currently not merged) that resulted in listing the QSP-20 finding. In addition to white-glove review, all automated tools including tests and test coverage was re-run for these two PRs, and Quantstamp confirmed that the tools do not report any new findings, and the coverage remains as originally reported. Any work outside the original audit scope (set by commit `cd5ee37`), PR87, and PR88 is excluded from the audit. This concerns especially the `GuestModule.sol` and `ExpirableUtil.sol`.

ID	Description	Severity	Status
QSP-1	Self Destruction of Main Upgradeable Module	High	Fixed
QSP-2	Dangerous Self-Authorization and Delegate Call	High	Acknowledged
QSP-3	Minimum Threshold Validation	High	Acknowledged
QSP-4	Transaction Racing via Nonce Space Separation	High	Acknowledged
QSP-5	Reentrancy Enabled by Nonces	High	Acknowledged
QSP-6	No Nonce and Signature Expiration	Medium	Acknowledged
QSP-7	Self-Calls via Hooks	Medium	Acknowledged
QSP-8	Unrestricted Proxy Deployment	Medium	Acknowledged
QSP-9	Sensitivity of Upgrades	Medium	Acknowledged
QSP-10	Owner Signature Gas Limit Problems	Low	Acknowledged
QSP-11	Shadowed Hooks	Low	Acknowledged
QSP-12	Non-Exhaustive Supported Interface List	Low	Acknowledged
QSP-13	Unlocked Pragma	Informational	Fixed
QSP-14	Unused Experimental Encoder	Informational	Fixed
QSP-15	Limited `isContract()` Check	Informational	Acknowledged
QSP-16	Reverting on Errors	Informational	Acknowledged
QSP-17	Multiple Inheritance	Informational	Acknowledged
QSP-18	Missing Hook Target Enforcement	Informational	Fixed
QSP-19	Case Sensitivity of Addresses	Undetermined	Unresolved
QSP-20	Gas Limit Relayer Exposure	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:

- Truffle
- Ganache
- SolidityCoverage
- Mythril
- Truffle-Flattener
- Slither

Steps taken to run the tools:

1. Installed Truffle: `npm install -g truffle`
2. Installed Ganache: `npm install -g ganache-cli`
3. Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
4. Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
5. Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
6. Installed the Mythril tool from Pypi: `pip3 install mythril`
7. Ran the Mythril tool on each contract: `myth -x path/to/contract`
8. Installed the Slither tool: `pip install slither-analyzer`
9. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Self Destruction of Main Upgradeable Module

Severity: *High Risk*

Status: Fixed

File(s) affected: `MainModuleUpgradable.sol`, `ModuleCalls.sol`

Description: The main module implementations pointed to by the proxy wallets can be interacted with directly. When `_signatureValidation()` is being executed, and `_signature.length` is 0, and `threshold` is zero, the resulting `imageHash` is also 0 due to `bytes32 imageHash = bytes32(uint256(threshold));`. This matches the uninitialized value of `imageHash` in `ModuleAuthUpgradable`. Thus, a user can submit a transaction directly to the module. The transaction can delegate a call to an external smart contract that can contain the selfdestruct instruction. This would lead to a selfdestruction of the module, rendering all wallets pointed to this implementation immediately inoperable.

Recommendation: Quantstamp recommends devising a mechanism that will prevent direct interaction with `ModuleCalls.sol` and all modules that inherit from it. Requiring that `threshold > 0` issues and treating 0 image hash as invalid within `isValidImage()` could resolve the issue as well.
Update: Fixed in https://github.com/arcadeum/wallet-contracts/pull/82

QSP-2 Dangerous Self-Authorization and Delegate Call

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `MainModule.sol`, `ModuleCalls.sol`, `ModuleAuthUpgradable.sol`, `ModuleHooks.sol`, `ModuleSelfAuth.sol`, `ModuleUpdate.sol`, `MainModuleUpgradeable.sol`

Description: The wallet's modules provide method `selfExecute()` that does require owner signature to execute calls. It also authorizes several potentially sensitive methods (`updateImageHash()`, `addHook()`, `removeHook()`, `updateImplementation()`) via the `onlySelf` modifier. This can be exploited via `ModuleCalls.sol` and the modules that inherit from it (`MainModule.sol` and `MainModuleUpgradable.sol`) as they allow delegating call.
In a potential exploit, the rightful owners of wallet Alice submit a properly authorized transaction to `execute` opting for delegate call with the destination being a malicious smart contract Mallory. Mallory in turn makes a static call to Alice. Mallory can call any function authorized via `onlySelf`. As a result, Mallory could:

- Make arbitrary calls via `selfExecute`
- Upgrade implementations
- Add hooks
- Remove hooks
- Change image hash

The existence of the delegate call option is dangerous on its own. In an alternate scenario, Mallory can manipulate the storage of the wallet itself. This is further simplified as the wallet stores information, such as its implementation or nonces, via hashed keys. Mallory can also self-destruct the wallet, or withdraw its Ether.

Recommendation: Due to the option of delegating calls, it is imperative that the wallet owners are always aware of the implementation of smart contacts that they are making delegate calls to. It is hard to imagine that all the users of Arcadeum's wallets will have sufficient technical abilities and knowledge, and even in such a scenario, the implementation of the smart contract is not always public beyond the level of byte code. Therefore, Quantstamp recommends removing the option of delegated calls from `ModuleCalls.sol`, and devising additional authorization mechanism for the `onlySelf` modifier.
Update: This finding was acknowledged by the Arcadeum team with the following response:
This is expected behaviour, wallets and signers should only sign `delegateCall: true` transactions if they 100% trust the destination of such call, this allows wallets to have extended functionality on the fly, without having to re-configure the wallet.
Further discussion can be found in https://github.com/arcadeum/wallet-contracts/issues/63.

QSP-3 Minimum Threshold Validation

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `ModuleAuth.sol`, `Factory.sol`

Description: The threshold is generated and encoded off-chain. When the wallet is being constructed, the factory does not perform any input validation, and allows threshold signature 0. If such a wallet is created, any Ethereum account will be able to transact with it.

Recommendation: Quantstamp recommends adding validation to the factory method, or require that `threshold > 0` inside `_signatureValidation()`.
Update: This finding was acknowledged by the Arcadeum team with the following response:
There is no way of adding such validation during the wallet creation, because the factory contract only knows the hash of the configuration and not the configuration itself. Adding such validations would require computing the hash on-chain during creation, increasing the cost of deployment for each wallet.
About `threshold > 0` require during signature validation, the wallet contract should behave as the defined configuration defines, so if a threshold of 0 is specified no signatures are required.
Further discussion can be found in https://github.com/arcadeum/wallet-contracts/issues/64.

QSP-4 Transaction Racing via Nonce Space Separation

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `ModuleCalls.sol`

Description: Transactions in Ethereum can be raced. Before a transaction gets mined, it appears in the blockchain's mempool for some period of time. The general public can read the transaction's meaning and race it with their own transactions for personal profit. Due to the nonce space separation, the Arcadeum wallet additionally allows racing the wallet's own transactions. This can be problematic especially in the multisig scenario. If Alice and Mallory jointly own a wallet and A submits a transaction to Arcadeum wallet X with nonce `alice.1` (nonce 1 inside space `alice`), Mallory can see transaction and race with her own transaction submitted to the very same wallet X with nonce `mallory.1`. Since the nonces do not collide, both the transactions can be mined. Mallory's transaction can be mined first and change the state of the blockchain to harm Alice when her transaction is mined.

Recommendation: Quantstamp recommends supporting only a single space of nonces.
Update: This finding was acknowledged by the Arcadeum team with the following response:
This is expected behaviour, signing transactions with different nonce spaces is a signal that those transactions are not sequence dependent, so they should be able to be executed on any order.
Further discussion can be found in https://github.com/arcadeum/wallet-contracts/issues/65.

QSP-5 Reentrancy Enabled by Nonces

Severity: *High Risk*

Status: Acknowledged

File(s) affected: `ModuleCalls.sol`

Description: The nonces in Arcadeum are separated into spaces. The method `execute` can be re-entered with a different nonce space. Assume that Alice and Mallory jointly own a wallet X and they both have sufficient weight to reach the signature threshold. Alice submits a transaction with nonce `alice.3` that does not get mined. This can be due to an external call failure and the `failOnRevert` flag being `true`, or due to the transaction getting stuck in the mempool due to low gas, or due to a blockchain reorganization that unrolls a block. Mallory can submit a transaction to X with nonce `mallory.1` that modifies the blockchain state to benefit Mallory and damage Alice when Alice's transaction gets mined, and re-enter `execute()` with Alice's transaction (its signature is still valid).
In a related scenario, Alice can sign a transaction with nonce being out of order (`alice.2` is expected, but Alice signs `alice.3`). Mallory now receives a wild card to submit a transaction with nonce `alice.2` and re-enter `execute()` with Alice's transaction (its signature is still valid).

Recommendation: Quantstamp recommends equipping the `execute()` (and potentially `selfExecute()`) methods with a reentrancy lock.
Update: The Arcadeum team acknowledged the finding in communication with Quantstamp. Some additional discussion about this issue is available in https://github.com/arcadeum/wallet-contracts/issues/66

QSP-6 No Nonce and Signature Expiration

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `MainModule.sol`, `MainModuleUpgradeable.sol`, `ModuleCalls.sol`

Description: When transactions are submitted to the wallets for execution, the signatures are revealed on-chain. If a transaction fails due to `revertOnError`, the nonce does not get incremented, and the signature remains and is publicly known. The failure can be a result of a call to an external contract, or a low signature threshold. In either case, the signatures can be replayed for the same transaction later, without the knowledge and consent of the account owner.

Recommendation: Quantstamp recommends devising an expiration mechanism for transactions.
Update: The Arcadeum team acknowledged the finding in communication with Quantstamp. Some additional discussion about this issue is available in https://github.com/arcadeum/wallet-contracts/issues/67

QSP-7 Self-Calls via Hooks

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `ModuleHooks.sol`

Description: Method `addHook()` does not validate method signatures against spaces of valid method selectors. A malicious wallet admin may use a hook to invoke calls to any method with an `onlySelf()` modifier.

Recommendation: Quantstamp recommends devising a validation mechanism for hooks.
Update: This finding was acknowledged by the Arcadeum team with the following response:
Hooks are invoked using delegateCalls so a "malicious admin" does not need to use any of the onlySelf methods to compromise the wallet, they can directly modify the storage.
Further discussion can be found in https://github.com/arcadeum/wallet-contracts/issues/68.

QSP-8 Unrestricted Proxy Deployment

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `Factory.sol`

Description: The factory method will allow the deployment of any proxy. The proxy does not need to be pointed to a wallet at all, or it can have a malicious implementation. This is amplified in the context of `create2` as the factory method can deploy self-destructible proxies and allow for metamorphic contracts.

Recommendation: Quantstamp recommends devising a mechanism for whitelisting deployable proxies.
Update: The Arcadeum team acknowledged the finding in communication with Quantstamp. Some additional discussion can be found in https://github.com/arcadeum/wallet-contracts/issues/69.

QSP-9 Sensitivity of Upgrades

Severity: *Medium Risk*

Status: Acknowledged

File(s) affected: `MainModule.sol`, `MainModuleUpgradable.sol`

Description: Wallets can be upgraded at users' discretion. The implementation of a wallet to upgrade to can be malicious. It can provide users with any capabilities, such as allowing calls bypassing authorization mechanisms, ownership changes, denying service, etc. This can be problematic especially in multisig scenarios when not all owners consent to the upgrade.

Recommendation: Quantstamp recommends informing users of potentially undesirable upgrade consequences, and considering the consensus of all wallet's owners for an upgrade to be feasible.
Update: The Arcadeum team acknowledged this finding and indicated that the users are assume to be aware of all the consequences. Some further discussion can be found on https://github.com/arcadeum/wallet-contracts/issues/70.

QSP-10 Owner Signature Gas Limit Problems

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ModuleCalls.sol`

Description: When transactions are submitted to the wallet for execution, an additional byte string parameter that encodes signatures/addresses of wallet owners is required. If the threshold requires many signatures, a transaction may not be executable without exhausting the gas limit. This can be problematic in the following situations:

- The threshold is large and requires many signatures in all cases.
- The threshold is large (e.g., 100000), there is an account whose signature carries extreme weight (e.g., 99999), and there are many additional accounts whose signatures carry small weights (e.g., 1). The accounts with small weights may be unable to transact without the extreme-weight account's consent as their signatures will not be packable into a transaction. This may provide a misleading impression to users.
- Several nested calls are needed. If wallet A calls execute in wallet B, which in turn calls execute in wallet C, which in turn calls wallet D, etc., the effects of the signature lengths accumulate through the call stack.

Recommendation: Quantstamp recommends calculating exact bounds for the number of signatures allowed in a call, and informing the users of the limits.
Update:: The Arcadeum team intends to document this behaviour. The documentation is pending. Further details can be found on https://github.com/arcadeum/wallet-contracts/issues/71.

QSP-11 Shadowed Hooks

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ModuleHooks.sol`

Description: It is possible to add hooks that are already defined as functions in the `ModuleHooks.sol`, but those hooks will not be invoked as the fallback function will not be hit. This may lead to confusing results -- the new hooks will not be functioning at all.

Recommendation: Quantstamp recommends verifying that this is the intended behaviour. If this is the case, the Arcadeum team can restrict users from adding hooks that shadow the existing ones, and document this.
Update:: The Arcadeum team intends to document this behaviour. The documentation is pending. Further details can be found on https://github.com/arcadeum/wallet-contracts/issues/72.

QSP-12 Non-Exhaustive Supported Interface List

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `ModuleHooks.sol`

Description: The `supportInterface()` methods may not accurately reflect supported interfaces. When adding a hook other than `IERC1155Receiver`, `IERC721Receiver`, or `IERC223Receiver`, the contract will end up having the hook but will not show its support queried through `supportsInterface`.

Recommendation: Quantstamp recommends either restricting hooks that can be added to only the predefined ones, or make it possible to reflect hooks in the `supportInterface()` method.
Update: The Arcadeum team acknowledged this finding. Some further discussion can be found on https://github.com/arcadeum/wallet-contracts/issues/73.

QSP-13 Unlocked Pragma

Severity: *Informational*

Status: Fixed

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." 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.

Recommendation: Quantstamp recommends locking the pragma for all contracts with the exception of libraries that are intended to be reusable.
Update: The issue was fixed in https://github.com/arcadeum/wallet-contracts/pull/83.

QSP-14 Unused Experimental Encoder

Severity: *Informational*

Status: Fixed

File(s) affected: `MainModule.sol`, `ModuleAuth.sol`, `MainModuleUpgradable.sol`, `Implementation.sol`, `ModuleCalls.sol`, `ModuleERC165.sol`, `ModuleHooks.sol`, `ModuleSelfAuth.sol`, `ModuleStorage.sol`, `ModuleUpdate.sol`, `IModuleCalls.sol`, `SignatureValidator.sol`

Description: The majority of contracts include ABIEncoderV2 that is marked as experimental and is not recommended for production use. In some of these contracts, the encoder is not used. Quantstamp recommends removing ABIEncoderV2 from such contracts.

Recommendation: Quantstamp recommends removing ABIEncoderV2 from contracts that do not require its presence.
Update: The issue is partly false-positive as the encode is no longer considered experimental in Solidity 0.6.8. Its unused imports were removed in https://github.com/arcadeum/wallet-contracts/pull/84.

QSP-15 Limited `isContract()` Check

Severity: *Informational*

Status: Acknowledged

File(s) affected: `LibAddress.sol`, `ModuleUpdate.sol`

Description: The `LibAddress.sol` library implements `isContract(address account)` function that returns `true` if there is a code of non-zero size associated with the provided parameter `account`. In certain cases, such as when invoked from a smart contract constructor, this function will return `false` which may be misleading. The Arcadeum wallets platform uses the function to validate the implementation upgrade and make the upgrade fail if `isContract()` returns `false` for the new implementation. This means that users will not be able to deploy an implementation and perform a wallet upgrade from the implementation's constructor. Upgrades will need to happen independently.

Recommendation: Quantstamp recommends publishing instructions for wallet implementation upgrades mentioning that this way of upgrading will fail.
Update: The Arcadeum team acknowledged this finding and intends to document it. No documentation was provided yet. Some further discussion about this issue can be found on https://github.com/arcadeum/wallet-contracts/issues/76.

QSP-16 Reverting on Errors

Severity: *Informational*

Status: Acknowledged

File(s) affected: `ModuleCalls.sol`

Description: By default, calls to `execute` and `selfExecute` do not revert on errors. The users need to actively set a flag for the revert to happen. This is not the default behaviour of the Ethereum platform which may be counter-intuitive.

Exploit Scenario: Quantstamp recommends replacing the field `revertOnError` with `doNotRevertOnError`.
Update: The Arcadeum team acknowledged this finding with the following response:
The problem with a transaction reverting on error is that the nonce does not get incremented, which is also unlike the default behavior of Ethereum transactions. Either way, there is a break from how regular Ethereum transaction works and we decided to use `revertOnError` to avoid double negation and make this functionality easier to understand. Clients and wallets can set this behavior to true by default by setting this boolean to true.
The response can be found on https://github.com/arcadeum/wallet-contracts/issues/77.

QSP-17 Multiple Inheritance

Severity: *Informational*

Status: Acknowledged

File(s) affected: `All modules`

Description: Contracts will sometimes inherit functionality from multiple parents. However, this can lead to an issue when these parents have conflicting functionality. When this happens, the end result of inheritance can lead to unexpected behavior.
The modules contain the `supportsInterface(bytes4 _interfaceID)`methods. These methods invoked via a sequence of calls to super initiating in the main module. The sequence will end as soon as `ERC165Module.supportsInterface(bytes4 _interfaceID)` is invoked. An improper inheritance order may result in some interface checks being skipped and may represent a problem when new modules are added. The Arcadeum team informed Quantstamp that future module additions are possible, and due to the upgrade mechanism, users can also implement their own modules.

Recommendation: Quantstamp verified that the implementation currently behaves as intended. However, to follow the best practices and avoid confusion, Quantstamp recommends reimplementing the `supportsInterface(bytes4 _interfaceID)` method in all the module so that it avoids calls to `super.supportsInterface()` when overriding these methods, and listing all interfaces explicitly. For future development, Quantstamp recommends paying close attention to the multiple inheritance, as well as always including a test for supported interfaces (such a test is currently present). Users who want to develop their own modules using the common modules provided by Arcadeum should be informed in the documentation of the nuances of such the multiple inheritance.
Update: The Arcadeum team acknowledged this finding and intends to document it. No documentation was provided yet. Some further discussion about this issue can be found on https://github.com/arcadeum/wallet-contracts/issues/78.

QSP-18 Missing Hook Target Enforcement

Severity: *Informational*

Status: Fixed

File(s) affected: `ModuleHooks.sol`, `MainModule.sol`, `MainModuleUpgradeable.sol`

Description: `ModuleHooks.sol` does not enforce the presence of `target` for each implemented interface. For example, the contract advertises that `onERC1155Received` is supported, however, in practice, there may not be a `target` set for this method. Also, `fallback() external payable {...}` does not fail in this case, which can be misleading for the caller.

Recommendation: Quantstamp recommends enforcing targets before advertising that interfaces are supported.
Update: After mutual discussion, this issue was marked as false positive. No action needs to be taken. The response of the Arcadeum team is available on https://github.com/arcadeum/wallet-contracts/issues/79.

QSP-19 Case Sensitivity of Addresses

Severity: *Undetermined*

Status: Unresolved

File(s) affected: `ModuleAuth.sol`, `Factory.sol`

Description: The image hash is sensitive to whether or not the addresses are in the check-sum format or not (or any other capitalization).

Recommendation: Quantstamp recommends noting in the documentation that this is the case, and paying attention to this in the off-chain code. An Ethereum address has 2^42 possible capitalizations, so an utterly random capitalization used to generate the image hash may result in a DoS on a created wallet.
Update: The issue is currently under review. The response of the Arcadeum team is available on https://github.com/arcadeum/wallet-contracts/issues/80.

QSP-20 Gas Limit Relayer Exposure

Severity: *Informational*

Status: Acknowledged

File(s) affected: `ModuleCalls.sol`,

Description: According to Issue 86, it appears that the Arcadeum team originally intended to protect the relayer from malicious users by specifying `gasLimit`. PR88 seems to remove the protection, allowing users to specify `gasLimit` as 0 and essentially asks the relayer to dump all the remaining gas `gasLeft()` into the delegate call. The requirement for `gasleft() > gasLimit` in PR87 can make transaction always fail if `gasLimit` exceeds the gas limit of a block, potentially causing harm to the relayer that spent some gas for relaying the transaction that executed these preparatory instructions and checks.

Recommendation: Quantstamp recommends documenting this risk for relayers actively recommending that relayers do not relay transactions before performing a gas analysis and comparing it to the set limit.
Update: The Arcadeum team acknowledged the finding and documented it in PR88.

Automated Analyses

Mythril

Mythril did not finish execution.

Slither

Slither reported problems with lack of Ether draining features, and the presence of assembly in the code. All relevant findings were included in this report.

Adherence to Specification

No specification was provided for the audit, aside from an initial code walkthrough. The code adheres to intentions expressed by the developer, and to the intentions of the method-level documentation.

Code Documentation

The code contains a good level of documentation. An external specification should be added though.

Adherence to Best Practices

With the exception of the following, the code adheres to best practices:

1. Multiple inheritance as outlined in the findings
2. The use of experimental encoder as outlined in the findings
3. Unlocked pragmas as outlined in the findings
4. modules/commons/ModuleCalls.sol#38: "of th given nonce space" -> "of the given nonce space"
5. modules/commons/ModuleCalls.sol#160: `emit TxFailed(_txHash, _reason);` does not have any context of the transaction that got reverted. Some hint to the user, perhaps an index for the transaction array, would be advisable.
6. modules/commons/ModuleCalls.sol#160: `emit TxFailed(_txHash, _reason);` does not convert `_reason` to string. It may be useful for readability purposes.
7. modules/commons/ModuleHooks.sol#61 inputs to `_writeHook(...)` should be validated: `_implementation != 0x0`
8. modules/commons/ModuleAuthUpgradable.sol# 37: `_isValidImage(_imageHash)` should check if `_imageHash != 0x0`
9. modules/commons/ModuleAuth.sol#125 and #144: `isValidSignature(...)` methods should explicitly return `0x0` if signature validation failed.
10. modules/common/Implementation.sol#14: "WARNING updating this value may brick the wallet" -> "break the wallet"
11. interfaces/IERC1271Wallet.sol#4: "interface IERC1271Wallet {" - extra space after "interface"
12. utils/LibBytes.sol#42: would it be better to use array.pop(...) instead? https://solidity.readthedocs.io/en/v0.6.0/types.html The current approach basically changes .length value that was made read-only in solidity V6. So, basically, this is a “hacky” way of doing .pop(...)
13. utils/LibBytes.sol: the require() statements should probably be placed at the top of each method: L72, L97, L144. On L97, `newIndex := add(index, 2)` - could be moved to L91, and the require(...) could be moved up also. Same for L119 and L144.
14. utils/LibBytes.sol: should check that `index` is within `data.length`: L85, L109, L131. Theoretically, if somebody provides an `index` that is too high (close to MAX_INT), the code would work fine: because of the overflow, require(newIndex <= data.length) would be fulfilled.

15. modules/commons/ModuleAuth.sol#29: weigth => weight

16. modules/commons/ModuleAuth.sol#40: the comment here doesn't seem to match the implementation, the implementation doesn't have the `uint8 nSigners`, it only has the `uint16 threshold`, as can be seen in L53-56.

Test Results

Test Suite Results

All tests pass. The test suite appears comprehensive.

```
Contract: ERC165
  Implement all interfaces for ERC165 on MainModule
     ✓ Should return implements IModuleHooks interfaceId (175ms)
     ✓ Should return implements IERC223Receiver interfaceId (98ms)
     ✓ Should return implements IERC721Receiver interfaceId (59ms)
     ✓ Should return implements IERC1155Receiver interfaceId (76ms)
     ✓ Should return implements IERC1271Wallet interfaceId (50ms)
     ✓ Should return implements IModuleCalls interfaceId (54ms)
     ✓ Should return implements IModuleCreator interfaceId (56ms)
     ✓ Should return implements IModuleHooks interfaceId (74ms)
     ✓ Should return implements IModuleUpdate interfaceId (64ms)
  Implement all interfaces for ERC165 on MainModuleUpgradable
     ✓ Should return implements IModuleHooks interfaceId (41ms)
     ✓ Should return implements IERC223Receiver interfaceId
     ✓ Should return implements IERC721Receiver interfaceId (71ms)
     ✓ Should return implements IERC1155Receiver interfaceId
     ✓ Should return implements IERC1271Wallet interfaceId
     ✓ Should return implements IModuleCalls interfaceId
     ✓ Should return implements IModuleCreator interfaceId
     ✓ Should return implements IModuleHooks interfaceId
     ✓ Should return implements IModuleUpdate interfaceId
     ✓ Should return implements IModuleAuthUpgradable interfaceId
  Manually defined interfaces
     ✓ Should implement ERC165 interface
     ✓ Should implement ERC721 interface
     ✓ Should implement ERC1155 interface

Contract: Factory
  Deploy wallets
     ✓ Should deploy wallet
     ✓ Should predict wallet address
     ✓ Should initialize with main module

Contract: LibBytes
  popLastByte
     ✓ Should pop last byte
     ✓ Should pop single byte
     ✓ Should fail to pop empty array (55ms)
  readFirstUint16
     ✓ Should read first uint16
     ✓ Should read first uint16 of 2 byte array
     ✓ Should fail first uint16 out of bounds
  readUint8Uint8
     ✓ Should read bool and uint8 at index zero
     ✓ Should read bool and uint8 at given index
     ✓ Should read bool and uint8 at last index
     ✓ Should fail read bool and uint8 out of bounds
     ✓ Should fail read bool and uint16 fully out of bounds
  readAddress
     ✓ Should read address at index zero
     ✓ Should read address at given index
     ✓ Should read address at last index (85ms)
     ✓ Should fail read address out of bounds
     ✓ Should fail read address totally out of bounds
  readBytes66
     ✓ Should read bytes66 at index zero
     ✓ Should read bytes66 at given index (51ms)
     ✓ Should read bytes66 at last index
     ✓ Should fail read bytes66 out of bounds
     ✓ Should fail read bytes66 totally out of bounds
  readBytes32
     ✓ Should read bytes32 at index zero
     ✓ Should read bytes32 at given index
     ✓ Should read bytes32 at last index
     ✓ Should fail read bytes32 out of bounds
     ✓ Should fail read bytes32 totally out of bounds

Contract: MainModule
  Authentication
     ✓ Should accept initial owner signature (103ms)
     ✓ Should reject non-owner signature (58ms)
     ✓ Should reject signature with invalid flag
    Network ID
       ✓ Should reject a transaction of another network id (54ms)
    Nonce
      Using non-encoded nonce
         ✓ Should default to space zero (40ms)
         ✓ Should work with zero as initial nonce
         ✓ Should emit NonceChange event (85ms)
         ✓ Should fail if nonce did not change (106ms)
         ✓ Should fail if nonce increased by two (78ms)
      using 0x00 space
         ✓ Should work with zero as initial nonce (39ms)
         ✓ Should emit NonceChange event (84ms)
```

```
      ✓ Should accept next nonce (97ms)
      ✓ Should fail if nonce did not change (81ms)
      ✓ Should fail if nonce increased by two (121ms)
      ✓ Should use nonces storage keys (39ms)
    using 0x01 space
      ✓ Should work with zero as initial nonce (40ms)
      ✓ Should emit NonceChange event (51ms)
      ✓ Should accept next nonce (69ms)
      ✓ Should fail if nonce did not change (79ms)
      ✓ Should fail if nonce increased by two (65ms)
      ✓ Should use nonces storage keys (41ms)
    using 0x1cae space
      ✓ Should work with zero as initial nonce
      ✓ Should emit NonceChange event (65ms)
      ✓ Should accept next nonce (61ms)
      ✓ Should fail if nonce did not change (65ms)
      ✓ Should fail if nonce increased by two (58ms)
      ✓ Should use nonces storage keys
    using 0x522a71ef48daadfa2237253670a3c91f4fc6ec6e space
      ✓ Should work with zero as initial nonce (45ms)
      ✓ Should emit NonceChange event (56ms)
      ✓ Should accept next nonce (83ms)
      ✓ Should fail if nonce did not change (55ms)
      ✓ Should fail if nonce increased by two (61ms)
      ✓ Should use nonces storage keys (53ms)
    using 0xffffffffffffffffffffffffffffffffffffffff space
      ✓ Should work with zero as initial nonce (39ms)
      ✓ Should emit NonceChange event (68ms)
      ✓ Should accept next nonce (72ms)
      ✓ Should fail if nonce did not change (57ms)
      ✓ Should fail if nonce increased by two (56ms)
      ✓ Should use nonces storage keys
    using two spaces simultaneously
      ✓ Should keep separated nonce counts (141ms)
      ✓ Should emit different events (92ms)
      ✓ Should not accept nonce of different space (47ms)
  Upgradeability
    ✓ Should update implementation (56ms)
    ✓ Should fail to set implementation to address 0 (59ms)
    ✓ Should fail to set implementation to non-contract (51ms)
    ✓ Should use implementation storage key (62ms)
  External calls
    ✓ Should perform call to contract (76ms)
    ✓ Should return error message (70ms)
   Batch transactions
      ✓ Should perform multiple calls to contracts in one tx (92ms)
      ✓ Should perform call a contract and transfer eth in one tx (82ms)
      ✓ Should fail if one transaction fails (84ms)
  Delegate calls
    ✓ Should delegate call to module (64ms)
   on delegate call revert
      ✓ Should pass if delegate call is optional
      ✓ Should fail if delegate call fails (53ms)
  Handle ETH
    ✓ Should receive ETH
    ✓ Should transfer ETH (53ms)
    ✓ Should call payable function (75ms)
  Optional transactions
    ✓ Should skip a skipOnError transaction (59ms)
    ✓ Should skip failing transaction within batch (106ms)
    ✓ Should skip multiple failing transactions within batch (103ms)
    ✓ Should skip all failing transactions within batch (84ms)
    ✓ Should skip skipOnError update implementation action (68ms)
  Hooks
   receive tokens
      ✓ Should implement ERC1155 single transfer hook
      ✓ Should implement ERC1155 batch transfer hook
      ✓ Should implement ERC721 transfer hook
      ✓ Should implement ERC223 transfer hook
   ERC1271 Wallet
      ✓ Should validate arbitrary signed data
      ✓ Should validate arbitrary signed hash
      ✓ Should reject data signed by non-owner
      ✓ Should reject hash signed by non-owner
   External hooks
      ✓ Should read added hook (62ms)
      ✓ Should return zero if hook is not registered
      ✓ Should forward call to external hook (90ms)
      ✓ Should not forward call to deregistered hook (89ms)
      ✓ Should pass calling a non registered hook
      ✓ Should use hooks storage key (43ms)
  Update owners
    After a migration
      ✓ Should implement new upgradable module
      ✓ Should accept new owner signature (48ms)
      ✓ Should reject old owner signature (67ms)
      ✓ Should fail to update to invalid image hash (57ms)
      ✓ Should fail to change image hash from non-self address
      ✓ Should use image hash storage key
     After updating the image hash
        ✓ Should have updated the image hash
        ✓ Should accept new owners signatures
        ✓ Should reject old owner signatures (39ms)
        ✓ Should use image hash storage key
  Multisignature
    With 1/2 wallet
      ✓ Should accept signed message by first owner (45ms)
      ✓ Should accept signed message by second owner (42ms)
      ✓ Should accept signed message by both owners (43ms)
      ✓ Should reject message without signatures (60ms)
```

```
      ✓ Should reject message signed by non-owner (60ms)
    With 2/2 wallet
      ✓ Should accept signed message by both owners (41ms)
      ✓ Should reject message without signatures (50ms)
      ✓ Should reject message signed only by first owner (52ms)
      ✓ Should reject message signed only by second owner (56ms)
      ✓ Should reject message signed by non-owner (68ms)
    With 2/3 wallet
      ✓ Should accept signed message by first and second owner (45ms)
      ✓ Should accept signed message by first and last owner (58ms)
      ✓ Should accept signed message by second and last owner (53ms)
      ✓ Should accept signed message by all owners (52ms)
      ✓ Should reject message signed only by first owner (59ms)
      ✓ Should reject message signed only by second owner (45ms)
      ✓ Should reject message signed only by last owner (82ms)
      ✓ Should reject message not signed (52ms)
      ✓ Should reject message signed by non-owner (60ms)
      ✓ Should reject message if the image lacks an owner (79ms)
    With 255/255 wallet
      ✓ Should accept message signed by all owners (3457ms)
      ✓ Should reject message signed by non-owner (7219ms)
      ✓ Should reject message missing a signature (5656ms)
    With weighted owners
      ✓ Should accept signed message with (3+1)/4 weight (79ms)
      ✓ Should accept signed message with (3+3)/4 weight (75ms)
      ✓ Should accept signed message with (3+3+1+1)/4 weight (139ms)
      ✓ Should accept signed message with (3+3+1+1+1)/4 weight (81ms)
      ✓ Should reject signed message with (1)/4 weight (94ms)
      ✓ Should reject signed message with (1+1)/4 weight (97ms)
      ✓ Should reject signed message with (1+1+1)/4 weight (96ms)
      ✓ Should reject signed message with (3)/4 weight (108ms)
      ✓ Should reject signed message with (0)/4 weight (143ms)
      ✓ Should reject message signed by non-owner (112ms)
  Gas limit
    ✓ Should forward the defined amount of gas (57ms)
    ✓ Should forward different amounts of gas (2003ms)
    ✓ Should fail if forwarded call runs out of gas
    ✓ Should fail without reverting if optional call runs out of gas (201ms)
    ✓ Should continue execution if optional call runs out of gas (284ms)
  Create contracts
    ✓ Should create a contract (104ms)
    ✓ Should create a contract with value (95ms)
    ✓ Should fail to create a contract from non-self
  Transaction events
    ✓ Should emit TxExecuted event (65ms)
    ✓ Should emit multiple TxExecuted events (55ms)
  Interanl bundles
    ✓ Should execute internal bundle (161ms)
    ✓ Should execute multiple internal bundles (539ms)
    ✓ Should execute nested internal bundles (169ms)
    ✓ Should revert bundle without reverting transaction (772ms)


  184 passing (41s)

Failed to generate 4 stack traces. Run Buidler with --verbose to learn more.
 ✓  Done in 64.16s.
```

Code Coverage

The coverage configuration was performed based on commit `07be0fc995553b8af9697de3ce25c22c949769ad`. The coverage appears good, however, it shows that some branches of the ERC165 are not hit, and that the claimed signature support based on EIP712 is not tested. Quantstamp recommends adding tests for these scenarios and features.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	100	100	100	100	
Factory.sol	100	100	100	100	
Wallet.sol	100	100	100	100	
contracts/interfaces/	100	100	100	100	
IERC1271Wallet.sol	100	100	100	100	
contracts/interfaces/receivers/	100	100	100	100	
IERC1155Receiver.sol	100	100	100	100	
IERC223Receiver.sol	100	100	100	100	
IERC721Receiver.sol	100	100	100	100	
contracts/migrations/	0	0	0	0	
Migrations.sol	0	0	0	0	… 14,18,22,23
contracts/mocks/	**94.44**	**70**	**100**	**94.55**	
CallReceiverMock.sol	100	100	100	100	

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
DelegateCallMock.sol	100	100	100	100	
ERC165CheckerMock.sol	76.92	50	100	78.57	14,19,26
GasBurnerMock.sol	100	100	100	100	
HookCallerMock.sol	100	70	100	100	
HookMock.sol	100	100	100	100	
LibBytesImpl.sol	100	100	100	100	
ModuleMock.sol	100	100	100	100	
contracts/modules/	100	100	100	100	
MainModule.sol	100	100	100	100	
MainModuleUpgradable.sol	100	100	100	100	
contracts/modules/commons/	100	95.24	97.73	99.04	
Implementation.sol	100	100	50	50	27
ModuleAuth.sol	100	100	100	100	
ModuleAuthFixed.sol	100	100	100	100	
ModuleAuthUpgradable.sol	100	100	100	100	
ModuleCalls.sol	100	100	100	100	
ModuleCreator.sol	100	100	100	100	
ModuleERC165.sol	100	100	100	100	
ModuleHooks.sol	100	75	100	100	
ModuleSelfAuth.sol	100	100	100	100	
ModuleStorage.sol	100	100	100	100	
ModuleUpdate.sol	100	100	100	100	
contracts/modules/commons/interfaces/	100	100	100	100	
IModuleAuth.sol	100	100	100	100	
IModuleAuthUpgradable.sol	100	100	100	100	
IModuleCalls.sol	100	100	100	100	
IModuleCreator.sol	100	100	100	100	
IModuleHooks.sol	100	100	100	100	
IModuleUpdate.sol	100	100	100	100	
contracts/utils/	85.71	77.27	100	88.24	
LibAddress.sol	100	100	100	100	
LibBytes.sol	100	100	100	100	
SignatureValidator.sol	73.33	50	100	71.43	66,70,75,92
All files	**93.65**	**82.56**	**93.98**	**93.1**	

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

```
cd7fed83658714a715a017e5d193e818f11e121fd1131be0855974744d2a63db ./contracts/Factory.sol

9b4fdec96ca9533f192adf06348f0253400722aebafb0747500c12c4c1fca5cf ./contracts/Wallet.sol

5afcece4dbc14093b4303827d5dbdfc851d933cf884971fa0d690f6829e64207 ./contracts/interfaces/IERC1271Wallet.sol
```

76963ce2100c4e752ab72e0af541d8276c5db01458d38cd3e8a1b478c83c30a8
./contracts/interfaces/receivers/IERC223Receiver.sol

b1cd9bd8499b8f5dafaa4b11594e98d4042469d3b5c390dd1422c2de1fc33d5f
./contracts/interfaces/receivers/IERC1155Receiver.sol

7366146eb098d5aaa14085b720c2e1b16638193812b90319364c17a3c0045912
./contracts/interfaces/receivers/IERC721Receiver.sol

dc14091238e3028d8eeee9328118445687252c085a857ee5b6864d4bcd9f5440 ./contracts/modules/MainModule.sol

107a6c0dc7b86f0968453fd5adc18c05085a11ba6f0be9a68c3f7339a10507af ./contracts/modules/MainModuleUpgradable.sol

4a4f6f8ba8e35c680c97b74eabadd71b0571237fd7dcb7fadb7a3e59ff94d022 ./contracts/modules/commons/ModuleCalls.sol

e2533bab996a2f15a40c11699124ea19f97c75d4e4a8b681b41029d2b2e56caf ./contracts/modules/commons/ModuleCreator.sol

84854a837e590358a047234b5c2091ee7923342c8e368eebbb87d07ed39aba77 ./contracts/modules/commons/ModuleAuth.sol

a8fbb5c0d34400c5aae011546ec0ba7f49ee57d932bc2ec5e3ca241ffbe5b882 ./contracts/modules/commons/ModuleERC165.sol

062b4d5132ede27ec59fe53be58de95e4a4f5c682f7e21bc5696ff1c5df15b85 ./contracts/modules/commons/ModuleHooks.sol

f3758cd78c6bdb67047f6dd6faf8607e3aa7889410e7d537a92489f35e8a6908 ./contracts/modules/commons/ModuleStorage.sol

38f206823b3190fffa885805ae16a03a6c7230b691aec5dc027bea700e12b15f ./contracts/modules/commons/Implementation.sol

77aef1d2bea479d84cba9e81707263da448b4b354d99bdd301992cd02b3f6269 ./contracts/modules/commons/ModuleUpdate.sol

06336480f0f2b7d00511fad8946cd7227f8747257648499ede9e1e11bd9d9bfb ./contracts/modules/commons/ModuleSelfAuth.sol

0cad59d85eaf30ad99339101e6d74f659abdabab834e2f999bbf3f1fa8a32dfb ./contracts/modules/commons/ModuleAuthFixed.sol

4283594a1e566baf5bd1ebf61108838725cae32a3ee0e242c19213d5f0683c4b
./contracts/modules/commons/ModuleAuthUpgradable.sol

1ae03058a0a15d87365d53ad9b6b082bc9dc7663d7281b34b36dbfd463c2ef30
./contracts/modules/commons/interfaces/IModuleHooks.sol

f3151bcff905a60a047f23699bd95716a909e9b9cc578188b06172dbea0c2c53
./contracts/modules/commons/interfaces/IModuleCalls.sol

c938ce5a701477cc7f6b683d2fcd410db1789d6e7ec4e99673e1e01a03b31cd8
./contracts/modules/commons/interfaces/IModuleCreator.sol

5bab2cd8aadac35471706a4146962c14b6e2a7f8ff98f478618ab6b0e85e2fae
./contracts/modules/commons/interfaces/IModuleUpdate.sol

7e5563418d52af06cd2cefb649b094c9babe5ebb622de3783af6f27d3b152779
./contracts/modules/commons/interfaces/IModuleAuthUpgradable.sol

19a04c808ed139f369371ba09931dca77813ea41e8670fac384715fad4dd261a
./contracts/modules/commons/interfaces/IModuleAuth.sol

d18461585b69068c3e4c0b424052f32d825e4cc93381326af1266d06bd6d80b0 ./contracts/utils/LibAddress.sol

8ab99fa9823de7da1fd658decd0764da7fc48165f11243b63bed5f85fa99394f ./contracts/utils/SignatureValidator.sol

fc1ed8932ab026da74d94c9a35fa7fe07c3cac73c57d47137a5ac3a3997a6287 ./contracts/utils/LibBytes.sol

c5ee823d8eeca0f1786c0b0e61e2844ab79862a373e253e4aaf2e22ff58c4c61 ./contracts/migrations/Migrations.sol

4169e4e921d60393c37a2eafe18c355220901a746316ae10c5215b2f8dac2209 ./contracts/mocks/CallReceiverMock.sol

d507ad9aff4a31875762a7513f8ccddab85219b131929ee1ffa939e57d8fb5fd ./contracts/mocks/LibBytesImpl.sol

c0e43df18552363899481188062b091f11d36532c7dc7389cbecd4db1510ba8e ./contracts/mocks/HookCallerMock.sol

b31322fac375ec8144c0a50913ff98dbcc095e85141cfc46936c384875b02056 ./contracts/mocks/GasBurnerMock.sol

07d6bb5edc784d4c6de08364670c117356eb1b8fe41d64ebc9798aaa5e95a9d3 ./contracts/mocks/DelegateCallMock.sol

df1250ed549a22b22824e55ffafecc610b829cdbbbf5184611c5e16b984ab65e ./contracts/mocks/HookMock.sol

e91fd141fa61d5faf3b9b663ff357493cf206db379518c37f561fccb12d770db ./contracts/mocks/ModuleMock.sol

d868f391016b421c51441b5ed8e42b102fc6f9d294d431a77f0d2d3e7ee992ea ./contracts/mocks/ERC165CheckerMock.sol

Tests

539fc189046ff8668d3e72b303244db17ff9ad2ebc493175c4442c9c7b552994 ./tests/LibBytes.spec.ts

f5d672b708a621abbca6d6cf4fd8cb0b945726d2bfe86ab8eb29da761b32e5f1 ./tests/MainModule.spec.ts

a7884294fe0f36e33f8bf2d075308f4487132ecf1c6ae369a2d2f84aedcf0a70 ./tests/MainModule.bench.ts

4589cd07d1a7c3f49d77a95bd6f9267b0b830aeff16451e468eaa0294c543531 ./tests/ERC165.spec.ts

02fc01471af8eec1c81288df69b61c9ab1a7d1c15909e17fa57bd3800004df84 ./tests/Factory.spec.ts

a1f2ca6ca10eae714306de1e07d1d5b9e3d5bca8e964a981827fa29b58945471 ./tests/utils/helpers.ts

56cb327ecd10477a223ab6b60318d2cf077c64e422cd2aff87c7d9c11d603e49 ./tests/utils/contract.ts

0ef646c2dea4704415e4d2bbe3bd7995228d7d5a18313620863777729fe5bffa4 ./tests/utils/index.ts

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 $1B 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.