Quantstamp Contract Security Certificate

April 29th 2020 — Quantstamp Verified

Multis

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

Executive Summary

Type	GSN-enabled Multisig
Auditors	Kacper Bąk, Senior Research Engineer Alex Murashkin, Senior Software Engineer Martin Derka, Senior Research Engineer
Timeline	2019-12-11 through 2020-04-29
EVM	Istanbul
Languages	Solidity, Javascript
Methods	Architecture Review, Computer-Aided Verification, Manual Review
Specification	GSN-enabled Multisig
Source Code	see table below

Repository	Commit
MULTISig	635f670
MULTISig	517c472

Goals

- Can user's funds get locked up in the contract?
- Is upgradeability implemented correctly?
- May unauthorized users perform transactions thru the multisig?

Changelog

- 2019-12-13 - Initial report
- 2020-01-10 - Reaudit based on commit `517c472`
- 2020-04-24 - Fallback function fix based on commit `de8c6dd`
- 2020-04-29 - Revised report based on commit `54f1694`

Overall Assessment	
	Overall the code is well-written, however, we found a few low-risk issues. We recommend addressing them and/or considering the consequences of ignoring the issues. **We strongly recommend testing the contracts on Istanbul mainnet to ensure that the contracts work as intended.** Furthermore, although our audit focused on the fork diff vs the Gnosis implementation [commit 95d51ae](), we reviewed the whole codebase. Furthermore, we assumed that the used OpenZeppelin contracts were audited and, if necessary, fixed. **Update:** the team has acknowledged or resolved a few of the reported issues. Also, a test suite has been added to the project. **Update:** Quantstamp confirms that the reported inability of the contract to accept Ether via `transfer()` and `send()` under Istanbul EVM is fixed in commit `54f1694`. However, it should be noted that the use of `msg.sender` goes against the [recommended API use of GSN network]() as this field would be the address of the RelayHub instead of the user. The Multis team considers the consequences of the mismatch in this scenario benign as the `msg.sender` address is only used for emitting an event. The change does not appear to have impact on the rest of the contract's functionality, however, the interactions were not subject to the re-audit.

Total Issues	**8**	(2 Resolved)
High Risk Issues	0	(0 Resolved)
Medium Risk Issues	0	(0 Resolved)
Low Risk Issues	**3**	(0 Resolved)
Informational Risk Issues	**3**	(1 Resolved)
Undetermined Risk Issues	**2**	(1 Resolved)

3 Unresolved
3 Acknowledged
2 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.

Summary of Findings

ID	Description	Severity	Status
QSP-1	Malicious co-owner may deplete the multisig creator's funds for GSN	Low	Unresolved
QSP-2	Privileged Roles and Ownership	Low	Acknowledged
QSP-3	Integer Overflow / Underflow	Low	Unresolved
QSP-4	Unlocked Pragma	Informational	Resolved
QSP-5	Race Conditions / Front-Running	Informational	Acknowledged
QSP-6	Theoretically possible integer overflow	Informational	Unresolved
QSP-7	Compatibility of the contracts with the Istanbul Ethereum hard fork	? Undetermined	Acknowledged
QSP-8	Malicious user can spam the array `deployedWallets`	? Undetermined	Resolved

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:

- Maian
- Mythril
- Securify
- Slither

Steps taken to run the tools:

1. Installed the Mythril tool from Pypi: `pip3 install mythril`
2. Ran the Mythril tool on each contract: `myth -x path/to/contract`
3. Ran the Securify tool: `java -Xmx6048m -jar securify-0.1.jar -fs contract.sol`
4. Cloned the MAIAN tool: `git clone --depth 1 https://github.com/MAIAN-tool/MAIAN.git maian`
5. Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract.sol`
6. Installed the Slither tool: `pip install slither-analyzer`
7. Run Slither from the project directory `slither .`

Assessment

Findings

QSP-1 Malicious co-owner may deplete the multisig creator's funds for GSN

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `GSNMultiSigWallet.sol`

Description: Current implementation requires trusting that the multisig owners behave well. A malicious co-owner (or when their account gets hacked) could: 1) repeatedly call `submitTransaction()` for random stuff; or 2) just alternate between calling `confirmTransaction()` and `revokeConfirmation()` repeatedly. Consequently, the multisig creator's balance used for GSN payments could get depleted. It is possible because the function `acceptRelayedCall()` accepts all requests regardless of the cost.

Recommendation: We recommend monitoring the usage of the contract and the balance used for GSN payments. In case of problems, the function `removeOwner()` may be used to kick out a poorly behaving owner.

QSP-2 Privileged Roles and Ownership

Severity: *Low Risk*

Status: Acknowledged

File(s) affected: `GSNMultisigFactory.sol`

Exploit Scenario:

1. The owner calls `removeMinter()` for the only minter that was available as of the initialization time.
2. Although the function `addMinter()` exists, nobody has the minter role, therefore, there is no way to add the minter back and make the contract operational again.

Recommendation: Privileged Roles and Ownership needs to be made clear to the users, especially because removing the owner and/or minter may lead to a DoS. Depending on the intended use, you may consider removing the function `removeMinter()` and/or owner altogether.

QSP-3 Integer Overflow / Underflow

Severity: *Low Risk*

Status: Unresolved

File(s) affected: `GSNMultiSigWallet.sol`

Description: Integer overflow/underflow occur when an integer hits its bit-size limit. Every integer has a set range; when that range is passed, the value loops back around. A clock is a good analogy: at 11:59, the minute hand goes to 0, not 60, because 59 is the largest possible minute. Integer overflow and underflow may cause many unexpected kinds of behavior. Specifically, integer underflow may occur in line 401 in the statement `to - from`.

Recommendation: We recommend checking that `to` >= `from`.

QSP-4 Unlocked Pragma

Severity: *Informational*

Status: Resolved

File(s) affected: `GSNMultiSigWallet.sol`, `GSNMultiSigWalletWithDailyLimit.sol`, `GSNMultisigFactory.sol`

Related Issue(s): SWC-103

Description: Every Solidity file specifies in the header a version number of the format `pragma solidity (^)0.4.*`. The caret (^) before the version number implies an unlocked pragma, meaning that the compiler will use the specified version *and above*, hence the term "unlocked."

Recommendation: For consistency and to prevent unexpected behavior in the future, it is recommended to remove the caret to lock the file onto a specific Solidity version.

QSP-5 Race Conditions / Front-Running

Severity: *Informational*

Status: Acknowledged

File(s) affected: `GSNMultiSigWallet.sol`

Related Issue(s): SWC-114

Description: A block is an ordered collection of transactions from all around the network. It's possible for the ordering of these transactions to manipulate the end result of a block. Specifically, there is a transaction order dependence (TOD) between the functions `revokeConfirmation()` and `executeTransaction()`.

Recommendation: We don't have a recommendation as of now, however, we wanted to point out this concern.

QSP-6 Theoretically possible integer overflow

Severity: *Informational*

Status: Unresolved

File(s) affected: `GSNMultiSigWalletWithDailyLimit.sol`

Related Issue(s): SWC-101

Description: The contract assumes that the daily limit for spent ether is a number that can be represented by `uint256` due to the type of `spentToday`. Lines 66 and 73 use regular addition and subtraction. If ether supply ever required more than 256 bits to represent it, the contract might need to be redeployed to avoid integer overflow.

Recommendation: Currently we have no recommendation. As of now, this issue is theoretical, not practical.

QSP-7 Compatibility of the contracts with the Istanbul Ethereum hard fork

Severity: *Undetermined*

Status: Acknowledged

File(s) affected: `GSNMultiSigWallet.sol`

Description: Gas usage is a main concern for smart contract developers and users. The recent hard fork of Ethereum, Istanbul, repriced the gas cost of instructions (see EIP-1679). The function `external_call()` hardcodes a gas cost constant `34710` in line 263.

Recommendation: To confirm compatibility of the contract with Istanbul fork, we recommend: 1) creating a test suite, 2) performing manual tests on the mainnet, and 3) performing gas analysis.
Update: the team informed us that they are going to add a test suite and perform manual tests on the mainnet to confirm the compatibility.

QSP-8 Malicious user can spam the array `deployedWallets`

Severity: *Undetermined*

Status: Resolved

File(s) affected: `GSNMultisigFactory.sol`

Description: The factory allows anybody to create a new wallet via the function `create()`. Consequently, there is no limit on the number of wallets that can be created. A malicious user could "spam" the array `deployedWallets` with a large number of empty wallets, which could make the array retrieval a more time-consuming process.

Recommendation: Currently we have no recommendation since the severity and likelihood of the problem, and consequences are unclear.

Automated Analyses

Maian

MAIAN failed to deploy the contract, and, therefore could not complete the analysis.

Mythril

Mythril reported the following:

- integer overflows in `GSNMultiSigWallet.getTransactionIds()`, `SafeMath.add()`, `UINT256_MAX`. We classified them as false positives.
- exception states in `address[] public owners;` in `Ownable.sol`, and in `_transactionIds[i - from] = transactionIdsTemp[i];` in `GSNMultiSigWallet.sol`. We classified them as false positives.
- integer underflow in `GSNMultiSigWallet.sol#401`. It is a true positive.

Securify

Securify reported the following:

- locked ether in `GSNMultiSigWallet.sol` due to the payable function. We classified it as a false positive since owners may submit a transfer transaction.
- missing input validation in multiple locations. Upon closer inspection, we classified them as false positives.

Slither

Slither reported locked ether in `GSNMultiSigWalletWithDailyLimit` since it has a payable fallback function, but it does not have an explicit function to withdraw the ether. We classified it as a false positive since owners may submit a transfer transaction.

Code Documentation

There is no documentation on the order of operations when it comes to creating a new contract, funding the contract, granting tokens to potential multisig creators, etc. Lack of documentation will make it harder for contract users to understand the expected behavior and deployment process.

Update: the team has added documentation in `README.md`. Two remarks:

- in line 6: "[todo: link to medium blog post about gasless]". We recommend resolving the TODO item.
- in line 30: "finaly". We recommend fixing the typo.

Adherence to Best Practices

The code adheres to best practices, however, in `GSNMultiSigWallet.sol#122` and `GSNMultiSigWalletWithDailyLimit.sol#36`: the commented out code should be either removed or annotated that it is a reference implementation. **Update:** resolved.

Test Results

Test Suite Results

The code comes with a test suite. It runs a ganache instance with the arguments `--allowUnlimitedContractSize` and `--gasLimit=97219750`. Specifying contract size and a higher gas limit for running tests does not imply that the mainnet deployment is working. We recommend running ganache with default parameters to make sure the deployment and tests work as expected.

```
GSNMultiSigWallet
   ✓ Fail execution of transaction (224ms)
   ✓ Execute transaction (247ms)
   ✓ Accept relay call (57ms)
GSNMultiSigWalletWithDailyLimit
   ✓ create multisig (57ms)
   ✓ receive deposits
   ✓ withdraw below daily limit (185ms)
   ✓ update daily limit (741ms)
   ✓ execute various limit scenarios (5904ms)
GSNMultisigFactory
   ✓ Add minter, mint and remove minter (386ms)
   ✓ Create contract from factory (263ms)
   ✓ Send money to contract (123ms)
   ✓ Receive money from contract transfer (118ms)
   ✓ Update daily limit (361ms)
   ✓ Add owner (366ms)
   ✓ Replace owner (587ms)
   ✓ Remove owner (1665ms)
   ✓ Revoke confirmation (466ms)
   ✓ Change requirement (360ms)
   ✓ Execute transaction (352ms)
   ✓ Fail execution of transaction (176ms)
   ✓ Fail execution of transaction below daily limit (655ms)
21 passing (39s)
```

Code Coverage

The code features reasonable coverage.

File	% Stmts	% Branch	% Funcs	% Lines	Uncovered Lines
contracts/	99.24	81.67	95.12	99.31	
DumbTransfer.sol	100	100	100	100	
GSNMultiSigWallet.sol	98.91	75	93.33	99.04	413
GSNMultiSigWalletWithDailyLimit.sol	100	100	100	100	
GSNMultisigFactory.sol	100	100	100	100	
All files	**99.24**	**81.67**	**95.12**	**99.31**	

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

```
40010c40c8b8e8c057d7e1658b3d240374d10e9bd818e0156a97944470e8c974 ./contracts/GSNMultiSigWalletWithDailyLimit.sol

47022d5d2b94a4289be051fe112b4b0c0b0d78e00c882f5019f57776e1582448 ./contracts/DumbTransfer.sol

41286b0c54c4acead151f1664b25fe936bc3762131e2b49c7ca4521a63c4871a ./contracts/GSNMultisigFactory.sol

8877bc69314fcc5aad5dffbe87db3cc7ea7b060a4485c0f8c2e4230fc3a18752 ./contracts/GSNMultiSigWallet.sol
```

Tests

```
b47d33045959c600266892ce564b7e892399acf011a65d612fcc6a9d2bcdd1de ./test/javascript/testGSNMultisigFactory.js

19c07869bb37682a5def7718762f2fc87d2092ca5c14472d2f2434e6e58d734f
./test/javascript/testGSNMultiSigWalletWithDailyLimit.js

3b9bb0abfdf2544581452e73221ec583afb079246e7ffd3afe76e2837d576080 ./test/javascript/utils.js

52efe2c0372bbbae910f97c0aafd49652782c7556bccdf4cf67aef265c1e34e3 ./test/javascript/testGSNMultiSigWallet.js
```

About Quantstamp

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

Quantstamp's team boasts decades of combined experience in formal verification, static analysis, and software verification. Collectively, our individuals have over 500 Google scholar citations and numerous published papers. In its mission to proliferate development and adoption of blockchain applications, Quantstamp is also developing a new protocol for smart contract verification to help smart contract developers and projects worldwide to perform cost-effective smart contract security audits.

To date, Quantstamp has helped to secure hundreds of millions of dollars of transaction value in smart contracts and has assisted dozens of blockchain projects globally with its white glove security auditing 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.

Finally, Quantstamp's dedication to research and development in the form of collaborations with leading academic institutions such as National University of Singapore and MIT (Massachusetts Institute of Technology) reflects Quantstamp's commitment to enable world-class smart contract innovation.