Quantstamp Security Assessment Certificate

July 9th 2021 — Quantstamp Verified

SEXP - Synthetic Asset Exchange on Tezos

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

Executive Summary

Type	Binary Option Defi App
Auditors	Poming Lee, Research Engineer Ed Zulkoski, Senior Security Engineer Christoph Michel, Research Engineer
Timeline	2021-05-23 through 2021-07-09
Languages	Go, Michelson
Methods	Architecture Review, Unit Testing, Functional Testing, Manual Review
Specification	README.md
Documentation Quality	Undetermined
Test Quality	Undetermined

Source Code

Repository	Commit
sexp-binary-options	c6e1cb8
sexp-binary-options	1ce0875

Total Issues	15 (10 Resolved)
High Risk Issues	1 (1 Resolved)
Medium Risk Issues	3 (2 Resolved)
Low Risk Issues	5 (2 Resolved)
Informational Risk Issues	4 (4 Resolved)
Undetermined Risk Issues	2 (1 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

To summarize, given the complexity of coding tracing for code written in Michelson language and relatively few material and bug-free tools that can be found to aid the audit process, and also the lack of documentation for the project itself, there are very likely still issues that we are not able to find. Quantstamp has on a best efforts basis identified 15 total issues, with 3 auditors performing audits side-by-side; however we highly suggest getting more reviews before launching this project. During auditing, we found 1 high-severity, 3 medium-severity, 5 low-severity issues, 2 undetermined-severity issues, as well as 4 informational-level findings. We made 3 best practices recommendations.
The documentation of the project is insufficient and the quality of the audit could be largely improved if there were more specifications that describe all the intended behaviors and precision requirements. The inclusion of extensive tests and/or formal methods to assure extensive quality and behavior could also help. Normally attackers would use fuzzing techniques to find holes in any smart contract logic with substantial value locked. Avoid implementing your own arithmetic like fixed-point arithmetic, use existing implementations or standards are also advantageous to help increase the security. The coverage data was not generated due to the fact that there are no existing tools that can be used to generate this data for tests for Michelson language. We strongly recommend the SEXP team to find a way to fix this and obtain a code coverage report that states that all the code coverage values are at least 90% before going live, to reduce the potential risk of having functional bugs in the code.
Disclaimer: Please be aware that Quantstamp was requested and had audited two contract files in the repository, and they are: `fa2_with_factory.tz` and `binary_option_market.tz`; not the whole system was audited. Also, this project utilized Tezos blockchain and Harbinger oracle service. All the dependencies and external infrastructures are not part of this audit. Economic attacks are outside the scope of this audit.

2021-07-09 update: during this reaudit, the SEXP team has either fixed or acknowledged all findings.

ID	Description	Severity	Status
QSP-1	Self transfers reduce balance	High	Fixed
QSP-2	Option tokens can get stuck at DEXes	Medium	Fixed
QSP-3	Market earnings is used to favor the long option bidder	Medium	Fixed
QSP-4	Dangerous external calls from `fa2_with_factory.tz` to arbitrary contact by anyone	Medium	Acknowledged
QSP-5	Price data from oracle contract could be manipulated by the signed data provider	Low	Acknowledged
QSP-6	Price data from oracle contract could be manipulated by people who owns lots of funds	Low	Acknowledged
QSP-7	Market can be resolved at any time after expiry	Low	Acknowledged
QSP-8	FA2.balance_of inconsistent behavior	Low	Fixed
QSP-9	`minCapital` equality	Low	Fixed
QSP-10	Error message does not match FA2 specification (insufficient balance)	Informational	Fixed
QSP-11	Error message does not match FA2 specification (undefined token)	Informational	Fixed
QSP-12	Error message does not match FA2 specification (not operator)	Informational	Fixed
QSP-13	Privileged roles and ownership	Informational	Fixed
QSP-14	Option tokens for the losing side will never be burned	Undetermined	Acknowledged
QSP-15	Anyone can burn their option tokens	Undetermined	Fixed

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.

Findings

QSP-1 Self transfers reduce balance

Severity: *High Risk*

Status: Fixed

Description: For `fa2_with_factory.tz`, the tzip-12 specification states regarding transfers:

> Transfers with the same address (`from_` equals `to_`) MUST be treated as normal transfers.

However, self transfers appear to deduct from the user's balance rather than it remaining constant.

Exploit Scenario: Given sender `tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx` and an example storage:

```
(Pair { Elt "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx"
           { Elt 0
                (Pair 200
                      { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) ;
             Elt 1
                (Pair 300
                      { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) } }
      (Pair (Pair "2022-01-01T03:41:46Z" "KT1Nxpog4bJiQzW4DfmtNXk2Hv1tZHLLCUNZ")
           { Elt "hello" 0xdeadc0de }))
```

The following parameter value deduct 50 of token 0 and 75 of token 1 from the user:

```
{ (Pair "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx" {
  (Pair "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx" (Pair 0 50));
  (Pair "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx" (Pair 1 75));
}) }
```

Recommendation: Ensure self transfers do not affect balances.

QSP-2 Option tokens can get stuck at DEXes

Severity: *Medium Risk*

Status: Fixed

Description: For `fa2_with_factory.tz`, the option token contract disables transfers after `expiration` (which is initialized with `market.trading_end`), unless the `SENDER` is the option market. It could be the case that option tokens are still locked up in a decentralized exchange (DEX) or any other protocol that holds the tokens in custody of the user, such that exercising them

would not credit the profits to the user but to the protocol instead. The user is then unable to exercise their locked options after expiration.

Recommendation: Consider whether a trading restriction is really necessary. As long as the price is locked in after the trading phase, it might not be a problem that tokens are still transferred as the losing side should be worthless, and the winning side's price should tend towards the exercise price.

Update: The trading restriction has been removed

QSP-3 Market earnings is used to favor the long option bidder

Severity: *Medium Risk*

Status: Fixed

Description: For `binary_option_market.tz`, the "market earnings" in the "market contract" is used to lower the price for long bids and increase the price for short bids. This is because the short bid price is directly derived from long bid price by directly inverting it, which is unfair and does not appear to be intended, based on the code comments.

Recommendation: The market earnings should be distributed proportional to the existing long/short bids.

Update: Fixed by changing the formula.

QSP-4 Dangerous external calls from `fa2_with_factory.tz` to arbitrary contact by anyone

Severity: *Medium Risk*

Status: Acknowledged

Description: `fa2_with_factory.tz`: entry point `balance_of`: can be called by any user to call any external contracts (by inserting it to the entry point `contract %callback`). This enables a user to have privilege over the `fa2_with_factory.tz` contract and could be used as a tool to conduct a complex attack.

Recommendation: Consider limiting the target calling contracts of this function to only the market contract with its entry point `%exercise`. Or limiting the sender of this function. Otherwise, state this risk explicitly to your public document.

Update: SEXP team decided to leave it unchanged because the `%balance_of` entrypoint is meant to be used by anyone and allowing whitelisted callers only would break composability with other contracts on Tezos blockchain.

QSP-5 Price data from oracle contract could be manipulated by the signed data provider

Severity: *Low Risk*

Status: Acknowledged

Description: `binary_option_market.tz`: the price data provided by the oracle that is used for determining the winning side of the option can be manipulated by the signed data provider of the Harbinger oracle.

Recommendation: This potential risk vector needs to be made clear to the users, especially when the economic incentives of performing this manipulation action is too high.

Update: SEXP team stated that they will make this risk clearer in the documentation.

QSP-6 Price data from oracle contract could be manipulated by people who owns lots of funds

Severity: *Low Risk*

Status: Acknowledged

Description: `binary_option_market.tz`: the price data provided by the oracle that is used for determining the winning side of the option can be manipulated by anyone that owns lots of funds and use those funds to manipulate the market data received by the signed data provider, for seconds, which could influence the result generated by the Price Normalizer Contract of the Harbinger oracle since it only averages the volume weighted average price from the last n updates.

Recommendation: Perform sanity checks of the received price data to avoid this type of last-minute price changes attack.

Update: SEXP team decided not to add a sanity check since the cost of carrying out this type of attack is too high. Following is the statement from SEXP team: “After investigating this possible attack vector and consulting with the Harbinger team, we think an attack like this would be very expensive. The Harbinger oracle requires each update to have a monotonically increasing timestamp. Signed price data (candles) are generated by Coinbase once a minute, if there are trades. The resulting VWAP is calculated from 6 data points (6 minutes if updates are pushed as soon as possible). Therefore, an attacker that manipulates the market price on Coinbase for just a few seconds wouldn’t be able to influence the Harbinger price too much.”

QSP-7 Market can be resolved at any time after expiry

Severity: *Low Risk*

Status: Acknowledged

Description: `binary_option_market.tz`: based on the specification provided in harbinger-repo the Harbinger oracle does not offer the transaction generation and price data push service for dapps. This indicates that the `(address %harbinger)` stored in the `storage` of the market contract might not be the contracts directly provided by the Harbinger oracle team, and the behavior of this contract is rather unknown. Furthermore, the market can be resolved via the Harbinger oracle callback `market.receive_prices` entry point at any time after `trading_end`. The code just checks that the oracle price's timestamp (`last_update`) is after the end and not in the future. (`trading_end <= last_update <= NOW`). A binary options market should resolve with the price right at `trading_end`. If an attacker would lose money if the market was resolved at the current price, they can delay other users from resolving the market, for example, by congesting the network. They can then wait for the price to swing in the other direction and resolve the market only when profitable for them.

Recommendation: Ensure watchtowers are robust to mitigate this potential issue. Please make sure that `(address %harbinger)` calls the entry point `receive_prices` immediately after the `trading_end` time is over, instead of making the length of this delay a manipulative parameter. Ideally, an oracle would be used that can be queried for the price of an asset at a specific time, the `trading_end` timestamp. If that's not practically feasible, make sure to resolve markets close to the `trading_end` and communicate the risk with the end users.

Update: SEXP team will try to make sure that the watchtower is robust enough to mitigate this issue.

QSP-8 FA2.balance_of inconsistent behavior

Severity: *Low Risk*

Status: Fixed

Description: For `fa2_with_factory.tz`, the `FA2.balance_of` entry point fails with a "no such token" message if the `owner` is in the `options big_map` but does not own any data in regard to the given token id (`storage.options[owner]` exists but `storage.options[owner][token_id]` does not). On the other hand, if the `owner` is currently not even set in the `options bigmap` (`storage.options[owner]` does not exist, i.e., has never been initialized), the transaction does not fail and instead declares the balance for the `(owner, token_id)` as

`0`. This contradicts the TZIP-12/FA2 token standard:

> If one of the specified `token_id`s is not defined within the FA2 contract, the entry point MUST fail with the error mnemonic "FA2_TOKEN_UNDEFINED".

When requesting a token ID different from `0` or `1` it should always fail with the "FA2_TOKEN_UNDEFINED" error, instead of returning a zero balance.

Recommendation: Fail with "FA2_TOKEN_UNDEFINED" in case a token ID different from `0` or `1` (the valid option token IDs) is requested. Otherwise, return the value stored in the `options big_map` or zero if no such value exists.

Update: Fixed by rejecting a token id that is neither `0` nor `1`.

QSP-9 `minCapital` equality

Severity: *Low Risk*

Status: Fixed

Description: For `binary_option_market.tz`, the `market.init` entry point allows `total_bids` that are equal to the `min_capital` (throwing on `if !(args.long + args.short >= storage.min_capital)`). However, when the admin cancels their bid it's checked that the total bids are strictly greater than `min_capital` (throwing on `if !(storage.min_capital < admin_bids)`).

Recommendation: Consider allowing a new total bid that is equal to the `min_capital` by using `LE` instead of `LT` in this code:

```
{ COMPARE;
  LT;
  # if storage.min_capital < admin_bids
  IF {  }
      { PUSH string "not enough capital left";
        FAILWITH } }
```

QSP-10 Error message does not match FA2 specification (insufficient balance)

Severity: *Informational*

Status: Fixed

Description: For `fa2_with_factory.tz`, the tzip-12 specification states regarding transfers:

> If the transfer amount exceeds the current token balance of the source address, the whole transfer operation MUST fail with the error mnemonic "FA2_INSUFFICIENT_BALANCE".

This case does not appear to be explicitly handled with that mnemonic. Instead, the pattern on `L61` uses `SUB; ISNAT; ASSERT_SOME` to check that the remaining balance is non-negative.

Exploit Scenario: Given sender `tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx` and an example storage:

```
(Pair { Elt "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx"
           { Elt 0
                 (Pair 200
                       { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) ;
             Elt 1
                 (Pair 300
                       { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) } }
     (Pair (Pair "2022-01-01T03:41:46Z" "KT1Nxpog4bJiQzW4DfmtNXk2Hv1tZHLLCUNZ")
           { Elt "hello" 0xdeadc0de }))
```

The following parameter value will reach the above opcodes:

```
{ (Pair "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx" {
  (Pair "tz1ZdZDr5pLFbNLzgWQBCpm27ZgMSQPUYJN5" (Pair 0 450))
}) }
```

Recommendation: Push the string "FA2_INSUFFICIENT_BALANCE" and use the `FAILWITH` opcode.

QSP-11 Error message does not match FA2 specification (undefined token)

Severity: *Informational*

Status: Fixed

Description: For `fa2_with_factory.tz`, the tzip-12 specification states regarding transfers:

> If one of the specified `token_ids` is not defined within the FA2 contract, the entry point MUST fail with the error mnemonic "FA2_TOKEN_UNDEFINED".

This is not the case on `L58`, `L61` which uses the string "no such token".
A similar issue exists for the `balance_of` as handled on `L79`.

Exploit Scenario: Given an example storage:

```
(Pair { Elt "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx"
           { Elt 0
                 (Pair 200
                       { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) ;
             Elt 1
                 (Pair 300
                       { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) } }
     (Pair (Pair "2022-01-01T03:41:46Z" "KT1Nxpog4bJiQzW4DfmtNXk2Hv1tZHLLCUNZ")
           { Elt "hello" 0xdeadc0de }))
```

The following parameter value will reach the above opcodes:

```
{ (Pair "tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx" {
  (Pair "tz1ZdZDr5pLFbNLzgWQBCpm27ZgMSQPUYJN5" (Pair 2 50))
}) }
```

Recommendation: Push the string "FA2_TOKEN_UNDEFINED" and use the `FAILWITH` opcode.

QSP-12 Error message does not match FA2 specification (not operator)

Severity: *Informational*

Status: Fixed

Description: For `fa2_with_factory.tz`, the

tzip-12 specification states regarding transfers:

> If the address that invokes a transfer operation is neither a token owner nor one of the permitted operators, the transaction MUST fail with the error mnemonic "FA2_NOT_OPERATOR".

However, on L58 the error message used is "not allowed".

Exploit Scenario: Given sender `tz1f1S7V2hZJ3mhj47djb5j1saek8c2yB2Cx` and an example storage:

```
(Pair { Elt "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1"
            { Elt 0
                  (Pair 200
                        { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) ;
              Elt 1
                  (Pair 300
                        { "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" }) } }
      (Pair (Pair "2022-01-01T03:41:46Z" "KT1Nxpog4bJiQzW4DfmtNXk2Hv1tZHLLCUNZ")
            { Elt "hello" 0xdeadc0de }))
```

The following parameter value will reach the above opcodes:

```
{ (Pair "tz1MZD3EecfFVHbteFYXZMpFnvFH1g6a2BA1" {
  (Pair "tz1ZdZDr5pLFbNLzgWQBCpm27ZgMSQPUYJN5" (Pair 0 50))
}) }
```

Recommendation: Push the string "FA2_NOT_OPERATOR" and use the `FAILWITH` opcode.

QSP-13 Privileged roles and ownership

Severity: *Informational*

Status: Fixed

Description: There is an action that could have important consequences for end-users. The option factory contract can change the option contract used by the market contract at will.

Recommendation: This centralization of power needs to be made clear to the users, especially depending on the level of privilege the contract allows to the owner.

Update: SEXP team stated that “The factory contract that produces the FA2 contract can’t be changed after it is deployed, so the FA2 contract produced is always the same. To change the FA2 contract produced, a new factory contract needs to be deployed. This means that users can be sure the behavior of the FA2 contract is going to be the same, unless a different factory contract is used. When updating the system, new factory & deployer contracts are deployed, the front-end is pointed at the new contracts and all new markets are created using the new factory & deployer contracts.”

QSP-14 Option tokens for the losing side will never be burned

Severity: *Undetermined*

Status: Acknowledged

Description: `binary_option_market.tz`: Option tokens for the losing side will never be burned.

Recommendation: Please confirm if this is intended.

Update: SEXP team decided not to change the code because “option tokens of the losing side are not burned, this was meant to save gas. Currently there is no way to recover the funds burned to pay for storage space on chain, so there is no incentive to clean up after the storage is no longer needed.”

QSP-15 Anyone can burn their option tokens

Severity: *Undetermined*

Status: Fixed

Description: `fa2_with_factory.tz`: Anyone can burn their option tokens by sending it to the market contract.

Recommendation: Please confirm if this is intended.

Code Documentation

[all fixed]

1. In `binary_option_market.tz`, the "DONE" comment block on `L75-L88` would be useful to add to a specification for the protocol, rather than included directly in the code itself.

Adherence to Best Practices

[all fixed]

1. `binary_option_market.tz`: TODO in the code: `# TODO handle missing record in GET here?? if we enforce skew limits in init, ASSERT_SOME should never fail here`.
2. `binary_option_market.tz`: `L174`: consider changing the revert message from "market not initialized" into "option contract address not initialized".
3. In `binary_option_market.tz`, there is commented code from `L256-L309` that should be removed.

Test Results

Test Suite Results

The tests failed to complete (error message as shown below) when Quantstamp tried to run them.

```
2021/07/07 09:43:13 market: KT1LP6eBRvZSF9N5rtzwR4TDyKFukeTdpewY
2021/07/07 09:43:13 options: KT1WMTz5ayyKiUNoNrrfdcAyHuLeiEbspEZ8
2021/07/07 09:43:13
2021/07/07 09:43:14
2021/07/07 09:43:14 amount: 12345600000000000000000000000000000000000000000000000000000000000
2021/07/07 09:43:14 side: False
--- FAIL: TestSelfTransfer (43.41s)
    --- FAIL: TestSelfTransfer/claim_22_long (7.55s)
        deploy_test.go:2883: test setup: bidding: contract error:
    --- FAIL: TestSelfTransfer/claim_22_short (7.89s)
        deploy_test.go:2883: test setup: bidding: contract error:
    --- FAIL: TestSelfTransfer/claim_tiny_long (6.03s)
        deploy_test.go:2883: test setup: bidding: contract error:
    --- FAIL: TestSelfTransfer/claim_tiny_short (8.00s)
        deploy_test.go:2883: test setup: bidding: contract error:
    --- FAIL: TestSelfTransfer/claim_huge_long (7.94s)
        deploy_test.go:2883: test setup: bidding: contract error:
    --- FAIL: TestSelfTransfer/claim_huge_short (5.99s)
        deploy_test.go:2883: test setup: bidding: contract error:
FAIL
exit status 1
FAIL    gitlab.com/smartcontractlabs/sexp-binary-options/market 925.035s
```

Code Coverage

Currently there is no existing tool for estimating the coverage data of the tests for Michelson language.

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

```
1680ec6cf65a759eefe9e1127b0b9670b3a6e92713c259201adda5939d68549c ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/fa12_mock.tz

a2d8b61ea5e04402a10cc3aa7636c5269e72c9fbdf8b9ff1bfd93aada32ea1a7 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/harbinger_normalizer.tz

c69c6e484191cfbf00fb53c59ed4bc9e87bcc1aeb59fe882dacaa9e099a4662c ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/deployer/binary_option_market.tz

9e3779effcd662bb18de57f6947f04add3f216a8a6ed8524e0ead224cc01f31d ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/deployer/fa2_with_factory.tz

6b2f5ba5c969748025e3ff06abcfaac9dee0500c2fc975ee9b50a359da211721 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/deployer/ledger_factory.tz

f9003568887a563a0af7d0568c1fa28d3d60f25ef716478c23ef38990fc4c88e ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/deployer/market_deployer.tz

6c9608a4fdbbdcbbf48b8c82e6af13eea7c56560c334a1f070451e84e07a9f5f ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/contracts/deployer/market_factory.tz
```

Tests

```
5720a05e77c6651f7e8253df7d682d049efb2efd0e0e02aed51482ada9d17b0b ./sexp-binary-options-1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/main.go

4f2f1c6b99ec27ab8a036796a3aba469f70ddb696c645395b52ce9dca595326f ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/watchtower/main.go

79a4345947302422b99bd54fc7854e8f8dd4ba1fa2e218e1912af95df4325a70 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/watchtower/storage.go

f95277d01df136becb5e3f3f9e92444bc523e7b2e6f48b3d59796408a6d06fd9 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/watchtower/w/commands.go

c69ba51b2da26a770e514a66517975181354fcbe3756a4e9a837904d97746e92 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/metadata/main.go

a08dbc604f27462e693a9c1926fdf3f1fe61275aeed6f85d5b17da7a378ae7f8 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/metadata/storage.go

886f7e5b5bea3a7fcf5ec0389782b615cc98dc6794101c870dea296a7e1a89cb ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/commands.go

c95c9b077ff5124f0e8b41d64ac920c4980b161078f999cae178da7828411eba ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/deploy.go

4ac0ba2610a49e2b69bcf9e2ec59d61a5f38c73478414e1b78cef2cf98a7ee8d ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/market/deploy_test.go

469937ced9fd21234c57eafa3518914c64080ab3fbe5e0a65b9fc29d3bcac5a9 ./sexp-binary-options-
1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/deployer/originateDeployer.go

2e69ee7a6ff112e16bb8b2e0c0652c3b986a18cffa9814d191295b41386d0fda ./sexp-binary-options-1ce08759bf9f5d9254ec67a61ac3cdb235998ac6/d/debug.go
```

Changelog

- 2021-06-14 - Initial report
- 2021-07-09 - final report

About Quantstamp

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

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

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

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