traceto.io Network Contracts

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

Executive Summary

Category	Description
Type	Protocol Contracts
Auditor(s)	Kacper Bąk, Senior Research Engineer Yohei Oka, Forward Deployed Engineer Nadir Akhtar, Software Auditing Intern John Bender, Senior Research Engineer
Timeline	2018-11-12 through 2018-11-30
Language(s)	Javascript, Solidity
Method(s)	Architecture Review, Unit Testing, Functional Testing, Computer-Aided Verification, Manual Review
Specification(s)	traceto.io Documentation: https://docs.traceto.io Whitepaper: https://ico.traceto.io/static/wp/traceto_Whitepaper_v1_35_en.pdf
Source code	Repository: Contract_KYC_workflow Commit: bea7bb7
Total Issues	30
High Risk Issues	1
Medium Risk Issues	4
Low Risk Issues	10
Informational Risk Issues	15
Undetermined Risk Issues	0

Severity Level	Explanation
High	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.

Severity Level	Explanation
Medium	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	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 pose an immediate threat to continued operation or usage, but is relevant for security best practices, software engineering best practices, or defensive redundancy.
Undetermined	The impact of the issue is uncertain.

Goals

This report focused on evaluating security of smart contracts, as requested by the traceto.io Network Contracts team. Specific questions to answer:

- Can users' funds get locked up in the contracts?
- Do the contracts have any privacy issues?
- Are there any issues with the centralization of power?

Changelog

- Date: 2018-11-23 - Initial report
- Date: 2018-11-30 - Final report for the first tranche

Overall Assessment

The contracts provide an implementation of TraceTo protocol. Quantstamp has found a significant number of issues with the code. The TraceTo team addressed most of them. Furthermore, we also give a set of recommendations to ensure that the code conforms to the best practices.

Quantstamp Audit Breakdown

Quantstamp's objective was to evaluate the traceto.io Network Contracts 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 below notes outline the setup and steps performed in the process of this audit.

Setup

Testing setup:

- Truffle v4.1.12
- Ganache v1.1.0
- solidity-coverage v0.5.8
- Oyente v1.2.5
- Mythril v0.2.7
- truffle-flattener v0.18.9
- MAIAN commit sha: ab387e1
- Securify

Assessment

Findings

Incorrect approval

Status: Fixed

Contract(s) affected: `TraceToServiceCredit.sol`

Severity: High

Description: In line 170, the `_sp` token allowance is added to `tracetoMetaInfo.getVerifierWL()` allowance.

Recommendation: Replace `_sp` with `tracetoMetaInfo.getVerifierWL()` allowance.

Service provider whitelist may become outdated

Status: Fixed

Contract(s) affected: `TraceToServiceCredit.sol` , `TraceToProfileResult.sol` , `TraceToMetaInfo.sol`

Severity: Medium

Description: The contracts `TraceToServiceCredit` and `TraceToProfileResult` take in the deployed `TraceToMetaInfo` address as part of the constructor and initialize contract addresses. However, these addresses may be updated in `TraceToMetaInfo` but not in the other contracts.

Exploit Scenario:

1. `TraceToMetaInfo` contract is deployed.
2. Service provider whitelist is set using `TraceToMetaInfo.setSPWL()` .
3. `TraceToProfileResult` is deployed with the `TraceToMetaInfo` address.
4. `tracetoSPList` is initialized in the constructor using `traceToMetaInfo.getSPWL()` .
5. Service provider whitelist is updated in `TraceToMetaInfo` .
6. `tracetoSPList` is not updated in `TraceToProfileResult` .

Recommendation: A possible solution is to call `traceToMetaInfo.getSPWL()` every time `tracetoSPList` is accessed.

Possible data overwrite and transaction ordering dependence

Contract(s) affected: `TraceToRequestorList.sol`

Severity: Medium

Description: The system is architected so that anybody can call the function `addPendingRequestorPR()` in `TraceToRequestorList` . Consequently, a malicious actor can overwrite the data any number of times. Depending on how the rest of the system processes the data, the malicious actor may need to exploit transaction ordering dependence to try to get their transaction mined first.

Exploit Scenario:

1. User calls `addPendingRequestorPR()` with the argument `_requestorPR` being `0x0` .
2. A malicious actor sees the transaction and calls `addPendingRequestorPR()` with `_requestorPR` being `0x1` .
3. The transaction with `_requestorPR` being `0x1` gets mined first and is accepted as a requestor.

Recommendation: The data overwrite may be prevented by checking an entry for given `_requestorPR` already exists. The transaction ordering dependence may be prevented by relying on `msg.sender` instead of an arbitrary `_requestorPR` .

Allowance Double-Spend Exploit

Status: Fixed

Contract(s) affected: `TraceToServiceCredit.sol` , `TraceToRMIServiceCredit.sol`

Severity: Medium

Description: As it presently is constructed, the function `setFinished()` is vulnerable to the allowance double-spend exploit, as with other ERC20 tokens.

Exploit Scenario:

1. Alice allows Bob to transfer `N` amount of Alice's tokens (`N>0`) by calling the `approve()` method on `Token` smart contract (passing Bob's address and `N` as method arguments)
2. After some time, Alice decides to change from `N` to `M` (`M>0`) the number of Alice's tokens Bob is allowed to transfer, so she calls the `approve()` method again, this time passing Bob's address and `M` as method arguments
3. Bob notices Alice's second transaction before it was mined and quickly sends another transaction that calls the `transferFrom()` method to transfer `N` Alice's tokens somewhere

4. If Bob's transaction will be executed before Alice's transaction, then Bob will successfully transfer `N` Alice's tokens and will gain an ability to transfer another `M` tokens
5. Before Alice notices any irregularities, Bob calls `transferFrom()` method again, this time to transfer `M` Alice's tokens.

Recommendation: The exploit (as described above) is mitigated through use of functions that increase/decrease the allowance relative to its current value, such as `increaseAllowance()` and `decreaseAllowance()`.

Pending community agreement on an ERC standard that would protect against this exploit, we recommend that developers of applications dependent on `approve()` / `transferFrom()` should keep in mind that they have to set allowance to 0 first and verify if it was used before setting the new value.

Centralization of Power

Status: Fixed

Contract(s) affected: `TraceToSPList.sol`

Severity: Medium

Description: Smart contracts will often have `owner` variables to designate the person with special privileges to make modifications to the smart contract. However, 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. For example, TraceTo Whitepaper puts an emphasis on decentralization, but the code allows the owner to assign an arbitrary reputation value to service providers via `setReputation()`.

Recommendation: Make clear which parts of the system are decentralized and which require special privileges to operate correctly.

Greedy Contract

Status: Fixed

Contract(s) affected: `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`, `TraceToProfileResult.sol`

Severity: Low

Description: A greedy contract is a contract that can receive ether which can never be redeemed. The following methods are marked as payable: `topup()` in file `TraceToServiceCredit.sol`, `topup()` in file `TraceToRMIServiceCredit.sol`, and `addPending()`, `addRMIPending()`, `requestProfileKey()`, `setFinished()`, `setRMIFinished()`, `setResult()`, `setRMIResult()`, `emitRENEW()`, `emitRMI()` in file `TraceToProfileResult.sol`.

Recommendation: The keyword payable does not seem necessary for the listed functions. We recommend removing it. Alternatively, add a method for transferring out the funds.

Gas Usage / `for` Loop Concerns

Contract(s) affected: `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`

Severity: Low

Description: Gas usage is a main concern for smart contract developers and users, since high gas costs may prevent users from wanting to use the smart contract. Even worse, some gas usage issues may prevent the contract from providing services entirely. The potentially problematic for loops happen in the functions `addPending()` in files `TraceToServiceCredit.sol` and `TraceToRMIServiceCredit.sol`, and in the function `removeSP()` in file `TraceToSPList.sol`.

Furthermore, the counter in lines 130 of the contracts `TraceToServiceCredit.sol` and `TraceToRMIServiceCredit.sol` is increased by using the function `add()`. In this case it is unnecessary as integer overflow cannot occur. Similarly, `spCount` is increased and decreased in the function `approveSP()` in file `TraceToSPList.sol`.

Recommendation: Gas concerns in for loops may be mitigated by letting the caller specify how many iterations they want to execute. Also, it is best to break such loops into individual functions as possible.

As for `spCount` getting increased and decreased in the function `approveSP()`, we recommend removing these operations as they have no impact on the executed code (unless getting an exception is intended behavior).

The function `removeSP()` may be simplified by eliminating loop. Instead of reassigning elements, the contract may delete an element and replace it with the last element.

Redundant event

Status: Fixed

Contract(s) affected: `Whitelist.sol`

Severity: Low

Description: The function `addAddressToWhitelist()` does not check if an entry is already present. Consequently, it may emit an event although no new operator is whitelisted. Similarly, the function `removeAddressFromWhitelist()` may emit an event if no address is removed from the whitelist.

Recommendation: Check if an entry is already present in the map.

Check if address is in `metaInfo[_requestorPR]`

Contract(s) affected: `TraceToRequestorList.sol`

Severity: Low

Description: The line 87 does not check the value of `metaInfo[_requestorPR]`. As `addAddressToWhitelist()` can be called independently from `approveRequestorPR()`, it may result in an address being in the whitelist but not in `metaInfo`.

Recommendation: Check that `metaInfo[_requestorPR]` value is set.

Check if address is in `metaInfo[_sp]`

Contract(s) affected: `TraceToSPList.sol`

Severity: Low

Description: The line 147 does not check the value of `metaInfo[_sp]`. As `addAddressToWhitelist()` can be called independently from `approveSP()`, it may result in an address being in the whitelist but not in `metaInfo`.

Recommendation: Check that `metaInfo[_sp]` value is set.

The expiry date may be reset

Status: Fixed

Contract(s) affected: `TraceToProfileResult.sol`

Severity: Low

Description: In lines 159 and 181, it seems that the intention is to only allow resetting an expiry date if the new date is earlier than the currently set date, or if it has not been initialized yet. However this check can be bypassed by setting to zero first.

Exploit Scenario:

1. Call `setResult()` with `_expire` corresponding to year 2020.
2. Call `setResult()` with `_expire` being `0`
3. Call `setResult()` with `_expire` corresponding to year 2030.

4. `profileInfo[_profile].expire` can now be set to anything, even a date larger than the original 2020.

Recommendation: Prevent resetting `_expire` to `0`.

`removeSP()` may unintentionally delete the first element of the `spList` array

Contract(s) affected: `TraceToSPList.sol`

Severity: Low

Description: `removeSP()` may unintentionally delete the first address in the list if it is called with `_sp` for which `metaInfo[_sp]` is the default value.

Recommendation: Check that `metaInfo[_sp]` is not the default values (was previously approved).

Integer Overflow / Underflow

Contract(s) affected: `TraceToSPList.sol`

Severity: Low

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. The lines 115 and 118 in the function `getSPList()` are vulnerable to integer overflow.

Recommendation: Use the `add()` function from SafeMath to check in line 115 that the addition can be performed.

`setReview()` calls `tracetoMetaInfo.getSPWL()`

Contract(s) affected: `TraceToRMIServiceCredit.sol`

Severity: Low

Description: The function `setReview()` is not overridden, and, consequently, it calls `tracetoMetaInfo.getSPWL()` instead of `tracetoMetaInfo.getRMISPWL()`.

Recommendation: Override `setReview()` and call `tracetoMetaInfo.getRMISPWL()`.

Requestor may not pay the service provider

Contract(s) affected: `TraceToServiceCredit.sol`

Severity: Low

Description: The function `setFinished()` is normally called by the requestor to pay the service provider for their work. It is possible, however, that the requestor never calls the function, and the service provider does not get paid.

Recommendation: We recommend providing an expiry window when `setFinished()` shall be called and a method of disputing the transaction. Alternatively, clearly inform users about the current issue.

Clone-and-Own

Status: Fixed

Contract(s) affected: `Ownable.sol`, `SafeMath.sol`, `Whitelist.sol`

Severity: Informational

Description: The codebase relies on the clone-and-own approach for code reuse. The clone-and-own approach involves copying and adjusting open source code at one's own discretion. From the development perspective, it is initially beneficial as it reduces the amount of effort. However, from the security perspective, it involves some risks as the code may not follow the best practices, may contain a security vulnerability, or may include intentionally or unintentionally modified upstream libraries.

Recommendation: Rather than the clone-and-own approach, a good industry practice is to use the Truffle framework for managing library dependencies. This eliminates the clone-and-own risks yet allows for following best practices, such as, using libraries.

Unlocked Pragma

Status: Fixed

Contract(s) affected: `TraceToMetaInfo.sol`, `TraceToRequestorList.sol`, `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`, `TraceToProfileResult.sol`, `TraceToSPList.sol`

Severity: Informational

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.

Misusing `require()` / `assert()` / `revert()`

Status: Fixed

Contract(s) affected: `SafeMath.sol`, `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`

Severity: Informational

Description: `require()`, `revert()`, and `assert()` all have their own specific uses and should not be switched around:

- `require()` checks that certain preconditions are true before a function is run and checks arguments.
- `revert()`, when hit, will undo all computation within the function.
- `assert()` is meant for checking that certain invariants are true. An `assert()` failure implies something that should never happen.

Recommendation: We recommend replacing the use of `assert()` with `require()`. Specifically, lines 93, 151, 162 in `TraceToServiceCredit.sol`, lines 93, 151, 152 in `TraceToRMIServiceCredit.sol`, and lines 21, 45, 57 in `SafeMath.sol`.

Ignoring the result of `transfer()`

Status: Fixed

Contract(s) affected: `TraceToMetaInfo.sol`, `TraceToRequestorList.sol`, `TraceToSPList.sol`, `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`, `TraceToProfileResult.sol`

Severity: Informational

Description: The function `emergencyERC20Drain()` ignores the result of `transfer()`. By the ERC20 standard, the method returns a boolean and does not necessarily revert the transaction upon failure (See: EIP20).

Recommendation: We recommend wrapping `transfer()` in a `require()` statement to make failed transfers easier to spot. Alternatively, return the value of `transfer()` as the return value of `emergencyERC20Drain()`.

Code clones of `emergencyERC20Drain()`

Status: Fixed

Contract(s) affected: `TraceToMetaInfo.sol`, `TraceToRequestorList.sol`, `TraceToSPList.sol`, `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`, `TraceToProfileResult.sol`

Severity: Informational

Description: The function `emergencyERC20Drain()` appears to be cloned among multiple contracts.

Recommendation: We recommend extracting the function `emergencyERC20Drain()` into a separate contract and reusing it via inheritance.

Code clones in `TraceToRMIServiceCredit.sol` and `TraceToServiceCredit.sol`

Status: Fixed

Contract(s) affected: `TraceToRMIServiceCredit.sol`, `TraceToServiceCredit.sol`

Severity: Informational

Description: The contracts `TraceToRMIServiceCredit.sol` and `TraceToServiceCredit.sol` share most of the code via cloning.

Recommendation: We recommend extracting the cloned code into a separate contract and reusing it via inheritance.

Magic constants

Status: Fixed

Contract(s) affected: `TraceToProfileResult.sol`

Severity: Informational

Description: Number constants (lines 158 and 180) are present in the code without any further explanation. If the number `10413763200` is a timestamp in seconds, then it is equivalent to `2299-12-31T16:00:00+00:00`.

Recommendation: Define and document named constants and use them throughout the code.

No need to explicitly inherit from `Ownable`

Status: Fixed

Contract(s) affected: `TraceToRequestorList.sol`, `TraceToSPList.sol`

Severity: Informational

Description: The contracts inherit from `Whitelist` which already has `Ownable` as its base contract.

Recommendation: Remove the explicit `Ownable` base contract.

Potentially sensitive data stored on the public blockchain

Status: Fixed

Contract(s) affected: `TraceToRequestorList.sol`

Severity: Informational

Description: The function `addPendingRequestorPR()` stores important data on the public blockchain, such as name, country, and email. It could be a privacy concern for requestors if these are supposed to be real names.

Recommendation: Revisit the data and make a deliberate decision whether these should be stored on the blockchain.

Potentially unusable function `getSPList()`

Status: Fixed

Contract(s) affected: `TraceToSPList.sol`

Severity: Informational

Description: The function `getSPList()` can become unusable if array is too large.

Recommendation: Add the ability to fetch a part of the array by specifying start and end index.

Redundant `spCount`

Contract(s) affected: `TraceToServiceCredit.sol`, `TraceToRMIServiceCredit.sol`

Severity: Informational

Description: The field `spCount` is used to store the length of the array `sp`. It is redundant, since the length can be accessed by `sp.length`.

Recommendation: Remove the field `spCount`.

Redundant modifier `onlySP()`

Status: Fixed

Contract(s) affected: `TraceToRMIServiceCredit.sol`, `TraceToServiceCredit.sol`

Severity: Informational

Description: The modifier `onlySP()` is not used in the code.

Recommendation: Remove the modifier.

Unnecessary use of the modifier `onlyOwner()`

Contract(s) affected: `TraceToProfileResult.sol`, `TraceToRMIServiceCredit.sol`

Severity: Informational

Description: The function `getRMIServiceBalance()` uses the modifier `onlyOwner()`, yet it calls the function `getBalance()` which does not have any modifiers.

Recommendation: Remove the use of `onlyOwner()`.

The function `approveSP()` may approve empty `pendingMetaInfo[_sp]`

Contract(s) affected: `TraceToSPList.sol`

Severity: Informational

Description: The function `approveSP()` may approve empty `pendingMetaInfo[_sp]`, and, consequently, an address can be added multiple times even when not intended.

Recommendation: The function `approveSP() should check that` addPendingSP() `has been called beforehand and that` pendingMetaInfo[_sp]` does not have the default value.

Inconsistent reputation range

Contract(s) affected: `TraceToServiceCredit.sol`

Severity: Informational

Description: According to the description of the function `setReview()`, reputation should range between 0 and 100. The check in line 146 ensures that the reputation range is between 0 and 10.

Recommendation: Depending on the intent, update the function description or the code to keep them consistent with each other.

Test Results

Test Suite Results

```
Contract: TraceToMetaInfo
  ✓ has an owner (45ms)
  ✓ should be able to set contract (227ms)
  ✓ should not be able to set contract from not owner (110ms)
  ✓ should be able to set data (122ms)
  ✓ should be not able to set invalid (135ms)
  ✓ should not be able to set data from not owner (63ms)

Contract: TraceToProfileToken
  ✓ has an owner
  ✓ should be able to add a profile token by t3V (108ms)
  ✓ should be able to set expiry by sp (152ms)
  ✓ should be not able to set expiry by not sp (166ms)
  ✓ should be able to add multiple profiles by t3V (216ms)
  ✓ should be not able to add the same profiles by t3V (137ms)
  ✓ should be not able to add profiles by not t3V (148ms)
  ✓ should be able to add a kyc token by rq (178ms)
  ✓ should be not able to add a kyc token by not rq (220ms)

Contract: TraceToProfileResult
  ✓ has an owner
  ✓ should be able to get balance
  ✓ should be able to set pending (106ms)
  ✓ should be able to set result by sp (211ms)
  ✓ should be not able to set result by not sp (177ms)
  ✓ should be not able to set result with invalid time (311ms)
  ✓ should be able to set finished by rq (249ms)
  ✓ should be not able to set finished by not rq (176ms)

Contract: TraceToRMIServiceCredit
  ✓ has an owner
  ✓ should revert if not approved (52ms)
  ✓ should revert if not rmi sp (42ms)
  ✓ should be able to topup by rq if approved (100ms)
  ✓ should be not able to topup a negative count by rq if approved (67ms)
  ✓ should be able to set a profile as pending (123ms)
  ✓ should be not able to set a profile as pending if there is not enough balance
  ✓ should be able to approve tokens to rmiSP after finished (191ms)
  ✓ should be able to approve tokens more than once to rmiSP after finished (444ms)

Contract: TraceToRequestorList
  ✓ has an owner
  ✓ should be able to add a pending requestor pr contract (92ms)
  ✓ should be able to approve a pending requestor pr contract by owner (122ms)
  ✓ should be not able to approve a pending requestor pr contract by not owner (95ms)
  ✓ should be able to remove a pending requestor pr contract by owner (146ms)
  ✓ should be not able to remove a pending requestor pr contract by not owner (132ms)

Contract: TraceToSPList
  ✓ has an owner
  ✓ should be able to add a pending sp (99ms)
  ✓ should be not able to add a pending sp with negative rate (100ms)
  ✓ should be able to approve a pending sp by owner (153ms)
  ✓ should be not able to approve a pending sp by not owner (120ms)
  ✓ should be able to remove a sp by owner (178ms)
  ✓ should be not able to remove a sp by not owner (167ms)
  ✓ should be able to approve more than one sp by owner (414ms)

Contract: TraceToServiceCredit
  ✓ has an owner
  ✓ should revert if not approved (53ms)
```

```
    ✓ should revert if not sp (38ms)
    ✓ should be able to topup by rq if approved (88ms)
    ✓ should be not able to topup by rq if overflow (68ms)
    ✓ should be not able to topup a negative count by rq if approved (63ms)
    ✓ should be able to set a profile as pending (125ms)
    ✓ should be able to set a profile as pending multiple times (206ms)
    ✓ should be not able to set a profile as pending if there is not enough balance
    ✓ should be able to approve tokens to sp after finished (174ms)
    ✓ should be able to approve tokens more than once to sp after finished (441ms)

  Contract: TraceToUnlockProfile
    ✓ has an owner
    ✓ should be able to request key by rqPR (45ms)
    ✓ should be not able to request key by not rqPR (39ms)
    ✓ should be able to get keys if shared (393ms)
    ✓ should be not able to get key if there is not enough keys (76ms)
    ✓ should be able to remove duplicate keys (341ms)

  Contract: TraceToVerifierList
    ✓ has an owner
    ✓ should be able to add a pending v (81ms)
    ✓ should be able to approve a pending v by owner (120ms)
    ✓ should be not able to approve a pending v by not owner (93ms)
    ✓ should be able to remove a v by owner (148ms)
    ✓ should be not able to remove a v by not owner (130ms)
    ✓ should be able to approve more than one v by owner (456ms)
    ✓ should be able to approve other tier v by owner (560ms)


  71 passing (48s)
```

Automated Analyses

Oyente

Repository: https://github.com/melonproject/oyente

Oyente is a symbolic execution tool that analyzes the bytecode of Ethereum smart contracts. It checks if a contract features any of the predefined vulnerabilities before the contract gets deployed on the blockchain.

Oyente Findings

Oyente reported integer overflow and underflow issues in the contracts `TraceToMetaInfo`, `TraceToMetaInfo`, `TraceToProfileResult`, `TraceToRequestorList`, `TraceToRMIServiceCredit`, `TraceToServiceCredit`, and `TraceToSPList`. Upon closer inspection, we classified them as false positives.

Mythril

Repository: https://github.com/ConsenSys/mythril

Mythril is a security analysis tool for Ethereum smart contracts. It uses concolic analysis, taint analysis and control flow checking to detect a variety of security vulnerabilities.

Mythril Findings

Mythril reported the following issues:

- Integer overflow in the contracts `TraceToMetaInfo` and `TraceToRequestorList`. Upon closer inspection, we classified them as false positives.
- Message call to an address provided as a function argument in the function `emergencyERC20Drain()` which takes `_token` as an argument. Upon closer inspection, we classified it as a false positive.

MAIAN

Repository: https://github.com/MAIAN-tool/MAIAN

MAIAN is a tool for automatic detection of trace vulnerabilities in Ethereum smart contracts. It processes a contract's bytecode to build a trace of transactions to find and confirm bugs.

MAIAN Findings

MAIAN reported no issues.

Securify

Repository: https://github.com/eth-sri/securify

Securify Findings

Securify has not completed the analysis -- it timed out.

Adherence to Specification

It is unclear whether the code fully conforms to the specification since many pieces of code lack documentation that would described the intent.

Code Documentation

Although most functions are documented, there is little to no documentation regarding their logic. Furthermore, contract data fields are not well documentation.

Adherence to Best Practices

In our opinion, the code largely adheres to best practices. A multitude of recommendations provided in this report has been applied by the TraceTo team.

Appendix

File Signatures

The following SHA-256 signatures were calulated using the following comnand: `shasum -a 256 ./contract_name.sol`

Contracts

```
contracts/TraceToSPList.sol: df53a78d6bf2319c20894700db55e83b478bfe7b9389f058eaecbc7732c865fb
contracts/TraceToRMIServiceCredit.sol: f10e5889e9adda974a402262c017a96e06e7c5a623c946fcb49cdb413a1bbe4b
contracts/TraceToUnlockProfile.sol: e471f3935a9a55cad8759b463645f82e10ab850fdf9497ab9d91265e9134fe74
contracts/TraceToToken.sol: 7e878d2196324260d5cee4a7cf72b32274208752784f278dfcff17d3b2f815b9
contracts/TraceToProfileResult.sol: c910af36bcc7da845f8ac6ca6685a93c2c46755526b31d53bcc425445f921964
contracts/Migrations.sol: e04e5e2f2863413bf987eba06088b6a0ea1b6ef370375de1383be005cabb0a17
contracts/TraceToProfileToken.sol: b3f3b0a00b97a61f158892a40f3ee4cfbd8c9d0509f2d094252e7974cf649309
contracts/TraceToServiceCredit.sol: 9d7c696564915b93b56337180a6a3c82a7b7462e38eef90bfae77b762c6cfbd5
contracts/TraceToVerifierList.sol: 41d32c7a0141877266523414123 0a09caa0ce68442e52dcc1dce014d27f35ace
contracts/lib/Whitelist.sol: be5777c0fd4466fb6a91d4472b397c9e36aae5f6e8d4d26f388b4ad56094d042
contracts/lib/Withdrawable.sol: f5481780e97a72cb27d3a01c4efcd90a232edf426d37e98e853c8f1b1c06d9d9
contracts/TraceToMetaInfo.sol: f640443b1bc482a0611be04d25c787bd559f411574ca310d3960410fe2a96a4e
contracts/TraceToRequestorList.sol: 51199c84baa1f5362662559959d1ad000f42a526c0a10d28ab150f7c53077965
```

Tests

```
test/TraceToSPList.js: 3af902cd636376da0fa142ee10e61e33f43ab277e76df1cdea3ca11a6cbe16d1
test/TraceToProfileToken.js: bc91746438e14536b085511ade5accd353510a5af71ce4f94dae627c213c0112
test/TraceToServiceCredit.js: 180d98ed180eb7a27ef8e3361685db7cf2494dc7f04b851a2688182c31572ab5
test/TraceToRMIServiceCredit.js: 84c32b0359b2299d26b04360e75d42c55ec8fcfecfe6a4f699dc0207b96bf513
test/TraceToProfileResult.js: d236bf9b94744df91e100055597cf92504a82f7d3451e670379549ea0a3e4010
test/TraceToMetaInfo.js: 5b2bf47c390e96b21644bab073671d63e359ccb35b41d093c326f2759dbc55aa
test/utils.js: cb1b906b15b81cadde67f3a44cf4d265aa4c7b5d51a740c4e871c4816eb1ef39
test/TraceToUnlockProfile.js: e269584b78c2f881a08bbc31fd8bad75460677c0962628b44aa510b86f0e82f5
test/TraceToVerifierList.js: 490ed26a2fc985e13d5c3b8f36586f1081f262ca09aad8b7ba1d529a4e620ad0
test/TraceToRequestorList.js: 70fa207166c55441d0d937ed9e1b45f84868b5efa4077aa96460bc594a921fc5
```

Steps

Steps taken to run the full test suite:

- Installed Truffle: `npm install -g truffle`
- Installed Ganache: `npm install -g ganache-cli`
- Installed the solidity-coverage tool (within the project's root directory): `npm install --save-dev solidity-coverage`
- Ran the coverage tool from the project's root directory: `./node_modules/.bin/solidity-coverage`
- Flattened the source code using `truffle-flattener` to accommodate the auditing tools.
- Installed the Mythril tool from Pypi: `pip3 install mythril`
- Ran the Mythril tool on each contract: `myth -x path/to/contract`
- Installed the Oyente tool from Docker: `docker pull luongnguyen/oyente`
- Migrated files into Oyente (root directory): `docker run -v $(pwd):/tmp -it luongnguyen/oyente`
- Ran the Oyente tool on each contract: `cd /oyente/oyente && python oyente.py /tmp/path/to/contract`
- Ran the MAIAN tool on each contract: `cd maian/tool/ && python3 maian.py -s path/to/contract contract`

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.

Purpose of report

The scope of our review is limited to a review of Solidity code and only the source code we note as being within the scope of our review within this report. Cryptographic tokens are emergent technologies and carry with them high levels of technical risk and uncertainty. The Solidity language itself remains under development and is subject to unknown risks and flaws. The review does not extend to the compiler layer, or any other areas beyond Solidity that could present security risks. The report is not an endorsement or indictment of any particular project or team, and the report does not guarantee the security of any particular project. This report does not consider, and should not be interpreted as considering or having any bearing on, the potential economics of a token, token sale or any other product, service or other asset.

No third party should rely on the reports in any way, including for the purpose of making any decisions to buy or sell any token, product, service or other asset. Specifically, for the avoidance of doubt, this report does not constitute investment advice, is not intended to be relied upon as investment advice, is not an endorsement of this project or team, and it is not a guarantee as to the absolute security of the project.

Disclaimer

While Quantstamp delivers helpful but not-yet-perfect results, our contract reports should be considered as one element in a more complete security analysis. A warning in a contract report indicates a potential vulnerability, not that a vulnerability is proven to exist.

Timeliness of content

The content contained in the report is current as of the date appearing on the report and is subject to change without notice, unless indicated otherwise by QTI; however, QTI does not guarantee or warrant the accuracy, timeliness, or completeness of any report you access using the internet or other means, and assumes no obligation to update any information following publication.

Links to other websites

You may, through hypertext or other computer links, gain access to web sites operated by persons other than Quantstamp Technologies Inc. (QTI). Such hyperlinks are provided for your reference and convenience only, and are the exclusive responsibility of such web sites' owners. You agree that QTI are not responsible for the content or operation of such web sites, and that QTI shall have no liability to you or any other person or entity for the use of third-party web sites. Except as described below, a hyperlink from this web site to another web site does not imply or mean that QTI endorses the content on that web site or the operator or operations of that site. You are solely responsible for determining the extent to which you may use any content at any other web sites to which you link from the report. QTI assumes no responsibility for the use of third-party software on the website and shall have no liability whatsoever to any person or entity for the accuracy or completeness of any outcome generated by such software.

Notice of Confidentiality

This report, including the content, data, and underlying methodologies, are subject to the confidentiality and feedback provisions in your agreement with Quantstamp. These material are not to be disclosed, extracted, copied, or distributed except to the extent expressly authorized by Quantstamp.