Quantstamp: The protocol for securing smart contracts

Quantstamp is the first smart contract security-auditing protocol. We are extending Ethereum with technology that ensures the security of smart contracts. Our team is made of up of software testing experts who collectively have over 500 Google Scholar citations.

Founders

Richard Ma, Cornell ECE
Algorithmic Portfolio Manager

Steven Stewart, MCS, BA
PhD, U. Waterloo
Software verification, Database implementation

Founding Team Members

Dr. Vajih Montaghami, PhD
Formal methods

Ed Zulkoski, B.S.
PhD-candidate, U. Waterloo
SAT/SMT solvers

Leonardo Passos, PhD
Compilers and Programming Languages

Advisors

Dr. Vijay Ganesh, Assistant Professor, U. Waterloo (Ex-Stanford, MIT)

Dr. Derek Rayside, P. Eng., Associate Professor, U. Waterloo (Ex-MIT)

Evan Cheng, Director of Engineering at Facebook
ACM Software System Award for LLVM

2017-October-7 Version 3.0

The Problem

Blockchain networks are secure but smart contracts are not. In June 2016, a hacker stole $55M in Ethereum coins from the DAO due to a bug in its smart contract[1]. In July 2017, another hacker stole over $30M in Ether from crypto companies due to a one word bug in the smart contract code in the Parity multi-sig wallet[2]. Security issues like these are a serious impediment to wider adoption of the Ethereum network because they erode trust in smart contracts.

Current efforts to validate smart contracts are inadequate. Engaging security consulting companies require human experts to audit smart contracts. This process is expensive and error-prone. Also, relying on a single company requires trusting that no bad actors exist in the company. A distributed system relying on consensus among many different actors is far more secure.

Security audit processes that rely on human experts cannot keep up with the exploding growth rate of smart contract adoption. Between June 2017 and October 2017, the number of smart contracts grew from 500K[3] to 2M[4]. Within a year, we expect there to be 10M smart contracts. This will create an exponential increase in the demand for auditing. There aren't enough security experts in the world to audit all smart contracts today, and this shortage will be even more acute in the future.

The potential costs of smart contract failures will also grow. As of October 2017, about $3.2B (11M ETH) was locked in smart contracts. The number of dollars locked in smart contracts will grow exponentially as Ethereum network and smart contract adoption grows. The potential cost of smart contract vulnerabilities will grow commensurately.

Quantstamp Protocol

The Quantstamp protocol solves the smart contract security problem by creating a scalable and cost-effective system to audit all smart contracts on the Ethereum network. Over time, we expect every Ethereum smart contract to use the Quantstamp protocol to perform a security audit because security is essential.

The protocol consists of two parts:

- An automated and upgradeable software verification system that checks Solidity programs. The conflict-driven distributed SAT solver requires a large amount of

[1] https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/
[2] https://www.coindesk.com/30-million-ether-reported-stolen-parity-wallet-breach/
[3] https://web.archive.org/web/20170602184510/https://etherscan.io/accounts/c
[4] https://etherscan.io/accounts/c

computing power, but will be able to catch increasingly sophisticated attacks over time.

- An automated bounty payout system that rewards human participants for finding errors in smart contracts. The purpose of this system is to bridge the gap while moving towards the goal of full automation.

The Quantstamp protocol relies on a distributed network of participants to mitigate the effects of bad actors, provide the required computing power and provide governance. Each participant uses Quantstamp Protocol (QSP) tokens to pay for, receive, or improve upon verification services. Below are the different types of participants.

- **Contributors** receive QSP tokens as an invoice for contributing software for verifying Solidity programs. All contributed code will be open source so that the community can have confidence in its efficacy. Most Contributors will be security experts. Contributions are voted in via the governance mechanism.

- **Validators** receive QSP tokens for running the Quantstamp validation node, a specialized node in the Ethereum network. Verifiers only need to contribute computing resources and do not need security expertise.

- **Bug Finders** receive QSP tokens as a bounty for submitting bugs which break smart contracts.

- **Contract Creators** pay QSP tokens to get their smart contract verified. As the number of smart contracts grows exponentially, we expect demand from Contract Creators to grow commensurately.

- **Contract Users** will have access to results of the smart contract security audits.

- **Voters**: The governance system is a core feature of the protocol. The validation smart contract is designed to be modular and upgradeable based on token holder voting (time-locked multi-sig). This governance mechanism reduces the chance of upgrade forks and decentralizes influence of the founding team over time.

Technology Roadmap

2017	
June	● Quantstamp founded by Richard and Steven
July	● Solidity Static Analyzer prototype built days after Parity Wallet hack
August	● Released first version of whitepaper
September	● Hired Ed, Krishna, Vajih, Leo
October	● Completed Request Network semi-automated audit ● Built automated truffle test generator ● Complete 2nd semi-automated audit with another company
November	● Complete 3rd semi-automated audit with another company ● QSP token launch ● Begin university partnerships with the University of Waterloo
December	● Build the Quantstamp validation/payment smart contract on Ethereum ● Complete the 4th semi-automated audit
2018	
January	● Build the Quantstamp validation node (an augmented Ethereum node)
February	● Add analysis software v1 to the validation node that returns the proof-of-audit hash and raw output ● Complete the 5th semi-automated audit using analysis software v1
March	● Begin testing phase and improvement of crypto-economic incentives ● Implement token holder governance system for the upgradeable protocol
April	● Deploy to test network after testing and validating system ● Begin academic review of the system
May	● Hold first Quantstamp hackathon
June	● Begin work on smart contract insurance with partners
July	● Hold token holder vote for mainnet after months of testing/incentive adjustment
August	● Release mainnet v1
September	● Begin work on distributed SAT consensus with BFT for Mainnet v2
October	● Add smart contract insurance alpha product on Mainnet smart contracts

Motivation

Our team has devoted their careers to helping developers produce more reliable code, representing years of combined research and experience in the discipline of software verification. The opportunity to apply these expertise towards the next generation of the digital revolution is extremely exciting for everyone involved. There is a clear and urgent need for more secure code.

Vulnerabilities in smart contracts threaten the adoption of blockchain technology and cryptocurrencies. Currently a lot of work is being done to scale Ethereum, however we think security is equally important. Without security of smart contracts, it's hard for people to trust them with anything other than risk capital. Our vision for the future is that smart contracts will be mainstream applications used by people to make their everyday lives easier. We will help bring about this vision for smart contracts by extending Ethereum with technology that ensures the security of smart contracts.

We believe that automated security audits will help developers to deploy code that the public can trust without having to write formal specifications that contain more lines of code than the program itself. Our aim is to automate checks and property verification as much as possible. Each of these objectives should contribute to a healthier blockchain ecosystem. This solution addresses a infrastructural-level problem.

Our strategy is to create a foundational protocol that could be eventually incorporated directly into the Ethereum platform and to create a safe environment needed for the first Ethereum killer app.

The remainder of this document details why a security protocol is a necessary technological advancement, and provides a high-level architecture of the platform.

Smart Contract Improvements

How we improve smart contract infrastructure

The protocol allows automated security checks on the smart contract code, and does so in a trustless[5] manner. Our approach offers the following two core advantages.

[5] We use the word "trustless" to indicate that the process is transparent and it is not necessary to trust a third-party, and deters bad actors from compromising the audit.

1. The protocol allows end-users to directly submit programs for verification, without the possibility of a bad actor manipulating the results of an audit

Imagine a bad actor at a security auditing company that allows a multi-million dollar bug to slip through, and then takes advantage of the live deployed contract. The consensus required by the Quantstamp protocol mitigates the effects of bad actors based on the economically dominant strategy - it would be too costly to try to manipulate the results. Verified smart contracts are produced with the proof-of-audit hash, which includes the version of the security library used by the verifier and a plain-text report is released based on consensus. In the future, we plan to offer smart contract insurance in partnership with 3rd parties to further mitigate risks of using smart contracts.

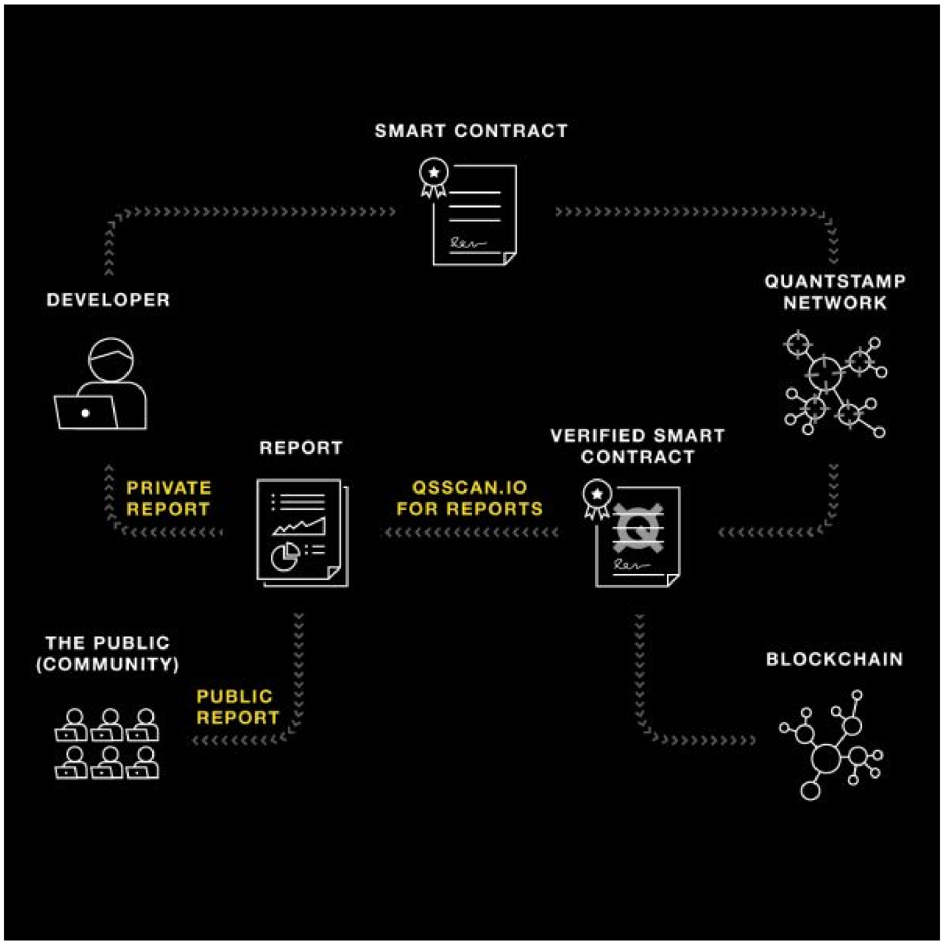
2. We incentivize miners by making the verification and certification of smart contracts part of the validation node software on Ethereum

In a blockchain architecture, "miners" are participating entities that try to add transactions to the chain. In the Quantstamp protocol, miners are called verifiers. A verifier needs to run the validation node software which watches for updates on the Quantstamp validation smart contract. The fee for performing the service makes verifiers honest. A verifier that certifies a contract produces a proof-of-audit hash and in turn, the verifier is awarded a token fee. In case a verifier finds a violation of security goals by a contract, s/he produces a counterexample that is a witness to the violation and the escrow smart contract pays a bounty fee to the verifier. Developers are responsible to address vulnerabilities when they are found, but now, they can address it before real stakes are involved.

How we improve the developer's process

Well-intentioned software developers need help to produce better code. As pointed out by Luu et al.[6], there is a semantic gap rooted in a misunderstanding of how code executes in the blockchain; consequently, there is a pressing need for better tools that can assist the developer in capturing vulnerabilities prior to deployment. The current way developers test code - manually via open source code reviews and unit tests (if they are diligent) - is not sufficient to meet the needs of blockchain technology, which ideally offers perfect security. All of the above methods are very manual methods that allow for human error. There is a need for an easy process of verifying smart contracts while minimizing the chance of serious vulnerabilities slipping through the cracks. The Quantstamp protocol provides this easy interface while also helping to protect developer reputations by proving on the blockchain that they have performed this auditing.

[6] Luu et al. describe this semantic gap in their paper "Making Smart Contracts Smarter." They propose to enhance the operational semantics of Ethereum and offer a symbolic execution tool called Oyente to find bugs in smart contracts. We pragmatically believe that very few developers, in practice, will ever utilize such tools, just as very few do in the ordinary practice of software engineering.



Quantstamp, by example

Suppose a developer plans to deploy a smart contract written in Solidity on Ethereum. There is substantial risk when writing code that accesses a monetary system, and the developer must be careful to ensure that no funds are lost due to vulnerabilities.

To minimize risk, the developer submits his code for a security audit via the Quantstamp Ethereum smart contract directly from his wallet, with the source code in the data field, and by sending QSP tokens. Depending on the security needs of the program, the developer can decide how much bounty to send. Then, the smart contract receives the request, and on the next Ethereum block validation nodes perform a set of security checks to validate the smart contract. Upon consensus, the proof-of-audit and the report data are added to the next Ethereum block along with the appropriate token payout. The report classifies issues based on a severity system from 1–10; a 1 is a minor warning, a 10 is a major vulnerability. From that point on, if a serious vulnerability is not immediately detected, the bounty remains until the specified time has elapsed. At the end of the time period, the bounty is returned to the developer who requested the audit.

When requesting an audit, the developer chooses a public or private security report. Private reports are encrypted using the public key of the smart contract and can be decrypted by the owner/developer. The developer and the public can access a web portal called qsscan.io to review any security report. The portal parses the information in the data field of the transactions via the Quantstamp smart contract, and displays it. By using the proof-of-audit hash, security reports viewed by the public exactly match the audited source code to prevent manipulation of report results.

A developer can perform security audits on a local machine prior to issuing a public audit, but may find that the computational overhead is too high. Quantstamp validator nodes are likely to have greater computational capacity in terms of memory and processing cores than the average developer's machine. In the same way, by aggregating the power of human hackers with a large bounty, the project is able to greatly surpass the coverage of a standard code review. Once the code is ready for deployment, the developer is ultimately motivated to produce a public security report in order to give users the reassurance that a decentralized security audit was performed.

When a security report identifies issues found within a smart contract, the developer can publicly annotate qsscan.io with feedback. This gives developers the power to flag false-positives in the report, and the community can validate the annotations.

Quantstamp does not guarantee flawless source code, but provides a much higher degree of assurance that the code is secure by using both automated and crowdsourcing methods. The Quantstamp team commits to continuously engage in research and development, making regular improvements to the security library. When there are new releases, developers can re-audit their smart contracts, demonstrating their commitment to securing code and increasing public confidence.
Non-developers will have more confidence in projects because they can see whether smart contract developers have audited their code, as well as which version was audited.

Technology

The technology that performs security audits is based on cutting-edge research into verification algorithms[7] and blockchain technology. The foundation is the **Validator Node** being developed by Quantstamp, a heavily modified Ethereum node containing an analytical toolkit that applies techniques from formal methods[8].

[7] Kröning & Strichman offer an algorithmic view of formal methods in *Decision Procedures* (Springer, 2008)
[8] These techniques include: static analysis, concolic testing and symbolic execution, and automated reasoning tools such as SAT and SMT. Our team has contributed to *MapleSAT*, an award-winning SAT solver.

Validation Protocol

The validation protocol for security audits rewards participants who provide compute resources for the purpose of running checks on smart contracts. These checks are run by validator nodes. The protocol ensures that the certification of smart contracts is part of the "proof-of-audit."

By introducing an Ethereum intermediary escrow/governance smart contract, the system can ensure transaction security for computation fees. If the receiving smart contract is not verified by the validator, the escrow will hold the transaction until the verification is complete. If the verification fails, the tokens are either automatically returned to the sender or held until the security violations are fixed.

The Quantstamp nodes are partners of the Ethereum network. Ethereum handles the network and transaction protocols, whereas the Quantstamp nodes handles the validation protocol for security audits and adds it to the data fields of transactions.

Design

The validation protocol handles both the distribution of computation and consensus. This protocol specifies how nodes in the network perform automated software verification and the bug bounty reward mechanism.

The core value proposition of our protocol is that it is trustless and deters bad actors from manipulating audit results. It is also upgradeable via decentralized governance through QSP tokens. This is how we design the protocol to achieve these goals.

Protocol Governance
We plan for the Quantstamp protocol to be an upgradeable protocol with a governance system controlled by the QSP token holders. The governance system controls the update of the validation smart contract and validation node. The validation smart contract is designed to be modular and upgradeable. The governance system itself will be added to the smart contract after the core features are implemented, as detailed in the development roadmap.

A time-locked multisig is used to govern upgrades. In the proposed approach, an upgrade transaction can be initiated by any member, and the more approvals the transaction has, the sooner it can be executed. A member can vote against an upgrade, which will mean that it will cancel out one of the other approved signatures. An upgrade that has been approved by all members can be executed after 1 hour. The amount of time required doubles for every 5% of members who don't vote and quadruples if they vote against.

Governance is a critical feature since validators and contributors will want to upgrade the protocol. A governance mechanism decreases the chance of upgrade forks, allows the protocol to incorporate contributor updates and ensures consistency among users. The decentralized governance feature allows the community to participate and decentralizes the influence of the founding team over time as contributors add to the community.

For example: validators would want to vote to change how workloads are distributed to increase their earning potential. Users would want to vote to incorporate higher-throughput algorithms that make the protocol faster.

Crypto-Economic Incentives
To prevent bad actors from manipulating the system, we construct an incentive system with a strategy of preventing rogue validator nodes from altering the audit results by making it too expensive to mount an attack. Verifiers are incentivized via a transaction fee in QSP tokens and handle a part of the computation. The proposed protocol requires a Byzantine fault tolerance of 2/3rds. If a 2/3rds consensus is not reached, tokens are not paid out. We reserve the right to improve this design during the testing and validation stage. The following sections will explain the fault tolerance design in more detail.

Adversarial Attack versus Distributed Computation
A single bad actor cannot manipulate the network because the other actors, driven by economic incentives, prevent the attack. To distribute our computation, each actor receives only a component of the overall verification problem. For distributed computations, we are currently considering using a t-masking quorum system, where two quorums intersect in at least $2t+1$ servers. This quorum system can handle a faulty system with at least t nodes. Since no single actor has access to the whole verification process, a bad actor is further deterred via the distribution of the computational process.

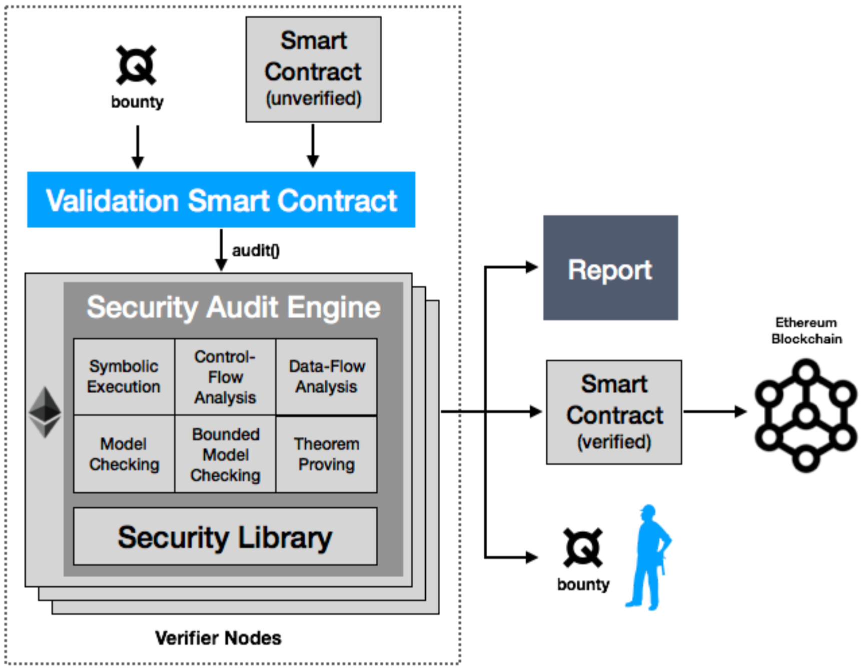
Prisoner's Dilemma
In game theory, the prisoner's dilemma is a paradox in which two individuals, acting in their own self-interest, choose a course of action that does not lead to the ideal outcome. Both choose to benefit themselves at the expense of the other, and both end up in a worse state than if they had cooperated.

Hypothetically, a verifier who finds a bug could choose *not* to take the bounty and to exploit it for future gains. Our economic incentives, however, drive verifiers to pursue the bounty instead of attempting to exploit a bug. The verifier that attempts to exploit a bug instead of reporting it has to assume that no other verifier will discover the same bug, report it, and have the bug fixed. Since the number of un- coordinated verifiers is large, it is very likely that some other verifier will find the bug and go for the bounty. Thus, a verifier who pursues actions based on self-interest is driven to claim the bounty, by design.

Security Audit Engine

The **Security Audit Engine** takes an unverified smart contract as input, performs the automated security and vulnerability checks, and produces a report. Verification results will be combined with a proof-of-audit hash with a version code used to track the scope of checks from that version of the security library.

The time taken to run the full tests in the security library scales with the complexity of the smart contract code; therefore, verification rewards are proportional to computation time. Verifiers require incentivization to motivate participation in this effort, and a token is issued for users to access its features.



The increased confidence the public gains when knowing that a smart contract was verified transparently by consensus will motivate developers to use these features. Overall confidence will be buoyed by the efforts of bounty hunters who attempt to find critical flaws.

Furthermore, as new vulnerabilities are discovered, the security library will evolve and new versions will be released. Users will be then be motivated to re-verify their smart contracts using the latest version of the security library, ensuring that their code is not open to attack due to newly discovered vulnerabilities. This is similar to how users of software can download patches to fix security vulnerabilities, or how users can update their antivirus application.

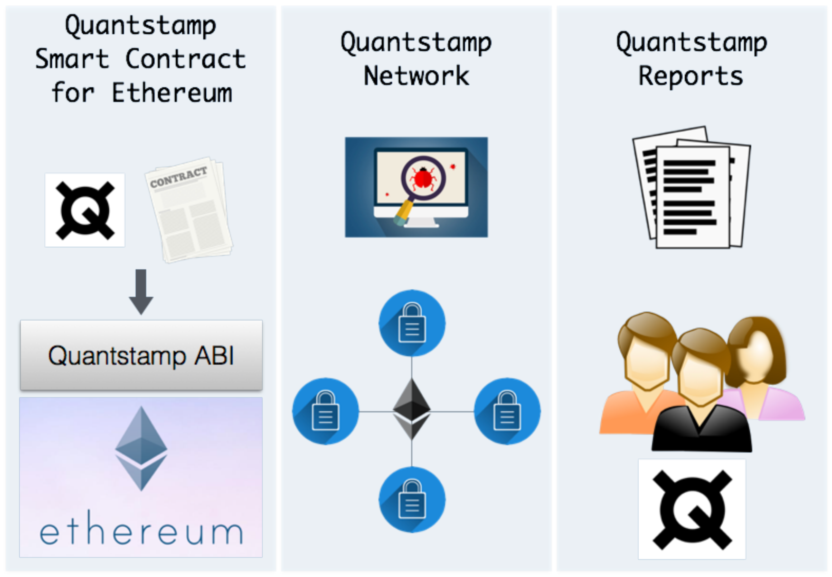
Architectural View

The Quantstamp protocol (QSP) is a scalable system to audit all smart contract projects on Ethereum. Our vision for the Quantstamp security protocol is that it will become part of the Ethereum protocol.

The QSP is split into three conceptual categories:

1. Quantstamp Validator Smart Contract for Ethereum
2. Quantstamp Network (QN) composed of heavily-modified Ethereum nodes
3. Quantstamp Reports (data carried by the smart contract transactions)

The QN is a network of verifier nodes that generates security reports by consensus. As a utility, the QSP is **platform-agnostic** - there can be many variations of the security library, one of which includes Solidity (for Ethereum), and variants that may cover other smart contract languages for different platforms.



Quantstamp Validation Smart Contract for Ethereum

The following list of functions are accessible to the end-user.

register()
Users can register an Ethereum address, which alerts the Quantstamp Network to monitor API calls of the registered address.

audit()

A user can submit source code for a security audit along with a token bounty fee. Upon success, the smart contract is digitally signed to prove that it passes critical security checks. At this point, an encrypted security report is made available. The bug bounty remains on the contract to incentivize bug finders until the specified time limit runs out.

upgrade()
Upgrade an existing smart contract. The new version of the smart contract must pass a security audit. Existing bounties are rolled forward.

Quantstamp Network for Ethereum

The Quantstamp Network (QN) is a specialized protocol capable of monitoring transactions related to a registered smart contract involving calls to the Quantstamp validation smart contract.

Quantstamp Reports

Quantstamp Reports provide a public view of the security audits performed by the QN. These reports will be made visible via a web-based user-interface at *qsscan.io*.

Reports can be public or private. Public reports are visible to everyone in a human-readable form. Private reports are encrypted using the public key of the registered user. Only the registered user can read the contents of the report. Once a smart contract is deployed, the final security report of the smart contract is public. This allows investors and other users to review the report before committing their funds.

Smart contract owners are encouraged to annotate the security report. Owners are encouraged to indicate a response to all issued security warnings and flagged issues. The response may be as simple as “this is a false positive” or “we are not concerned about this issue,” or may be highly detailed. The onus is on the owner to provide as much information as possible to anyone who may want to read the security report in order to increase the level of trust. A “trust score” will be computed for each smart contract based upon a combination of the findings in the security report, the size of the bounty, the length of time the bounty has been active and feedback from the community.

Tradecraft

In real world practice, peer reviewing and unit testing are the major software verification techniques in use. While peer review is an effective approach, it is still prone to human error, and manual testing is always limited in coverage and scope. Software verification using automated reasoning tools can help close the gap. Although research into automated reasoning tools started several decades ago, their practical importance has progressed rapidly in the last few years.

The Security Audit Engine builds upon a *tradecraft* of tools and techniques founded upon the study of discrete mathematics, logic and computer science. It interacts with the security library, which provides access to security checks (to be performed) and properties (to be verified).

We summarize the tradecraft that supports the Security Audit Engine below.

Computer-aided reasoning tools

Computer-aided reasoning tools, such as SAT/SMT solvers (below), have had a dramatic impact on software engineering and security in recent years. The key reason for the adoption of solvers in software engineering is the continuous improvement in their performance and expressive power.

SAT solvers

SAT (satisfiability) solvers support software verification tools. Computer programs are modeled as Boolean formulas, which are passed to the solver. When modeling program behaviour and testing for particular conditions, a Boolean formula can be constructed such that the existence of a satisfying assignment signifies the presence of a bug. A SAT solver reports “satisfiable” if it can find a solution or, if none exists, reports “unsatisfiable.”

SAT-solvers are important tools in several areas of software engineering, including software verification, program analysis, program synthesis and automatic testing. Additional applications span a variety of problem domains that include electronic design automation, computer-aided design and others. SAT-solvers are surprisingly efficient, combining decision heuristics, deductive reasoning and various experimentally validated techniques.

SMT solvers

An SMT solver is a tool that decides satisfiability of formulas in combination of various first-order theories. It is a generalization of a SAT solver and can handle richer theories than propositional logic. Common first-order theories, which can model fragments of computer code for vulnerability analysis, include equality, bit vectors, arrays, rationals, integers, and difference logic. This is a very active research area, and there are many applications: software verification, programming languages, test case generation, planning and scheduling, and more. Well known SMT solvers include Yices (SRI), Z3 (Microsoft), CVC3 (NYU, Iowa), STP (Stanford), MathSAT (U. Trento, Italy), Barcelogic (Catalonia, Spain).

Model-checking

Model checking is based on abstracting on the behavior of code in an unambiguous manner, which often leads to the discovery of inconsistencies. This technique explores all possible system states in a brute-force manner.

In contrast to model-checking, *bounded* model-checking (BMC) is a technique for verifying that a given property (typically expressed as an assertion by a user) holds for a program in the number of loop iterations and recursive calls bounded by a given number *k*, placing a bound on the size of the execution path for finding a bug. This problem can be reduced to solving the Boolean satisfiability problem using SAT-solvers.

The utility of bounded model-checking is in part supported by the *small-scope hypothesis*. This hypothesis states that most bugs have small counterexamples, and has proven to be an effective idea for finding bugs in software models. This hypothesis is the basis for so-called *lightweight* formal methods.

Static program analysis

Static analysis determines properties of a program without actually executing the program. Automated tools can assist programmers and developers in carrying out static analysis. Static analysis has been used to find potential null pointer bugs and to verify that device drivers always respect API usage requirements.

Symbolic execution and Concolic Testing

Concolic testing is a hybrid software verification technique that performs symbolic execution, a classical technique that treats program variables as symbolic variables along a concrete execution path. Symbolic execution is used with an automated theorem prover to generate new test cases. Its main focus is finding bugs rather than proving correctness.

Incremental releases and the subscription model

Software releases for the Security Library will have critical, major and minor update version tags. When developers deploy code, they have the ability to flag the contract for re-verification upon each critical/major/minor release on a subscription payment model. For very financially sensitive contracts, developers can choose re-verification on all releases. For less sensitive contracts, they can choose re-verification only on critical releases. When developers flag the contract for verification, and a subsequent verification fails, they will be notified by the network and can take immediate action.

The market price of the token transaction fee is an essential component of the platform that will balance computational resource supply and recurring demand. Because the market price of the token is free-floating, decentralized verifier nodes are incentivized by market forces to dynamically bring on additional resources to meet demand.

A developer can choose to not subscribe because they are confident in their application and do not want to pay subscription fees, but have a critical vulnerability in the code that is only uncovered at a later date by a new release. There is a possibility that vulnerabilities may be discovered at a later date in contracts that have already been deployed to the network with an earlier version of the Security Library.

Bug Finders

In open source software, developers are often unrewarded for finding bugs. Recently[9], Emin Gün Sirer found two critical vulnerabilities in BitGo while on vacation, and wrote a friendly email to alert them. In a common experience among security developers, he received a thankless reply and later was actually snubbed by the BitGo CTO. The automated bounty reward payout of QSP tokens will allow skilled developers to submit bugs to the validator contracts and earn immediate rewards and public recognition without all the back-and-forth with companies.

Bounties in QSP tokens are submitted when the source code is sent to the Quantstamp validator smart contract and then held in escrow. Bug finders can use any means at their disposal to break the code, and if a smart contract is found to have major vulnerabilities, then the verifier is awarded the bug bounty that was held in escrow. Validator nodes have run validation software that can verify the submitted bug.

We believe that it will be possible for skilled developers to earn an income purely via bug finding, by manually searching for security flaws in public smart contracts on our platform. Financially sensitive contracts worth millions of dollars, should in theory have bouty contracts worth at least tens of thousands of dollars before being deployed live. This will increase the security of our platform and also incentivize more security experts to spend time in the ecosystem and develop their skills.

Security Disclosure Strategy

Attackers might choose to leverage the security library as a tool for finding vulnerabilities in existing smart contracts. Any detected vulnerabilities could then be used as a starting point for planning an attack. It is not our intention to facilitate the efforts of attackers, no matter how unlikely it is that they would succeed.

In theory, this unfortunate scenario could be avoided from the start if all deployed smart contracts were pre-audited by the QSP without ever providing attackers with access to the complete security library. For this reason, we will take the following actions:

1. We will implement a staging period during the library release process, during which time we will generate encrypted security reports that smart contract owners can access.
2. We will publish public statistics indicating the frequency with which critical issues are present in smart contracts with the hope of motivating smart contract owners to read security reports and take appropriate actions.

[9] http://hackingdistributed.com/2017/07/20/parity-wallet-not-alone/

3. To avoid giving clues to would-be attackers, we will ensure that the existence of a report will not be indicative of the existence of a vulnerability, nor will characteristics of the encrypted report, such as its size, offer any reliable clues.

Whenever a new version of the security library is released, there may be a window of time in which previously audited smart contracts have newly detectable vulnerabilities. This again, could give an attacker the opportunity to use the security library as a starting point for planning an attack, even if that window of opportunity is relatively small. This is a secondary purpose of the independent verifier system - by leveraging human intelligence with bounties, we can bridge the gap between inadequate automated checking and the converse - sophisticated automated attacks.

Distributed and Parallel SAT

Software verification offers many benefits: better code, better testing, less hacks, and is an effective way to improve software security. The SAT Solver is an important tool in this effort. In this section, we offer a cursory discussion of SAT and Parallel SAT.

The Quantstamp Network offers a fascinating and exciting opportunity for the domain of SAT. Quantstamp is building a new kind of distributed SAT solver where consensus and redundancy are built-in, and participants are incentivized to solve all varieties of SAT problems in their quest to claim tokens and certify contracts. The application of this technology to smart contracts is particularly exciting because there is so much at stake.

The Satisfiability Problem (SAT)

A problem instance of SAT consists of a Boolean formula f in n variables. A SAT-solver determines the existence or non-existence of a satisfying variable assignment; in other words, an assignment of either *true* or *false* to each variable such that the formula itself is *true*. Most solvers require that f be specified in conjunctive normal form (CNF), wherein the formula consists of a conjunction of clauses, each consisting of a disjunction of literals[10].

The typical SAT-solver[11] engages in the following three step workflow from high-level encoding of the problem through the actual solving procedure.

1. Encoder
    a. Encodes the problem as a Boolean formula in conjunctive normal form (CNF) such that a satisfying assignment indicates a property violation

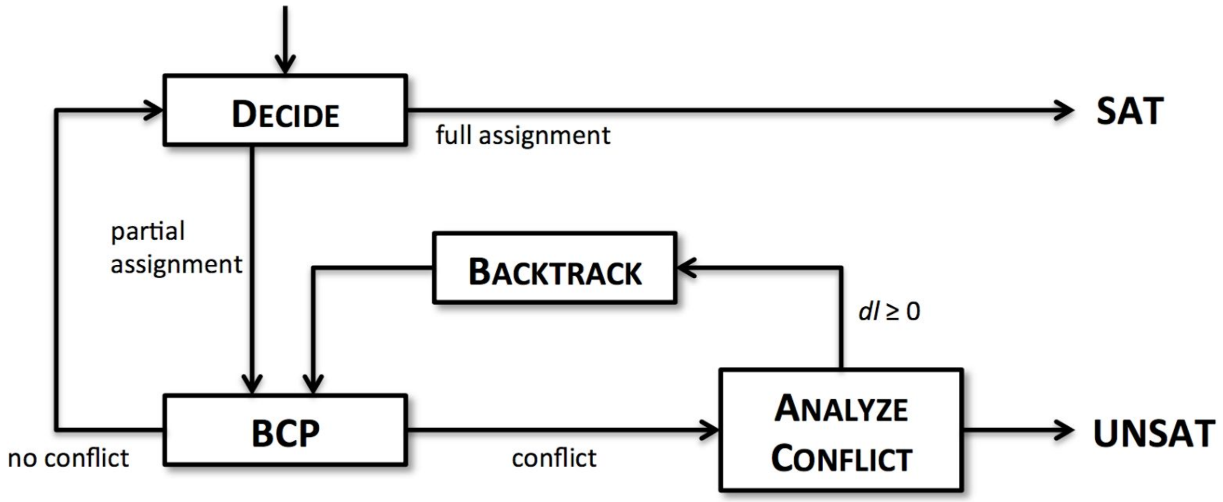
[10] The conversion of an arbitrary Boolean formula to CNF can be carried out in linear time with respect to the number of formula variables using Tseitin's translation
[11] We draw upon Norbert Manthey's excellent PhD dissertation, "Towards Next Generation Sequential and Parallel SAT Solvers" found here: http://www.cs.sfu.ca/~mitchell/cmpt-827/2015-Fall/Projects/Parallel-Manthey-PhD.pdf

    b. Usually polynomial time complexity
    c. Requires less than 1% of total time
2. Preprocessor
    a. Often performs simplification and re-encoding
    b. Polynomial time complexity
    c. Requires about 20% of the total time
3. Solving procedure
    a. Conflict-driven clause learning (CDCL)
    b. Variable ordering and other heuristics
    c. **Exponential time complexity** in the size of the input
    d. Requires about 80% of the total time

The solving procedure is the most challenging, requiring 80% of the computational effort and whose time complexity is exponential in the worst case.

Many strategies have been developed for solving SAT formulas, but the most widely adopted and successful solvers are based on the Davis-Putnam-Logemann-Loveland (DPLL) algorithm. When combined with clause learning and clever implementation tricks, DPLL- SAT has enabled the practical use of SAT-solvers for a wide-range of applications, reflecting the importance of SAT as a central problem in computer science.

An overview of DPLL-SAT[12] is depicted below.



The Decide module carries out decisions, guided by VSIDS (a variable-ordering heuristic). The BCP module carries out unit propagation until either no new unit facts can be derived, or a conflicting (unsat) clause is identified. Such clauses are handed over to Analyze Conflict, which traces the reason for the clause becoming unsatisfied, and generates a “learnt” clause that is added to the clause database. Learnt clauses prevent the propagation of assignments that lead to

[12] Kröning & Strichman offer an architectural view of SAT in *Decision Procedures* (Springer, 2008)

the conflict. The Backtrack module rewinds the search to an earlier state, undoing the assignments that lead to the conflict. The formula is said to be UNSAT upon identifying a top-level conflict clause, or SAT when no new decisions can be made and all clauses are satisfied by the current set of assignments.

Parallel SAT Solvers

Techniques such as model checking and automated theorem proving, which typically rely on SAT-solvers, may require anywhere from milliseconds to hours of computing effort on commodity machines. For some of the hardest problems, solving time can extend to days, weeks, or longer. Recent advancements in algorithms, heuristics, and parallel solvers are helping. Solvers who share the workload can outperform those who don't. Parallel SAT solvers attempt to use more cores to overcome sequential slowdowns.

A typical parallel SAT-solver use a master-slave (or task farm) approach, splitting the search space and analyzing the subspaces in parallel in separate processes. A prime example of this approach is Parallel MiniSAT[13] (PMSAT). PMSAT is a distributed parallel SAT-solver, implemented in C++ using the Message Passing Interface (MPI) for communication between nodes. PMSAT is novel in the following ways: (1) how it partitions the search space in terms of variable selection and assumptions generation; (2) how assumptions are pruned; (3) how learnt clauses are shared; and (4) automatic settings. A master controls the scheduling of the clients and distributes various tasks between them. More than one partitioning heuristic is available to the user, and sharing of learnt clauses is allowed. Conflict-learning is used to prune the outstanding tasks and potentially to stop processes whose search space is irrelevant. Two choices are available for variable selection: (1) frequent variables, or (2) variables that appear in bigger clauses.

The task farm approach is used with a master and several workers. A worker receives a set of assumptions from the master, and returns the result of searching its subtree. The master partitions the workspace according to a configured mode of operation. When a worker finds UNSAT, it may send a vector of learnt clauses and/or a vector of conflicts, the latter of which is used by the master to remove all untested assumptions that will lead to UNSAT. After receiving UNSAT, the master sends another set of assumptions, possibly with learnt clauses, to the idle worker. Upon receiving SAT, the master ends execution. The vector of conflicts is sent directly to the master in the result message, with an array size of 20 (multiple messages can be sent, if needed). Typically, more assumptions are created than there are available workers, which accounts for workers that end early so that they can get busy right away.

Another kind of parallel SAT-solver adopts a portfolio approach; *i.e.*, it relies on running multiple solvers on the same SAT instance in parallel. This technique is the state-of-the-art in

[13] Luis Gil, Paulo Flores, and Luis Miguel Silveira. *PMSAT: a parallel version of MiniSAT*. Journal on Satisfiability, Boolean Modeling and Computation, 6:71–98, 2008.

parallel SAT solvers, which is presently dominated by ManySAT[14]. With ManySAT, it has been found that using different search heuristics, or even different SAT solvers, has lead to large gains in performance. Performance gains have also been observed by sharing learnt clauses among the different solver instances.

Aigner et al.[15] discuss a plain parallel portfolio (PPP) solver that synchronizes on termination, but otherwise does not share any information. Their multi-core implementation uses shared memory, and asks the question: does memory as a shared resource become a bottleneck? If so, how much slowdown occurs? Performance degradation due to congestion of the memory system is seen as an upper bound on the expected slowdown for portfolio systems. Portfolio solvers like ManySAT and Plingeling have an architecture in which the original formula and shared clauses are copied by each solver, simplifying the design and minimizing synchronization overhead. Solvers that attempt to parallelize at a more fine-grained level do not scale as well. The drawback is that neither the formula nor learnt clauses are physically shared and thus *n* times more memory is needed, and it might be expected that there would be more memory system congestion causing slowdown; however, experiments demonstrate that most memory access are local (satisfied by core-local caches), which keeps the slowdown low even for a large number of solvers running in parallel.

Parallel SAT and consensus

As noted previously regarding Quantstamp's validation protocol, the distributed computation is partitioned into components of the overall verification problem. This an important mechanism for inhibiting bad actors. Relating this back to SAT in the distributed setting, in search space partitioning, each partition, or subspace, is solved by a sequential SAT-solver. In the Quantstamp Network, when possible, disjoint subspaces are mapped to partitions of nodes called **zones**. Within each zone, nodes work to find a satisfying assignment within a discrete subspace. Since the partitions are disjoint, identifying a satisfying assignment in one zone implies a satisfying assignment for the original formula. The encoding process of step one ensures that when a formula is satisfiable, then a bug exists in the original system. The final output of a zone requires 2/3rds consensus of the participating partition, just as a 2/3rds majority is required for consensus in general.

Common vulnerabilities for Ethereum/Solidity

The blockchain implementation of Nick Szabo's idea[16] of a smart contract is a computer program whose correct execution is enforced without relying on a trusted authority. The Ethereum

[14] Youssef Hamadi, Said Jabbour, and Lakhdar Sais. *ManySAT: a parallel SAT solver*. Journal on Satisfiability, Boolean Modeling and Computation, 6:245–262, 2008.
[15] Martin Aigner, Armin Biere, Christoph Kirsch, Aina Niemetz, and Mathias Preiner. *Analysis of portfolio style parallel SAT solving on current multi-core architectures*. In Proceeding of the Fourth International Workshop on Pragmatics of SAT (POS13). Citeseer, 2013.
[16] Formalizing and Securing Relationships on Public Networks: http://firstmonday.org/ojs/index.php/fm/article/view/548/469

protocol supports stateful contracts, meaning that the values of state variables persist across multiple invocations. A contract is invoked when it receives transactions from users at its unique address.

If such transactions are accepted by the blockchain, all participants of the mining network execute the contract code. The network then agrees, by the consensus protocol, on the output and next state of the contract. Given that Ethereum smart contracts are immutable and the effects of the transactions cannot be reversed, it is essential to be able to reason effectively about code prior to deployment.

Atzei et al.[17] describe a taxonomy of vulnerabilities and unexpected behaviours of smart contracts written in Solidity for Ethereum. Although this taxonomy is specific to Ethereum, it is likely that similar vulnerabilities will exist for other platforms that use contracts in the future. We summarize this taxonomy below based on their findings.

Call to the unknown	Some Solidity primitives have the non-obvious side effect of invoking the fallback function of the recipient. This can lead to unexpected behaviour and may be exploitable by an attacker. (We discuss this in the section on the Parity/Multisig vulnerability.)
Exception disorder	The are two different behaviours for how exceptions are handled that depend on how contracts call each other. For some, side effects of the whole transaction are reverted; for others, only the side effects of the invocation of another smart contract are reverted. These irregularities can affect the security of contracts.
Gasless send	When a user sends ether to a contract, it is possible to incur an out of gas exception.
Type casts	The compiler can do some type-checking, but there are circumstances where types are not checked which can lead to unexpected behaviour.
Reentrancy	The fallback mechanism may allow a non-recursive function to be re-entered before its termination, which could lead to loops of invocations that consume all gas. (The "DAO attack" infamously exploited this vulnerability.)
Keeping secrets	Declaring a field as private does not guarantee its secrecy because

[17] Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. *A Survey of Attacks on Ethereum Smart Contracts SoK*. In Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204, Matteo Maffei and Mark Ryan (Eds.), Vol. 10204. Springer-Verlag New York, Inc., New York, NY, USA, 164-186. DOI: https://doi.org/10.1007/978-3-662-54455-6_8

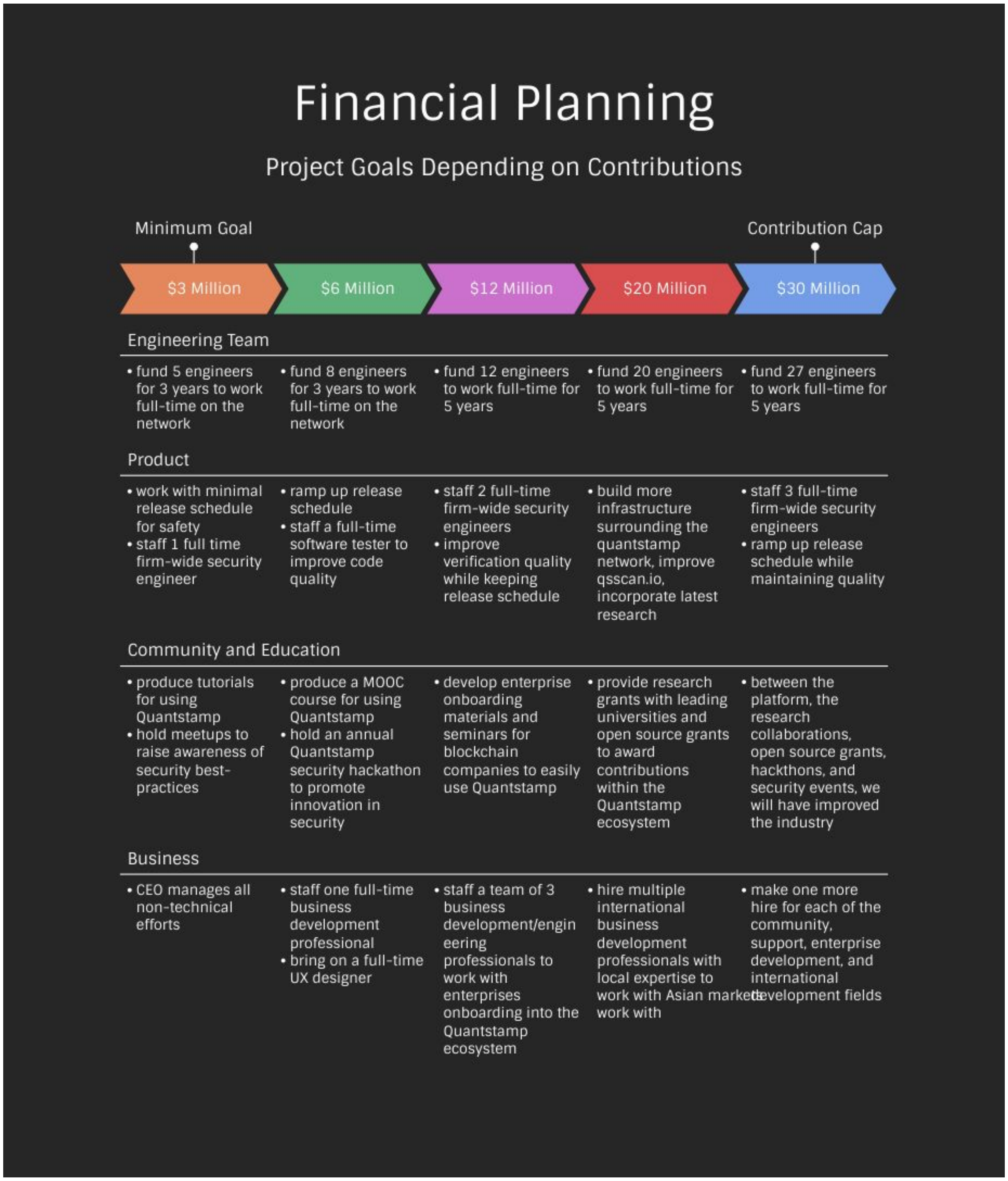
	the blockchain is public and the contents of a transaction are inspectable. Cryptographic techniques may need to be employed to protect secrets.
Immutable bugs	Deployed contracts cannot be altered, including when they have bugs, and there is no direct way to patch it. (An exception to this occurred after the DAO attack when a controversial hard fork of the blockchain nullified the effects of transactions involved in the attack.)
Ether lost in transfer	Ether sent to orphaned addresses is lost forever, and there is no way to detect when an address is an orphan.
Stack size limit	The call stack is bounded by 1024 frames and a further invocation triggers an exception. (A hard fork of the Ethereum blockchain in October 2016 has addressed this vulnerability.)
Unpredictable state	The state of a contract upon sending a transaction to the network is not guaranteed to be the state of the contract when it actually executes. Additionally, miners are not required to preserve the order of transactions when grouping them into a block. Attackers can exploit this "transaction-order dependence" vulnerability.
Generating randomness	A malicious miner can craft his block to bias the outcome of pseudo-random generator number in his favor. For example, this could be advantageous for lotteries, games, etc.
Time constraints	Many applications use time constraints to determine which actions are permitted in the current state. If a miner holds a stake on a contract, he could gain an advantage by choosing a suitable timestamp for a block he is mining.

Below are a sample of checks that would be implemented in the Security Library for Solidity.

Constant functions	The compiler does not enforce that a constant method is not modifying state; instead, this should be enforced.
Contracts that receive ether directly	Contracts that receive Ether directly need to implement a fallback function in order to receive Ether, otherwise the function throws an exception and sends back the Ether. There can be an alert when the fallback function is not implemented, since there are situations where the programmer would want to do this.
Fallback function	A contract can have exactly one fallback function, and it cannot spend more than 2300 gas. We can automatically test that the programmer is spending less than 2300 gas inside that fallback

	function.
Reentrancy exploit	When calling another contract, the called contract can change state variables of the calling contract via its functions. It's possible to check that calls to external functions happen after changes to state variables in the current contract so that it is not vulnerable to a reentrancy exploit. https://gist.github.com/chriseth/c4a53f201cd17fc3dd5f8ddea2aa3ff9
Implicit declaration	A variable declared anywhere within a function will be in scope for the entire function, regardless of where it is declared. It is also initialized to a default value for the entire scope of the function. It is possible that poorly written code can access an implicitly declared variable with a default value. When this happens, our report would generate an alert.
Transaction owner	When checking tx.origin, it gets the original address that kicked off the transaction. A malicious actor can use an attack wallet to drain all funds if the smart contract code required tx.origin == owner, since in this case tx.origin would be the address of the attack wallet.
Gas forwarding	There is an extremely dangerous feature called addr.call.value(x)() that can forward gas to a receiving contract and opens up the ability to perform more expensive actions. This is a problem that needs to be explored more in-depth later.

Financial Planning



Financial Planning

Project Goals Depending on Contributions

Minimum Goal				Contribution Cap
$3 Million	$6 Million	$12 Million	$20 Million	$30 Million
Engineering Team				
• fund 5 engineers for 3 years to work full-time on the network	• fund 8 engineers for 3 years to work full-time on the network	• fund 12 engineers to work full-time for 5 years	• fund 20 engineers to work full-time for 5 years	• fund 27 engineers to work full-time for 5 years
Product				
• work with minimal release schedule for safety • staff 1 full time firm-wide security engineer	• ramp up release schedule • staff a full-time software tester to improve code quality	• staff 2 full-time firm-wide security engineers • improve verification quality while keeping release schedule	• build more infrastructure surrounding the quantstamp network, improve qsscan.io, incorporate latest research	• staff 3 full-time firm-wide security engineers • ramp up release schedule while maintaining quality
Community and Education				
• produce tutorials for using Quantstamp • hold meetups to raise awareness of security best-practices	• produce a MOOC course for using Quantstamp • hold an annual Quantstamp security hackathon to promote innovation in security	• develop enterprise onboarding materials and seminars for blockchain companies to easily use Quantstamp	• provide research grants with leading universities and open source grants to award contributions within the Quantstamp ecosystem	• between the platform, the research collaborations, open source grants, hackthons, and security events, we will have improved the industry
Business				
• CEO manages all non-technical efforts	• staff one full-time business development professional • bring on a full-time UX designer	• staff a team of 3 business development/engineering professionals to work with enterprises onboarding into the Quantstamp ecosystem	• hire multiple international business development professionals with local expertise to work with Asian markets work with	• make one more hire for each of the community, support, enterprise development, and international development fields

Research contributions by our team

The following table comprises a partial selection of software verification projects connected to our combined research efforts. When necessary, we will adapt these proven techniques towards achieving our goal of securing smart contracts on the blockchain.

Name	**Contributors (alphabetical order)**	**Description**
Alloy and the Alloy Analyzer	Vajih Montaghami Derek Rayside Steven Stewart	Alloy is a relational logic that enables developers to model and reason about software abstractions. The Alloy Analyzer is capable of mechanically generating examples of a user's model. It was originally developed at MIT as part of the Software Design Group under the guidance of Dr. Daniel Jackson. http://alloy.mit.edu/alloy/
Bordeaux	Derek Rayside	Bordeaux is a technique and extension of Alloy for producing near-border examples, an important capability for improving debugging for identifying partial over-constraint bugs in software models. https://github.com/drayside/bordeaux
Clafer	Ed Zulkoski	Clafer is a general-purpose lightweight modeling language developed at GSD Lab, University of Waterloo and MODELS group at IT University of Copenhagen. Lightweight modeling aims at improving the understanding of the problem domain in the early stages of software development and determining the requirements with fewer defects. Clafer's goal is to make modeling more accessible to a wider range of users and domains. http://www.clafer.org/
Margaux	Derek Rayside Vajih Montaghami	Margaux is a tool for pattern-based debugging that can guide a user to find a bug. The github page includes an architectural diagram for how a debugger using discriminating examples can guide developers towards correcting flaws in logical reasoning. https://github.com/vmontagh/margaux
MapleSAT	Vijay Ganesh	The award-winning Maple series are a family of

MapleCOMSPS MapleGlucose	Ed Zulkoski	conflict-driven clause-learning SAT solvers developed at the University of Waterloo under the supervision of Dr. Vijay Ganesh. https://sites.google.com/a/gsd.uwaterloo.ca/maplesat/
MathCheck	Vijay Ganesh Ed Zulkoski	A constraint programming system that combines SAT solvers with computer-algebra systems. Extended known results on two conjectures related to hypercubes. https://sites.google.com/site/uwmathcheck/
Miramichi	Derek Rayside Steven Stewart	Miramichi is an experimental parallel SAT-solver that leverages GPUs for performance acceleration. https://bitbucket.org/sstewart2015/miramichi4j
Moolloy	Derek Rayside Steven Stewart	Moolloy is an extension to a relational logic for expressing discrete multiobjective optimization problems, with applications in science, software engineering, and finance. https://github.com/TeamAmalgam/moolloy
Petitcodiac	Derek Rayside Steven Stewart	Petitcodiac is an experimental solver for quantifier-free linear real arithmetic (LRA) that leverages OpenMP and GPUs. SMT-solvers, such as Yices and Microsoft's Z3, typically use a variation of the simplex procedure also employed by Petitcodiac. https://github.com/sstewart2012/peticodiac
STP	Vijay Ganesh	STP is a constraint solver (or SMT solver) aimed at solving constraints of bitvectors and arrays. These types of constraints can be generated by program analysis tools, theorem provers, automated bug finders, cryptographic attack tools, intelligent fuzzers, model checkers, and by many other applications. https://github.com/stp/stp

Demo: Locating The Parity Multisig Vulnerability

We provide a demonstration of a generalizable technique for automatically locating vulnerabilities similar to the Parity Multisig Wallet flaw that lead to a $32.6 million theft.

This simple analyzer constructs multiple AST (Abstract Syntax Tree) visitors and uses these to extract the program variables and call structure of a Solidity contract. The analyser finds any public method that directly or indirectly exposes a non-public state variable modification, and alerts the developer. Using call-graphs we can capture a class of vulnerabilities that can be located as solutions to reachability[18] problems. In this demo, we have two example solidity contracts to show how the analyser identifies a direct and an indirect vulnerability.

Github code for the demo: https://github.com/quantstamp/solidity-analyzer

[18] https://en.wikipedia.org/wiki/Reachability

Frequently Asked Questions

Q. What is Quantstamp?

Quantstamp is a security verification protocol for smart contracts that improves the security of Ethereum. The advantages of the security protocol include automation, trust, governance, and ability to compute hard problems over a distributed network.

Q. What is the Quantstamp team going to deliver?

The Quantstamp team will be developing the following:

1. Quantstamp validation node (a heavily modified Ethereum client)
2. The security library, containing code that performs automated checks
3. Validation smart contracts that handle bounty payment, voting mechanism and governance

A security library may also be developed to support languages other than Solidity.

Q. Aren't human security audits and code reviews the state-of-the-art?

Writing correct, bug-free software is very difficult. (Every seasoned developer eventually comes around to this conclusion.) One member of our team noted, anecdotally, that at a previous software company, the backlog of bugs was in the hundreds, and the project manager was constantly juggling a list of 20-30 features and bugs to work on in every 2-week sprint, struggling to make any significant progress. In spite of an abundance of bugs, customers expressed satisfaction about the product and mainly only reacted strongly when "show-stoppers" were uncovered. Unfortunately, once you give programmers access to a monetary system via smart contracts, just about any bug can be a show-stopper.

To improve software, most developers believe that they merely need to conduct more code reviews and write more unit tests, but the cost/benefit calculation seldom favours increased testing. Although reliance on unit testing and code reviews may be acceptable for low-risk applications, it is not acceptable when writing code for critical systems. Instead, computer chip manufacturers, airplane and automobile manufacturers, and many others rely on automated software verification to complement other best-practices. For similar reasons, our approach is to take advantage of the years of research that have developed these sophisticated techniques.

Q. Can I really trust a computer to find vulnerabilities better than a human can on his own?

While it is true that unit testing and code reviews go a long ways towards improving the quality of software, it has been shown that techniques based on formal methods are better at finding the most subtle and critical bugs that evade human inspection. This is true, in large part, because of the ability of automated reasoning tools to simulate critical execution paths in a manner that well exceeds the limitations of human cognition.

Another way to look at this is to consider what has transpired in recent years in algorithmic trading. For years, it was believed that humans were better at trading than computers, until eventually the computers took over[19]. With a quick online search for "computers have taken over Wall Street," you'll find numerous articles on this phenomenon.

Perhaps, not surprisingly, something similar is already underway with automated security audits: maybe, when we start, we cannot match an experienced human except on the cost/speed tradeoff, but with each new release the automated solution will be able to catch more and more security issues in a transparent way until eventually the algorithms will beat humans.

In the meantime, we leverage human intelligence via an automated bounty for bugs that are found by independent verifiers (white hat hackers).

Q. Why build a security auditing protocol? Instead, why not form a security consulting company?

Scalability to handle the millions of smart contracts audits on the Ethereum platform once they have resolved the Ethereum transaction scaling issues via Plasma/Casper/PoS. Empowering the first Ethereum killer app.

Q. Why not use Why3 or similar tool for formal verification instead?

Existing projects such as Why3 are too inaccessible for the typical smart contract developer to use. A similar argument can be made about the adoption of alternative programming paradigms, such as functional programming (OCaml, Haskell, Clojure), where there ends up being a lot of hype and promise but, upon closer inspection, not a lot of adoption by actual developers, who still prefer Java, C#, C++, and Python. For all these reasons and more, Quantstamp automates as much of the security auditing process as possible by embedding it into the Ethereum network with our client nodes, and relieving the developer from having to learn specialized techniques.

[19] The Quants Are Taking Over Wall Street: https://www.forbes.com/sites/nathanvardi/2016/08/17/the-quants-are-taking-over-wall-street/

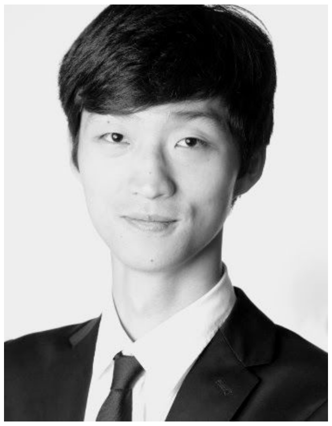
Detailed Bios

Co-founders

Steven Stewart
University of Waterloo ECE, Software Verification

Steven is a PhD candidate at the University of Waterloo (ECE) where, under Derek Rayside and Krzysztof Czarnecki, he focuses on improving the performance of software verification tools and solvers using distributed computing and GPUs.

Previously, Steven co-founded a San Francisco-based startup called Many Trees Inc that used GPUs for machine learning and Big Data analytics. In his spare time, he likes tinkering with in-memory databases accelerated using GPUs. He spent nearly 5 years as part of Canada's cryptologic agency in the Department of National Defense. Dropped out of PhD to work on Quantstamp.



Richard Ma
Cornell ECE, Algorithmic Portfolio Manager

Algorithmic Portfolio Manager at Bitcoin HFT Fund. Ex-Tower Research Capital Quant Strategist. Programmed production algorithmic trading software in C++/Python/R on competitive US, European, and Asian derivatives exchanges. Wrote tens of thousands unit tests and built production-grade integration and validation testing software. Due to Richard's extreme testing and risk-management methodology, his HFT trading systems had zero notable incidents in nearly a decade of reliably handling millions of dollars of investor capital.

Founding team members

Dr. Vajih Montaghami, PhD
Formal Methods

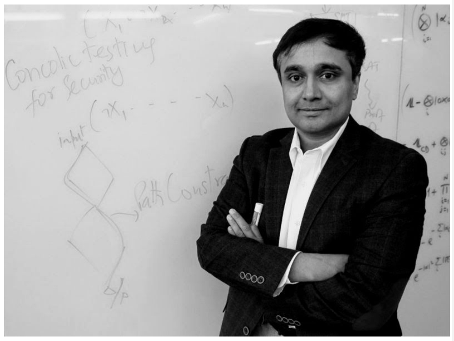
Vajih Montaghami received his PhD from the University of Waterloo for his work on verifying and debugging lightweight formal models. He focused on declarative software model formal analysis, programming language static analysis, imperative code systemization, and software architecture analysis and evaluation.

During his PhD study, he worked at Google and experienced dealing with large-scale data analysis systems. He worked on automating end-to-end testing of a machine learning algorithm applied to a massive data source. More recently, at Amazon, Vajih helped develop highly scalable systems as a backend software engineer.

Ed Zulkoski,
B.S., Mathematics and Computer Science

Edward Zulkoski is a Ph.D. candidate in the Department of Computer Science at the University of Waterloo under the supervision of Vijay Ganesh and Krzysztof Czarnecki. He recently completed an internship at Microsoft Research under the direction of Dr. Christopher Wintersteiger. His PhD research is focused on studying and exploiting the structural properties of SAT and SMT formulas. His earlier work investigated combinations of SAT solvers with computer algebra systems, and optimization techniques for multi-objective product line optimization. Ed was awarded a Ph.D. Fellowship from IBM Canada’s Centers for Advanced Studies Research.

Advisors



Dr. Vijay Ganesh,
Assistant Professor, University of Waterloo

Dr. Vijay Ganesh is an assistant professor at the University of Waterloo. Prior to that, he was a research scientist at MIT, and completed his PhD in computer science from Stanford University in 2007. Vijay's primary area of research is the theory and practice of automated reasoning aimed at software engineering, formal methods, security, and mathematics.

Vijay has won numerous awards, most recently the ACM Test of Time Award at CCS 2016, the Early Researcher Award in 2016, Outstanding Paper Award at ACSAC 2016, an IBM Research Faculty Award in 2015, two Google Research Faculty Awards in 2013 and 2011, and a Ten-Year Most Influential paper award at DATE 2008. In total, he has won 9 best paper awards/honors.

	Dr. Derek Rayside, P. Eng, Associate Professor, University of Waterloo Derek Rayside is an Associate Professor of Electrical & Computer Engineering at the University of Waterloo. His primary research areas are lightweight formal methods and program analysis. He received his doctorate in Computer Science at MIT. Derek is an advisor to a Waterloo startup that was recently acquired by Microsoft.

Addendum A

Why we should be concerned about smart contracts

There is increasing evidence that a troubling percentage, perhaps greater than 40 percent, of Ethereum smart contracts are vulnerable. It would be difficult to conclude that the remaining smart contracts are safe because they may contain as yet unidentified vulnerabilities. This is not a knock on Ethereum, as it is reasonable to assume that any platform that enables the execution of arbitrary code that accesses the monetary system is at serious risk. The onus is clearly on the developer to "get it right."

The DAO and others

Code is law. Or so they say.

On June 17, 2016, what is now referred to as The DAO has since become synonymous with perhaps one of the greatest would-be heists of modern times. To the tune of $55 million, an Ether thief discovered a bug in a smart contract that allowed repeated ATM-like withdrawals. There was no eject button, and once a smart contract is deployed, there's no turning back. To the attacker's delight, smart contracts are immutable and publically available for the unscrupulous to study and exploit.

Date	**Losses**	**Description**
June 17, 2016	$55 million	The DAO exploit[20] is perhaps the best-known. A non-recursive function could be re-entered before termination, leading to loops of invocations that consume all gas. The unhandled exception meant that repeated withdrawals were possible in the calling function.
June 20, 2017	$32.6 million	A vulnerability in Parity's multisignature wallet was exploited by hackers[21]. In this case, some Solidity primitives have the non-obvious side effect of invoking the fallback function of the recipient. This can lead to unexpected behaviour and may be exploitable by an attacker.
July 31, 2017	$1 million	There was an error in the smart contract of the REX token sale[22]. Specifically, when generating the contract bytes for deployment, a mistake was made defining the constructor

[20] https://www.multichain.com/blog/2016/06/smart-contracts-the-dao-implosion/
[21] https://www.cnbc.com/2017/07/20/32-million-worth-of-digital-currency-ether-stolen-by-hackers.html
[22] https://blog.rexmls.com/the-solution-a2eddbda1a5d

		parameters. Instead of a quoted string for an address, a Javascript hex string was used. Although this was not a theft by an attacker, it was a preventable loss.

Of course, what followed was the (in)famous and controversial Ethereum hard fork, intended to correct the apparent wrong-doing of the attacker. Perhaps, to the outsider, it's surprising that the hard fork would be controversial; after all, who could condone the actions of the world's greatest thief? But, therein lies the problem: if, in fact, code is law, then should it not be respected for how it was written? Although the developer of the smart contract undoubtedly did not intend to offer an ATM service, the code itself, as written, most certainly *did* permit this behaviour. If code is law, then the code *and* the law permitted the theft and there was no wrongdoing.

Whatever your thoughts are on the code is law question, in our view one thing is certain: never assume that a smart contract is safe. So long as code has access to a monetary system, and so long as human beings want to make money, then no code is ever truly safe. All we can really do is minimize the risk, and even better is when we can provably eliminate certain types of vulnerabilities that are well-known to be exploitable and damaging. While it is true that there does not exist any fully automated solution that can, without a shadow of a doubt, catch all possible bugs in a computer program, we can confidently state that the risk can be greatly minimized. In fact, one could argue that any bug worth finding will tend to be found, and those that are not will tend to not matter.

Still, were there only one incident -- however damaging it was -- then perhaps our worries would be out of proportion. The occasional theft could be absorbed as a kind of nuisance tax, and not necessarily perceived as a catastrophe. (Ho hum another theft. It happens.) Unfortunately, there is no such thing as bug insurance (not yet) and faulty code, when it surfaces, can indeed be catastrophic. Beyond that, it's simply impractical for there to be a hard fork whenever there is a theft.

Of course, finding a bug isn't easy. Even if the bug could self-identify, it would be difficult for an automated solution to be absolutely certain without somehow understanding the original intentions behind the code. Sometimes what looks like a bug is actually a feature! What can we do?

Our response: learn and keep learning. Identify patterns and classes of vulnerabilities. Use established techniques and improve them when necessary. Wrap this all up and make it a part of a security library whose outputs are verified by decentralized consensus. Incentivize contributors, and harness both the power of white and black hat hackers to assist in the effort. Reward them when they succeed. Keep developers accountable.

Recent studies

The full extent to which security vulnerabilities plague smart contracts is unknown; however, recent studies make it abundantly clear that there really is a *plague*. Below, we summarize a short selection of research papers that characterize some of the most serious vulnerabilities, some of which highlight just how easy it is for developers to unknowingly make mistakes.

Making Smart Contracts Smarter Loi Luu, Duc-Hiep Chu, Hrishi Olickel, Prateek Saxena, and Aquinas Hobor. 2016. *Making Smart Contracts Smarter*. In Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). ACM, New York, NY, USA, 254-269. DOI: https://doi.org/10.1145/2976749.2978309	Both malicious miners and users can exploit certain classes of vulnerabilities that the authors deem to be due to a "semantic gap" between how the developer thinks code executes versus how it actually does. In their study, 8,519 out of 19,366 (44%) Ethereum smart contracts contained "semantic gap" vulnerabilities, involving a total balance of over 6 million ETH[23].
Formal verification of smart contracts: Short Paper Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet, Anitha Gollamudi, Georges Gonthier, Nadim Kobeissi, Natalia Kulatova, Aseem Rastogi, Thomas Sibut-Pinote, Nikhil Swamy, and Santiago Zanella-Béguelin. 2016. *Formal Verification of Smart Contracts: Short Paper*. In Proceedings of the 2016 ACM Workshop on Programming Languages and Analysis for Security (PLAS '16). ACM, New York, NY, USA, 91-96. DOI: https://doi.org/10.1145/2993600.2993611	The authors translate Solidity to F* to analyze EVM bytecode. They perform checks to capture whether the code undoes side effects that can persist when a call to send() fails, and also to detect the reentrancy problem that plagued The DAO. The limitations of their tool restrict analysis to only 46 smart contracts; however, the authors state that of those only a handful passed their checks, suggesting that "a large-scale analysis of published contracts would likely uncover widespread vulnerabilities."
Demystifying Incentives in the Consensus Computer Loi Luu, Jason Teutsch, Raghav Kulkarni, and Prateek Saxena. 2015. *Demystifying Incentives in the Consensus Computer*. In Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). ACM, New York, NY, USA, 706-719. DOI: https://doi.org/10.1145/2810103.2813659	The authors show that Turing-complete scripting exposes miners to a new class of attacks: "Honest miners are vulnerable to attacks in cryptocurrencies where verifying transactions per block requires significant computational resources." To address this problem, they propose an incentive structure to the consensus protocol where cheating provides no intrinsic advantage.
A survey of attacks on Ethereum smart	The authors present a taxonomy of security

[23] To be precise, the value of 6,169,802 ETH on 2017-July-23 is about $1.4 billion USD.

contracts Nicola Atzei, Massimo Bartoletti, and Tiziana Cimoli. 2017. *A Survey of Attacks on Ethereum Smart Contracts*. In Proceedings of the 6th International Conference on Principles of Security and Trust - Volume 10204, Matteo Maffei and Mark Ryan (Eds.), Vol. 10204. Springer-Verlag New York, Inc., New York, NY, USA, 164-186. DOI: https://doi.org/10.1007/978-3-662-54455-6_8	vulnerabilities observed across the corpus of Ethereum smart contracts. In general, these vulnerabilities emerge due to subtleties of Solidity that are unknown or misunderstood by developers.
Step by step towards creating a safe smart contract D. Delmolino et al. *Step by step towards creating a safe smart contract: Lessons and insights from a cryptocurrency lab*. Cryptology ePrint Archive, Report 2015/460, 2015. http://eprint.iacr.org/	The authors show how even a very simple contract for playing Rock, Paper, Scissors can contain several logical flaws. These are characterized as *Contracts that do not refund*, *Lack of cryptography to achieve fairness*, *Incentive misalignment*.
Safer smart contracts through type-driven development J. Pettersson and R. Edström. *Safer smart contracts through type-driven development: Using dependent and polymorphic types for safer development of smart contracts*. Masters Thesis in Computer Science, Chalmers University of Technology of Gothenburg, Sweden, 2016.	Three classes of errors are highlighted that are common in smart contracts: *unexpected states*, *failure to use cryptography*, and *full call stack*. The authors propose using dependent and polymorphic types and a functional language called Idris to make smart contract development safer.

While the above papers are only a sample, a noteworthy percentage of smart contracts reportedly have known vulnerabilities. Our perspective is that it is possible to prevent many of these by performing automated checks and formally verifying expected properties. While it is likely that some attackers will focus their efforts on high profile, opportunistic heists of large magnitude, many others will be content with multiple smaller grabs less likely to garner much attention. Everybody is at risk.

Addendum B

Off-chain Tools for Developers

In addition to the decentralized security platform, we are interested in developing a set of off-chain tools aimed at simplifying the development, debugging, and deployment of smart contracts. This includes the application of recent work by one of our team members into creating smarter debugging tools.

Smart Debugging using discriminating examples

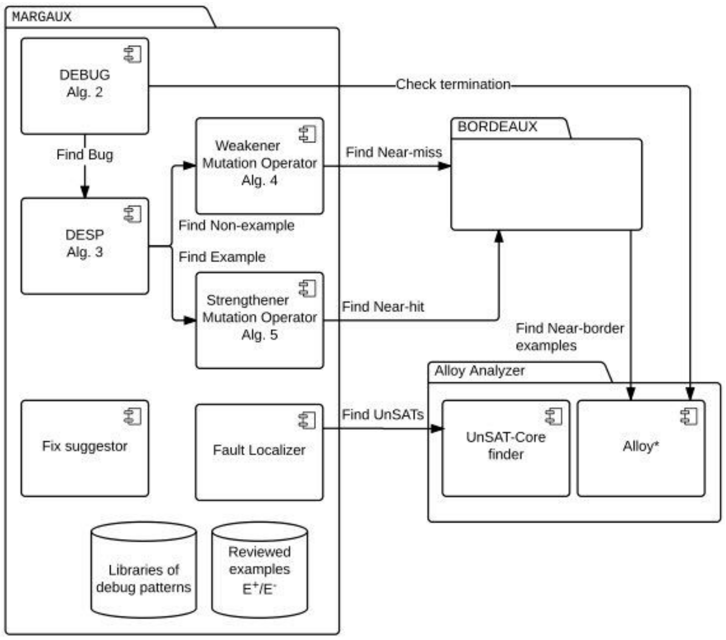
Software models with mathematical or logical foundations have proven valuable to software engineering practice by enabling software engineers to focus on essential abstractions, while eliding less important details of their software design. Like any human-created artifact, a model might have imperfections at certain stages of the design process: it might have internal inconsistencies, or it might not properly express the engineer's design intentions.

We introduce the idea of a smart debugger that helps a non-expert developer to find flaws and vulnerabilities based on the proven localization, understanding, and fix strategy. This work is explored in depth in the dissertation *Debugging Relational Declarative Models with Discriminating Examples* by founding team member Vajih Montaghami and PhD supervisor Dr. Derek Rayside (University of Waterloo).

The need to debug arises because the *expressed* meaning differs from the *intended* meaning, but the user does not know where or why. Debugging can be a cumbersome and time-consuming task that persists throughout the software lifecycle. Zeller[24], in his seminal book on debugging imperative programs, evokes an inspiring image: *Some people are true debugging gurus. They look at the code and point their finger at the screen and tell you: "Did you try X?" You try X and voila!, the failure is gone.* What has the debugging guru done? They have identified, localized, and corrected the bug[25], and they have done this by first forming a hypothesis.

Recently, tools and techniques have been developed to provide some automated support for this vision in the context of relational logic models for software abstractions. Two such tools are called Bordeaux and Margaux (depicted in the architectural diagram below). These tools first help the user identify and understand the bug by forming a hypothesis about what might be wrong with the model and computing a discriminating example for the user to accept or reject. If the user judges that a bug has been identified, then further automated analysis helps localize

[24] A. Zeller. Why programs fail: a guide to systematic debugging. Morgan Kaufmann, 2009.
[25] A. J. Ko, R. Abraham, L. Beckwith, A. Blackwell, M. Burnett, M. Erwig, C. Scaffidi, J. Lawrance, H. Lieberman, B. Myers, M. B. Rosson, G. Rothermel, M. Shaw, and S. Wiedenbeck. The state of the art in end-user software engineering. ACM Computing Surveys, 43(3):21:1–21:44, Apr. 2011., and J. F. Krems. Expert strategies in debugging: experimental results and a computational model. In Cognition and Computer Programming, pages 241–254. Ablex Publishing Corp., 1994.



which part of the model needs to change, and might provide a high-level conceptual description of the correction (but the user still needs to make the correction by hand).

Examples, like test-cases for programs, are more valuable if they reveal a discrepancy between the expressed model and the engineer's design intentions. We propose the idea of discriminating examples for this purpose. A discriminating example is synthesized from a combination of the engineer's expressed model and a machine-generated hypothesis of the engineer's true intentions. A discriminating example either satisfies the model but not the hypothesis, or satisfies the hypothesis but not the model. It shows the difference between the model and the hypothesized alternative.

Validating that the model is a true expression of the engineer's intent is an important and difficult problem. One of the key challenges is that there is typically no other written artifact to compare the model to: the engineer's intention is a mental object. One successful approach to this challenge has been automated example-generation tools, such as the Alloy Analyzer. These tools produce examples (satisfying valuations of the model) for the engineer to accept or reject. These examples, along with the engineer's judgment of them, serve as crucial written artifacts of the engineer's true intentions.

Smart debugging[26] can ease the burden on the developer, who often struggles to recognize gaps between what he intends the code to do versus what it really does. A smart debugger enables the developer, who likely lacks training in formal methods, to apply localization, understanding, and fixing of bugs.

[26] A practical smart debugger can guide the human intellect towards bridging semantic gaps with the use of discriminating examples to correct flaws in logical reasoning and supply automatic error localization tools.

Important Legal Disclaimer

Quantstamp Technologies Inc. (the “**Company**” or “**Quantstamp**”) Tokens (the “**Tokens**” or “**QSP Tokens**”) to be offered at the Quantstamp Token Pre-Sale and the Public Sale (collectively, the “**Token Sale**”) are not intended to constitute securities in any jurisdiction. This document (the “**White Paper**”) does not constitute a prospectus or offer document of any sort and is not intended to constitute an offer of securities or a solicitation for investment in securities in any jurisdiction.

This White Paper does not constitute or form part of any opinion on any advice to sell or any solicitation of any offer by Quantstamp to purchase any QSP Tokens, nor shall it or any part of it, nor the fact of its presentation form the basis of, or be relied upon in connection with, any contract or investment decision.

No person is bound to enter into any contract or binding legal commitment in relation to the sale and purchase of the QSP Tokens and no cryptocurrency or other form of payment is to be accepted on the basis of this Whitepaper.

Any agreement between Quantstamp and you as a purchaser in relation to any sale or purchase of QSP Tokens is to be governed a separate Quantstamp Token Sale Terms and Conditions document (the “**Terms**”). In the event of any inconsistencies between the Terms and this Whitepaper, the former shall prevail.

You are not eligible and you are not to purchase any QSP Tokens in the Quantstamp Token Sale if you are a citizen, resident (for tax purposes or otherwise) or green card holder of the United States of America or a citizen of the People’s Republic of China.

No regulatory authority has examined or approved of any of the information set out in this Whitepaper. No such action has been or will be taken under the laws, regulatory requirements or rules of any jurisdiction. The publication, distribution or dissemination of this Whitepaper does not imply that the applicable laws, regulatory requirements or rules have been complied with.

There are risks and uncertainties associated with Quantstamp and its business and operations, the QSP Tokens and the Quantstamp Token Sale.

This Whitepaper, any part thereof and any copy thereof must not be taken or transmitted to any country where distribution or dissemination of this Whitepaper is prohibited or restricted.

CLOSED SYSTEM UTILITY

As of the date of publication of this paper, the Tokens have no known potential uses outside of the Quantstamp ecosystem, and are not permitted to be sold or otherwise traded on third-party exchanges. This paper does not constitute advice nor a recommendation by Quantstamp, its officers, directors, managers, employees, agents, advisors or consultants, or any other person to any recipient of this paper on the merits of the participation in the Token Sale. Quantstamp Tokens should not be acquired for speculative or investment purposes with the expectation of making a profit or immediate re-sale. No promises of future performance or value are or will be made with respect to Quantstamp Tokens. Accordingly, no promise of inherent value, no promise of continuing payments, and no guarantee that Quantstamp Tokens will hold any particular value is made. Unless prospective participants fully understand and accept the nature of Quantstamp and the potential risks inherent in Quantstamp Tokens, they should not participate in the Token Sale.

Quantstamp Tokens are sold as a functional good and all proceeds received by Quantstamp may be spent freely by Quantstamp absent any conditions, save as set out herein.

DISCLAIMER OF LIABILITY

To the maximum extent permitted by the applicable laws, regulations and rules, Quantstamp shall not be liable for any indirect, special, incidental, consequential or other losses of any kind, in tort, contract or otherwise (including but not limited to loss of revenue, income or profits, and loss of use or data), arising out of or in connection with any acceptance of or reliance on this Whitepaper or any part thereof by you.

NO REPRESENTATIONS AND WARRANTIES

Quantstamp does not make or purport to make, and hereby disclaims, any representation, warranty or undertaking in any form whatsoever to any entity or person, including any representation, warranty or undertaking in relation to the truth, accuracy and completeness of any of the information set out in this Whitepaper.

In particular, no representations or warranties whatsoever are made with respect to Quantstamp or the Tokens:

(a) merchantability, suitability or fitness for any particular purpose;

(b) that the contents of this document are accurate and free from any error(s);

(c) that such contents do not infringe any third party rights. Quantstamp shall have no liability for damages of any kind arising out of the use, reference to, or reliance on the contents of this document, even if advised of the possibility of such damages;

This Whitepaper references third party data and industry publications. Quantstamp believes that these references are accurate; however, Quantstamp does not provide any assurances as to the accuracy or completeness of this data. We have not independently verified the data sourced from third party sources in this paper, or ascertained the underlying assumptions relied upon by such sources.

REPRESENTATIONS AND WARRANTIES BY YOU

By accessing and/or accepting possession of any information in this Whitepaper or such part thereof, you represent and warrant to Quantstamp as follows:

(a) you acknowledge that the QSP Tokens do not constitute securities in any form

in any jurisdiction;

(b) you acknowledge that this White Paper does not constitute a prospectus or offer document of any sort and is not intended to constitute an offer of securities in any jurisdiction or a solicitation for investment in securities and you are not bound to enter into any contract or binding legal commitment and no cryptocurrency or other form of payment is to be accepted on the basis of this Whitepaper;

(c) you acknowledge that no regulatory authority has examined or approved of the information set out in this Whitepaper, no action has been or will be taken under the laws, regulatory requirements or rules of any jurisdiction and the publication, distribution or dissemination of this Whitepaper to you does not imply that the applicable laws, regulatory requirements or rules have been complied with;

(d) you agree and acknowledge that this Whitepaper, the undertaking and/or the completion of the Quantstamp Token Sale, or future trading of the QSP Tokens on any cryptocurrency exchange, shall not be construed, interpreted or deemed by you as an indication of the merits of Quantstamp, the QSP Tokens and the Quantstamp Token Sale;

(e) the distribution or dissemination of this Whitepaper, any part thereof or any copy thereof, or acceptance of the same by you, is not prohibited or restricted by the applicable laws, regulations or rules in your jurisdiction, and where any restrictions in relation to possession are applicable, you have observed and complied with all such restrictions at your own expense and without liability to Quantstamp;

(f) you agree and acknowledge that in the event that you wish to purchase any QSP Tokens, the QSP Tokens are not to be construed, interpreted, classified or treated as:

(i) any kind of currency other than cryptocurrency;

(ii) debentures, stocks or shares issued by any person or entity;

(iii) rights, options or derivatives in respect of such debentures, stocks or shares;

(iv) rights under a contract for differences or under any other contract the purpose or pretended purpose of which is to secure a profit or avoid a loss;

(v) units in a collective investment scheme;

(vi) units in a business trust;

(vii) derivatives of units in a business trust; or

(viii) any other security or class of securities.

(g) you are fully aware of and understand that you are not eligible to purchase any QSP Tokens if you are a citizen, resident (tax or otherwise) or green card holder of the United States of America or a citizen or resident of the Republic of Singapore;

(h) you have a basic degree of understanding of the operation, functionality, usage, storage, transmission mechanisms and other material characteristics of cryptocurrencies, blockchain-based software systems, cryptocurrency wallets or other related token storage mechanisms, blockchain technology and smart contract technology;

(i) you are fully aware and understand that in the case where you wish to purchase any QSP Tokens, there are risks associated with Quantstamp and its business and operations and the Tokens;

(j) you agree and acknowledge that Quantstamp is not liable for any indirect, special, incidental, consequential or other losses of any kind, in tort, contract or otherwise (including but not limited to loss of revenue, income or profits, and loss of use or data), arising out of or in connection with any acceptance of or reliance on this Whitepaper or any part thereof by you; and

(k) all of the above representations and warranties are true, complete, accurate and non-misleading from the time of your access to and/or acceptance of possession of this Whitepaper or such part thereof.

CAUTIONARY NOTE ON FORWARD-LOOKING STATEMENTS

All statements contained in this Whitepaper, statements made in press releases or in any place accessible by the public and oral statements that may be made by Quantstamp's respective directors, executive officers, employees or other representatives acting on behalf of Quantstamp that are not statements of historical fact, constitute “forward- looking statements”. Some of these statements can be identified by forward-looking terms such as “aim”, “target”, “anticipate”, “believe”, “could”, “estimate”, “expect”, “if”, “intend”, “may”, “plan”, “possible”, “probable”, “project”, “should”, “would”, “will” or other similar terms. However, these terms are not the exclusive means of identifying forward-looking statements. All statements regarding Quantstamp’s financial position, business strategies, plans and prospects and the future prospects of the industry which Quantstamp is in are forward-looking statements. These forward-looking statements, including but not limited to statements as to Quantstamp’s revenue and profitability, prospects, future plans, other expected industry trends and other matters discussed in this Whitepaper regarding Quantstamp are matters that are not historical facts, but only predictions.

These forward-looking statements involve known and unknown risks, uncertainties and other factors that may cause the actual future results, performance or achievements of Quantstamp to be materially different from any future results, performance or achievements expected, expressed or implied by such forward-looking statements. These factors include, amongst others:

(a) changes in political, social, economic and stock or cryptocurrency market conditions, and the regulatory environment in the countries in which Quantstamp conducts its respective businesses and operations;

(b) the risk that Quantstamp may be unable to execute or implement its business strategies and future plans;

(c) changes in interest rates and exchange rates of fiat currencies and cryptocurrencies;

(d) changes in the anticipated growth strategies and expected internal growth of Quantstamp;

(e) changes in the availability and fees payable to Quantstamp in connection

with its respective businesses and operations;

(f) changes in the availability and salaries of employees who are required by Quantstamp to operate their respective businesses and operations;

(g) changes in competitive conditions under which Quantstamp operates, and

the ability of Quantstamp to compete under such conditions;

(h) changes in the future capital needs of Quantstamp and the availability of financing and capital to fund such needs;

(i) war or acts of international or domestic terrorism;

(j) occurrences of catastrophic events, natural disasters and acts of God that affect the

business and/or operations of Quantstamp;

(k) other factors beyond the control of Quantstamp; and

(l) any risk or uncertainties associated with Quantstamp and its businesses and operations and the QSP Tokens.

All forward-looking statements made by or attributable to Quantstamp or persons acting on behalf of Quantstamp are expressly qualified in their entirety by the factors listed above. Given the risks and uncertainties that may cause the actual future results, performance or achievements of Quantstamp to be materially different from that expected, expressed or implied by the forward-looking statements in this Whitepaper, undue reliance must not be placed on these statements. These forward-looking statements are applicable only as of the date of this Whitepaper.

Neither Quantstamp, nor any other person represents, warrants and/or undertakes that the actual future results, performance or achievements of Quantstamp will be as discussed in those forward-looking statements. The actual results, performance or achievements of Quantstamp may differ materially from those anticipated in these forward- looking statements.

Nothing contained in this Whitepaper is or may be relied upon as a promise, representation or undertaking as to the future performance or policies of Quantstamp.

Further, Quantstamp disclaims any responsibility to update any of those forward-looking statements or publicly announce any revisions to those forward-looking statements to reflect future developments, events or circumstances, even if new information becomes available or other events occur in the future.

MARKET AND INDUSTRY INFORMATION AND NO CONSENT OF OTHER PERSONS

This Whitepaper includes market and industry information and forecasts that have been obtained from internal surveys, reports and studies, where appropriate, as well as market research, publicly available information and industry publications. Such surveys, reports, studies, market research, publicly available information and publications generally state that the information that they contain has been obtained from sources believed to be reliable, but there can be no assurance as to the accuracy or completeness of such included information.

Save for Quantstamp and its directors, executive officers and employees, no person has provided his or her consent to the inclusion of his or her name and/or other information attributed or perceived to be attributed to such person in connection therewith in this Whitepaper and no representation, warranty or undertaking is or purported to be provided as to the accuracy or completeness of such information by such person and such persons shall not be obliged to provide any updates on the same.

While Quantstamp has taken reasonable actions to ensure that the information is extracted accurately and in its proper context, Quantstamp has not conducted any independent review of the information extracted from third party sources, verified the accuracy or completeness of such information or ascertained the underlying economic assumptions relied upon therein. Consequently, neither Quantstamp nor its respective directors, executive officers and employees acting on their behalf make any representation or warranty as to the accuracy or completeness of such information and shall not be obliged to provide any updates on the same.

TERMS USED

To facilitate a better understanding of the QSP Tokens being offered for purchase Quantstamp, and the business and operations of Quantstamp, certain technical terms and abbreviations, as well as, in certain instances, their descriptions, have been used in this Whitepaper. These descriptions and assigned meanings should not be treated as being definitive of their meanings and may not correspond to standard industry meanings or usage.

Words importing the singular shall, where applicable, include the plural and vice versa and words importing the masculine gender shall, where applicable, include the feminine and neuter genders and vice versa. References to persons shall include corporations.

NO ADVICE

No information in this Whitepaper should be considered to be business, legal, financial or tax advice regarding Quantstamp, the QSP Tokens and the Quantstamp Token Sale. You should consult your own legal, financial, tax or other professional adviser regarding Quantstamp and its business and operations and the QSP Tokens. You should be aware that you are bearing the financial risk of any purchase of QSP Tokens for an indefinite period of time.

NO FURTHER INFORMATION OR UPDATE

No person has been or is authorised to give any information or representation not contained in this Whitepaper in connection with Quantstamp and their respective businesses and operations, the QSP Tokens and, if given, such information or representation must not be relied upon as having been authorised by or on behalf of Quantstamp. The Quantstamp Token Sale shall not, under any circumstances, constitute a continuing representation or create any suggestion or implication that there has been no change, or development reasonably likely to involve a material change in the affairs, conditions and prospects of Quantstamp or in any statement of fact or information contained in this Whitepaper since the date hereof.

RESTRICTIONS ON DISTRIBUTION AND DISSEMINATION

The distribution or dissemination of this Whitepaper or any part thereof may be prohibited or restricted by the laws, regulatory requirements and rules of any jurisdiction. In the case where any restriction applies, you are to inform yourself about, and to observe, any restrictions which are applicable to your possession of this Whitepaper or such part thereof at your own expense and without liability to Quantstamp.

Persons to whom a copy of this Whitepaper has been distributed or disseminated, provided access to or who otherwise have the Whitepaper in their possession shall not circulate it to any other persons, reproduce or otherwise distribute this Whitepaper or any information contained herein for any purpose whatsoever nor permit or cause the same to occur.

RISKS AND UNCERTAINTIES

Prospective purchasers of QSP Tokens should carefully consider and evaluate all risks and uncertainties associated with Quantstamp, the QSP Tokens, the Quantstamp Token Sale, all information set out in this Whitepaper and the Terms prior to any purchase of QSP Tokens. If any of such risks and uncertainties develops into actual events, the business, financial condition, results of operations and prospects of Quantstamp could be materially and adversely affected. In such cases, you may lose all or part of the value of the QSP Tokens.

IF YOU ARE IN ANY DOUBT AS TO THE ACTION YOU SHOULD TAKE, YOU SHOULD CONSULT YOUR LEGAL, FINANCIAL, TAX OR OTHER PROFESSIONAL ADVISOR(S).