ChainSoft: Collaborative Software Development using Smart Contracts

Michał Król, Sergi Reñé, Onur Ascigil and Ioannis Psaras
Dept. of Electronic & Electrical Engineering,
University College London
WC1E 7JE, Torrington Place, London, UK
{m.krol, s.rene, o.ascigil, i.psaras}@ucl.ac.uk

ABSTRACT

In recent years, more and more companies require dedicated software to increase the efficiency of their business. However, with rapidly changing technologies it is often inefficient to maintain a dedicated team of developers. On the other hand, outsourcing software development requires considerable effort and trust between involved parties to ensure the quality of the code and adequate payment.

We present ChainSoft - a platform for outsourcing software development and automatic payments between parties that distrust each other, by means of blockchain technology. ChainSoft allows any developer to create software and submit software, includes automatic code verification and enforce users' proper behavior. We implement our system using Ethereum Smart Contracts and Github/Travis CI and present first evaluation proving its security and low usage cost.

1. INTRODUCTION

Software development and information technology (IT) is a fast growing industry with a total revenue over 3.8 trillion dollars globally, and it continues to grow each year [1]. In US alone, there are more than $100K$ software and IT companies with 99% being small and medium-sized firms with under 500 employees [2]. In addition to the monetary worth, the industry is also growing in terms of workforce every year—the US government predicts that the software development workforce will grow by 22% over the next ten years [2]. Moreover, the industry is increasingly demanding specialised skills for possibly large range of domains and specialty in particular software systems with considerable complexity. As a result, it is becoming increasingly difficult—especially for the small and medium companies that constitute the vast majority of the industry—to form a team of developers with expertise in wide range of domains. This leads to the development tasks in such companies being increasingly outsourced to third-party contractors, and at the same time larger companies seeking developers globally to meet the increasing demand.

However, outsourcing software development currently requires considerable effort and resources to find appropriate programmers for the task at hand. Developers and employers need to establish trust—developers must ensure that the requirements of the task are clearly stated and do not change after the agreement, and the employers must ensure that all the requirements of task are fulfilled. In this paper, we propose ChainSoft—a Blockchain-based mechanism to automate outsourcing of software development. ChainSoft brings together mutually untrusted parties to collaborate on software development on a per-task basis. The procedure involves a software requestor first publishing a task together with a set of tests to verify that the code to be provided by third-party contributors match the requirements. Each published task is associated with a reward, and the first contributor to post a solution (to a code repository whose location is provided by the task owner) that passes the tests automatically obtains the reward. ChainSoft utilizes smart contracts to ensure that a contract owner (the requestor) pays contributors to achieve trust in a distributed software development environment.

Examples of tasks that can be achieved through ChainSoft include writing a new piece of software as well as adding new features, finding vulnerabilities (i.e., bugs) in an existing software [3, 4], and so on. In the case of discovering bugs, the owner publishes his software (either the binaries or the code in the case of open source software) and the contributors post specific tests that trigger errors/failure conditions in the owner's code. The obvious challenge with this approach is the specification of precise tests by the contract owners to verify the correctness of the solution. However, existing software development practices, e.g., continuous integration (CI), involve iteratively testing code upon adding new features or enhancements. Therefore, we do not consider having to provide test program as a limitation.

Existing solutions for outsourcing software development involve human-in-the-loop for the verification of tasks, and therefore, lack an automated reward collection mechanism, which is crucial for mutually untrusted

parties to collaborate [5, 4]. ChainSoft's automated reward collection mechanism is built on existing systems including BlockChain-based smart contracts; CI systems that automatically build, run tests and publishes test results upon commits by a contributor; and a service called Oracle that enables a BlockChain contract to interact with the outside world in a secure way, i.e., make network requests from a contract to obtain the result of tests.

The rest of the paper is organized as follows. First, we present background on the existing technology that enables ChainSoft in Section 2, followed by the threat model and assumptions in Section 3. In Section 4, we present an overview of the ChainSoft architecture and in Section 5, we present the evaluation. This is followed by the Related Work and conclusions in Sections 6 and 7, respectively.

2. BACKGROUND

2.1 Blockchain and Smart Contracts

The blockchain technology [6] implements a distributed, append-only ledger in the form of connected blocks. Once information is stored in the blockchain it cannot be removed or altered. Network participants use a consensus protocol to agree on current state of the ledger. As long as the majority of participants are honest, the integrity of the ledger is assured. Blockchain is widely used to record transactions in multiple cryptocurrencies (*i.e.,* Bitcoin [6], Ethereum [7]). A common extension consists of scripting languages enables to include logic as part of the transaction and allows deployment of Smart Contracts - code submitted to blockchain and executed by all miners. Solidity - the scripting language of Ethereum is Turing-complete allowing to implement wide range of applications running on the blockchain.

2.2 Oracle

While turing-complete Smart Contracts allow to implement any logic on top of blockchain, their usefulness remains limited without an external source of information. To improve their usefulness, Smart Contracts additionally require access to data about real-world state and events. Data feeds (also known as "oracles") aim to meet this need. Oracles (i.e. Oraclize [8], Town Crier [9]) are contracts on the blockchain that contact trustworthy websites and serve data requests by other contracts. However, smart contracts themselves lack network access, and HTTPS does not digitally sign data for out-of-band verification. At the same time, data provided by oracles can determine behaviour of smart contracts and money transactions and thus must be trusted. Recently, TLS Notary was proposed as a solution to this problem [10]. TLS Notary can provide a proof that certain information were retrieved from a trusted HTTPS server and ensure data integrity.

2.3 Travis CI

Tracis CI [1] is a Continuous Integration (CI) and deployment system aimed at building and testing software projects hosted at GitHub [2]. Travis CI automatically detects when a commit has been made and pushed to a GitHub repository and tries to build the project and run tests. Travis CI is configured by adding a YAML [11] format text file specifying the programming language used, the desired building and testing environment (including dependencies which must be installed before the software can be built and tested). Travis CI supports integration with Docker allowing to recreate custom environments and various deployment platforms (i.e. Heroku, AWS CodeDeploy). Travis CI checks out the relevant branch and run the commands specified in .travis.yml (later called environment file), which usually build the software and run any automated tests. When that process has completed, Travis can notify developers in the way it has been configured to do so, for example, by sending an email containing the test results. The service expose also an API allowing to query it with HTTPS requests.

3. THREAT MODEL AND ASSUMPTIONS

We consider that the following actors participate in the transaction:

- Requestor - submits a software development task to the network and makes the payment.
- Software Developer - develops and submits the requested software for a specified reward.
- Payment System - a smart contract running on a blockchain.
- Oracle - Github with Travic CI.

Our threat model assumes that when the Requestor submits its task to the blockchain, it does not know which Developer will execute it. We assume that for the requestor, the value of the developed software (V_{Ri}) is higher than the price she is willing to pay (P_{Ri}). At the same time, the cost of developing the software (C_i) is lower than the offered price (P_{Ri}). Both parties wish to finalise the transaction, but mutually distrust one another. Each party is potentially malicious, i.e., they may attempt to steal funds, avoid making payments, and forge results if it benefits them. Any time each party may drop, send, record, modify, and replay arbitrary messages in the protocol. Both the requestor and the software developer trust the blockchain, their own

[1] `https://travis-ci.org/`
[2] `https://github.com/`

environments, and the oracle. The rest of the system, such as the network between the parties and the other party's software stacks and hardware are untrusted. We assume that anyone can submit a solution (software), but only the first valid one will be rewarded. To prevent parallel development by multiple parties, a simple deposit scheme could be applied (similar to [12]).

4. OVERVIEW

ChainSoft is a platform for outsourcing software development and automatic payments between mutually distrusting parties. An overview of the system interactions is presented on Fig. 1. A requestor, requiring software first needs to create its description. The description consists of a set of tests the software needs to pass in order to fulfil the requirements. The description is submitted to a GitHub repository and made publicly visible. The requestor then submits the task to the smart contract including the reward and a pointer to the GitHub repository. At this point, anyone can start writing software that will pass all the specified tests. Once a developer succeeds, he uploads the solution to his GitHub repository cloning the set of tests from the task description. He can now claim the reward from the contract. Before making the payment, the contract will verify that the solution includes all the tests from the description and the build succeeds. If its the case, the payment will be released and the money transferred directly to the developer.

4.1 Contract

The smart contract running on top of the blockchain is the main component of ChainSoft. It provides an interface for users to interact with and communicates with outside world using the oracle (Sec. 2). Listing 1 presents the interface of the contract. When a requestor submits a new task, the transaction must contain funds to be set as the reward and a GitHub repository with a task description (Sec. 4.2). First, the contract checks if the specified repository exists and contains an environment file. The environment file contains a description for Travis CI on how to create the environment, compile the project and run tests and is a required component for the whole system. If the check is successful, the contract creates a structure with the task description. At this point the task description and the reward are publicly visible and developers can start working on the project. We provide an additional function "addReward()" to increase the reward for completing given task. Any user use this method to send more funds to the contract and thus increase the incentive for developers to finish the project. Money sent to the contract by the requestor or any other user remain lock and cannot be manually retrieved. However, when a tasks is submitted it is possible to specify a deadline after which, if no valid solution is submitted, the task will be withdrawn and the founds returned to the requestor/participating users.

Similarly to submitting tasks, when a developer submits a solution using "submitSolution()", he must provide a link to a GitHub repository and an access token. The contract checks its existence and verify if the solution passes all the tests and fulfil the specified requirements (Sec. 4.3). The name of the solution repository must start with the account number of the developer, to prevent someone else from stealing the reward. If the check is successful, the reward is sent to the developer and the task is marked as completed. The notified requestor, can now clone the solution and start using the software. Only the first developer with a valid solution can claim the reward.

We implement a set of events allowing users to subscribe and be notified when one of occurs.

4.2 Task Description

Following the Test Driven Development (TDD) approach, a requestor defines the solution by writing a set of tests. A submitted solution must pass all the tests in order to be considered as a valid one. For our platform, such an approach has two main benefits. First, developers are encouraged to write only minimal amount of code satisfying the tests. Second, it formalises the requirements description and allows to avoid disputes between requestors and software developers on what exactly was supposed to be done.

While requestors should carefully write tests, the wide range of testing frameworks together with the environmental .yml file allow to create very specific sets of requirements. The environmental file can set up a docker image and thus recreate the target deployment platform concerning the target OS, installed tools, libraries and supported programming languages.

According to the complexity of the software developed, different types of testing are available for requestors in order the check the code fulfils the design requirements, in terms on verification functionality, integration, performance, etc:

- **Unit Tests**: Requestors can make use of unit testing to test small and individual software functions, not depending on any other code or external resource, such as network or database. Unit tests should essentially just give the function that is tested some inputs, and then check what the function outputs is correct. Most of coding language provide testing tools, such as JUnit [13] in Java, or CppUnit [14] and Google Test C++ Framework [15] in C++.
- **Integration Tests**: In order to verificate software functions depending on other functions or external software or systems, integration tests must be

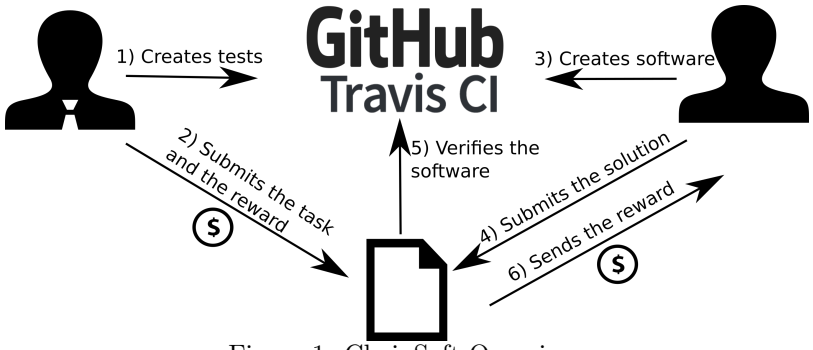

Figure 1: ChainSoft Overview.

used. Integration tests can be used, for example, to validate a database related test or for situations where unit testing is not enough. Integration tests can be written with the same tools as unit tests.

- **Other Tests**: There are also other types of testing that can be considered and specified in the environment file. These tests may include, for example, performance tests, regression tests, security tests, or any required type of testing necessary to ensure and verificate the correctness and functionality of the software. However there are other types of test, such as some functional tests or usability tests, that are not considered in the Chainsoft development cycle, since it can be too complex or subjective to be automatically tested by the platform.

Once the developed software has been created, the requestor can verify the code, specify more precise requirements and submit a new task with a new reward. Any developer can then take over and build on top of the previously submitted code.

ChainSoft can also be used as an automatic bug bounty platform. The requestor needs to write tests that will succeed when the software fails, raises an exception or provides an unexpected output. In this case, already compiled binaries can be used, so the requestor does not have to reveal its source code. When a bug is found, the requestor should patch its software and publish a new task, allowing other hackers to look for additional bugs.

4.3 Task Verification

We implement ChainSoft on top of the GitHub and Travis CI development tools. A developer, submitting its solution, prepares its own git repository with the tests defined by the requestor, environment (.yml) file, and the produced code. In order to determine if a solution fulfils the specified requirements, the platforms needs to:

- Compare if the included tests are the same as the ones submitted by the requestor.
- Recreate the specified environment and compile the whole project.
- Run the tests against the submitted solution.

Comparing the included tests is essential for the system. Without it, a malicious developer could alter the verification tests to accept any code and claim the reward. However, downloading and comparing all the test files on the smart contract is highly inefficient and would require a lot of resources. On the other hand, GitHub and Travis have no knowledge of the task description. To improve the performance, we add an additional precompilation phase in the environment file. The section creates a file containing a list of all the test files and their checksums. The whole process is shown on Fig. 2.

Before compiling the project, Travis CI creates the checksum file, computes its hash and checks it against a static value defined by the requestor. If the hashes do not match, the compilation is stopped and the whole process fails. This approach assures that the solution contains the same set of tests as the task description and requires only minimal amount of work on the smart contract that is independent from the number of test files (see Sec. 5).

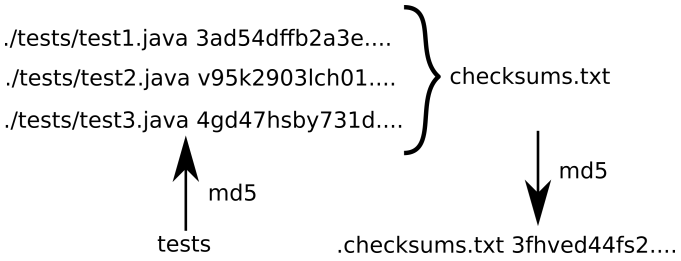

Figure 2: Creating the checksum file.

```
pragma solidity ^0.4.0;

contract Software is usingOraclize{
    address owner;

    struct Task {
        string testRepo;
```

```
        uint reward;
        address requestor;
        bool active;
        string solutionRepo;
        address developer;
    }

    event taskAdded(uint id, string
    testRepo, uint reward);
    event taskSolved(uint taskId);
    event solutionSubmitted(uint taskId);
    event solutionRejected(uint taskId);

    function submitTask(string repo)
      public payable returns(uint);

    function addReward(uint taskID,
        string testRepo)
      public payable returns(uint);

    function submitSolution(string repo,
          string token)
      public returns(uint);
}
```

Listing 1: ChainSoft Smart Contract label

5. EVALUATION

We evaluate ChainSoft by performing a security analysis and running and investigating the behaviour of the smart contract on a test Ethereum network. We implement our solution using cpp-ethereum - a C++ Ethereum client[3] on a Dell XPS 13 laptop.

5.1 Security Analysis

In this section, we provide the intuition behind the security properties of the protocol. We defer formal proofs of security to the extended version of the paper.

The logic implemented on the smart contract relies heavily on the oracle. An oracle itself consist of an immutable smart contract that communicates with the GitHub/Travis CI. When returning a result of a query, the oracle provides an verifiable proof based on TLS Notary, confirming that the response came from the valid HTTPS server. ChainSoft relies thus on GitHub/-Travis CI being honest and correctly run the specified tests. To improve the security, the smart contract could query additional services providing similar functionality [4]. A more secure alternative consist of running a dedicated server based on Intel SGX enclave. Even if the GitHub/Travis CI are honest, their service can become unavailable. Both requestors and developers should verify if the service is up before submitting their request. If the service is unavailable, a user has to resubmit the task/solution later on, paying additional transaction fee.

When a requestor creates a new task, it looses control over the submitted reward. The reward will be returned if no valid solution is submitted within the specified deadline. Developers must thus carefully analyse the task before committing to create the software. A developer should work on the solution using a private repository so no one can use the already created code to claim the reward. When submitting the task, the user provides a Travis CI access token to the contract. At this point, anyone (including the contract) can see the code and its status. This is a required step to prevent the developer from getting the reward and not revealing the code, but if an attacker intercepts the message and completely isolates the developer from the network, he can claim the reward himself.

ChainSoft does not allow to lock a task for a specific developer. It is thus possible that multiple users will concurrently work on the same solution and only the fastest one will get the reward. However, a simple notification mechanism, where developers inform that they are working on a project could help others to make a better decision on whether on not commit to a project and encourage cooperation between different coders. If multiple developers/entities are involved in the software development process, they must manually split the reward between them.

5.2 Smart Contract

We analyse the cost of invoking each function of the contract and its deployment (Tab. 1). The tests were performed on the Rinkeby test network[5]. Ethereum allows specifying a priority for newly submitted transactions. The slow priority ones are processed within 10 minutes, standard priority within 5 minutes and the fastest priority within 2 minutes. With faster processing time comes increased transaction cost. With the slowest transaction processing, the system is cheap to exploit, keeping the cost much below 1$[6] (to be further split between requestors and developer). However, with the fastest transaction, the cost increases up to almost 8$. The delay even up to 10 minutes and the cost lower than 4$ are acceptable for a software development project that usually takes days and significant amounts of money to finish.

6. RELATED WORK

There are existing platforms to foster collaboration between developers by providing incentives through rewards. The closest one to ChainSoft is the GitCoin [5] platform, where entities called "bounty submitters" publish their open source codes along with tasks in the form of GitHub issues to be resolved by third parties. Once a developer submits a solution, the bounty submitter

[3] `https://github.com/ethereum/cpp-ethereum`
[4] i.e. GitLab - `https://gitlab.com`
[5] `www.rinkeby.io`
[6] Costs calculated using `https://ethgasstation.info/`

Function	Gas	Ether Slow ($)	Ether Standard ($)	Ether Fast ($)
Deploy	4191210	0.00013 (0.057$)	0.01387 (6.324$)	0.03656 (16.673$)
submitTask	217410	0.00003 (0.013$)	0.00278 (1.267$)	0.00806 (3.675$)
addReward	38248	0.00001 (0.007$)	0.00145 (0.662$)	0.00421 (1.919$)
submitSolution	164241	0.00001 (0.002$)	0.00053 (0.241$)	0.00153 (0.698$)
Total per task		0.00006 (0.027$)	0.0064 (2.92$)	0.01688 (7.698$)

Table 1: Cost of invoking contract functions.

must *manually accept* the solution in order for a third party developers to get their payments, which happen through Ethereum-based contracts. However, we argue that having a human-in-the-loop for accepting solutions defeats the purpose of trust establishment through Ethereum-based smart contracts: there is no guarantee that one gets a reward for resolving an issue.

Bounty0x [4] is another project aimed at building a similar system where rewards (i.e., bounties) can be issued for software tasks. The platform enables anyone to post bounties, and disbursing payments to bounty hunters (i.e., third-party developers). Bounty0x also introduces human-in-the-loop for the verification of completed tasks by introducing third-party verifiers called bounty sheriffs. The sheriffs collect rewards (tokens) for verifying the accuracy and quality of bounty hunters' submissions. A reputation system is introduced to evaluate trustworthiness of both the the hunters and the sheriffs in the platform. This project is currently is in alpha stage and the details of the reputation system are not clearly specified. For instance, it is not clear whether collusions can be prevented between bounty hunters and sheriffs or between issuers and sheriffs in which case the platform can start rewarding dishonest hunters or not reward honest hunters. Also, the platform must provide sufficiently high reward for validation of submissions, because this can be a heavy task by itself. We argue that the task issuer (as the employer) is the only entity that should judge the quality of a submission, because it is not clear what happens in the event of a dispute between verifiers and issuers.

7. CONCLUSION AND FUTURE WORK

We presented ChainSoft, a new platform for secure outsourcing software development. Our system allows any user to submit tasks or solutions and enforces users' proper behaviour using Smart Contracts and trusted oracles. Our solution is secure against rational adversary, cost efficient and introduces minimal overhead. In our future work, we plan to extend our system by allowing development outsourcing without making the code public. We also plan to investigate multiple or trusted oracles using trusted execution environments.

8. REFERENCES

[1] Gartner says global IT spending to reach $3.7 trillion in 2018. `https://www.gartner.com/newsroom/id/3845563`.

[2] Software job growth. `https://software.org/reports/2017-us-software-impact/`.

[3] Hydra project. `https://thehydra.io`.

[4] Rewarding the token economy. white paper v1.0. a trustless bounty hunting network with a staking and token burning based review system for submissions. `https://bounty0x.io/whitepaper_en.pdf`, January 2018.

[5] Gitcoin. grow open source. `https://gitcoin.co`.

[6] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2008.

[7] Vitalik Buterin et al. Ethereum white paper, 2013.

[8] A scalable architecture for on-demand, untrusted delivery of entropy. `http://www.oraclize.it/papers/random_datasource-rev1.pdf`.

[9] Fan Zhang, Ethan Cecchetti, Kyle Croman, Ari Juels, and Elaine Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, pages 270–282. ACM, 2016.

[10] Tlsnotary - a mechanism for independently audited https sessions. `https://tlsnotary.org/TLSNotary.pdf`, 2014.

[11] Oren Ben-Kiki, Clark Evans, and Brian Ingerson. Yaml ain't markup language (yaml^{TM}) version 1.1. *yaml. org, Tech. Rep*, page 23, 2005.

[12] Michał Król and Ioannis Psaras. Spoc: Secure payments for outsourced computations. In *Proceedings of the 2018 NDSS DISS workshop*, 2018.

[13] Junit 5. the new major version of the programmer-friendly testing framework for java 8 and beyond.

[14] cppunit test framework. `http://www.myurl.com`.

[15] Google's c++ test framework. `https://github.com/google/googletest`.