SmartDEMAP: A Smart Contract Deployment and Management Platform

Markus Knecht(✉) and Burkhard Stiller

Communication Systems Group CSG, Department of Informatics IfI,
University of Zürich, Binzmühlestrasse 14, 8050 Zürich, Switzerland
markus.knecht2@uzh.ch

Abstract. Smart contracts on a blockchain behave exactly as specified by their code. To be sure that a smart contract behaves as expected, the end-user has to either analyze its code or trust a potentially anonymous developer or auditor to do so. This approach proposes a smart contract deployment and management platform that can execute development tools and code quality tools in a trusted way and uses this to reduce the trust required into the smart contract developer or auditor. Additionally, such a platform can provide new capabilities for developers aiding them in the creation of smart contracts.

1 Introduction

Smart contracts are programs which run in a trusted execution environment provided by a blockchain [2]. The code of smart contracts can dictate how valuable assets, associated with a smart contract, are handled. A flaw in the code can lead to the loss or theft of the handled assets [10]. Developing bug-free software is challenging even for skilled professionals [7]. Programming languages and tools like formal verification or automated tests can support that process. Before a smart contract is trusted with assets, such as cryptocurrency coins or a ownership certificate, it must be ensured that the code implements the expected and specified behavior. A end-user can ensure this by analyzing the code, by trusting the developer to have implemented the specified behavior, or by trusting an auditor to verify that the code implements the specified behavior. Analyzing the code is not an option for most end-users, because of the complexity of the task as well as the required time.

This paper proposes *SmartDEMAP* a smart contract deployment and management platform which reduces the trust required into smart contract developers and auditors by imposing restrictions on the smart contracts that can be deployed on it. The restrictions are enforced by executing formal verifiers [1,4], compilers, automated bug-finders [8], or other development and code quality tools on smart contracts. Such restrictions could consist of a formal proof of some specified properties, enforcing a programming language, or requiring a negative result from an automatic bug finder. *SmartDEMAP* can reduce the trust needed into third parties, without requiring expertise in software auditing.

D. Tuncer et al. (Eds.): AIMS 2017, LNCS 10356, pp. 159–164, 2017.
DOI: 10.1007/978-3-319-60774-0_15

SmartDEMAP allows to run development and code quality tools in a trusted way to do deploy-time and run-time checks to increase smart contract quality and robustness. To accelerate the integration into the development process, we propose to develop a custom smart contract programming language that is aware of the existence of *SmartDEMAP*. Such a language can generate code that facilitates the provided functionality and gives a developer easy access to it. Existing languages can integrate *SmartDEMAP* by providing libraries to interact with it.

2 Hypotheses

An investigation into the current state of smart contract development has shown that there currently is a high risk for end-users when interacting with smart contracts, as shown by the "The DAO" incident [10], where an attacker exploited a bug to steel 3.6 million in ether. The following hypotheses are premises for developing and analysing *SmartDEMAP*. In the project it should be researched how well these premises can mitigate the respective risks.

Hypothesis 1: A platform on the blockchain, which provides access to trusted execution of development and code quality tools, enables the development of smart contracts which can manage, verify, and analyze the code of other smart contracts in order to increase their robustness as well as reducing the trust required in developers and auditors.

Sub-hypothesis 1.1: The ability to associate attributes with a smart contract based on a trusted analysis of its code, where the results can be queried and analyzed by other smart contract or external sources, enables the detection of misbehaving smart contracts.

Sub-hypothesis 1.2: A smart contract that controls the compilation and deployment of other smart contracts by using development and code quality tools, allows a developer to add new features or bug fixes to a smart contract after it has been deployed, without the need for end-users to trust the developer.

Sub-hypothesis 1.3: A custom smart contract programming language with the ability of accessing code analysis at run-time can prevent certain exploits.

3 Related Work

There are two categories of work related to *SmartDEMAP*. On one hand there are smart contract specific development and code quality tools including programming languages. On the other hand there is research on how resource intensive computations can be executed in a trusted way despite the resource limitations of smart contract enabled blockchains [2,11].

For the tools it is important that they work in a reliable way and can not be fooled by a fine tuned input. If a compiler guarantees a certain semantic which do not hold in the generated byte code, then a trusted execution of the compiler will not help either. An earlier Solidity version had such a problem [9]. Formal verification [1,4] and automatic bug-finding [8] are other relevant research topics for *SmartDEMAP*. Research into these topics is relatively new and the developed tools are not in wide use and geared more towards trained professionals. *SmartDEMAP* could change that by allowing users not trained in these tools to still benefit from their results.

Theoretical results already exist concerning the execution of complex computations in a way, such that the results can be trusted [3,5]. Their currently is a project developing a concrete implementation [6] based on the theoretical foundations from [3,5], and will allow smart contracts to trigger a trusted computation and access the result.

One part of the current research promises new tools that can be used to improve the development process and reduce the exploitability of smart contracts. Another part promises ways to run complex computations in a trusted way, which can be utilized during the execution of smart contracts. There is no research investigating if and how these two approaches could be combined. *SmartDEMAP* will close that research gap.

4 Smart Contract Exploits

In recent years, different exploits have been found which are usable against some of the existing smart contracts [8]. The most prominent example is the "The DAO" theft [10]. Contrary to centralized software development, smart contracts operate in an open environment where arbitrary adversaries can exist [8] and thus attacks can originate from inside the same virtual machine. Additionally, it is substantially harder to correct a bug because smart contract code is unchangeable after it is deployed on a blockchain [2,11]. Most problems occur when unknown code is executed, because it may have been deployed by an adversary. Such vulnerabilities can lead to loss or theft of valuable assets and are often hard or even impossible to fix. Most users of such smart contracts do not have the expertise and time to ensure that it is safe to trust the smart contract with their assets.

5 Platform-Based Smart Contract Management

The auditors and developers of smart contracts are often anonymous and their trustworthiness is unknown. Some smart contracts include code which allows a privileged entity to exchange parts of the code. This is done to make it possible to replace code containing a flaw with a fixed version. On the other hand this could be used to inject code that violates a specified behavior.

SmartDEMAP determines a new mechanism, which allows only code to be deployed that does not violate an associated behavior specification. The behavior can be specified as a formal specification and on deployment needs a proof that it conforms to the specification. Other approaches like defining a test suite and only allow code to be deployed that passes the test suite will be investigated in addition to the formal verification approach. This has only a benefit if the formal verification tool or test suite can be run in a trusted way. This reduces the trust required in developers and auditors and replaces it with trust into the tools and their input (formal specification or test suite).

To achieve this, the approach to be designed will follow a blockchain-based path, with a platform for management, analysis, and deployment of smart contracts (*SmartDEMAP*). The platform will manage a set of tools and use them to enforce that smart contracts deployed on it fulfill as set of specifiable criteria. Such a tool set contains formal verifiers, compilers, automatic test suites, and automatic bug-finders. These tools are often complex and *SmartDEMAP* will provide a way to ensure that these tools fulfill their purpose. This indicates that each *SmartDEMAP* instance needs a entity fulfilling this role. This may be another instance or a known third party (e.g. Microsoft, Amazon, Google) as well as a consortium of people founded exclusively for that purpose. This system reduces the trust needed in the code quality tools and replaces it with trust in the tool verification entity.

6 Improved Smart Contract Programming Language

SmartDEMAP benefits from a custom programming language, which is aware of it and uses its features during compilation or at run-time. Such a custom programming language can incorporate *SmartDEMAP* to give additional guarantees by generating the respective run-time checks based on the platforms capabilities. Further it provides a simple way for developers to access the platforms services. A new programming language provides the opportunity to analyze existing languages as well as common exploits of smart contracts programmed in these languages. A smart contract programming language covering this aspect could prevent some exploits and common pitfalls by design.

One currently preferred approach is a language based on a process calculi as suggested in [4]. This does prevent by design some of the common exploits, such as the reentrancy exploit that brought “The DAO” to its knees [10]. This exploit is prevented because no state is shared between different processes and unlike a function a process cannot be called again if it is still running, and thus, no unexpected state change can occur. This work evaluates if such a language efficiently can be compiled to existing smart contract virtual machines and which exploits could be prevented on the language level.

7 Methodology

This project is of high importance if smart contract should become safe to use by non-experts. On one hand *SmartDEMAP* can give them a higher degree of certainty, that it is safe to interact with a smart contract without the risk of unexpected behavior. On the other hand *SmartDEMAP* and the custom programming language help the developer to deliver smart contracts that are harder to exploit.

The project is approached by developing a model of *SmartDEMAP* and the custom programming language that describes their respective capabilities and guarantees. Beside the model the platform as well as a compiler for the language are implemented as a proof of the practical feasibility.

The biggest risk involved in the project is that the currently developed tools like formal verifiers as well as the trusted execution infrastructure will not be available in time or do not satisfy the needs of *SmartDEMAP*. The developed models are used to prove that certain exploits can be prevented fully or at least to which degree if the described platform and the custom programming language is used. The expected proofs are:

1. A proof that it is possible to decide if a unknown smart contract can be called without the risk of becoming vulnerable to certain exploits.
2. A proof that a developer can only deploy code that result in a behavior that conforms to a formal specification.
3. A proof that smart contracts programmed in the custom language are not vulnerable against certain exploits.

The evaluation of which exploits are preventable this way is another expected result from this project. Beside the theoretical results an implementation of *SmartDEMAP* on the EVM [11] is expected.

References

1. Bhargavan, K., Delignat-Lavaud, A., Fournet, C., Gollamudi, A., Gonthier, G., Kobeissi, N., Kulatova, N., Rastogi, A., Sibut-Pinote, T., Swamy, N., Zanella-Béguelin, S.: Formal verification of smart contracts: short paper. In: 2016 ACM Workshop on Programming Languages and Analysis for Security, PLAS 2016, Vienna, Austria, pp. 91–96 (2016)
2. Buterin, V.: A next-generation smart contract and decentralized application platform. Technical report (2014). Accessed 15 Nov 2016
3. Canetti, R., Riva, B., Rothblum, G.N.: Practical delegation of computation using multiple servers. In: 18th ACM Conference on Computer and Communications Security, CCS 2011, Chicago, Illinois, USA, pp. 445–454 (2011)
4. Edstroem, R., Pettersson, J.: Safer smart contracts through type-driven development. Master's thesis, Chalmers University of Technology and University of Gothenburg (2016)
5. Jain, S., Saxena, P., Stephan, F., Teutsch, J.: How to verify computation with a rational network, June 2016
6. Teutsch, J., Reitwiessner, C.: A scalable verification solution for blockchains (2017). http://people.cs.uchicago.edu/%7Eteutsch/papers/truebit.pdf
7. Anand, K., Rai, K., Madan, L.: Software crisis. Int. J. Innov. Res. Technol. **1** (2014)
8. Luu, L., Chu, D.-H., Olickel, H., Saxena, P., Hobor, A.: Making Smart Contracts Smarter. Cryptology ePrint Archive, Report 2016/633 (2016)

9. Reitwiessner, C.: Security alert solidity variables can be overwritten in storage (2016). https://blog.ethereum.org/2016/11/09/analysis-storage-corruption-bug. Accessed 03 Dec 2016
10. Vessenes, P.: Deconstructing the DAO Attack: A Brief Code Tour (2016). http://vessenes.com/deconstructing-thedao-attack-a-brief-code-tour. Accessed 03 Dec 2016
11. Wood, G.: Ethereum: A Secure Decentralised Generalised Transaction Ledger (2015). http://gavwood.com/paper.pdf. Accessed 03 Dec 2016