Summary of: A Federated Society of Bots for Smart Contract Testing

Emanuele Viglianisi
Fondazione Bruno Kessler
Trento, Italy
eviglianisi@fbk.eu

Mariano Ceccato
University of Verona
Verona, Italy
mariano.ceccato@univr.it

Paolo Tonella
Università della Svizzera italiana
Lugano, Switzerland
paolo.tonella@usi.ch

***Abstract*—The peculiar novelty of smart contracts is a computational model where irreversible transactions are stored in a distributed persistent data storage, namely the blockchain. The technical nature of this new type of software opens to new kinds of faults, which require specific test capabilities to be revealed. In this paper we present SOCRATES, an extensible and modular framework to automatically test smart contracts. The distinctive features of SOCRATES are: (1) a collection of composable behaviours that exercise smart contracts in the blockchain; (2) it deploys a society of bots, with the purpose of detecting defects arising from multi-user interactions, which are impossible to reveal when deploying a single bot. Our empirical investigation demonstrates that SOCRATES is able expose both known and previously unknown faults in smart contracts that are actively run in the official Ethereum blockchain. Moreover, we show that a society of multiple bots is more efficient in fault exposure than a single bot alone.**

***Index Terms*—Smart contracts; Software testing; Ethereum.**

I. CONTEXT

A major extension to crypto currencies, such as Ethereum, consists of *smart contracts*. These are executable pieces of code that are saved in the blockchain and whose legitimate execution is certified by the nodes of a crypto currency network. The execution model of smart contracts is quite unique and novel. In fact, after a smart contract is persisted in the blockchain, its code and all of its transactions are immutable, even when programming mistakes are later detected. This means that incorrect execution results remain immutable, forever. For this reason, comprehensive and accurate testing of smart contracts is essential to reveal programming defects before incorrect transactions are immutably stored in the blockchain.

In a typical smart contract, multiple users interact by playing different roles. For instance, Figure 1 depicts an interaction scenario for a token contract: the *spender* delegates an *initiator* to move a given amount of tokens *t* from the account owned by the *initiator*. Then, the *initiator* moves the tokens from the account owned by the *spender* to the account own by the *receiver*.

A single user is not sufficient to test this scenario. In fact, this scenario requires three distinct actors who interact coherently, according to the specific contract roles, i.e., a *spender*, who authorizes the *initiator* to move some value to the *receiver*.

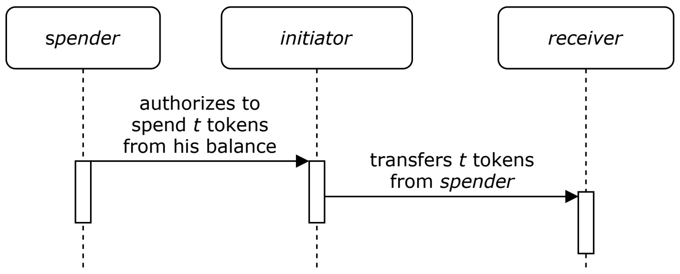
spender
initiator
receiver
authorizes to spend t tokens from his balance
transfers t tokens from spender

Fig. 1. Example of interaction among three distinct users

II. TESTING FRAMEWORK

Smart contracts, by definition, mediate transactions occurring among multiple end users, who play different roles. Therefore, smart contracts are, intrinsically, multi-role programs. To properly automate smart contract testing, we propose SOCRATES (Smart ContRActs TESting), a novel, extensible and modular testing framework, that relies on a federated society of interacting bots. Each of these bots performs a distinct contract role (i.e., a distinct end user role).

Each contract might explicitly define different roles, or roles might implicitly emerge from distinct contract permissions granted to distinct user profiles (e.g., token-owner, contract-owner). Our simulator module concurrently runs many bots, by iteratively assigning execution slots to them, with the objective of detecting implementation defects that might be sheltered in a possibly intricated and complicated interaction pattern admitted by a contract that supports diverse roles with specific responsibilities and privileges. The bots of SOCRATES are guided by different *behaviours* and the correct execution of the smart contract under test is verified through a set of *invariants*.

This extended abstract summarizes the content of a journal paper [1], where the interested reader can find more details about SOCRATES and about its empirical validation.

A. Behaviours

Each bot realizes a specific behaviour or a combination of predefined behaviours, which allow the bot to generate inputs for contract transactions, according to a collection of configurable strategies. SOCRATES comes with 4 built-in behaviours: *Random*, *Boundary*, *Overflow* and *Combined*.

Moreover, the framework can be extended with domain specific behaviours that can realise contract specific execution strategies.

B. Invariants

The testing oracles supported by SOCRATES take the form of contract invariants. Six built-in invariants are available for testing a given smart contract. This collection includes one generic invariant (related to integer overflow) and five invariants that are specific to the protocol required by the EIP20 interface, i.e., one of the most prominent interfaces that smart contracts implement when they need to use tokens.

Moreover, SOCRATES is extensible and supports the definition of new (contract-specific) invariants. As a matter of fact, in our empirical validation, we defined and assessed six contract-specific invariants to test real Ethereum smart contracts.

III. Empirical Validation

Our experimental assessment shows that SOCRATES is effective at revealing faults found in real-world smart contracts that are deployed in production in the official Ethereum blockchain. The considered smart contracts are actively used, with actual transactions associated taking place and involving real monetary value. SOCRATES was used to test 1,905 real smart contracts coming from the official Ethereum blockchain. It could detect 148 true invariant violations, with only 32 false alarms. Moreover, when compared with the state of the art smart contract fuzzer Echidna[1], SOCRATES could identify substantially more true invariant violations.

IV. Related Work

Most existing tools for smart contract testing [2], [3], [4] are focused on security issues (e.g., reentrancy vulnerability) rather than functional invariants (e.g., each successful token transfer should log a *Transfer* event) that contracts are supposed to ensure. Correspondingly, they can detect only violations to security properties associated with known vulnerabilities. No general, extensible framework exists for functional testing of smart contracts and the most related tool, Echidna, is purely random and does not support the composition of a set of (extensible) bot behaviors. Moreover, Echidna does not include pre-defined invariants to classify the outcome of contract execution and to detect whether a defect has been exposed.

V. Conclusion

Once persistent in the blockchain, smart contracts cannot be updated and programming defects cannot be patched. So, accurate and thorough testing of smart contracts is very important to detect programming mistakes before deployment, when they can still be fixed.

SOCRATES is an automated testing framework for smart contracts. It is based on a federated society of bots, that are meant to test the potentially intricated interactions among the diverse roles defined for a contract. Our empirical assessment shows that our solution is effective in spotting programming defects that break one or more contract invariants, even in smart contracts that are deployed and run in the official blockchain, and deal with actual monetary value. Moreover, the collected experimental evidence shows that relying on a society of bots, rather than on a single bot behaviour, is a crucial feature to accurately test smart contracts.

For a more complete presentation of SOCRATES, the reader is invited to refer to the original journal publication [1].

[1]Echidna https://github.com/trailofbits/echidna

References

[1] E. Viglianisi, M. Ceccato, and P. Tonella, "A federated society of bots for smart contract testing," *Journal of Systems and Software*, p. 110647, 2020.

[2] N. Grech, "MadMax: Surviving out-of-gas conditions in Ethereum smart contracts," in *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA)*, 2018.

[3] B. Jiang, Y. Liu, and W. Chan, "ContractFuzzer: Fuzzing smart contracts for vulnerability detection," in *Proceedings of the International Conference on Automated Software Engineering (ASE)*, 2018.

[4] L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor, "Making smart contracts smarter," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2016, pp. 254–269.