Human abstractness may make smart contracts smarter, researchers report

September 14 2023, by Mary Fetzer

Credit: Unsplash/CC0 Public Domain

Smart contracts, or computer programs that automatically execute certain agreed-upon actions when agreed-upon conditions are met, are considered safer for online transactions than traditional contracts, but

they are not error-proof. Researchers from the Penn State College of Information Sciences and Technology (IST), as part of a multi-institution effort, developed an end-to-end model-based framework in place of traditional programming code to make smart contracts easier to develop, easier to verify and, ultimately, safer to use.

They published their proposal in *IEEE Transactions on Dependable and Security Computing*.

"As with most software, the code used to program smart contracts is prone to error and vulnerabilities," said Aron Laszka, assistant professor in the College of IST and lead researcher on the project. "Our project focused on the significant technical challenges involved with verifying whether that code does what it was intended to do, especially when interacting with other smart contracts."

Smart contracts are stored on blockchain platforms, similar to those used to store virtual currency like Bitcoin. According to Laszka, the blockchain platform is intended to make smart contracts—which often handle assets of considerable value—more secure from tampering. But while the platform guarantees the smart contract will execute correctly, it does not verify that the code of the contract is correct.

When the predetermined conditions of a smart contract are met, a specific action is executed on a blockchain and updated so the transaction cannot be changed. But when the smart contract does not behave as expected, determining the problem can be challenging, according to the researchers.

"It's challenging to verify smart contracts that were manually written using programming language," he said. "Software bugs may not be detected until after the smart contract has been deployed, at which point it can be exploited."

Laszka offered the example of an online auction. The requirements written into the auction code make it so that once the auction has closed, no further bids can be placed. When deployed, however, the auction allows the highest bidder to be replaced after closing. Post-deployment verification tools may determine that the instruction—the programming language—is wrong, but they do not precisely indicate where the problem lies or what programmers need to do fix it.

Laszka pointed to security breaches over recent years—attackers maliciously extracting assets from smart contracts or destroying the contracts entirely—as evidence that developers need more efficient verification tools to ensure that a smart contract will fulfill its requirements.

"Across academia and industry, there are a lot of verification tools for programming language and machine code, and there are companies that can be hired to perform contract audits," Laszka said. "But the feedback provided by these tools and services can be low-level and not necessarily useful."

According to Laszka, incidents such as security breaches often exploit the interaction among multiple smart contracts, but prior research on smart contract verification, vulnerability discovery and secure development typically considers only individual contracts in isolation.

"To address this gap, we introduced a framework, which we call VeriSolid, for the formal verification of contracts using an abstract-state machine-based model that executes the contract exactly as prescribed," Laszka said. "This approach enables developers to think about and verify the behavior of a set of interacting contracts at a high level of abstraction."

According to the researchers, this change begins at the development

stage. A high-level abstract model would enable developers to express in a simple, user-friendly manner how the contract should work.

"We believe it's easier for humans to work with abstract concepts than with lines of programming language code," Laszka said. "If verification tools within the model find that something is wrong, we can provide feedback at this higher level of abstraction to identify the problem."

In the case of the online auction, the model's verification feedback would lead developers directly to the problem: the highest bidder changed because the bidding functionality is still available after the auction has closed.

"With our proposed model, the smart contract can be verified before deployment," Laszka said. "Further, the tools can actually generate source code from the model to be deployed on the blockchain as if the developer had written it manually in programming language."

The researchers used VeriSolid to generate Solidity code—a programming language for implementing smart contracts on blockchain platforms.

"This code is functionally and behaviorally equivalent to verified models, enabling the creation of correct-by-design smart contracts," Laszka said. "Additionally, we introduced a graphical notation, called deployment diagrams, for specifying possible interactions between contract types."

This positioned the researchers to present a framework for the automated verification, generation and deployment of contracts that conform to a deployment diagram.

"The high-level model form allows developers to specify desired

properties—for both standalone and interacting smart contracts—in a way they are unable to do with low-level programming language," Laszka said. "In addition, we synchronize verification and deployment as a common framework, allowing a contract to be published on a blockchain network once verified."

More information: Keerthi Nelaturu et al, Correct-by-Design Interacting Smart Contracts and a Systematic Approach for Verifying ERC20 and ERC721 Contracts With VeriSolid, *IEEE Transactions on Dependable and Secure Computing* (2022). DOI: 10.1109/TDSC.2022.3200840

Provided by Pennsylvania State University