VERISOLID: Correct-by-Design Smart Contracts for Ethereum

Anastasia Mavridou¹, <u>Aron Laszka²</u>, Emmanouela Stachtiari³, Abhishek Dubey⁴

- ¹ NASA Ames
- ² University of Houston
- ³ Aristotle University of Thessaloniki
- ⁴ Vanderbilt University

Smart Contracts on Blockchains

- Smart contract: general purpose computation on a blockchain (or other distributed ledger) based computational platform
- Recently popularized by Ethereum
 - smart contracts may be developed using high-level languages, such as Solidity
 - enables the creation of decentralized applications
- Envisioned to have a wide range of applications
 - financial (self-enforcing contracts)
 - Internet of Things

. . .

decentralized organizations



insecurity Smart Contract Security in Practice

- Notable incidents (amounts vary over time with variations in exchange rate)
 - The DAO attack: ~\$500 million taken
 - Parity wallet freeze: ~\$70 million frozen
 - Parity wallet hack: ~\$21 million taken
- Recent analysis: 34,200 contracts (out of 1M publicly deployed contracts) have security issues / vulnerabilities¹
- Distributed ledgers are immutable by design
 - smart contract vulnerabilities cannot be patched*
 - erroneous (or malicious) transactions cannot be reverted*

* without undermining the trustworthiness of the contract / ledger

¹ Ivica Nikolic, Aashish KolluriChu, Ilya Sergey, Prateek Saxena, and Aquinas Hobor, "*Finding the greedy, prodigal, and suicidal contracts at scale*," ACSAC'18.

Securing Smart Contracts

- Vulnerabilities often arise due to semantic gap
 - difference between assumptions that developers make about execution semantics and the actual semantics
 - Solidity resembles JavaScript, but it does not work exactly like
- Existing approaches
 - design patterns, e.g., Checks-Effects-Interactions
 - tools for finding (typical) vulnerabilities
 - OYENTE
 - MAIAN
 - •
 - tools for verification and static analysis
 - SECURIFY
 - RATTLE
 - ...

Contract Vulnerability Discovery and Verification



Correct-by-Design Contract Development



- Advantages of model-based approach
 - specification of desired properties with respect to a high-level model
 - providing feedback to developer with respect to a high-level model

VERISOLID: Correct-by-Design Smart Contracts





VERISOLID Model

Formal, transition-system based language for contracts



each contract may be represented as a transition system

Definition: A smart contract is a tuple $(D, S, S_F, s_0, a_0, a_F, V, T)$

- D custom data types and events
- S states
- $S_F \subset S$ final states
- $s_0 \in S$ initial state
- $a_0 \subset \mathbb{S}$ initial action
- $a_F \oplus fallback$ action
- V contract variables
- T transitions (names, source and destination states, guards, actions) parameter and return types)

implemented as functions in the generated code ⁹

S: subset of Solidity statements

Example Model: Blind Auction Contract as a Transition System



VERISOLID Semantics

- We define semantics in the form of Structural Operational Semantics
- Basic transition rule:

$$\begin{aligned} t \in T, \quad name = t^{name}, \quad s = t^{from} \\ M = Params(t, v_1, v_2, \ldots), \quad \sigma = (\Psi, M) \\ & \text{Eval}(\sigma, g_t) \rightarrow \langle (\hat{\sigma}, N), \text{true} \rangle \\ & \langle (\hat{\sigma}, N), a_t \rangle \rightarrow \langle (\hat{\sigma}', N), \cdot \rangle \\ & \hat{\sigma}' = (\Psi', M'), \quad s' = t^{to} \\ \hline & \langle (\Psi, s), name(v_1, v_2, \ldots) \rangle \rightarrow \langle (\Psi', s', \cdot) \rangle \end{aligned}$$

- transition t changes ledger state from Ψ to Ψ' and contract state from s to s'
- We also define semantics for erroneous transitions (e.g., exceptions) and for supported Solidity statements $\mathbb S$
- Transitions work "as expected" from a transition system *

* with Solidity-related additions, such as exceptions and fallback functions

VERISOLID Verification

- Instead of searching for vulnerabilities, we verify that a model satisfies desired properties that capture correct behavior
- Deadlock freedom: contract cannot enter a non-final state in which there are no enabled transitions
- Safety and liveness properties
 - specified using Computational Tree Logic (CTL)
 - we provide several CTL templates to facilitate specification



where **X** and **Y** can be transitions or statements

• example:

bid cannot happen after close







VERISOLID Verification Process

- First, transform a contract into an augmented transition system, which captures behavior using transitions
 - based on the formal operational semantics of supported Solidity statements



Augmented transition withdraw

VERISOLID Verification Process

- First, transform a contract into an augmented transition system, which captures behavior using transitions
 - based on the formal operational semantics of supported Solidity statements

Theorem: The original contract and the corresponding augmented transition system are observationally equivalent.

- Second, transform an augmented transitions system into an observationally-equivalent Behavior-Interaction-Priority (BIP) model
- Over-approximation of contract behavior
 - satisfied safety properties are **satisfied** by the actual contract
 - violated liveness properties are violated by the actual contract
- Verification using nuXmv model checker

satisfied properties + violated properties (with violating transition traces)

Verification Examples



VERISOLID Framework

- Generate equivalent Solidity code from VERISOLID contract models
 - based on the formal operational semantics of transitions
- VERISOLID IDE



- integrated graphical and text editor
- integrated verification and code generation

Conclusion

- VERISOLID advantages
 - high-level model with **formal semantics** (which are familiar to most developers)
 - verification of **desired behavior** (instead of searching for typical vulnerabilities)
 - high-level feedback to the developer (for violated properties)
 - Solidity **code generation** (instead of error-prone coding)

Source code: <u>http://github.com/anmavrid/smart-contracts</u>

Live demo at: <u>http://cps-vo.org/group/SmartContracts</u> (requires free registration) Thank you for your attention!

Questions?

aronlaszka.com alaszka@uh.edu

