Symbolic Value-Flow Static Analysis

Symbolic Value-Flow Static Analysis: Deep, Precise, Complete Modeling of Ethereum Smart Contracts

Pavel Šimovec

468914@mail.muni.cz

Faculty of Informatics, Masaryk University

November 19, 2021

Yannis Smaragdakis et al. *Symbolic Value-Flow Static Analysis: Deep, Precise, Complete Modeling of Ethereum Smart Contracts (Artifact)*. Sept. 2021. DOI: `10.5281/zenodo.5494813`. URL: `https://doi.org/10.5281/zenodo.5494813`

Symvalic analysis

Symvalic analysis

- a precise, path sensitive static analysis

Symvalic analysis

- a precise, path sensitive static analysis
- mixes values and symbolic expressions

Symvalic analysis

- a precise, path sensitive static analysis
- mixes values and symbolic expressions
- scalable precision through dependencies

Symvalic analysis

- a precise, path sensitive static analysis
- mixes values and symbolic expressions
- scalable precision through dependencies
- applied to ethereum smart contracts

Why smart contracts?

Why smart contracts?

- correctness is crucial

Why smart contracts?

- correctness is crucial
- code and execution of high value contracts is publicly available

Why smart contracts?

- correctness is crucial
- code and execution of high value contracts is publicly available
- code is compact

Success of symvalic analysis

Success of symvalic analysis

- authors were able to find 6 major security vulnerabitilites

Success of symvalic analysis

- authors were able to find 6 major security vulnerabitilites
- \$350k in bounties

Precision/completeness of common approaches

Precision/completeness of common approaches

- symbolic execution - precise but incomplete

Precision/completeness of common approaches

- symbolic execution - precise but incomplete
- static analysis approaches can be complete but imprecise

Example - symbolic analysis

Example - symbolic analysis

```
address admin ;  // set up at construction , not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {

  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;


  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, " only admin ");
}
```

adjusted	token	amount	sendTo	admin
1030	0x6f..	1000	0x3f6..	0x5a1..

Example - value-flow static analysis

Example - value-flow static analysis

```
address admin ;  // set up at construction , not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {

  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;


  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, " only admin ");
}
```

adjusted
6180
1030
515

amount
1000
500
30

token
0x6f..
owner
user
0xe22..

Symvalic analysis adds dependencies

Symvalic analysis adds dependencies

```
address admin ;  // set up at construction , not in contract code

function withdrawToken (IERC20 token, uint256 amount, address sendTo) external {

  onlyAdmin();

  uint256 adjusted = amount * 103 / 100;


  if (amount >= 10000 && amount < 100000)
    token.transfer(sendTo,adjusted) ; …
}

function onlyAdmin() internal view {
  require (msg.sender == admin, " only admin ");
}
```

adjusted	amount	token
6180	1000	0x6f..
1030	500	owner
515	30	user
		0xe22..

How does symvalic analysis work?

How does symvalic analysis work?

- datalog-based analysis rules

How does symvalic analysis work?

- datalog-based analysis rules
- top-down reasoning - solving equations

How does symvalic analysis work?

- datalog-based analysis rules
- top-down reasoning - solving equations
- bottom up reasoning, up to bounded expression size

Completeness and soundness of symvalic analysis

Completeness and soundness of symvalic analysis

- neither sound nor complete

Completeness and soundness of symvalic analysis

- neither sound nor complete
- aim for a good balance between completeness and precision

Completeness and soundness of symvalic analysis

- neither sound nor complete
- aim for a good balance between completeness and precision
- incompleteness - overapproximation, but prefers concrete values (in favor to precision over completeness) not guaranteed to model all values in real execution, but values that analysis considers are likely to be realizable

Completeness and soundness of symvalic analysis

- neither sound nor complete
- aim for a good balance between completeness and precision
- incompleteness - overapproximation, but prefers concrete values (in favor to precision over completeness) not guaranteed to model all values in real execution, but values that analysis considers are likely to be realizable
- precision - aim to get high precision without losing scalability avoiding state explosion symvalic analysis computes dependencies only on small subset of variables, therefore the analysis can be imprecise (false positives)

Domains of analysis

Domains of analysis

- $t, u, v \in V$, a set of variables,

Domains of analysis

- $t, u, v \in V$, a set of variables,
- $\mathit{fun} \in F$, a set of functions,

Domains of analysis

- $t, u, v \in V$, a set of variables,
- $\mathit{fun} \in F$, a set of functions,
- $i, j \in I$, a set of instruction labels,

Domains of analysis

- $t, u, v \in V$, a set of variables,
- $fun \in F$, a set of functions,
- $i, j \in I$, a set of instruction labels,
- $n \in N$, the set of natural numbers.

Instruction set

Instruction	*Operand Types*	*Description*
$i :$ `v = t` $\odot$ `u`	$I \times V \times V \times V$	Binary operations
$i :$ `v =` $\phi(u_i)$	$I \times V \times V^n$	Phi instructions
$i :$ `v =` $[\![$`u`$]\!]$	$I \times V \times V$	Loads
$i :$ $[\![$`v`$]\!]$ `= u`	$I \times V \times V$	Stores
$i :$ `jumpif v` j	$I \times V \times I$	Conditional jumps
$i :$ `fun`(u_j)	$I \times F \times V^n$	Calls

Notation used in rules

Notation	Description
$\textsc{Def}(\mathtt{fun}(\bar{\mathtt{a}})\mathtt{:i})$	Function fun is defined with formal argument vector $\bar{\mathtt{a}}$ and first instruction i.
$i \xrightarrow{next} j$	instruction i has j as a possible next.
$\textsc{Ok}(D_1 \oplus D_2 \ldots)$	Dependencies combination is valid (no conflicting dependencies for same variable).
$\mathsf{v} \rightarrow e\ \langle D \rangle$	Variable v may hold symbolic expression e under dependencies D.
$[\![a]\!] \Rightarrow e$	Storage location a (a symbolic expression) may have contents e.
$\lvert i \rvert\ \langle D \rangle$ or $\lvert i \rvert\ \langle d^L; d^T \rangle$	Instruction i is reachable with dependencies D. (Expanded: local deps. d^L, transaction deps. d^T.)
$\textsc{Oracle}(\mathsf{v}) = e$	The symbolic solver (or default logic) suggests value e for external arg./environment variable v.
$\textsc{normalize}(e) = e_0$	Expression e normalizes (simplifies) to e_0.

Analysis rules

$$(\textsc{Next})\ \frac{|i|\ \langle D\rangle \qquad i \xrightarrow{next} j \qquad \neg(i:\ \texttt{jumpif} * *)}{|j|\ \langle D\rangle}$$

$$(\textsc{JumpIf-T})\ \frac{|i:\ \texttt{jumpif v}\ j|\ \langle D_i\rangle \qquad \texttt{v} \rightarrow \texttt{true}\ \langle D_v\rangle}{|j|\ \langle D_i \oplus D_v\rangle}$$

$$(\textsc{JumpIf-F})\ \frac{|i:\ \texttt{jumpif v}\ j|\ \langle D_i\rangle \qquad \texttt{v} \rightarrow \texttt{false}\ \langle D_v\rangle \qquad i \xrightarrow{next} k \qquad k \neq j}{|k|\ \langle D_i \oplus D_v\rangle}$$

$$(\textsc{BinaryOp})\ \frac{|i:\ \texttt{v} = \texttt{t} \odot \texttt{u}|\ \langle D_i\rangle \qquad \texttt{t} \rightarrow e_t\ \langle D_t\rangle \qquad \texttt{u} \rightarrow e_u\ \langle D_u\rangle}{\texttt{v} \rightarrow \textbf{NORMALIZE}(e_t \odot e_u)\ \langle D_i \oplus D_t \oplus D_u\rangle}$$

$$(\textsc{Phi})\ \frac{|i:\ \texttt{v} = \phi(\ldots\texttt{u}\ldots)|\ \langle D_i\rangle \qquad \texttt{u} \rightarrow e\ \langle D_u\rangle}{\texttt{v} \rightarrow e\ \langle D_i \oplus D_u\rangle}$$

Analysis rules

$$(\textsc{Load})\quad \frac{|i:\ \mathsf{v} = [\![\mathsf{u}]\!]|\ \langle {d_i}^L; {d_i}^T\rangle \qquad \mathsf{u} \to e_u\ \langle {d_u}^L; {d_u}^T\rangle \qquad [\![e_u]\!] \Rightarrow e}{\mathsf{v} \to e\ \langle {d_i}^L \oplus {d_u}^L \oplus [\mathsf{v} \to e]; {d_i}^T \oplus {d_u}^T\rangle}$$

$$(\textsc{Store})\quad \frac{|i:\ [\![\mathsf{v}]\!] = \mathsf{u}|\ \langle D_i\rangle \qquad \mathsf{v} \to e_v\ \langle D_v\rangle \qquad \mathsf{u} \to e_u\ \langle D_u\rangle \qquad \textbf{Ok}(D_i \oplus D_v \oplus D_u)}{[\![e_v]\!] \Rightarrow e_u}$$

$$(\textsc{Call})\quad \frac{|i:\ \mathsf{fun}(\bar{\mathsf{u}})|\ \langle {d_i}^L; {d_i}^T\rangle \qquad \forall\ j\colon \mathsf{u}_j \to e_j\ \langle {d_j}^L; {d_j}^T\rangle, \textbf{Ok}({d_i}^L \oplus {d_j}^L) \qquad \textbf{Def}(\mathsf{fun}(\bar{\mathsf{a}}) : l)}{|l|\ \langle \bigoplus_k [\mathsf{a}_k \to e_k]; \bigoplus_k {d_k}^T \oplus {d_i}^T\rangle \qquad \forall k : \mathsf{a}_k \to e_k\ \langle [\mathsf{a}_k \to e_k]; \varnothing\rangle}$$

$$(\textsc{External-Args})\quad \frac{\textbf{Def}(\mathsf{fun}(\bar{\mathsf{a}}) : *) \qquad \textbf{Oracle}(\mathsf{a}_k) = e_k}{\mathsf{a}_k \to e_k\ \langle [\mathsf{a}_k \to e_k]; \varnothing\rangle}$$

$$(\textsc{Sender})\quad \frac{\textbf{Oracle}(\mathsf{sender}) = e}{\mathsf{sender} \to e\ \langle \varnothing; [\mathsf{sender} \to e]\rangle}$$

MUNI
FACULTY
OF INFORMATICS