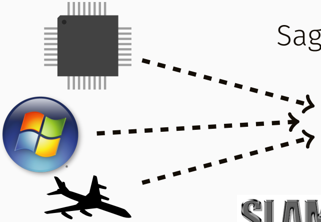
Not All Bugs Are Created Equal, But Robust Reachability Can Tell The Difference

Guillaume Girol[1] Benjamin Farinier[2] Sébastien Bardin[1]

[1]CEA, List, Université Paris-Saclay, France

[2]TU Wien, Vienna, Austria

Formal Verification



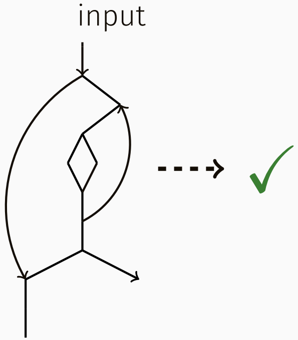
Sage

Infer

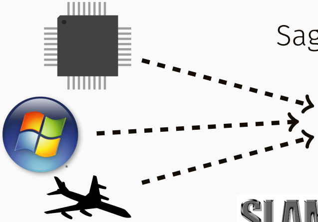
Java PathFinder

Astrée

CBMC



Formal Verification



Sage

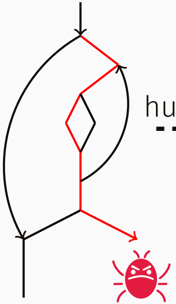
Infer

Java PathFinder

Astrée

CBMC

bad input

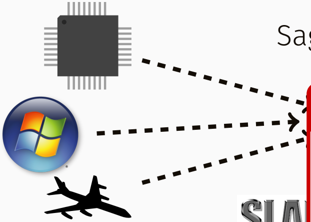


human

Fix

```
 ptr = foo();
-ptr->bar();
+if (ptr) ptr->bar();
 return;
```

Formal Verification



Frama-C Software Analyzers

Sage

SLAM

KLEE

Infer

Java PathFinder

Astrée

CBMC

bad input

Core concept

Reachability

Fruitful since the 70's

Fix

```
ptr = foo();
r->bar();
(ptr) ptr->bar();
turn;
```

Problem 1 with reachability in bug finding

The number of issues found can be overwhelming

Prioritisation?

Problem 2 with reachability: false positives in security-oriented bug finding

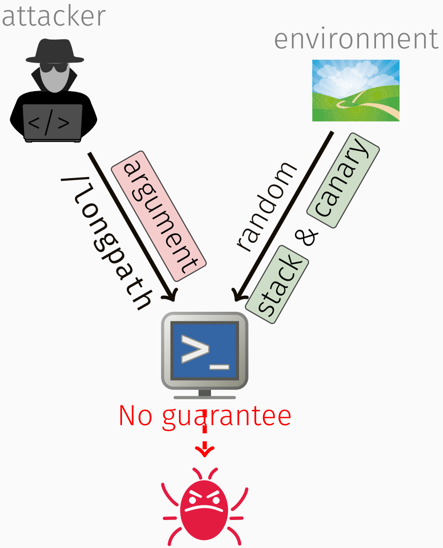
What reachability tells us: one bad input

CVE-2019-20839 is triggered whenever

- the attacker passes argument `/longpath`
- the stack canary is `0x01010180`
- stack starts at `0xfff00000`

Real life false positives

Formally reachable, but
in reality, cannot be triggered **reliably**



False positives in practice

- Randomisation-based protections (stack canaries, ASLR, …)

  Bug only works for the right randomness

- Bugs involving uninitialized memory

  Bug only works for the right initial memory

- Undefined behavior

  Even exists for compiled executables!

- Stubbing I/O or opaque functions with symbolic output

  Bug only works if the hash function is attacker-chosen

- Underspecified initial state

  Under-constrained symbolic execution

Our Goals

- A formal notion refining reachability without false positives
  - Focus effort on more severe bugs first
- Amenable to automated verification
  - Should be provable on compiled executables

Contributions

- Defining robust reachability, a way to draw a line between "reliably reachable" and "reachable but a false positive".
  - Comparison to Non-Interference, HyperLTL, ...
- Expanding Symbolic Execution and Bounded Model Checking to prove robust reachability
  - Standard optimisations (path pruning, concretisation) must be revisited
  - Path merging increases deduction power
- A prototype based on BINSEC, experimental evaluation and benchmark
  - New insight on the exploitability of 4 CVEs
  - Reasonable overhead

functional available reusable CAV Artifact AEC Evaluated

Defining Robust Reachability

Choose a threat Model

Partition input into controlled input $\boxed{a}$ and uncontrolled input $\boxed{x}$

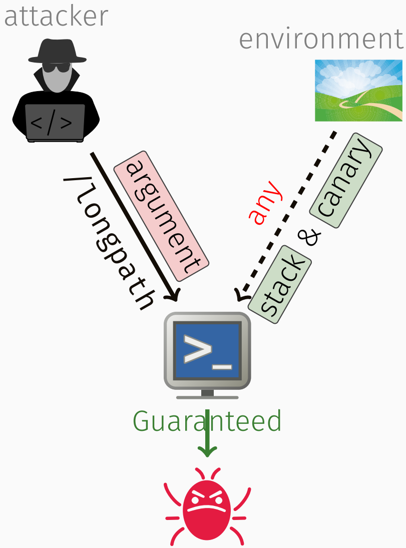
$(\boxed{a}, \boxed{x}) \vdash \ell$ means "with inputs $\boxed{a}$ and $\boxed{x}$, the program executes code at ℓ"

Reachability of location ℓ

$$\exists \boxed{a}, \boxed{x}. (\boxed{a}, \boxed{x}) \vdash \ell$$

Robust Reachability of ℓ

$$\exists \boxed{a}. \forall \boxed{x}. (\boxed{a}, \boxed{x}) \vdash \ell$$



Keeping it provable

Scope

No interactive systems Would require additional quantifier alternations

No quantitative approach Would require a new kind of model counters

(We tried briefly, and it looks prohibitively expensive)

Alternative formalisms (1): Non Interference

	Behavior does not depend on $\boxed{x}$	Implies reachability
Non Interference	for all $\boxed{a}$	no
Robust reachability	for a single $\boxed{a}$	yes

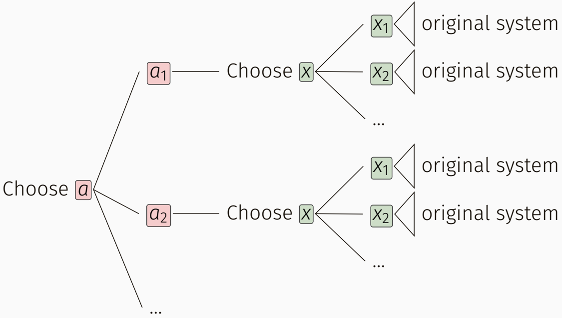
$$\text{Non-interference} + \text{Reachability} \underset{\not\Leftarrow}{\Rightarrow} \text{Robust Reachability}$$

Alternative formalisms (2)

As a hyperproperty, robust reachability is pure hyperliveness

- not a trace property (most studied case)
- not (k-)hypersafety ($\Rightarrow$ not solvable with self-composition)

Temporal logics: Expressible in CTL, HyperLTL, but no provers for generic programming languages



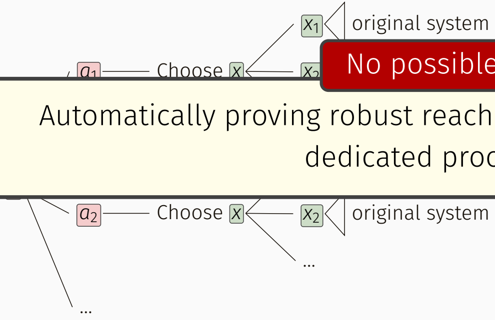
Robust reachability of ℓ in CTL:

$$\mathbf{EXAF}\ell$$

As a hyperproperty, robust reachability is pure hyperliveness

- not a trace property (most studied case)
- not (k-)hypersafety ($\Rightarrow$ not solvable with self-composition)

Temporal logics: Expressible in CTL, HyperLTL, but no provers for generic programming languages

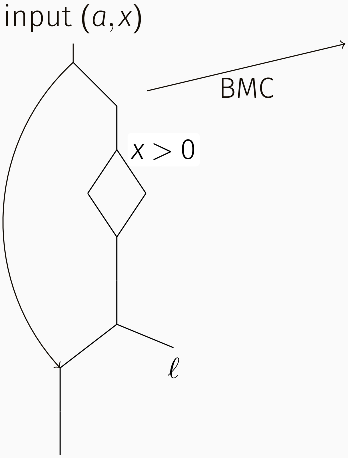


No possible reuse

Automatically proving robust reachability on programs requires a dedicated proof method

$\mathrm{EXAF}\ell$

Proving robust reachability: universally quantified SMT formulas



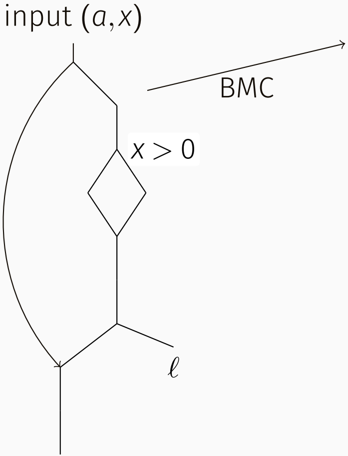
Unrolled program

$\varphi(a,x)$

Reachability

$\exists a,x.\varphi(a,x)$

Proving robust reachability: universally quantified SMT formulas



Unrolled program

$\varphi(a,x)$

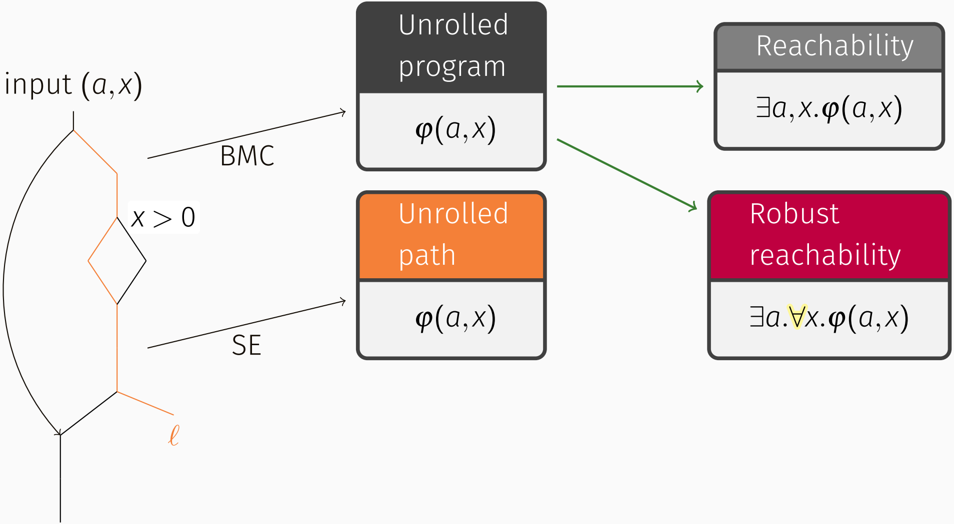
Reachability

$\exists a,x.\varphi(a,x)$

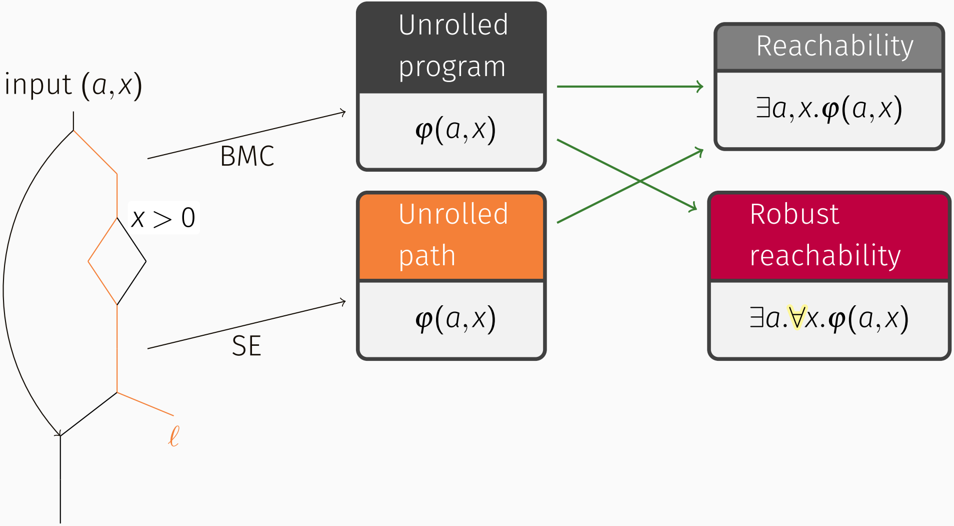
Robust reachability

$\exists a.\forall x.\varphi(a,x)$

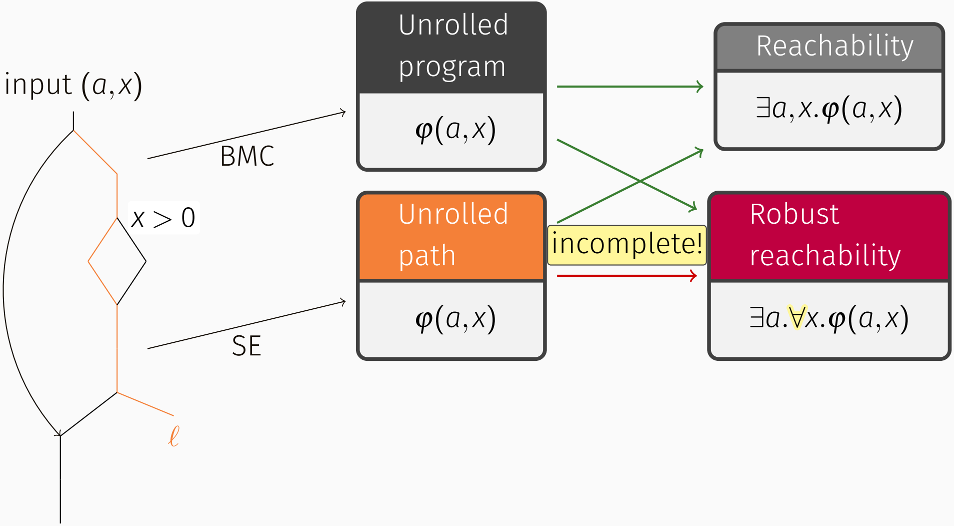
Proving robust reachability: universally quantified SMT formulas



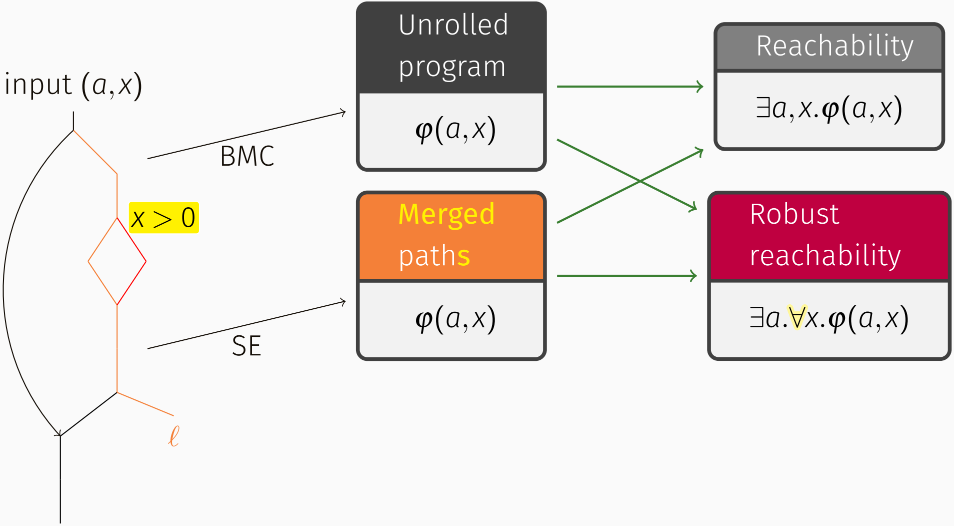
Proving robust reachability: universally quantified SMT formulas



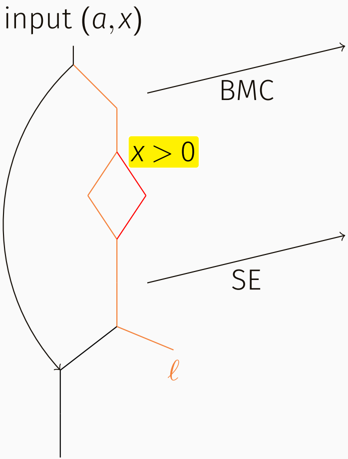
Proving robust reachability: universally quantified SMT formulas



Proving robust reachability: universally quantified SMT formulas



Proving robust reachability: universally quantified SMT formulas



Unrolled program

$\varphi(a,x)$

Merged paths

$\varphi(a,x)$

Reachability

$\exists a,x.\varphi(a,x)$

Robust reachability

$\exists a.\forall x.\varphi(a,x)$

Path merging

Optional in SE

Required for completeness in Robust SE

Proving robust reachability: other adaptions

assume ψ: $\exists a.\forall x.\psi \Rightarrow \phi$ instead of $\exists a.\forall x.\psi \land \phi$

path pruning: no extra quantifier (or lose completeness)

concretization: only works on controlled values

$$\exists \boxed{a}.\forall \boxed{x}.\varphi \xrightarrow[\boxed{x}\text{ to }90]{\text{concretize}} \exists \boxed{a}.\underbrace{\forall \boxed{x}.\boxed{x} = 90}_{\perp} \land \varphi$$

Other more advanced enhancements to SE probably also need to be revisited

Proof of concept implementation

- A binary-level **R**obust **SE** and **R**obust **BMC** engine based on BINSEC
- Discharges quantified SMT(arrays+bitvectors) formulas to Z3
- Evaluated against 46 reachability problems including CVE replays and CTFs

	BMC	SE	RBMC	RSE	RSE+$_{\text{merging}}^{\text{path}}$
Correct	22	30	32	37	44
False positive	**14**	**16**			
Inconclusive			1	7	
Resource exhaustion	10		13	2	2

Robust variants of SE and BMC

No false positives, more time-outs/memory-outs, 15% median slowdown

Case studies: 4 CVEs

CVE-2019-14192 in U-boot (remote DoS: unbounded `memcpy`) Robustly reachable

CVE-2019-19307 in Mongoose (remote DoS: infinite loop) Robustly reachable

CVE-2019-20839 in libvncserver (local exploit: stack buffer overflow)

- Without stack canaries: Robustly reachable
- With stack canaries: Timeout

CVE-2019-19307 in Doas (local privilege escalation: use of uninitialized memory)

- Doas = OpenBSD's equivalent of sudo
- Depends on the configuration file `/etc/doas.conf`
- **Use robust reachability in a more creative way**

CVE-2019-19307 in Doas: beyond attacker-controlled inputs

Reinterpret "controlled input" differently:

the attacker controls nothing, only executes

the sysadmin controls the configuration file: controlled input

the environment sets initial memory content *etc*: uncontrolled inputs

The meaning of robust reachability here

Are there configuration files which make the attacker win all the time?

Yes: for example typo "`permit ww`" instead of "`permit www`"

CVE-2019-19307 in Doas: beyond attacker-controlled inputs

Reinterpret “controlled input” differently:

the attacker controls nothing, only executes
the sysadmin controls the configuration file: controlled input
the environment sets initial memory content *etc*: uncontrolled inputs

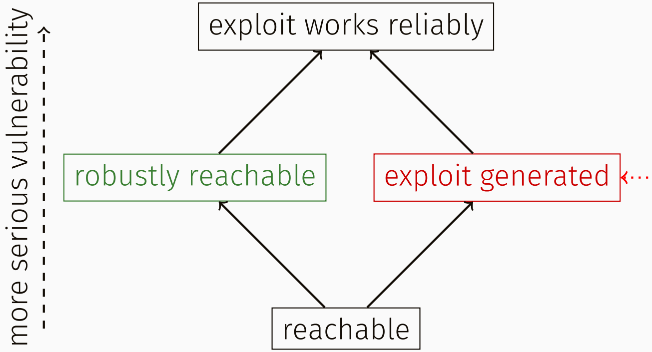
The meaning of robust reachability here

Are there configuration files which make the attacker win all the time?
Yes: for example typo “`permit ww`” instead of “`permit www`”

Versatility of Robust Reachability

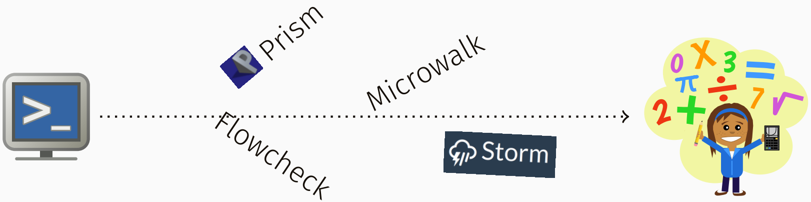
“Controlled inputs” are not limited to
“controlled by the attacker”

Related work (1): Approximating security-relevance



Automatic Exploit Generation
(Avgerinos *et al*, 2014)
(Heelan, 2013)

Related work (2): Quantitative approaches



Qualitative: less precise	**Quantitative:** slower
Model Checking	Probabilistic Model Checking
Non-Interference	Quantitative Information Flow
Robust Reachability	Future work?

A small experiment suggests solver queries would be orders of magnitude slower

Related work (3)

Flakiness (O'Hearn, 2019) Effort to get rid of tests with non deterministic outcomes: *particular case of non-robustness*

Fairness in Model Checking (Hart *et al.*, 1983) Same high-level idea: filter-out "uninteresting" behaviors

Higher order test generation (Godefroid, 2011) $\forall\exists$ queries to soundly approximate opaque functions (like hash functions) in Dynamic SE

Take Away

Standard reachability leads to false positives: bugs that are technically reachable, but unreproducible in practice

Robust reachability is a stronger property expressing that the attacker can reach the target reliably

Can be proved by variants of SE and BMC with reasonable overhead, but usual optimisations must be revisited

Source code: `https://github.com/binsec/cav2021-artifacts`
Precompiled artifacts: `https://zenodo.org/record/4721753`