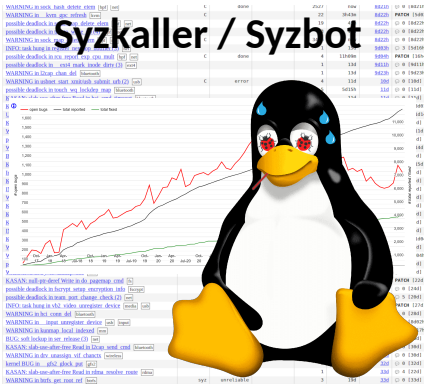
Attacker Control and Bug Prioritization

Guilhem Lacombe - Sébastien Bardin

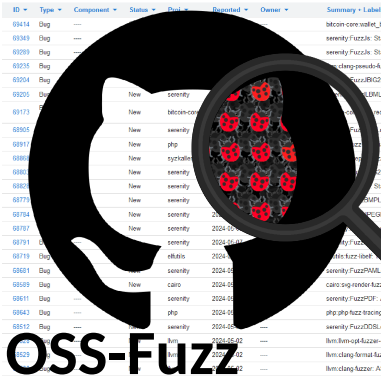
USENIX Security 2025

Automated bug-finding has become highly effective

fuzzing, symbolic execution, abstract interpretation...

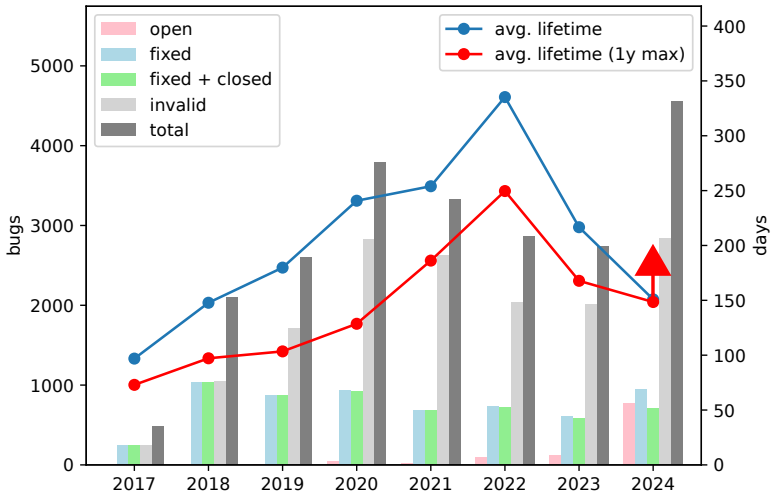


23k bugs, 1.5k still open



$>$ 10k vulns.

Bug-fixing cannot keep up!



Syzbot 2018→2023: **+47% post-discovery lifetime!**

Motivating example: *not all bugs are equally dangerous*

Vuln A: Not that dangerous?

size < 40

```
char buf[256];
...
if(size < 40)
    size -= 40;
memcpy(buf, msg, size);
```

size $\in [2^{64} - 40; 2^{64} - 1]$
$\Rightarrow$ crash

Vuln B: DANGER!!!

size > 256

```
char buf[256];
if(size > 296)
    size = 296;
...
memcpy(buf, msg, size);
```

size $\in [257; 296]$
$\Rightarrow$ ret. addr. overwrite

We need *efficient bug prioritization*

Fuzzing & CO

▼

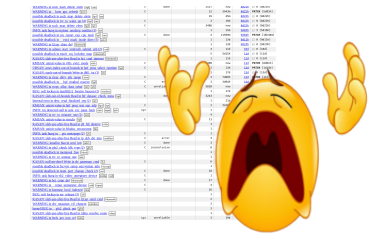
Crashing Inputs

▼

???

▼

Ranked Vulnerabilities



TO FIX		
OOB	+++	...
UAF	++	
OOB	++	
OOB	+	

Existing approaches are lackluster

	Automated	Correct	Explainable	
manual analysis	✗	✓?	✓	scalability?
vuln. type ⇒ threat level	✓	✗	✓	bug capabilities?
Auto. Exploit Gen.	✓	✓/ ✗	✓	genericity?
Machine Learning	✓	✗	✗	bug capabilities?

Core issue: no generic approach accounting for the capabilities granted to an attacker

Motivating example: *straightforward solutions are not good enough!*

Vuln A: Not that dangerous?

size < 40

```
char buf[256];
...
if(size < 40)
    size -= 40;
memcpy(buf, msg, size);
```

size $\in [2^{64} - 40; 2^{64} - 1]$
$\Rightarrow$ crash

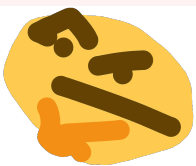
Vuln B: DANGER!!!

size > 256

```
char buf[256];
if(size > 296)
    size = 296;
...
memcpy(buf, msg, size);
```

size $\in [257; 296]$
$\Rightarrow$ ret. addr. overwrite

	Type	Tainted	# size
A	OOB write	✓	40
B	OOB write	✓	40



Our approach: *vulnerability evaluation based on Attacker Control*

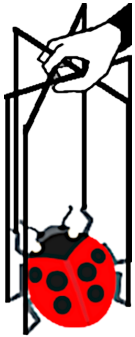
General approach

formal program analysis ⇒ evaluation of well-defined aspects of exploitability

Today's topic: **Attacker Control**

- theoretical framework + algorithms
- highly automated implementation in *Colorstreams*

- application to real-world vulnerabilities (40 CVEs across experiments)
  + correct analysis of CVE-2022-30790 (vs. original human analysis)



Current scope

What kinds of bugs are we looking at?

low-level memory and pointer corruptions

- typical "critical" fuzzer / ASAN bugs
- **prerequisites:** triggering input + type / ASAN report

Examples

- **pointer corruption:** p = pointer
- **buffer overflow:** p = offset, size, written data

Attacker Control: *intuitive definition*

Intuition
control = ability to obtain desired effects through inputs

Attacker Control: *intuitive definition*

~~Intuition~~ *Actually workable and generic concept*

control = ability to obtain ~~desired effects~~ *different values for p* through inputs

Key insight

- **quantitative aspect**: more is better
- **qualitative aspect:** $\neq$ values have $\neq$ threat levels $\Rightarrow$ need weights

Attacker Control: *core definition*

Domains of Control

The set of **feasible values**.

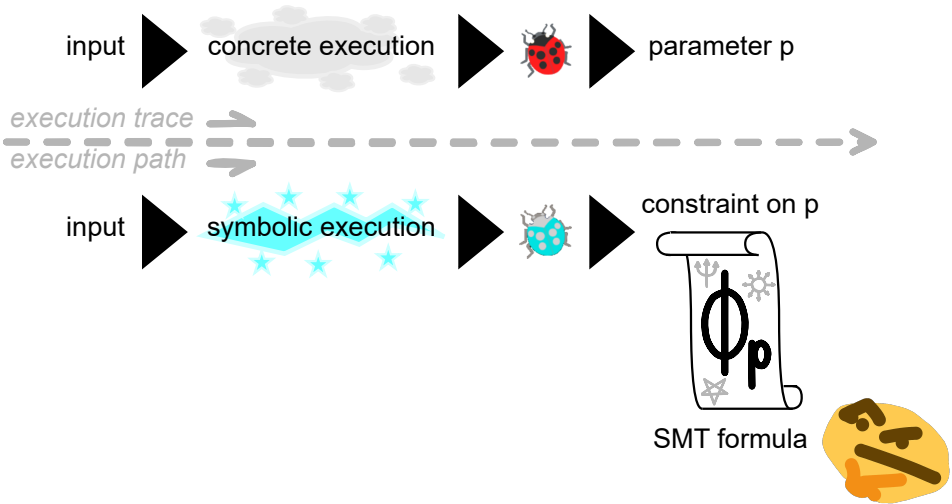
OOB sizes for our motivating example

- **Vuln. a:** $[2^{64} - 296; 2^{64} - 257]$
- **Vuln. b:** $[1; 40]$

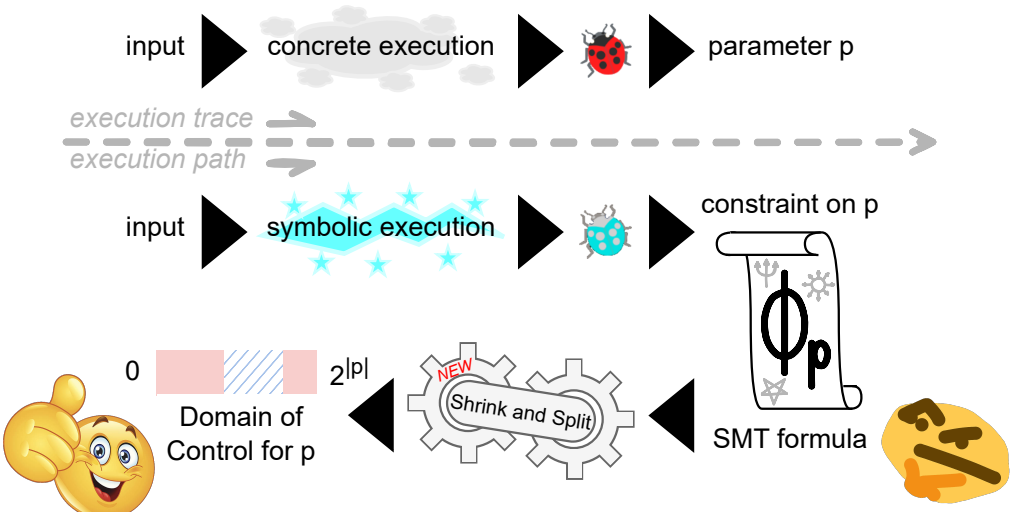
Useful for a human! But how to compute and score them automatically?

- ✗ domain propagation *(precision: over approx without under approx)*
- ✗ solution enumeration *(scalability: too slow)*

Computing domains of control with Shrink and Split: *overview*



Computing domains of control with Shrink and Split: *overview*



READY? START!

0 | *all possible values* | $2^{|p|}$

Computing domains of control with Shrink and Split: *algorithm*

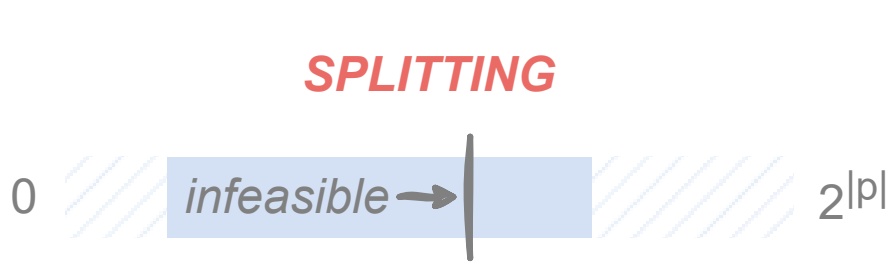
SHRINKING

Finding min / max values

$min/max_{v \in I}$ s.t. $\phi_p \wedge p = v$

how: Z3 min / max or binary search

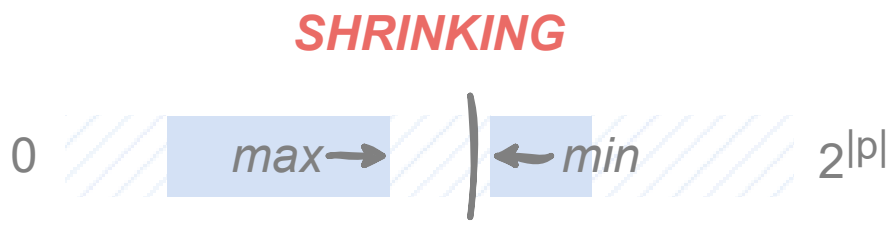
Computing domains of control with Shrink and Split: *algorithm*



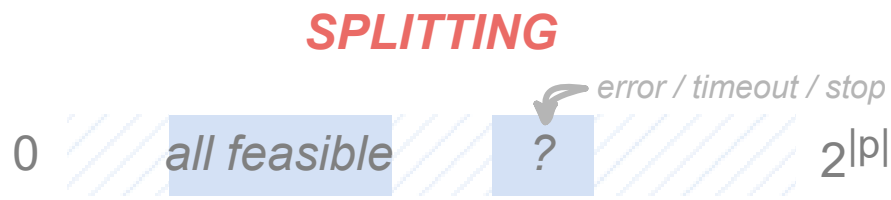
Finding an infeasible value

with quantifiers: $\exists\ v \in I$ s.t. $\forall\ x,\ \phi_p(x) \neq v$

Computing domains of control with Shrink and Split: *algorithm*



Computing domains of control with Shrink and Split: *algorithm*



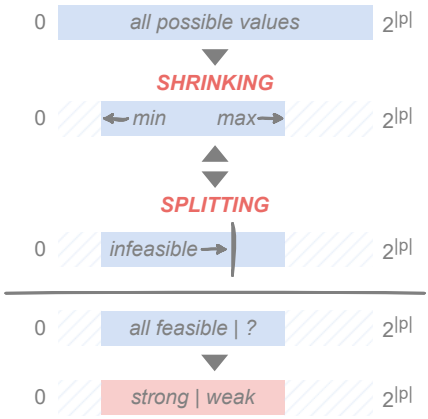
Proving all values are feasible

$\nexists\ v \in I$ s.t. $\forall\ x,\ \phi_p(x) \neq v$

$\iff$ infeasible value query returns unsat

Computing domains of control with Shrink and Split: *algorithm*

Computing domains of control with Shrink and Split: *algorithm*



Guarantees

- *strongly controlled intervals*
  $\Rightarrow$ **under-approx**
- *strongly and weakly controlled intervals*
  $\Rightarrow$ **over-approx**

What if the domain is full of holes?

Solutions:

- split limit
- regularity constraints (ex: *fixed bits*)

Computing scores from domains of control

Weighted Quantitative Control (wQC)

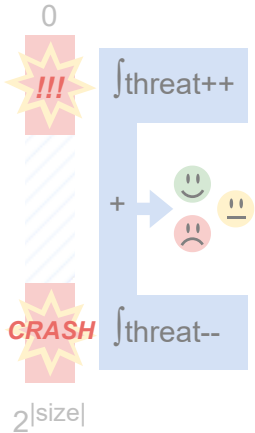
Sum of feasible value threat levels.

$\Rightarrow$ integrate a weight function $\omega(n)$ over the domain of control

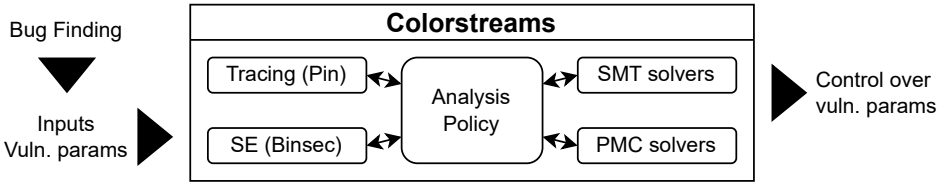
Motivating example

Bias toward local OOBs: $\omega : x \mapsto \frac{1}{ln(2)x}$

- **Vuln. A:** $wQC(oob\_size) \approx 2^{-58}$
- **Vuln. B:** $wQC(oob\_size) \approx 0.08$



Implementation: Colorstreams



- **open source release:** `https://github.com/binsec/colorstreams`
- precise + dynamic + binary-level
- SE with BINSEC
- tracing ⇒ single-path ⇒ performance
  multi-path behaviour: analyze multiple traces + models for library functions

Evaluation

Ground-truth benchmark

- 39 programs, 14 real-world vulnerabilities
- OOB reads & writes, use-after-frees, code pointer corruptions...
- **manually created / analyzed**

$\Rightarrow$ evaluate **precision**

Realistic benchmark

- 26 out-of-bounds vulns from the MAGMA fuzzing benchmark*
- OOB reads & writes
- **highly automated analysis**

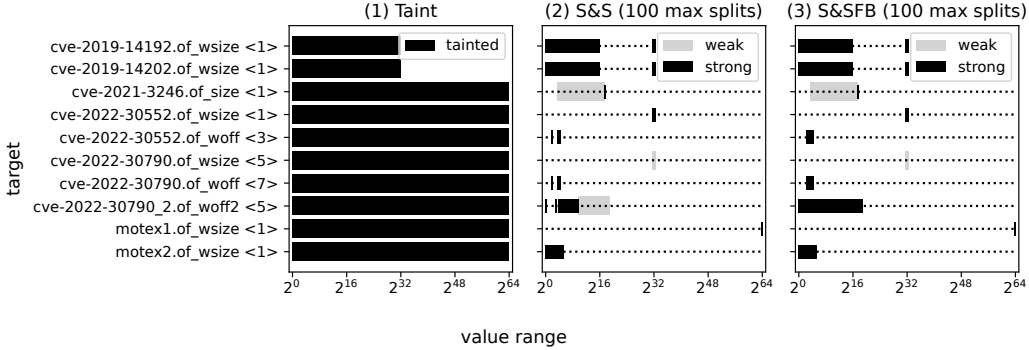
$\Rightarrow$ evaluate **practicality**

*https://hexhive.epfl.ch/magma/

Precision: *evaluating buffer OOB write vulnerabilities*

Type of vuln: out-of-bounds write
Parameters: offset, size (+ data)

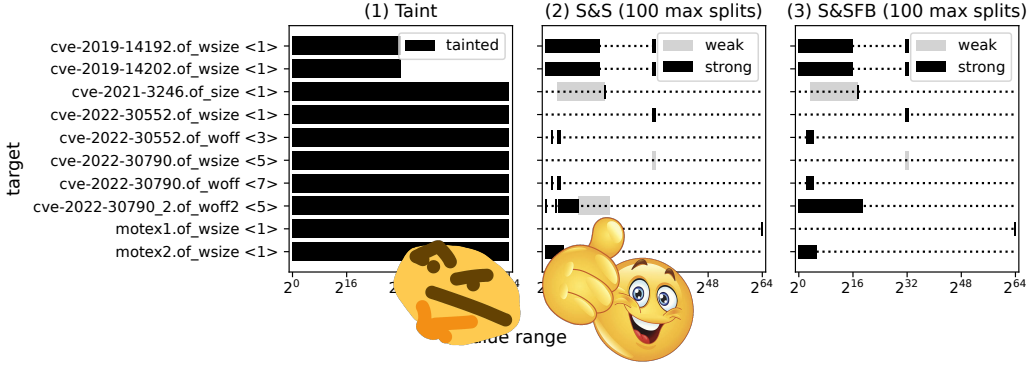
Most useful at a glance?



Precision: *evaluating buffer OOB write vulnerabilities*

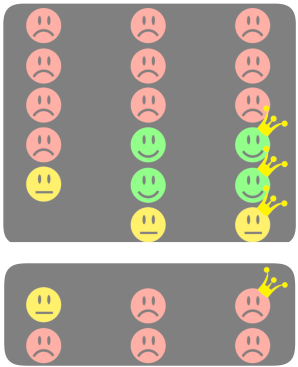
Type of vuln: out-of-bounds write
Parameters: offset, size (+ data)

Most useful at a glance?

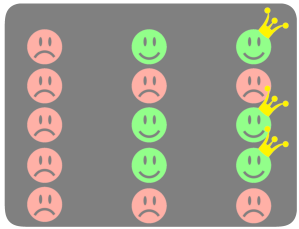


Precision: *differentiating different values makes a difference!*

Vulnerability	CVSS	Expect	Us
OOB writes			
cve-2021-3246			
cve-2019-14192			
cve-2019-14202			
cve-2022-30790			
cve-2022-30552			
cve-2022-30790-2			
OOB reads			
cve-2023-37837			
heartbleed			



Vulnerability	CVSS	Expect	Us
Code ptr corr.			
cve-2021-26567			
cve-2020-14393			
cve-2024-41881			
cve-2024-43700			
cve-2023-43338			



Precision: *improving human analysis*

Analysis of CVE-2022-30790 from human experts*

metadata corruption in linked list ⇒ arbitrary write

- does not look like arbitrary write...
- actually identical to CVE-2022-30552
- humans make mistakes???

(full code review in the paper)

*https://research.nccgroup.com/2022/06/03/technical-advisory-multiple-vulnerabilities-in-u-boot-cve-2022-30790-cve-2022-30552/

Practicality: *automatic analysis of the MAGMA vulnerabilities*

Automation based on taint

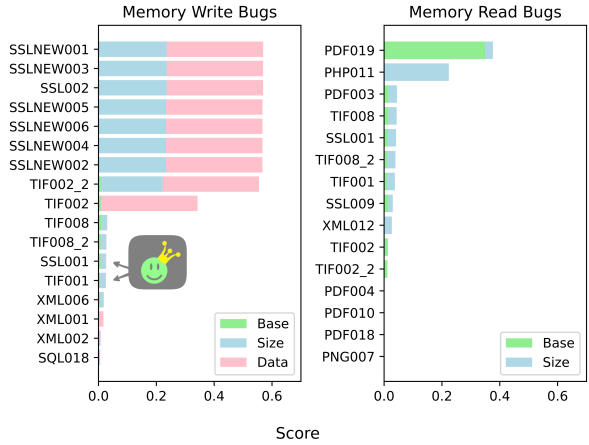
- ➤ symb. input selection
- ➤ OOB detection

⇒ no manual analysis
⇒ no manual instrumentation

Runtime

6m avg. / program (7s - 20m)

Close to fully automated prioritization of ASAN bugs!



Conclusion

Bug prioritization based on Attacker Control

- **domains of control** = set of obtainable values
- SE + **Shrink and Split** → precise computation of domains of control
- scoring with weighted Quantitative Control → qualitative + quantitative

⇒ automated prioritization of real-world bugs

Ongoing works

- further automation + fuzzing integration
- combining multiple paths / traces
- control + robustness*?
- more complex domains of control
- more vulnerability types

*Girol, G., Lacombe, G., Bardin, S., PLDI 2024

Thank you for your attention.
Any questions?

https://doi.org/10.5281/zenodo.14699098

open-source release

https://github.com/binsec/colorstreams

https://binsec.github.io/

Bonus: vulnerability capability awareness / evaluation in AEG

evocatio [CCS'22]

fuzzing to find new capabilities (ex: new OOB offset / size values)
but: no scoring or ranking

KOOBE [USENIX'20]

uses constraints on OOB offset / size to characterize capabilities
but: capability comparison

- $A > B \iff B$ constant and included in A
- $A = B \iff A$ identical to B

$\Rightarrow$ too simple for our needs!