Attacker Control and Bug Prioritization

(Work in progress)

Guilhem Lacombe [1,2]
Supervised by Sébastien Bardin [1,2]

[1]CEA LIST (LSL) [2]Université Paris-Saclay

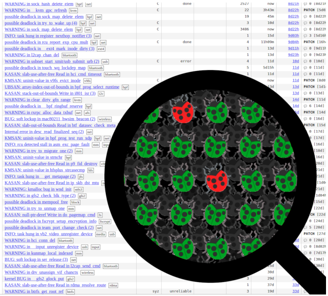
GT MFS Annual Meeting
Oléron 05/04/24

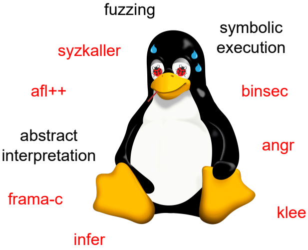
We find too many bugs!

Bugs are found faster than they can be fixed!

A concrete example: Syzbot[1]

- 24/7 fuzzing (mainly Linux)
- >4k since 2017
- ∼1k still open ↗





- developers cannot fix them all
- but not all of them are equally dangerous

[1] https://syzkaller.appspot.com/upstream

Motivating example

Vulnerability a

size < 40

```c
char buf[256];

if(size > 296)
      size = 296;
if(size < 40) // should be size > 40
      size -= 40;
memcpy(buf, msg, size);
```

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

$\Rightarrow$ crash

$\Rightarrow$ maybe not that dangerous

Vulnerability b

size > 256

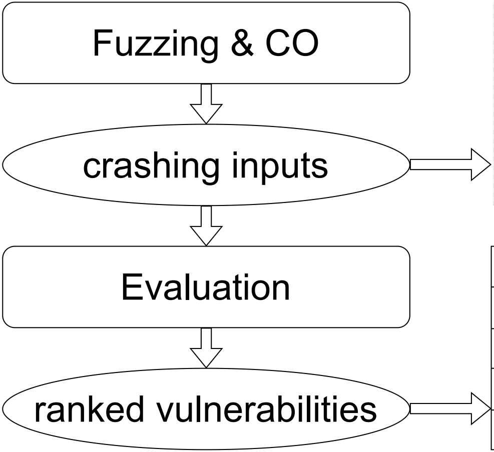
```c
char buf[256];

if(size > 296)
      size = 296;
if(size < 40) // should be size > 40
      size -= 40;
memcpy(buf, msg, size);
```

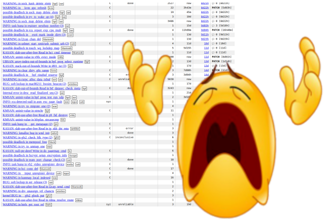
write size $\in [257; 296]$

$\Rightarrow$ return address overwritten

$\Rightarrow$ DANGER!!!

We need efficient bug prioritization





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

Existing approaches are lackluster

Approach	Pros	Cons
vulnerability type ⇒ threat level	+ easy + scalable	- imprecise (a and b are both OOB writes)
Automated Exploit Generation[1]	+ strong indicator (on success)	- lack of genericity - false negatives
AI[2]	+ scalable	- lack of transparency - focus on user reports
Robust Reachability[3]	+ reliability indicator	- not the full picture

⇒ **lack of formal methods research on this subject**

[1] Avgerinos et al., NDSS 2011
[2] Le et al., ACM Computing Surveys 2022
[3] Girol et al., CAV 2021

Goals and Challenges

Goals

- precise bug prioritization based on formal methods
- good-enough scalability
- fully automated

Goals and Challenges

Goals

- precise bug prioritization based on formal methods
- good-enough scalability
- fully automated

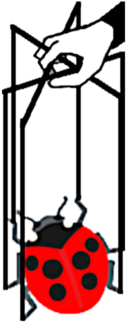
Challenges

- what is exploitability? non-exploitability?
- precision vs. genericity
- poor scalability of precise analysis techniques

Main proposition

Evaluate vulnerabilities based on **Attacker Control**

- the ability of attackers to obtain desired effects
- *without assuming their goals*



Our contributions

Exploration of formal definitions for control + algorithms

- *[new]* weak / strong control
- existing notions of quantitative information flow
  $\rightarrow$ quantitative control
- *[new]* domains of control

+ why taint analysis is not enough

Our contributions

Exploration of formal definitions for control + algorithms

- *[new]* weak / strong control
- existing notions of quantitative information flow
  $\rightarrow$ quantitative control
- *[new]* domains of control

\+ why taint analysis is not enough

Shrink and Split algorithm

measuring domains of control based on qualitative notions

- more scalable than counting
- more nuanced results

\+ promising experiments on real-world vulnerabilities

Introduction

Defining Attacker Control

Algorithms

Implementation and Experiments

Conclusion

Intuitive definition

example: $v \sim$ buffer overflow size

What does attacker control over v mean?

Intuitive definition

example: $v \sim$ buffer overflow size

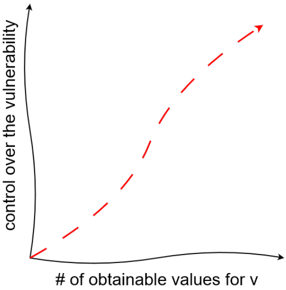
What does attacker control over v mean?

> **Intuition**
>
> control = ability to obtain desired values

more obtainable values

$\Rightarrow^?$ more control

$\Rightarrow^?$ higher exploitability



Straightforward solutions

Qualitative [new definitions]

Weak Control (WC): at least 2 obtainable values
Strong Control (SC): all values are obtainable

Straightforward solutions

Qualitative [new definitions]

Weak Control (WC): at least 2 obtainable values
Strong Control (SC): all values are obtainable

Quantitative [more standard]

Quantitative Control (QC): $\sim$ channel capacity

$$QC(v, l) = \frac{\mathit{ln} \text{ \# of obtainable values}}{\mathit{ln} \text{ max \# of values}}$$

Straightforward solutions

Qualitative [new definitions]

Weak Control (WC): at least 2 obtainable values
Strong Control (SC): all values are obtainable

Quantitative [more standard]

Quantitative Control (QC): $\sim$ channel capacity

$$QC(v, I) = \frac{\mathit{ln}\ \#\ \text{of obtainable values}}{\mathit{ln}\ \max\ \#\ \text{of values}}$$

Motivating Example

- **Vuln. a:** WC, $\neg SC$, $QC \approx 0.08$
- **Vuln. b:** WC, $\neg SC$, $QC \approx 0.08$

We need something less one-dimensional.

A more promising approach

> **Evaluate the Domains of Control**
> The **set** $DoC_{v,l}$ **of obtainable values** for v at location l.

A more promising approach

Evaluate the Domains of Control

The **set** $DoC_{v,l}$ **of obtainable values** for v at location l.

Motivating example

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

A more promising approach

Evaluate the Domains of Control

The **set** $DoC_{v,l}$ **of obtainable values** for v at location l.

Motivating example

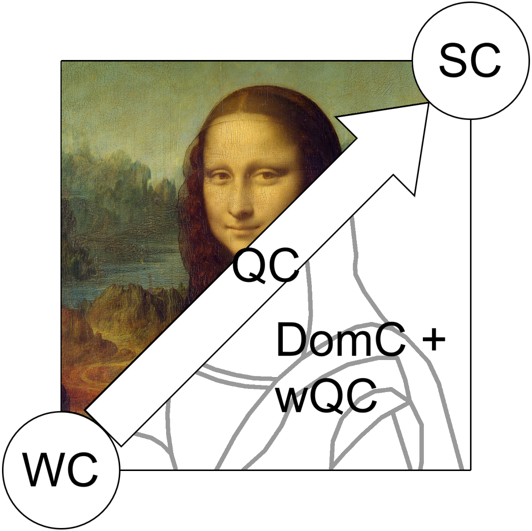
- **Vuln. a:** $DoC_{oob\_size} = [2^{64} - 296; 2^{64} - 257]$
- **Vuln. b:** $DoC_{oob\_size} = [1; 40]$

Bonus: Scoring domains of control

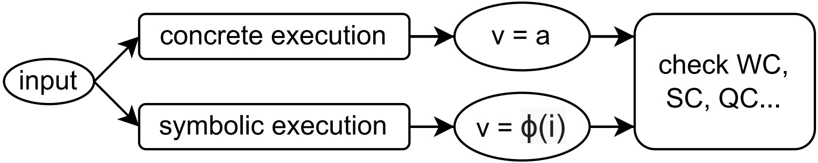
Weighted QC (wQC): different threat level $\omega(n)$ for each value n
$\Rightarrow$ With $\omega : x \mapsto \frac{1}{ln(2)x}$ (bias toward smaller values / locality):

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

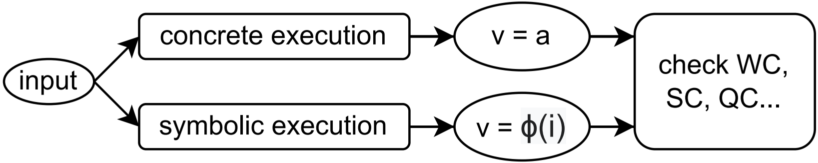
Recap



Algorithms for Weak, Strong and Quantitative Control



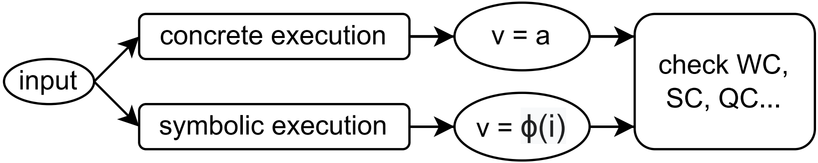
Algorithms for Weak, Strong and Quantitative Control



Weak Control

Quantifier-Free SMT: $sat(\phi(i) \neq a)$

Algorithms for Weak, Strong and Quantitative Control



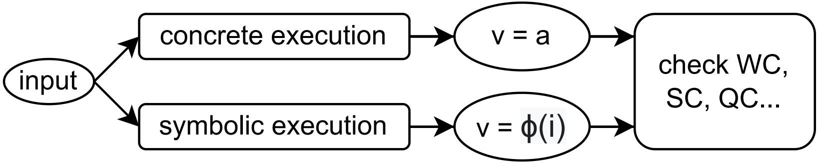
Weak Control

Quantifier-Free SMT: $sat(\phi(i) \neq a)$

Strong Control

Quantified SMT: $sat(\forall a, \exists\ i$ such that $\phi(i) = a)$
counterexample: get model for a in $\forall i,\ \phi(i) \neq a$

Algorithms for Weak, Strong and Quantitative Control



Weak Control

Quantifier-Free SMT: $sat(\phi(i) \neq a)$

Strong Control

Quantified SMT: $sat(\forall a, \exists\ i$ such that $\phi(i) = a)$
counterexample: get model for a in $\forall i$, $\phi(i) \neq a$

Quantitative Control

(Projected) Model Counting: count models for a in $\phi(i) = a$

Issues with standard techniques

Taint Analysis

- can only *disprove* (weak) control
- false positives: $t - t$
- false negatives: load/write

Issues with standard techniques

Taint Analysis

- can only *disprove* (weak) control
- false positives: $t - t$
- false negatives: load/write

Quantified SMT

- scalability (sometimes)

Issues with standard techniques

Taint Analysis

- can only *disprove* (weak) control
- false positives: $t - t$
- false negatives: load/write

Quantified SMT

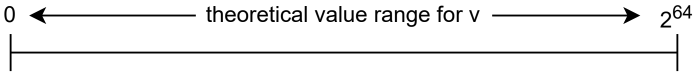
- scalability (sometimes)

Projected Model Counting

- scalability!

Measuring Domains of Control with Shrink and Split

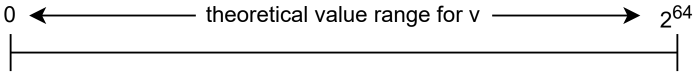
Step 0: initalization



- $DoC_{v,l} \subset [0, 2^{64}]$

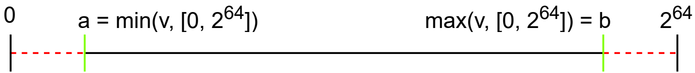
Measuring Domains of Control with Shrink and Split

Step 0: initalization



- $DoC_{v,l} \subset [0, 2^{64}]$

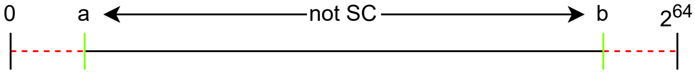
Step 1: shrinking



- $DoC_{v,l} \subset [a, b]$
- Z3: *minimize* and *maximize* (MaxSMT)
- update constraint to exclude infeasible values

Measuring Domains of Control with Shrink and Split

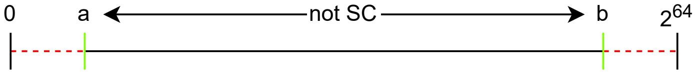
Step 2: checking for Strong Control



- no SC, we keep going!

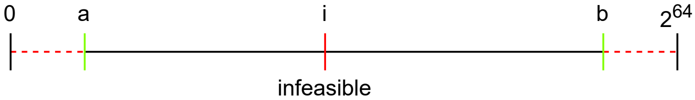
Measuring Domains of Control with Shrink and Split

Step 2: checking for Strong Control



- no SC, we keep going!

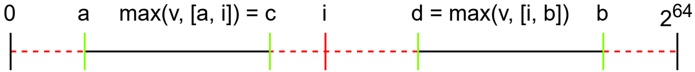
Step 3: splitting



- $DoC_{v,l} \subset [a, i[\cup]i, b]$
- i is an SC counterexample

Measuring Domains of Control with Shrink and Split

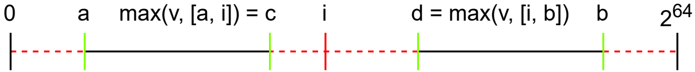
Repeat!



- $DoC_{v,I} \subset [a, c] \cup [d, b]$

Measuring Domains of Control with Shrink and Split

Repeat!

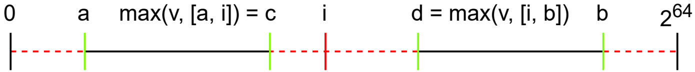


- $DoC_{v,I} \subset [a, c] \cup [d, b]$

- we stop on SC

Measuring Domains of Control with Shrink and Split

Repeat!



- $DoC_{v,I} \subset [a, c] \cup [d, b]$

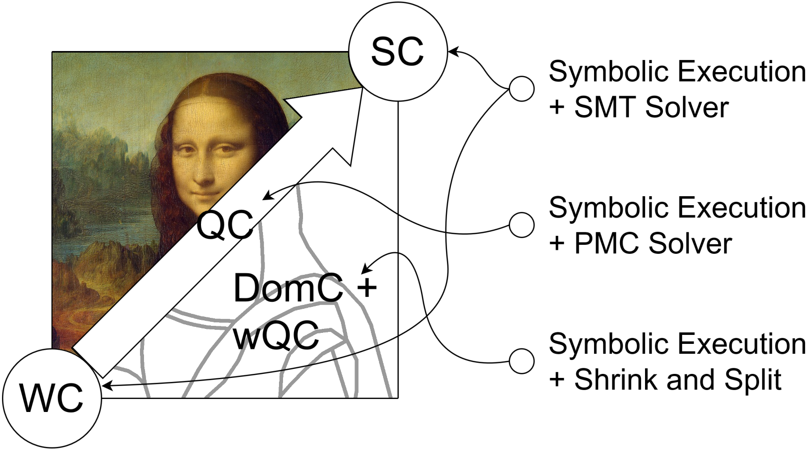
- we stop on SC

- $DoC_{v,I} = [a, c] \cup [d, b]$

Measuring Domains of Control with Shrink and Split

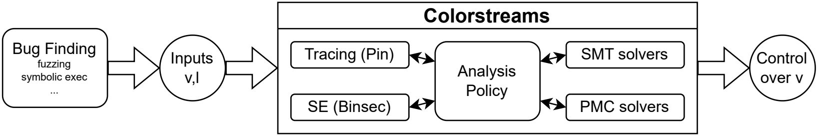
- **output:** set of intervals
- **max guarantees:** SC on each interval (no interrupt)
- **min guarantees:** WC on each interval
- refinement process ⇒ approximate results on interrupt
- bridges gap between qualitative and quantitative analysis

Recap


SC
QC
DomC +
wQC
WC
Symbolic Execution
+ SMT Solver
Symbolic Execution
+ PMC Solver
Symbolic Execution
+ Shrink and Split

Implementation

Colorstreams



- precise dynamic binary-level analysis
- symbolic execution through Binsec
- single-path (for now)

Evaluation

Benchmark

- 31 programs
- 9 real-world vulnerabilities

Evaluation

Benchmark

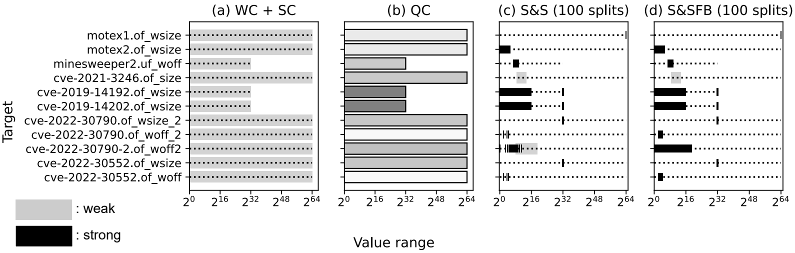
- 31 programs
- 9 real-world vulnerabilities

Research questions

- Is evaluating domains of control more precise in practice?
- How scalable are our algorithms in practice?

Evaluating Buffer Out-Of-Bounds Write Vulnerabilities

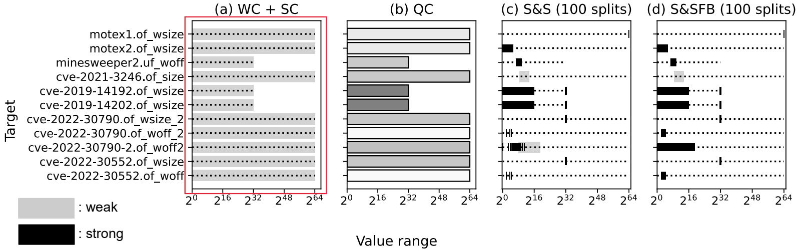
What does control look like in practice?



- only out-of-bounds values

Evaluating Buffer Out-Of-Bounds Write Vulnerabilities

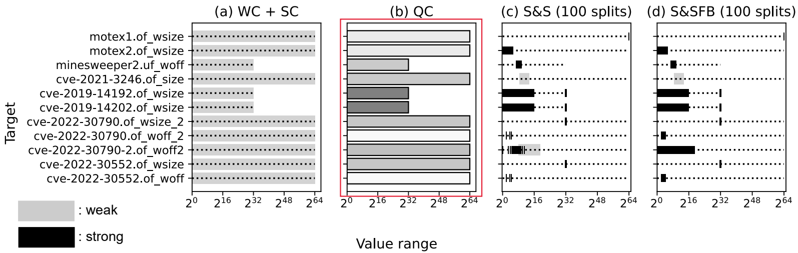
WC and SC are not so useful on their own



- In all cases we have WC but not SC...

Evaluating Buffer Out-Of-Bounds Write Vulnerabilities

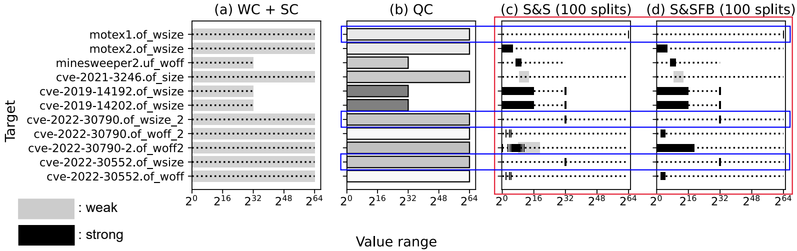
QC does not tell us much



- In all cases, there is **some** control
- It equalizes when we combine write offset and size + size of v

Evaluating Buffer Out-Of-Bounds Write Vulnerabilities

But the Domains of Control are different (sometimes)!



Improvements over QC

motex1, *cve-2022-30790*, *cve-2022-30552*:

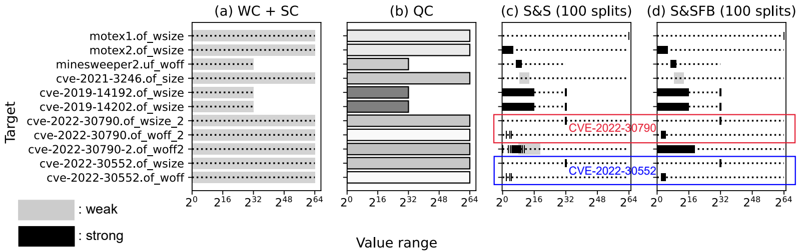
- **QC:** mid to high level of control
- **Domains:** only very large write sizes

Improving human analysis in the case of CVE-2022-30790

Analysis from human experts[1]

metadata corruption in linked list ⇒ arbitrary write

Domains of Control analysis



- does not look like arbitrary write
- looks (is) identical to *CVE-2022-30552*
- turns out, humans make mistakes

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

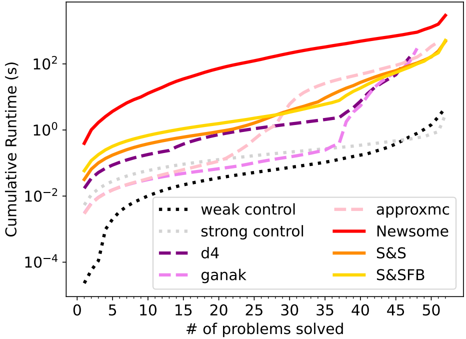
Recap: differentiating different values makes a difference!

Vulnerability	CVSS	WC / SC	QC	wQC	Truth
motex1 ($\sim$ a)		😐 ✗	😐 ✗	🙂	🙂
motex2 ($\sim$ b)		😐	😐	😐	😐
minesweeper2*		😐	😐	🙂 ✗	😐
cve-2021-3246	☹ ✗	😐	☹ ✗	😐	😐
cve-2019-14192	☹	😐 ✗	☹	☹	☹
cve-2019-14202	☹	😐 ✗	☹	☹	☹
cve-2022-30790	☹ ✗	😐 ✗	☹ ✗	🙂	🙂
cve-2022-30552	☹ ✗	😐 ✗	☹ ✗	🙂	🙂
cve-2022-30790-2		😐	☹ ✗	😐	😐

*single-path analysis is an issue here

Domains of Control analysis (wQC) $\Rightarrow$ more nuance

Scalability



Shrink and Split (S&S) performs quite well!

- decently fast (∼ approx PMC, << Newsome et al.[1])
- always gives results (vs. PMC: no result on timeout)

[1] Newsome et al., PLAS 2009

Conclusion

Bug prioritization based on Attacker Control

- formal definitions + algorithms
  - Domains of Control in particular
  - taint / counting are not up to the task
- **Shrink and Split**, a reasonable approach for DoC analysis
  - scalable + can approximate + strong guarantees
- prioritization of real-world bugs with decent performance

Conclusion

Bug prioritization based on **Attacker Control**

- formal definitions + algorithms
  - Domains of Control in particular
  - taint / counting are not up to the task
- **Shrink and Split**, a reasonable approach for DoC analysis
  - scalable + can approximate + strong guarantees
- prioritization of real-world bugs with decent performance

Ongoing works

- further automation
- improve domains of control scoring with wQC
- combining multiple paths
- write a paper and get published!

The End

Thank you for your attention.
Any questions?

(several positions available in the BINSEC team)