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
Introduction

Defining Attacker Control

Algorithms

Implementation and Experiments

Conclusion
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

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

developers cannot fix them all

but not all of them are equally dangerous
Motivating example

**Vulnerability a**

\( \text{size} < 40 \)

```plaintext
1  char buf[256];
2
3  if(size > 296)
4      size = 296;
5  if(size < 40) // should be size > 40
6      size -= 40;
7  memcpy(buf, msg, size);
```

\( \text{write size} \in [2^{64} - 40; 2^{64} - 1] \)

\( \Rightarrow \) crash

\( \Rightarrow \) maybe not that dangerous

**Vulnerability b**

\( \text{size} > 256 \)

```plaintext
1  char buf[256];
2
3  if(size > 296)
4      size = 296;
5  if(size < 40) // should be size > 40
6      size -= 40;
7  memcpy(buf, msg, size);
```

\( \text{write size} \in [257; 296] \)

\( \Rightarrow \) return address overwritten

\( \Rightarrow \) DANGER!!!
We need efficient bug prioritization.

Fuzzing & CO → crashing inputs → Evaluation → ranked vulnerabilities

<table>
<thead>
<tr>
<th>TO FIX</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OOB</td>
<td>+++</td>
</tr>
<tr>
<td>UAF</td>
<td>++</td>
</tr>
<tr>
<td>OOB</td>
<td>++</td>
</tr>
<tr>
<td>OOB</td>
<td>+</td>
</tr>
</tbody>
</table>

+++ ...

OOB

UAF

OOB

OOB
Existing approaches are lackluster

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

⇒ **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

Challenges

▶ what is exploitability? non-exploitability?
▶ precision vs. genericity
▶ poor scalability of precise analysis techniques
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
    → 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
  → 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: \( v \sim \text{buffer overflow size} \)

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  \[\Rightarrow ? \text{ more control}\]
\[\Rightarrow ? \text{ higher exploitability}\]
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

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

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

**Motivating Example**

- **Vuln. a:** WC, ¬SC, QC ≈ 0.08
- **Vuln. b:** WC, ¬SC, QC ≈ 0.08

*We need something less one-dimensional.*
A more promising approach

Evaluate the Domains of Control

The set $\text{DoC}_{v,l}$ of obtainable values for $v$ at location $l$. 
A more promising approach

Evaluate the Domains of Control

The set $\text{DoC}_{v,l}$ of obtainable values for $v$ at location $l$.

Motivating example

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

Bonus: Scoring domains of control

Weighted QC (wQC): different threat level $\omega(n)$ for each value $n$ ⇒

$\omega: x \rightarrow 7 \ln(2) x$ (bias toward smaller values / locality):

- **Vuln. a:** $\text{wQC}_{\text{oob\_size}} \approx 2^{-58}$
- **Vuln. b:** $\text{wQC}_{\text{oob\_size}} \approx 0.08$
A more promising approach

Evaluate the Domains of Control
The set $\text{DoC}_{v,l}$ of obtainable values for $v$ at location $l$.

Motivating example

- **Vuln. a:** $\text{DoC}_{\text{oob size}} = [2^{64} - 296; 2^{64} - 257]$
- **Vuln. b:** $\text{DoC}_{\text{oob size}} = [1; 40]$

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:** $wQC(\text{oob size}) \approx 2^{-58}$
- **Vuln. b:** $wQC(\text{oob size}) \approx 0.08$
Recap
Introduction

Defining Attacker Control

Algorithms

Implementation and Experiments

Conclusion
Algorithms for Weak, Strong and Quantitative Control

input

- concrete execution → \( v = a \)
- symbolic execution → \( v = \phi(i) \)

check WC, SC, QC...

Weak Control
- Quantifier-Free SMT: \( \text{sat} (\phi(i) \neq a) \)

Strong Control
- Quantified SMT: \( \text{sat} (\forall a, \exists i \text{ such that } \phi(i) = a) \)
  counterexample: get model for \( a \) in \( \forall i, \phi(i) \neq a \)

Quantitative Control
- (Projected) Model Counting: count models for \( a \) in \( \phi(i) = a \)
Algorithms for Weak, Strong and Quantitative Control

Weak Control

Quantifier-Free SMT: \( \text{sat}(\phi(i) \neq a) \)
Algorithms for Weak, Strong and Quantitative Control

Weak Control
Quantifier-Free SMT: \( \text{sat}(\phi(i) \neq a) \)

Strong Control
Quantified SMT: \( \text{sat}(\forall a, \exists i \text{ such that } \phi(i) = a) \)
counterexample: get model for \( a \) in \( \forall i, \phi(i) \neq a \)
Algorithms for Weak, Strong and Quantitative Control

Weak Control
Quantifier-Free SMT: \( \text{sat}(\phi(i) \neq a) \)

Strong Control
Quantified SMT: \( \text{sat}(\forall a, \exists i \text{ such that } \phi(i) = a) \)
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: Initialization**

\[ 0 \leftarrow \text{theoretical value range for } v \rightarrow 2^{64} \]

- \( DoC_{v,l} \subset [0, 2^{64}] \)
Measuring Domains of Control with Shrink and Split

**Step 0: initialization**

\[ 0 \rightarrow \text{theoretical value range for } v \rightarrow 2^{64} \]

- \( \text{DoC}_{v,l} \subset [0, 2^{64}] \)

**Step 1: shrinking**

\[ 0 \rightarrow a = \min(v, [0, 2^{64}]) \rightarrow \max(v, [0, 2^{64}]) = b \rightarrow 2^{64} \]

- \( \text{DoC}_{v,l} \subset [a, b] \)
- \( \text{Z3: minimize and maximize (MaxSMT)} \)
- \( \text{update constraint to exclude infeasible values} \)
Measuring Domains of Control with Shrink and Split

Step 2: checking for Strong Control

\[ a = \min(v, [0, 2^{64}]) \quad \text{max}(v, [0, 2^{64}]) = b \]

theoretical value range for \( v \)

\[ a \text{ infeasible} \quad b \]

\[ a \quad \text{not SC} \quad b \]

\[ \text{no SC, we keep going!} \]
Measuring Domains of Control with Shrink and Split

Step 2: checking for Strong Control

\[ a = \min(v, [0, 2^{64}]) \quad \text{max}(v, [0, 2^{64}]) = b \]

\[ 0 \quad a \quad \text{not SC} \quad b \quad 2^{64} \]

► no SC, we keep going!

Step 3: splitting

\[ a = \min(v, [0, 2^{64}]) \quad \text{max}(v, [0, 2^{64}]) = b \]

\[ 0 \quad a \quad i \quad b \quad 2^{64} \]

\[ \text{infeasible} \]

► \( DoC_v, i \subset [a, i \cup i, b] \)

► \( i \) is an SC counterexample
Measuring Domains of Control with Shrink and Split

Repeat!

\[ 0 \quad a \quad \max(v, [a, i]) = c \quad i \quad d = \max(v, [i, b]) \quad b \quad 2^{64} \]

▶ \( DoC_{v,l} \subset [a, c] \cup [d, b] \)
Measuring Domains of Control with Shrink and Split

Repeat!

\[
\begin{align*}
0 & \quad a \quad \max(v, [a, i]) = c \quad i \quad d = \max(v, [i, b]) \quad b \quad 2^{64} \\
\end{align*}
\]

\[\text{DoC}_{v, l} \subset [a, c] \cup [d, b]\]

\[
\begin{align*}
0 & \quad a \quad \text{SC} \quad c \quad d \quad \text{SC} \quad b \quad 2^{64} \\
\end{align*}
\]

\[\text{we stop on SC}\]
Measuring Domains of Control with Shrink and Split

Repeat!

\[
\begin{align*}
0 & \quad a & \text{max}(v, [a, i]) = c & i & \text{d} = \text{max}(v, [i, b]) & b & 2^{64} \\
& & & & & & \\
\end{align*}
\]

- \(DoC_{v,l} \subset [a, c] \cup [d, b]\)

we stop on SC

\[
\begin{align*}
0 & \quad a & c & \text{d} & b & 2^{64} \\
& & & & & & \\
\end{align*}
\]

- \(DoC_{v,l} = [a, c] \cup [d, b]\)
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 $\Rightarrow$ approximate results on interrupt
- bridges gap between qualitative and quantitative analysis
Recap

Symbolic Execution + SMT Solver
Symbolic Execution + PMC Solver
Symbolic Execution + Shrink and Split
Introduction

Defining Attacker Control

Algorithms

Implementation and Experiments

Conclusion
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
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

Recap: differentiating different values makes a difference!

<table>
<thead>
<tr>
<th>Vulnerability</th>
<th>CVSS</th>
<th>WC / SC</th>
<th>QC</th>
<th>wQC</th>
<th>Truth</th>
</tr>
</thead>
<tbody>
<tr>
<td>motex1 (∼ a)</td>
<td>😞</td>
<td>✗</td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>motex2 (∼ b)</td>
<td>😞</td>
<td></td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>minesweeper2*</td>
<td>😞</td>
<td></td>
<td>☉</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>cve-2021-3246</td>
<td>😞</td>
<td>✗</td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>cve-2019-14192</td>
<td>😞</td>
<td></td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>cve-2019-14202</td>
<td>😞</td>
<td></td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>cve-2022-30790</td>
<td>😞</td>
<td>☉</td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>cve-2022-30552</td>
<td>😞</td>
<td>☉</td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
<tr>
<td>cve-2022-30790-2</td>
<td>☉</td>
<td></td>
<td>☹</td>
<td>☉</td>
<td>☉</td>
</tr>
</tbody>
</table>

*single-path analysis is an issue here

Domains of Control analysis (wQC) ⇒ more nuance
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
Introduction

Defining Attacker Control

Algorithms

Implementation and Experiments

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!
Thank you for your attention.
Any questions?

(several positions available in the BINSEC team)