Find your Path – SOK Path exploration (in progress)

Dairo de Ruck

DistriNet

KU LEUVEN GENT

Path Exploration

- A module of the symbolic engine

Path Exploration

- Algorithmic choice

Path Exploration

- Algorithmic choice
  - Next exploration step

Path Exploration

- Algorithmic choice
  - Next exploration step
  - (sometimes) context-aware

Path Exploration

- Algorithmic choice
  - Next exploration step
  - (sometimes) context-aware
  - Significant impact on global efficiency

Path Exploration

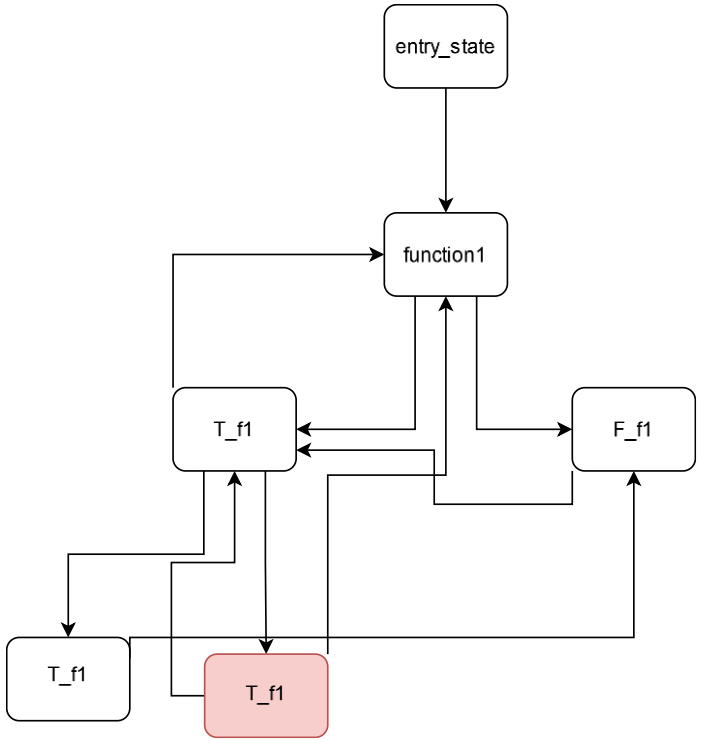
- Algorithmic choice
  - Next exploration step
  - (sometimes) context-aware
  - Significant impact on global efficiency

Just an afterthought

Path Exploration – working example

- How to detect bug in the red Basic Block

- Exhaustive vs Efficient

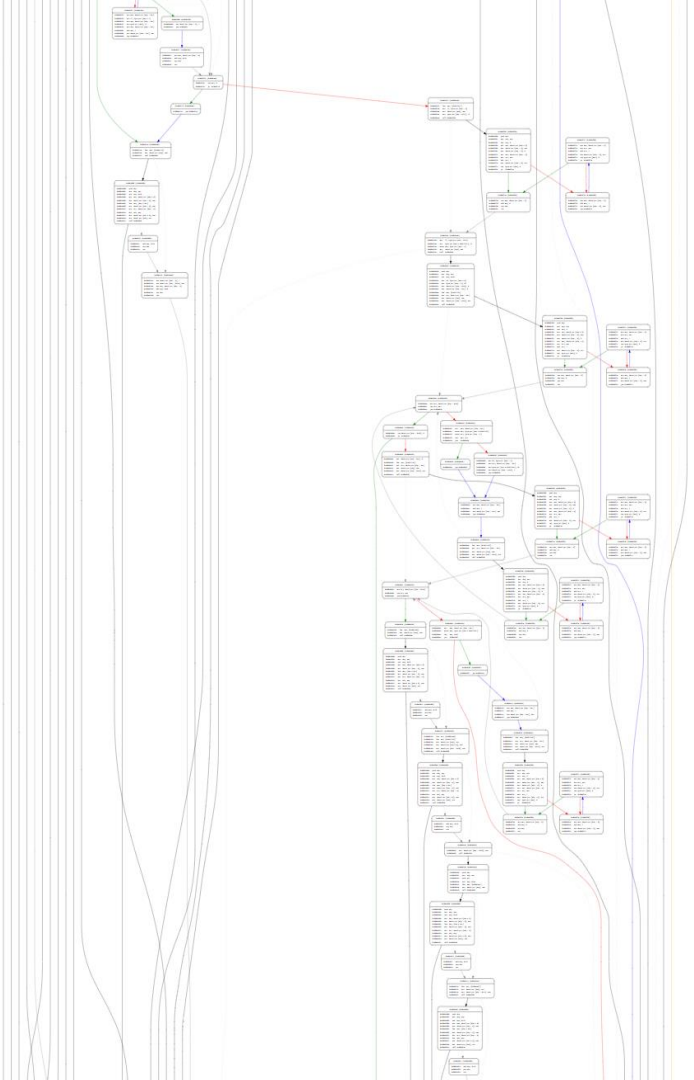


DistriNet

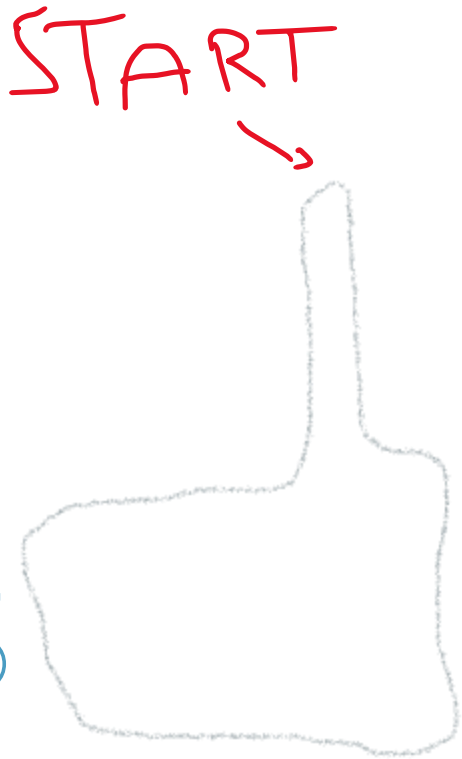
Path Exploration

- Looking at a **real** Control Flow Graph

- This choice has a major impact



Path Exploration



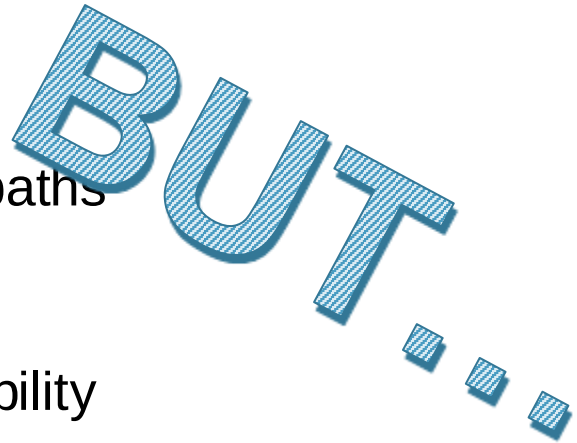
CFG was too big for PowerPoint to handle…
Use your imagination ☺

Path Exploration

- “Optimal” symbolic execution:
  - Explore all possible paths
  - Find *EVERY* vulnerability

Path Exploration

- “Optimal” symbolic execution:
  - Explore all possible paths
  - Find *EVERY* vulnerability



Path Exploration

- … An Optimal world does not exist:
  - Memory is finite
  - Time is money
  - World peace is yet to be achieved

Path Exploration

- … An Optimal world does not exist, so:
  - Find as many “targets” as possible within a reasonable time

Path Exploration

- … An Optimal world does not exist, so:
  - **Find** as much “targets” **as possible** within a **reasonable time**

SOK

SOK

- Various characteristics defined
  - Overarching selection strategies
  - Use of Search Space Optimizations (SSO)

	Goal	Selection Strategy			SSO			Year	Extension of	Operates on
	Targeted/Overall	Naive/Heuristic	Learning based	Knowledge based	State merging	Path Pruning	Constraint reasoning			
[17]	O	N+H						'08		LLVM IR
[63]*	T			x				'09		.NET
[68]	T			x				'10	KLEE	LLVM IR
[21]	O		x	x				'11	angr	VEX IR
[39]*	T	H						'11	SAGE	binary
[36]*	O			x	x			'12	KLEE	LLVM IR
[18]	O	H						'12	AEG	binary
[37]*	O			x				'13	KLEE	LLVM IR
[20]*	T	N		x		x		'13	KLEE	LLVM IR
[13]	T	H						'14		LLVM IR
[14]	O			x		x		'14		?
[47]*	O	?				x	x	'15	KLEE	LLVM IR
[28]*	T			x		x	x	'16	KLEE	LLVM IR
[42]	T			x			x	'16	KLEE	LLVM IR
[69]*	O			x				'19	angr	VEX IR
[60]	O	H						'20		AST
[43]*	O		x			x		'20	SAGE	binary
[16]	O			x			x	'20	KLEE	LLVM IR
[62]*	O		x					'20	KLEE	LLVM IR
[30]*	O		x					'21	KLEE	LLVM IR
[48]*	O		x					'21	angr	VEX IR
[65]	T			x				'21	angr	VEX IR
[66]	T			x				'21	angr	VEX IR
[26]*	O			x				'21	SPF	Java bitcode
[41]*	O		x	x			x	'22	Cover	VEX IR
[56]	O			x			x	'22		binary
[45]	T	H						'23		binary
[31]	T			x		x		'23		WebASM
[51]	O							'23		binary
[67]	T	N				x	x	'23		binary
[49]	O			x				'23		C-prog
[54]*	O	N	x	x			x	'23	Cover	VEX IR
[32]	O				x			'23	SPF	Java bitcode
[22]*	O	H						'23		binary

Path Exploration

- How to execute path exploration in real life?

Path Exploration

- How to execute path exploration in real life?
  - “Blind”
  - Informed (in *any* way)

Path Exploration

- How to execute path exploration in real life?
  - “Blind”

Global view

  - Informed (in *any* way)

Targeted view

Path Exploration - Features

- Relate to the SUT (subject under test)

- .. or to the algorithm in use

Path Exploration - Features

- Relate to the SUT
  - (CFG, node connectivity, loops, branch hit probabilities,…)
- .. or to the algorithm in use
  - (DFS, BFS, RNG element, lowest path costs, …)

Path Exploration - Features

- More advanced:
  - Potential presence of bugs

Path Exploration - Features

- More advanced:
  - Potential presence of bugs
  - Certain presence of bugs

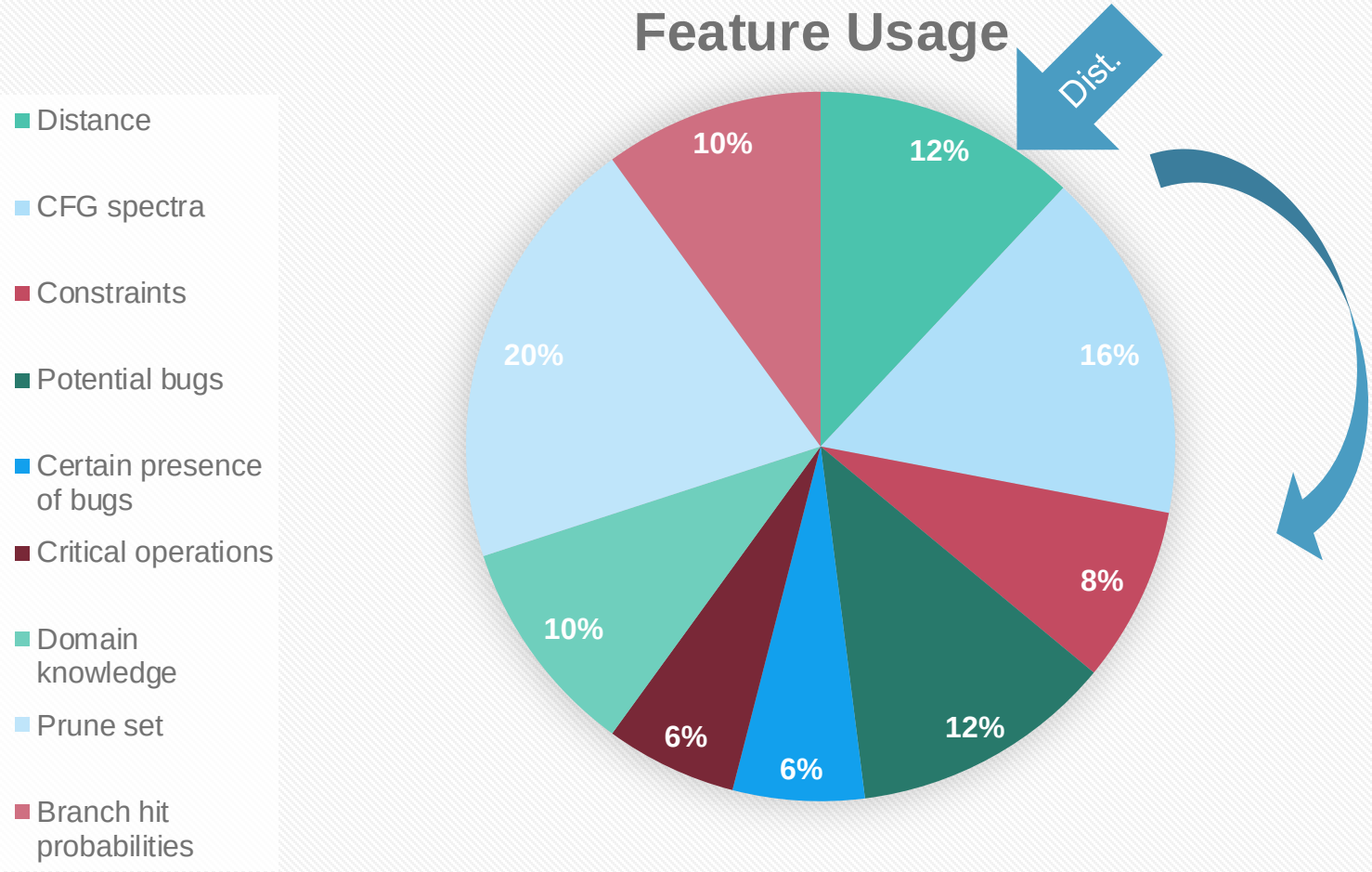
Path Exploration - Features

- More advanced:
  - Potential presence of bugs
  - Certain presence of bugs
  - Insertion of domain knowledge

Path Exploration - Features

- More advanced:
  - Potential presence of bugs
  - Certain presence of bugs
  - Insertion of domain knowledge
  - Preprocessing SAST/DAST step
  - …

Feature Usage



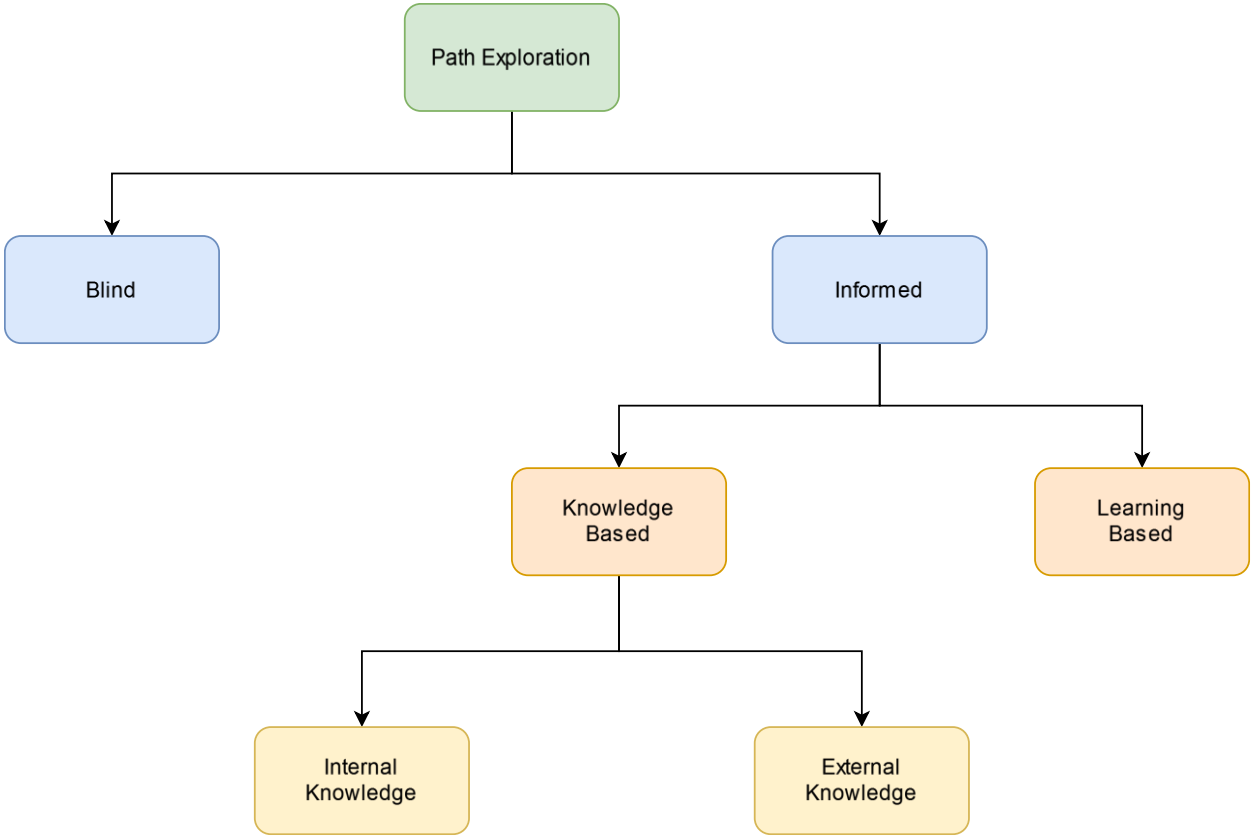
Path Exploration - Techniques

- “Blind” search
- Informed search
  - Knowledge based search
    - …
  - Learning based search
    - …

Path Exploration - Techniques

- **“Blind” search**
  - DFS
  - BFS
  - Random (state/path) Search
- Informed search
  - Knowledge based search
    - …
  - Learning based search
    - …

Path Exploration - Techniques



DistriNet

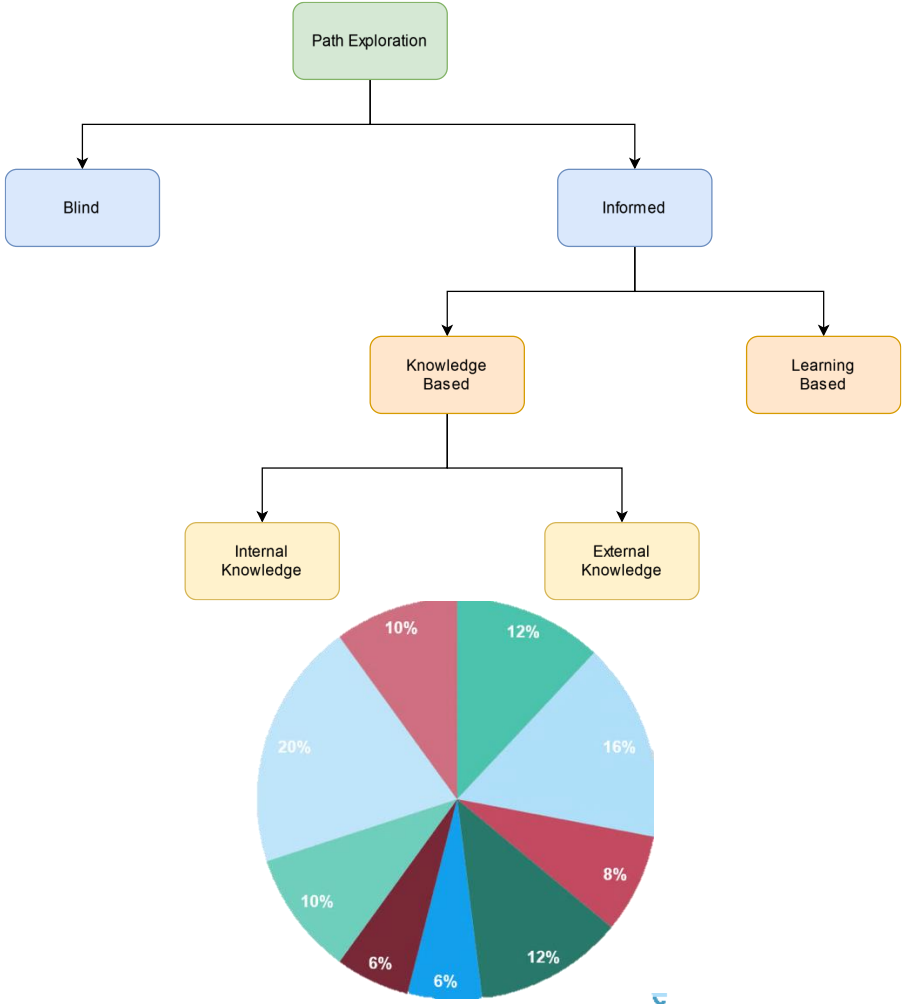
So, what does that mean

- “Blind” heuristics
  - More general → less efficient
  - BUT predictable memory usage
- Current Learning based approaches
  - (Over)engineered to a single set of ‘’vulnerabilities’’
  - No sense of global landscape
- Use of other external information
  - Point to specific locations
- Hybrid approaches (Fuzzer/SymEx or SAST/SymEx)
  - Very efficient, as they can complement each others' weaknesses

So, what does that mean

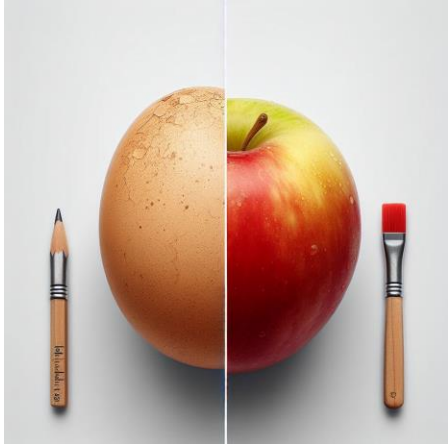
- Categorization of the path exploration techniques

- Categorization of features used (f. ex., pie chart)



DistriNet

So, what does that mean



- No uniform way to compare techniques…
  - angr vs KLEE vs new implementation
  - Targeted vs global
  - … vs …
- No uniform benchmark…
  - GNU Core utils, Juliet, RW bins, LAVA bench, SPEC CPU, ….
- No uniform metric
  - Coverage, vulnerabilities detected, accuracy, F1, vulnerability types, ….

So, what does that mean

2 options

- Reimplement the more prominent techniques

OR

- Create a uniform way to compare engines/techniques

So, what does that mean

2 options

- Reimplement the more prominent techniques

OR

- Create a uniform way to compare engines/techniques

WIP

- Comparative study of the exploration techniques
  - Reimplemented in the same engine
  - On the same set of binary

WIP

- Comparative study of the exploration techniques
  - Reimplemented in the same engine
    - TBD
  - On the same set of binary
    - CGC, CORE Utils, JULIET, …

Where do you think the added value lies in a benchmark?

Should there be more guidelines w.r.t. reproducibility?

Other caveats w.r.t. benchmarking these solutions?

Thank you!

Dairo.deruck@kuleuven.be

https://distrinet.cs.kuleuven.be/