Docovery : Toward Generic Automatic *Document Recovery*

Tomasz Kuchta

Miguel Castro

Cristian Cadar

Manuel Costa

Imperial College London

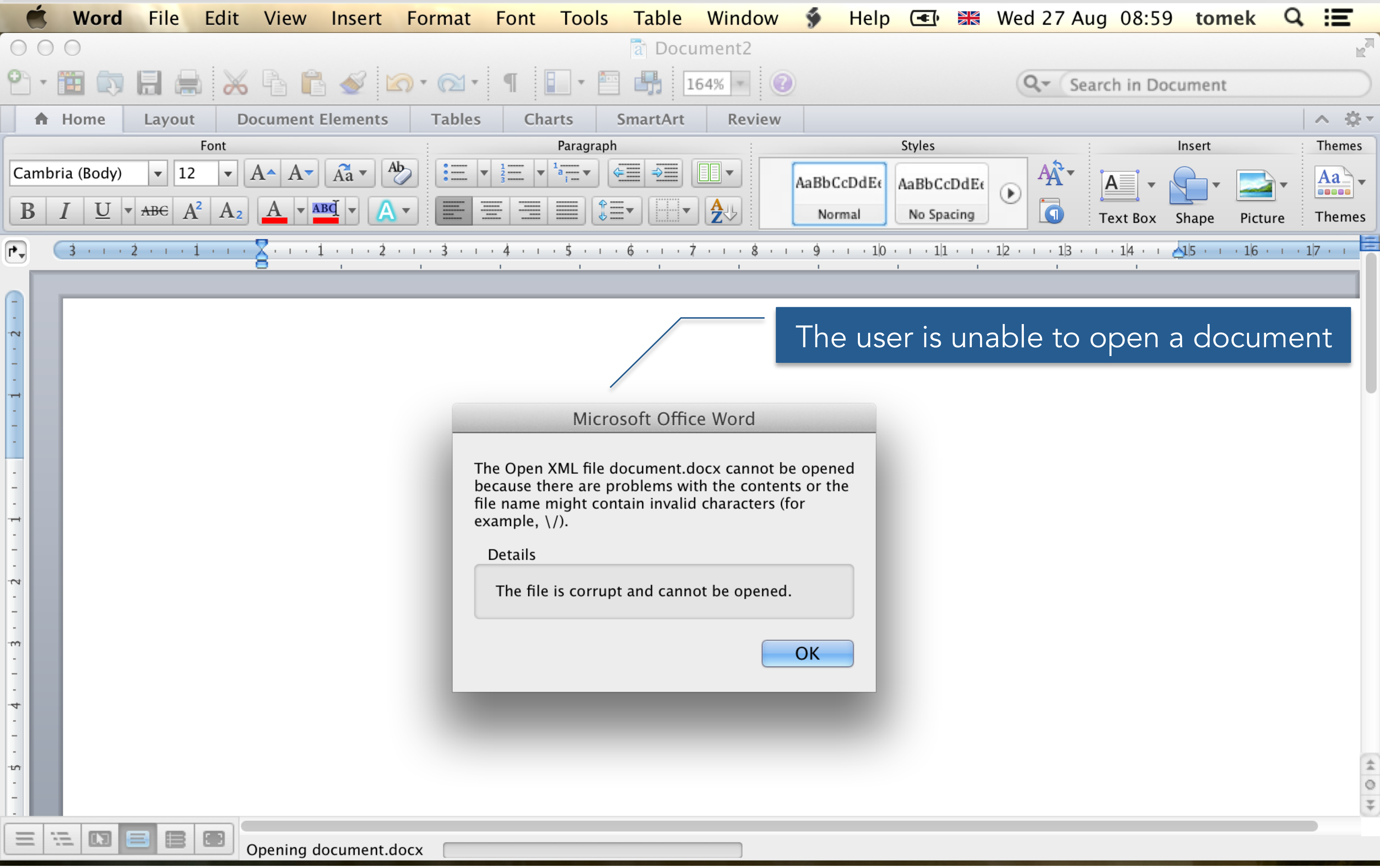
Microsoft Research

SOFTWARE RELIABILITY GROUP

ASE'14, 18th September 2014

This work is supported by Microsoft Research through its PhD Scholarship Programme

Motivation


Word File Edit View Insert Format Font Tools Table Window Help
Wed 27 Aug 08:59 tomek
Document2
164%
Search in Document
Home Layout Document Elements Tables Charts SmartArt Review
Font Paragraph Styles Insert Themes
Cambria (Body) 12
Normal No Spacing
Text Box Shape Picture Themes

The user is unable to open a document

Microsoft Office Word

The Open XML file document.docx cannot be opened because there are problems with the contents or the file name might contain invalid characters (for example, \/).

Details

The file is corrupt and cannot be opened.

OK

Opening document.docx

Document is corrupt

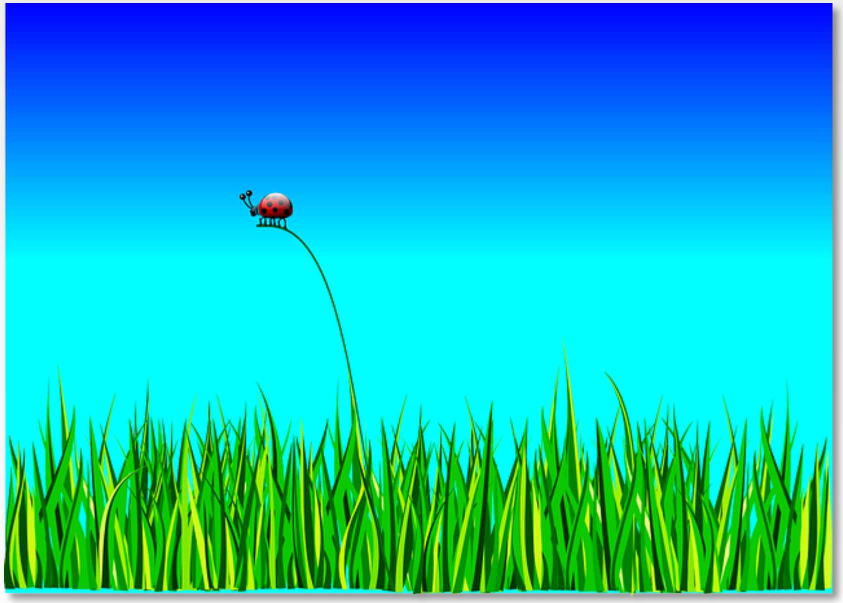
Storage failure, network transfer failure, power outage

Application has bugs

Buffer overflows, divisions by zero

Assertion failures, exceptions

Incompatibility across versions / applications



Such problems are highly user-visible

They account for a large number of security vulnerabilities

Error

Please contact support.

The root cause of the problem

Application is unable to handle corrupt or uncommon documents

Example: pine – a text mode e-mail client

Special “From:” field crashes the program

```
From: "\"\"\"\"\"\"\"\"\"...\"\"\"\"\"\""@host.fubar
```

What can we do about that?

Try to fix the program

Automatic patch generation

[GenProg, WCCI’08, ICSE’09; *SemFix*, ICSE’13; etc.]

Try to protect the program

Automatic input filter generation

[*Vigilante*, SOSP’05; *Shieldgen*, S&P’07; etc.]

What can we do about that?

Try to fix the document

Use format specification [DS repair, OOPSLA'03]

Learn and apply the correct values [*SOAP*, ICSE'12]

Truncate the document

Try to guess the right value

Or …

Research Question

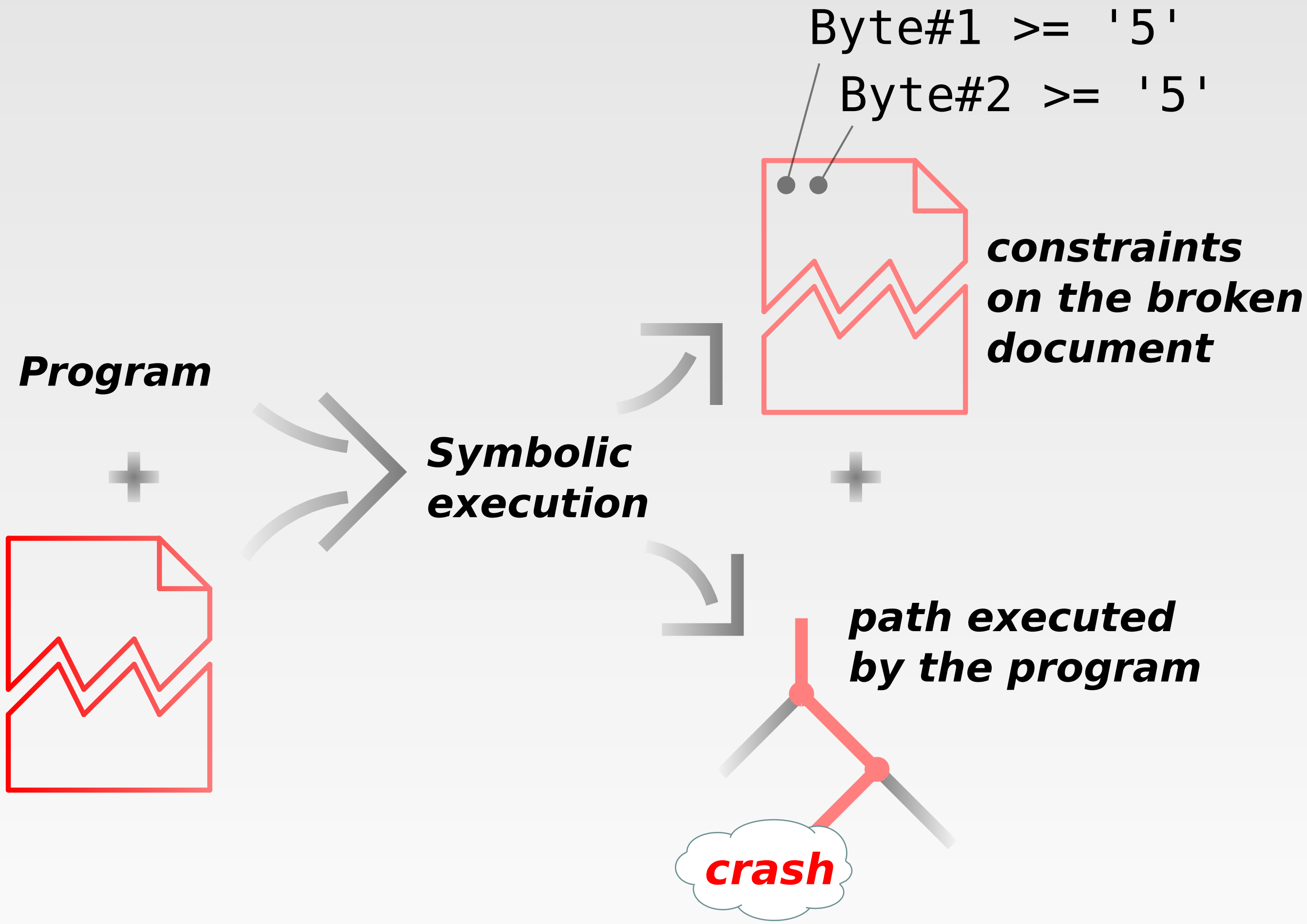
Is it possible to fix a broken document,
without assuming any input format,
in a way that preserves the original contents
as much as possible?

Docovery

Docovery: The General Idea

Program

Docovery: The General Idea



Program

+

Symbolic execution

Byte#1 >= '5'

Byte#2 >= '5'

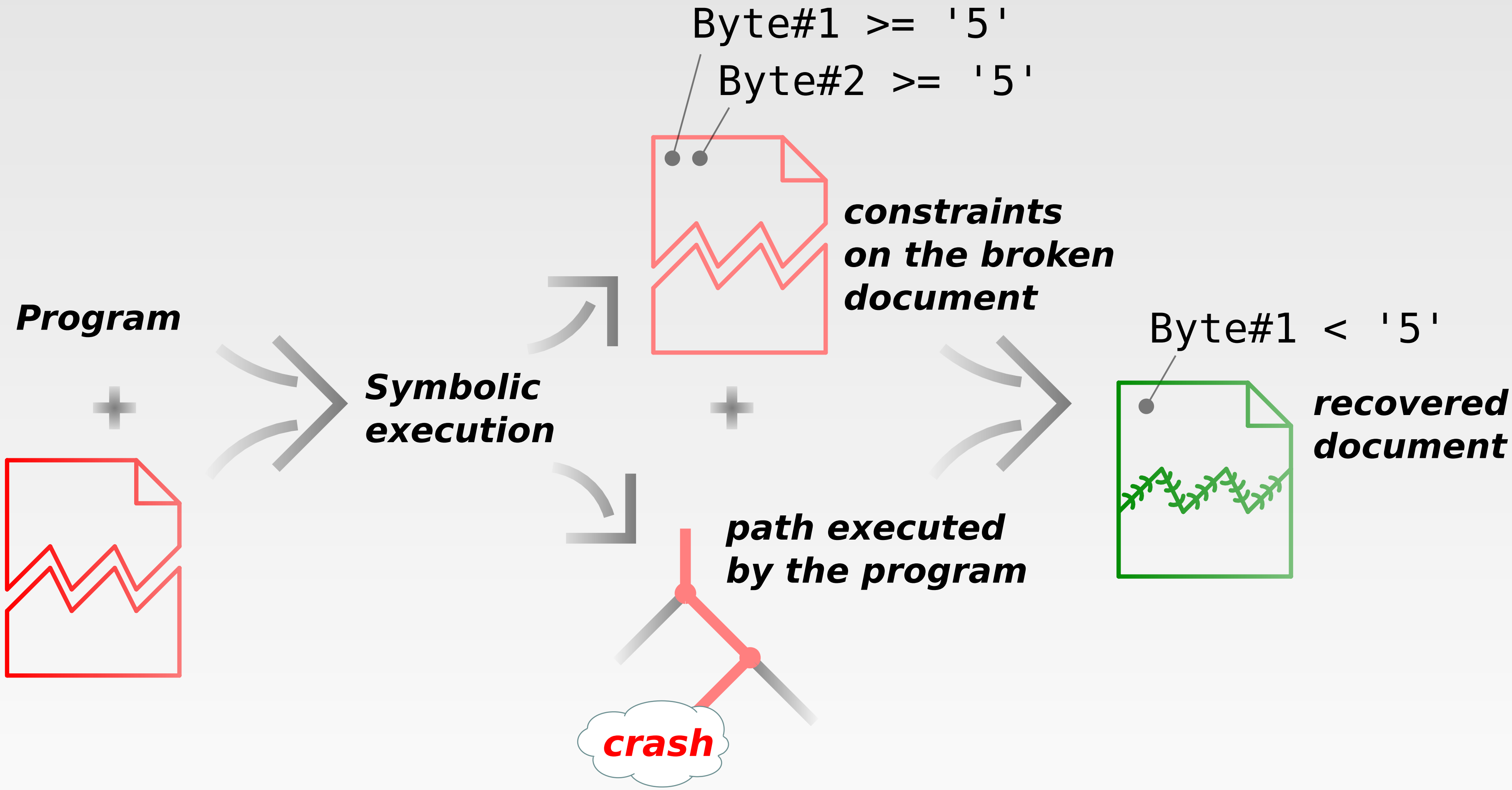
constraints on the broken document

+

path executed by the program

crash

Docovery: The General Idea



Program

+

Symbolic execution

Byte#1 >= '5'

Byte#2 >= '5'

constraints on the broken document

+

path executed by the program

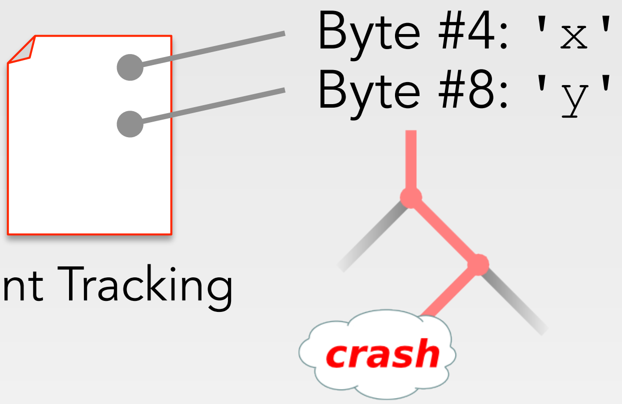
crash

Byte#1 < '5'

recovered document

DOCOVERY: MAIN STEPS

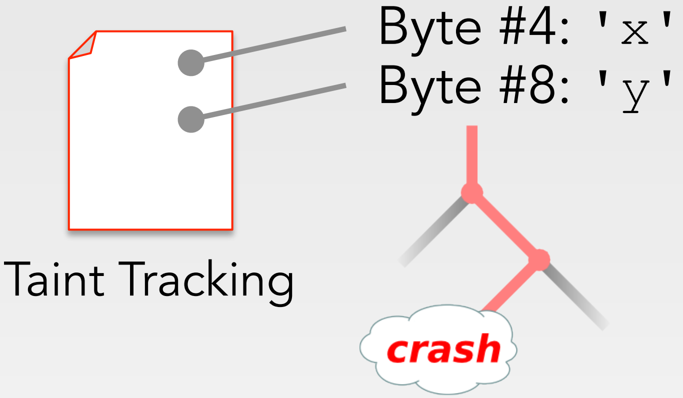
1 **Identify Potentially Corrupt Bytes**



Taint Tracking

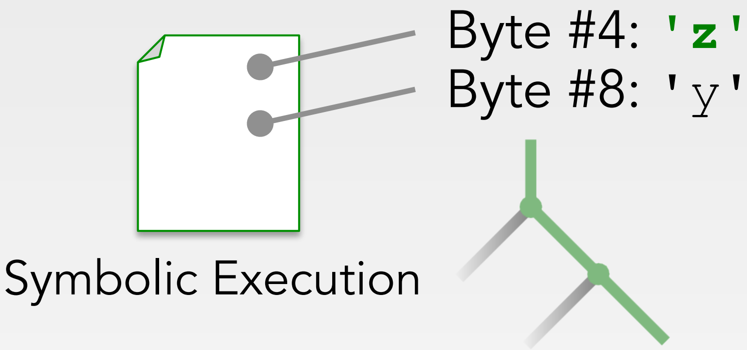
DOCOVERY: MAIN STEPS

1 **Identify Potentially Corrupt Bytes**



Taint Tracking

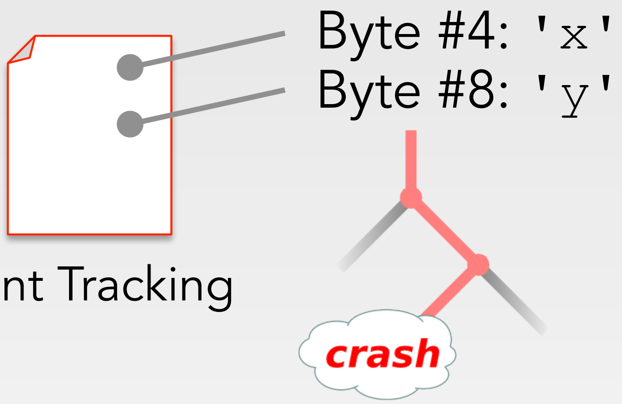
2 **Change The Bytes To Execute Another Path**



Symbolic Execution

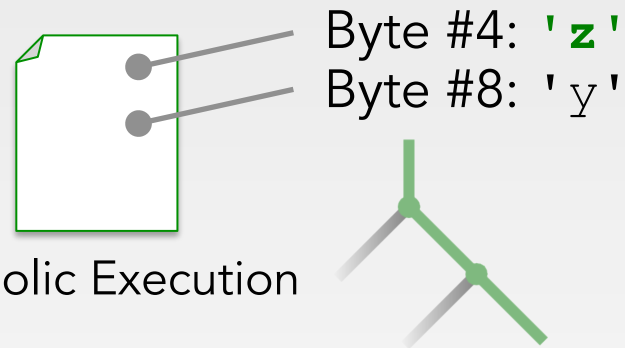
Docovery: Main Steps

1 Identify Potentially Corrupt Bytes



Taint Tracking

2 Change The Bytes To Execute Another Path

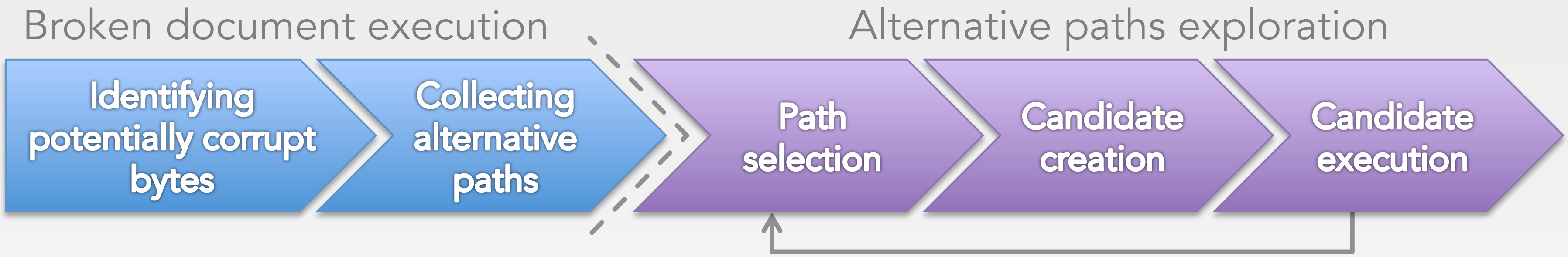


Symbolic Execution

3 Pick The Best Candidate

Levenshtein distance
and manual inspection

Docovery process



Broken document execution

Alternative paths exploration

Identifying potentially corrupt bytes

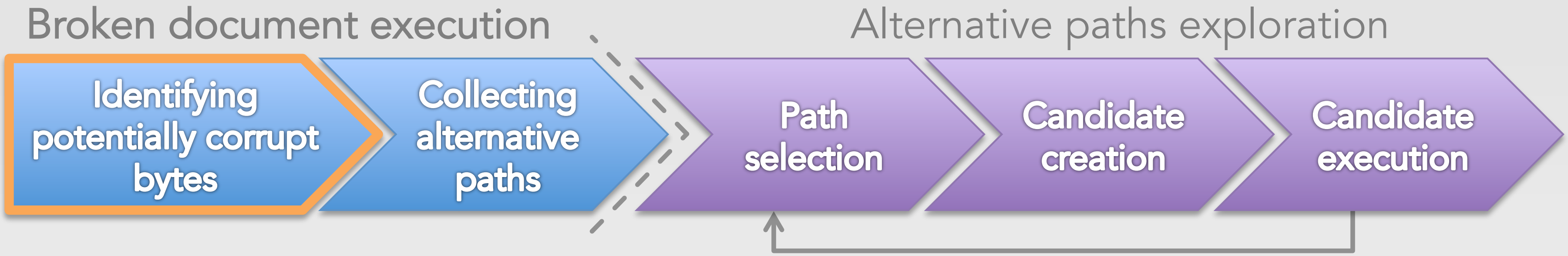
Collecting alternative paths

Path selection

Candidate creation

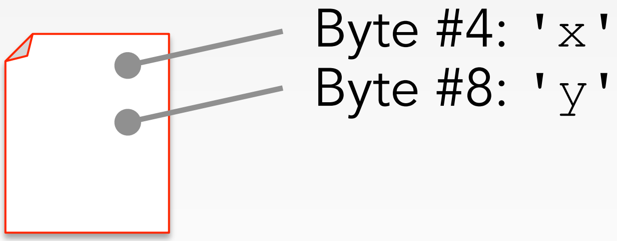
Candidate execution

Docovery: The Process

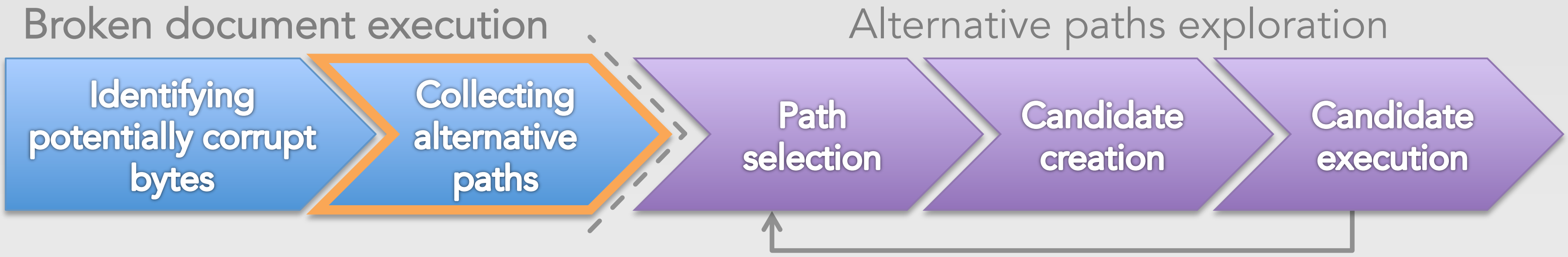

Broken document execution
Alternative paths exploration
Identifying potentially corrupt bytes
Collecting alternative paths
Path selection
Candidate creation
Candidate execution

Taint tracking

- Track the flow of data from a source (input) to a sink (point of crash)
- Identifying potentially corrupt bytes
  - Byte-level precision
  - No control flow dependencies
  - No address tainting

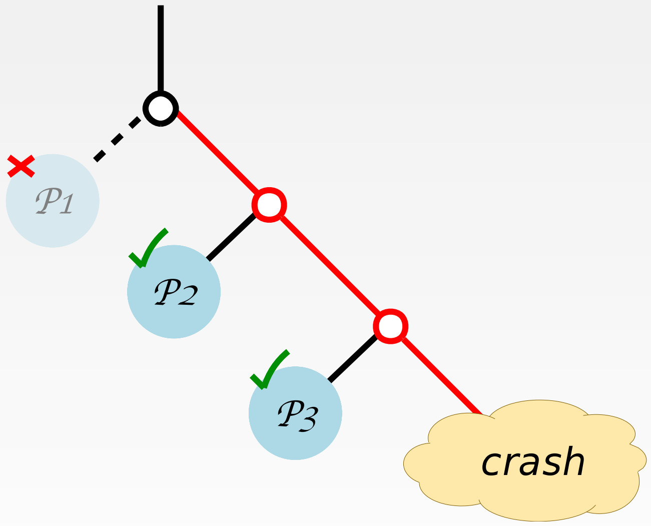


DOCOVERY: THE PROCESS

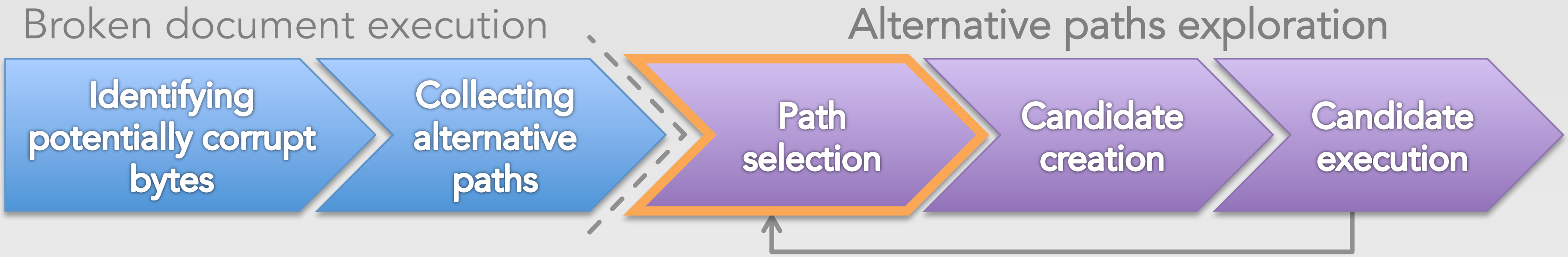

Broken document execution
Alternative paths exploration
Identifying potentially corrupt bytes
Collecting alternative paths
Path selection
Candidate creation
Candidate execution

Collecting alternative paths

- Mark the potentially corrupt bytes as symbolic
- Lazily verify feasibility



DOCOVERY: THE PROCESS



Broken document execution

Alternative paths exploration

Identifying potentially corrupt bytes

Collecting alternative paths

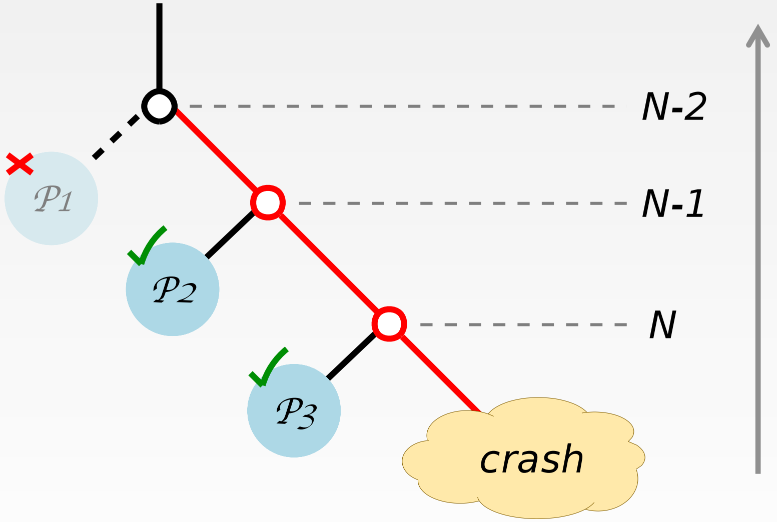
Path selection

Candidate creation

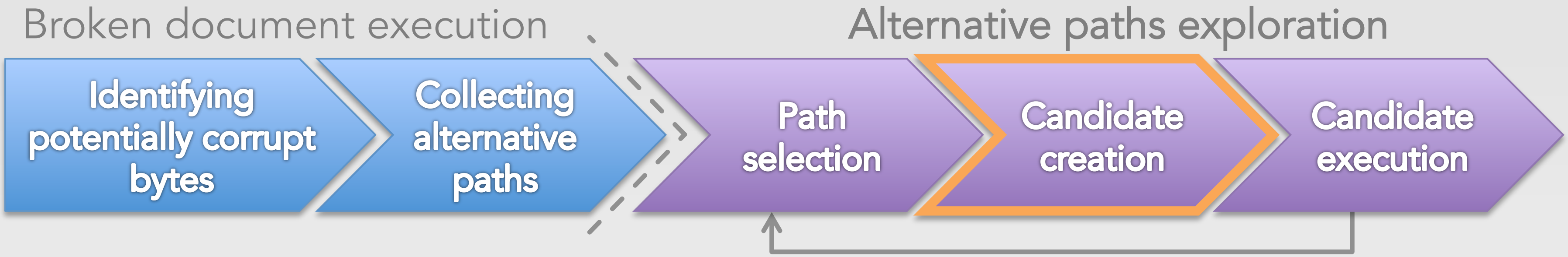
Candidate execution

Path selection

- Last N deepest paths are collected
- Start from the paths closest to the crash point



Docovery: The Process



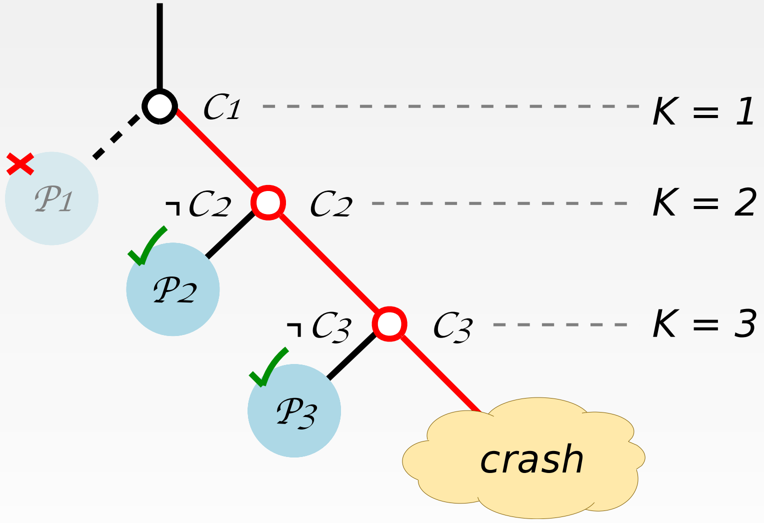
Broken document execution
Alternative paths exploration
Identifying potentially corrupt bytes
Collecting alternative paths
Path selection
Candidate creation
Candidate execution

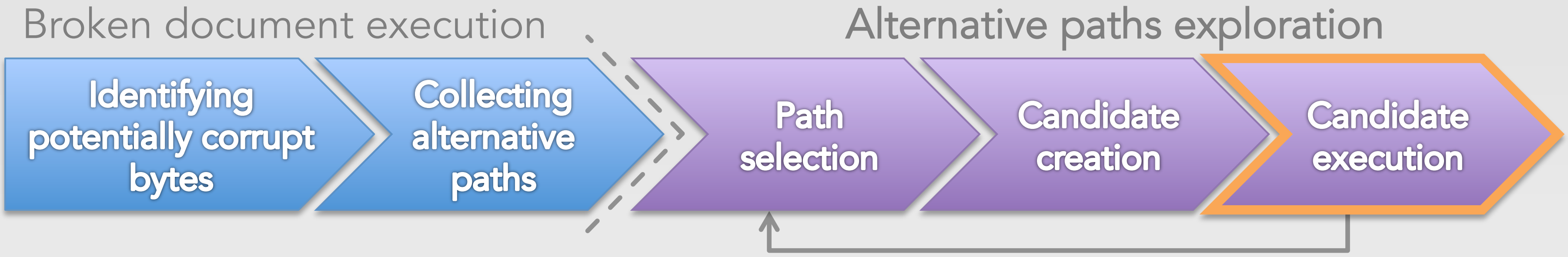
Negate the Kth constraint and drop the remaining

Ask constraint solver for a satisfying assignment

Path P_3 : $C_1 \wedge C_2 \wedge \neg C_3$

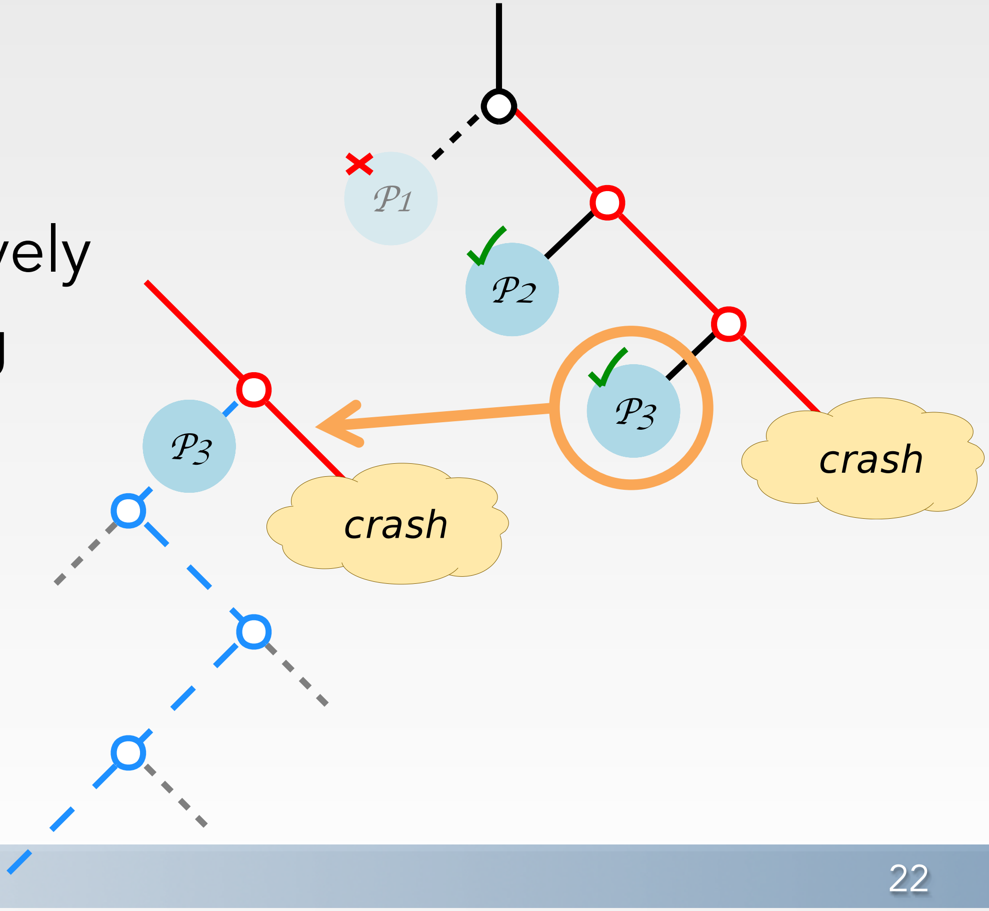
Path P_2 : $C_1 \wedge \neg C_2$




Broken document execution
Alternative paths exploration
Identifying potentially corrupt bytes
Collecting alternative paths
Path selection
Candidate creation
Candidate execution

Candidate execution

- Store the candidate
- Re-run the program natively
- Successful if not crashing



Evaluating candidate documents

Levenshtein distance (edit distance)

Byte-level similarity metric

Independent of document format

Smaller distance = higher similarity

Semi-automatic evaluation of program output

Looking for warnings / errors, exit code

Similarity to the correct output

Evaluation

Implementation

- Built on top of KLEE [OSDI'08]
- Using ZESTI functionality [ICSE'12]
- Interprets LLVM bitcode of C applications

Benchmarks

- **`pr`** – a pagination utility
- **`pine`** – a text-mode e-mail client
- **`dwarfdump`** – a debug information display tool
- **`readelf`** – an ELF file information display tool

Benchmark	Document type	Document Sizes	Max number of changed bytes
`pr`	Plain text	up to 256 pages / 1080 KB	1
`pine`	MBOX mailbox	up to 320 e-mails / 2.3 MB	24
`dwarfdump`	DWARF executables	up to 1.1 MB	1
`readelf`	ELF object files	up to 1.5 MB	8

Bugs

Known, real-world bugs injected manually

`pr, pine, readelf` – buffer overflow

`dwarfdump` – division by zero

Benchmark	'Buggy' sequence
`pr`	`Lorem ipsum...0x08 0x08...0x09 EOF`
`pine`	`...From: "\"\"\"\"\"\"\"\...\"\"\"\""@host.fubar...`
`dwarfdump`	`...GCC: (Ubuntu/Linaro 4.6.3...0x00 0x00...`
`readelf`	`...0xFD 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF...`

Taint tracking results

Regardless of document size

Benchmark	Document	Number of potentially corrupt bytes
pr	1 – 256 pages / 4.4 – 1080 KB	1
pine	5 – 320 e-mails / 13 KB – 2.3MB	25
dwarfdump	62 KB – 1.1 MB	2
readelf	54 KB – 1.5 MB	16

```
pr: Lorem ipsum...08 08...09 EOF
pine: "\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"\"..."@host
dwarfdump: ...GCC: (Ubuntu/Linaro 4.6.3...00 00...
readelf: ...40 01 00 00 00 00 00 00...FD FF FF FF FF FF FF FF...
```

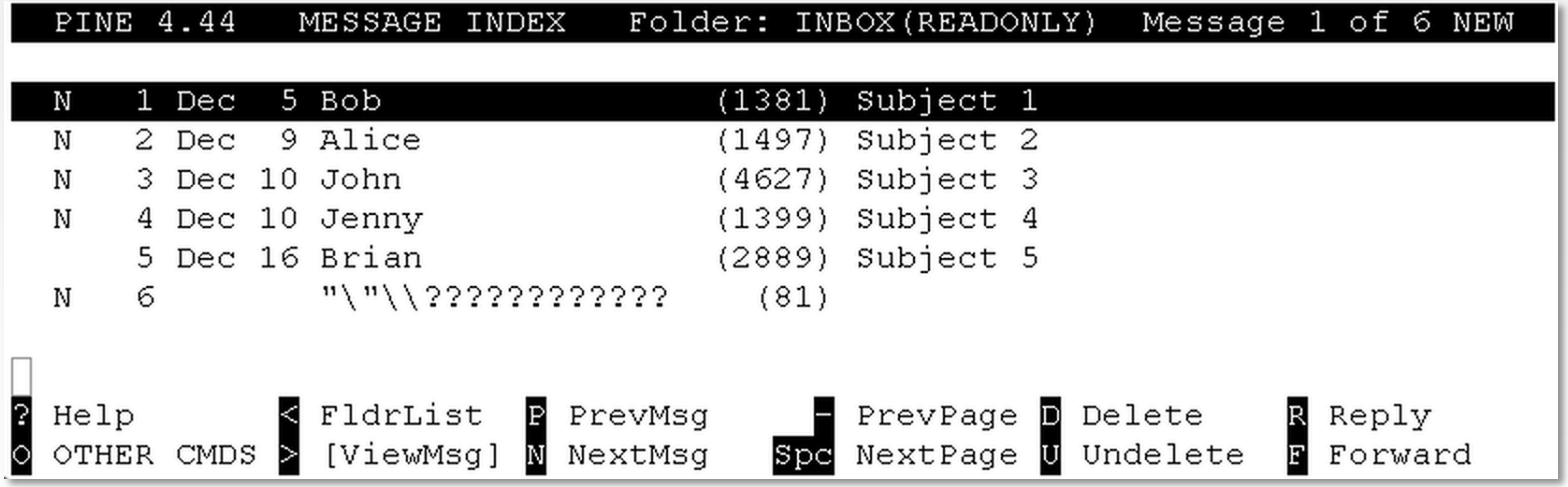
Candidates for `pr`

Document	'Buggy' sequence
Original	`Lorem ipsum...0x08 0x08...`**`0x09`**` EOF`
Candidate A	`Lorem ipsum...0x08 0x08...`**`0x00`**` EOF`
Candidate B	`Lorem ipsum...0x08 0x08...`**`0x0C`**` EOF`
Candidate C	`Lorem ipsum...0x08 0x08...`**`0x0A`**` EOF`

All the candidates print out correctly

Candidates for `pine`

Document	'Buggy' sequence
Original	`From: "\"\"\"\"................\""@host.fubar`
Candidate A	`From: "\"\...\0x0E...\0x0E\"...\""@host.fubar`
Candidate B	`From: "\"\...\\\0x0E..\0x0E\"..\""@host.fubar`
Candidate C	`From: "\"\...\0x00\"...........\""@host.fubar`



```
  PINE 4.44   MESSAGE INDEX   Folder: INBOX(READONLY)   Message 1 of 6 NEW

  N   1 Dec  5 Bob                 (1381) Subject 1
  N   2 Dec  9 Alice               (1497) Subject 2
  N   3 Dec 10 John                (4627) Subject 3
  N   4 Dec 10 Jenny               (1399) Subject 4
      5 Dec 16 Brian               (2889) Subject 5
  N   6        "\"\\????????????     (81)


? Help         < FldrList   P PrevMsg       - PrevPage D Delete     R Reply
O OTHER CMDS   > [ViewMsg]  N NextMsg     Spc NextPage U Undelete   F Forward
```

EVALUATION: SAMPLE RECOVERY RESULTS

Candidates for **dwarfdump**

Document	'Buggy' sequence
Original	...GCC: (Ubuntu/Linaro 4.6.3...**0x00** **0x00**...
Candidate A	...GCC: (Ubuntu/Linaro 4.6.3...**0x01** 0x00...
Candidate B	...GCC: (Ubuntu/Linaro 4.6.3...0x00 **0x01**...

Candidate A: debug dump, success return code

Candidate B: error

Candidates for `readelf`

Document	'Buggy' sequence
Original	… **40 01 00 00 00 00 00 00** … **FD FF FF FF FF FF FF FF** …
Candidate A	… 40 01 00 00 00 00 00 00 … **F0 01 00 00 00 00 00 80** …
Candidate B	… **FE FF FF FF FF FF FF FF** … FD FF FF FF FF FF FF FF …
Candidate C	… **00 00** 00 00 00 00 00 00 … FD FF FF FF FF FF FF FF …

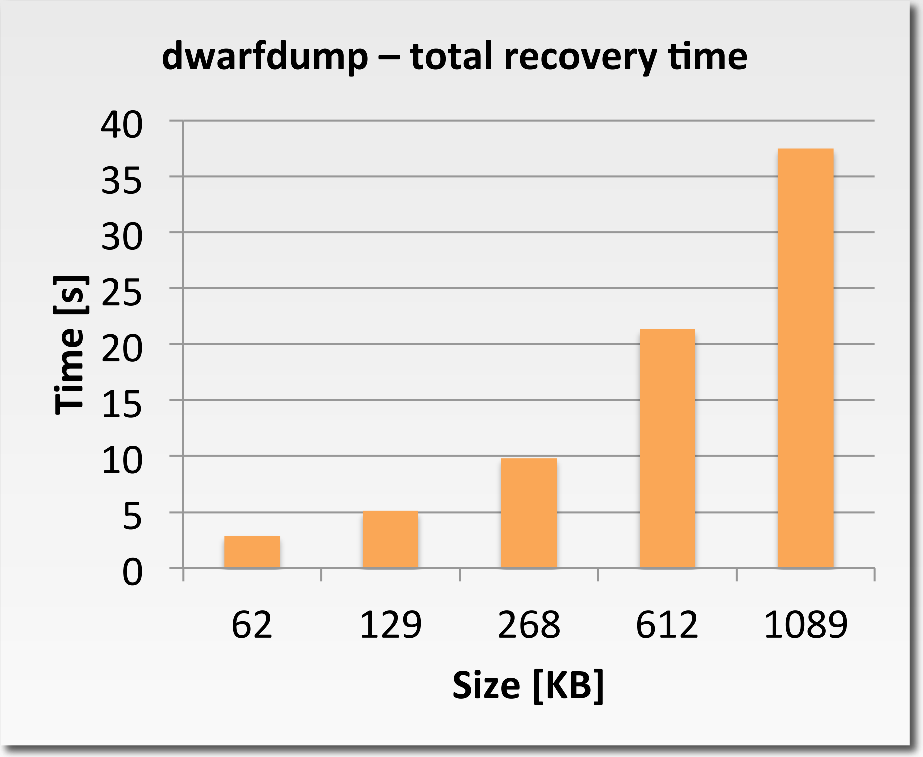
Candidate A: most of output, but with a warning

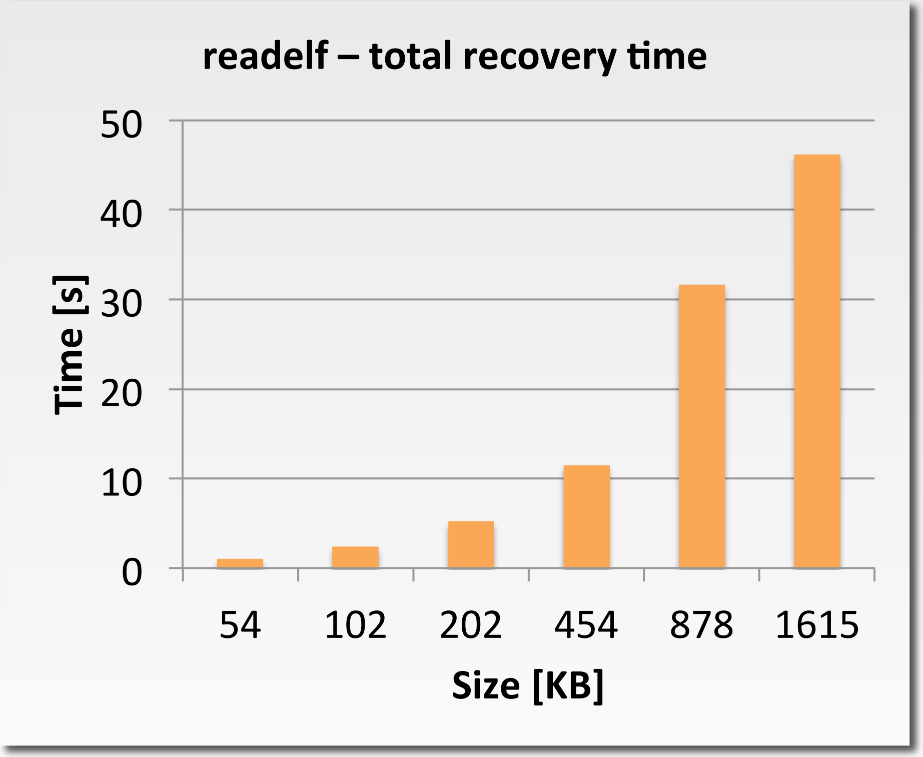
Candidate B: almost no output and an error

Candidate C: almost no output (no debug data)

Performance varies across applications

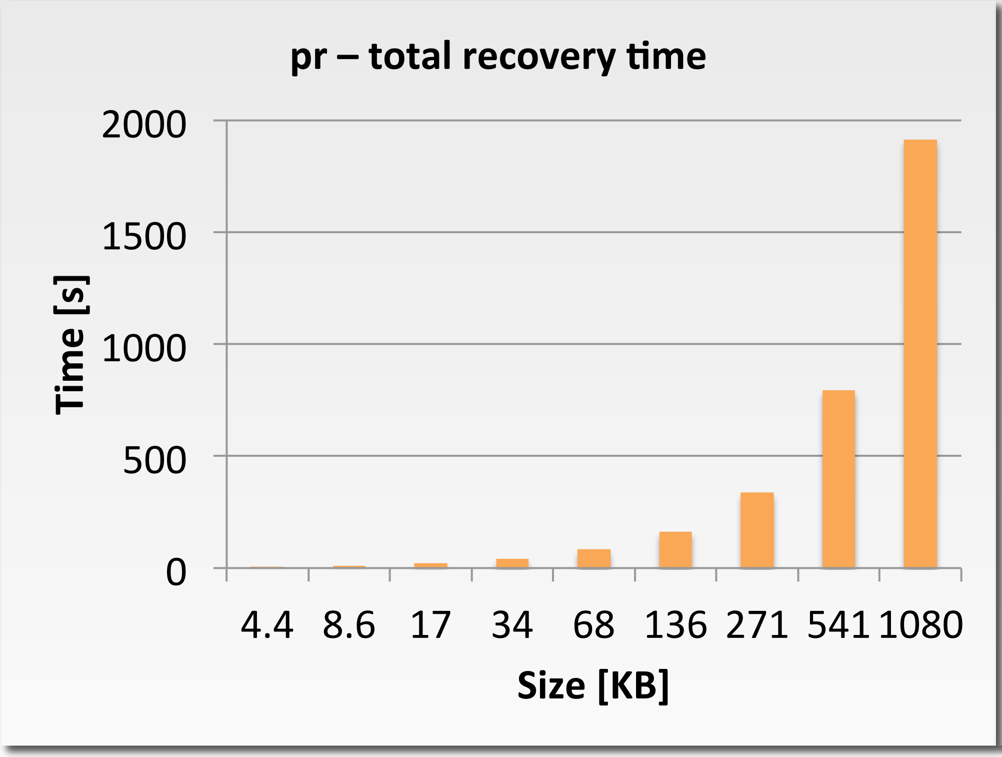
Sometimes, the recovery is cheap

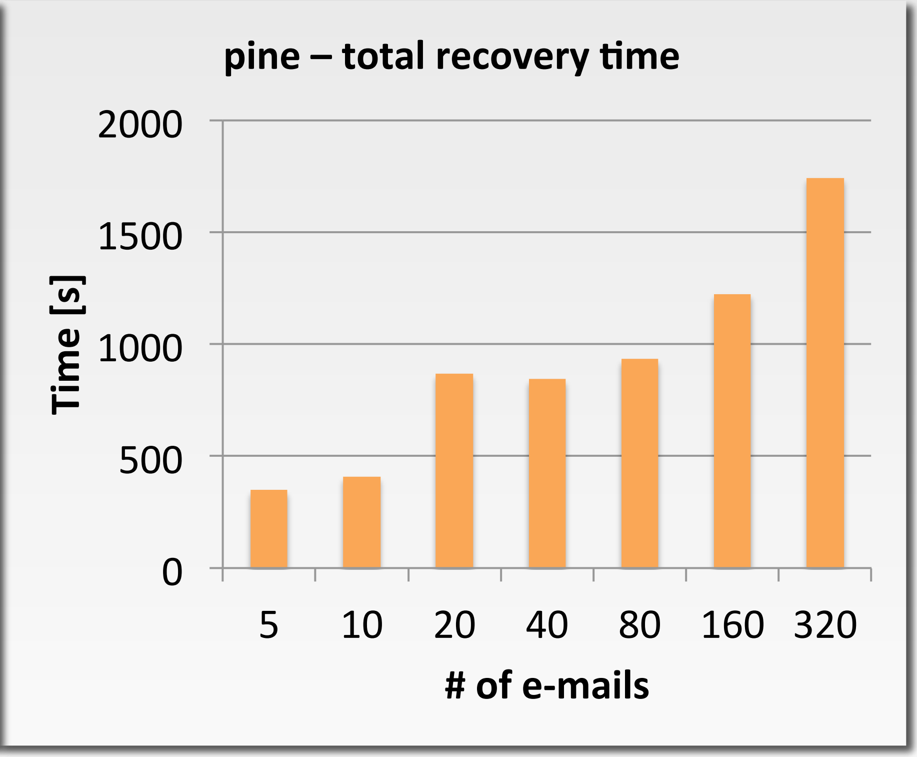




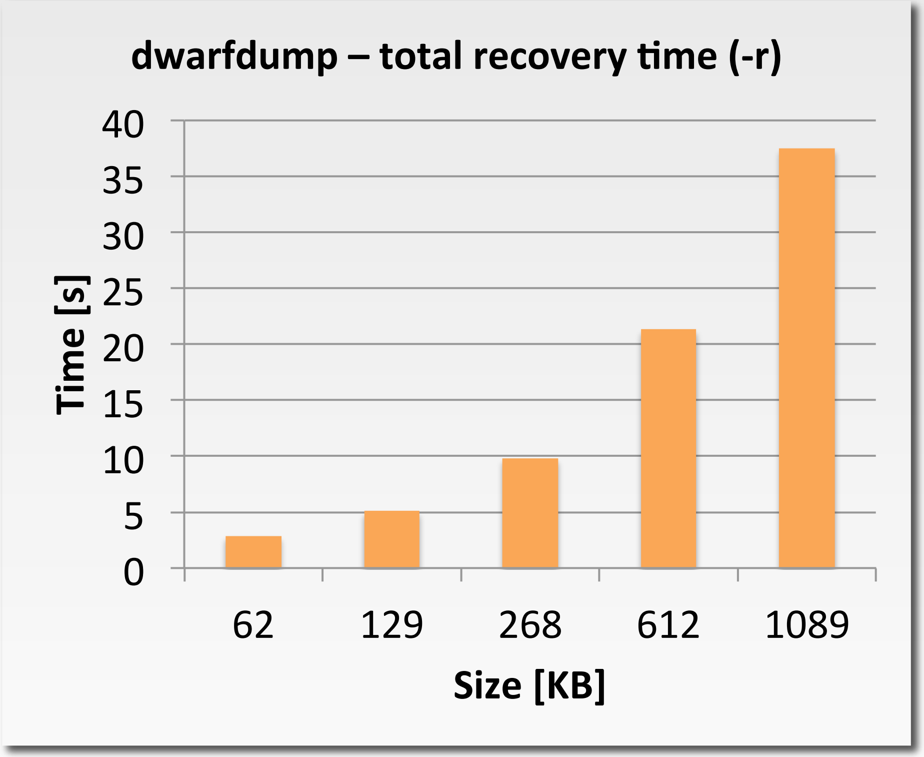
Performance varies across applications

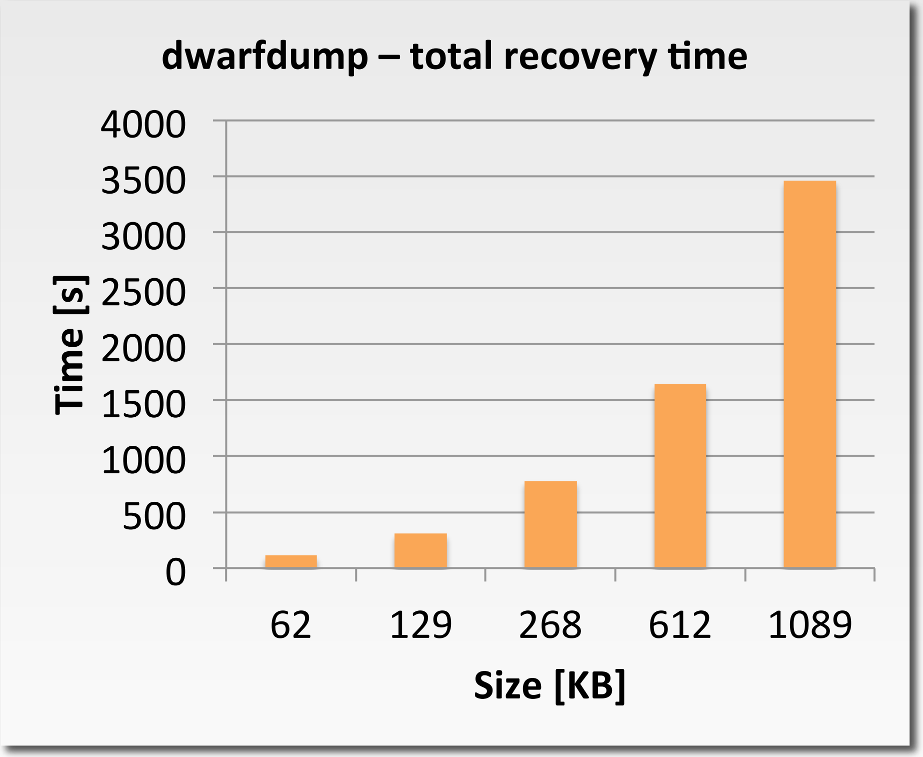
Sometimes, the recovery is expensive





Performance depends on the executed path



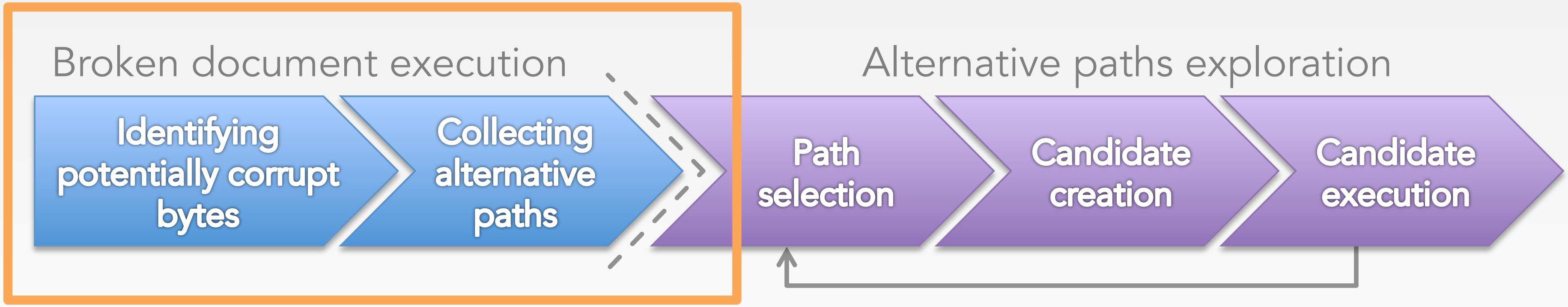


Performance

Most time spent on taint tracking and collecting alternative paths

First recovery candidate usually within minutes after path exploration starts

All collected paths usually explored within minutes


Broken document execution
Alternative paths exploration
Identifying potentially corrupt bytes
Collecting alternative paths
Path selection
Candidate creation
Candidate execution

Most time

Limitations of Docovery

Fundamental

Scalability: complex, highly-structured documents

Supports only byte mutations

Implementation

Can’t handle multiple faults

Handles only generic errors

No support for document modifications (read-only)

Requires C source code of the program

Summary

Docovery

A novel technique for format-independent document recovery

- Uses taint tracking and symbolic execution techniques
- Recovery candidates explore alternative execution paths

Successfully recovered

- Text files
- Mailboxes
- Executables
- Object files

http://srg.doc.ic.ac.uk/projects/docovery

SOFTWARE RELIABILITY
GROUP