2011 IEEE Symposium on Security and Privacy

Differential Slicing: Identifying Causal Execution Differences for Security Applications

Noah M. Johnson[1], Juan Caballero[2], Kevin Zhijie Chen[1], Stephen McCamant[1], Pongsin Poosankam[1, 3], Daniel Reynaud[1], and Dawn Song[1]

[1]University of California, Berkeley
[2]IMDEA Software Institute
[3]Carnegie Mellon University

左昌國
Seminar @ ADLab, NCU-CSIE

Outline

- Introduction
- Problem Definition and Overview
- Trace Alignment
- Slice-Align
- Evaluation
- Related Work
- Conclusion

Introduction

- Why does the program crash?
- At what situation does the malware do malicious behaviors?
- How do you solve above problems if you don't have the source code?
  - Static analysis
  - Dynamic analysis
  - …
  - Too much time spent

Introduction

- This paper,
  - proposes “Differential Slicing”
    - Given 2 execution traces of a program with a target difference
    - Automatically finds the input and environment differences that caused the target difference
    - Generates a causal difference graph
      - Simply expressed what happened

Problem Definition and Overview

- The goal is to “understand” the target difference
  - To identify the input differences that caused the target difference.
  - To understand the sequence of events that let from the input differences to the target difference.

➔ To build the causal difference graph

Problem Definition and Overview

Passing trace

Input differences? (byte level)

```
$ vuln_cmp bar bazaar
Strings are not equal

$ vuln_cmp "" foo
<<crashed at line 11>>
```

Failing trace

Then the passing trace and the failing trace can be used for Trace Alignment.

```
1  char *s1=NULL, *s2=NULL;
2  int main(int argc, char **argv) {
3    if(argc < 3)
4      return 1;
5    int len1 = strlen(argv[1]);
6    int len2 = strlen(argv[2]);
7    if (len1)
8      s1 = (char *)malloc(len1);
9    if (len2)
10     s2 = (char *)malloc(len2);
11   strncpy(s1, argv[1], len1);
12   strncpy(s2, argv[2], len2);
13   if (strcmp(s1,s2) != 0)
14     printf("Strings are not equal\n");
15   return 0;
16 }
```

Target difference

Figure 1: Motivating example program, vuln_cmp.c.

Problem D

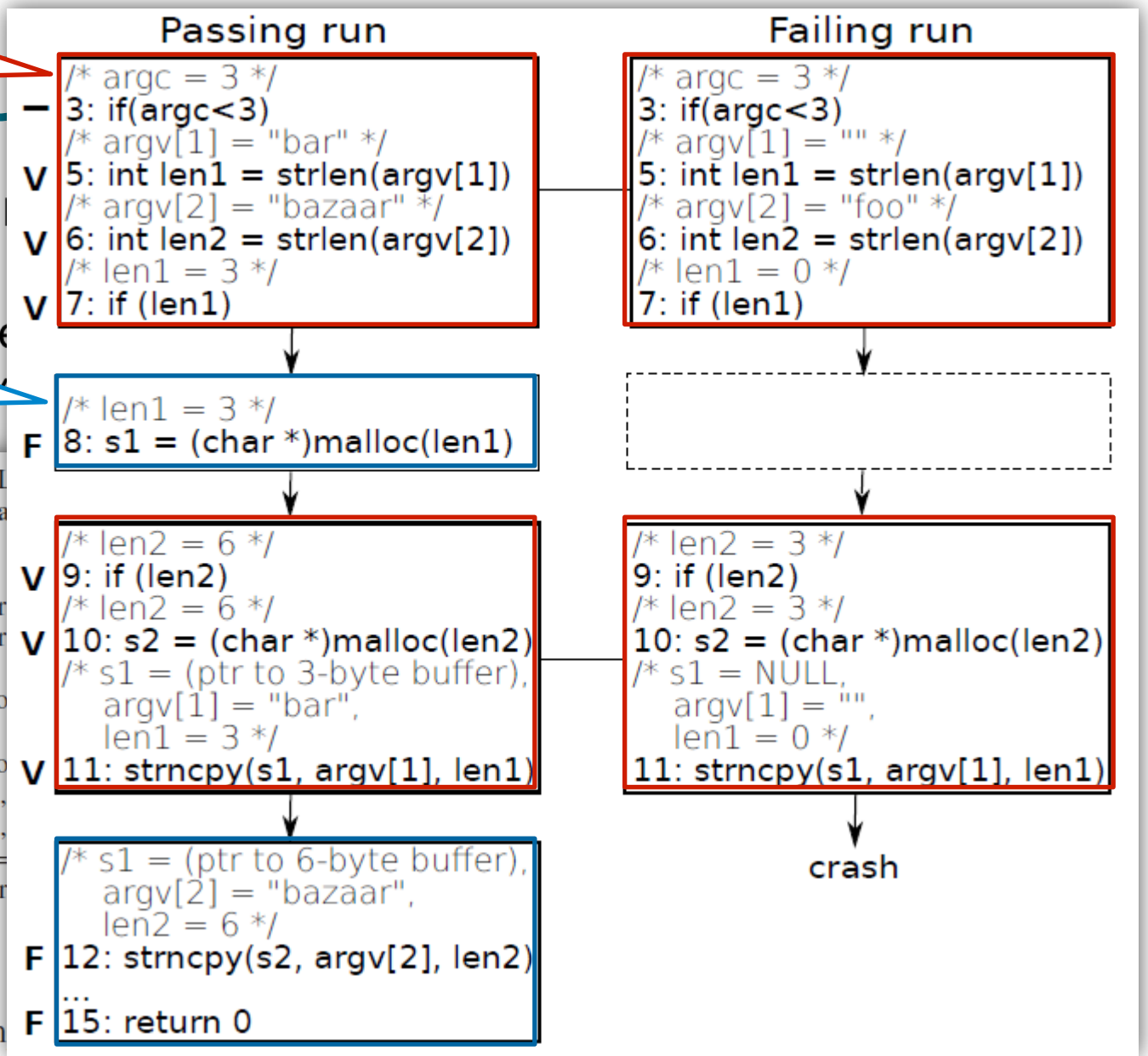
- Trace Alignme
  - Given passing
    r
    s

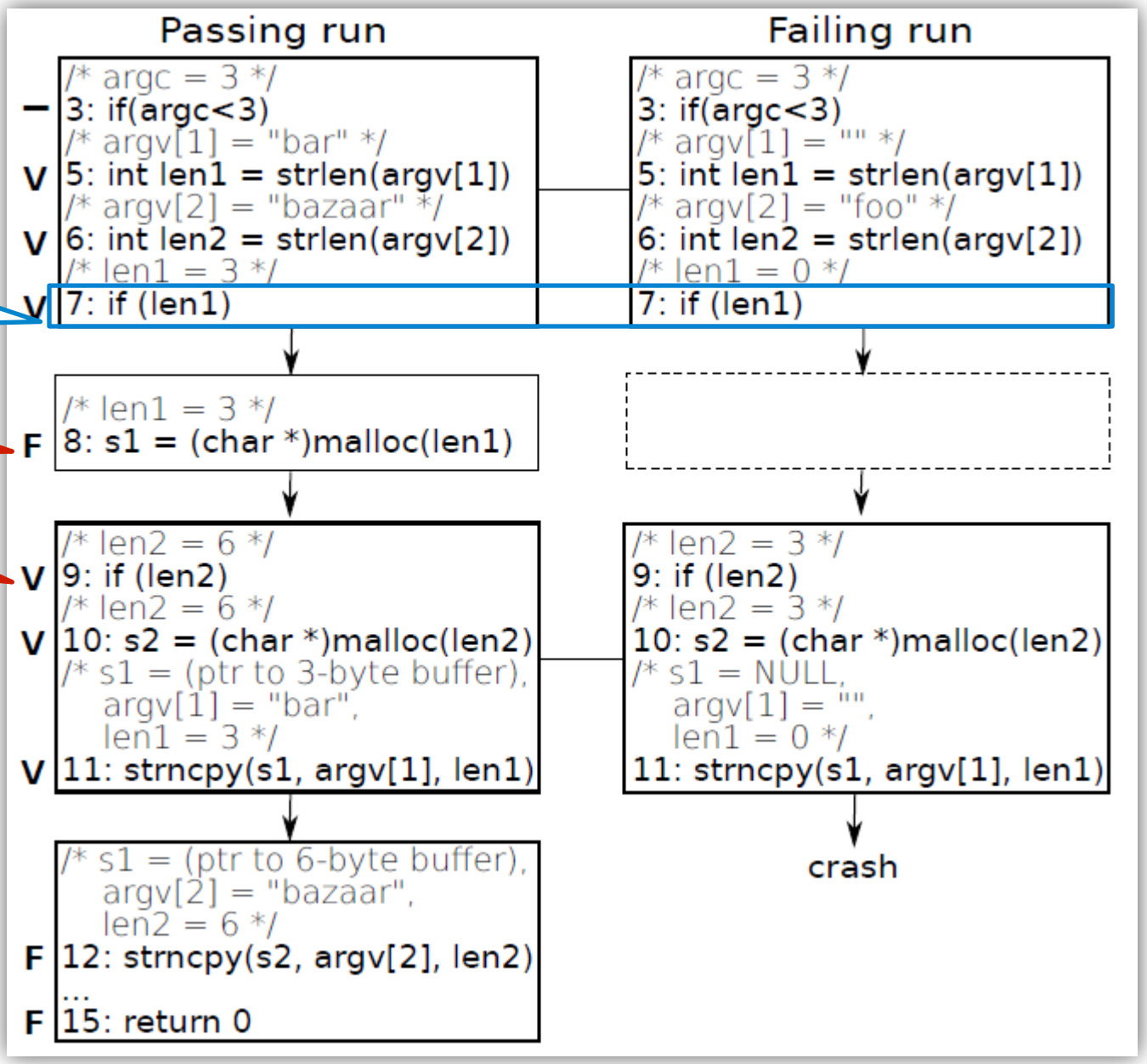
Aligned region

Disaligned region

```
1  char *s1=NULL, *s2=NUL
2  int main(int argc, cha
3    if(argc < 3)
4      return 1;
5    int len1 = strlen(ar
6    int len2 = strlen(ar
7    if (len1)
8      s1 = (char *)mallo
9    if (len2)
10     s2 = (char *)mallo
11   strncpy(s1, argv[1],
12   strncpy(s2, argv[2],
13   if (strcmp(s1,s2) !=
14     printf("Strings ar
15   return 0;
16 }
```

Figure 1: Motivating exam

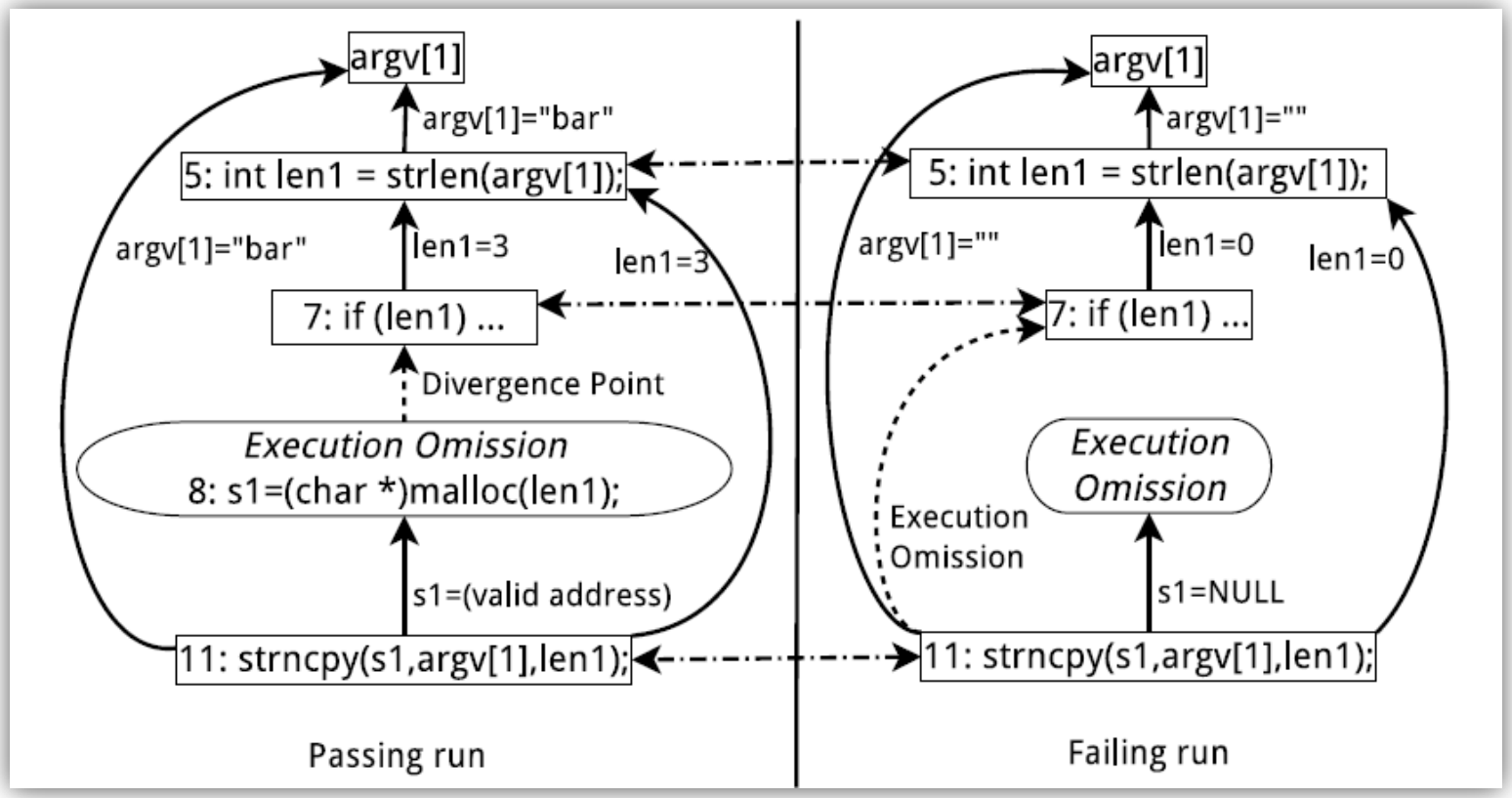




Flow differences = disaligned statements

Problem Definition and Overview

- Causal difference graph
  - The causal difference graph contains the sequences of execution differences leading from the input differences to the target differences.



Problem Definition and Overview

- 6k lines of Objective Caml code
  - Trace alignment and post-dominator module : 4k lines
  - Slice-Align module : 2k lines

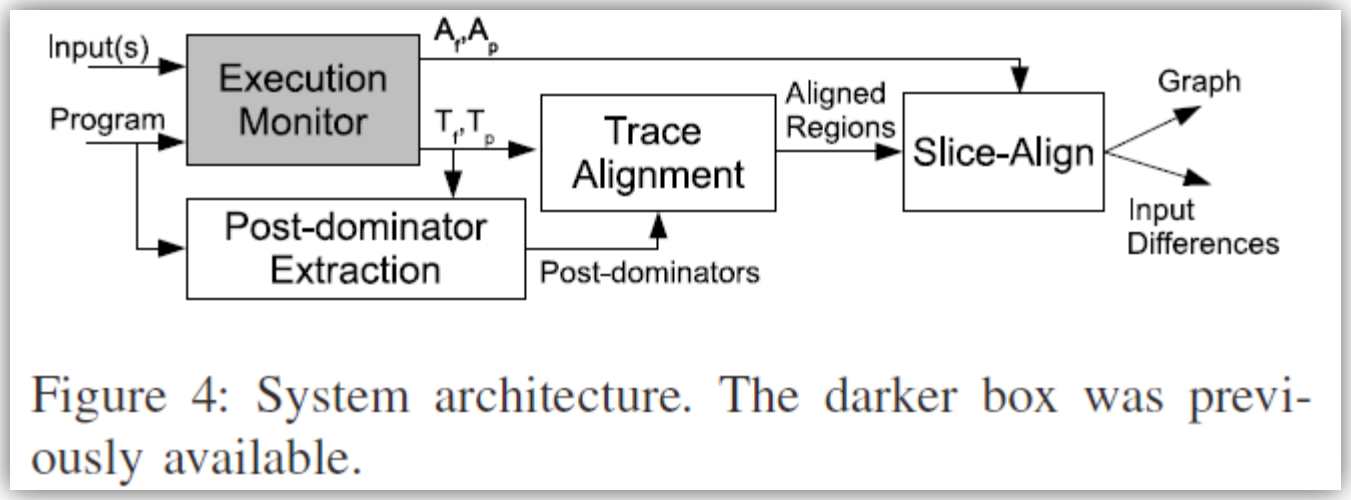
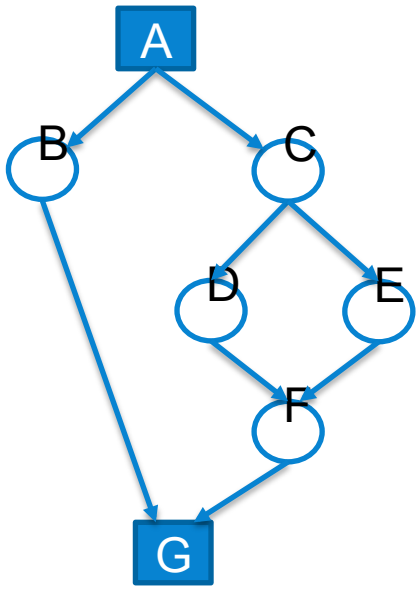


Figure 4: System architecture. The darker box was previously available.

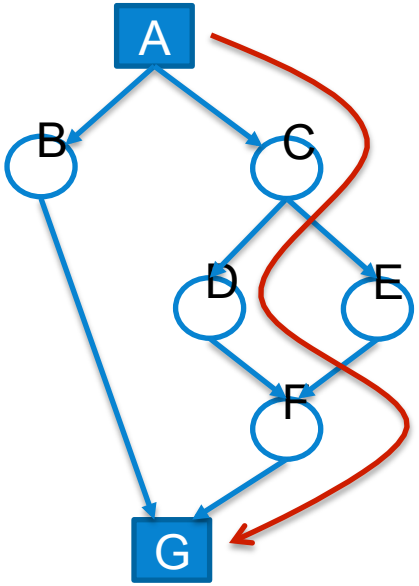
Trace Alignment

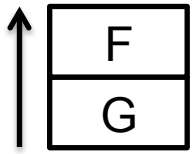
- Dominate
  - A node *d* dominates node *n* iff every path from entry node to *n* passes through *d.* (node *d* is a dominator of node *n*)
  - Node *id* immediately dominates *n* if *id* dominates *n*, and no other node *p* such that *id* dominates *p* and *p* dominates *n*. (*id* is the only immediate dominator of *n*)
- Post Dominate
  - Same as dominate, from node *n* to the exit node
  - Immediate post dominator



Trace Alignment

- Execution Indexing
  - Execution Indexing captures the structure of the program at any given point in the execution, identifying the execution point, and uses that structure to establish a correspondence between execution points across multiple executions of the program.
  - Xin et al. use an indexing stack to deal with branch or method call.



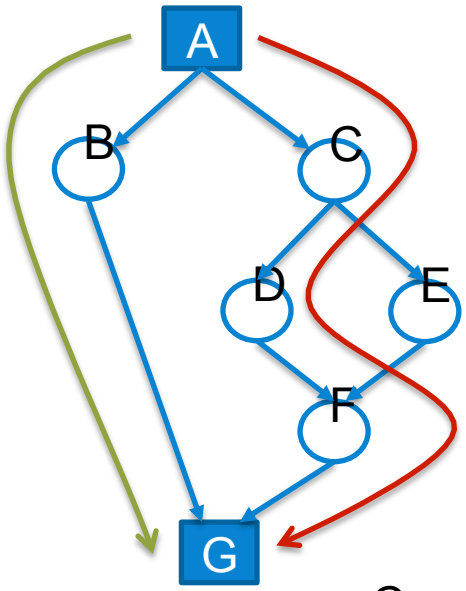


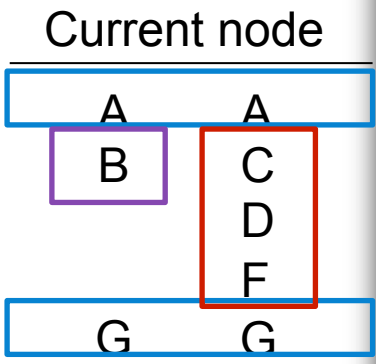
stack

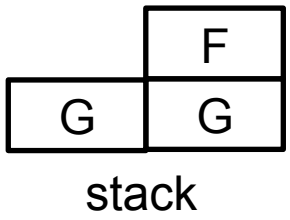
Current node

A
C
D
F
G

Trace Alignment







Input: A_0, A_1 *// anchor points*
Output: RL *// list of aligned and disaligned regions*
EI_0, EI_1 : *execution index stacks* $\leftarrow Stack.empty()$;
$insn_0, insn_1 \leftarrow A_0, A_1$; *// current instructions*
$RL \leftarrow \varnothing$;
while $insn_0, insn_1 \neq \perp$ **do**
  $cr \leftarrow$ regionBegin$(insn_0, insn_1,$ aligned$)$
  // Aligned-Loop: Traces aligned. Walk until disaligned
  while $EI_0 = EI_1$ **do**
    foreach $i \in 0, 1$ **do**
      $EI_i \leftarrow$ updateIndex$(EI_i, insn_i)$;
      $cr \leftarrow$ regionExtend$(insn_i, cr)$;
      $insn_i{+}{+}$;
    end
  end
  $RL \leftarrow RL \cup cr$;
  $cr \leftarrow$ regionBegin$(insn_0, insn_1,$ disaligned$)$
  // Disaligned-Loop: Traces disaligned. Walk until realigned
  while $EI_0 \neq EI_1$ **do**
    while $|EI_0| \neq |EI_1|$ **do**
      $j \leftarrow (|EI_0| > |EI_1|)\ ?\ 0 : 1$;
      while $|EI_j| \geq |EI_{1-j}|$ **do**
        $EI_j \leftarrow$ updateIndex$(EI_j, insn_j)$;
        $cr \leftarrow$ regionExtend$(insn_j, cr)$;
        $insn_j{+}{+}$;
      end
    end
  end
  $RL \leftarrow RL \cup cr$;
end

Figure 5: Algorithm for trace alignment.

Slice-Align

- worklist
  - A pool of instructions to be operated

Case		Name	Passing	Failing
1		Extra Execution	Aligned	Disaligned
2		Execution Omission	Disaligned	Aligned
3		Execution Difference	Disaligned	Disaligned
4		Invalid Pointer	Aligned	Aligned
	4a	Wild Read	Aligned	Aligned
	4b	Wild Write	Aligned	Aligned

Table I: The divergence types.

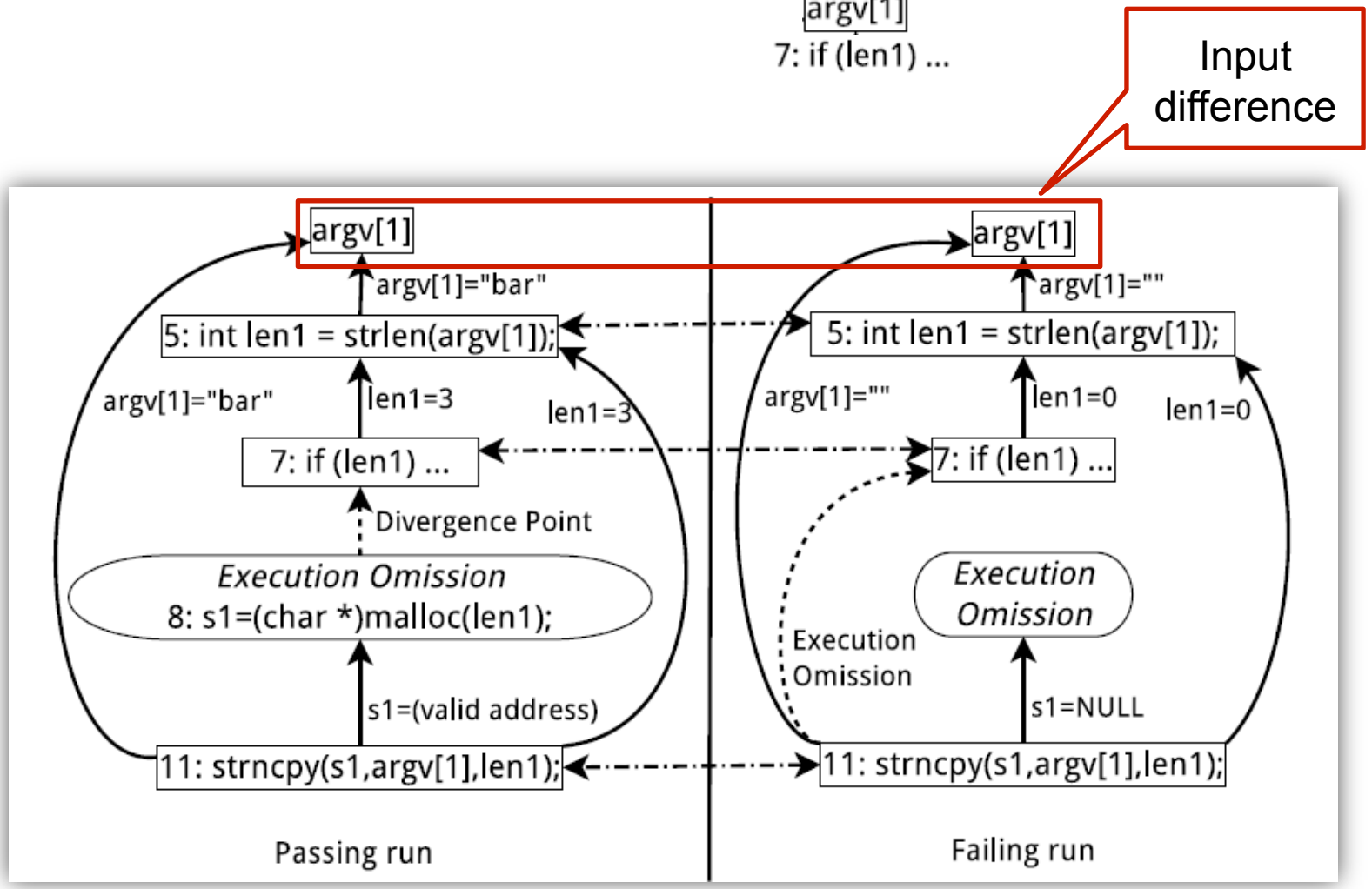
Input: *TD /* target difference */*, *RL /* alignment results */*
Output: N, E *// nodes and edges in causal difference graph*
worklist : stack of instruction-pairs $\leftarrow (TD_p, TD_f)$;
processed : boolean lookup table $\leftarrow \varnothing$;
while !*worklist.isEmpty()* **do**
  $(insn_p, insn_f) \leftarrow worklist$.pop();
  $N_{p,f} \leftarrow N_{p,f} \cup insn_{p,f}$;
  $processed(insn_p, insn_f) \leftarrow true$;
  if isAligned$(insn_p, insn_f, RL)$ **then**
    $slice\_operands \leftarrow$ valDifferences$(insn_p, insn_f)$;
    forall $operand \in slice\_operands$ **do**
      $dep \leftarrow$ immDataDeps$(operand)$;
      $E_{p,f} \leftarrow E_{p,f} \cup$ newEdge$(insn_{p,f} \rightarrow dep_{p,f})$;
      if !$processed(dep_p, dep_f)$ **then**
        $worklist$.push(dep_p, dep_f);
    end
  end
  else
    $dtype \leftarrow$ divRegionType$(insn_p, insn_f, RL)$;
    switch *dtype* **do**
      // See Table I for explanation of divergence types
      case *ExtraExec* **or** *ExecOmission* **or** *ExecDiff*
        $div \leftarrow$ domDivPt$(dtype, insn_p, insn_f, RL)$;
        $E_{p,f} \leftarrow E_{p,f} \cup$ newEdge$(insn_{p,f} \rightarrow div_{p,f})$;
        if !$processed(div_p, div_f)$ **then**
          $worklist$.push(div_p, div_f);
      end
      case *InvalidPointer*
        if wildWrite$(insn_p, insn_f)$ **then**
          $aligned_p \leftarrow$ alignedInsn$(insn_f, RL)$;
          if !$processed(aligned_p, insn_f)$ **then**
            $worklist$.push$(aligned_p, insn_f)$;
        end
      end
    end
  end
end

Figure 6: Algorithm for Basic graph.

Slice-Align

worklist

11: strncpy(s1,argv[1],len1);
8: s1 = (char *)malloc(len1)
5: int len1 = strlen(argv[1]);
argv[1]
7: if (len1) ...



Slice-Align

- Edge pruning and address normalization
  - Pruning edges in the graph when an operand of an aligned instruction has the same value in both execution traces.
  - Heap pointer pruning
    - The pointer is pruned if
      1. The allocation site for the live buffers that contain the pointed-to addresses are aligned
      2. The offset of those pointed-to addresses, with respect to the start address of the live buffer they belong to, is the same
  - Stack pointer pruning
    - (in the thread stack range) normalized by subtracting the stack base address
  - Data section pointer pruning
    - (in the same module) normalized by subtracting the module base address

Evaluation

Name	Program	Vuln. CVE	OS
reader-e1	Adobe Reader 9.2.0	Unknown	XP SP3
reader-e2	Adobe Reader 9.2.0	Unknown	XP SP3
reader-u1	Adobe Reader 9.2.0	Unknown	XP SP3
reader-u2	Adobe Reader 9.2.0	Unknown	XP SP3
reader-u10	Adobe Reader 9.2.0	Unknown	XP SP3
reader-u11	Adobe Reader 9.2.0	Unknown	XP SP3
reader-u14	Adobe Reader 9.2.0	Unknown	XP SP3
firebird	Firebird SQL 1.0.3	2008-0387	XP SP2
gdi-2008	gdi32.dll v2180	2008-3465	XP SP2
gdi-2007	gdi32.dll v2180	2007-3034	XP SP2
tftpd	TFTPD32 2.21	2002-2226	XP SP3
conficker	W32/Conficker.A	N/A	XP SP3
netsky	W32/Netsky.C	N/A	XP SP3

Table II: Programs and vulnerabilities in the evaluation.

Evaluation

- Evaluating the Causal Difference Graph

Name	Total instructions		Disaligned instructions		Disaligned regions	
	Passing	Failing	Passing	Failing	All	Slice-Align
reader-e1	2,800,163	1,819,714	1,307,465	327,016	983	471
reader-e2	1,616,642	1,173,531	446,273	3,162	75	5
reader-u1	2,430,400	1,436,993	2,034,582	1,041,175	111	32
reader-u2	1,921,514	1,053,840	656,183	14,586	38	23
reader-u10	408,618	272,994	144,517	8,893	39	4
reader-u11	1,868,942	1,112,828	1,504,189	748,075	389	235
reader-u14	1,194,053	155,906	601,789	119,085	524	59
tftpd	626,622	350,323	415,086	138,787	87	4
firebird	6,698	1,282	5,551	135	4	4
gdi-2008	42,124	4,310	38,743	929	1	1
gdi-2007	36,792	4,310	33,508	1,026	1	1

Table III: Total disaligned instructions and regions compared with disaligned regions in graph.

Evaluation

- Graph size
  - #IDiff = number of input differences

Name	Basic pruning			Extended pruning		
	Pass	Fail	# IDiff	Pass	Fail	# IDiff
reader-e1	3,651	3,616	7	2,324	2,292	7
reader-e2	4,854	4,853	21	81	84	1
reader-u1	2,753	2,751	13	204	201	1
reader-u2	135	135	1	100	100	1
reader-u10	45	43	1	36	34	1
reader-u11	1,584	1,562	1	1,158	1,135	1
reader-u14	1,714	1,695	6	425	420	1
tftpd	254	254	1	254	254	1
firebird	45	46	1	45	46	1
gdi-2008	100	101	1	96	97	1
gdi-2007	11	12	1	7	8	1

evaluation

- Performance
  - Less than 1 hour to generate a graph

Name	Trace size (MB)		Tracing (sec.)		Trace align (sec.)	Slice-Align (sec.)
	Pass	Fail	Pass	Fail		
reader-e1	202	106	482	365	1,684	3,510
reader-e2	143	67	345	337	1,180	1,291
reader-u1	200	133	403	406	714	101
reader-u2	110	61	208	295	152	208
reader-u10	24	16	267	275	39	24
reader-u11	152	101	155	161	462	364
reader-u14	160	107	195	192	837	239
tftpd	3.6	2.0	13	12	50	12
firebird	2.5	0.1	1	1	1	0.2
gdi-2008	2.4	0.4	2	0.8	2	0.5
gdi-2007	2.1	0.4	2	0.8	2	0.3

Table V: Performance evaluation.

Evaluation

- User Study(informal)
  - Subject A: an analyst at a commercial security research company
  - Subject B: a research scientist

Subj.	Sample 1 (no graph)			Sample 2 (Causal difference graph)		
	sample	time (hr)	found cause?	sample	time (hr)	found cause?
A	reader-e2	13	✓	reader-u10	5.5	✓
B	reader-u10	3	✗	reader-e2	3	✓

Table VI: Results for user study.

Evaluation

- Identifying input differences in malware analysis
- W32/Conficker.A
  - Keyboard layout: Ukrainian(failing trace), US-English(passing trace)
  - Target difference: `CreateThread` API call
  - Result:
    - Input difference: `user32.dll::GetKeyboardLayoutList` function return value
- W32/Netsky.C
  - Makes the computer speaker beep continuously if the system time between 6am and 9pm on Feb. 26, 2004
  - Target Difference: Beep function call
  - Resault:
    - Input difference: `kernel32.dll::GetLocalTime` system call

Conclusion

- Producing causal difference graph
  - Input difference information
  - Execution difference from input difference to target difference
- Reducing the graph size
- Reducing the input difference candidates