POMP: Postmortem Program Analysis with Hardware-Enhanced Post-Crash Artifacts

Jun Xu[1], Dongliang Mu[12], Xinyu Xing[1], Peng Liu[1], Ping Chen[1], Bing Mao[2]

1. Pennsylvania State University
2. Nanjing University

Grim Reality

- Despite intensive in-house software testing, programs inevitably contain defects
  - Accidentally terminate or crash at post-deployment stage
  - Exploited as security loopholes
- Software debugging is expensive and needs intensive manual efforts
  - Debugging software cost has risen to $312 billion per year globally [1]
  - Developers spend 50% of their programming time finding and fixing bugs [2]

[1] http://silvaetechnologies.eu/blg/216/debugging-software-cost-has-risen-to-$312-billion-per-year-globally
[2] https://www.roguewave.com/company/news/2013/university-of-cambridge-reverse-debugging-study

Techniques facilitating software debugging

- Heavyweight technique
  - Program tracing
  - Relay debugging
- Lightweight technique
  - Examine stack trace in debugger (e.g. [1])
  - Audit execution logs (e.g., [2])
  - Analyze crash dump (e.g., [3])

	Cost	Capability
Heavyweight tech.	High overhead	High effectiveness
Lightweight tech.	Low overhead	Low effectiveness

	Trace execution	Log program states
Heavyweight tech.		
Lightweight tech.		

[1] Liblit et al. Building a Better Backtrace: Techniques for Postmortem Program Analysis. TR'02
[2] Yuan et al. Improving Software Diagnosability via Log Enhancement. ASPLOS'11
[3] Cui et al. RETracer: Triaging Crashes by Reverse Execution from Partial Memory Dumps. ICSE'16

Ultimate Goals

- Automated
  - Automating software failure diagnosis without manual efforts
- Accurate
  - Do not mistakenly pinpointing the root causes of software failure
- Nonintrusive (Runtime overhead free)
  - Introducing no runtime overhead at post-deployment stage

POMP for Postmortem Program Analysis

- POMP diagnoses software crash using software crash report (core dump)
- POMP introduces no overhead at a running software because a crash report is created only when a program accidentally terminates

Core dump

- In general, a core dump carries information such as
  - Memory snapshot
  - Registers
  - Others (e.g., signals received)
- A core dump provides only a partial chronology of how a program reached a crash site [3]
- A core dump is barely used a source for automated software failure diagnosis

[3] Liblit et al. Building a Better Backtrace: Techniques for Postmortem Program Analysis

Hardware-enhanced Core dump

- POMP enhances a core dump through Intel Processor Trace (PT)
- Intel PT enables execution tracing with nearly no overhead
- A hardware-enhanced core dump unveils not only a crashing state but more importantly captures the control flow a crashing program follows.

Register	eax	0x02
	ebx	0xff28
	esp	0xff14
Mem Address	0xff1c	0x02
	0xff18	0x01
	0xff14	0x00
	0xff10	0xff18

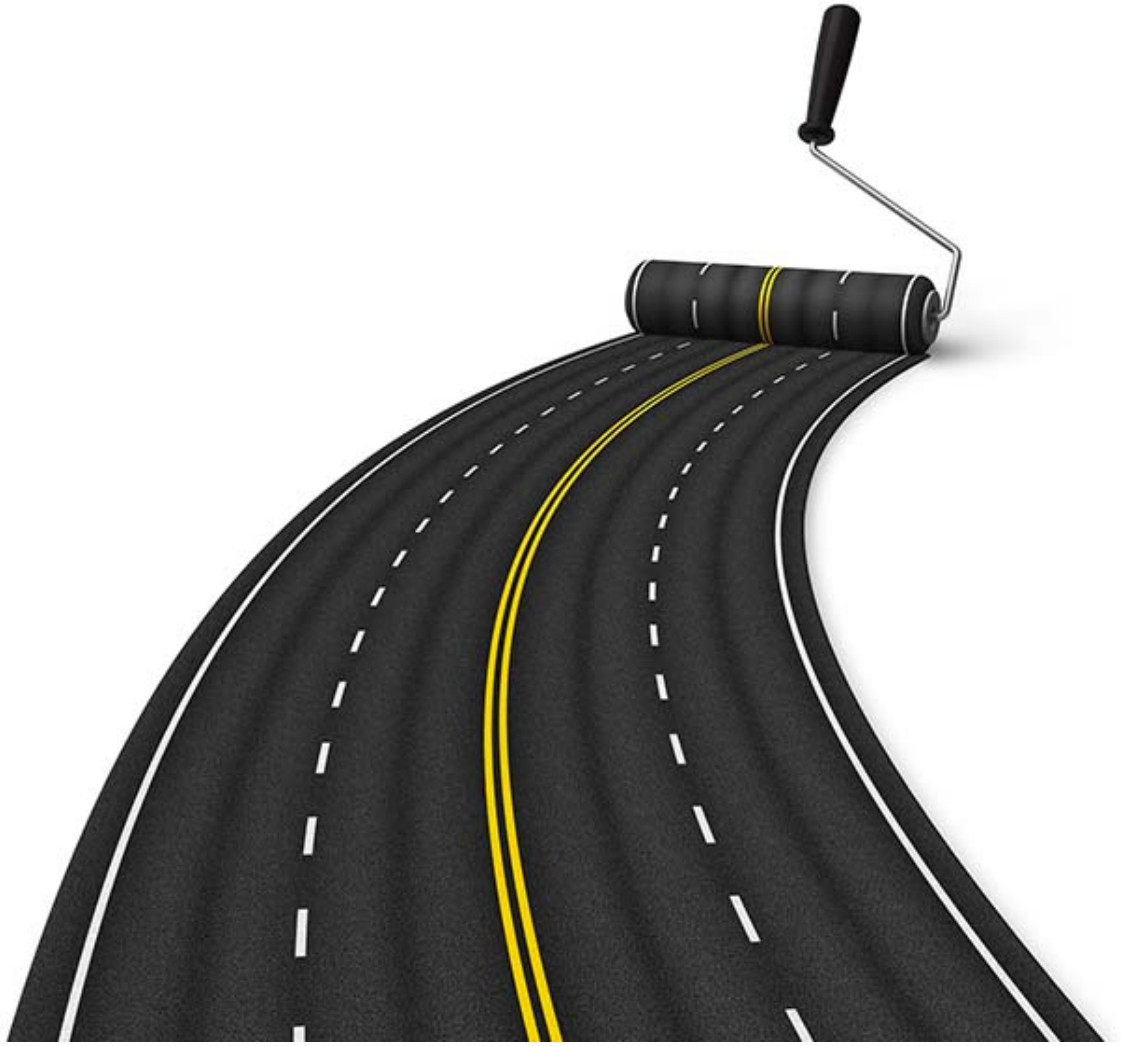
Memory state

A1: push ebp
A2: mov ebp, esp
A3: sub esp, 0x14
A4: mov [ebp-0xc], test
A5: lea eax, [ebp-0x10]
A6: push eax
A7: call child
A8: push ebp
A9: mov ebp, esp
…

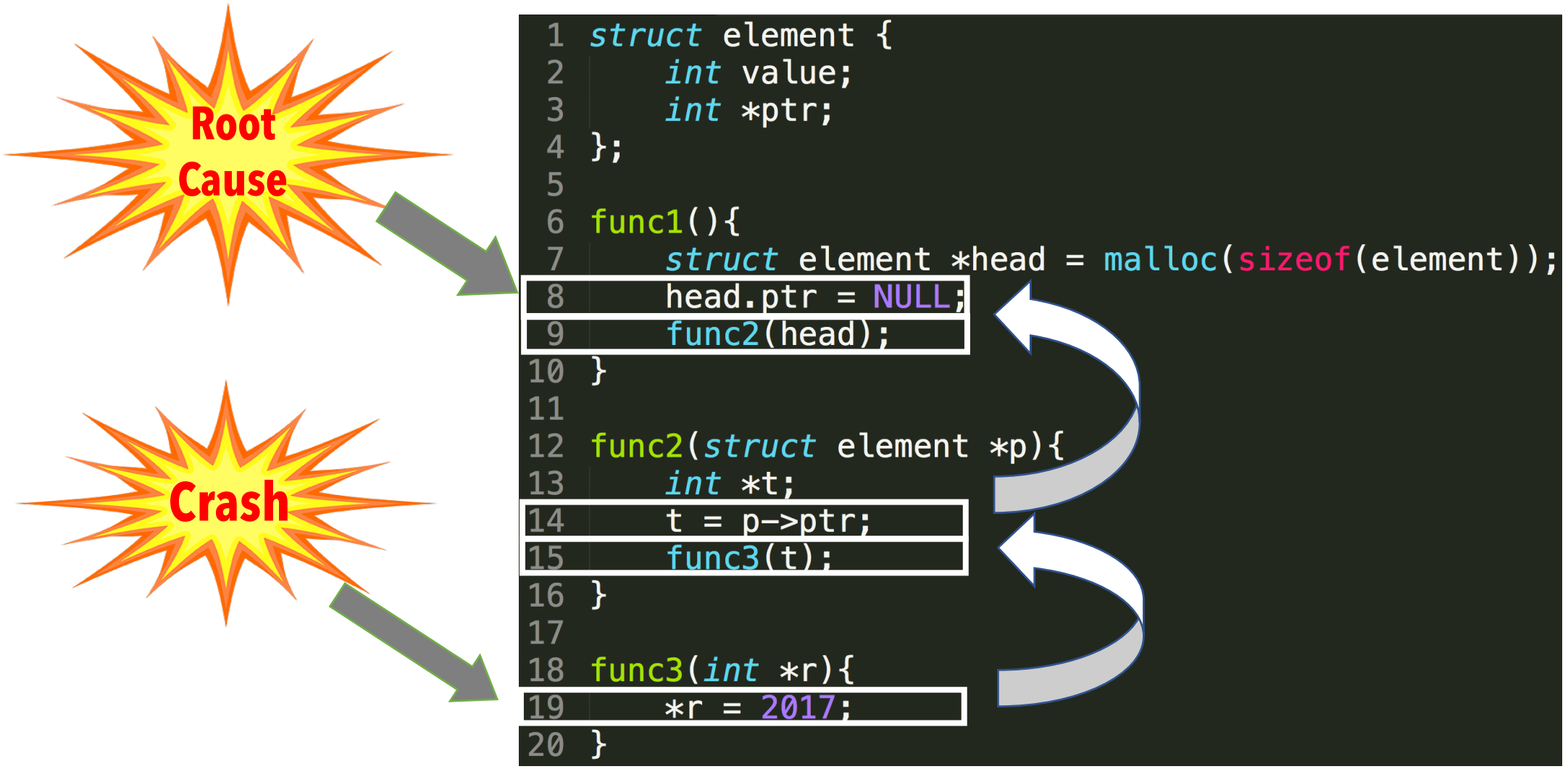
Execution trace

Roadmap

- Design overview
- Technical challenge
- Technical details
- Evaluation in real world crashes
- Summary



Example of Program Failure Debugging

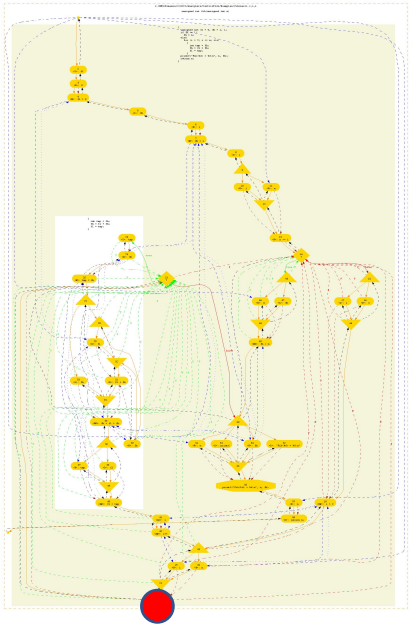


Overview of Automated Program Failure Diagnosis

1) Reversely execute an instruction trace (starting from the crash site)
2) Reconstruct the data flow that a program followed prior to its crash
3) Examine how a bad value was passed to the crash site

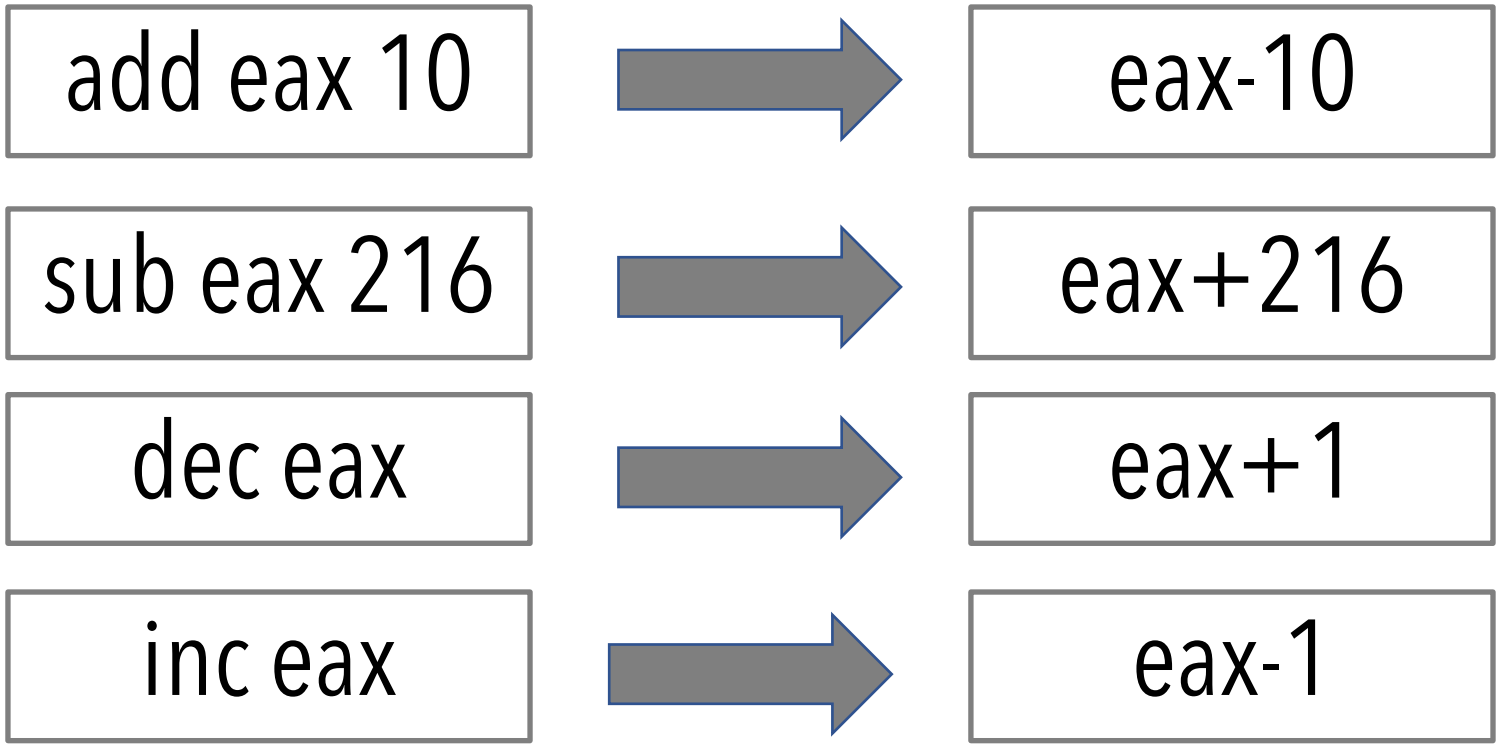
Reversely executing

```
push   ebp
mov    ebp,esp
and    esp,0xfffffff0
sub    esp,0x20
mov    DWORD PTR [esp+0x1c],0x0
cmp    DWORD PTR [ebp+0x8],0x1
jg     0x8048500 <main+57>
mov    eax,DWORD PTR [ebp+0xc]
mov    edx,DWORD PTR [eax]
mov    eax,0x8048622
mov    DWORD PTR [esp+0x4],edx
mov    DWORD PTR [esp],eax
call   0x8048388 <printf@plt>
mov    DWORD PTR [esp],0x0
call   0x80483b8 <exit@plt>
mov    eax,DWORD PTR [ebp+0xc]
add    eax,0x4
mov    eax,DWORD PTR [eax]
mov    DWORD PTR [esp+0x4],0x8048638
mov    DWORD PTR [esp],eax
call   0x8048498 <IsPasswordCorrect>
mov    DWORD PTR [esp+0x1c],eax
cmp    DWORD PTR [esp+0x1c],0x1
jne    0x804852a <main+99>
call   0x8048484 <UnlockSecret>
jmp    0x8048536 <main+111>
mov    DWORD PTR [esp],0x8048640
call   0x8048398 <puts@plt>
mov    eax,0x0
```



Data flow

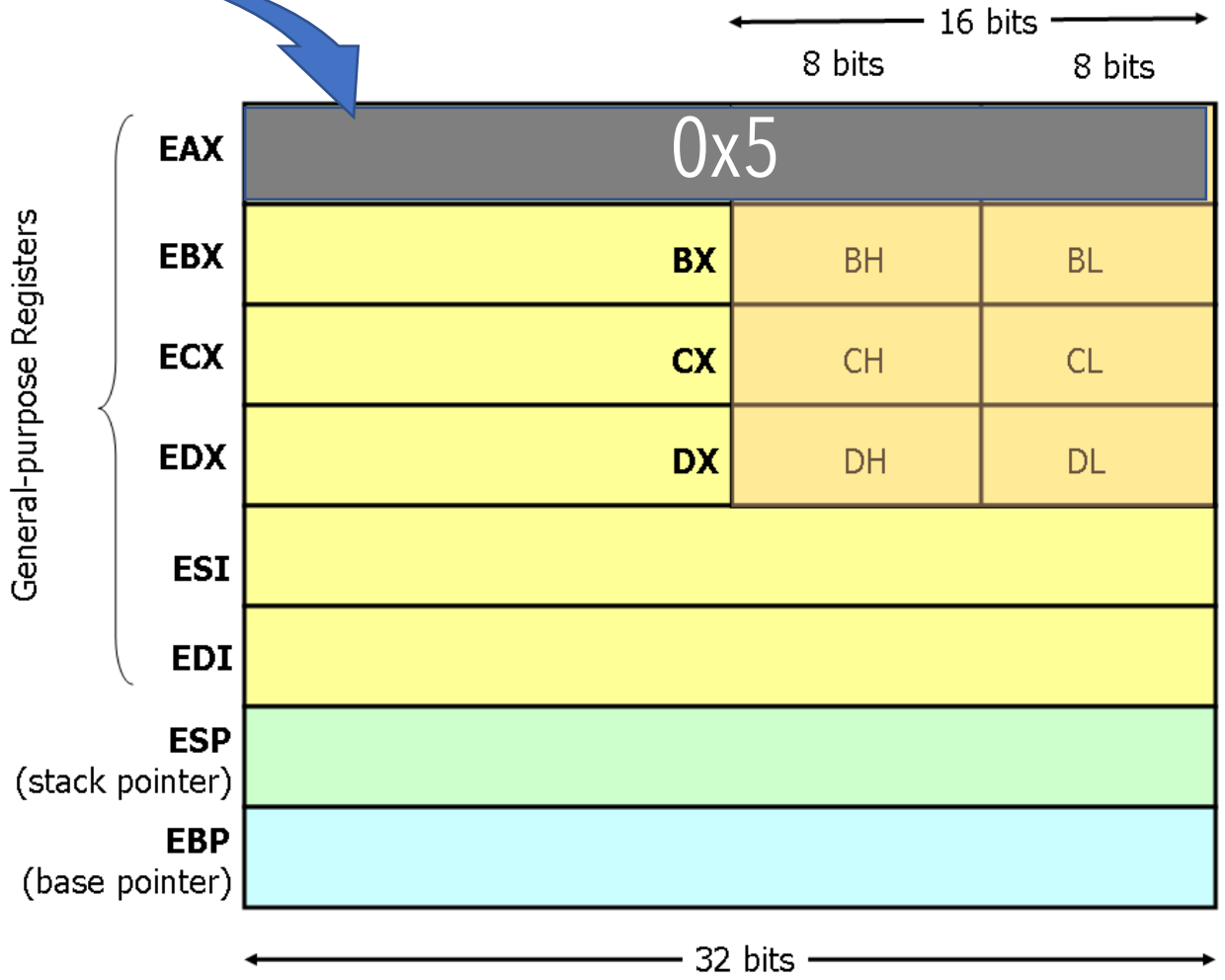
Reverse Execution – Invertible Instructions



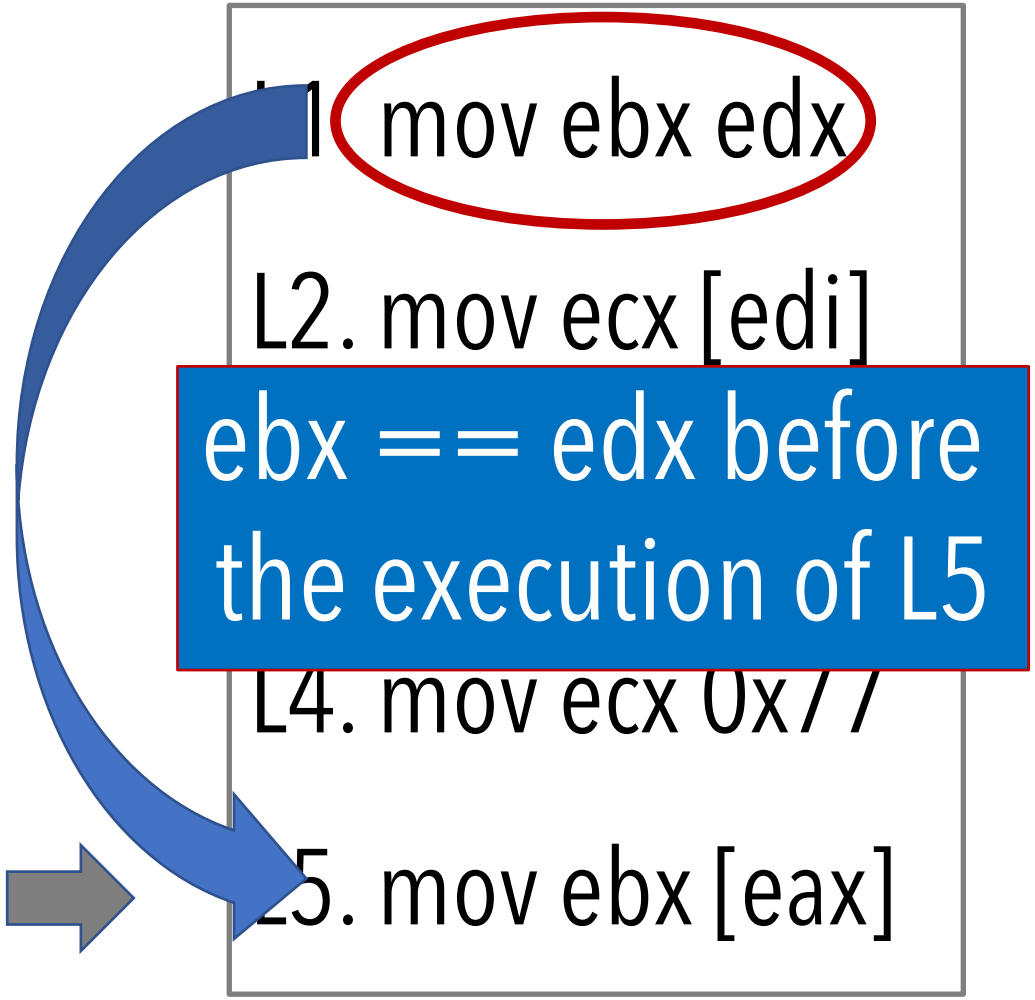
Reverse Execution – Non-invertible Instructions

mov eax 0x5

xor ecx ecx

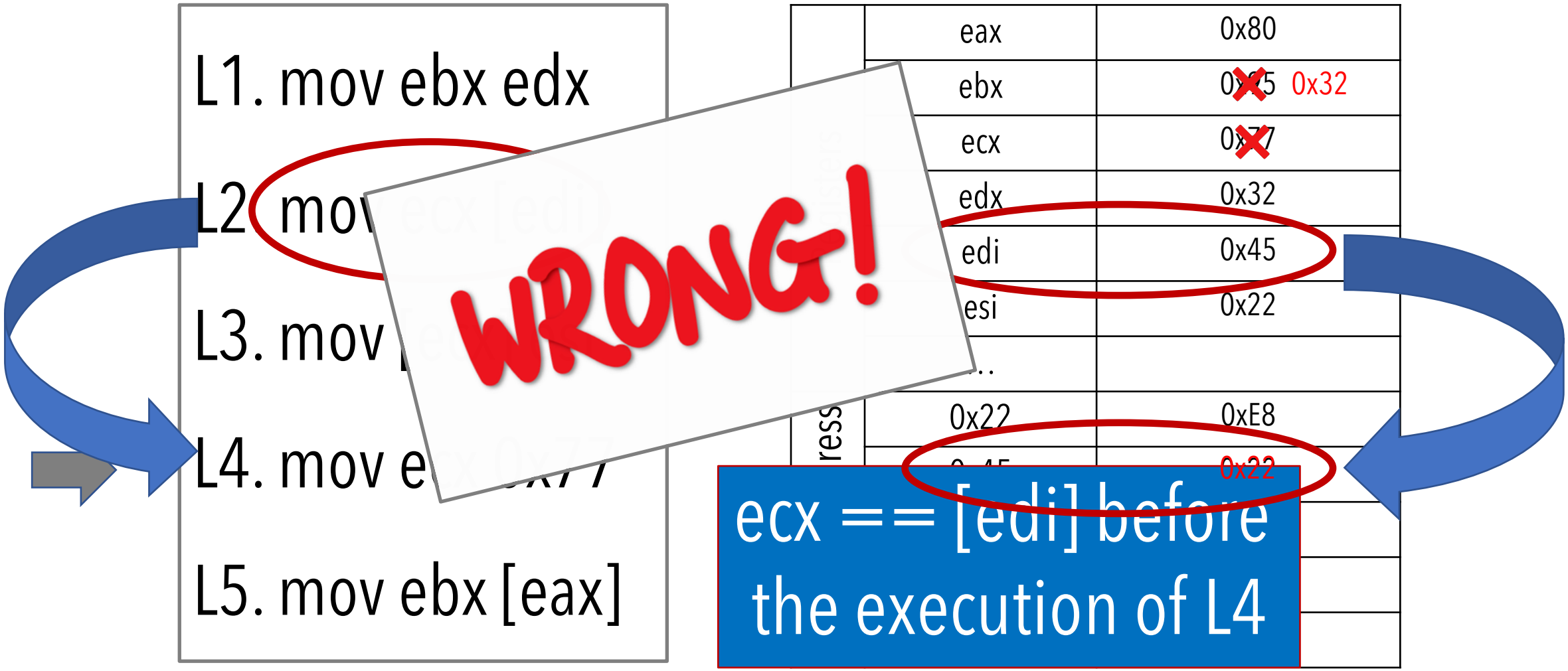


Reverse Execution with Backward Data Flow Analysis



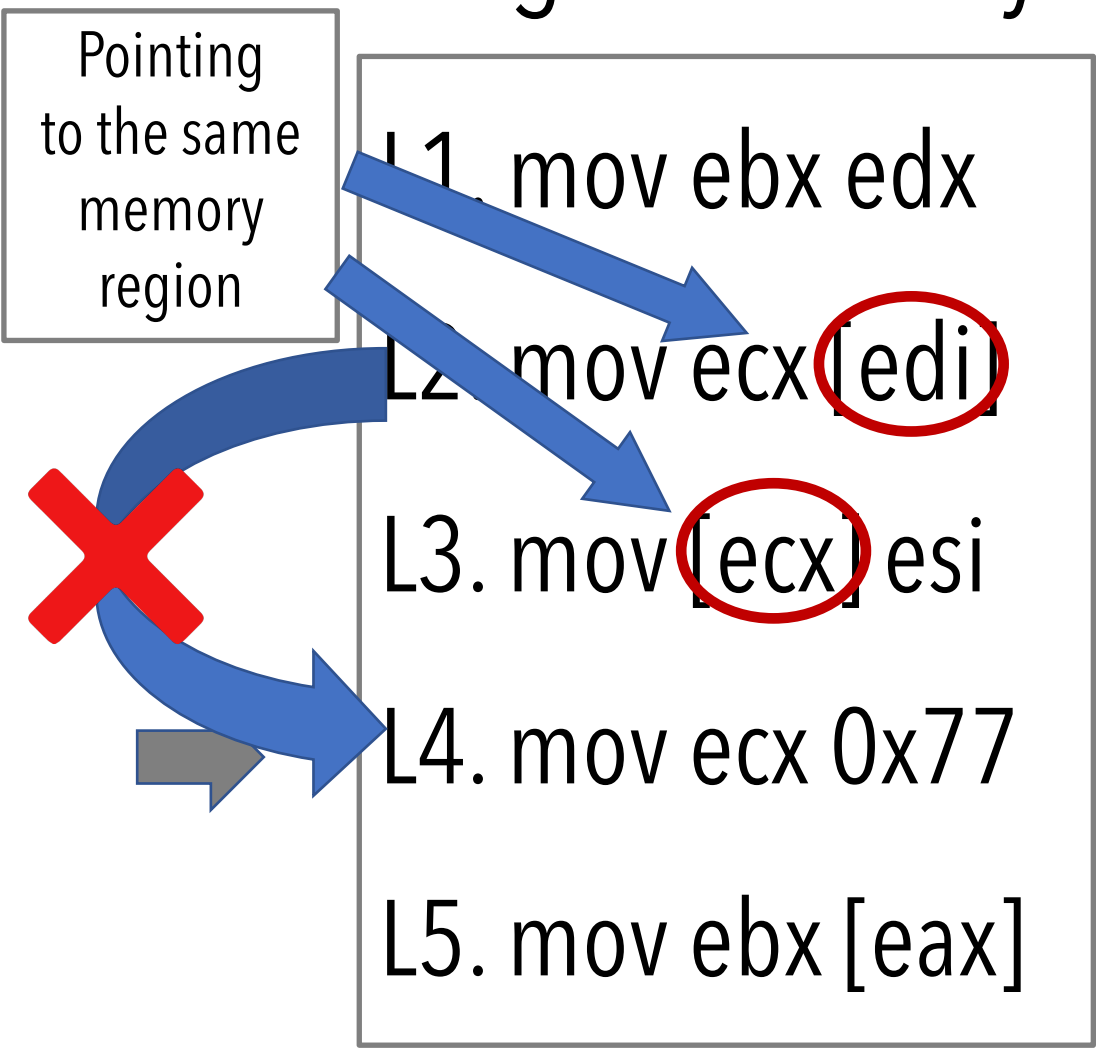
Registers	eax	0x80
	ebx	0x45
	ecx	0x77
	edx	0x32
	edi	0x45
	esi	0x22
	…	
Memory Address	0x22	0xE8
	0x45	0x22
	0x77	0xB7
	0x80	0x95
	…	

Challenge – Memory Alias Issue



Challenge – Memory Alias Issue

Pointing to the same memory region

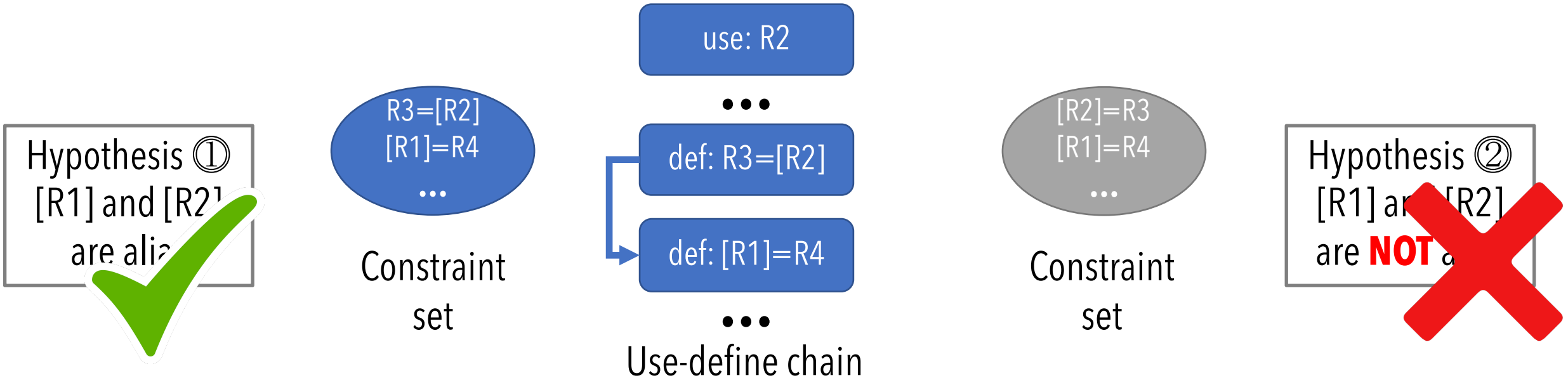


Registers	eax	0x80
	ebx	0x45 0x32
	ecx	0x77
	edx	0x32
	edi	0x45
	esi	0x22
	…	
ress	0x22	0xE8

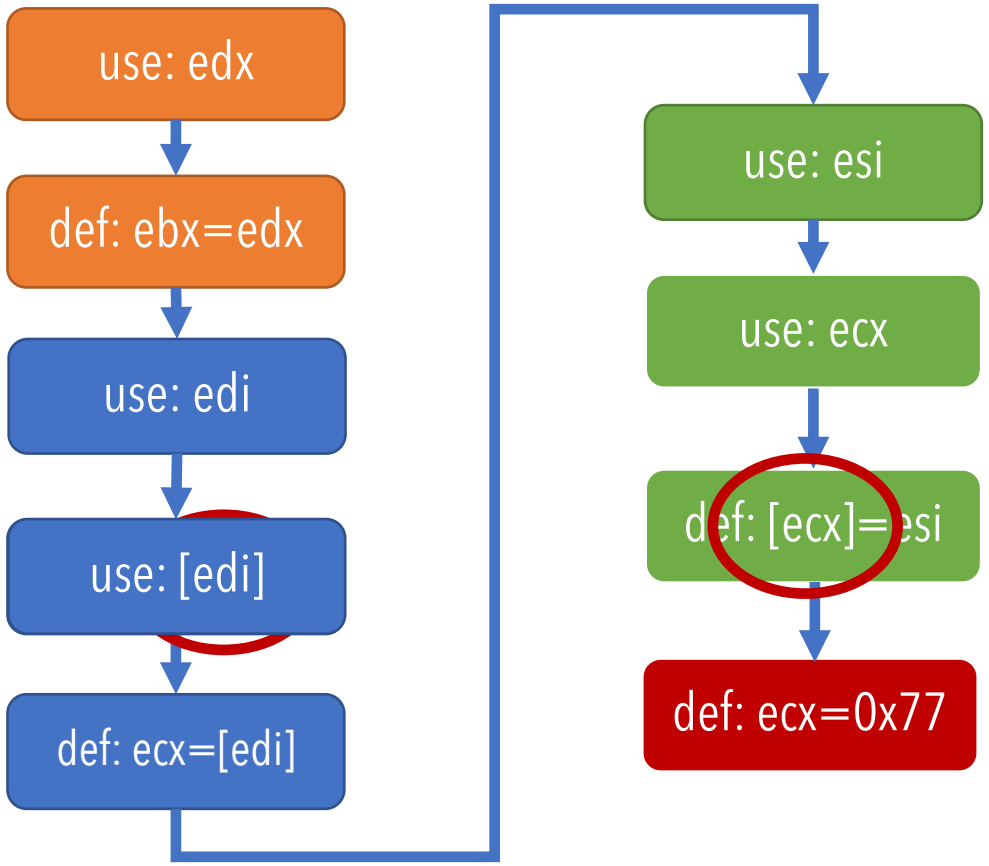
edi == ecx right before the execution of L4

Hypothesis Testing

1) Making two hypotheses for each pair of memory
2) Constructing use-define chain and extracting constraints under each hypothesis
3) Testing each of the hypotheses by using the constraints in each set



Hypothesis Testing



Hypothesis ①
NOT alias

ecx≠edi
ecx=[edi]
[ecx]=esi

Hypothesis ②
alias

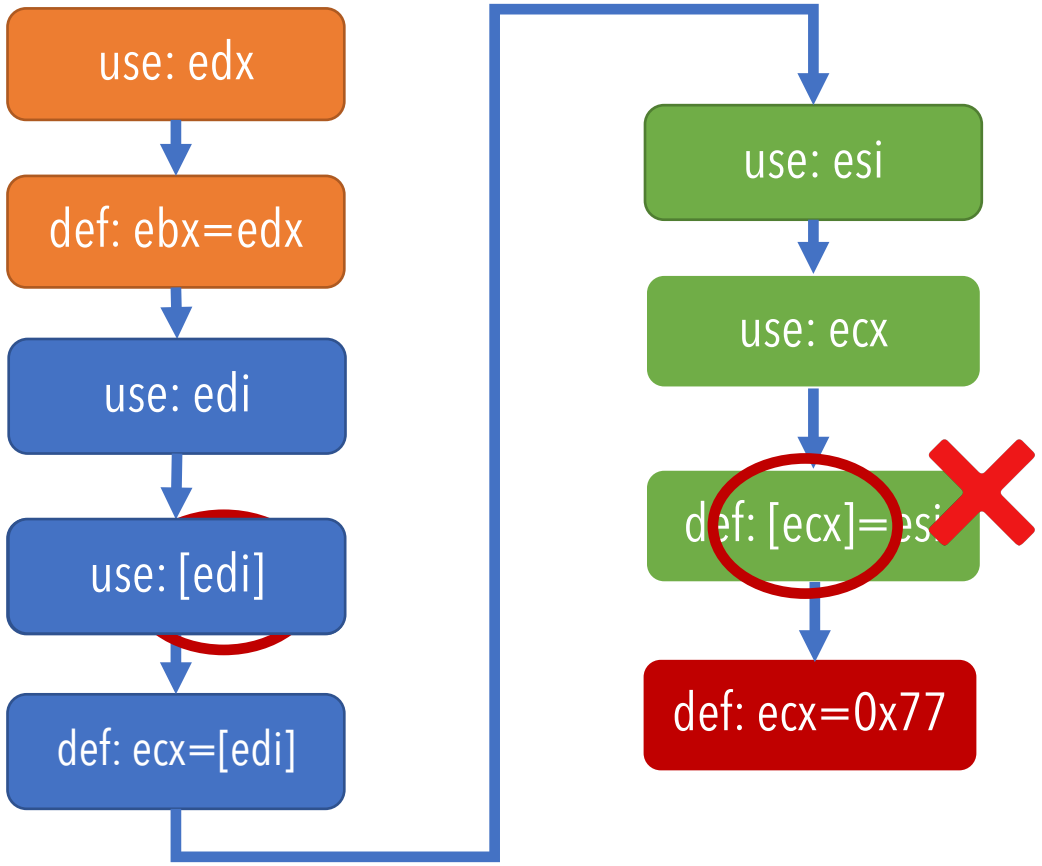
L1. mov ebx edx

L2. mov ecx [edi]

L3. mov [ecx] esi

L4. mov ecx 0x77

Hypothesis Testing



Hypothesis ①
NOT alias

ecx=[edi]
ecx≠edi
[ecx]=esi

Hypothesis ②
alias

ecx=[edi]
ecx=edi
[ecx]=esi

L1. mov ebx edx

L2. mov ecx [edi]

L3. mov [ecx] esi

L4. mov ecx 0x77

Hypothesis Testing

Registers	eax	0x80
	ebx	0x~~45~~ 0x32
	ecx	0x~~77~~ ????
	edx	0x32
	edi	0x45
	esi	0x22
	…	
Memory Address	0x22	0xE8
	0x45	0x~~22~~
	0x77	0xB7
	0x80	0x95
	…	

Hypothesis ①
NO[illegible]s

ecx=[edi]
ecx≠edi
[ecx]=esi

Hypothesis ②
[illegible]s

ecx=[edi]
ecx=edi
[ecx]=esi

Goal: Restore the value of ecx prior to the execution of L4

L1. mov ebx edx

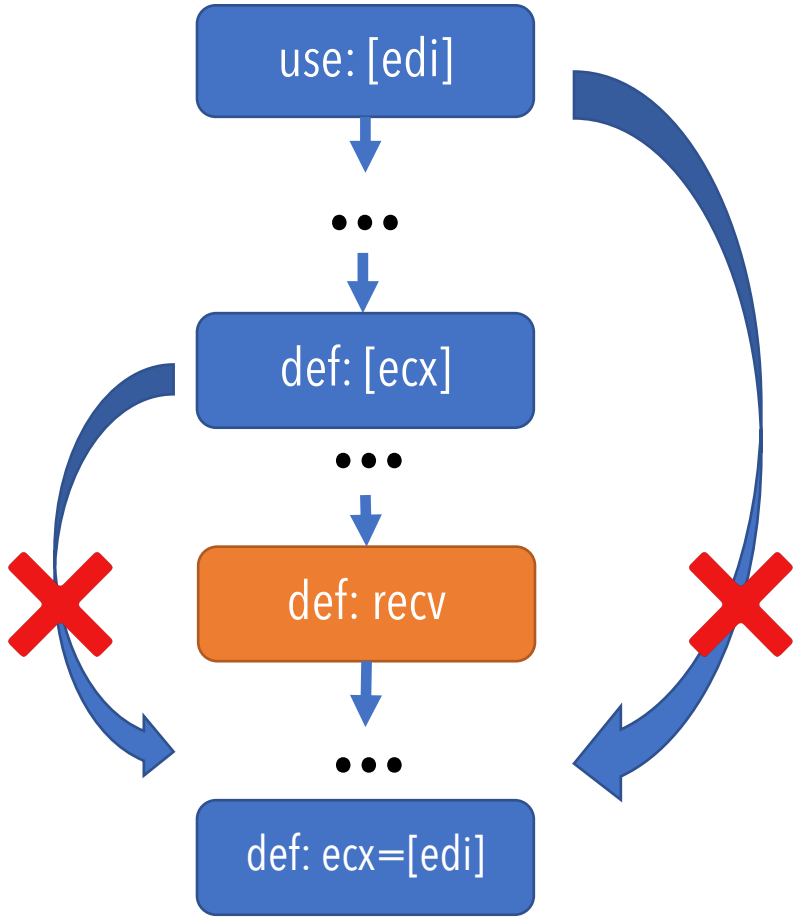
L2. mov ecx [edi]

L3. mov [ecx] esi

L4. mov ecx 0x77

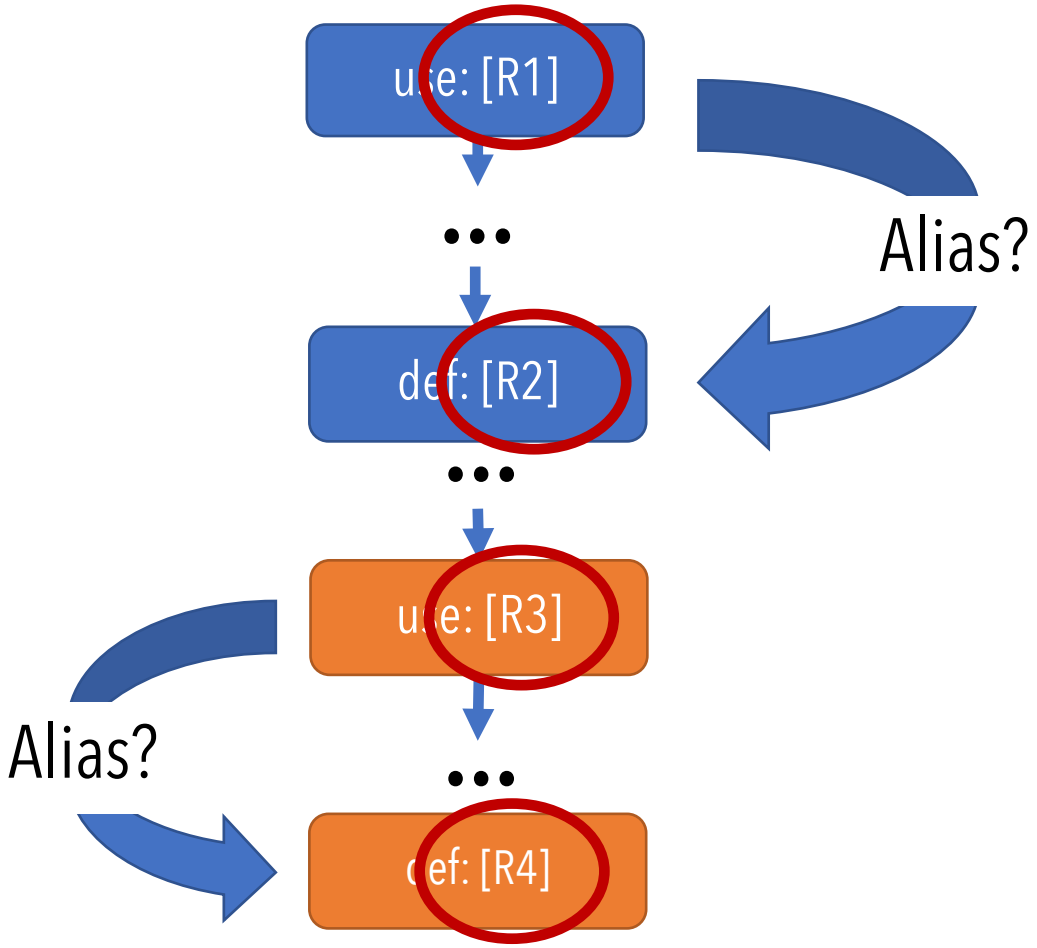
Limitations of Hypothesis Test and Strategy (1)

- Limitation: Not having sufficient evidences to reject a hypothesis because
  - Execution trace includes system calls
  - POMP does not trace execution into OS kernel
  - POMP may be unable to perform accurate data flow analysis
- Strategy: Treat some system calls (e.g., recv / write) as a definition which intervenes all the memory access propagation because
  - a non-deterministic memory region can potentially overlap with any memory regions in user space



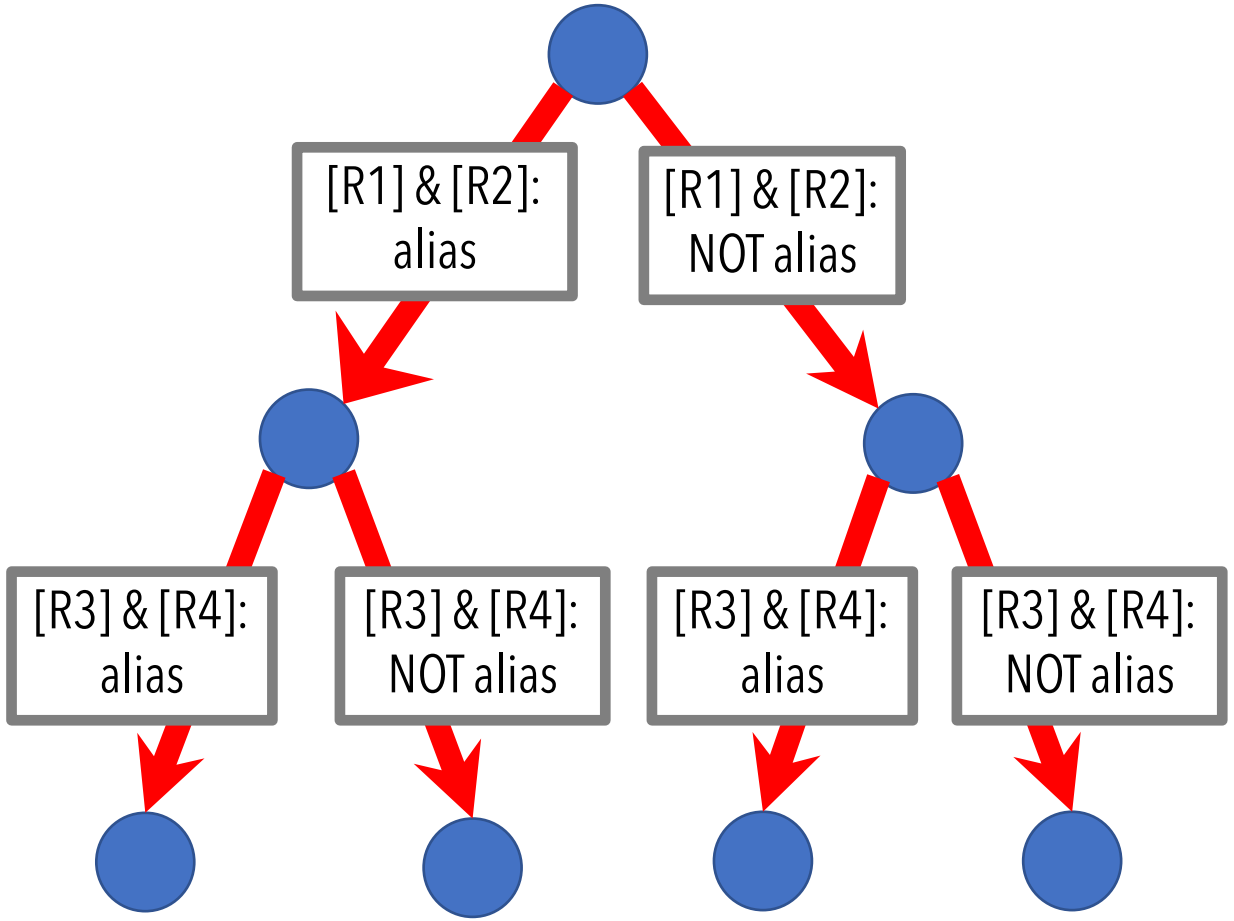
Limitations of Hypothesis Test and Strategy (2)

- Limitation: Require intensive computation to reject/accept a hypothesis because
  - One hypothesis test could involve other hypothesis tests, making hypothesis test a recursive procedure
  - In theory, recursive hypothesis test could potentially introduce a computation complexity of $O(n^m)$
- Strategy: Perform a recursive hypothesis test with limited number of recursion depths (m=2 in our implementation)



Limitations of Hypothesis Test and Strategy (2)

- Limitation: Require intensive computation to reject/accept a hypothesis because
  - One hypothesis test could involve other hypothesis tests, making hypothesis test a recursive procedure
  - In theory, recursive hypothesis test could potentially introduce a computation complexity of $O(n^m)$
- Strategy: Perform a recursive hypothesis test with limited number of recursion depths (m=2 in our implementation)



Evaluation of POMP

- 31 program crashes resulting from real world vulnerabilities:
  - Stack, heap & integer overflow
  - Use After Free and invalid free
  - Null pointer dereference
- Crashing program ranges from
  - Sophisticated software (e.g., GBD)
  - Lightweight software (e.g., corehttp)

Program		*Vulnerability*		*Diagnose Results*						
Name	Size(LoC)	CVE-ID	Type	Trace length	Size of mem (MB)	# of taint	Ground truth	Mem addr unknown	Root cause	Time
coreutils-8.4	138135	2013-0222	Stack overflow	50	56.61	3	2	1	✓	1 sec
coreutils-8.4	138135	2013-0223	Stack overflow	90	59.66	2	2	0	✓	1 sec
coreutils-8.4	138135	2013-0221	Stack overflow	92	120.95	3	2	0	✓	1 sec
mcrypt-2.5.8	37439	2012-4409	Stack overflow	315	0.59	3	2	3	✓	3 sec
BinUtils-2.15	697354	2006-2362	Stack overflow	867	0.37	16	7	0	✓	1 sec
unrtf-0.19.3	5039	NA	Stack overflow	895	0.34	7	4	10	✓	1 min
psutils-p17	1736	NA	Stack overflow	3123	0.34	7	3	28	✓	4 min
stftp-1.1.0	1559	NA	Stack overflow	3651	0.39	29	6	15	✓	4 min
nasm-0.98.38	33553	2004-1287	Stack overflow	4064	0.58	3	2	4	✓	44 sec
libpng-1.2.5	33681	2004-0597	Stack overflow	6026	0.35	6	2	86	✓	5 min
putty-0.66	90165	2016-2563	Stack overflow	7338	0.45	4	2	21	✓	30 min
Unalz-0.52	8546	2005-3862	Stack overflow	10905	0.40	14	10	7	✓	30 sec
LaTeX2rtf-1.9	14473	2004-2167	Stack overflow	17056	0.37	11	5	122	✓	8 min
aireplay-ng-1.2b3	62656	2014-8322	Stack overflow	18569	0.59	2	2	223	✗	7 min
corehttp-0.5.3a	914	2007-4060	Stack overflow	25385	0.32	19	6	0	✓	52 min
gas-2.12	595504	2005-4807	Stack overflow	25713	4.17	3	2	346	✓	40 min
abc2mtex-1.6.1	4052	NA	Stack overflow	29521	0.33	12	2	12	✓	1 min
LibSMI-0.4.8	80461	2010-2891	Stack overflow	50787	0.33	46	5	730	✓	4 sec
gif2png-2.5.2	1331	2009-5018	Stack overflow	70854	0.51	49	4	396	✓	46 min
O3read-0.03	932	2004-1288	Stack overflow	78244	0.32	7	2	20	✓	15 min
unrar-3.9.3	17575	NA	Stack overflow	102200	2.43	33	5	1033	✓	6 hour
nullhttp-0.5.0	1849	2002-1496	Heap overflow	141	0.54	3	2	0	✓	1 sec
inetutils-1.8	98941	NA	Heap overflow	28720	0.40	237	7	111	✓	14 min
nginx-1.4.0	100255	2013-2028	Integer overflow	158	0.62	11	4	0	✓	1 sec
Python-2.2	416060	2007-4965	Integer overflow	3426	0.89	31	7	117	✓	3 min
0verkill-0.16	16361	2006-2971	Integer overflow	10494	4.27	1	NA	0	✗	2 sec
openjpeg-2.1.1	169538	2016-7445	Null pointer	67	0.37	10	5	5	✓	1 sec
gdb-7.5.1	1651764	NA	Null pointer	2009	2.94	23	2	79	✓	1sec
podofo-0.9.4	60147	2017-5854	Null pointer	42165	0.65	7	4	80	✓	2 min
Python-2.7	906829	NA	Use-after-free	551	2.14	6	1	0	✓	0.17 sec
poppler-0.8.4	183535	2008-2950	Invalid free	672	1.39	16	4	0	✓	13 sec

Evaluation of POMP (cont.)

Program		*Vulnerability*		*Diagnose Results*						
Name	Size(LoC)	CVE-ID	Type	Trace length	Size of mem (MB)	# of taint	Ground truth	Mem addr unknown	Root cause	Time
coreutils-8.4	138135	2013-0222	Stack overflow	50	56.61	3	2	1	✓	1 sec
coreutils-8.4	138135	2013-0223	Stack overflow	90	[illegible]	2	2	0	✓	1 sec
coreutils-8.4	138135	2013-0221	Stack overflow	92	[illegible]95	3	2	0	✓	1 sec
mcrypt-2.5.8	37439	2012-4409	Stack overflow	315	[illegible]59	3	2	3	✓	3 sec
BinUtils-2.15	697354	2006-2362	Stack overflow	867	[illegible]0.37	16	7	0	✓	1 sec
unrtf-0.19.3	5039	NA	Stack overflow	895	[illegible]54	7	4	10	✓	1 min
psutils-p17	1736	NA	Stack overflow	3123	0.34	7	3	28	✓	4 min
abc2mtex-1.6.1	4053	NA	Stack overflow	26521	0.33	12	2	12	✓	1 min
LibSMI-0.4.8	80461	2010-2891	Stack overflow	50787	0.33	46	5	730	✓	4 sec
gif2png-2.5.2	1331	2009-5018	Stack overflow	70854	0.51	49	4	396	✓	46 min
O3read-0.03	932	2004-1288	Stack overflow	78244	0.32	7	2	20	✓	15 min
unrar-3.9.3	17575	NA	Stack overflow	102200	2.43	33	5	1033	✓	6 hour
nullhttp-0.5.0	1849	2002-1496	Heap overflow	141	0.54	3	2	0	✓	1 sec
inetutils-1.8	98941	NA	Heap overflow	28720	0.40	237	7	111	✓	14 min
nginx-1.4.0	100255	2013-2028	Integer overflow	158	0.62	11	4	0	✓	1 sec
Python-2.2	416060	2007-4965	Integer overflow	3426	0.89	31	7	117	✓	3 min
0verkill-0.16	16361	2006-2971	Integer overflow	10494	4.27	1	NA	0	✗	2 sec
openjpeg-2.1.1	169538	2016-7445	Null pointer	67	0.37	10	5	5	✓	1 sec
gdb-7.5.1	1651764	NA	Null pointer	2009	2.94	23	2	79	✓	1sec
podofo-0.9.4	60147	2017-5854	Null pointer	42165	0.65	7	4	80	✓	2 min
Python-2.7	906829	NA	Use-after-free	551	2.14	6	1	0	✓	0.17 sec
poppler-0.8.4	183535	2008-2950	Invalid free	672	1.39	16	4	0	✓	13 sec

Diagnose Results

Trace length | Size of mem (MB) | # of taint | Ground truth | Mem addr unknown | Root cause | Time

Evaluation of POMP (cont.)

Observation 1:

- POMP marks slightly more instructions than the ones that truly contribute to program crash;

Observation 2:

- Compared with the execution trace (from a fault point to a crashing site), POMP significantly reduces the amount of instructions that software developer and security analysts need to manually examine.

stftp-1.1.0	1559	NA	Stack overflow					

Diagnose Results

Trace length	Size of mem (MB)	# of taint	Ground truth	Mem addr unk	cause	Time
50	56.61	3	2		✓	1 sec
90	59.66	2	2	0	✓	1 sec
92	120.95	3		0	✓	1 sec
315	0.59	3		3	✓	3 sec
867	0.37	16		0	✓	1 sec
895	0.34	7		10	✓	1 min
3123	0.34	7		28	✓	4 min
3651	0.39	29	6	15	✓	4 min
4064	0.58	3	2	4	✓	44 sec
6026	0.35	6	2	86	✓	5 min
7338	0.45	4	2	21	✓	30 min
10905	0.40	14	10	7	✓	30 sec
17056	0.37	11	5	122	✓	8 min
18569	0.59	2	2	223	✗	7 min
25385	0.32	19	6	0	✓	52 min
25713	4.17	3	2	346	✓	40 min
29521	0.33	12	2	12	✓	1 min
50787	0.33	46	5	730	✓	4 sec
70854	0.51	49	4	396	✓	46 min
78244	0.32	7	2	20	✓	15 min
102200	2.43	33	5	1033	✓	6 hour
141	0.54	3	2	0	✓	1 sec
	0.40	237	7	111	✓	14 min
	0.62	11	4	0	✓	1 sec
	0.89	31	7	117	✓	3 min
1	4.27	1	NA	0	✗	2 sec
67	0.37	10	5	5	✓	1 sec
2009	2.94	23	2	79	✓	1sec
42165		7	4	80	✓	2 min
551		6	1	0	✓	0.17 s
672	1.39		4	0	✓	

50	3	2
90	2	2
92	3	2
315	3	2
867	16	7
895	7	4
3123	7	3
3651	29	6
4064	3	2
6026		2
7338		2
10905		10
17056	11	5
18569	2	2
25385	19	6
25713	3	2
29521	12	2
50787	46	5
70854	49	4
78244	7	2
102200	33	5
141	3	2
28720	237	7
	11	4
	31	7
494	1	NA
67	10	5
2009	23	2
42165	7	4
551	6	1
672	16	4

Evaluation of POMP (cont.)

It contains a system call (sys_read) ; system call writes a data chunk to a certain memory region; reverse execution fails to determine its location and size

Observation 3:

- For 2 crashes, POMP fails to include failure root cause into the instruction set identified by POMP.

Integer overflow traps the execution into a long-term iteration; POMP fails to include the root cause instruction into the execution trace restored.

Program		Vulnerability		Diagnose Results						
				ize of em (MB)	# of taint	Ground truth	Mem addr unknown	Root cause	Time
				56.61	3				
				59.66	2				
				120.95	3				
				0.59	3				
				0.37	16				
				0.34	7				
				0.34	7				
stftp-1.1.0	1559	NA	Stack overflow	3651	0.39	29				
nasm-0.98.38	33553	20	Stack overflow	4064	0.58	3				
libpng-1.2.5	33681	200	Stack overflow	6026	0.35	6				
putty-0.66	90165	2016	Stack overflow	7338	0.45	4	2	21	✓	30 min
Unalz-0.52	8546	2005-	Stack overflow	10905	0.40	14	10	7	✓	30 sec
LaTeX2rtf-1.9	14473	2004-2167	Stack overflow	17056	0.37	11	5	122	✓	8 min
aireplay-ng-1.2b3	62656	2014-8322	Stack overflow	18569	0.59	2	2	223	✗	7 min
gas-2.12	595504	2005-4807	Stack overflow	25713	4.17	3	2	346	✓	40 min
abc2mtex-1.6.1	4052	NA	Stack overflow	29521	0.33	12	2	12	✓	1 min
LibSMI-0.4.8	80461	2010-2891	Stack overflow	50787	0.33	46	5	730	✓	4 sec
gif2png-2.5.2	1331	2009-5018	Stack overflow	70854	0.51					
O3read-0.03	932	2004-1288	Stack overflow	78244	0.32					
unrar-3.9.3	17575	NA	Stack overflow	102200	2.43					
nullhttp-0.5.0	1849	2002-1496	Heap overflow	141	0.54					
inetutils-1.8	98941	NA	Heap overflow	28720	0.40					
nginx-1.4.0	100255	2013-2028	Integer overflow	158	0					
Python-2.2	416060	2007-4965	Integer overflow	3426		31	7	117	✓	3 min
0verkill-0.16	16361	2006-2971	Integer overflow	10494		1	NA	0	✗	2 sec
openjpeg-2.1.1	169538	2016-7445	Null pointer	67	0.37	10	5	5	✓	1 sec
gdb-7.5.1	1651764	NA	Null pointer	2009	2.94	23	2	79	✓	1sec
0verkill-0.16	16361	2006-2971	Integer overflow	10494	4.27	1	NA	0	✗	2 sec
poppler-0.8.4	183535	2008-2950	Invalid free	672	1.39	16	4	0	✓	13 sec

Evaluation of POMP (cont.)

Program		*Vulnerability*		*Diagnose Results*						
Name	Size(LoC)	CVE-ID	Type	Trace length	Size of mem (MB)	# of taint	Ground truth	Mem addr unknown	Root cause	Time
coreutils-8.4	138135	2013-0222	Stack overflow	50	56.61	3	2	1	✓	1 sec
						2	2	0	✓	1 sec
						3	2	0	✓	1 sec
						3	2	3	✓	3 sec
						16	7	0	✓	1 sec
						7	4	10	✓	1 min
						7	3	28	✓	4 min
						9	6	15	✓	4 min
						3	2	4	✓	44 sec
						6	2	86	✓	5 min
						4	2	21	✓	30 min
						4	10	7	✓	30 sec
						1	5	122	✓	8 min
						2	2	223	✗	7 min
						9	6	0	✓	52 min
gas-2.12	595504	2005-4807	Stack overflow	25713	4.17	3	2	346	✓	40 min
abc2mtex-1.6.1	4052	NA	Stack overflow	29521	0.33	12	2	12	✓	1 min
LibSMI-0.4.8	80461	2010-2891	Stack overflow	50787	0.33	46	5	730	✓	4 sec
gif2png-2.5.2	1331	2009-5018	Stack overflow	70854	0.51	49	4	396	✓	46 min
O3read-0.03	932	2004-1288	Stack overflow	78244	0.32	7	2	20	✓	15 min
unrar-3.9.3	17575	NA	Stack overflow	102200	2.43	33	5	1033	✓	6 hour
inetutils-1.8	98941	NA	Heap overflow	28720	0.40	237	7	111	✓	14 min
nginx-1.4.0	100255	2013-2028	Integer overflow	158	0.62	11	4	0	✓	1 sec
Python-2.2	416060	2007-4965	Integer overflow	3426	0.89	31	7	117	✓	3 min
0verkill-0.16	16361	2006-2971	Integer overflow	10494	4.27	1	NA	0	✗	2 sec
openjpeg-2.1.1	169538	2016-7445	Null pointer	67	0.37	10	5	5	✓	1 sec
gdb-7.5.1	1651764	NA	Null pointer	2009	2.94	23	2	79	✓	1sec
podofo-0.9.4	60147	2017-5854	Null pointer	42165	0.65	7	4	80	✓	2 min
Python-2.7	906829	NA	Use-after-free	551	2.14	6	1	0	✓	0.17 sec
poppler-0.8.4	183535	2008-2950	Invalid free	672	1.39	16	4	0	✓	13 sec

Observation 4:

- For most test cases, POMP takes seconds or tens of minutes for program failure diagnosis; for some other test cases, it could take several hours to process.

Summary

- POMP can reversely execute a crashing program and restore the memory footprints
- The memory footprints restored can be used for data flow construction and program failure diagnosis
- POMP reduces the code space that software developers and security analysts need to manually examine, which significantly facilitates program diagnosis failure
- POMP can handle program crashes including those resulting from memory corruption vulnerabilities

Thank you very much!

POMP source code is available at https://github.com/junxzm1990/pomp.git