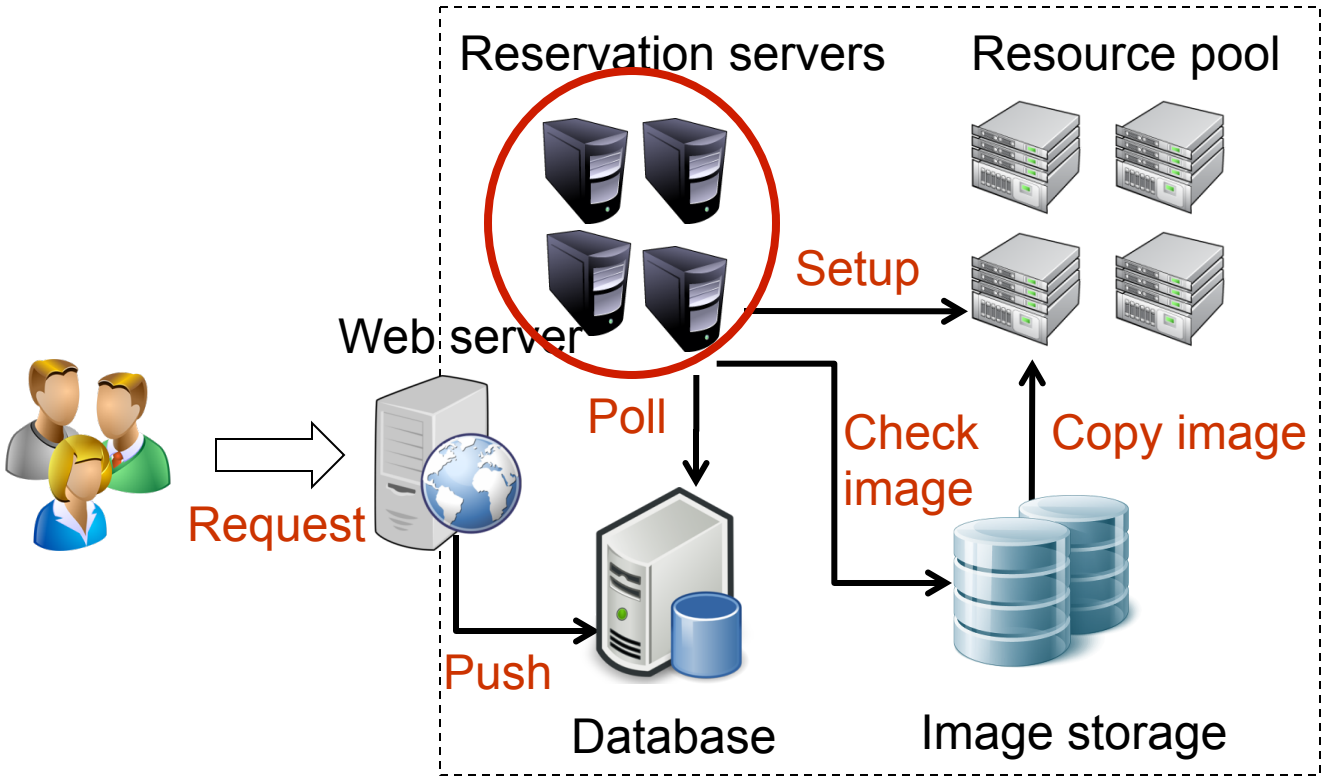
NC STATE UNIVERSITY Computer Science

Insight: In-situ Online Service Failure Path Inference in Production Computing Infrastructures

Hiep Nguyen,
Daniel J. Dean, Kamal Kc, Xiaohui Gu
North Carolina State University

Online Services



Virtual Computing Lab (VCL)

Online Service Non-crashing Failures

- Online services are prone to non-crashing failures
  - Number of non-crashing failures in VCL: 154 in one month (April 2014); 1813 in one year (2013)
  - Non-crashing failures often go unnoticed
  - Error message does not tell why failures occur

lighttpd failure:

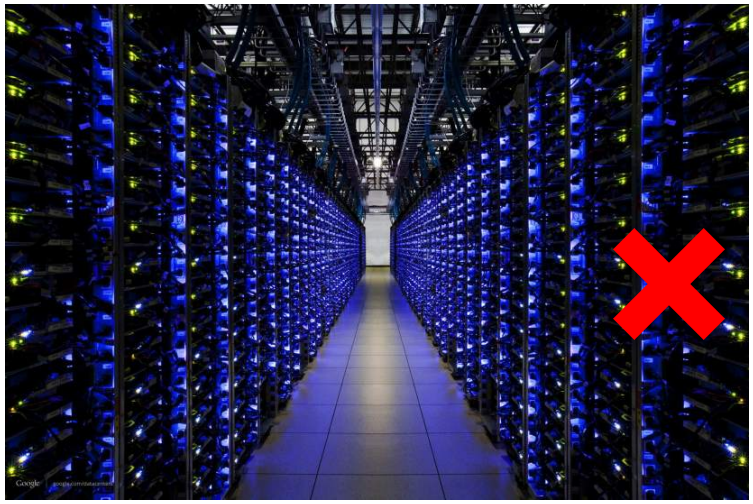
```
2014-01-25 21:23:28: (mod_proxy.c.1249) no proxy-handler found for: /
```

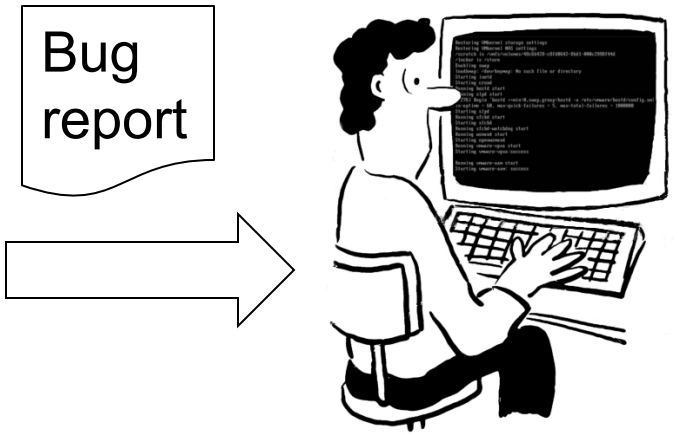
Authentication failure in VCL:

```
|2836|66:58|new| ---- WARNING ----
|2836|66:58|new| 2013-05-04 22:07:57|2836|66:58|new|
new.pm:process(295)|failed to load vmsk1 with kvmlinux-v0
…
|2836|66:58|new| ---- CRITICAL ----
|2836|66:58|new| 2013-05-04 22:07:57|2836|66:58|new|
State.pm:reservation_failed(213)| reservation failed on vmsk1:
process failed after trying to load or make available
```

Approach 1: Offline Failure Reproduction

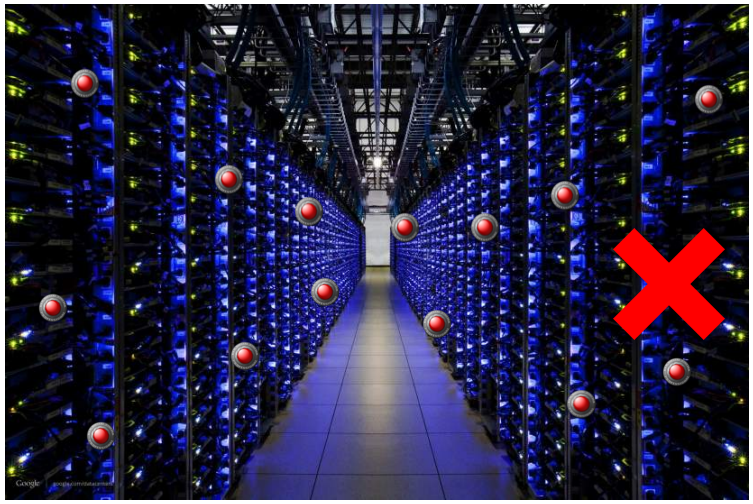
- Offline failure reproduction is hard
  - Lacking environment information
  - Missing interacting components
  - Absent third-party libraries

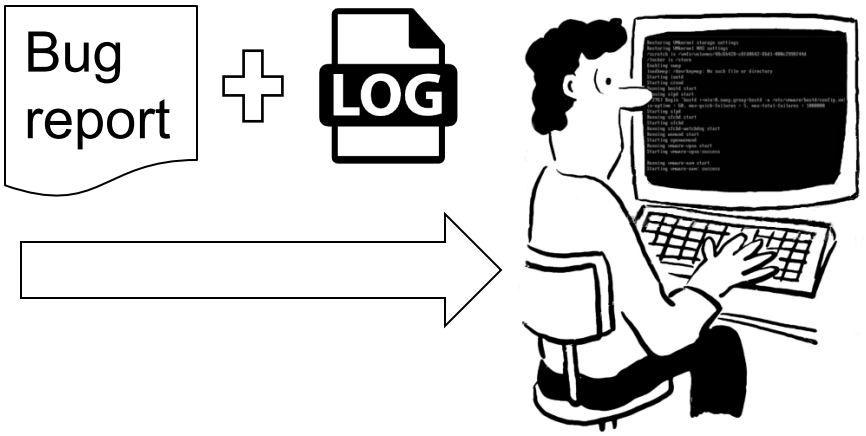




Approach 2: Record and Replay

- Intrusive system recording
  - High overhead
  - Privacy concerns
  - Deployment challenges





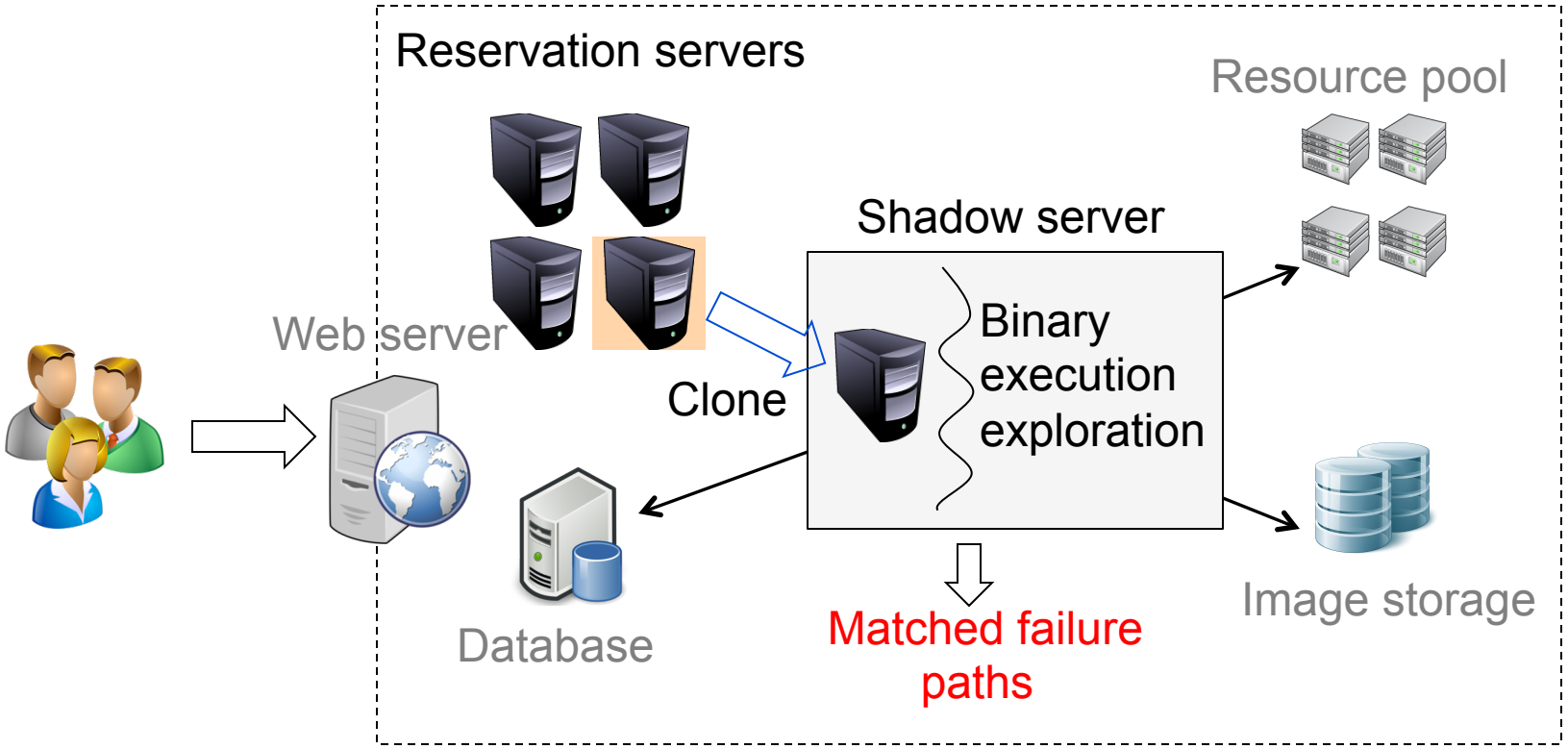
Approach 3: Onsite Failure Diagnosis

- Triage: diagnosing production run failures at the user's site [Tucek et al. SOSP' 07]
  - Relies on repeated replays to compare good and bad runs
  - Requires runtime checkpointing for replays
  - Performs diagnosis directly on the production server
  - Can incur long service downtime
  - Does not fully leverage runtime environment data

Our Key Observations

- Production environments provide lots of clues
  - Environment data: inputs, configuration files, interacting components
  - Runtime outputs: console logs, system call traces
- Onsite failure path search is more efficient
  - Significantly smaller search scope
  - Does not require intrusive recording for replay
- Decouple failure analysis from the production service execution
  - Capture runtime state using dynamic virtual machine cloning
  - Minimize production service downtime

Our Approach



Virtual Computing Lab (VCL)

Challenges

- No source code access
  - Binary-based approach
- Fast failure path inference
  - Leverage “fresh” environment data at the failure moment
- Low overhead
  - No intrusive recording
- Different programming languages
  - Compiled programs (e.g., C/C++)
  - Interpreted programs (e.g., Perl, Java)

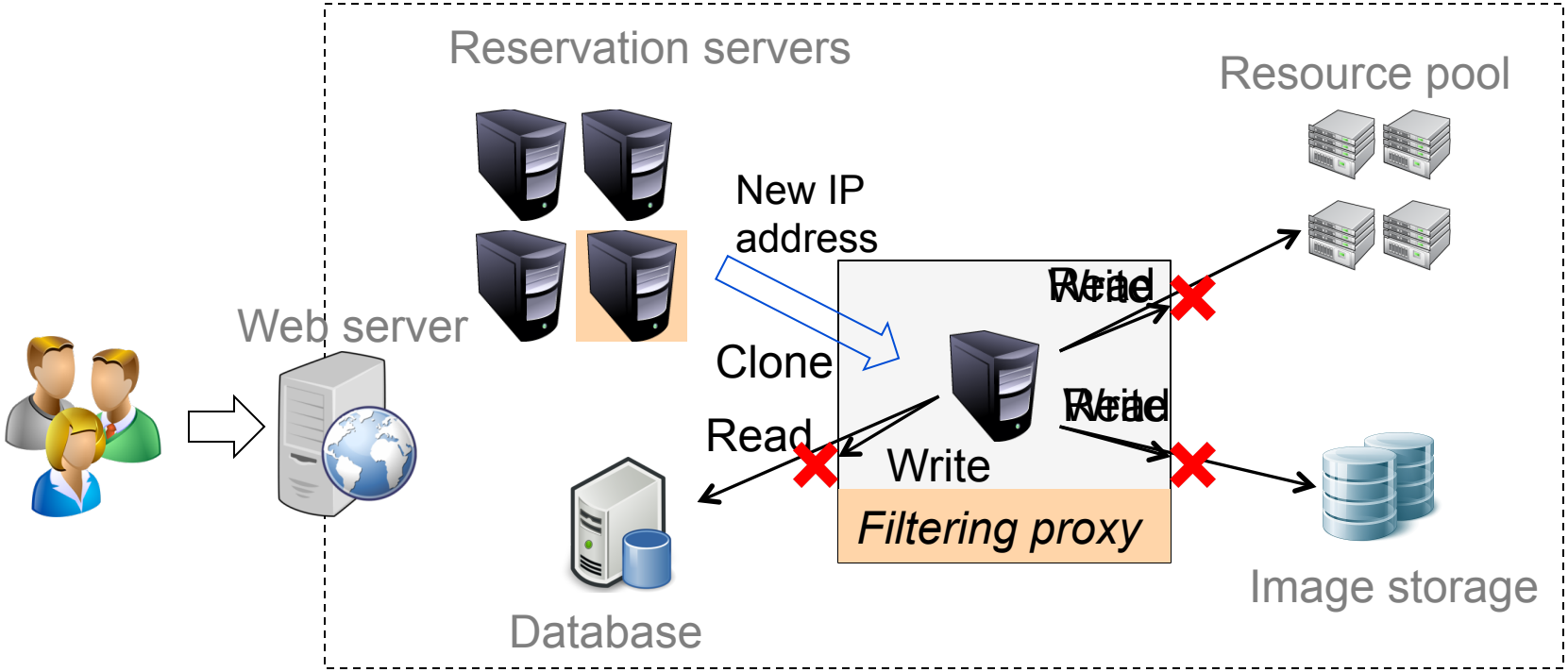
Building Blocks

- *Dynamic* shadow server creation
  - Use live VM cloning
  - Decouple analysis from the production run

- *Guided* binary execution exploration
  - Leverage the production environment data and runtime outputs as guidance to search the failure paths
  - No source code required

Dynamic Shadow Server Creation

- Automatic reconfiguration
  - Reset the IP address for the shadow server
  - Reconfigure firewall
- Policy-driven output filtering proxy



Guided Binary Execution Exploration

Console log:

```
1. Checking request
state in database

2. Start processing
reservation
```

False

False

```
1   log("Checking request state in database");
2   my @selected_rows = database_select($select_statement);
3   if ((scalar @selected_rows) == 0) {
4       log("0 rows returned from request state select
            statement, request was probably deleted,
            returning 0");
5       return 0;
6   }else{
7       if ((scalar @selected_rows) > 1) {
8           log("More than 1 row returned from request state
                select statement, returning 0");
9           return 0;
10      }else{
11          log("Start processing reservation");
12      }
13  }
```

Unmatched

Unmatched

Matched

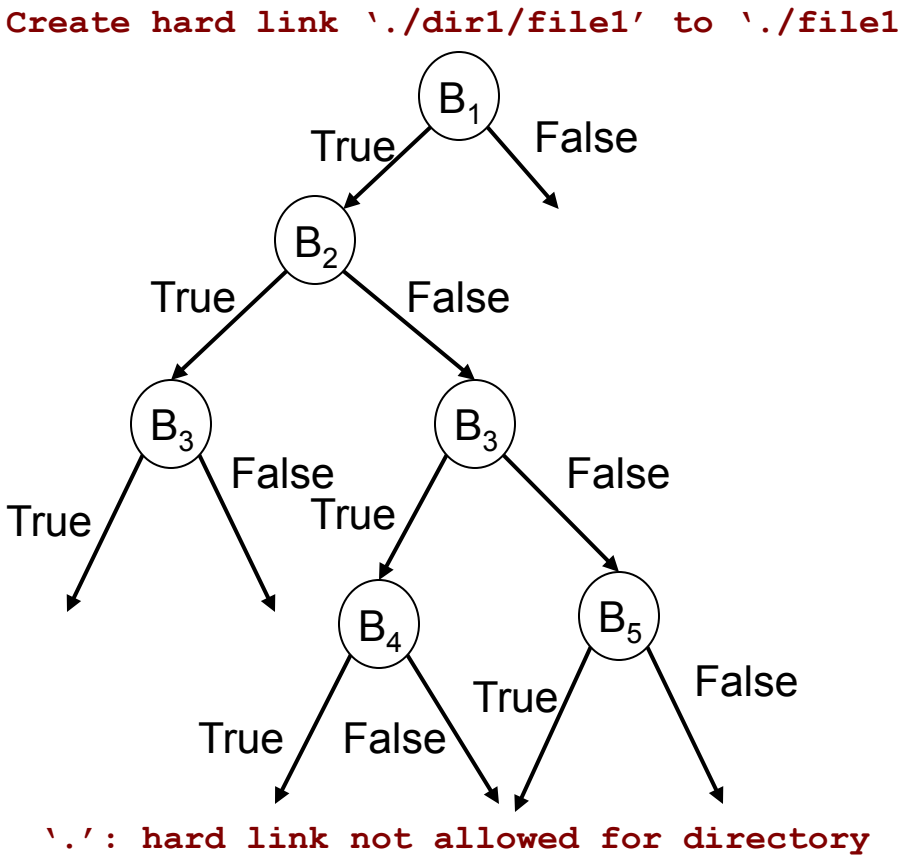
Problem with Sparse Console Log

- Too many possible paths

Console log:

```
1.  Create hard link `./dir1/
    file1' to `./file1'

2.  `.': hard link not allowed
    for directory
```



Leveraging System Call Sequences

Console log:

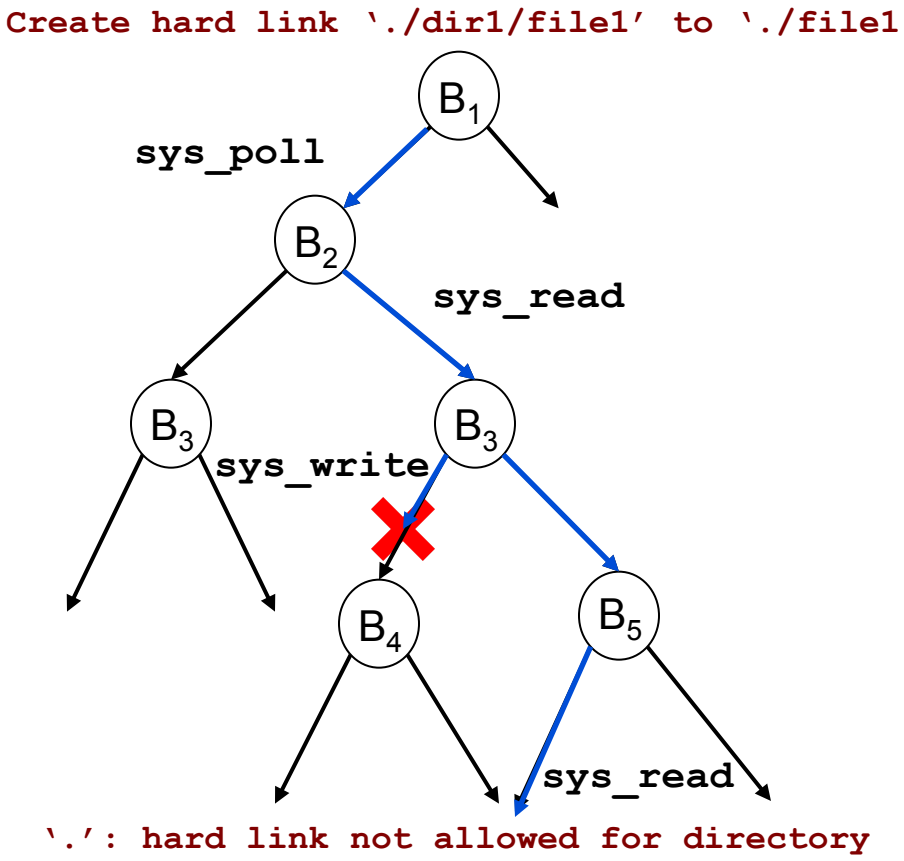
```
1. Create hard link './dir1/
   file1' to './file1'

   sys_poll

   sys_read

   sys_read

2. '.': hard link not allowed
   for directory
```



Implementation

- Currently support Perl and C/C++ programs
  - Modified Perl interpreter for Perl programs
    - Add handling for branch opcodes (e.g., `OP_COND_EXPR`)
    - Intercept interpreter's execution stack to change branch conditions
  - Pin tool for C/C++ programs
    - Intercept branch statements (e.g., `JZ, JNE, JE`)
    - Modify the branch conditions by changing EFLAGS register
  - Uses SystemTap for monitoring system calls

Tested Failure Cases

	System	LOC	Failure path length (Num. of funtions)	Failure name	Num. of console log messages
Production failures	VCL	145K	112	Overlapping reservation failure	132
			299	Network failure	290
			298	Authentication failure	409
			147	Image corruption failure	178
Reported open source software failures	Apache httpd	176K	176	Authentication failure	1
			164	CGI failure	1
	Squid	110K	588	Non-crashing stop failure	195
	Lighttpd	38K	730	Proxy failure	3
	PBZIP2	3.9K	41	Decompression failure	14
	aget	1.5K	2	Download failure	1
	rmdir	0.2K	2	Option failure	2
	ln	0.6K	1	Option failure	1
	touch	0.5K	1	Time failure	1

Call Path Difference in VCL Failures

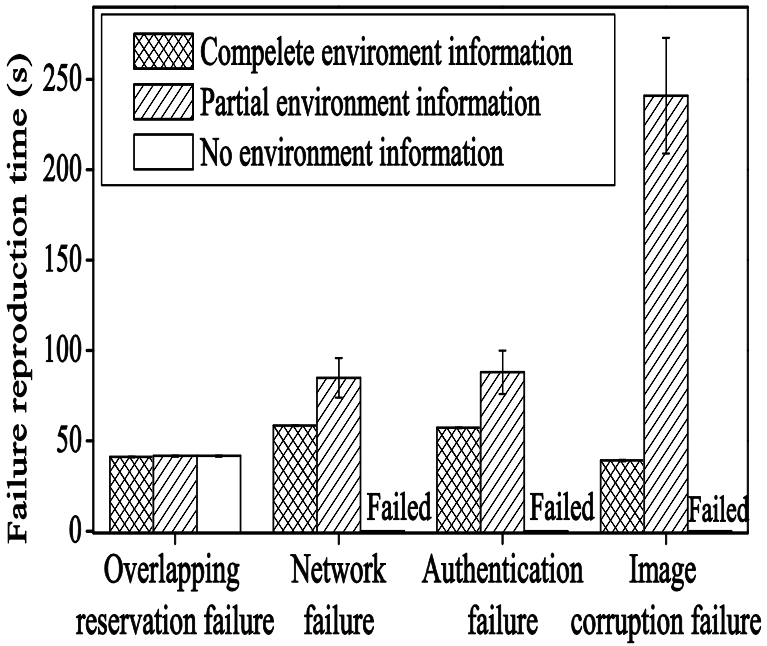
Failure name	Complete environment data	Partial environment data	No environment data
Overlapping reservation failure	0	0	0
Network failure	0	0	Failed
Authentication failure	0	0	Failed
Image corruption failure	0	0	Failed

Call Path Difference in Open Source Software Failures

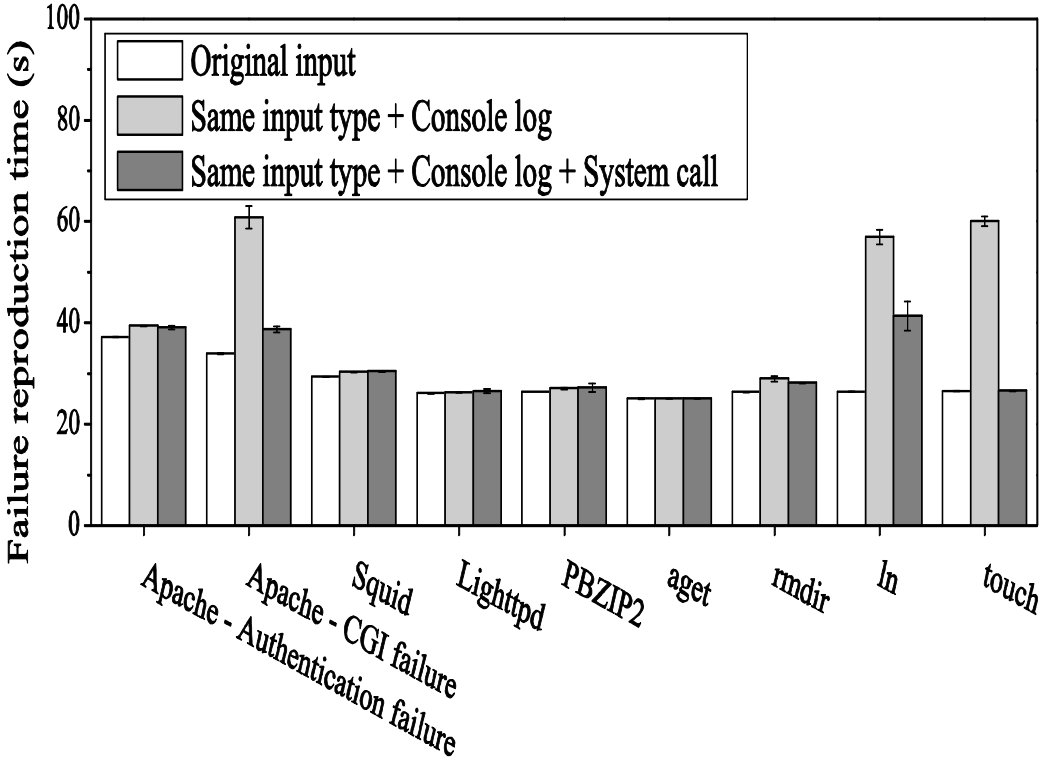
- Reproduced failure paths always cover root cause functions and branches

Failure name	Original input	Same input type + console log	Same input type + console log + system call
Apache (authentication failure)	0	17	11
Apache (CGI failure)	0	140	9
Squid (non-crashing stop failure)	0	0	0
Lighttpd (proxy failure)	0	0	0
PBZIP2 (decompression failure)	0	1	0
aget (download failure)	0	0	0
rmdir (option failure)	0	0	0
ln (option failure)	0	0	0
touch (time failure)	0	0	0

Failure Reproduction Time



VCL failures



Open source software failures

Overhead

Performance impact

Shadow system	< 0.3%
System call tracing	< 1.5%
Shadow creation time	< 30s
Stop-and-copy time	< 70ms

Logging overhead (1 day)

System	**Input log**	**Console log**	**System call log**
VCL	130MB	490MB	N/A
Apache	20MB	0.3MB	12MB

Conclusion

- Insight: In-situ failure path inference system
  - Enable failure path inference *inside* the production environment
  - Use *shadow component* to decouple failure analysis from the production run
  - *Guided* binary execution exploration to find high fidelity failure paths quickly without source code

Thank you!

Acknowledgement

- This work was sponsored in part by NSF CNS0915567 grant, NSF CNS0915861 grant, U.S. Army Research Office (ARO) under grant W911NF-10-1-0273, and Google Research Awards