Deterministic Record-and-Replay on top of the Linux Kernel

Taming Syscalls, Signals, and Scheduling for Automatic Failure Diagnosis

Tongping Liu · Teyon (teyon.ai) · ex-UMass tenured professor

The bug that vanishes when you look at it.

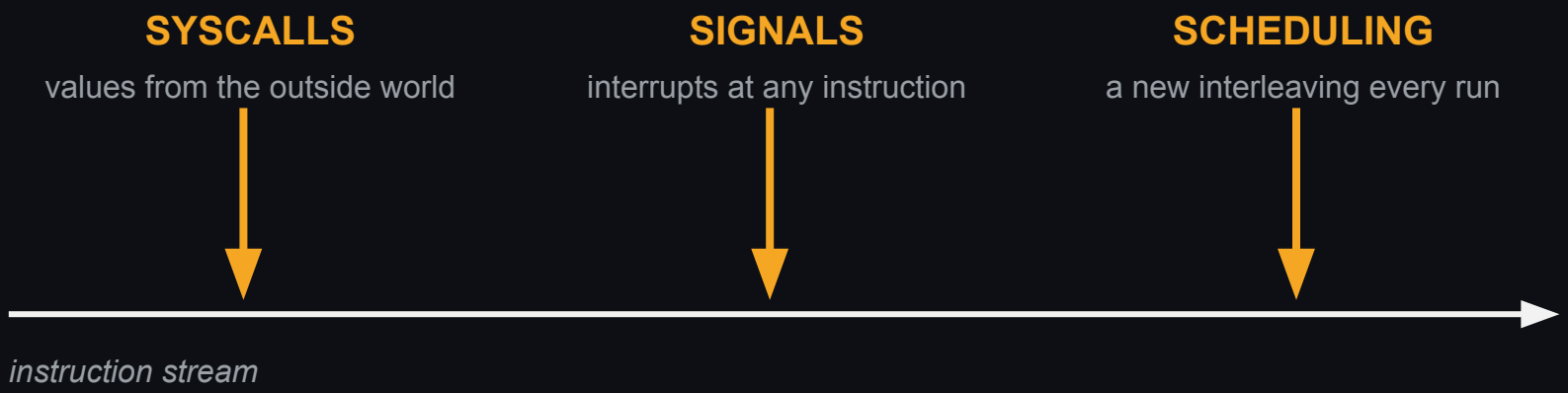
```
$ python train.py
...
step 18232  loss 2.417
Segmentation fault (core dumped)
```

?

```
$ python train.py
...
step 18232  loss 2.417
step 18233  loss 2.415   ✓
```

Run it again. It passes.

Determinism is stolen by three demons.



Same program. Same input. Different execution.

Record the boundary, not the world.

<3%

recording overhead

(rr: 17× slower)

Replay recomputes the rest.

It sat in papers for a decade.

Then AI made it necessary.

2011 – 2020

SOSP · PLDI · OOPSLA · ICSE

solved in papers

2026

🔥 **10,000-GPU clusters**

needed in production

Validated Moat of Watcher: Built on Three Pillars

10+ years (2010~2020) of Grand Slam of System Research

SOSP
Symposium on Operating Systems Principles
Systems · #1

PLDI
Programming Language Design & Implementation
Prog Language · #1

ICSE
International Conference on Software Engineering
Software Eng. · #1

Pillar 1 **Low-Overhead Deterministic Replay** — *SOSP'11 (Dthreads) · PLDI'18 (iReplayer)*

The core foundation for <5% overhead, ready for in-production use.

Pillar 2 **Autonomous Failure Diagnosis** — *OOPSLA'20 (Watcher)*

Traces failures back to their exact root causes in minutes.

Pillar 3 **Evidence-Based Failure Classification** — *ICSE'16 (DoubleTake)*

Automatically classifies failures into detailed types.

Top peer-reviewed publications · US Patent Granted · **Decade of compounding research advantage.**

Watcher: In-Situ Failure Diagnosis

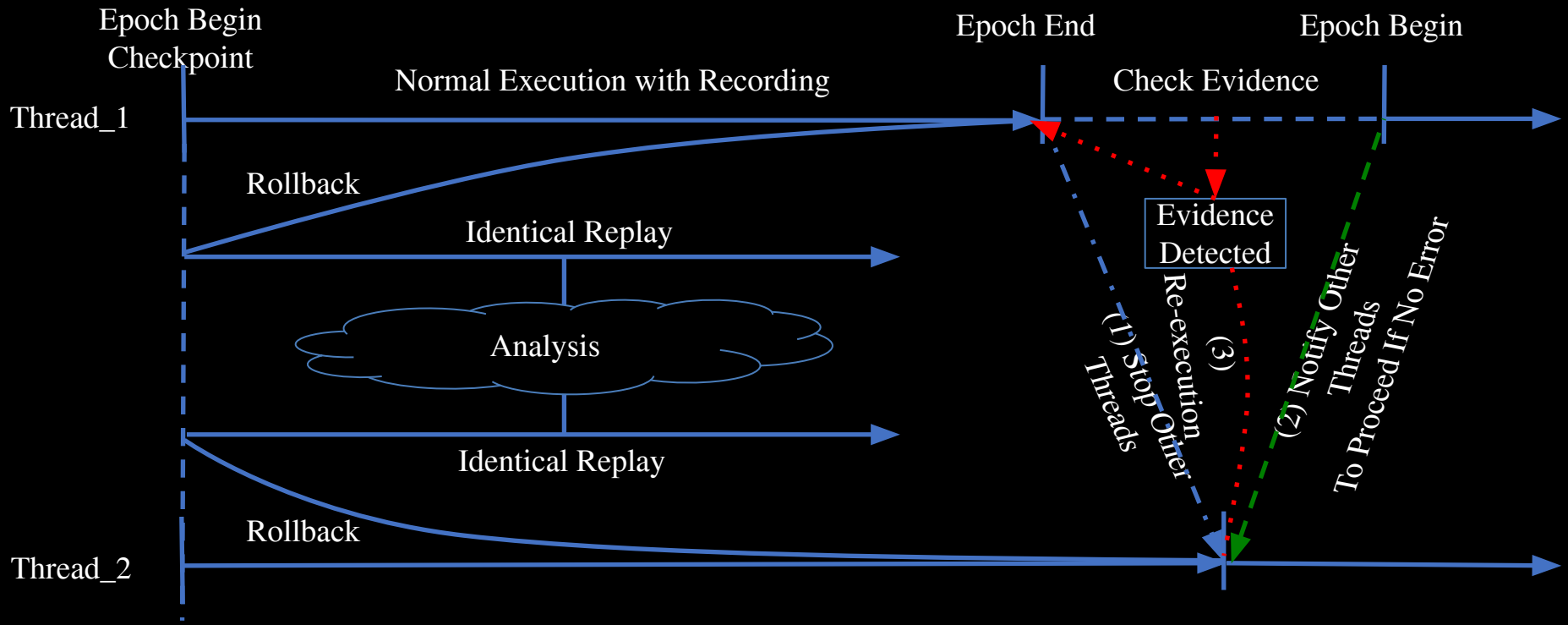
Watcher diagnoses a real crash within minutes

```
root@7862e09bfed1:/home/Watcher/effectiveness/aubio# ./build/bin/aubioquiet -i ./id000007,sig08,src000068,opext_AO,pos48
Floating point exception
root@7862e09bfed1:/home/Watcher/effectiveness/aubio# LD_PRELOAD=/home/Watcher/Watcher/libwatcher.so ./build/bin/aubioquiet -
i ./id000007,sig08,src000068,opext_AO,pos48
Incident: Faulty Signal 8 (Floating point exception) received at thread-0: IP address-0x7fffb71adadf, Address-0x7fffb71adadf
************************************************************
Thread 16284 fails with a read/write at 0x7fffffffe304:
#0 [0x7fffb71adadf]: File: /home/Watcher/effectiveness/aubio/aubio-0.4.6/build/../src/io/source_wavread.c, Function: new_aub
io_source_wavread, Line: 256, Code:"duration = read_little_endian(buf, 4) / blockalign;"



------------ Level 1: Thread 0 writes 0 at address 0x7fffb71ad804 with callstack: ------------
File: /home/Watcher/effectiveness/aubio/aubio-0.4.6/build/../src/io/source_wavread.c, Function: new_aubio_source_wavread, Li
ne: 187, Code:"blockalign = read_little_endian(buf, 2);"


------------ Level 2: Thread 0 writes 0 at address 0x7fffb71ad1ec with callstack: ------------
File: /home/Watcher/effectiveness/aubio/aubio-0.4.6/build/../src/io/source_wavread.c, Function: read_little_endian, Line: 70
, Code:"ret += buf[i] << (i * 8);"


------------ Level 3: Thread 0 writes 0 at address 0x7fffb6f3fa3e with callstack: ------------
File: /home/Watcher/effectiveness/aubio/aubio-0.4.6/build/../src/io/source_wavread.c, Function: new_aubio_source_wavread, Li
ne: 186, Code:"bytes_read += fread(buf, 1, 2, s->fid);"
```

Click to watch 1 min demo

iReplayer: *Checkpoint + Record + Replay*

Epoch-based recording and deterministic replay for multi-threaded applications



Checkpoint: The Epoch Baseline

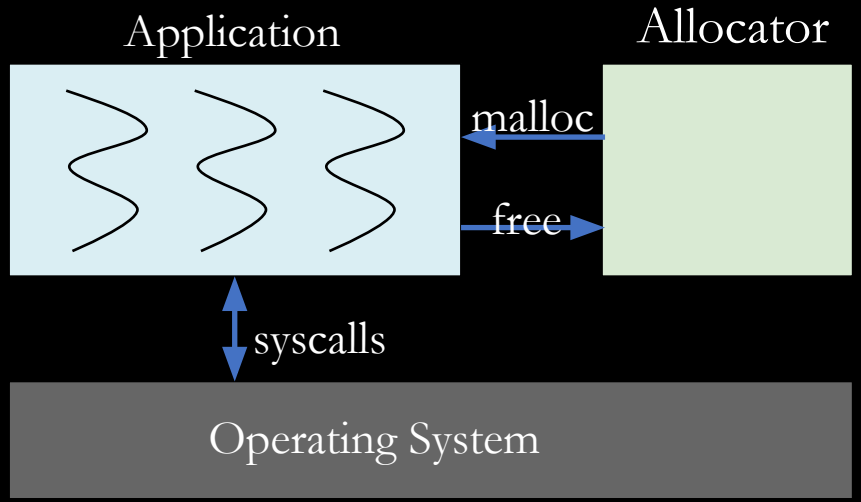
High-fidelity capture of the absolute memory state

- The Goal:
  - Establish the starting point of ground Truth

- The Scope:
  - Each thread heap
  - Each thread stack
  - Globals (each library has its own .data and .bss)

Record: The Deterministic Boundary

Precise logging of explicit non-deterministic events



- External Events
  - Memory allocations
  - Syscalls
- Internal Events
  - Synchronizations, e.g. lock

```
Thread1:                Thread2:

Lock(&lock1);           Lock(&lock1);
x = 1;                  x = 2;
Unlock(&lock1);         Unlock(&lock1);
```

Challenge-1: efficient recording

Challenge-2: implicit non-determinism (e.g., race)

Replay: The Bit-Exact Reconstruction

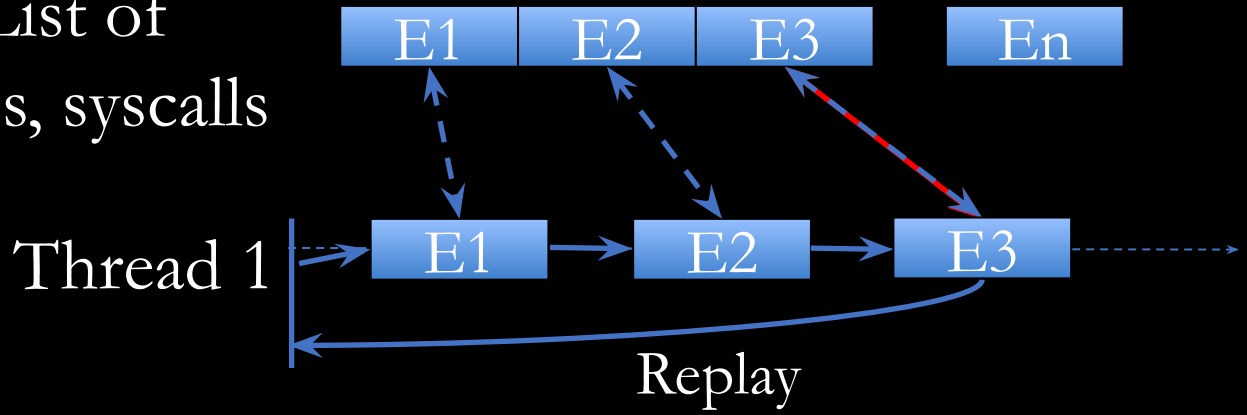
Surgical resolution of state asymmetry and race conditions

- Memory State Recovery (Stack Challenge*)
  - Current stack is larger ⇒ cleaning
  - Current stack is smaller ⇒ temporary stack for recovery
- Deterministic Event Injection
  - Syscall
  - Explicit Synchronizations
  - I/O Emulation
- Race Condition Resolution

Race Resolution: The Offline Search

Synthesize the determined schedule via multiple replays

List of syncs, syscalls



Upon divergence, iReplayer keeps replaying until an identical schedule is found!

Validation: The Identical Replay

Empirical proof of bit-exact reconstruction

- Identical Replay
  - Same order of sync and syscall events
  - Same heap image
- Reproducing racy executions (e.g. crasher)
  - **99.8718%**: one re-execution
  - 0.1088%: two re-executions
  - 0.0121%: three re-executions
  - 0.0073%: >= four re-executions

Efficiency: The <3% Online Overhead

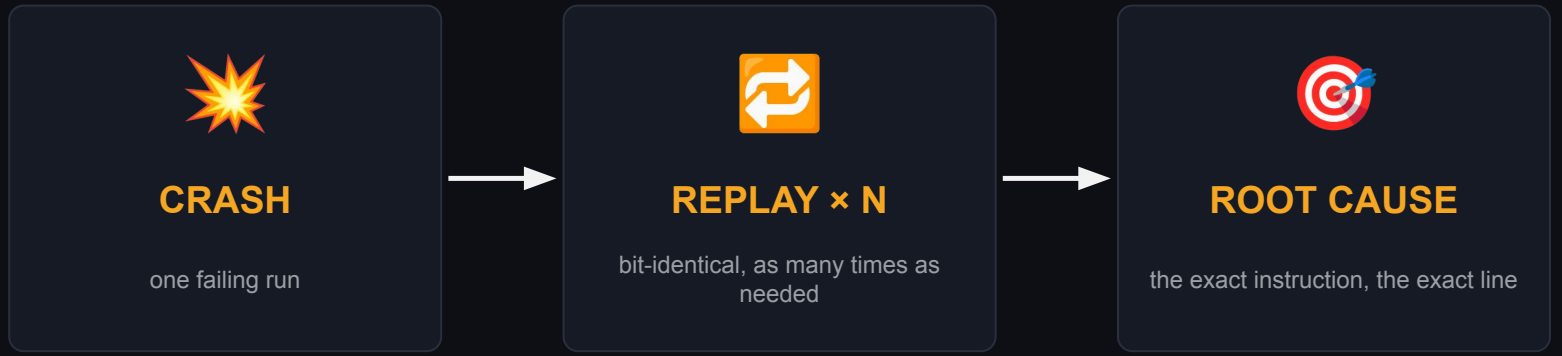
Capturing execution ground truth with imperceptible overhead

Table 3. Performance overhead

Applications	IR-Alloc	iReplayer	CLAP	RR
blackscholes	1.001	1.021	1.113	8.011
bodytrack	0.993	0.990	-	27.283
canneal	0.880	0.962	-	6.884
dedup	0.664	0.817	1.074	5.138
ferret	1.017	0.998	3.519	13.275
fluidanimate	1.044	1.493	2.183	34.082
streamcluster	1.102	1.100	2.383	52.280
swaptions	0.979	0.990	2.964	29.921
x264	0.991	1.032	9.100	16.228
aget	1.000	1.032	1.013	1.065
apache	1.002	1.056	-	-
memcached	1.000	1.001	1.001	1.822
pbzip2	0.974	0.895	-	26.852
pfscan	1.015	1.006	1.032	8.462
sqlite	0.973	1.087	3.853	15.059
average	0.976	1.027	2.658	17.597

- Overall overhead: 2.7%
  - Only two are larger than 10%
  - fluidanimate: 54M syncs per second
- Competitor - rr
  - 17X slower
  - Fake multithreading on single core

If you can replay it, you can diagnose it — automatically.



Without ground truth, a crash is a mystery.

With it, it's just math.

Diagnosis: From Replay to Root Cause

Leverage execution ground truth to compute the causal chain

iReplayer Engine
Execution ground truth (< 3% overhead, 100% identicality)

Watcher Analyzer
Autonomous Fault Analysis

Autonomous Root Cause Analysis

Invert the execution timeline to trace the origin

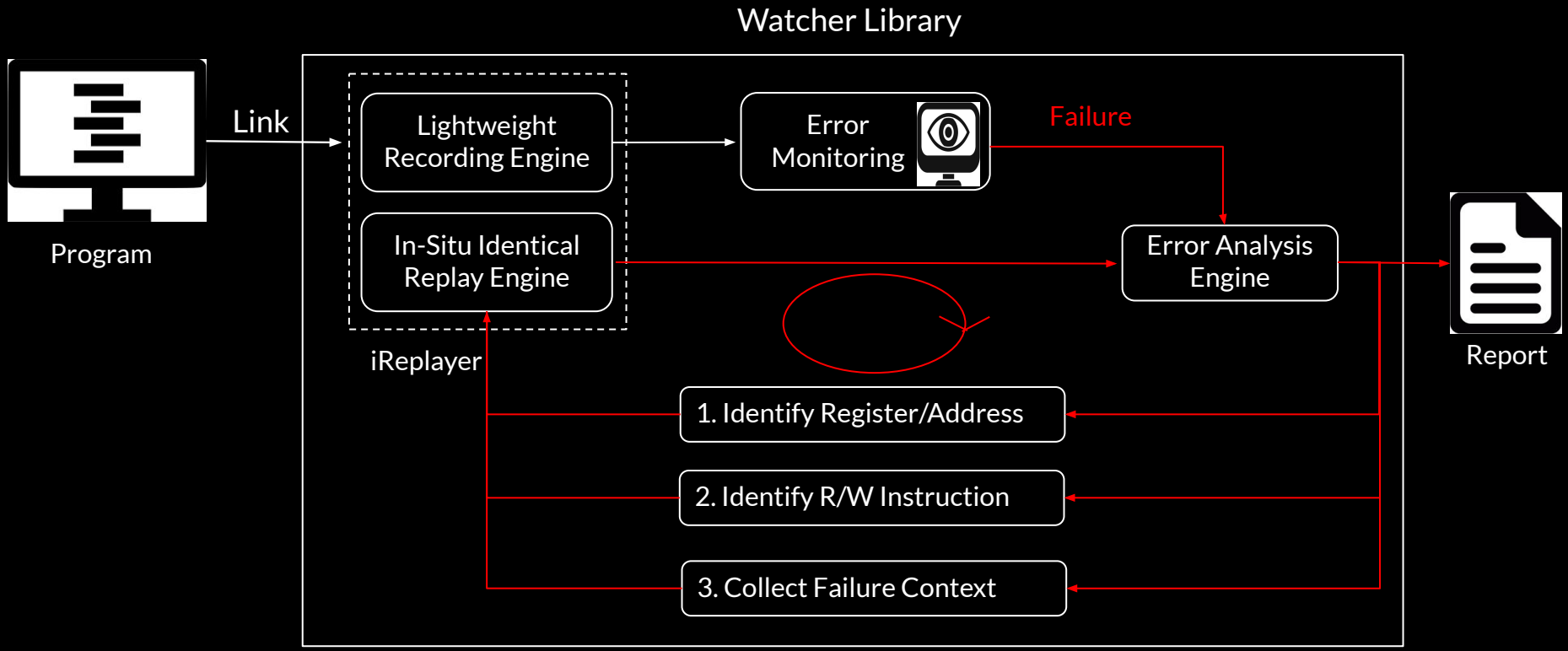
Without causality, a crash is a mystery. With execution ground truth, it’s just math.

The Approach:

Watcher automatically traces the dynamic dataflow backward from the point of failure. It isolates the exact instruction that corrupted the memory state via multiple replays.

Architecture: The Iterative Diagnosis

Extract deterministic dataflow across multiple replays



Mechanics: The Hybrid Trace Architecture

Fuse silicon-level tracing and binary analysis

1. Hardware-Assisted Tracing (e.g., Processor Trace / LBR)
   Value: *Reducing the number of replays*

2. Adaptive Breakpoints & Watchpoints
   Value: *Rapidly isolating causal conditions without complicated analysis*

3. Binary Analysis
   Value: *Mapping hardware traces to pinpoint registers, watchpoint targets, and resolute STLF (Store-to-Load Forwarding)*

Capability: The Universal Resolution

Diagnosing sequential and concurrent root causes in seconds

Application	Reference	Description	Type	Level	Var.	Inst.	Time (s)	Software Breakpoint		Hardware Trace	
								Rerun (#)	Time (s)	Rerun (#)	Time (s)
Aireplay-ng	CVE-2014-8322	Stack overflow	Seq.	2	1	2	0.165	13	1.31	5	3.02
Aubio	CVE-2017-17054	Divide-by-zero	Seq.	3	1	3	0.164	16	3.12	6	3.61
Cppcheck-148	Bugbase [Kasikci et al. 2015]	Null pointer	Seq.	1	1	1	0.14	4	0.86	3	1.41
Cppcheck-152	Bugbase [Kasikci et al. 2015]	Null pointer	Seq.	1	1	1	0.157	5	0.93	3	1.48
Curl-721	Bugbase [Kasikci et al. 2015]	Null pointer	Seq.	1	1	1	0.172	4	0.39	3	0.52
Exiv2	CVE-2018-9303	Assertion	Seq.	3	1	4	0.134	11	1.91	7	5.29
Gas	CVE-2005-4807	Stack overflow	Seq.	3	1	3	0.14	13	2.82	7	4.15
Gif2png	CVE-2009-5018	Stack overflow	Seq.	2	1	2	0.141	10	1.49	5	2.13
HTTrack	Issue #20247	Null pointer	Con.	1	1	1	2.167	3	6.26	3	6.56
Libming	CVE-2018-7875	Null pointer	Seq.	1	1	1	0.161	3	0.26	3	0.5
Libpng	CVE-2004-0597	Stack overflow	Seq.	2	1	2	0.134	4	0.87	5	2.89
Libtiff	CVE-2017-7595	Divide-by-zero	Seq.	3	1	3	0.158	18	1.91	7	2.72
Memcached	CVE-2011-4971	Invalid address	Seq.	4	3	11	0.22	62	4.75	21	11.72
Nasm	CVE-2004-1287	Stack overflow	Seq.	3	1	3	0.157	13	2.77	7	3.99
Openjpeg	CVE-2016-7445	Null pointer	Seq.	1	1	1	0.139	8	2.1	3	1.37
Pbzip2	Bugbase [Kasikci et al. 2015]	Null pointer	Con.	1	1	1	0.314	4	0.73	3	1.06
Pfscan	Symbiosis [Machado et al. 2015b]	Assertion	Con.	1	1	1	0.163	9	1.49	3	2.21
Polymorph	Bugbench [Lu et al. 2005]	Stack overflow	Seq.	2	1	2	0.159	8	1.07	5	1.95
Sam2p	Issue #33	Divide-by-zero	Seq.	2	2	3	1.219	10	8.32	7	9.82
Sqlite	Ticket #cfa2c908f2	Assertion fails	Seq.	2	1	2	0.139	8	2.07	5	3.54
Stringbuffer	Symbiosis [Machado et al. 2015b]	Abort	Con.	3	1	3	0.17	23	2.23	7	3.15
Tcpdump	CVE-2017-5205	Invalid address	Seq.	1	1	1	0.14	10	1.26	3	2.38
Transmission	Issue #1818	Assertion	Con.	1	1	1	0.786	5	3.73	3	4.2
Unrar	3LRVS [ZadYree 2011]	Stack overflow	Seq.	2	1	2	0.165	7	1.21	5	2.19
Average						2.3	0.32	11.29	2.24	5.38	3.41

Q1: Tell me about the last time a crash or an OOM ruined your days.

Q2: Where are the gaps in your current tooling? What do you wish existed?

Q3: Who's willing to send me their scenario? I'll run it and show you what we find.

TEYON: Built to run **forever**

Recovers from every fault. Explains every byte.

I left tenure to build this. ***Come build it with me.***

Design partners

Teams fighting crashes and OOMs at scale.

Founding engineer

Systems · C++/CUDA · Thrives on Hard Problems.

teyon.ai · tongping@teyon.ai · linkedin.com/in/tongping