Finding bugs in HPC Systems with the perf_fuzzer

Vince Weaver

vincent.weaver@maine.edu

University of Maine

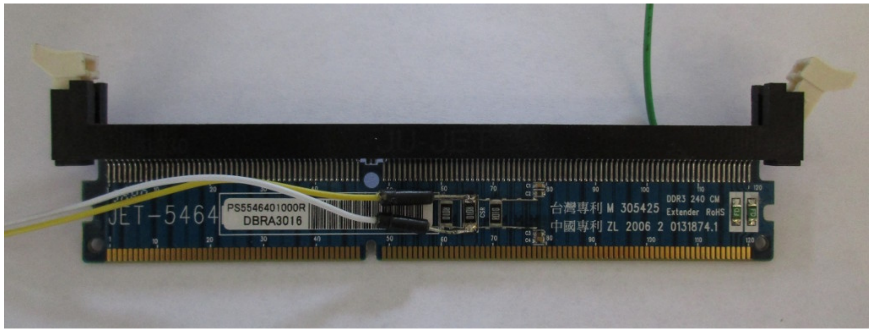
ICL Lunch Talk — 27 May 2022

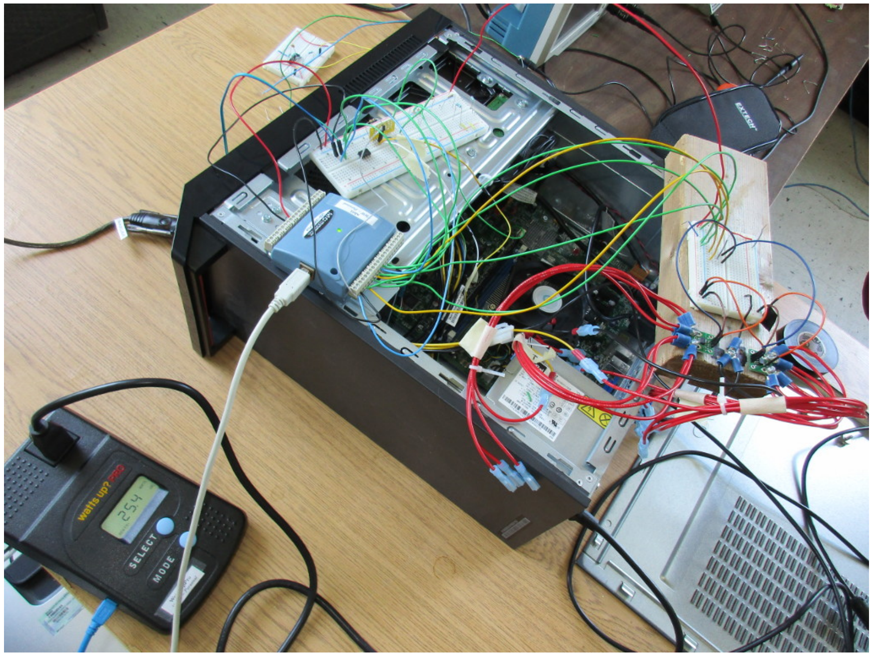
10 Years since my last ICL Lunch Talk

Which was partially presented on an old Apple II computer

What Have I Been Up To Since Then?

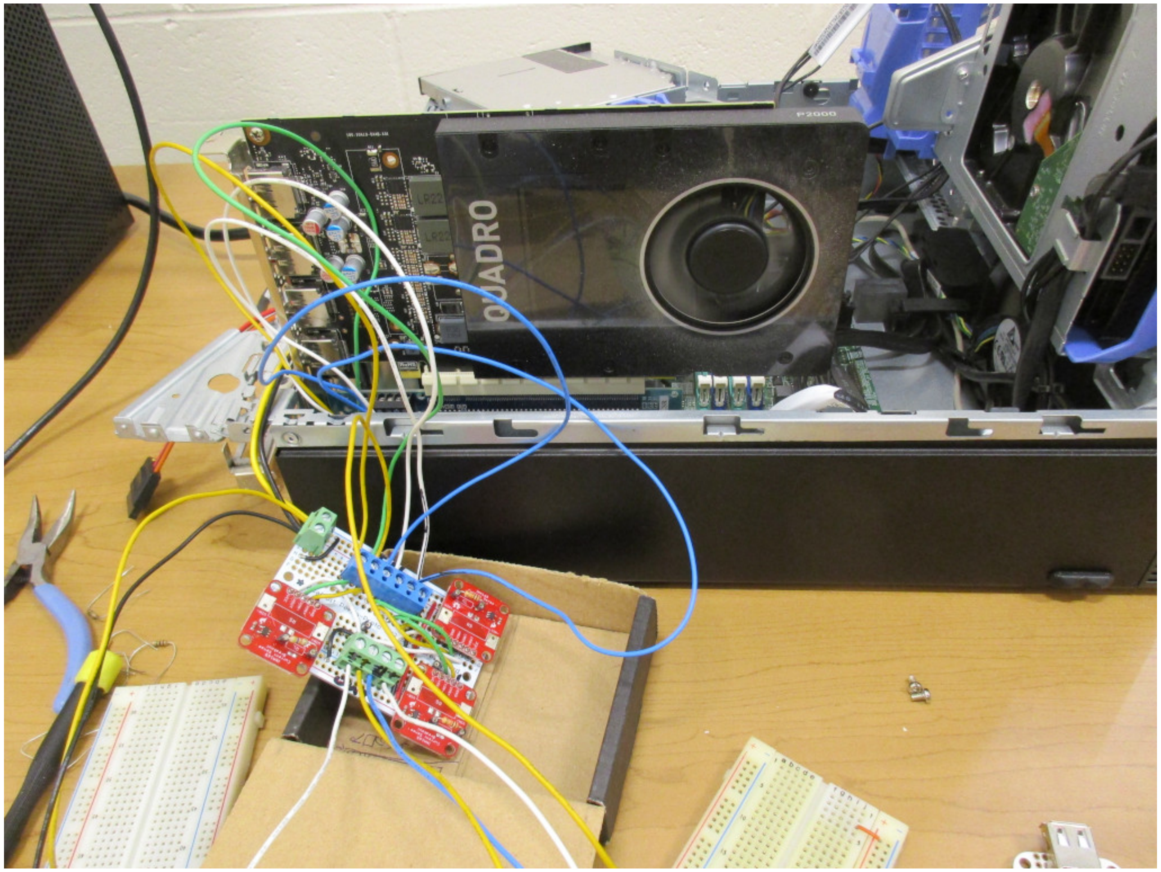
Validating DRAM RAPL Power Measurements





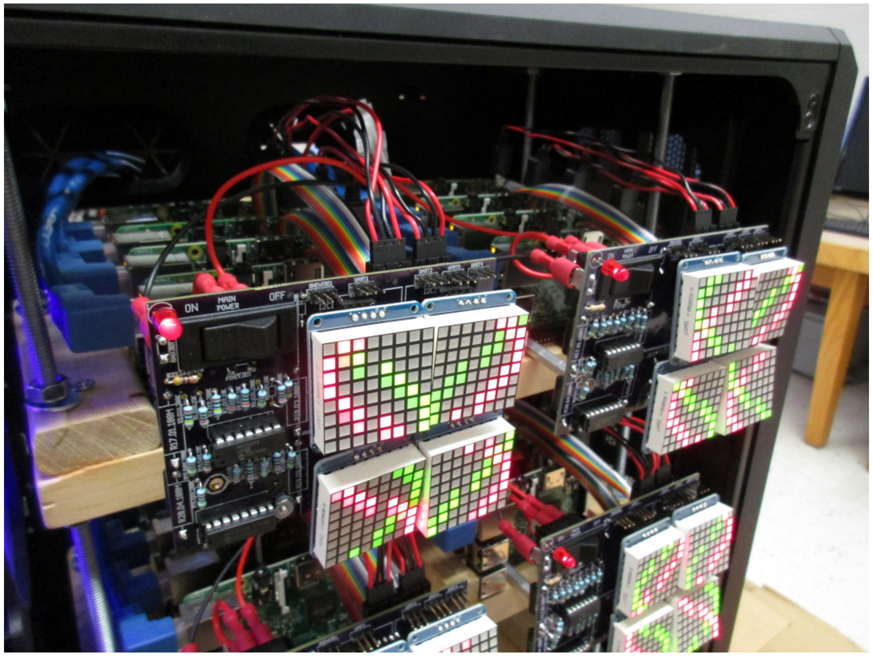
https://web.eece.maine.edu/~vweaver/projects/rapl/rapl_validation.html

Validating GPU Power Measurements



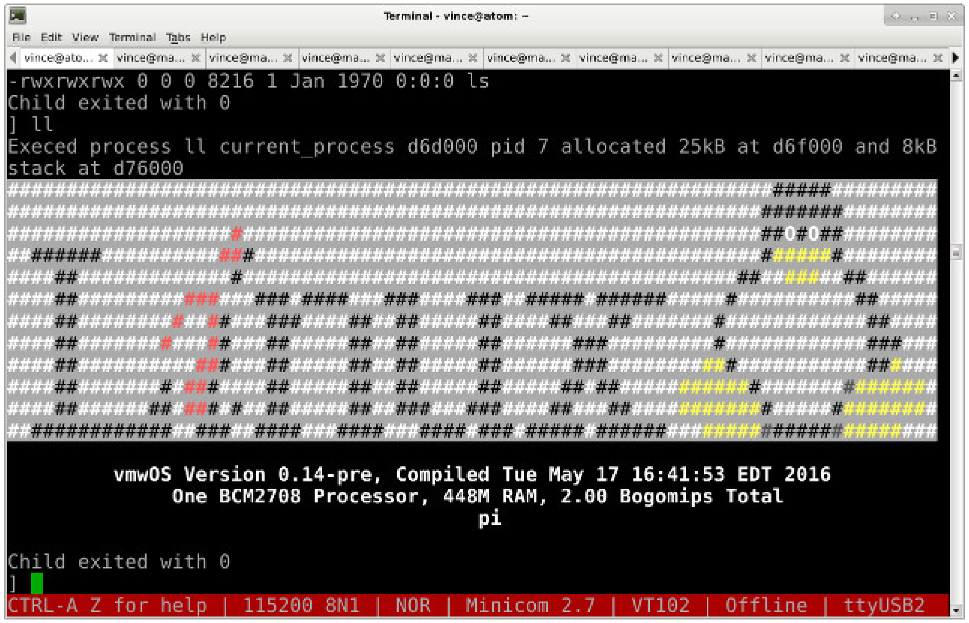
PAPI / perf_event / HPC Work

- Fast performance counter reads on x86 and ARM
- Raspberry Pi Cluster

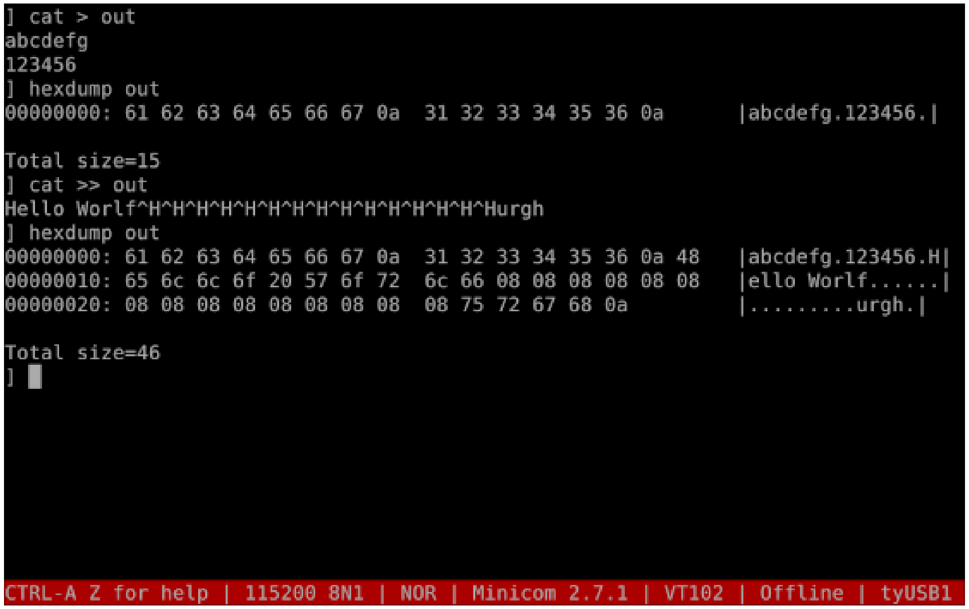


https://web.eece.maine.edu/~vweaver/projects/pi-cluster/

VMWos – a custom Raspberry Pi Operating System


Terminal - vince@atom: ~
File Edit View Terminal Tabs Help
vince@ato... vince@ma... vince@ma... vince@ma... vince@ma... vince@ma... vince@ma... vince@ma... vince@ma... vince@ma...

```
-rwxrwxrwx 0 0 0 8216 1 Jan 1970 0:0:0 ls
Child exited with 0
] ll
Execed process ll current_process d6d000 pid 7 allocated 25kB at d6f000 and 8kB
stack at d76000
###############################################################################
###############################################################################
##################################################################O#O##########
###############################################################################
###############################################################################
###############################################################################
###############################################################################
###############################################################################
###############################################################################
###############################################################################
###############################################################################
###############################################################################

         vmwOS Version 0.14-pre, Compiled Tue May 17 16:41:53 EDT 2016
              One BCM2708 Processor, 448M RAM, 2.00 Bogomips Total
                                        pi

Child exited with 0
]
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7 | VT102 | Offline | ttyUSB2
```

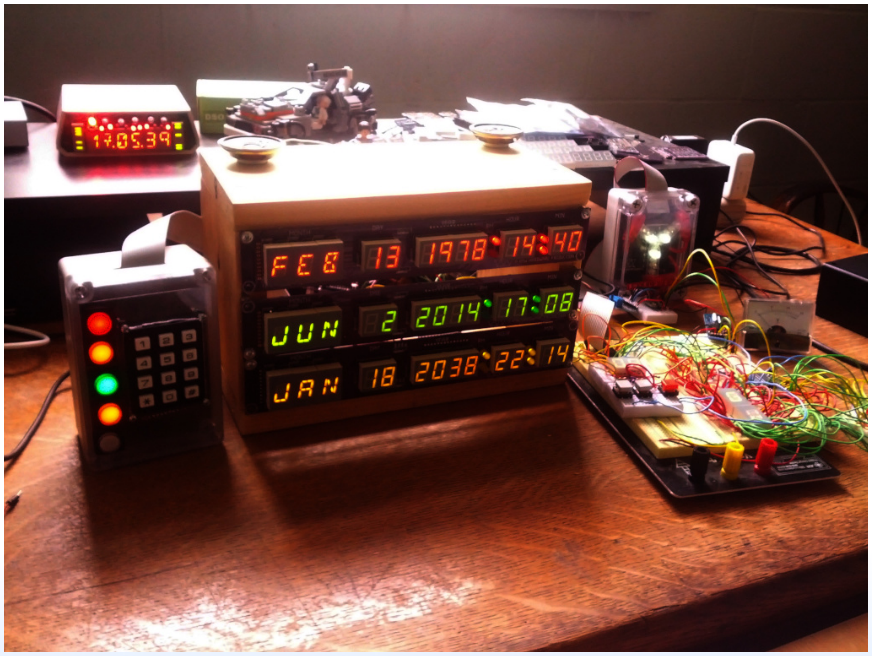


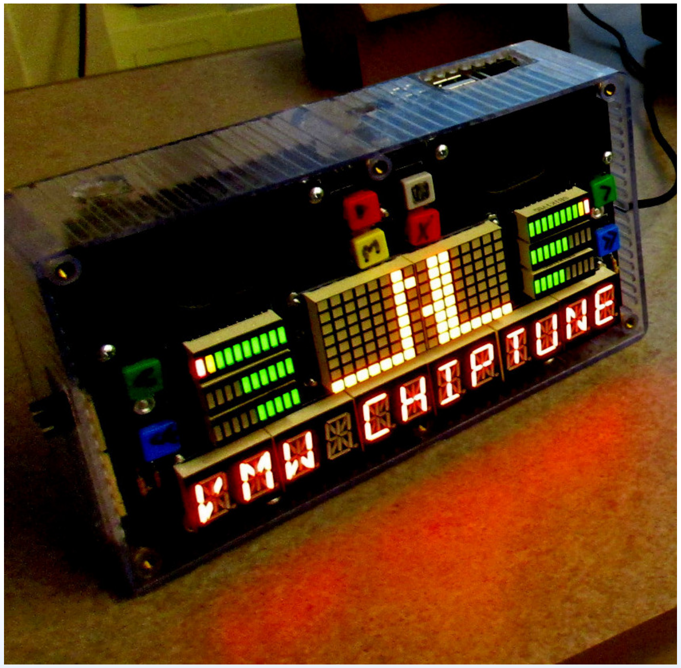
```
] cat > out
abcdefg
123456
] hexdump out
00000000: 61 62 63 64 65 66 67 0a  31 32 33 34 35 36 0a     |abcdefg.123456.|

Total size=15
] cat >> out
Hello Worlf^H^H^H^H^H^H^H^H^H^H^H^H^H^H^Hurgh
] hexdump out
00000000: 61 62 63 64 65 66 67 0a  31 32 33 34 35 36 0a 48  |abcdefg.123456.H|
00000010: 65 6c 6c 6f 20 57 6f 72  6c 66 08 08 08 08 08 08  |ello Worlf......|
00000020: 08 08 08 08 08 08 08 08  08 75 72 67 68 0a        |.........urgh.|

Total size=46
]
CTRL-A Z for help | 115200 8N1 | NOR | Minicom 2.7.1 | VT102 | Offline | tyUSB1
```

http://www.deater.net/weave/vmwprod/vmwos/

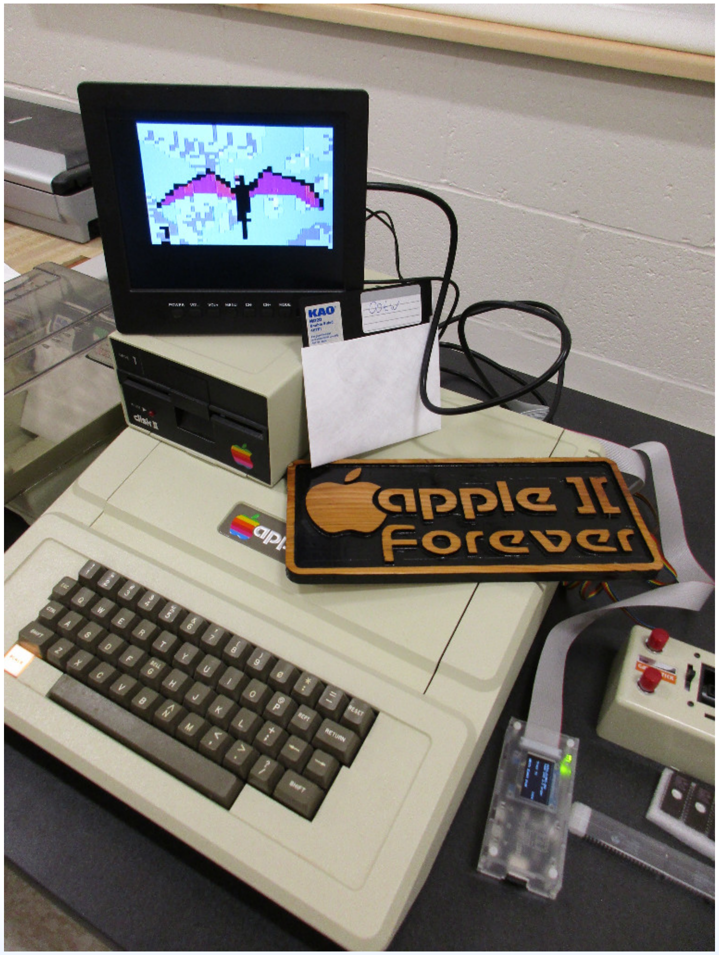
Fun Hardware Projects

- Related to Embedded Systems course I teach
- Lots of projects with blinking lights/music





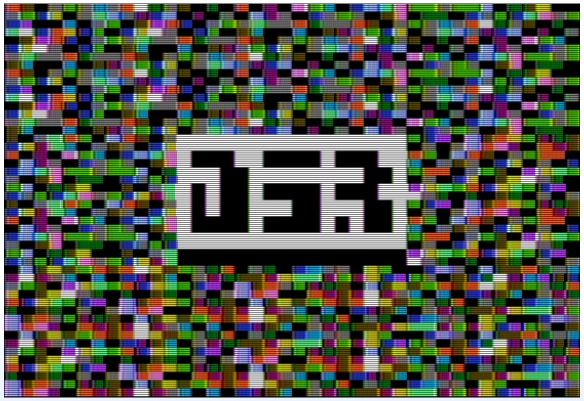
Apple II/II+ Background (1977)

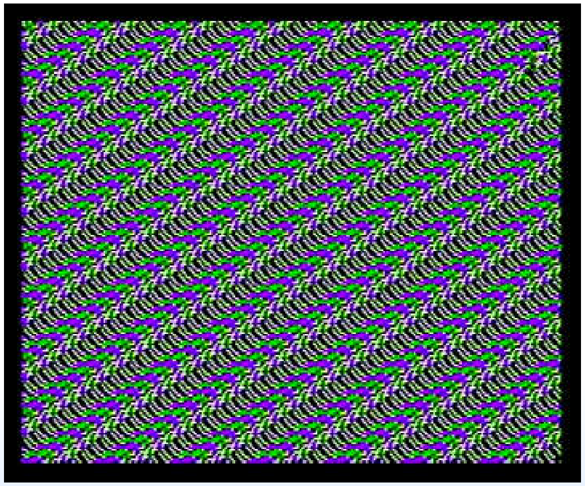


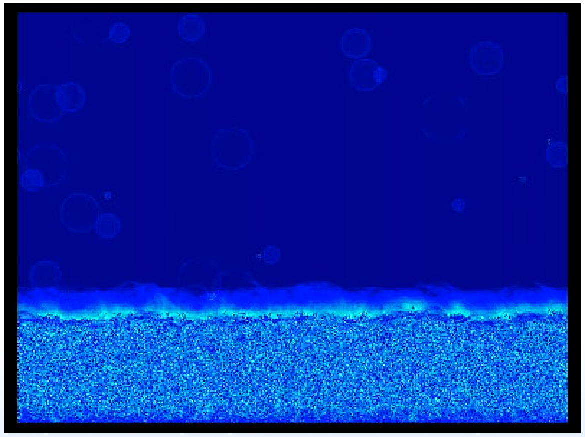
- 1MHz 6502
- 4k-48k RAM
- Discrete 7400 series logic
- cassette (later 140k disks)
- Bitbang Speaker
- 40x24 text mode
- 40x48 15 color lo-res
- 140x192 6-color hi-res
- need 16k for hi-res graphics
- BASIC in ROM

Joined the Demoscene

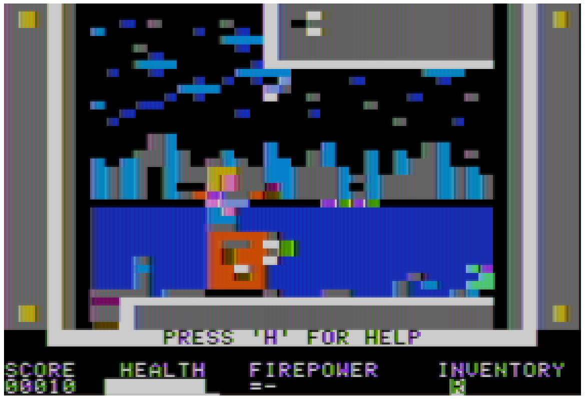
- Essentially European Programming Competitions
- I specialize in Apple II and Raspberry Pi demos, especially size-coding (demos less than 256 bytes)
- Ongoing! (Ascension Holiday in Europe)

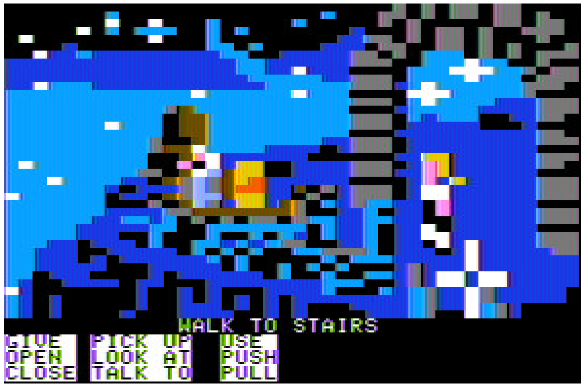


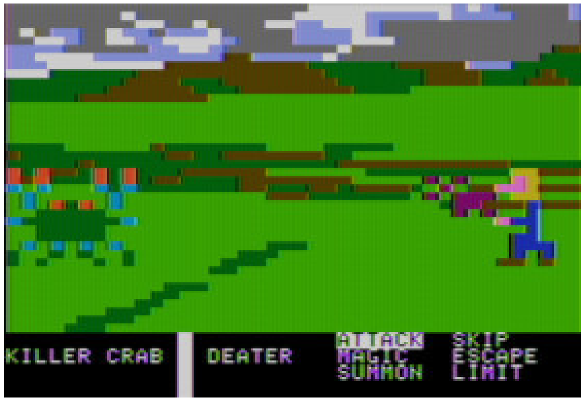


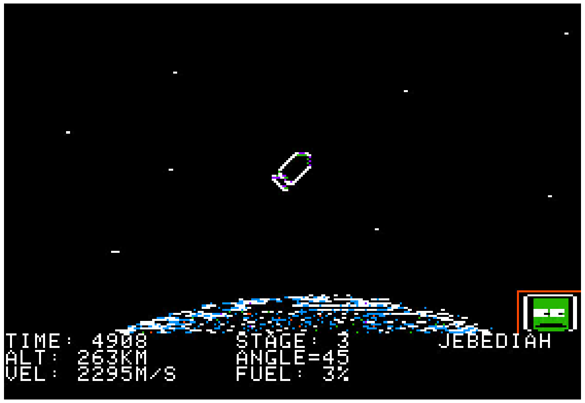


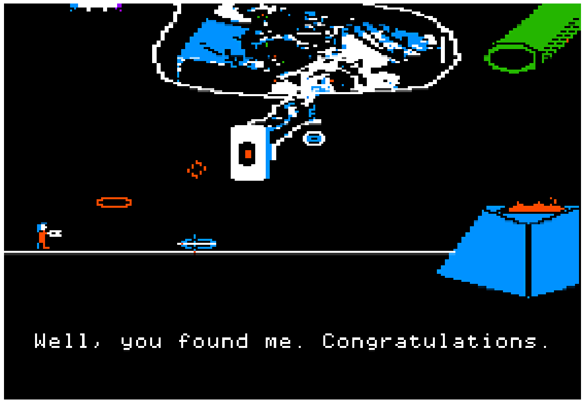
Game Demakes

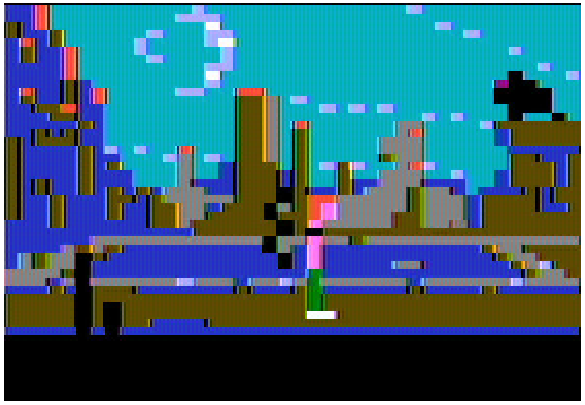




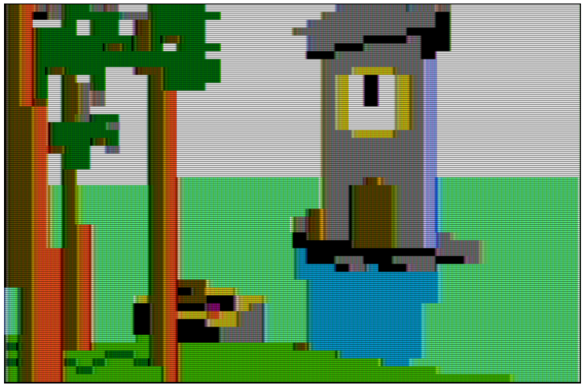


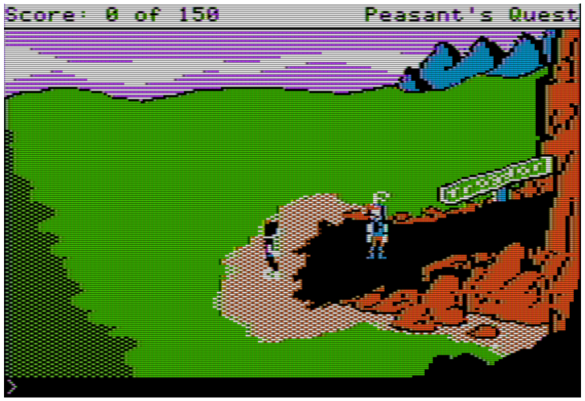


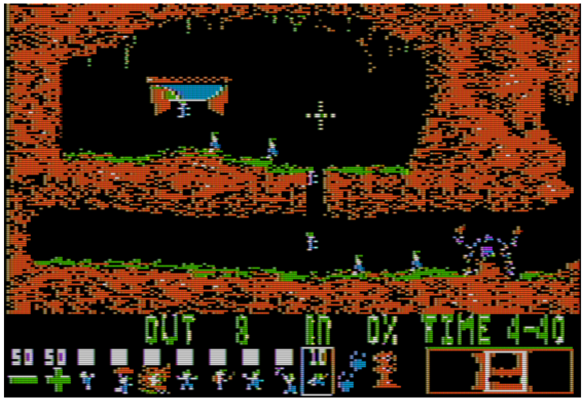

Well, you found me. Congratulations.



More Game Demakes







http://www.deater.net/weave/vmwprod/demakes/

Back to Academic HPC Work

Are there bugs in HPC code/systems?

- More worrying, are there security bugs in such systems?
- If there are, does anyone care?

(I've been told that large HPC installations are so hardened they wouldn't be affected by regular security bugs)

This work came from a bug where PAPI crashed Linux

- Just running PAPI unit tests crashed Linux machine completely
- Managed to trigger this on ICL server during the ICL retreat while we were all remote
  (I think the statute of limitations has passed so I can admit that)

How Can You Avoid a Crash?

- Send a bug report / bugfix upstream to the Linux developers (we did)
- Make a test suite you run on every new kernel to make sure the same bug doesn't happen again `https://github.com/deater/perf_event_tests`
- These are reactionary though, can only help *after* a bug is found, which is too late
- **Can we pre-emptively find new bugs before they are a problem?**

Why do we need an Operating System anyway?

- On most processors to gather performance info need to access special hardware registers (MSRs on x86)
- Giving a user full access to these can be dangerous, on x86 can easily take over whole system if unrestricted MSR access
- The Operating System can abstract away differences in machines, as well as make sure only good MSR accesses are happening

How Can User Code Crash Linux?

- Linux Kernel is written in C, which famously doesn't check bounds when accessing memory
- If you can get code to write values off the end of memory allocations, can corrupt data
- Worse, if you can over-write code you can take control of computer
- Local variables are stored on the stack, off the end of that is stored return value
- Anything that crashes program can be exploitable

How Does User Code Talk to the Kernel?

- System Calls (syscalls)
- Syscalls are implemented in various ways on modern systems
- The traditional way on Linux was:
  - Put the syscall number in a register
  - Set the parameters in various registers
  - Run a syscall instruction (often a software interrupt).
  - The kernel then notices and calls the appropriate internal code

A Simple Syscall

```
ssize_t write(int fd, const void *buf, size_t count);
```

- Only three inputs, in theory could audit all possible paths to code.
- Even with just 3 inputs, inside the kernel there are a lot of issues (file descriptors have many types, etc).

A Complex Syscall — perf_event_open()

```
int perf_event_open(struct perf_event_attr *attr,
                    pid_t pid, int cpu, int group_fd,
                    unsigned long flags);
```

- The `perf_event_attr` struct has 40+ fields that interact in complex ways with the other arguments
- For more info check the (extremely long) manpage documentation
- It's a convoluted and complex manpage, (I can say that as I wrote most of it)

Testing perf_event_open()

- PAPI and other perf tools use this interface
- There are too many combinations of arguments to test every possible combination in a reasonable amount of time
- Is there a way to automatically scan for errors?

fuzzing

- Automatically scanning for errors by trying random inputs
- Term invented by Barton Miller (Wisconsin) in the 1980s when noticed line noise on bad dial-up connection crashed many UNIX utils
- It is now a well-established technique, with many fuzzers being available for code at all levels of the programming stack

First steps with Trinity

- Kernel developer Dave Jones has a generic kernel fuzzer known as Trinity
- I contributed `perf_event_open` support to Trinity
- This found a serious root-exploit, CVE-2013-2094
- This led me to making a more targeted fuzzer

The perf_fuzzer

`https://web.eece.maine.edu/~vweaver/projects/perf_events/fuzzer/`

- Targeted fuzzer, aimed only at `perf_event_open()` interface
- It knows what valid events look like, and creates almost but not quite valid events when testing
- Also tests other, related, system calls that operate on file descriptors returned by `perf_event_open()` close, read, write, ioctl, mmap, prctl, fork, poll, access

Why not just use Syzkaller?

- Since the introduction of `perf_fuzzer` other more advanced kernel fuzzers have been developed
- Most well known is probably Vyukov's Syzkaller
- `perf_fuzzer` still finds bugs missed by this, due to its targeted nature (rather than pure random search)
- Could probably spend time enhancing Syzkaller to do better

List of Bugs Found

- Over 30 major (crashing or exploitable bugs found)
- Many WARNING or BUG messages triggered
- Also various correctness and compatibility bugs found

Short Summary of Major Bugs Found

Linux perf_event security bugs found by fuzzers. (T=Trinity, P=perf_fuzzer, H=honggfuzz, S=Syzkaller)

Which	Type	CVE	Fixed in Linux	Description
T	root exploit	CVE-2013-2094	3.9 8176cced706b5e5d	32/64 bit cast
P	crash	-	3.10 9bb5d40cd93c9dd4	mmap accounting hole
P	crash	-	3.10 26cb63ad11e04047	mmap double free
P	panic	-	3.11 d9f966357b14e356	ARM array out of bounds
P	root exploit	CVE-2013-4254	3.11 c95eb3184ea1a3a2	ARM event validation
P	panic	-	3.11 868f6fea8fa63f09	ARM64 array out of bounds
P	panic	-	3.11 ee7538a008a45050	ARM64 event validation
P	panic	-	3.13 6e22f8f2e8d81dca	alpha array out-of-bounds
P/T	crash	CVE-2013-2930	3.13 12ae030d54ef2507	perf/ftrace wrong permissions check
P	crash	-	3.14 0ac09f9f8cd1fb02	pagefault ftrace cr2 corruption
P	crash	-	3.15 46ce0fe97a6be753	race when removing event
P	crash	-	3.15 ffb4ef21ac4308c2	function cannot handle NULL return
P	reboot	-	3.17 3577af70a2ce4853	race in perf_remove_from_context()
P	crash	-	3.19 98b008dff8452653	misplaced parenthesis in rapl_scale()
P	crash	-	3.19 c3c87e770458aa00	fix the grouping condition
P	crash	-	3.19 a83fe28e2e453924	Fix put_event() ctx lock
P	crash	-	3.19 af91568e762d0493	IVB-EP uncore assign events

Short Summary of Major Bugs Found (page2)

P	crash	-	4.0	d525211f9d1be8b5	Fix perf_callchain() hang
H	memleak	-	4.0	a83fe28e2e453924	fix put_event() ctx leak
P	crash	-	4.1	8fff105e13041e49	arm64/arm reject groups spanning PMUs
P	crash	-	4.1	15c1247953e8a452	snb_uncore_imc_event_start crash
P	crash	-	4.2	57ffc5ca679f499f	Fix AUX buffer refcounting
P	panic	-	4.5	fb822e6076d97269	powerpc: Oops destroying hw_breakpoint event
P	crash	-	4.8	0b8f1e2e26bfc6b9	crash in perf_cgroup_attach
P	crash	-	4.9	7fbe6ac02485504b	vmalloc stack unwinder crash
P(?)	exploit	CVE-2017-6001	4.10	321027c1fe77f892	perf_event_open() vs. move_group race
S	bug	-	4.11	e552a8389aa409e2	Fix use-after-free in perf_release()
P	crash	-	4.15	99a9dc98ba52267c	BTS causes crash with KPTI meltdown fixes
P	crash	-	4.20	472de49fdc53365c	BTS crash, uninitialized ptr
S	crash	-	5.3	1cf8dfe8a661f046	Race between close() and fork()
P	panic	-	5.5	242bff7fc515d8e5	i915 null pointer dereference
P	crash	-	5.12	d88d05a9e0b6d935	NULL pointer dereference with PEBS on haswell

perf_fuzzer is open source

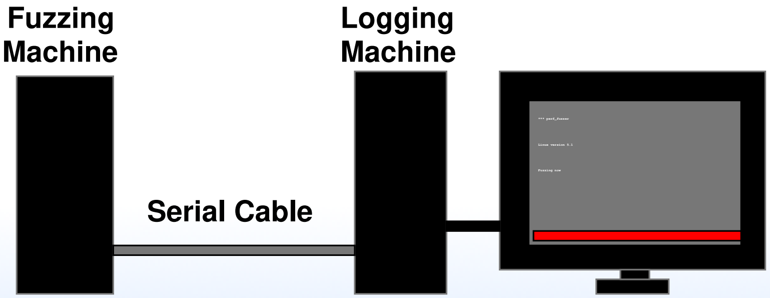
- This means other people have used it to find bugs
- A number of bugs found in ARM devices, Android phones famously had really buggy perf implementations (why was it even enabled)
- Someone (not me) possibly was even getting bug bounties for reporting these

Use by Community

- Linux Kernel perf developers use the perf_fuzzer to test patches before submitting
- Most notably the ARM developers heavily use it

Fuzzing Setup

- Run fuzzer on one machine
- Logging machine over serial port
- Why separate machine? Crashes can crash so hard the log doesn't make it to disk or even the display
- I've had machines crash so hard they took out the whole Ethernet subnet



```
*** perf_fuzzer

Linux version 5.1

Fuzzing now
```

Fuzzing Output

```
*** perf_fuzzer 0.32-rc0 *** by Vince Weaver

        Linux version 5.18.0-rc1+ x86_64
        Processor: Intel 6/60/3

        Stopping after 50000
        Watchdog enabled with timeout 60s
        Will auto-exit if signal storm detected
        Seeding RNG from time 1653664198

        To reproduce, try:
                echo 1 > /proc/sys/kernel/nmi_watchdog
                echo 0 > /proc/sys/kernel/perf_event_paranoid
                echo 750 > /proc/sys/kernel/perf_event_max_sample_rate
                ./perf_fuzzer -t OCIRMQWPFpAi -s 50000 -r 1653664198

        Fuzzing the following syscalls: mmap perf_event_open close read write ioctl fork prctl poll
        Also attempting the following: busy-instruction-loop accessing-perf-proc-and-sys-files trashi
        *NOT* attempting the following: signal-handler-on-overflow

        Pid=2351163, sleeping 1s
```

```
==================================================
Starting fuzzing at 2022-05-27 11:09:59
==================================================
Cannot open /sys/kernel/tracing/kprobe_events
Iteration 10000, 124955 syscalls in 42.79 s (2.920 k syscalls/s)
        Open attempts: 120366  Successful: 942  Currently open: 902
                EPERM : 23
                ENOENT : 1049
                E2BIG : 9450
                EBADF : 6826
                EACCES : 5030
                ENODEV : 4
                EINVAL : 96821
                ENOSPC : 5
                EOVERFLOW : 4
                EOPNOTSUPP : 212
                Trinity Type (Normal 90/29979)(Sampling 18/30089)(Global 800/30137)(Random 34/30161)
                Type (Hardware 227/16831)(software 294/16366)(tracepoint 64/16025)(Cache 55/15082)(cp
        Close:  40/42 Successful
        Read:   45/60 Successful
        Write:  0/50 Successful
        Ioctl:  21/67 Successful: (ENABLE 8/8)(DISABLE 2/2)(REFRESH 3/8)(RESET 3/4)(PERIOD 0/3)(SET_
        Mmap:   418/1082 Successful: (MMAP 418/1082)(TRASH 154/169)(READ 130/133)(UNMAP 182/193)(AUX
        Prctl:  900/900 Successful
        Fork:   445/445 Successful
        Poll:   890/902 Successful
        Access: 136/961 Successful
        Overflows: 0  Recursive: 0
        SIGIOs due to RT signal queue full: 0
```

Bug Found

```
[96289.009646] BUG: kernel NULL pointer dereference, address: 0000000000000150
[96289.017094] #PF: supervisor read access in kernel mode
[96289.022588] #PF: error_code(0x0000) - not-present page
[96289.028069] PGD 0 P4D 0
[96289.030796] Oops: 0000 [#1] SMP PTI
[96289.034549] CPU: 0 PID: 0 Comm: swapper/0 Tainted: G        W         5.11.0-rc5+ #151
[96289.043059] Hardware name: LENOVO 10AM000AUS/SHARKBAY, BIOS FBKT72AUS 01/26/2014
[96289.050946] RIP: 0010:intel_pmu_drain_pebs_nhm+0x464/0x5f0
[96289.056817] Code: 09 00 00 0f b6 c0 49 39 c4 74 2a 48 63 82 78 09 00 00 48 01 c5 48 39 6c 24 08 7
[96289.076876] RSP: 0000:ffffffff822039e0 EFLAGS: 00010097
[96289.082468] RAX: 0000000000000002 RBX: 0000000000000155 RCX: 0000000000000008
[96289.090095] RDX: ffff88811ac118a0 RSI: ffffffff82203980 RDI: ffffffff82203980
[96289.158414] Call Trace:
[96289.161041]  ? update_blocked_averages+0x532/0x620
[96289.166152]  ? update_group_capacity+0x25/0x1d0
[96289.171025]  ? cpumask_next_and+0x19/0x20
[96289.175339]  ? update_sd_lb_stats.constprop.0+0x702/0x820
[96289.181105]  intel_pmu_drain_pebs_buffer+0x33/0x50
[96289.186259]  ? x86_pmu_commit_txn+0xbc/0xf0
[96289.190749]  ? _raw_spin_lock_irqsave+0x1d/0x30
```

Tracking down and Reporting Bugs

- Time consuming
- Kernel oops report usually isn't enough (for security modern kernels make it harder to match symbols/addresses)
- If new bug can "git-bisect" kernel to find where introduced
- Even if straightforward bug can take a while to make sure bug gets fixed properly (kernel bureaucracy)

Reproducible Test Cases

- Devs like small repeatable test cases
- By saving random number seed and other info can often (but not always) get repeatable fuzzer runs
- These can still be millions of instructions
- Sometimes can get those by recording traces, but this takes forever

Is this the only security issue with perf measurement?

- `perf_event` is often disabled by default. Why?
- Partly this was due to the bugs found by fuzzer (sorry!)
- Other types of attacks, information leakage, where one user can figure out what another is doing be carefully measuring time / cycles / other metric of a shared resource (cache, CPU)
- This is much easier if perf enabled, so often disabled

Current Status

- Can now fuzz for months with no bugs
- Linux kernel developers run the fuzzer before submitting so bugs happen less often
- Can't rest! new platforms and perf features added all the time
- Many of the features are hardware dependent so might not catch all issues on the few machines I test on
- Other fuzzers (Syzkaller) with more manpower behind

Results of Research

- It took 7+ years but the `perf_event_open()` syscall seems to be robust against fuzzing
- Lots of trouble getting published / grant money
- Did get a couple of non-academic publications, including a relatively highly cited tech report

Future Work: Is other software vulnerable?

What about the `perf` tool?

- Can easily crash on malformed (poorly documented) `perf.data` analysis file
- Was trying to generate these from PAPI
- Making a small `perf.data` fuzzer found more bugs
- In theory could write an exploit where you write a malicious `perf.data` file and get/trick someone to open it with a buggy version of perf
- This work is ongoing

Questions?

`vincent.weaver@maine.edu`

`https://web.eece.maine.edu/~vweaver/`

We're always looking for Grad Students