Analyzing Block Device Timing Events as a Source of Entropy

by Kirill Sinitski and Mike Ounsworth

ICMC18

Random Numbers

```
int getRandomNumber()
{
    return 4;  // chosen by fair dice roll.
               // guaranteed to be random.
}
```

Motivation

- Comment in Linux kernel's random.c says:

"add_disk_randomness() uses what amounts to the seek time .., as input to the entropy pool. **Note that high-speed solid state drives with very low seek times do not make for good sources of entropy**, as their seek times are usually fairly consistent."

- Our work aims to provide data to validate / invalidate this statement.

About us

- FIPS and Common Criteria Labs
  - Accredited testing laboratories
  - NIAP, NIST, CSE
- Entrust Datacard
  - Identity and Access Management
  - ID and financial card printers
  - SSL certificates

Analyzing Block Device Timing Events as a Source of Entropy

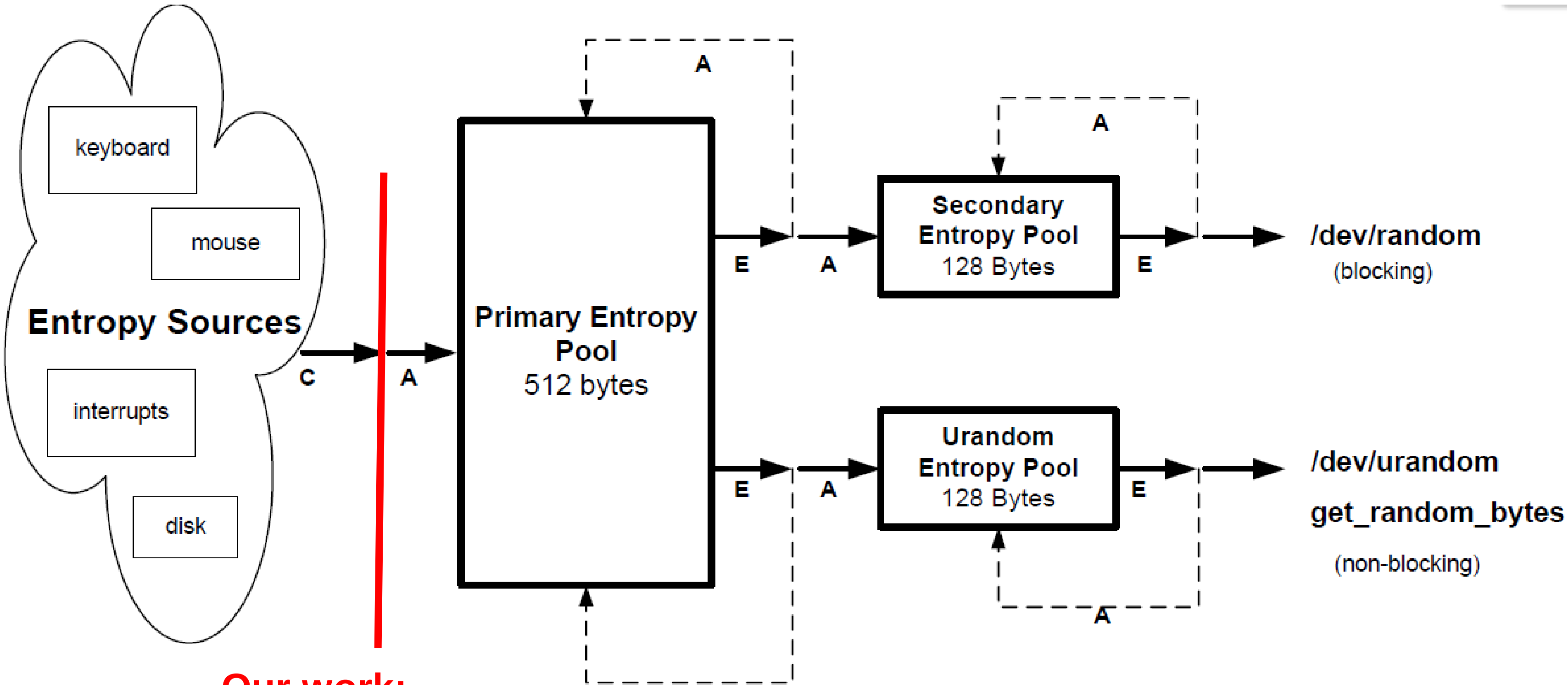
KERNEL

Entropy - Definitions

- General:
  Lack of order or unpredictability.
- Physics:
  A measure of disorder in the universe.
- Information Theory:
  Uncertainty of a random variable.
- Computing:
  The randomness collected by an operating system or application for use in cryptography or other uses that require unpredictable data.
  - Linux PRNG collects entropy from the timestamps of various events (user inputs, disk timings, and interrupt timings)

Linux Random Number Generator

<linux kernel>/drivers/char/random.c



Our work:

Analysis of raw entropy collected

Entropy sources for random.c (as of 4.16)

- `add_device_randomness(..)`
- `add_input_randomness(..)`
- `add_interrupt_randomness(..)`
- `add_disk_randomness(..)`
- `add_hwgenerator_randomness(..)`

Timing of events

- Jiffies
  - Kernel-level interrupt-based timer
  - Timer tick rate ‘HZ’ set at kernel compile time
    - Typically 100 – 250 Hz (4 ms to 10 ms intervals)
  - (mostly) Hardware agnostic

- CPU cycles
  - Hardware-specific
  - CPU clock cycles
  - Could be variable, depending on CPU features (e.g., dynamic frequency scaling)

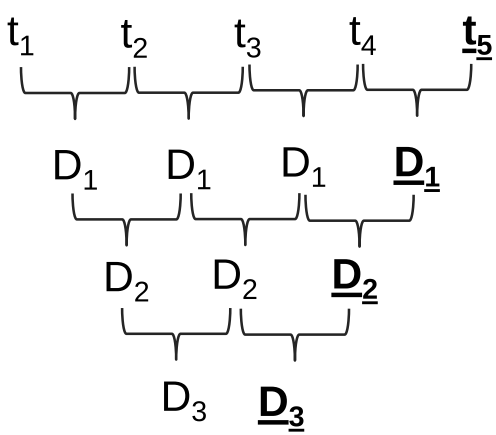
What is inside an entropy event?

```
static void add_timer_randomness(struct timer_rand_state *state, unsigned num,
                                 const char * source)
{
    struct {
        long jiffies;      timestamp of event in jiffies
        unsigned cycles;   timestamp of event in cycles
        unsigned num;      event type (keyboard scan code, interrupt code)
    } sample;
    …
    mix_pool_bytes(&input_pool, &sample, sizeof(sample), NULL);
    …
    credit_entropy_bits(&input_pool, min_t(int, fls(delta>>1), 11));
}
```

Sample event: `jiffies: 4294671006, cycles: 2129478438, num: ide1`
`entropy_credited: 2`

credit_entropy_bits(..) Linux Entropy Estimator

- How many bits of randomness in a given timestamp sample t_n?
- Polynomial interpolation – 1st,2nd,3rd Differences



MIN_n - the minimum D_i value for the newly added sample t_n.

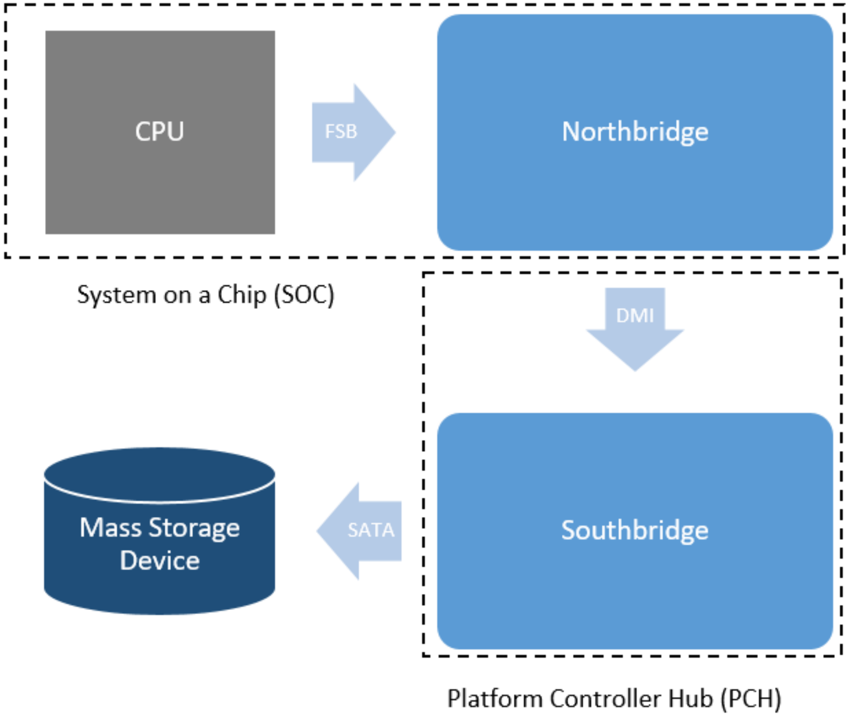
$$ent_n = \begin{cases} 0 & \text{if } MIN_n \leq 1 \\ \lfloor \log_2 (MIN_n) \rfloor & \text{if } 2 \leq MIN_n \leq 2^{12} \\ 11 & \text{otherwise} \end{cases}$$

Analyzing Block Device Timing Events as a Source of Entropy

FIRMWARE & CONTROLLERS

Hardware Architecture

- Hub Architecture (HA)
- Platform Controller Hub (PCH)
- System on a Chip (SoC)



- Test platform – Intel C610/X99 'Wellsburg" PCH
  - 20 Gbit/s DMI 2.0, SATA 3.0 controller

Interfaces

- Storage Device Interfaces
  - AHCI vs Parallel ATA (IDE emulation)
  - AHCI HBA (host bus adapter) DMA (direct memory access)
- SATA
  - Native Command Queuing (NCQ)
  - Tagged Command Queuing (TCQ)
  - SATA 3.2 at 1969 MB/s, ATA-3 at 33 MB/s
- NVMe
  - Non-volatile Memory Express (NVMe)
  - PCIe and M.2

Boot Process

- Basic Input/Output System (BIOS)
  - BIOS executes Master Boot Record (MBR)
  - MBR executes Grand Unified Bootloader (GRUB)
  - GRUB executes start_kernel()
- Universal Extensible Firmware Interface (UEFI)
  - Boot Services load EFI System Partition (ESP)
  - Kernel boot stub (acting as an EFI application)
  - EFI handover to GRUB2
  - GRUB2 executes start_kernel()

Analyzing Block Device Timing Events as a Source of Entropy

HARDWARE

Test Platform

- Test platform – Dell Power Edge T430
  - Intel C610/X99 with Intel Xeon E5-2603v3
  - 20 Gbit/s DMI 2.0, integrated SATA 3.0 controller

Hard Drives and Performance

- Seagate Hybrid Drive ST2000DX001 2TB
  - 210 MB/s, 4.2ms seek
  - 7200 RPM, 64MB Cache
- WD VelociRaptor WD1500HLFS 150GB
  - 126 MB/s, 4.6ms seek
  - 10000 RPM, 16MB Cache
- Seagate Cheetah 15K.5 1TB
  - 128 MB/s, 6.6ms seek
  - 15000 RPM, 16MB Cache
- Western Digital Black 2TB
  - 164 MB/s, 12.0ms seek
  - 7200 RPM, 64 MB Cache

Solid State Drives and Performance

- Intel 540S Series 480GB
  - 560 MB/s, 0.05 ms access
- Intel 335 Series 180GB
  - 500 MB/s, 0.2 ms access
- WD SiliconEdge Blue 128GB
  - 250 MB/s, 0.25 ms access

Analyzing Block Device Timing Events as a Source of Entropy

TEST HARNESS

Log each entropy event to kernel logs

```
random.c :: add_timer_randomness( .. )
{
  …
  mix_pool_bytes(&input_pool, &sample, sizeof(sample), NULL);
  …
  credit_entropy_bits(&input_pool, min_t(int, fls(delta>>1), 11));

  // Test Harness – log raw data
  printk(KERN_DEBUG, source, input_pool.entropy_total,
        input_pool.entropy_count, sample.jiffies,
        sample.cycles, min_t(int, fls(delta>>1), 11) );
}
```

Typical `journalctl -k` log entry:

```
Jan 12 09:56:24 localhost kernel: DISK_RANDOMNESS: ent_total: 42,
ent_count: 244, jiffies: 4294671331, cycles: 2364333088, min_t: 2
```

Experimental setup

Rationale: wanted to best represent entropy collection in a datacenter environment.

Setup:

- Dell PowerEdge T430 with Intel Xeon E5-2603v3
- CentOS 7.2 Minimal
- Kernel 3.10.0-327 (from centos.org mirror)
  - Re-compiled using default options to insert our test harness into random.c
- Continuous reboot cycles
  - Until 1.5 million events or 3 days elapsed (~ 400 – 3,000 reboots)

reboot.sh

```
…
# log data
journalctl -k -p 7 | grep RANDOMNESS >> $DATA_FILE
…
if (( ELAPSED_TIME > EXPRIMENT_MAX_RUNTIME
        -o `wc -l < $DATA_FILE` > NUM_SAMPLES_TO_COLLECT ))
then
        echo "Ending experiment”
else
        reboot
fi
```

Analyzing Block Device Timing Events as a Source of Entropy

DATA

Data

- Analyzed
  - Cycles and Jiffies extracted and processed
  - Statistical analysis of 6 bit and 8 bit sample with NIST tool
- Collected, not analyzed
  - Internal entropy estimate per sample is extracted
    - Internal Linux estimate vs. post-hoc statistical analysis
  - Each reboot is logged with a timestamp
    - Variability between individual reboots
    - State of entropy pool at various stages during the boot cycle
    - Other possible research …

Set it free

- All our data, harness, scripts, and procedures are available on Github:

https://github.com/ounsworth/LinuxEntropyAnalysis

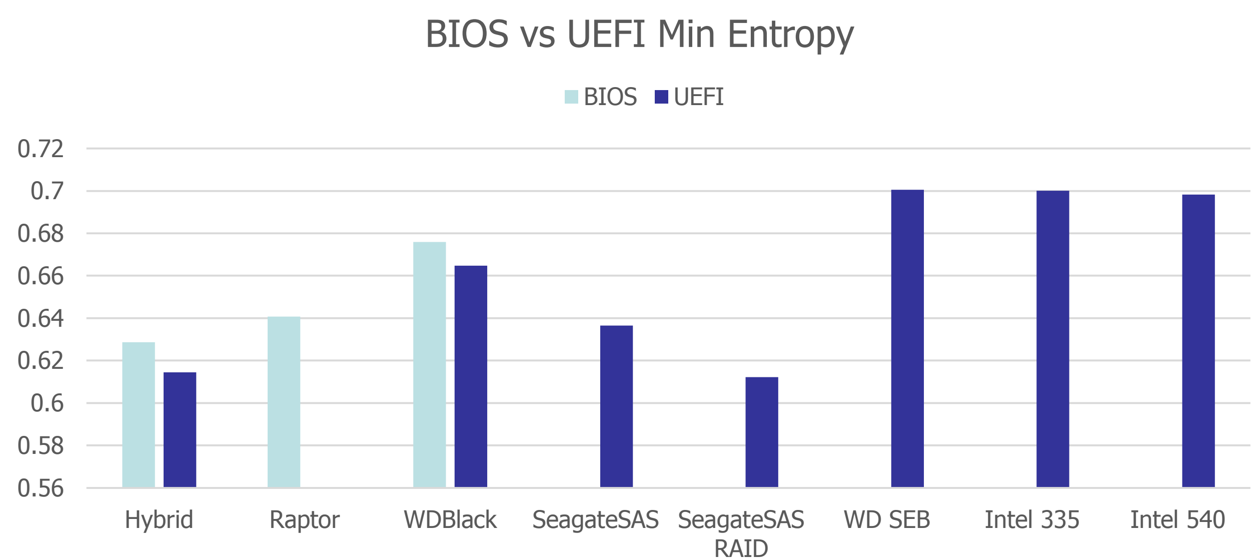
Analyzing Block Device Timing Events as a Source of Entropy

RESULTS

Post-hoc Entropy Analysis

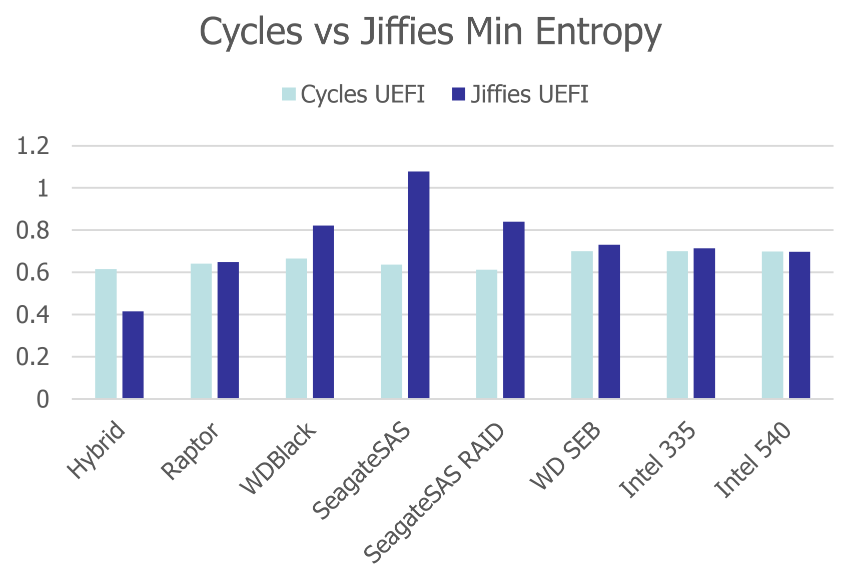
- NIST python tool
  - Collision
  - Partial collection
  - Markov
  - Compression
  - Frequency
- Min Entropy
  - Lowest of the test scores

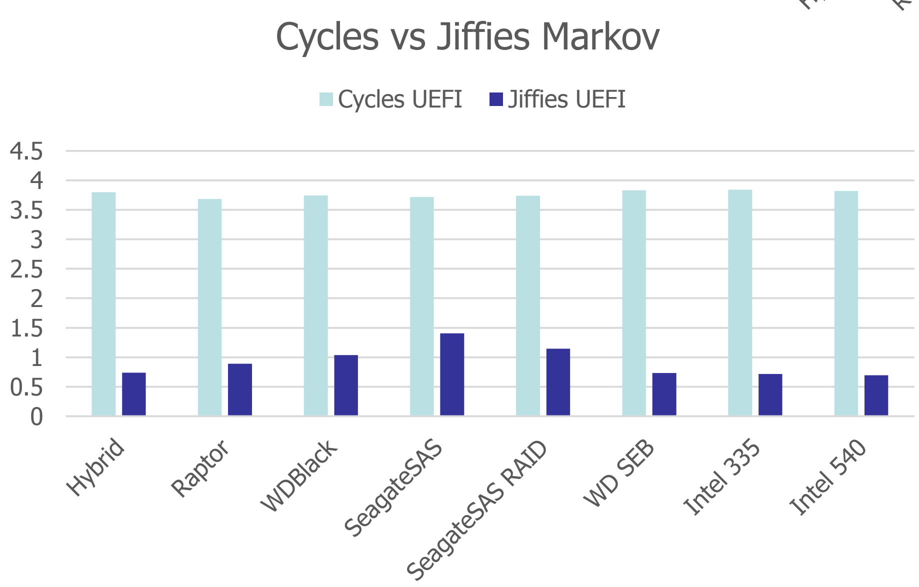
BIOS vs. UEFI


BIOS vs UEFI Min Entropy
BIOS
UEFI
0.72
0.7
0.68
0.66
0.64
0.62
0.6
0.58
0.56
Hybrid
Raptor
WDBlack
SeagateSAS
SeagateSAS RAID
WD SEB
Intel 335
Intel 540

Cycles vs. Jiffies

- Min Entropy

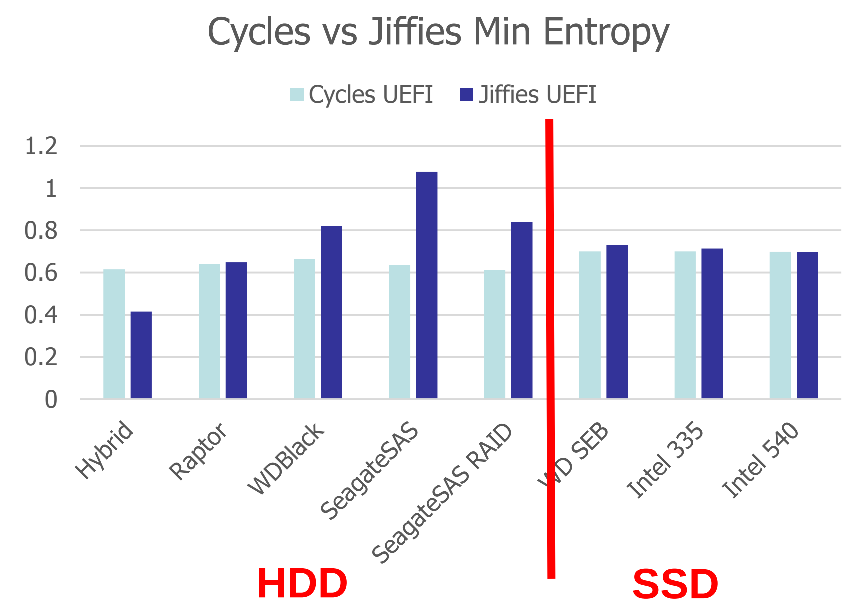


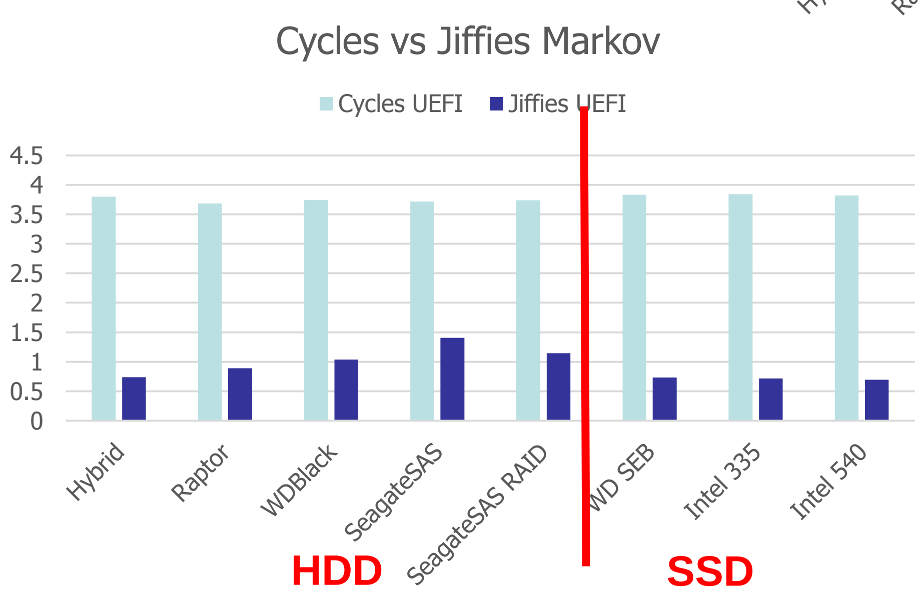


- Markov

HDD vs SSD

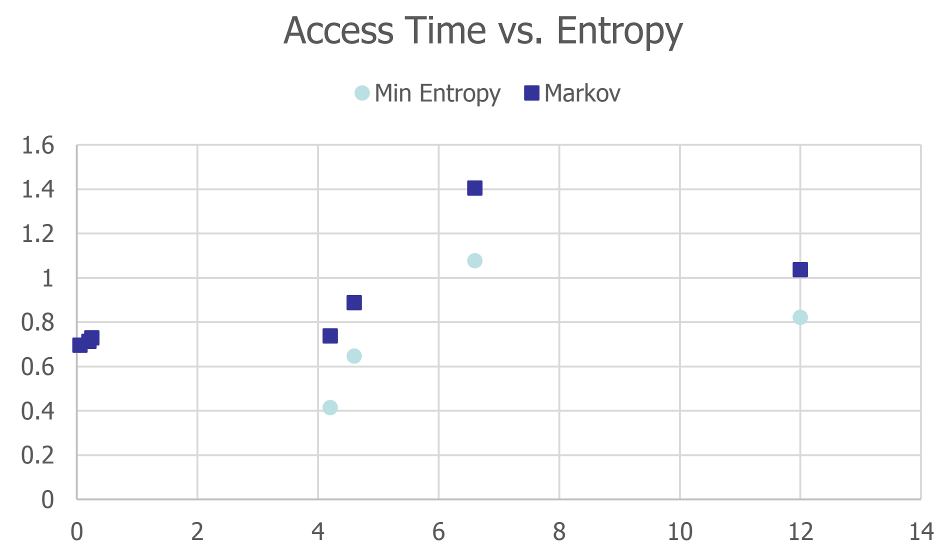
- Min Entropy





- Markov

Hardware Considerations



- Published Access Time/Latency numbers were not verified

Summary of Findings

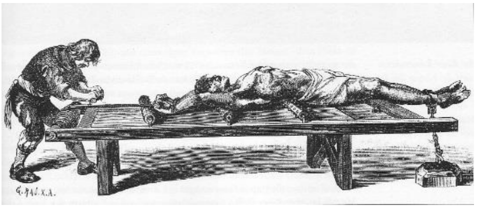
- BIOS vs UEFI are comparable for entropy generation
- Cycles vs Jiffies are comparable per non-duplicate event entropy using min-entropy
- Both SSD and HDD generate comparable entropy per event
- HDD generates 15x as many entropy events per reboot cycle as SSD
- HDDs generate ~50% duplicate jiffie events
- SDDs generate ~1% duplicate jiffie events

Speculation

- Speculation (noun) - the forming of a theory or conjecture <u>without</u> firm evidence
- SATA SSDs generate good per event entropy, just less events than HDDs
- Markov is more discerning test than min-entropy
- Disk drive speed and access times had no detectable effect on per event entropy within the same system, controller performance probably determines entropy
- NVMe parallelism has a potential to drastically change SDD performance as a source of entropy

Entropy Assessment Report

- Part of FIPS and CC certifications
  - Recommendation for Random Number Generation Using Deterministic Random Bit Generators, NIST SP 800-90A
  - Recommendation for the Entropy Sources Used for Random Bit Generation, NIST SP 800-90B
- Entropy Assessment Report (EAR)
  - Per source data analysis
  - Operating Conditions
  - Health Testing
  - Description of conditioning function
  - Description of DRBG seed creation



Questions?


HEY, KID PSSST. WANNA GET INTO A SEMANTIC ARGUMENT?
MY DAD WARNED ME ABOUT PEOPLE LIKE YOU!