9 A Vulnerability in Reduced Dakarand from PoC||GTFO 01:02

by joernchen of Phenoelit

I'm not a math guy, so this is a poor man's RNG analysis. Try it yourself at home!

9.1 Introduction

In PoC||GTFO 01:02, Dan Kaminsky proposed the following code for use as a Random Number Generator, arguing that the phase difference between a fast clock and a slow clock is sufficient to produce random bits in a high level language. This is a reduced version of his Dakarand program, with the intent of the reduction being that if there is any vulnerability within the code, that vuln ought to be exploitable.

```
// These functions form an RNG.
function millis()         {return Date.now();}
function flip_coin()
  {n=0; then = millis()+1; while(millis()<=then) {n=!n;} return n;}
function get_fair_bit()
  {while(1) {a=flip_coin(); if(a!=flip_coin()) {return(a);}}}
function get_random_byte()
  {n=0; bits=8; while(bits--){n<<=1; n|=get_fair_bit();} return n;}

// Use it like this.
report_console = function() {while(1){console.log(get_random_byte());}}
report_console();
```

Actually the above code boils down to the function flip_coin, which takes a boolean value n=0 and continuously flips it until the next millisecond. The outcome of this repeated flipping shall be a random bit. We neglect the get_fair_bit function mostly in this analysis, as it just slows down the process and adds almost no additional entropy. For gathering random bits we are just left with the clock ticking for us.

9.2 A Naive Analysis

In order to analyze the output of the RNG we need some of its output, so I simply put up a small HTML piece which would pull out 100.000 random bytes out of the above RNG and log it to the HTML document. Then a severe 90-minute DoS on my Firefox 24 happened, after which I managed to copy and paste one hundred thousand uint8_t results into a text file.

After messing with several tools like ministat, sort and uniq I could show with the following ruby script that this RNG (on my machine) has a strong bias towards bytes with low hamming weights:

```
#!/usr/bin/env ruby

f=File.open(ARGV[0])

h = Hash.new
f.each_line do |m|
  n = m.to_i
  if h[n].nil?
    h[n]=1
  else
    h[n] = h[n]+1
  end
end

t = h.sort_by do |k,v| v end
```

CAPTAIN VIDEO

Ideal for communicating with your microprocessor!

RS-232 interface x 32 characters $400
TV/TTY kbd./display (16 lines x 64 characters) $500
Keyboard/CRT Monitor (24 lines x 80 characters) $700

ORDER FORM

Ship To____________
____
____
Subtotal $______
Calif. residents add sales tax: ______
Please find a check enclosed in the amount of total order: $______
______ ______
Bill my BankAmericard. exp.
______ ______
Mastercharge: exp.
______ ______
My signature as purchaser: date

MICON INDUSTRIES
Micon Industries
252 Oak Street
Oakland, CA 94607
(415) 763-6033

```
t.each do |a|
  puts "Num:\t#{a[0]}"+
       "\tCount:\t#{a[1]}"+
       "\tWeight:\t#{a[0].to_s(2).split("").reject{|j|j=="0"}.count}"
end
```

The shortened output of this script on the 100k 8bit numbers is as follows. Note that the heavy hamming weights, like `11111111` are least common and the light hamming weights, like `00000000` are most common.

```
Value  Count  Weight
  255     22       8
  254     23       7
  251     28       7
  253     29       7
  127     32       7
  239     34       7
  191     34       7
  223     36       7
  247     37       7
  ...    ...     ...
  132   1173       2
   64   1821       1
   32   1881       1
   16   1922       1
    1   1934       1
    8   2000       1
    4   2042       1
    2   2133       1
  128   2145       1
    0   3918       0
```

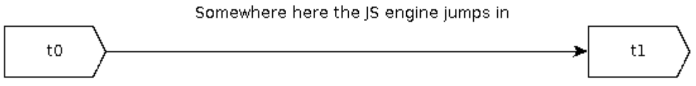
The table lists the Number which is the output of the RNG along with this number's hamming weight as well as the count of this number in total within the 100.000 random bytes. For a random distribution of all possible bytes we could expect roughly a count of 390 for each byte. But as we see, the number 0 with the hamming weight 0 peaks out with a count of 3918, whereas 255 with the hamming weight of 8 is generated 22 times by the RNG. That's not fair!

9.3 My fair bit is not fair!

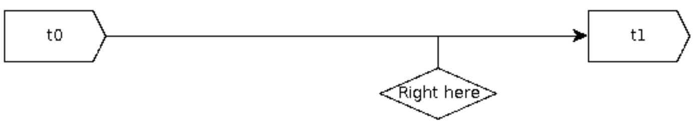
Real statistical analysis of an RNG is hard, and I will not attempt it here. Still, looking at a few simple distributions might give us a hint (alas, only a hint) of what might behind the unfairness.

First, a short recap on how this RNG works:

We've got a 1 millisecond timeslot from t0 to t1, where at t1 the flip_coin method will stop. The first call to get_random_byte can happen anywhere between t0 and t1:

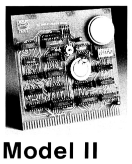


Let's say it is here:



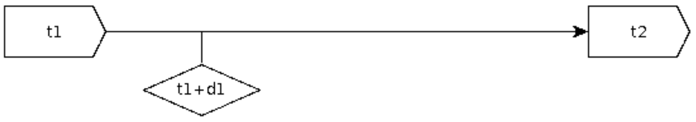
Now the algorithm happily flips the bit until t1 and hands over the result of this flipping as a random bit (note that we're omitting get_fair_bit here).

What is a
CLOCALPEEP?
Another name for the **CCB-II**, which is:
- a clock
  hour, minute, second
- a calendar
  day, day of week, month, year
- an audio alarm



All on one board for your
TRS-80 Model II
It includes a pacemaker battery which will give over 8 years of continuous timekeeping.
From the folks who brought you the best CP/M® for the Model II.
$175 plus shipping
Prepaid, COD, Mastercharge or Visa orders accepted. California residents add 6% sales tax.
TRS-80 is a trademark of Tandy Corp.
CP/M is a registered trademark of Digital Research Inc.
PICKLES & TROUT
P.O. BOX 1206, GOLETA, CA 93116, (805) 967-9563
Warning: Installation requires opening the Model II, which may void its warranty. We suggest that you wait until the warranty period has expired before installing the CCB-II.

Although we cannot predict the output of a single run of flip_coin, things get a bit more predictable when we make a lot of consecutive calls to flip_coin. Let's say we need the time d to process and store the result of flip_coin. So the next time we flip_coin we are at t1 + d1:



Now the RNG flips the coin until t2 in order to give us a random bit. As we are calling the RNG more than twice in a row, the next flip_coin is at t2+d2, and so on.

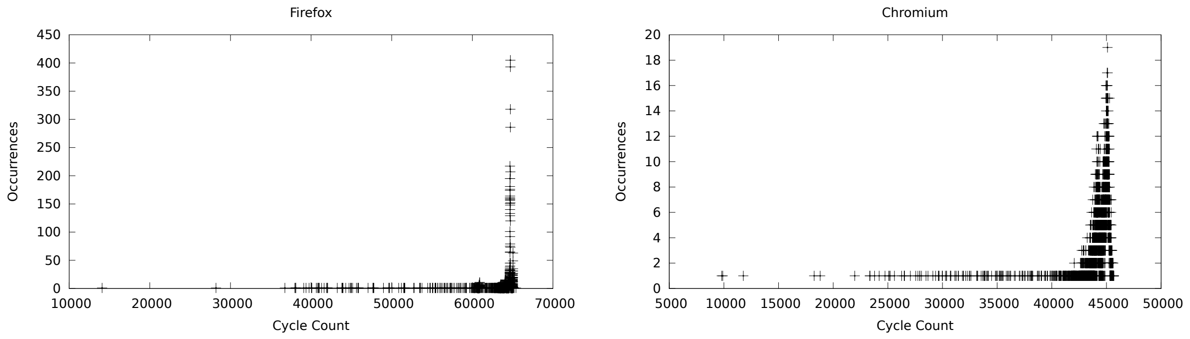
The randomness and fairness of the RNG's random bit depends on how fairly and randomly we get odd and even values of d, since that the same amount of flips yields the same bit as we have a static start value of 0/false.[11] So it makes sense to look at the distribution of d. To visualize this and to compare it with another browser I came up with this slight modification of the RNG that counts the flips and records them right inside the HTML page:

```
function flip_coin()
{i=0;n=0; then=millis()+1; while(millis()<=then) {n=!n;i++} return [n,i];}

function get_fair_bit()
{while(1) {a=flip_coin(); if(a[0]!=flip_coin()[0]) {return(a);}}}

function doit(){
  var i = 10000;
  while(i--){
    var d = document.getElementById(''target'');
    var content = document.createTextNode( get_fair_bit().toString() + ''\n'');
    d.appendChild(content);
  }
}
```

Loading the page in Chromium and Firefox and throwing them into gnuplot, we get:



We can see that the graph for Chromium has a lot more variance in the number of coin flip within a millisecond than that for Firefox. Although, strictly speaking, it might still be possible to get good randomness with poor variance if the few frequent values were to alternate just so due to some underlying scheduling magic, it seems reasonable to expect that the same magic would also increase the variance in the flip numbers.

We can also see, with the help of simple UNIX tools, that Chromium counts do not peak out to a certain value, unlike those of Firefox:

[11]The second coin flip in get_fair_bit complicates it a bit, but it cannot substantially improve the RNG's entropy if it lacks in the first place.

```
$ sort iter_Firefox|uniq -c|sort -n
...
    176 64683
    181 64671
    195 64673
    195 64684
    207 64717
    217 64672
    286 64718
    318 64721
    393 64719
    405 64720
```

vs.

```
$ sort iter_Chromium|uniq -c|sort -n
...
    15 45147
    15 45282
    16 44947
    16 45004
    16 45010
    16 45076
    16 45086
    17 45059
    17 45107
    19 45092
```

9.4 Closing words

In conclusion we see that in Firefox under stress Dan's RNG appears to fail at exactly the point he wanted to use as the main source of randomness. The tiny clock differentials used to gather the entropy are not given often enough in Firefox. There is still much room to stress this RNG implementation. Bonus rounds would include figuring exactly what the significant difference between the Firefox and Chromium JavaScript runtime is that causes this malfunction on Firefox. Also attacks on other JavaScript runtimes would be interesting to see. It might even be the case that this implementation has different results under different conditions with respect to CPU load.

A broader question occurs: The Dakarand RNG relies on what could be called a "code clock." It may be that in many kinds of environments stressed code clocks tend to go into phase with one another. *Driven by stress to seek comfort in each other's rhythms, their chance encounters may grow into something more close and intimate, grinding into periodic patterns. Which, of course, is bad for randomness. Can we learn to tell such environments from others, where periodization with stress doesn't happen?* –PML

MODEL CC-7 SPECIFICATIONS:

A. Recording Mode: Tape saturation binary. This is not an FSK or Home type recorder. **No voice capability. No Modem. (NRZ)**

B. Two channels (1) Clock, (2) Data. OR, Two data channels providing four (4) tracks on **the cassette. Can also be used for Bi-Phase, Manchester codes etc.**

C. Inputs: Two (2). Will accept TTY, TTL or RS 232 digital.

D. Outputs: Two (2). Board changeable from RS 232 to TTY or TTL digital.

E. Runs at 2400 baud or less. Synchronous or **Asynchronous. Runs at 4800 baud or less. Synchronous or Asynchronous. Runs at 3.1"/sec. Speed regulation ± .5%**

F. Compatability: Will interface any computer or terminal with a serial I/O. (Altair, Sphere, M6800, PDP8, LSI 11, IMSAI, etc.

G. Other Data: (110-220 V), (50-60 Hz); 3 Watts total; UL listed 955D; three wire line cord; on/off switch; audio, meter and light operation monitors. Remote control of motor optional. Four foot, seven conductor remoting cable provided. Uses high grade audio cassettes.

H. Warrantee: 90 days. All units tested at 300 and 2400 baud before shipment. Test cassette with 8080 software program included. This cassette was recorded and played back during quality control.

ALSO AVAILABLE: MODEL CC-7A with variable speed motor. Uses electronic speed control at 4"/sec. or less. **Regulation ± .2%**
Runs at 4800 baud Synchronous or Asynchronous without external circuitry.
Recommended for quantity users who exchange tapes. Comes with speed adjusting tape to set exact speed.

DIGITAL DATA RECORDER $149.95
FOR COMPUTER or TELETYPE USE
Any baud rate up to 4800

Uses the industry standard tape saturation method to beat all FSK systems ten to one. No modems or FSK decoders required. Loads 8K of memory in 17 seconds. This recorder, using high grade audio cassettes, enables you to back up your computer by loading and dumping programs and data fast as you go, thus enabling you to get by with less memory. Can be software controlled.

Model CC7 . . . $149.95
Model CC7A . . . $169.95

NATIONAL multiplex
CORPORATION

NEW – 8080 I/O BOARD with ROM. Permanent Relief from "Bootstrap Chafing"

This is our new "turnkey" board. Turn on your Altair or Imsai and go (No Bootstrapping). Controls one terminal (CRT or TTY) and one or two cassettes with all programs in ROM. Enables you to turn on and just type in what you want done. Loads, Dumps, Examines, Modifies from the keyboard in Hex. Loads Octal. For the cassettes, it is a fully software controlled Load and Dump at the touch of a key. Even loads MITS Basic. Ends "Bootstrap Chafe" forever. Uses 512 bytes of ROM, one UART for the terminal and one USART for the Cassettes. Our orders are backing up on this one. No. 2SIO (R)

Kit form $140. – Fully assembled and tested $170.00

Send Two Dollars for Cassette Operating and Maintenance Manual with Schematics and Software control data for 8080 and 6800. Includes Manual on I/O board above. Postpaid

Master Charge & BankAmericard accepted.

On orders for Recorders and Kits please add $2.00 for Shipping & Handling.
(N.J. Residents add 5% Sales Tax)

3474 Rand Avenue, Box 288
South Plainfield, New Jersey 07080
(201) 561-3600

This page intentionally left blank.
Draw your own damned picture.