Make Love! -- The lojikil way

part 0: the heart is a random thing

wtf loji?

- what *are* random numbers?
- what delineates *random*?
- what are random *number generators*?
- and how do they work?

A POX UPON CLAN CALLAHAN

- we saw MD5/SHA1 in client code (we thought)
- looked through impl
- realized we didn't *exactly* know how csPRNGs were implented...
- made this talk

random numbers

- statistically, numerically, and bit-level random
- multiple tests
- multiple sources [hardware, weather data, disk reads...]

... but really... random?

- entropy
- naturally occuring sources
- mixed pools

en·tro·py

/ˈentrəpē/

noun

1. PHYSICS
   a thermodynamic quantity representing the unavailability of a system's thermal energy for conversion into mechanical work, often interpreted as the degree of disorder or randomness in the system.
2. lack of order or predictability; gradual decline into disorder.
   "a marketplace where entropy reigns supreme"
   synonyms: deterioration, degeneration, crumbling, decline, degradation, decomposition, breaking down, collapse; More

RNGs

- multiple styles: psuedo-random, hardware, cryptographically secure pseudo-random...
- blame Callahan

RNGs (cont.)

- period: the number of values generated before the cycle begins again
- seed (or seed state): initial values for the RNG
- distribution, correlation, &c. all as normal

PRNG

- simple mathematical formula
- easy to implement (LCG ~~ 1 line of code)
- `X[n + 1] = (a * X[n] + c) mod m`
- `a`, `c`, `m` all have mathematical relationships

Linear Congruential Generator

```
uint64_t seed = 2592, a = 25214903917, c = 11;
uint64_t tmp = 0, mod = 0xDEADBEEF54;
uint64_t
lcg(uint64_t mod, uint64_t a, uint64_t c, uint64_t seed) {
    return ((a * seed) + c) % mod;
}
```

Pros/Cons

Pros:

- easy to understand
- fast
- easy to implement

Cons:

- small period
- those magic numbers matter
- easy to fuck up

enter: cryptographically secure PRNGs

- entropy pools
- mixed operations (mixed hashing & encrypting cycles)
- re-keying

Entropy pools

- think back to the hardware slide
- collect "entropy" from other sources

... so more data == better?

Here's an interesting example of an attack that can be carried out by this malicious source:

1. Generate a random r.
2. Try computing H(x,y,r).
3. If H(x,y,r) doesn't start with bits 0000, go back to step 1.
4. Output r as z.

from https://blog.cr.yp.to/20140205-entropy.html

arc4random

- `getentropy`
- fill buffer pool (per process)
- `arc4(buffer data)`
- *RANDOM!*

arc4random (cont.)

```
/* from OpenBSD */
u_int32_t
arc4random(void)
{
        u_int32_t val;

        _ARC4_LOCK();
        _rs_random_u32(&val);
        _ARC4_UNLOCK();
        return val;
}
```

Yarrow-ish

- "simple" algorithm
- can use any hash/block cipher

Yarrow-ish (cont.)

- `E()` := XTea (64-bit block cipher)
- `Pg` := n/3 ~~ 21
- Key := random (chosen by fair roll of dice)
- Repool every `Pg` block outputs
- Counter = (0 -> inf) mod key size
- output: `E(counter, key)`

XTea

```
void
encipher(uint32_t* v, uint32_t* k)
{
    uint32_t v0=v[0], v1=v[1], i;
    uint32_t sum=0, delta=0x9E3779B9;
    for(i=0; i < 64; i++)
    {
        v0 += (((v1 << 4) ^ (v1 >> 5)) + v1) ^
        (sum + k[sum & 3]);
        sum += delta;
        v1 += (((v0 << 4) ^ (v0 >> 5)) + v0) ^
        (sum + k[(sum>>11) & 3]);
    }
    v[0]=v0;
    v[1]=v1;
}
```

XTea

- broken, but whatever
- simple block cipher
- encipher == decipher (kinda)

Yarrow-ish (setup)

```
uint64_t
xtea_rand() {
    /* initial key from CRC32 spec */
    static uint32_t blocks_output = 0, key[4] =
    {0x00000000, 0x3b6e20c8, 0x76dc4190, 0x4db26158};
    static uint64_t ctr = 0;
    uint32_t vcounter[2] = {0};
    uint64_t res = 0;
    time_t t = 0;

    vcounter[0] = time(&t);
    vcounter[1] = t + ctr;
```

Yarrow-ish (rotation of key)

```
    if(blocks_output >= 21) {
        encipher(&vcounter[0], &key[0]);
        key[0] = vcounter[0];
        key[1] = vcounter[1];
        vcounter[0] = time(&t);
        vcounter[1] = ctr + 1;
        encipher(&vcounter[0], &key[0]);
        key[2] = vcounter[0];
        key[3] = vcounter[1];
        vcounter[0] = time(&t);
        vcounter[1] = ctr + 2;
        ctr += 2;
    }
```

Yarrow-ish encrypt

```
    encipher(&vcounter[0], &key[0]);
    res = (vcounter[0] << 32) + vcounter[1];
    blocks_output += 1;
    ctr += 1;
    return res;
}
```

Testing

- "rigorous" testing: 20k samples from each
- "rigorous" testing: Burp Sequencer

Results!

aka "who cares?" (Russian Accent)

- all three suck
- but one sucks less than the others!

LCG

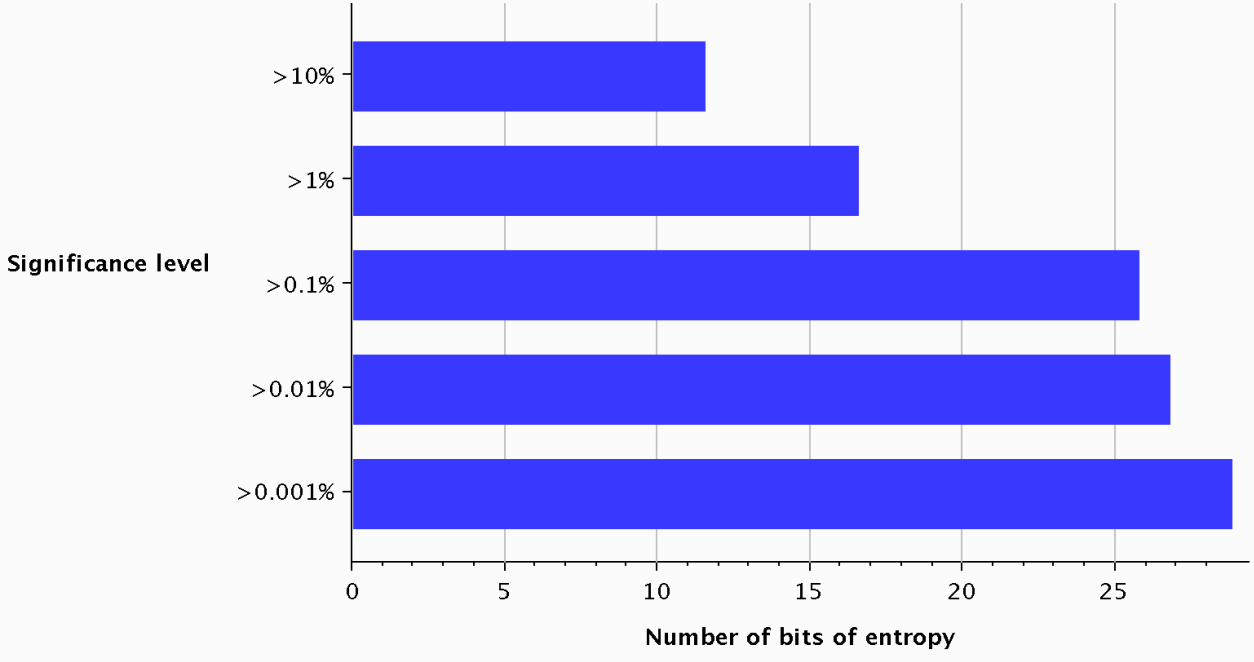
~ 17 bits of entropy

Overall result

The overall quality of randomness within the sample is estimated to be: very poor.
At a significance level of 1%, the amount of effective entropy is estimated to be: 17 bits.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.



arc4random

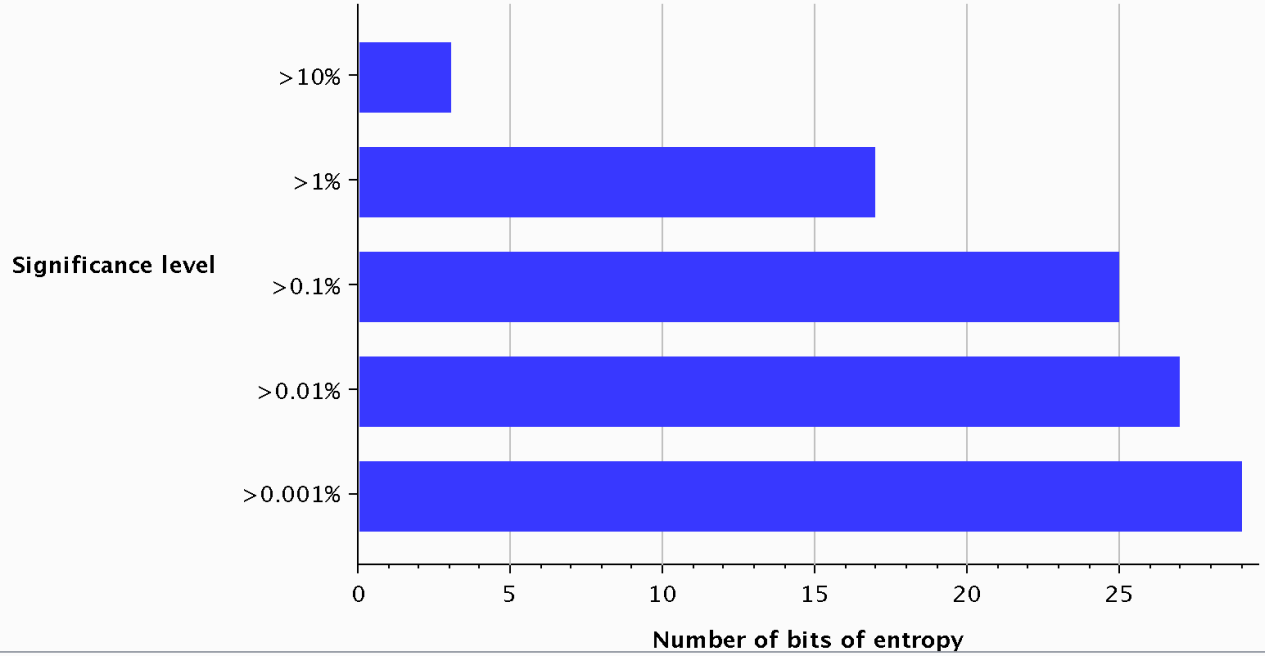
~ 17 bits of entropy

Overall result

The overall quality of randomness within the sample is estimated to be: very poor.
At a significance level of 1%, the amount of effective entropy is estimated to be: 17 bits.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.



Yarrow-ish

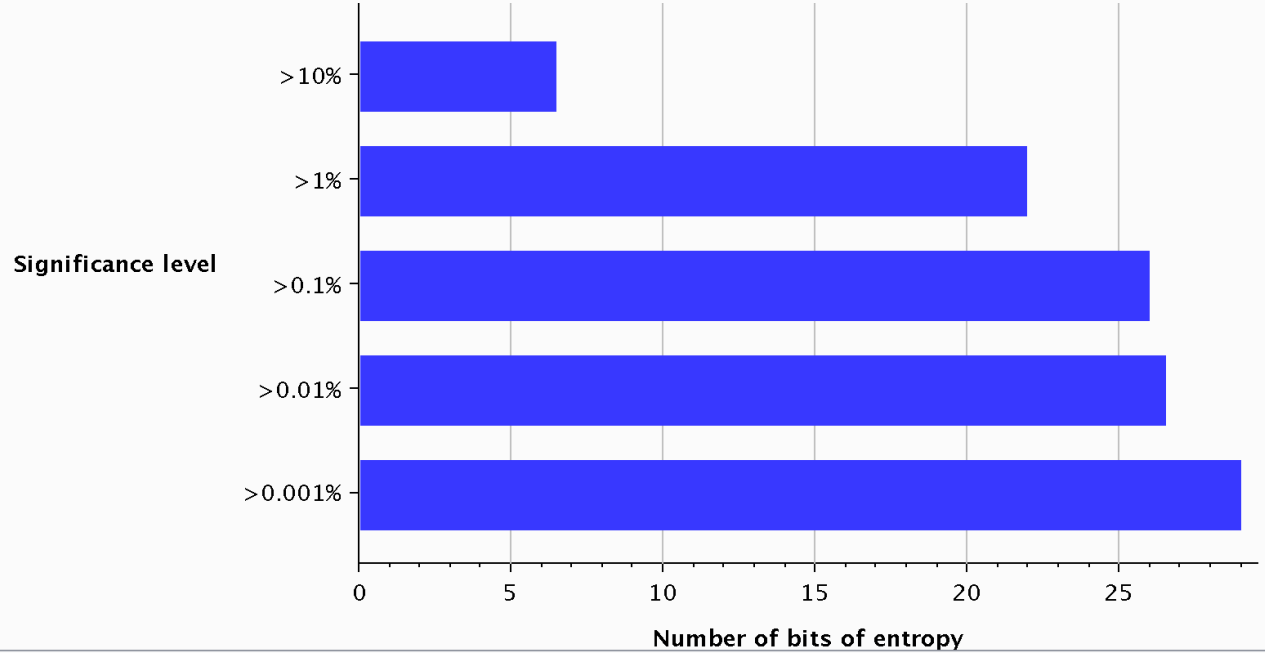
~ 22 bits of entropy!

Overall result

The overall quality of randomness within the sample is estimated to be: very poor.
At a significance level of 1%, the amount of effective entropy is estimated to be: 22 bits.

Effective Entropy

The chart shows the number of bits of effective entropy at each significance level, based on all tests. Each significance level defines a minimum probability of the observed results occurring if the sample is randomly generated. When the probability of the observed results occurring falls below this level, the hypothesis that the sample is randomly generated is rejected. Using a lower significance level means that stronger evidence is required to reject the hypothesis that the sample is random, and so increases the chance that non-random data will be treated as random.



Summary

- csPRNG == entropy pools + cipher
- in ~60 lines, created a less shitty sys than both LCG and arc4random
- ... but still pretty shitty

Thanks!

- Questions?