Generación de Números Aleatorios Para Criptografía

Marco Vanotti

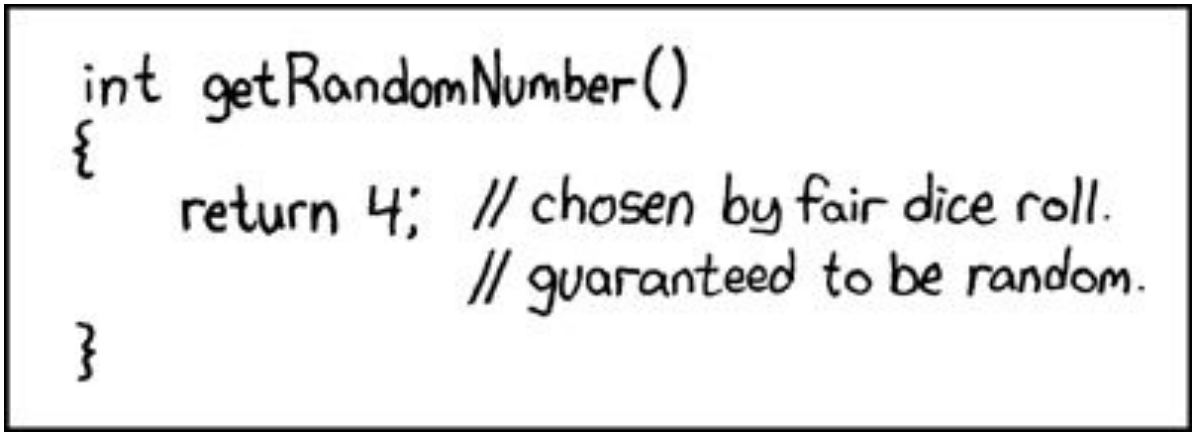
Randomness… ¿Para qué?

- Address Space Layout Randomization (ASLR)
- HTTPS (TLS)
- Generación de Claves Criptográficas
- Salts, nonces

CSPRNG vs PRNG (/dev/urandom vs rand())

- PRNG sólo garantiza:
  - Los datos son uniformes.
  - Pasen test estadisticos.
- CSPRNG tiene que satisfacer:
  - Next-Bit Test.
  - Backwards Resistance.

Nuestro propio CSPRNG



Fuente: xkcd.com/221

Nuestro propio CSPRNG

- Vamos a necesitar ENTROPÍA
  - Algo que el atacante no conozca.
  - Difícil de medir.
  - Se puede obtener mediante mediciones de tiempo de eventos
    - Tiempo entre paquetes de red, eventos del disco, presión de teclas
    - Tiempos de acceso a memoria o ejecución de instrucciones (jitterentropy)
  - O mediante hardware especializado
    - RDRAND/RDSEED (Intel)
    - Trusted Platform Module (TPM)
    - Osciladores
    - Lamparas de lava

- Vamos a querer CONDICIONAR esa entropía.
- Vamos a querer ESTIRAR esa entropía.

Preliminares: Hash Criptográfico

- Mapean datos de tamaño arbitrario a strings de tamaño fijo.
  - Ej: SHA256 da como output 256 bits.
- Funciones de una vía
  - Dado x es fácil encontrar f(x)
  - Dado f(x) es muy difícil encontrar x.
- Es difícil de encontrar x != y, tales que h(x) == h(y)
  - No son collision-free
- **Queremos que se comporte como un mapeo random entre el input y el espacio de output.**

Preliminares: Cifrados de Bloques

- Trabaja con una clave K, y un bloque de N bits.
  - Mapea el bloque de N bits en otro bloque de N bits.

- Si elegimos K de forma aleatoria, esperamos que el mapeo sea indistinguible a una permutación aleatoria, elegida uniformemente del espacio de salida.

- Dado un algoritmo de cifrado por bloque se puede usar para generar una secuencia aleatoria de gran longitud.

  E(K, Nonce || counter)

  Para leer más: Cryptography Engineering
  A Graduate Course in Applied Cryptography

Nuestro Propio CSPRNG (basico, roto)

- Acumular entropía en un pool.
  - POOL = HASH(POOL || ENTROPIA_NUEVA)

- Usar nuestro pool de entropía como clave para un cifrado simétrico.
  - K = POOL
  - NONCE = NONCE + 1
  - Generamos una secuencia aleatoria usando K como clave y NONCE.
- Ejemplo:
  - HASH: SHA256
  - Cifrado: CHACHA20
    - Key: 256 bits
    - Nonce: 96 bits.
    - Encripta hasta 2^32 bloques.
    - cr.yp.to/chacha.html

¡PROBLEMAS!

- Entropy Depletion
  - Si nos piden números aleatorios antes de tener toda la entropía necesaria, pueden adivinar el estado interno.

- ¿Next Bit Test?
  - En general sí, pero una secuencia MUY larga no pasa el next bit set (cada bloque es distinto al anterior).

- ¿Backwards Resistance?
  - Depende de qué tan frecuentemente agreguemos nueva entropía.

- **¿Early Boot?**

Algunas Mejoras

- Acumular entropía antes de agregarla al CSPRNG.
  - Bloquear hasta que no tengamos suficiente entropía inicial.
  - Luego agregar la entropía toda de golpe, no de a poco.
  - ¿Cuando tenemos suficiente entropía?

- Limitar tamaño del output.
  - En general no hace falta más de 1MB por llamada.

- Key-Erasure…
  - Luego de dar números aleatorios podemos cambiar la clave del csprng con datos nuevos.
  - Hay que tener cuidado para hacerlo de forma eficiente
    - ¿Qué pasa si lo hacemos cada vez que nos piden 1 byte de datos random?
  - blog.cr.yp.to/20170723-random.html

CSPRNG: Fortuna

schneier.com/academic/fortuna

- Resuelve el problema de estimar la entropía
  - 32 pools, cada uno con el "doble" del anterior.
- Lento
  - Tarda más en llegar a la cantidad necesaria de entropía.
- Resistente a entropy depletion
  - Si un atacante adivina el estado interno, eventualmente se va a recuperar.

En sistemas operativos...

- Linux usa chacha20
  - drivers/char/random.c
- openBSD y freeBSD usan chacha20 para arc4random
  - (openbsd) src/lib/libc/crypt/arc4random.c
  - (freebsd) source/libkern/arc4random.c
- freeBSD tiene una implementación de Fortuna para /dev/urandom
  - dev/random/fortuna.c
- op_tee (TrustZone) usa fortuna
  - core/crypto/rng_fortuna.c

/dev/random vs /dev/urandom

Usar /dev/urandom!

www.2uo.de/myths-about-urandom

sockpuppet.org/blog/2014/02/25/safely-generate-random-numbers

Problemas Históricos

- Debian Fiasco
  - Desarrollador comentó línea que agregaba entropía usando datos sin inicializar.
- SecureRandom Bug
  - Reusaba NONCEs, miles de dólares perdidos en Bitcoin Wallets para Android.
- Backdoor en Dual_EC_DRBG
- Certificados con Claves repetidas
  - Por generar claves en early boot sin tener suficiente entropía.

Links de cosas habladas

Discusión sobre bloquear si no hay entropía en LKML

Entropy Attacks!

Recommendations for Randomness in Operating Systems (Paper)

Side Channel Attacks via Deep Learning (DEFCON 27)

BLACKSWAN: side channel attack on CTR_DRBG

(No Mencionado) Blum Blum Shub