Univerza v Ljubljani

Fakulteta za računalništvo in informatiko

Žiga Mahkovec

Analiza generatorjev psevdo-naključnih števil v operacijskem sistemu Linux

Seminarska naloga

Ljubljana, september 2004

Kazalo

Povzetek

Varnost večine kriptografskih sistemov je v veliki meri odvisna od varnosti izbire naključnih števil (npr. DES ključev ali RSA praštevil). Ta morajo ustrezati določenim statističnim lastnostim in predvsem biti nepredvidljiva. Pri programskem generiranju si moramo v odsotnosti posebne strojne opreme pomagati z različnimi viri naključnosti in algoritmi za generiranje psevdo-naključnih števil (PRNG). Jedro operacijskega sistema Linux vsebuje kriptografsko krepak gonilnik za PRNG. Sistem je prestal več statističnih testov, poleg tega pa ustreza kriterijem dobrih sistemov za generiranje naključnih števil. Čeprav niso znani praktični napadi nanj, pa so možne določene izboljšave, ki bi še povečale varnost in uporabnost generatorja.

V uvodnem poglavju je podana definicija naključnih števil, generatorjev naključnih števil in statističnih testov. V drugem poglavju je opisana zasnova generatorjev psevdo naključnih števil. Tretje poglavje podrobno predstavi generator `/dev/random`, v četrtem pa je podana analiza generatorja. V petem poglavju generator primerjamo z drugimi generatorji.

Ključne besede

Kriptografija, naključna števila, PRNG, `/dev/random`, statistični testi, Linux

Abstract

The security of many cryptographic systems depends upon the generation of secure random numbers (examples include DES keys and RSA primes). Good random number generators must not be predictable and should pass appropriate statistical tests. In the lack of special hardware, software generators must be used to generate random numbers, using various external sources of randomness. The Linux kernel contains a cryptographically strong pseudo-random number generator (PRNG). The generator has passed several statistical tests and follows the guidelines for designing good random generators. There are no known attacks on the generator; however, certain improvements are possible in the areas of security and usability.

In the introduction we define random numbers, random number generators and statistical tests. Chapter 2 examines the design of pseudo-random number generators. The design of `/dev/random` is presented in detail in chapter 3 and analyzed in chapter 4. Chapter 5 offers comparison with other generators.

Keywords

Cryptography, radnom numbers, PRNG, `/dev/random`, statistical tests, Linux

1 Uvod

1.1 Naključna števila

Pri obravnavi kriptografskih sistemov pogosto naletimo na pojem naključnih vrednosti, na primer enkratni ščit, skrivni ključ pri šifriranju DES, praštevili p in q pri RSA algoritmu, itd. Pri vseh teh primerih morajo biti vrednosti "naključne", tako da onemogočijo napadalcu uspešen napad z metodo požrešnega iskanja ključev. Naključna števila so zato pomemben del kriptografskih sistemov.

Naključne vrednosti lahko dobimo s pomočjo fizičnih pojavov, kot so met kovanca, radioaktivni razpad, termični šum ipd. Vendar pa je v praksi takšno pridobivanje prepočasno ali predrago, zato se v programski opremi pogosto poslužujemo generatorjev psevdo-naključnih števil (angl. pseudo-random number generator, PRNG).V seminarski nalogi bomo analizirali generator naključnih števil `/dev/random` [10], ki je sestavni del jedra operacijskega sistema Linux.

1.2 RNG, PRNG in CSPRNG

Ločimo več razredov generatorjev naključnih števil:

- *generator naključnih števil (RNG)* je algoritem, ki generira zaporedje statistično neodvisnih in enakomerno porazdeljenih naključnih števil.
- *generator psevdo-naključnih števil (PRNG)* je deterministične [1] algoritem, ki iz naključnega zaporedja dolžine k tvori zaporedje dolžine ℓ, ki je "navidez" naključno:

  $$f : (\mathbb{Z}_2)^k \to (\mathbb{Z}_2)^\ell,\ \ell \gg k$$

  $(\mathbb{Z}_2)^k$: seme, $(\mathbb{Z}_2)^\ell$: psevdo-naključno zaporedje

  Namen PRNG generatorjev je, da kratko zaporedje naključnih števil razširijo v zaporedje, ki ga napadalec ne more učinkovito razločiti od naključnega zaporedja. Enostaven primer takega generatorja je linearni kongruenčni generator:

[1] PRNG je deterministične, ker bo za isto seme vedno generiral isto naključno zaporedje

$$s_i = (as_{i-1} + b) \bmod M$$

Števila a, b in M so parametri generatorja, s_0 pa je seme. Linearni kongruenčni generator proizvaja števila, ki so enakomerno porazdeljena, vendar so predvidljiva — iz delnega zaporedja števil lahko ugotovimo preostanek zaporedja tudi brez poznavanja parametrov.

- *kriptografsko krepak generator psevdo-naključnih števil (CSPRNG)* je algoritem PRNG, za katerega velja, da ni predvidljiv. Formalno to pomeni, da prestane test naslednjega bita, torej ne obstaja algoritem v polinomskem času, ki bi iz zaporedja dolžine n napovedal $n + 1$. element zaporedja z verjetnostjo, večjo od $\frac{1}{2}$.

  Test naslednjega bita je univerzalen – zaporedje ga prestane natanko tedaj, ko prestane vse polinomske statistične teste (to je algoritme, ki lahko v polinomskem času razločijo zaporedje od naključnega z verjetnostjo, večjo od $\frac{1}{2}$).

  Primeri CSPRNG generatorjev so Blum Blum Shub, PGP 5.x generator, Yarrow in pa `/dev/random`, ki ga bomo analizirali v seminarski nalogi.

1.3 Statistični testi

Kvaliteto generatorjev psevdo-naključnih števil preverjamo s statističnimi testi. Za nek generator ne moremo podati matematičnega dokaza, da res generira naključna števila. Lahko pa s pomočjo statističnih testov odkrijemo določene pomanjkljivosti.

Pri statističnem testiranju najprej vzamemo vzorec zaporedja, ki ga generira PRNG. Zaporedje potem pošljemo skozi več testov, ki preverjajo, če le-to ustreza nekemu atributu naključnega zaporedja. Primer: za binarno zaporedje velja, da mora imeti približno enako število ničel in enic.

Rezultat vsakega statističnega testa je verjetnostna spremenljivka p, katere vrednost mora biti na intervalu $[0, 1)$. Pri tem vrednosti, ki so relativno blizu 0 ali 1 ne pomenijo, da je generator prestal statistični test z manjšim intervalom zaupanja. Za vsak test je namreč določena pričakovana verjetnostna porazdelitev (npr. normalna porazdelitev), ta pa je pogosto zgolj približek. V primeru, da zaporedje katerega od testov ne prestane (torej verjetnostna spremenljivka pade iz intervala $[0, 1)$, ga ponavadi zavržemo.

Osnovni testi, ki jih lahko izvajamo na binarnih zaporedjih števil, so:

- frekvenčni (monobit) test: preverja, če je število ničel in enic v zaporedju približno enako,
- serijski (dvobitni) test: preverja pogostost podzaporedij 00, 01, 10 in 11,
- poker test: preverja porazdelitev podzaporedij,
- test ponavljajočih podzaporedij: preverja število ponavljajočih podzaporedij (samih ničel ali enic) različnih dolžin,
- avtokorelacijski test: preverja korelacijo med testiranim zaporedjem in zaporedjem, ki ga dobimo iz testiranega zaporedja z zamikanjem.

Obstajajo pa tudi programski paketi, ki vsebujejo baterije različnih, tudi bolj kompleksnih testov. Trije paketi, ki smo jih uporabili tudi za testiranje generatorja `/dev/random`, so ENT, DIEHARD in nabor NIST 800-22.

1.3.1 ENT

Programski paket ENT (*A Pseudorandom Number Sequence Test Program* [12]) izvaja različne enostavne teste nad binarnimi naključnimi vrednostmi, shranjenimi v datoteki. Računa entropijo, skrčenje z optimalno kompresijo, χ^2 distribucijo, povprečno vrednost bajtnih števil, Monte Carlo vrednost števila π in serijski korelacijski koeficient.

1.3.2 DIEHARD

DIEHARD [7] je zelo znan nabor testov, ki se pogosto uporablja za testiranje generatorjev naključnih števil. Avtor programskega paketa je George Marsaglia, napisan pa je v programskem jeziku `C` (prenos iz jezika `Fortran`).

DIEHARD vsebuje 17 tipov testov, med ostalim GCD test, razdalje med naključno izbranimi rojstnimi datumi, red binarnih matrik 31×31 in 32×32, število enic v binarnem zaporedju, igranje iger s kockami, itd.

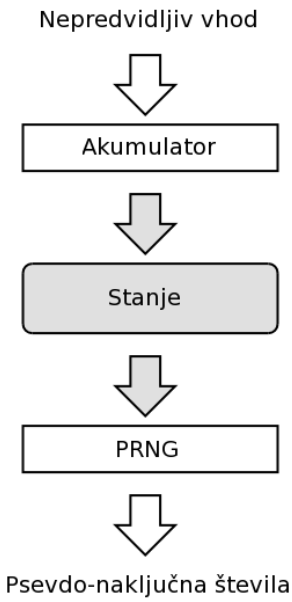
1.3.3 NIST 800-22

Nabor testov NIST 800-22 je izdala organizacija NIST (*National Institute of Standards and Technology*) [9]. Nabor vsebuje 16 testov, med ostalim:

- monobit test,
- test ponavljajočih zaporedij,
- redi binarnih matrik,
- iskanje DFT ekstremov,
- Maurerjev univerzalni test,
- LZW kompresija.

2 Generatorji psevdo-naključnih števil

Programski generatorji naključnih števil so najbolj pogosto sestavljeni iz dveh komponent: zaloge naključnosti s pripadajočo mešalno funkcijo (generator sam) in pa akumulatorjem naključnosti, ki pridobiva naključna števila in jih dodaja v zalogo [3].



Slika 2.1: Shema PRNG

Akumulator skrbi za zalogo naključnih števil, generator pa to zaporedje razširi z enim od PRNG algoritmov. V določenih primerih razširjanje sploh ni potrebno (npr. če akumulator dovolj hitro pridobiva naključne vrednosti).

2.1 Vhodni podatki, stanje in izhod generatorja

Kot vhodne podatke za naključna števila lahko uporablja akumulator različne vire naključnosti. Ločimo jih na strojne (npr. strojni generator Intel RNG [4], zvočna

kartica) in na programske (vhod tipkovnice, miške, zakasnitve trdega diska, itd.). Zaželjeno je, da akumulator pridobiva podatke iz večjega števila virov, saj je tako onemogočen vpliv napadalca na notranje stanje.

Notranje stanje generatorja je zaščiteno polje naključnih števil, iz katerih generator pridobiva izhod.

2.2 Kriteriji za načrtovanje varnih PRNG

Pri načrtovanju varnih generatorjev psevdo-naključnih števil se držimo določenih načel [3, 2, 11]:

- Generator mora biti odporen na analizo notranjega stanja. Če napadalcu uspe pridobiti informacije o delu notranjega stanja, ne sme biti biti sposoben rekonstrukcije celotnega notranjega stanja.
- Odporen mora biti tudi na napade z izbranim vhodom. Če napadalec lahko vpliva na vhodne podatke, ne sme imeti možnosti vplivanja na notranje stanje.
- Generator mora biti odporen na analizo izhodnih podatkov. Napadalec ne sme imeti sposobnosti iz izhodnih podatkov sklepati o notranjem stanju.
- Notranje stanje mora biti dobro zaščiteno. Odporno mora biti pred tehnikami, kot so preiskovanje datotek virtualnega pomnilnika. Notranje stanje tudi nikoli ne sme biti vidno izven sistema.
- Pri implementaciji generatorja moramo eksplicitno označiti in dokumentirati vse uporabljane kriptografske metode — tako omogočimo lažje preverjanje izvorne kode.
- Generator mora vzdrževati količino entropije, ki se nahaja v notranjem stanju. Uporabnik mora imeti informacijo o meri naključnosti podatkov, ki jih vrača generator.
- Izhodni podatki generatorja morajo biti podvrženi statističnim testom, tako da se sproti odkrivajo napake. V primeru napake na strojnem generatorju lahko ta na primer začne vračati povsem deterministične vrednosti.

2.3 Tipi napadov

Napade na generatorje psevdo-naključnih števil lahko razdelimo v tri skupine [6].

2.3.1 Direktni kriptoanalitični napad

Pri direktnem kriptoanalitičnem napadu je napadalec sposoben razločiti med izhodom generatorja PRNG in naključnim izhodom. Takšen napad je v teoriji možen za vsak generator; izjema je na primer PRNG, ki se uporablja samo za generiranje ključev za 3-DES – v tem primeru izhod generatorja nikoli ni viden, torej tudi direktni kriptoanalitični napad ni možen.

2.3.2 Napad z vhodom

Pri napadu z izbranim vhodom napadalec izkoristi poznavanje ali nadzor nad vhodom v generator za kriptoanalizo generatorja — torej za razločevanje med izhodom generatorja in naključnimi vrednostmi. Napade lahko nadalje razvrstimo v napade z znanim vhodom, napade s ponovljenim vhodom in napade z izbranim vhodom.

Napad z izbranim vhodom je na primer možen, kadar generator kot vir uporablja zakasnitve trdega diska, ki pa je priklopljen preko mreže. Napadalec lahko prestreže mrežno povezavo in manipulira z vhodnimi podatki.

2.3.3 Napad na notranje stanje

O takem napadu govorimo, kadar napadalec uspe priti do notranjega stanja (npr. zaradi predora zaščite ali kriptoanalitskega uspeha). Napad uspe, če napadalec na osnovi notranjega stanja lahko sklepa o preteklem izhodu generatorja ali predvidi nadaljnji izhod.

Takšni napadi so pogosti, kadar generator prične delovati v začetnem stanju z majhno količino entropije.

3 Zasnova generatorja /dev/random

Za študij zasnove generatorja `/dev/random` je najbolje analizirati izvorno kodo gonilnika. Ta je prosto dostopna kot del izvorne kode jedra operacijskega sistema Linux (dostopne na naslovu `http://www.kernel.org`). Gonilnik `/dev/random` se nahaja v datoteki `/usr/src/linux/drivers/char/random.c`. Nekaj o implementaciji pa izvemo tudi iz različnih diskusij v novičarski skupini `linux-kernel` (`http://lkml.org`).

Avtor gonilnika je Theodore Ts'o, zadnja različica (1.98) pa je nastala leta 1999. Obsega okrog 2500 vrstic kode v programskem jeziku `C`.

3.1 Vhodni podatki

`/dev/random` uporablja več virov entropije: čas vnosa s tipkovnice, vnosa z miške, dostopni časi do trdega diska in pa časi določenih prekinitev (IRQ). Za vsak vhodni podatek se oceni entropija, nato pa se ta s CRC mešalno funkcijo doda v notranje stanje. Mešalna funkcija je GFSR (angl. generalized feedback shift register), ki temelji na linearni rekurzivni enačbi:

$$x_{\ell+n} := x_{\ell+m} \oplus x_\ell \quad (\ell = 0, 1, \ldots),$$

kjer je x_ℓ binarno zaporedje.

Mešalna funkcija GFSR sicer ni kriptografsko močna, vendar je hitra. Ker se entropija pridobiva med prekinitvami na nivoju jedra operacijskega sistema, ima hitrost izvajanja velik pomen.

Pri vnosu s tipkovnice se v mešalno funkcijo vmeša čas dogodka in pa koda pritisnjene tipke. Pri vnosu z miške se poleg časa dodajo koordinate.

Dostopni časi do trdega diska so naključen pojav zaradi zračne turbulence v ohišjih diskov [1]. Ta vpliva na drobne (in predvsem naključne) razlike v dostopnih časih. S tem virom lahko v povprečju pridobimo 100 bitov entropije na minuto.

Generator omogoča tudi ročno dodajanje naključnih podatkov, s pisanjem v datoteko `/dev/random`. Tako lahko uporabnik sam doda druge vire naključnosti. Vendar pa se v tem primeru mera entropije, ki jo vzdržuje generator, ne povečuje — tako je preprečen napad z izbranim vhodom.

3.2 Ocena entropije

Generator vseskozi vodi oceno entropije [1], ki se nahaja v notranjem stanju. Ocene se izvršujejo na osnovi časov dogodkov, torej vnosa s tipkovnice, miške in prekinitev. Pri tem se upošteva razlika med časom nekega dogodka in časom zadnjega dogodka istega razreda.

Vsebina dogodka (npr. koda tipke, koordinate miške ali tip prekinitve) ne vpliva na entropijo. Ta se sicer skupaj s časom doda v notranje stanje, a ne poveča entropije.

Za oceno entropije se uporablja algoritem delta-delta-3 — upoštevajo se časovne razlike prve, druge in tretje stopnje. Razlika prve stopnje je razlika v času med trenutnim dogodkom in zadnjim dogodkom istega razreda. Razlika druge stopnje je razlika med časovnima razlikama prve stopnje; razlika tretje stopnje je razlika med časovnima razlikama druge stopnje. Pri tem se računa z absolutnimi vrednostmi časovnih razlik. Poleg tega se upošteva samo 12 bitov časovne razlike, najmanj pomemben bit pa se zavrže.

Tabela 3.1 prikazuje primer:

Čas dogodka	1004		1012		1024		1025		1030		1041
Razlika 1.st		8		12		1		5		11	
Razlika 2.st			4		11		4		6		
Razlika 3.st				7		7		2			

Tabela 3.1: Časovne razlike delta-delta-3

Za oceno entropije se vzame logaritem z osnovo 2 najmanjše od treh razlik. Primer za zadnji dogodek, ob času 1041: razlika prve stopnje je 11, druge stopnje 6 in tretje stopnje 2. Ocena entropije je $\log_2 2$, oziroma 1 bit.

3.3 Notranje stanje

Notranje stanje generatorja sestoji iz 512-bajtnega polja. Slika 3.1 prikazuje strukturo, kot je zapisana v izvorni kodi. Poleg osnovnega polja se uporablja še eno, sekundarno polje s 128 bajti. To se uporablja za reinicializacijo notranjega stanja (angl. *catastrophic reseed*).

Generator poleg naključnih vrednosti vzdržuje tudi mero entropije teh podatkov, to je vsoto entropij posameznih dogodkov (izračunanih z delta-delta-3 algorit-

[1] Entropija je mera nedoločenosti ali neurejenosti sistema. Večja kot je entropija, več informacije potrebujemo, da bi ga opisali.

mom). Pri uporabi vmesnika /dev/random generator vrne samo toliko naključnih števil, kolikor je entropije v notranjem stanju; če je teh manj, kot jih uporabnik zahteva, počaka, da akumulator pridobi dovolj nove entropije.

```
#define DEFAULT_POOL_SIZE 512
#define SECONDARY_POOL_SIZE 128

struct entropy_store {
        /* mostly-read data: */
        struct poolinfo poolinfo;
        __u32           *pool;

        /* read-write data: */
        spinlock_t lock ____cacheline_aligned_in_smp;
        unsigned        add_ptr;
        int             entropy_count;
        int             input_rotate;
};

/* The default global store */
static struct entropy_store *random_state;
/* secondary store */
static struct entropy_store *sec_random_state;
```

Slika 3.1: Struktura za notranje stanje

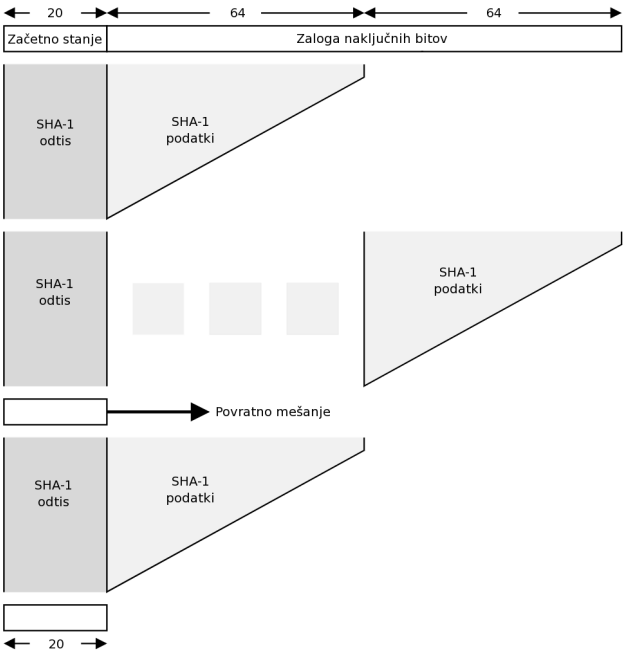
Notranje stanje je shranjeno v pomnilniku, zato za njegovo zaščito skrbi operacijski sistem. Napadalcu je vidno le v primeru, da le-ta pridobi nadzor nad strežnikom, kjer je generator. Vendar pa ima v tem primeru napadalec že popolno kontrolo nad generatorjem.

3.4 Izhodni podatki

Ko uporabnik zahteva naključna števila od generatorja, je postopek naslednji (slika 3.2):

1. izračuna se zgoščevalna funkcija SHA-1 za prva dva 64-bajtna bloka v notranjem stanju,
2. dobljena vrednost se vrne v notranje stanje z isto mešalno funkcijo, kot se uporablja za vhodne podatke (GFSR),
3. prvih 64 bajtov notranjega stanja se še enkrat pošlje čez zgoščevanje SHA-1, zato da se prikrijejo podatki, ki so se dodali v notranje stanje,
4. rezultat je končni 20-bajtni SHA-1 odtis; v primeru, da uporabnik zahteva manj podatkov, se rezultat poreže; če jih zahteva več, se postopek povratnega mešanja ponavlja;

Zgoščevalna funkcija SHA-1, ki se uporablja znotraj gonilnika je modificiran algoritem SHA-1. Del, ki poravna bite in doda dolžino podatkov, ni implementiran; v poštev pride zgolj kompresijski del zgoščevalne funkcije.



Slika 3.2: Izhodna funkcija `/dev/random`

3.5 Vmesnika `/dev/random` in `/dev/urandom`

Generator daje na voljo dva vmesnika: `/dev/random` in `/dev/urandom` (slika 3.3).

```
$ ls -l /dev/*random
crw-r--r--  1 root root 1, 8 Feb 23  2004 /dev/random
crw-r--r--  1 root root 1, 9 Sep  7 16:09 /dev/urandom
```

Slika 3.3: Vmesnika gonilnika `/dev/random`

Razlikujeta se v delovanju v primeru, ko entropije v notranjem stanju zmanjka. `/dev/random` v tem primeru čaka na nove vhodne podatke in pri tem blokira izhod. `/dev/urandom` pa nasprotno ponavlja postopek povratnega mešanja, dokler ni na voljo dovolj naključnih števil.

`/dev/urandom` torej deluje kot generator psevdo-naključnih števil. Varnost generatorja je v tem primeru odvisna od varnosti algoritma SHA-1, za katerega pa, vsaj v času pisanja seminarske naloge, še ni učinkovitega napada. Zato je za večino aplikacij uporaba tega vmesnika dovolj varna; prednost uporabe je predvsem v tem, da je generiranje naključnih števil veliko hitrejše.

Z vmesnikom `/dev/random` pa gonilnik generira resnično naključna števila. Tudi v primeru razbitja SHA-1 napadalec brez znanja o vhodnih podatkih ne more predvideti naključnih števil, ki jih bo generator vračal v prihodnosti.

4 Analiza generatorja `/dev/random`

Generator `/dev/random` smo testirali z vidika naključnosti in varnosti:

1. izhod generatorja in vhodne podatke smo testirali z različnimi statističnimi testi,
2. preverili smo zadovoljevanje kriterijev za razvoj varnih generatorjev psevdonaključnih števil,
3. analizirali smo izvorno kodo in iskali napake in možne izboljšave,
4. preverili smo teoretične možnosti za napad na generator,
5. izpostavili smo slabosti generatorja;

4.1 Statistični testi

4.1.1 Testiranje izhoda

Za potrebe statističnega testiranja izhoda smo generirali dve datoteki z vmesnikom `/dev/urandom`, velikosti 250000B za ENT in NIST teste ter 11MB za DIEHARD teste.

ENT

Slika 4.1 prikazuje rezultat statističnih testov ENT.

```
Entropy = 7.999273 bits per byte.

Optimum compression would reduce the size
of this 250000 byte file by 0 percent.

Chi square distribution for 250000 samples is 251.69, and
randomly would exceed this value 50.00 percent of the times.

Arithmetic mean value of data bytes is 127.3372 (127.5 = random).
Monte Carlo value for Pi is 3.144818317 (error 0.10 percent).
Serial correlation coefficient is 0.002713
(totally uncorrelated = 0.0).
```

Slika 4.1: Rezultati ENT statističnih testov

Generator je prestal vse teste. Povprečna entropija naključnih vrednost je 7.999273 bitov na bajt. Podatkov se praktično ne da kompresirati. Povprečna vrednost bajtnih števil je blizu sredini (127.5), napaka pri računanju števila π z metodo Monte Carlo pa je 0.10 odstotka.

Za primerjavo: slika, kompresirana z algoritmom JPEG, ima povprečno entropijo 7.980627 bitov na znak, povprečna vrednost bajta je 125.93, napaka pri računanju vrednosti števila π pa je 0.90.

DIEHARD

Slika 4.2 prikazuje rezultate statističnih testov DIEHARD v skrajšani obliki. Generator je prestal vseh 229 testov. Končna vrednost spremenljivke p je 0.324874 in je na zahtevanem intervalu $[0, 1)$ [1].

```
        RESULTS OF BIRTHDAY SPACINGS FOR urandom.out
 (no_bdays=1024, no_days/yr=2^24, lambda=16.00, sample size=500)
         Chisquare degrees of freedom: 17
 ------------------------------------------------------------------
        Rank test for binary matrices (31x31) for urandom.out
        chi-square = 3.439 with df = 3;  p-value = 0.671
 ------------------------------------------------------------------
        Rank test for binary matrices (32x32) for urandom.out
        chi-square = 5.257 with df = 3;  p-value = 0.846
 ------------------------------------------------------------------
        Rank test for binary matrices (6x8) for urandom.out
 The KS test for those 25 supposed UNI's yields p = 0.064268
 ------------------------------------------------------------------
        Test result COUNT-THE-1's in bytes for urandom.out
   (Degrees of freedom: 5^4-5^3=2500; sample size: 2560000)
               chisquare       z-score         p-value
               2610.78          1.567          0.941396
 ------------------------------------------------------------------
       Results for the MINIMUM DISTANCE test for urandom.out
 The KS test for those 10 p-values: 0.464570
 ------------------------------------------------------------------
           The 3DSPHERES test for urandom.out
         p-value for KS test on those 20 p-values: 0.183674
 ------------------------------------------------------------------
                RESULTS OF SQUEEZE TEST FOR urandom.out
         Chi-square with 42 degrees of freedom:33.214351
         z-score=-0.958593, p-value=0.168319
 ------------------------------------------------------------------
                Results of the OSUM test for urandom.out
         p-value for 10 kstests on 100 sums: 0.050257
 ------------------------------------------------------------------
Test for lengths of runs up and runs down, 100,000 each for
urandom.out:
 Number of rngs required: 688430, p-value:  0.512
 ------------------------------------------------------------------
        RESULTS OF CRAPS TEST for urandom.out
   p-value for no. of wins: 0.408292
   p-value for throws/game: 0.432762
 ------------------------------------------------------------------
        RESULTS OF CRAPS TEST2 for urandom.out
   p-value for no. of wins: 0.855572
   p-value for throws/game: 0.986610
 ------------------------------------------------------------------
        ***** TEST SUMMARY *****
Overall p-value after applying KStest on 229 p-values = 0.324874
```

Slika 4.2: Rezultati DIEHARD statističnih testov

[1] Ker so nekatere pričakovane verjetnostne porazdelitve rezultatov statističnih testov zgolj približki, se vsi rezultati znotraj tega intervala smatrajo kot uspešni.

NIST 800-22

Generator je prestal 185 od 188 NIST testov. Neuspešnih je bilo zgolj 8 različic testa z iskanjem pogostosti določenih vzorcev (angl. *Template Matching Test*). Pri teh testih se v naključnem zaporedju išče pogostost različnih podzaporedij fiksne dolžine. Ker so ti testi zelo strogi, je to še vedno zelo dober rezultat.

4.1.2 Testiranje vhoda

Problem pri testiranju izhoda generatorja je, da testiramo predvsem, kako dobro distribuira vrednosti zgoščevalna funkcija SHA-1. Podobne rezultate bi dobili tudi, če bi generator uporabljal funkcijo SHA-1 za popolnoma deterministično zaporedje.

Za temeljito testiranje generatorja je zato potrebno oceniti naključnost virov entropije — vhodnih podatkov. Za potrebe testiranja smo zato modificirali izvorno kodo gonilnika, tako da smo izločili postopek zgoščanja SHA-1. Nato smo prevedli Linux jedro s spremenjenim gonilnikom in ponovili teste. Slika 4.3 prikazuje spremembo izvorne kode, ki je bila za to potrebna.

```
--- random.c     2004-09-06 21:30:39.000000000 +0200
+++ random-nohash.c      2004-09-08 16:45:56.022580328 +0200
@@ -268,6 +268,8 @@
 #define BATCH_ENTROPY_SIZE 256
 #define USE_SHA

+#define DISABLE_HASH
+
 /*
  * The minimum number of bits of entropy before we wake up a read on
  * /dev/random.  Should be enough to do a significant reseed.
@@ -1416,6 +1418,12 @@
                                        sec_random_state->entropy_count);
                }

+#if DISABLE_HASH
+               memset(tmp, 0, sizeof(tmp));
+               memcpy(tmp, r->pool + (ret/4 % (r->poolinfo.poolwords-4)), 16);
+#else
+
                /* Hash the pool to get the output */
                tmp[0] = 0x67452301;
                tmp[1] = 0xefcdab89;
@@ -1448,6 +1455,8 @@
                x ^= (x >> 16);         /* Fold it in half */
                ((__u16 *)tmp)[HASH_BUFFER_SIZE-1] = (__u16)x;
 #endif
+
+#endif

                /* Copy data to destination buffer */
                i = min(nbytes, HASH_BUFFER_SIZE*sizeof(__u32)/2);
```

Slika 4.3: Spremembe izvorne kode gonilnika

Pri modificiranem gonilniku smo namesto vmesnika `/dev/urandom` testirali vmesnik `/dev/random`, saj smo tako izločili tudi faktor povratnega mešanja. Ker je ta vmesnik veliko počasnejši, testi DIEHARD pa zahtevajo vsaj 11MB podatkov, smo izvajali zgolj ENT in NIST teste.

ENT

Slika 4.4 prikazuje rezultat statističnih testov ENT nad modificiranim gonilnikom.

```
Entropy = 7.999067 bits per byte.

Optimum compression would reduce the size
of this 250000 byte file by 0 percent.

Chi square distribution for 250000 samples is 323.65, and
randomly would exceed this value 0.50 percent of the times.

Arithmetic mean value of data bytes is 127.6996 (127.5 = random).
Monte Carlo value for Pi is 3.144914319 (error 0.11 percent).
Serial correlation coefficient is 0.001076
(totally uncorrelated = 0.0).
```

Slika 4.4: Rezultati ENT statističnih testov z modificiranim gonilnikom

Tudi tokrat je generator prestal teste, torej so tudi viri entropije dovolj naključni.

NIST 800-22

Generator je prestal 180 od 188 NIST testov. Spet so bili neuspešne zgolj različice strogih testov z iskanjem vzorcev (angl. *Template Matching Test*).

Statistično testiranje modificiranega gonilnika je tako pokazalo, da so viri entropije dovolj naključni.

4.2 Ustrezanje kriterijem varnih PRNG

Implementacija `/dev/random` se drži večine načel razvoja varnih generatorjev naključnih števil:

- Generator je odporen na analizo notranjega stanja. Mešalna funkcija GFSR in povratno mešanje s SHA-1 zagotavljata, da delna informacija o notranjem stanju ne pomaga pri kriptoanalizi generatorja.
- Odporen je tudi na napade z izbranim vhodom. Entropijo lahko povečujejo samo vhodni podatki, za katere skrbi jedro operacijskega sistema. Edini način za napad je popoln nadzor nad strežnikom, ki generira naključna števila.
- Analiza izhodnih podatkov je možna samo z razbitjem zgoščevalne funkcije SHA-1. Pri uporabi vmesnika `/dev/random` namesto `/dev/urandom` tudi to ne zadostuje, saj mera entropije v notranjem stanju zagotavlja konstantno zalogo resnično naključnih vrednosti.

- Notranje stanje je dobro zaščiteno. Shranjeno je v pomnilniku, tako da varnost zagotavlja operacijski sistem Linux. Ker gonilnik teče znotraj jedra, je zaščita še večja — vsakršna izzvana napaka bo povzročila zaustavitev sistema.
- Implementacija generatorja je prosto dostopna, kot tudi preostala izvorna koda operacijskega sistema Linux. To omogoča enostavno revizijo implementacije in odprt postopek izboljšave gonilnika.
- Generator se drži načela o vzdrževanju količine entropije v notranjem stanju. Pri uporabi vmesnika `/dev/random` gonilnik tudi vrača samo toliko naključnih števil, kolikor je entropije na zalogi.

Ne vsebuje pa generator avtomatičnih statističnih testov izhodnih podatkov. V teoretičnem primeru odpovedi vseh virov entropije, ki so na voljo, je možno deterministično delovanje generatorja. Tak scenarij bi zahteval na primer odsotnost miške in tipkovnice ter okvaro diska, ki bi povzročila konstantne dostopne čase. Tak scenarij pa je malo verjeten, saj bi okvara diska prej povzročila zaustavitev delovanja strežnika.

4.3 Analiza izvorne kode

Ena izmed velikih prednosti generatorja `/dev/random` pred ostalimi je predvsem prosta dostopnost izvorne kode. Vsakršna napaka v implementaciji ali napačna izbira kriptografskih metod je vidna množici razvijalcev in kriptografov. Ker je postopek razvoja jedra operacijskega sistema Linux odprt, je omogočena zunanja revizija in hitro odpravljanje napak.

Tudi sami pri analizi izvorne kode nismo našli napak. Implementacije kriptografskih algoritmov (npr. GFSR in SHA-1) so vzete iz drugih paketov, zato so tudi dobro preizkušene [8].

4.4 Možni napadi na generator

Preverili smo teoretične možnosti za tri tipe napadov na generator: direktni kriptoanalitični napad, napad z vhodom in napad na notranje stanje.

Direktni kriptoanalitični napad je možen samo v primeru razbitja zgoščevalne funkcije SHA-1. Ker napad na SHA-1 v času pisanja seminarske naloge ne obstaja, lahko to možnost opustimo.

Napadi z izbranim vhodom tudi ne pridejo v poštev, saj tak napad zahteva popolno kontrolo nad strežnikom, kjer teče gonilnik (ko ima napadalec torej tudi

možnost potvarjanja naključnih števil). Podobno je z napadi z znanim in ponovljenim vhodom, saj se vhodni podatki pridobivajo znotraj jedra operacijskega sistema.

Tudi napad na notranje stanje zahteva razbitje algoritma SHA-1 ali nadzor nad strežnikom. Napad, ki izkorišča nizko stopnjo entropije ob zagonu sistema, je preprečen z naslednjim postopkom: ob vsaki zaustavitvi sistema se na trdi disk shrani notranje stanje; ob ponovnem zagonu se tako generator lahko takoj reinicializira na zadnje stanje.

Generator je torej ob predpostavkah, da je algoritem SHA-1 varen in strežnik nedostopen, varen pred temi napadi.

4.5 Slabosti sistema

Izpostaviti je potrebno tri slabosti generatorja `/dev/random`: mešalna funkcija GFSR, napake v ocenah entropije in stradanje gonilnika.

4.5.1 Mešalna funkcija GFSR

Funkcija GFSR se uporablja za mešanje vhodnih podatkov v notranje stanje. Uporablja se predvsem zaradi svoje hitrosti. Vhodni podatki se namreč pridobivajo med prekinitvami (IRQ), prekinitvene zahteve pa se morajo zaradi odzivnosti sistema izvajati čim hitreje. GFSR v tem primeru predstavlja kompromis med varnostjo in hitrostjo.

Uporaba zgoščevalne funkcije SHA-1 bi bila s kriptografskega stališča boljša rešitev. Glede na to, da je bila zadnja različica gonilnika napisana v letu 1999, argument hitrosti nima več veljave: algoritem SHA-1 je sicer okrog 10-krat počasnejši od GFSR [11], vendar so se v tem času hitrosti procesorjev povečale še za večji faktor.

4.5.2 Napake v ocenah entropije

Algoritem delta-delta-3 za oceno entropije poraja več vprašanj. Vprašljiva je izbira treh stopenj odvodov časovnih razlik za vse vrste naključnih dogodkov. Poleg tega je algoritem občutljiv na učinek kvantizacije — pri desetkrat počasnejši uri bo tudi entropija dogodka, ki sicer nastopi ob enakih intervalih, večja za $\log_2 10$.

Težava pa se pojavi tudi pri povratnem mešanju izhodnih podatkov. Entropija se pri tem poveča, čeprav gre zgolj za mešanje obstoječih naključnih vrednosti. To pa je v nasprotju z načeli razvoja generatorjev naključnih števil.

4.5.3 Stradanje gonilnika

Pri uporabi vmesnika `/dev/random` se lahko pojavi problem stradanja gonilnika. Ta vmesnik namreč vrača samo toliko naključnih vrednosti, kolikor je entropije v notranjem stanju. Ko entropije zmanjka, vmesnik čaka na nove vhodne podatke.

Problem se lahko pojavi pri strežnikih, kjer je poraba naključnih števil velika (npr. zaradi pogostega generiranja sejnih ključev pri spletnih strežnikih). Še dodatno oviro predstavlja dejstvo, da so taki strežniki pogosto brez miške in tipkovnice, tako da so edini vir entropije dostopni časi diska in prekinitve. Pojav turbulence v ohišju trdega diska, ki omogoča naključnost dostopnih časov, pa v povprečju zagotavlja zgolj 100 naključnih bitov na minuto.

V tem primeru moramo uporabiti druge vire naključnosti ali celo strojni generator naključnih števil, kot je na primer Intel RNG.

5 Primerjava z drugimi generatorji

5.1 Yarrow

Yarrow je specifikacija za razvoj generatorjev naključnih števil, zasnoval pa jo je Bruce Schneier [5].

Podobno kot `/dev/random` pridobiva entropijo iz časov vhoda z miške in tipkovnice. Vendar pa vsak posamezen dogodek ne pripomore k entropiji — dogodki se sprva zapisujejo v polje, ko pa se to napolni, se pošlje v funkcijo za oceno entropije.

Funkcija za oceno entropije polja izbira med tremi različnimi možnimi entropijami za vsak tip dogodka. Prvo oceno poda razvijalec sam; drugo oceno da velikost kompresiranega entropijskega polja (pri tem je uporabljen hiter kompresijski algoritem); tretja ocena je polovica velikosti polja. Najmanjša od treh ocen se vzame kot entropija polja.

Naključna števila skupaj z oceno entropije se shranijo v prvo (hitro) polje notranjega stanja. Ko mera entropije doseže 100 bitov, se SHA-1 odtis tega polja doda v drugo (hitro) polje notranjega stanja, skupaj z oceno entropije. Ko entropija drugega polja doseže 160 bitov, se iz njegovega SHA-1 odtisa generira ključ. Ta ključ se uporablja za šifriranje števca, katerega velikost določi razvijalec. Dobljena šifra je rezultat generatorja. Ključ za šifriranje se ponovno generira, ko je na voljo novih 160 bitov entropije, ali pa vsakih 10 poskusov (karkoli nastopi prej).

V osnovi je Yarrow zasnovan bolj varno od generatorja `/dev/random`. Ocene entropije so bolj konzervativne; uporabljeni so tudi izključno kriptografsko močni algoritmi. Ker je Yarrow zgolj specifikacija za generatorje naključnih števil, brez delujoče referenčne implementacije za operacijski sistem Linux, statističnih testov za primerjavo nismo izvajali; znano pa je, da je implementacija generatorja Yarrow za operacijski sistem FreeBSD prestala vse DIEHARD teste.

5.2 Strojni generator Intel RNG

Strojni generator naključnih števil Intel RNG je prisoten v vezjih Intel 810 in naprej — integriran je torej v matično ploščo osebnega računalnika. Deluje na osnovi termičnega šuma: dva upornika s šumom povzročata modulacijo počasnega osci-

latorja, ki vpliva na frekvenco hitrega oscilatorja. Kljub naključni naravi pojava pa je potreben še tako imenovani von Neumannov korektor, ki odpravlja pristranost hitrega števca. Korektor pregleduje pare bitov in sprejme kvečjemu en bit: $(1,0) \rightarrow 0$, $(0,1) \rightarrow 1$, $(0,0) \rightarrow$ brez izhoda, $(1,1) \rightarrow$ brez izhoda.

Ker je vir en sam in gre v osnovi za naključen pojav, se ocene entropije ne izvajajo. Izhod generatorja je podvržen še zgoščevalni funkciji SHA-1.

Prednost generatorja je poleg dobrega vira naključnosti predvsem hitrost. Sposoben je namreč proizvajati 75K bitov na sekundo. Za primerjavo: `/dev/random` generator na strežniku brez miške in tipkovnice zagotavlja zgolj 100 bitov na sekundo; ob uporabi osebnega računalnika z miško in tipkovnico pa v povprečju generira 600 bitov na sekundo.

Generator Intel RNG je bil tudi certificiran s strani Cryptography Research Inc. (CRI), prestal pa je tudi vse teste FIPS 140-1. Statističnih testov nismo izvajali, saj bi rezultati prikazali zgolj naključnost zgoščevalne funkcije SHA-1; do vhodnih podatkov pa ni mogoče priti, saj gre za strojni generator.

6 Zaključek

Analiza generatorja naključnih števil `/dev/random` je pokazala, da je ta dovolj varen za uporabo v kriptografskih aplikacijah. Prestal je zahtevne statistične teste DIEHARD in NIST, tako z vidika izhoda kot virov entropije.

Analiza izvorne kode gonilnika je pokazala, da implementacija sledi načelom razvoja varnih generatorjev naključnih števil. Odprta narava izvorne kode jedra operacijskega sistema Linux tudi omogoča enostavno preverjanje in popravljanje napak.

Našli smo tudi nekaj slabosti, ki vplivajo na varnost in uporabnost sistema. Uporabo funkcije GFSR bi bilo primerno zamenjati s SHA-1. Generator Yarrow je s tega vidika varnejši, saj namesto kriptografsko šibke linearne rekurzije uporablja zgoščevalno funkcijo SHA-1. Glede na to, da so v zadnjem času odkrili slabosti zgoščevalnih funkcij MD5, SHA-0 in RIPEMD-160, pa bi bila morda na mestu tudi nadgradnja z algoritmom SHA-256 ali močnejšim.

Ena izmed slabosti je tudi stradanje gonilnika pri strežnikih z veliko porabo naključnih števil. Ta primer lahko rešimo z uporabo strojnega generatorja, na primer Intel RNG.

Literatura

[1] D. Davis, R. Ihaka, and P. Fenstermacher. Cryptographic randomness from air turbulence in disk drives. *Advances in Cryptology - CRYPTO '94 Proceedings*, pages 114–120, 1994.

[2] D. Eastlake, S.D. Crocker, and J.I. Schiller. Randomness requirements for security, RFC 1750. *Internet Engineering Task Force*, 1998.

[3] P. Gutmann. Software generation of random numbers for cryptographic purposes. *Proceedings of the 1998 Usenix Security Symposium*, pages 243–257, 1998.

[4] B. Jun and P. Kocherm. The Intel random number generator cryptography research. *Intel*, 1999.

[5] J. Kelsey, B. Schneier, and N Ferguson. Yarrow-160: Notes on the design and analysis of the Yarrow cryptographic pseudorandom number generator. 1999.

[6] J. Kelsey, B. Schneier, D. Wagner, and C. Hall. Cryptanalytic attacks on pseudorandom number generators. *Fast Software Encryption, Fifth International Workshop Proceedings*, pages 168–188, 1998.

[7] G. Marsaglia. *DIEHARD: a battery of tests for random number generators*, 1985.
`http://stat.fsu.edu/~geo/diehard.html`.

[8] C. Plumb. Truly random numbers. *Dr. Dobb's Journal*, 19(13):113–115, 1994.

[9] A. Rukhin, J. Soto, J. Nechvata, et al. A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST Special Publication 800-22*, 2001.

[10] T. Ts'o. */dev/random driver source code (random.c)*, 1999.
Linux Kernel Source Code, `http://www.kernel.org`.

[11] J. Viega. Practical random number generation in software. *19th Annual Computer Security Applications Conference (ACSAC 2003)*, page 129, 2003.

[12] J. Walker. *ENT: A Pseudorandom Number Sequence Test Program*, 1989.
`http://www.fourmilab.ch/random`.