**Davor Munda**
Kriptografija in računalniška varnost
1. letnik podiplomski študij, 2007/08
FRI, Ljubljana

# Analiza generatorjev psevdo-naključnih števil v operacijskem sistemu Linux

Ljubljana, 28.12.2008 **Mentor: prof. dr. Aleksandar Jurišić**

## *Povzetek*

Analizirali smo delovanje generatorjev psevdo-naključnih števil v operacijskem sistemu Linux. Generator je integriran v Linux jedro in deluje na osnovi zbiranja entropije. Naša analiza izhaja iz članka avtorjev Gutterman, Pinkas in Reinman iz l. 2006. Avtorji so analizirali delovanje generatorja v Linux jedru 2.6.10, ki je bilo objavljeno l. 2004. Mi smo primerjali ta generator z generatorjem v Linux jedru 2.6.24, ki je bilo objavljeno l. 2008. Zanimala nas je razlika v delovanju generatorjev in odpravljanje pomanjkljivosti na katere so opozorili avtorji v članku iz l. 2006. Izhode obeh generatorjev smo tudi testirali z NIST statističnimi testi značilnosti.

Ugotovili smo, da ima Linux generator v splošnem dobre značilnosti za generatorje psevdo-naključnih števil, ima pa tudi določene pomanjkljivosti. Izgleda, da razvijalci generatorja delno upoštevajo ugotovitve avtorjev članka iz l. 2006. Tako so že odpravili nekatere pomanjkljivosti kot je npr. vpliv šuma na generator.

Ostajata pa dve slabosti generatorja. Prva je kriptografsko šibek generator v vgrajenih sistemih. Druga pa blokada delovanja kriptografsko močnejšega vmesnika in posledično nujnost uporabe kriptografsko šibkejšega vmesnika. Linux generator psevdo-naključnih števil namreč posreduje svoj izhod preko dveh vmesnikov.

# Kazalo

# 1. Uvod

Generatorji naključnih števil so kritični dejavnik v kriptografskih sistemih. Naključnost rabimo, da lahko preprečimo napade, ki znajo poiskati in izkoristiti matematične zakonitosti generiranih ključev. Za varno delovanje kriptografskih aplikacij je zelo pomembno, da generatorji generirajo čimbolj naključna števila. Varnost kriptografskih sistemov je odvisna tudi od varnosti generatorjev naključnih števil.

Ker so fizikalni viri naključnosti kot sta npr. termični šum in fotoefekt za praktično uporabo običajno predragi, se v večini sistemov uporabljajo generatorji psevdo-naključnih števil. Generatorji psevdo-naključnih števil (PRNG) so deterministični algoritmi, ki na osnovi krajše naključne binarne sekvence (seme) generirajo veliko daljšo binarno sekvenco, ki ima značilnosti kot, da bi bila naključna sekvenca[1].

Ker za generiranje uporabljamo deterministično algoritem, ki ga napadalec pozna in ker napadalec delno pozna tudi vir za seme, moramo generirati kriptografsko varno zaporedje, ki ga napadalec ne more predvideti. Minimalna varnostna zahteva je, da mora imeti naključno seme dovolj veliko dolžino, da požrešni napad po vseh možnih semenih ni izvedljiv. Dve glavni zahtevi pa sta, da ima psevdo-naključna sekvenca takšne statistične značilnosti, ki jih ni možno ločiti od značilnosti resnično naključne sekvence in da se generirani biti ne dajo predvideti z omejenimi računskimi viri [2]. Obe zahtevi sta podrobneje opisani z naslednjima praviloma.

Generator psevdo-naključnih števil zadosti vsem statističnim testom v polinomskem času[2], če ne obstaja algoritem v polinomskem času, ki bi znal razločiti generirano sekvenco od čisto naključne sekvence enake dolžine z verjetnostjo večjo od $\frac{1}{2}$ .

Generator psevdo-naključnih števil zadosti testu naslednjega bita, če ne obstaja algoritem v polinomskem času, ki bi znal iz podzaporedja dolžine $n$ predvideti naslednji bit $n$+1 z verjetnostjo večjo od $\frac{1}{2}$ .

Ker velja, da generator psevdo-naključnih števil zadosti testu naslednjega bita natanko takrat, ko zadosti vsem statističnim testom v polinomskem času, velja test naslednjega bita za univerzalen test [2]. Generatorjem psevdo-naključnih števil, ki zadostijo testu naslednjega bita pravimo kriptografsko varni generatorji (CSPRNG). Primera takih generatorjev sta RSA in Blum-Blum-Shub.

## 1.1. Implementacija generatorja psevdo-naključnih števil s programsko opremo

V implementaciji generatorja psevdo-naključnih števil s programsko opremo moramo, zraven seveda kvalitetne generacije števil, ki imajo dobre statistične značilnosti za naključna števila, paziti, da uporabimo kvalitetno naključno seme (seed). Napadalec ne sme poznati ali imeti možnost, da ugotovi, kako seme je uporabljeno za inicializacijo postopka generacije psevdo-naključnih števil.

Pri semenu ocenjujemo kvantiteto in kvaliteto. S kvantiteto zagotovimo, da napadalec ne more uporabiti požrešnega napada z vsemi možnimi semeni. Kvaliteta semena pa se meri z njegovo entropijo. Popolnoma naključno seme ima maksimalno entropijo.

Poskrbeti moramo, da pridobimo maksimalno možno informacijo iz zunanjih virov, ki imajo največjo entropijo. Če rabimo več informacije za seme, jo dodajamo iz virov z manjšo entropijo. Kombinacija uporabe več virov oteži napad na generator.

1 Isto seme daje kot rezultat isto psevdo-naključno zaporedje.

2 Čas izvajanja testa je omejen na polinom stopnje dolžine generirane sekvence (časovna kompleksnost $O(n^l)$; $l$ je dolžina sekvence).

**Tabela 1: Možni viri za seme**

| Značilnosti sistema | Spremenljive vrednosti | Zunanji viri |
|---|---|---|
| Konfiguracijske datoteke, konfiguracija diskov, spremenljivke okolja. | Vsebina zaslona, datum in čas, zadnja pritisnjena tipka, vsebina log datotek, pomnilniška statistika, mrežna statistika, statistika procesov. | Premiki in časi aktivnosti miške, časi pritiskov na tipke tipkovnice, aktivnost vhoda za mikrofon. |
| Entropija<br>majhna → visoka | | |

## 1.2. Implementacija generatorja psevdo-naključnih števil v operacijskem sistemu Linux

Linux je operacijski sistem, ki ga je leta 1991 predstavil Linus Torvalds in temelji na Minix operacijskem sistemu, ki je izvedenka Unix operacijskih sistemov. Linux se objavlja po načelih GNU GPL, ki jih je leta 1983 predstavil Richard Stallman. Linux je odprto kodni projekt, ki je danes aktualen kot alternativa MS Windows sistemom.

Linux generator psevdo-naključnih števil je del Linux jedra, generirana števila pa so dostopna preko programskega vmesnika (API). Psevdo-naključna generirana števila se uporabljajo za namene naključnih identifikatorjev, za številčenje TCP sekvenc, za izdelavo gesel, za generacijo SSL ključev, ipd. Generator posreduje svoj izhod preko dveh vmesnikov */dev/random* in */dev/urandom*. Vmesnik */dev/random* je kriptografsko varnejši, vendar posreduje le omejeno količino števil in blokira delovanje dokler ni generirana zahtevana količina psevdo-naključnih števil. Vmesnik */dev/urandom* nikoli ne blokira delovanja, vendar pa posreduje kriptografsko manj varna psevdo-naključna števila.

Linux generator psevdo-naključnih števil je v celoti implementiran v izvorni datoteki *drivers/char/random.c* [6]. Koda je odprta in se spreminja skozi verzije Linux jedra. Problem je, da delovanje generatorja ni dokumentirano. Tudi posegi v generator niso dokumentirani in v splošnem ne vemo kako generator deluje in kakšne posledice imajo določeni posegi v generator. Za uspešno analizo delovanja generatorja moramo izvajati obratni inženiring (reverse engineering) in s testiranji opazovati efekte posegov v generator.

Leta 2006 so izraelski znanstveniki Gutterman, Pinkas in Reinman objavili analizo delovanja Linux generatorja psevdo-naključnih števil [4]. V članku so med drugim opozorili tudi na slabosti generatorja. Njihova analiza je narejena na generatorju v Linux jedru 2.6.10, ki je bilo objavljeno decembra l. 2004. Mi smo naredili primerjavo tega generatorja z generatorjem v Linux jedru 2.6.24, ki je bilo objavljeno januarja l. 2008. Zanimalo nas je predvsem v kolikšni meri so odpravljene slabosti na katere so opozorili izraelski znanstveniki.

Članek smo organizirali po poglavjih. V drugem poglavju smo povzeli ugotovitve avtorjev članka [4]. V tretjem poglavju smo analizirali razlike med generatorjema v Linux jedrih 2.6.10 in 2.6.24. V četrtem poglavju smo objavili rezultate NIST statističnih testov značilnosti za generatorja v obeh Linux jedrih. V petem poglavju pa smo podali kratek pregled nekaterih predlogov za izboljšavo Linux generatorja psevdo-naključnih števil.

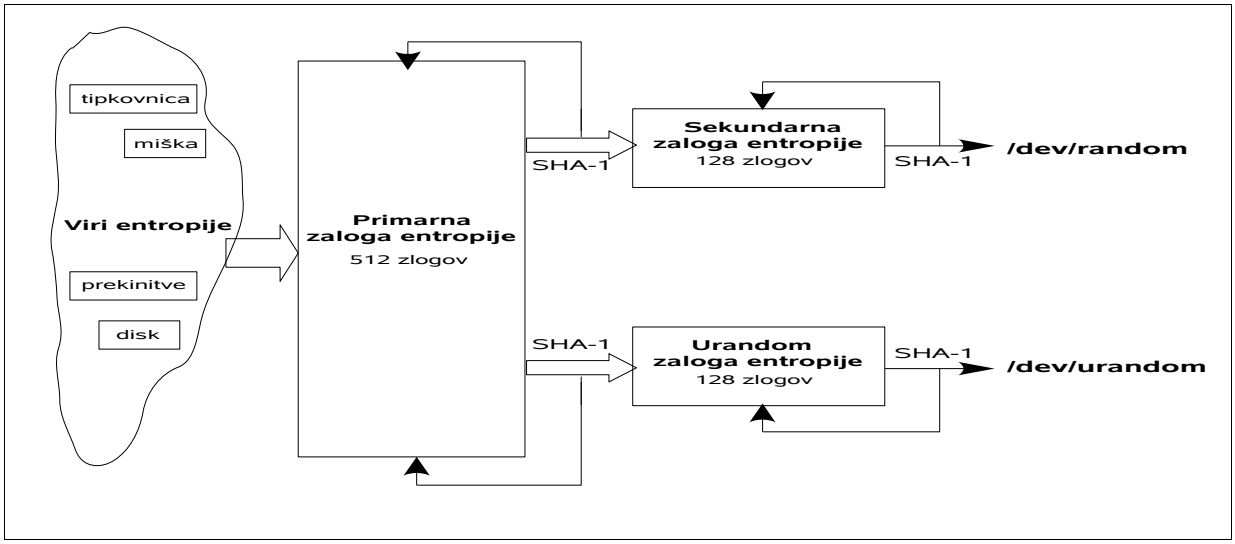
# 2. Analiza generatorja psevdo-naključnih števil za Linux jedro 2.6.10

Linux jedro 2.6.10 je bilo objavljeno decembra l. 2004. Generator psevdo-naključnih števil je del Linux jedra. Izraelski znanstveniki Gutterman, Pinkas in Reinman so spomladi l. 2006 objavili analizo delovanja tega generatorja. Povzeli bomo njihove ugotovitve.

## 2.1. Struktura Linux generatorja psevdo-naključnih števil v jedru 2.6.10

Psevdo-naključna števila se v Linuxu generirajo s tremi asinhronimi postopki. Prvi postopek je zbiranje entropije v operacijskem sistemu. Entropija se zbira preko različnih dogodkov znotraj jedra operacijskega sistema. Z drugim postopkom se polni zaloga entropije, ki služi kot vir za pridobivanje psevdo-naključnih vrednosti. Tretji postopek pa skrbi za zahteve po pridobivanju psevdo-naključnih vrednosti, ki praznijo zalogo entropije. Biti, ki se odvzemajo zalogi entropije, gredo skozi zgoščevalni algoritem SHA-1[3]. Del odvzetih bitov, obdelanih s SHA-1, se povratno vrača nazaj v zalogo entropije. Tako dobimo neke vrste linearni pomikalni register s povratno vezavo (LFSR). Algoritem SHA-1 vnaša nelinearnost v sicer linearen sistem.

**Slika 1: Struktura Linux generatorja psevdo-naključnih števil v jedru 2.6.10**




Tri zaloge entropije (primarna, sekundarna in urandom) predstavljajo notranje stanje LFSR. Primarna zaloga entropije se polni iz virov entropije in povratno ob praznjenju. Ostali dve zalogi se polnita iz primarne zaloge in povratno ob praznjenju. Vsaka od zalog entropije ima svoj števec s katerim vrednoti količino entropije. Števci imajo stanja od 0 do velikosti zaloge v bitih (enkrat 0,...,4096 in dva krat 0,...,1024). Za vsak bit, ki gre ven iz zaloge, se ustrezni števec zmanjša za 1. S pridobivanjem entropije pa se ustrezni števec veča tako, da se računa pridobljena entropija. Vmesnik */dev/random* blokira delovanje, ko pade njegov števec entropije na 0. Njegovo delovanje se zaustavi dokler se v sistemu ne nabere zahtevana količina entropije.

3 SHA-1 [2] je zgoščevalni algoritem, ki ga je objavila agencija NIST. Uporablja se v varnostnih protokolih (npr. SSL, SSH) in za digitalno podpisovanje.

V primarno zalogo entropije se dodajajo biti preko zunanjih virov. Možni zunanji viri za pridobivanje entropije so: aktivnosti tipkovnice in miške, vhodno/izhodni dostopi do diskov in sistemske prekinitve[4]. Taki dogodki producirajo 32 bitov informacije, ki predstavlja čas dogodka in 32 bitov informacije, ki opisuje dogodek (npr. katera tipka je bila pritisnjena). Tako pridobljena informacija se v zalogo entropije dodaja paketno s posebnim postopkom. Ko je primarna zaloga entropije polna, se entropija dodaja v sekundarno zalogo. Ko je sekundarna zaloga polna, se samo vsebina, brez naraščanja entropije, naprej dodaja v primarno zalogo. Entropija iz zunanjih virov se nikoli neposredno ne dodaja v urandom zalogo.

Psevdo-naključni biti se črpajo iz vseh treh zalog entropije. Linux jedro za svoje potrebe (npr. generiranje naključnih identifikatorjev) črpa iz urandom zaloge. Sicer pa se iz sekundarne (random) in urandom zaloge črpa preko obeh vmesnikov (*/dev/random* in */dev/urandom*). Ob črpanju bitov iz sekundarne zaloge ali iz urandom zaloge entropije, se zaloga polni iz primarne zaloge entropije in povratno. Ob črpanju iz primarne zaloge entropije (pretakanje), se primarna zaloga ravnotako povratno polni.

### 2.1.1. Inicializacija generatorja

Inicializacija generatorja psevdo-naključnih števil se izvrši ob zagonu sistema. Ob zagonu imamo na razpolago uro in konstantne parametre operacijskega sistema ter dostope do diska in sistemske prekinitve preko katerih dodajamo entropijo v sistem. V primeru, ko nimamo zadostnih zunanjih virov za pridobivanje entropije, ob zagonu ne moremo pridobiti zadostne količine entropije, da bi lahko generirali kriptografsko varne psevdo-naključne nize.[5]

Generator zato povezuje delovanje med ugašanjem in zagonom sistema. Med ugašanjem sistema posebna ukazna datoteka (script) prebere 512 zlogov (bytes) iz vmesnika */dev/urandom* in jih shrani na datotečni sistem. Med zagonom se tako shranjena informacija preko vmesnika */dev/urandom* vpiše v primarno zalogo entropije.[6] Tak način dodajanja entropije v primarno zalogo ne povečuje števca za oceno entropije.

### 2.1.2. Zbiranje entropije

Omenili smo že, da se entropija v sistemu zbira preko zunanjih virov. Taki viri so tipkovnica, miška, diski in sistemske prekinitve. Dogodki, ki povečujejo entropijo, so sestavljeni iz dveh 32 bitnih besed. Prva beseda vsebuje čas dogodka, druga beseda pa tip dogodka. Čas dogodka je na nekaterih arhitekturah (npr. PC) predstavljen s številom CPE urinih period od zagona, na drugih arhitekturah pa s sistemskim števcem takta *jiffies*[7]. Tip dogodka se obravnava različno za posamezne vire entropije.

4 Tipkovnica in miška delujeta preko mehanizma prekinitev, dostopi do diskov so DMA (neposreden dostop do pomnilnika), ostale prekinitve pa prispevajo k entropiji, če se registrirajo v sistem z argumentom *SA_SAMPLE_RANDOM* (na tak način se registrirajo predvsem mrežni gonilniki).

5 Nekateri sistemi (vgrajeni) nimajo diskov, tipkovnice in miške, kar otežuje pridobivanje entropije.

6 Inicializacija generatorja preko ukazne datoteke, ki ni del jedra, je lahko del Linux distribucije. Vgrajeni sistemi s samo bralnim datotečnim sistemom (RO) tako inicializacijo onemogočajo in posledično dobimo predvidljivo začetno stanje generatorja.

7 V Linux jedru 2.6.x je takt *jiffies* 4 mili sekunde.
Na PC arhitekturi se število CPE urinih period hrani v 64 bitnem registru, za zbiranje entropije se vzame spodnjih 32 bitov.

**Tabela 2: Število bitov informacije za tip dogodka po virih entropije**

| Tipkovnica | Miška | Disk | Prekinitve |
|---|---|---|---|
| 8 | 12 | 3 | 4 |

Informacija za tip dogodka

- Tipkovnica: koda tipke (0,...,255).
  Za kodo tipke rabimo 8 bitov.
- Miška: (*tip* << 4) ⊕ *koda* ⊕ (*koda* >> 4) ⊕ *vrednost*
  *tip* je pritisk/sprostitev tipke oz. začetek/konec gibanja,
  *koda* je pritisnjen gumb oz. os gibanja,
  *vrednost* je pritisk/sprostitev tipke oz. smer drsenja (drsni gumb) oz. velikost premika.
  10 bitov označuje premik, 2 bita pa tipke.
- Disk: 0x100 + ((*major* << 20) | *minor*)
  *major* in *minor* sta glavna in pomožna številka naprave (device).
  Računamo z maksimalno 8 diski v sistemu in tako dobimo 3 bite informacije.
- Prekinitve: številka prekinitvenega kanala (0,...,15).
  Za kodo številke IRQ rabimo 4 bite.

### 2.1.3. Ocenjevanje količine entropije

Ocena količine zbrane entropije je pomembna predvsem zaradi delovanja vmesnika */dev/random*, ki blokira delovanje, ko zmanjka entropije. Ocena se računa samo iz časovnih značk dogodkov, tip dogodka pa ne vpliva na oceno entropije. Vsaka zaloga entropije ima svoj števec za oceno entropije. Pisanje vsebine preko vmesnikov */dev/random* in */dev/urandom* ne vpliva na vsebino števcev entropije, čeprav se vsebina zbira v primarni zalogi entropije.

Formula za izračun ocene entropije:
$\delta_n = t_n - t_{n-1}$ ; $t_n$ je 32 bitna časovna značka *n*-tega dogodka
$\delta^2{}_n = \delta_n - \delta_{n-1}$
$\delta^3{}_n = \delta^2{}_n - \delta^2{}_{n-1}$
Ocena entropije, ki jo prispeva dogodek: $log_2(min(|\delta_n|, |\delta^2{}_n|, |\delta^3{}_n|)_{[19..30]})$ ;
[19..30] označuje bite $b_{12}$, $b_{11}$, ..., $b_1$.[8]

Tudi, če je tako izračunana entropija enaka 0, se vsebina vpiše v zalogo entropije, števec entropije pa se ne poveča. Ko črpamo bite iz zaloge entropije, se ustrezni števec zmanjša linearno za število bitov. Ko se biti entropije pretakajo iz ene zaloge v drugo se za število bitov linearno zmanjša oz. poveča ustrezni števec entropije. Števci entropije ne morejo pasti pod vrednost 0 in ne narasti nad količino bitov, ki jih imamo v zalogah entropije (4096, 1024 in 1024)[9].

8 Minimum delimo z 2 in upoštevamo samo spodnjih 12 bitov (tako dobimo zalogo vrednosti rezultata logaritma 0,1,...,11).

9 Ko pade števec entropije na 0, ne pada več. Podobno, ko doseže svojo zgornjo mejo, ne narašča več.

### 2.1.4. Osveževanje vsebine zaloge entropije

Mehanizem za osveževanje vsebine zaloge entropije je zasnovan na mehanizmu Mersennovega sukalca (TGFSR[10]) z dolžino 128 32 bitnih besed za notranje stanje[11]. Entropija, ki se zbira z dogodki, se v zalogo entropije dodaja po paketih, ko se zgodi vsaj 128 dogodkov[12]. Vpisu entropije v zalogo entropije sledi še reinicializacija TGFSR, kar pomeni, da ne dosežemo maksimalne možne dolžine periode oz. generacija ni več linearna funkcija inicialnega stanja. Sicer pa se entropija v zalogo entropije dodaja še ob inicializaciji in na zahtevo preko vmesnikov (*/dev/random* in */dev/urandom*) ter povratno ob črpanju bitov iz zaloge in ob pretakanju iz ene zaloge v drugo.

Zaloga entropije se polni na osnovi primitivnega polinoma[13], ki je določen z velikostjo zaloge.

Primitivni polinom za primarno zalogo entropije (zaloga je velikosti 128 32 bitnih besed):

$x^{128}+x^{103}+x^{76}+x^{51}+x^{25}+x+1$ .

Primitivni polinom za sekundarno in urandom zalogo entropije (zalogi sta velikosti 32 32 bitnih besed):

$x^{32}+x^{26}+x^{20}+x^{14}+x^{7}+x+1$ .

**Psevdokoda 1: algoritem za dodajanje entropije v primarno zalogo entropije**[14]

```
table[8] = [0x00000000, 0x3b6e20c8, 0x76dc4190, 0x4db26158,
            0xedb88320, 0xd6d6a3e8, 0x9b64c2b0, 0xa00ae278];
add_entropy(pool, i, w) begin
      temp := w;
      temp := temp xor pool[i];
      temp := temp xor pool[(i+1) mod 128];
      temp := temp xor pool[(i+25) mod 128];
      temp := temp xor pool[(i+51) mod 128];
      temp := temp xor pool[(i+76) mod 128];
      temp := temp xor pool[(i+103) mod 128];
      temp := (temp >> 3) xor table[temp & 7];

      pool[i] := temp;
end.
```

*pool* označuje zalogo entropije, *i* je indeks pozicije kamor se vpiše vrednost in *w* je 32 bitna beseda, ki jo vpisujemo v zalogo.

Pri dogodkih imamo 32 bitov informacije za tip dogodka in 32 bitov informacije za čas dogodka. Pri vsakem dogodku tako pridobimo dve 32 bitni besedi za zalogo entropije.

10 Prednost TGFSR [14] je podaljšanje periode, slabost pa slabša kriptografska varnost.

11 Tako dobimo maksimalno dolžino periode $2^{128x32}$-1.

12 Entropija se zbira z dogodki in se shranjuje v vrsto. Da ne bremenimo sistema z računanjem med prekinitvami, se entropija neposredno ne vpisuje v zalogo entropije, ampak se vpisuje paketno, izven prekinitev.

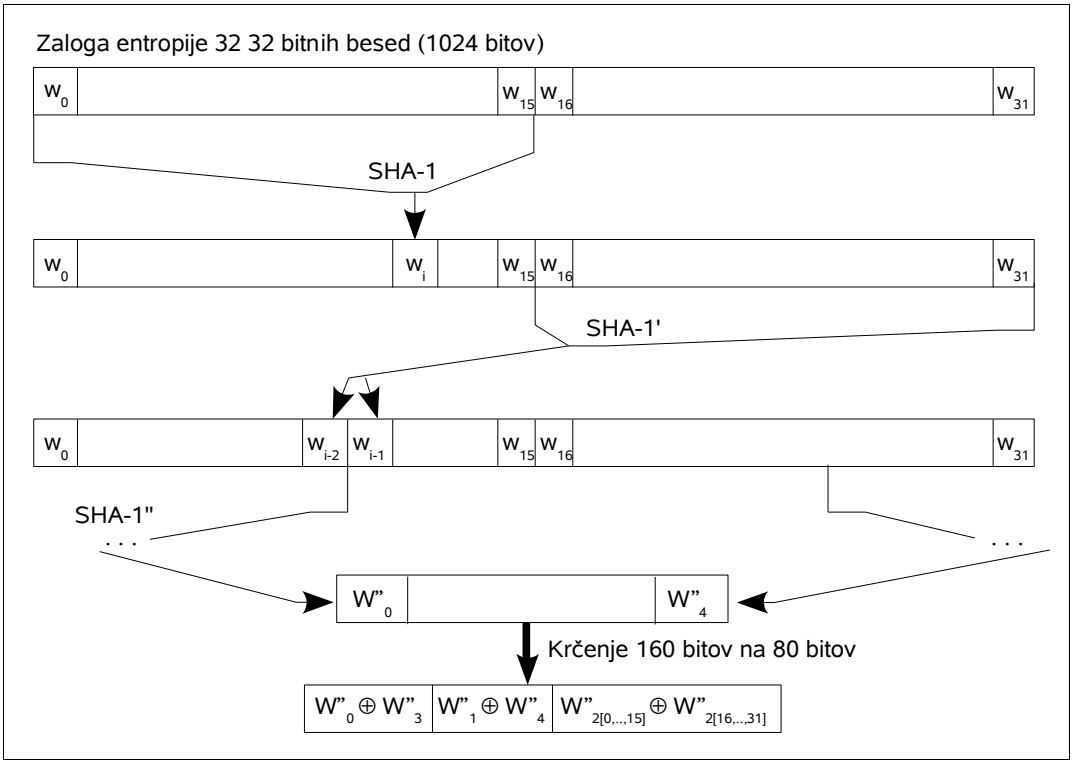
13 Primitivni polinom je nerazcepni polinom s celoštevilskimi koeficienti za katerega ne obstaja praštevilo, ki bi delilo vse njegove koeficiente. V praksi se primitivni polinomi uporabljajo pri generaciji LFSR sekvenc, ker zagotavljajo maksimalno dolge cikle [2] ($2^{L}$-1).

14 Algoritem za dodajanje v sekundarno in urandom zalogo entropije je podoben, le konstante, ki označujejo primitivni polinom so druge.

## 2.1.5. Črpanje bitov iz zaloge entropije

Črpanje bitov iz zaloge entropije ni preprosto. Ob črpanju gre informacija skozi zgoščevalni algoritem, osvežuje se vsebina zaloge in osvežujejo se števci za oceno entropije. Entropija se črpa iz sekundarne zaloge preko vmesnika */dev/random* in iz urandom zaloge preko vmesnika */dev/urandom* ter za potrebe Linux jedra. Iz primarne zaloge se entropija črpa ob pretakanju v sekundarno ali urandom zalogo. Entropija se črpa po blokih velikosti 80 bitov, dokler ne izčrpamo zahtevane količine[15]. Ob črpanju se uporabja zgoščevalni algoritem SHA-1.

**Slika 2: Algoritem za črpanje bloka bitov iz sekundarne in urandom zaloge entropije[16]**




SHA-1 se izvaja na prvih 16 besedah zaloge, 32 bitni rezultat se vpiše na *i*-to mesto v zalogo. SHA-1' se izvaja na drugih 16 besedah zaloge, 64 bitni rezultat pa se vpiše na *i-2* in *i-1* mesti v zalogo. Potem se izvede še SHA-1" na 16 besedah zaloge do *i-2* besede, ki daje 160 bitni rezultat.[17] Na koncu se dobljenih 160 bitov še skrči na 80 bitov. Teh 80 bitov predstavlja psevdo-naključne bite, ki jih dobimo kot rezultat generatorja.

Vidimo, da s pomočjo zgoščevalnega algoritma SHA-1, ob črpanju bitov iz zaloge entropije, spreminjamo tudi vsebino zaloge. Če napadalec ne pozna inverzne funkcije SHA-1, ne more na osnovi izhoda ugotoviti notranjega stanja generatorja.

15 Črpanje iz sekundarne zaloge entropije se blokira, ko pade ocena entropije na 0 in se nadaljuje, ko entropija v sistemu zopet naraste.

16 Algoritem za črpanje iz primarne zaloge entropije je nekoliko drugačen [4]. SHA-1 zgoščevanje se izvaja na vsaki od osmih skupin 16 32 bitnih besed, končni rezultat, ki se pretoči, pa je ravnotako dolžine 80 bitov oz. zahtevano število bitov.

17 Indeks *i* predstavlja trenutno pozicijo na katero se vpisuje vsebina zaloge entropije. Indeks se krožno pomika po zalogi entropije v smeri nazaj.

## 2.2. Analiza Linux generatorja psevdo-naključnih števil v jedru 2.6.10

Narejen je opis kriptoanalitičnega napada na generator, analiza entropije, ki se dodaja z dogodki, opis slabosti generatorjev v vgrajenih sistemih in splošni pregled slabosti generatorja.

### 2.2.1. Kriptoanalitični napad na generator

Iz strukture generatorja je razvidno, da se izhod (biti, ki jih črpamo) postavi po tem, ko se osveži vsebina zaloge entropije. Kar pomeni, da, če poznamo stanje zaloge entropije v nekem trenutku, lahko enostavno izračunamo zadnji izhod, ki je bil pred tem trenutkom. To napadalcu omogoči uspešnejši napad na notranje stanje generatorja.

Takšen napad je učinkovit, če med napadom ne pride do osveževanja zaloge entropije. Ker sekundarna zaloga entropije ne daje izhoda, ko entropija dovolj pade, je takšen napad aktualen predvsem na urandom zalogo, ki daje izhod, tudi brez, da bi se stanje entropije osveževalo. Napad je lahko aktualen tudi na sekundarno zalogo entropije, vendar mora biti izveden v trenutku, ko je števec entropije dovolj visok. Napad na primarno zalogo entropije pa je lahko učinkovit, če je entropija, ki se dodaja v zalogo, predvidljiva.

Scenarij napada je natančno opisan v [4]. Opisani sta dve možnosti. Prva možnost je splošni napad, za katerega rabimo $2^{96}$ iteracij, kar je še vedno za danes običajne računalnike računsko nedosegljivo v realnem času[18]. Druga možnost, ki predvideva srečo, da napadamo v trenutku, ko je indeks za pozicijo, na katero se dodaja entropija oz. se vpisuje rezultat SHA-1, na eni od 18 predpostavljenih pozicij[19], pa zahteva $2^{64}$ iteracij, kar pa je za danes običajne računalnike že dosegljivo v realnem času.

### 2.2.2. Analiza entropije, ki se dodaja z dogodki

Opisali smo že kako se z dogodki dodaja vsebina v zaloge entropije in kako se ocenjuje dodana vsebina. Za posamezni dogodek se dodata dve 32 bitni besedi v zalogo entropije, ocena entropije pa se računa na osnovi časovnih značk dogodkov.

V članku [4] so se pri analizi entropije, ki se dodaja z dogodki, omejili na dogodke, ki jih prožijo dostopi do diskov. Ugotovljeno je, da entropija v sistemu zelo počasi narašča, ker si dogodki večinoma sledijo v prekratkih časovnih intervalih, ki praktično ne povečujejo entropije. Za dodajanje entropije v sistem, mora med posameznimi dogodki preteči več časovnih enot.[20] S pokusom so vsakih 15 minut pridobili 16 bitov entropije. Takšno delovanje lahko povzroča blokiranje delovanja oz. zavrnitev storitve za vmesnik */dev/random*.

### 2.2.3. Slabosti generatorjev v vgrajenih sistemih

Kot primer generatorja v vgrajenih sistemih članek [4] navaja OpenWRT Linux distribucijo, ki se uporablja kot operacijski sistem za brezžične usmerjevalnike (wireless routers). Takšen usmerjevalnik uporablja SSL, SSH in enkripcijo za brezžično komunikacijo. Varnost teh storitev je

18 Za požrešni napad na urandom ali na sekundarno zalogo entropije rabimo $2^{1024}$ iteracij.

19 Vseh možnih pozicij je 32.

20 Empirično so izmerili, da se entropija v sistem praktično dodaja, ko je med dogodki več kot 84 časovnih enot. Če je med dogodki krajši čas, je dodana entropija praktično zanemarljiva.
Tukaj je treba omeniti še, da v vgrajenih sistemih, v katerih imamo že itak omejene vire entropije, običajno nimamo x86 arhitektur in posledično računamo entropijo preko *jiffies*, ki pa narašča neprimerno počasneje kot urina perioda CPE.

odvisna od generatorja psevdo-naključnih števil.

Takšen sistem ima zelo omejene vire entropije. Nima miške, tipkovnice in diska. Uporablja disk z bliskovnim pomnilnikom (flash), ki ne prispeva entropije v sistem. Sistem tudi nima shranjevanja stanja */dev/urandom* zaloge entropije med posameznimi zagoni.[21] Edini vir entropije v takem sistemu so mrežne prekinitve. Zunanji opazovalec lahko relativno enostavno spremlja brezžični mrežni promet in posledično pozna stanje mrežnih prekinitev. S podatki, ki jih napadalec lahko pridobi, zna v celoti predvideti izhod generatorja. Torej je taka implementacija generatorja kriptografsko izredno šibka.

## 2.2.4. Pregled slabosti generatorja

### 2.2.4.1. Zavrnitev storitve (denial of service)

Opisali smo že, da vmesnik */dev/random* blokira svoje delovanje, ko zmanjka entropije v sistemu. Ta značilnost omogoča dva napada, ki lahko povzročita zavrnitev storitve. Prva možnost je enostavno branje vmesnika */dev/random*. Primer je ukaz *dd if=/dev/random* s katerim lahko blokiramo delovanje vmesnika. Druga možnost, ki jo lahko izvedemo celo na daljavo, pa je napad na vmesnik */dev/urandom*. Ker se delovanje tega vmesnika nikoli ne blokira, vmesnik pa se polni iz primarne zaloge entropije, lahko izpraznimo primarno zalogo entropije in posledično onemogočimo polnjenje sekundarne zaloge entropije. Tak napad je mogoč, ker se entropija v sistemu zelo počasi nabira. Preprost primer takega napada je vzpostavitev velikega števila TCP povezav. Vsaka TCP povezava zahteva 128 zlogov (bytes) iz vmesnika */dev/urandom*.

### 2.2.4.2. Uganljivo geslo

Običajno je prva uporabnikova operacija v sistemu prijava v sistem (login). Prijava je kombinacija uporabniškega imena in gesla. V vgrajenih sistemih napadalec lahko pozna začetno stanje generatorja. Če napadalec lahko bere generirana psevdo-random števila v času prijave v sistem in takoj po prijavi, lahko ugotovi katero geslo je bilo uporabljeno za prijavo v sistem.

Za takšen napad na geslo je potrebna uporaba tipkovnice in vgrajeni sistem, ki ima zelo omejene vire entropije. Običajno vgrajeni sistemi nimajo niti tipkovnice in je takšen scenarij bolj teoretičen kot praktičen.

### 2.2.4.3. Šum vpliva na izhod generatorja

Ko je primarna zaloga entropije polna, se entropija dodaja neposredno v sekundarno zalogo entropije. Ta lastnost generatorja napadalcu omogoča, da z generiranjem dogodkov, ki dodajajo entropijo v sistem, predvidi izhod vmesnika */dev/random*.

### 2.2.4.4. Notranje stanje generatorja razkriva predhodni izhod

Ker algoritem za črpanje bitov iz zaloge entropije najprej osveži vsebino zaloge in potem

21 Vgrajeni sistemi običajno uporabljajo flash pomnilnik kot bralni pomnilnik (RO) iz katerega se bere vsebina ob zagonu. Sistem deluje iz pomnilniškega diska (ramdisk). Vsebina pomnilniškega diska se ob ponovnem zagonu izgubi, na flash pomnilnik pa se med delovanjem ne piše.

postavi izhod, lahko napadalec, ki pozna notranje stanje generatorja, ugotovi prejšnji oz. zadnji izhod.

## 2.3. Priporočila za izboljšanje Linux generatorja psevdo-naključnih števil

Glede na opravljeno analizo generatorja, avtorji članka [4] priporočajo nekatere izboljšave generatorja:

1. Za vgrajene sisteme je potrebno pridobiti dodatne vire entropije. Tako bi povečali kriptografsko varnost generatorjev v vgrajenih sistemih.
2. Določiti je treba zgornjo mejo oz. kvote za črpanje psevdo-naključnih števil po posameznih porabnikih teh števil. Tako bi lahko preprečili napade, ki povzročijo zavrnitev storitve (denial of service).
3. Omejiti je potrebno vpliv tipkovnice na generator. Tako lahko preprečimo napad na geslo, ki ga vnašamo preko tipkovnice.
4. Entropija naj se ne dodaja neposredno v sekundarno zalogo entropije, ampak le v primarno zalogo. Tako omejimo vpliv šuma na izhod generatorja.
5. Najprej naj se generira izhod iz generatorja, šele potem pa naj se osveži notranje stanje. Tako bi otežili napad na notranje stanje generatorja.
6. Upoštevanje modela Barak-Halevi [10] pri bodočih posegih v generator.[22] To bi poenostavilo strukturo generatorja, ker kompleksnost strukture skriva slabosti generatorja, ne poveča pa njegove kriptografske varnosti.
   Nadalje pa avtorji članka predlagajo še razširitev modela Barak-Halevi v smislu, da vsak uporabnik sistema dobi svoj generator, ki se razlikuje od ostalih. Tako bi potencialni napadalec lahko napadel le generator določenega uporabnika.

22 Barak in Halevi [10] sta predstavila formalni model za generatorje psevdo-naključnih števil, ki je preprost in robusten.

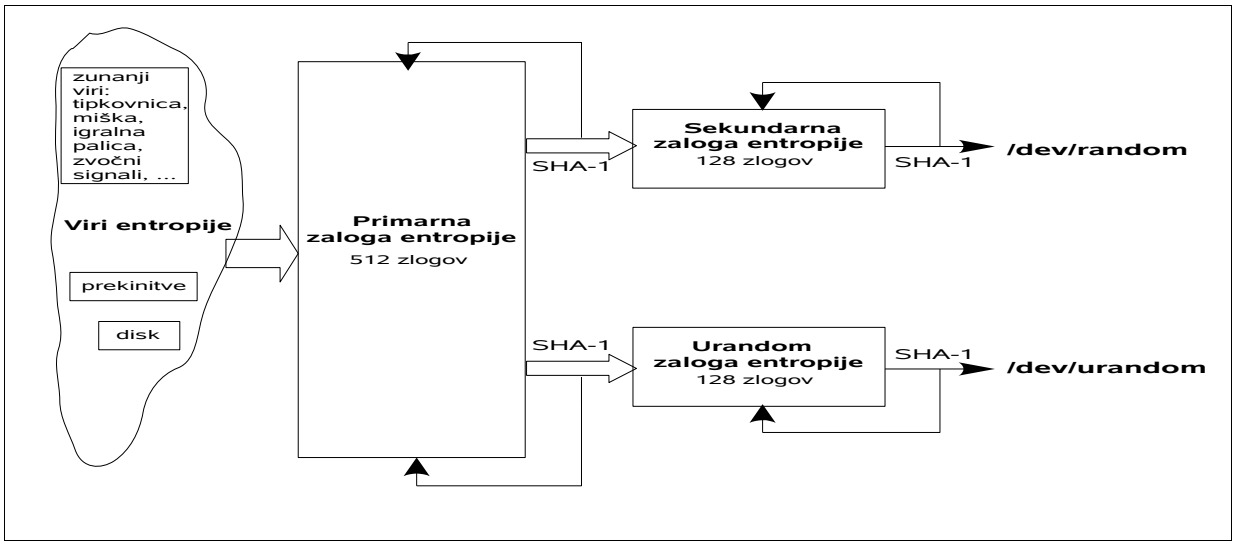
# 3. Analiza generatorja psevdo-naključnih števil za Linux jedro 2.6.24 v smislu razlik z generatorjem v jedru 2.6.10

Analizirali bomo razlike med generatorjema v Linux jedrih 2.6.10 in 2.6.24.

## 3.1. Struktura Linux generatorja psevdo-naključnih števil v jedru 2.6.24

V Linux jedru 2.6.24 imamo nekoliko spremenjeno strukturo generatorja psevdo-naključnih števil. Viri entropije iz dostopov do diskov in iz sistemskih prekinitev ostajajo nespremenjeni. Možni ostali viri entropije pa so zraven tipkovnice in miške še razne druge zunanje naprave. Zunanje naprave, ki dodajajo entropijo v sistem so tipkovnica, miška, grafična tablica (pero in miška), zvočni signali (beep, bell), igralna palica (joystick), dogodki ACPI[23] (nadzor napajanja), zaslon na dotik, temperaturni senzorji (npr. za krmiljenje ventilacije) in tipke na zunanjih napravah kot je npr. IR krmilnik za digitalni video.

**Slika 3: Struktura Linux generatorja psevdo-naključnih števil v jedru 2.6.24**




Vidimo, da je v strukturi generatorja opazna razlika z jedrom 2.6.10 v virih entropije. Imamo pa več razlik v samem delovanju generatorja.

### 3.1.1. Opis razlik v delovanju generatorja psevdo-naključnih števil v jedru 2.6.24 glede na generator v jedru 2.6.10

Informacija za tip dogodka za zunanje naprave (razen diskov in sistemskih prekinitev) se obravnava tako kot se je v jedru 2.6.10 obravnaval tip dogodka za miško[24]. Na oceno entropije, podobno kot za jedro 2.6.10, vpliva samo čas dogodka.

Algoritem za izračun ocene entropije je zelo podoben algoritmu v jedru 2.6.10 z razliko, da

23 Advanced Configuration and Power Interface.

24 $(tip << 4) \oplus koda \oplus (koda >> 4) \oplus vrednost.$

se kot čas dogodka upošteva samo takt *jiffies*. Tako dobimo enakost algoritma za različne arhitekture. Pri računanju logaritma pa se upošteva spodnjih 12 bitov minimuma[25]. Torej ni več deljenja z 2, kar poveča rezultat logaritma za 1.

Entropija se neposredno iz dogodkov dodaja samo še v primarno zalogo entropije in ne več dodatno v sekundarno zalogo. Za vsak dogodek se v primarno zalogo entropije doda, v odvisnosti od arhitekture, 4 ali 3 32 bitne besede. Na x86 (PC) arhitekturah se zraven časa dogodka in tipa dogodka doda še 64 bitna vsebina registra TSC[26]. Na arhitekturah brez TSC registra pa se zraven časa dogodka in tipa dogodka doda še 32 bitna vsebina, ki je arhitekturno odvisna. Na nekaterih arhitekturah, npr. ARM[27], je konstantno 0. Ni več paketnega dodajanja entropije, ampak se entropija v zalogo entropije dodaja takoj ob dogodku za vsak dogodek posebej.

Pri algoritmu za črpanje bloka bitov iz zaloge entropije je razlika na vhodu v SHA-1 glede na arhitekturo procesorja. Za arhitekture little-endian (npr. PC arhitektura) se pred SHA-1 obrnejo zlogi v 32 bitnih besedah (byte reverse)[28]. Rezultat SHA-1, ki se vpisuje nazaj v zalogo entropije, se v jedru 2.6.24 jemlje iz medpomnilnika po vrsti (1., 2. beseda itd.), v jedru 2.6.10 pa se iz medpomnilnika jemlje vsaka druga beseda (1., 3. beseda itd.). Algoritem SHA-1 je identičen, le njegova uporaba se nekoliko razlikuje, kar rezultira v spremembo vrednosti, ki se vpisujejo nazaj v zalogo entropije in posledično v spremembo vrednosti na izhodu generatorja. Razlika v uporabi SHA-1 nima vpliva na kriptografsko varnost.

Pri črpanju bitov iz zaloge entropije obstaja razlika v pretakanju entropije iz primarne zaloge. V jedru 2.6.24 je maksimalna količina zlogov (bytes), ki se ob črpanju pretoči iz primarne zaloge entropije omejena na 8 zlogov. Te omejitve v jedru 2.6.10 ni.

Ko črpamo entropijo iz sekundarne zaloge entropije, mora, po pretakanju iz primarne v sekundarno zalogo, v primarni zalogi ostati najmanj 16 zlogov oz. 128 bitov entropije. Te omejitve pri črpanju urandom zaloge nimamo. S tem zagotovimo, da en porabnik sekundarne zaloge entropije ne more blokirati drugega. Še vedno pa lahko porabnik urandom zaloge entropije izprazni primarno zalogo, le, da zaradi omejitve največ 8 zlogov naekrat, tega ne more narediti v enem koraku.

Imamo še razliko v pisanju vsebine preko vmesnikov */dev/random* in */dev/urandom*. Vsebina se ne dodaja več v primarno zalogo entropije, ampak v obe ostali zalogi (sekundarna in urandom) ne glede na vmesnik preko katerega se piše. Podobno kot pri jedru 2.6.10 pa tako pisanje ne vpliva na vsebino števcev entropije.

## 3.2. Učinek sprememb v delovanju generatorja psevdo-naključnih števil v jedru 2.6.24 glede na slabosti generatorja v jedru 2.6.10

### 3.2.1. Zavrnitev storitve (denial of service)

Enostavno branje vmesnika */dev/random* ne more več blokirati ostalih uporabnikov tega vmesnika, ker ne moremo izprazniti primarne zaloge entropije.

Napad na vmesnik */dev/urandom* še vedno lahko izprazni primarno zalogo entropije in posledično enemogoči polnjenje sekundarne zaloge entropije.

25 Ocena entropije je $log_2(min(|\delta_n|, |\delta^2{}_n|, |\delta^3{}_n|)_{[20..31]})$ ; [20..31] označuje bite $b_{11},...,b_0$.

26 TSC je 64 bitni register, katerega vsebina je število CPE urinih period od zagona sistema.

27 ARM je zelo priljubljena arhitektura za vgrajene sisteme.

28 Na vhodu SHA-1 dobimo tako big-endian format. Big-endian format se uporablja v omrežnih protokolih.

Zaradi algoritma za izračun ocene entropije, ki praktično dodaja entropijo v sistem, ko med dogodki preteče več časovnih enot[29] in z uporabo časovne značke za dogodke *jiffies*, se konkretno na arhitekturi PC, ki deluje v statičnem okolju, entropija zbira neprimerno počasneje[30] kot za jedro 2.6.10, kar olajša napad na blokado vmesnika */dev/random*.

### 3.2.2. Uganljivo geslo

Za teoretičen napad na geslo je potrebna uporaba tipkovnice in vgrajeni sistem. Tukaj ni razlik v kriptografski varnosti generatorja glede na jedro 2.6.10.

### 3.2.3. Šum vpliva na izhod generatorja

Entropija se z dogodki ne dodaja več neposredno v sekundarno zalogo entropije. Napadalec tako z generiranjem šuma ne more več neposredno vplivati na izhod vmesnika */dev/random*.

### 3.2.4. Notranje stanje generatorja razkriva predhodni izhod

Algoritem za črpanje bitov iz zaloge entropije še vedno najprej osveži vsebino zaloge in potem postavi izhod. Napadalec, ki pozna notranje stanje generatorja, lahko ugotovi zadnji izhod podobno kot za jedro 2.6.10. Zaradi uporabe SHA-1 pa napadalec tako ne more na osnovi izhoda ugotoviti notranjega stanja generatorja. Tukaj ni razlik v kriptografski varnosti generatorja glede na jedro 2.6.10.

### 3.2.5. Upoštevanje priporočil Gutterman, Pinkas, Reinman za izboljšave generatorja psevdo-naključnih števil

Avtorji članka [4] priporočajo izboljšave generatorja, ki deluje v jedru 2.6.10. Poglejmo, po posameznih točkah, v kolikšni meri so bila njihova priporočila upoštevana s strani razvijalcev generatorja.

1. Za povečano kriptografsko varnost generatorjev v vgrajenih sistemih ni bilo narejeno ničesar. Sicer je bilo dodanih več različnih zunanjih virov za entropijo, vendar vgrajeni sistemi teh virov praviloma nimajo na razpolago.
2. Črpanje psevdo-naključnih števil preko vmesnika */dev/random* je omejeno, da ne izprazni primarne zaloge entropije. Zavrnitev storitve (denial of service) pa ostaja aktualna, ker preko vmesnika */dev/urandom* še vedno lahko izpraznimo primarno zalogo entropije. Konkretno na arhitekturi PC, ki nima aktivnih raznih zunanjih virov entropije (npr. strežnik), se sedaj entropija zbira še neprimerno počasneje kot prej in je blokada delovanja */dev/random* še enostavnejša[31].
3. Vpliv tipkovnice na generator je omejen z dodajanjem raznih drugih zunanjih naprav kot virov entropije.
4. Entropija se ne dodaja več neposredno v sekundarno zalogo entropije. Tako je omejen vpliv

29 Pojasnilo v opombi 20.
30 Npr. 250 Hz proti $3 \cdot 10^9$ Hz za 3 GHz CPE takt.
31 Razmerje v opombi 30 je 12 milijonov.

šuma na izhod generatorja.

5. Zaporedje, najprej osveži notranje stanje in potem generiraj izhod, ostaja nespremenjeno.
6. Model Barak-Halevi [10] se ne upošteva.

# 4. NIST statistični testi

Izvedli smo statistične teste značilnosti in primerjali rezultate Linux generatorjev */dev/random* in */dev/urandom* za Linux jedri 2.6.10 in 2.6.24.

## 4.1. Metoda

Za testiranje smo uporabili programski paket *sts-1.6* [8]. Teste smo izvajali na štirih binarnih datotekah. Za vsako Linux jedro smo kreirali dve binarni datoteki, eno z bitnim tokom iz */dev/urandom* in eno z bitnim tokom iz */dev/random*. Vsaka od datotek je obsegala 131 072 zlogov oz. 1 048 576 bitov[32].

Za jedro 2.6.10 smo podatke pridobili na operacijskem sistemu *Linux Ubuntu 5.04*, za jedro 2.6.24 pa na operacijskem sistemu *Linux Ubuntu 8.04*. Oba operacijska sistema sta delovala na navidezni delovni postaji *VMware Workstation 6.0.4* nameščeni na prenosnem računalniku *HP Compaq nc8430* na operacijskem sistemu *Linux SUSE 10.1*.

NIST teste smo izvajali na prenosnem računalniku *HP Compaq nc8430* na operacijskem sistemu *Microsoft Windows XP Professional*. Za vsako datoteko smo izvedli 101 serij testov. Eno serijo testov na nizu bitov iz celotne datoteke in 100 serij testov na zaporednih bitnih podnizih dolžine 10 240 bitov te iste datoteke[33].

Testiranje vseh bitov smo izvajali z vsemi 16-imi NIST testi: delež enic in ničel v celotni sekvenci (frequency), delež enic v blokih dolžine 16 384 (block-frequency), število enic in ničel v naključno izbranih podnizih (cumulative-sums), pogostost preskokov iz ena na nič in obratno (runs), iskanje najdaljšega podniza enic (longest-run), linearna odvisnost podnizov fiksne dolžine (rank), spektralna analiza za ponavljajoče vzorce (fft), iskanje 9 bitnih predefiniranih aperiodičnih vzorcev (nonperiodic-templates), iskanje 9 bitnih predefinirah vzorcev (overlapping-templates), Maurer-ov test za stiskanje sekvence z argumenti 7 blokov dolžine 1280 (universal), ocena entropije za bloke dolžine 15 (apen), naključni sprehod osmih testov (random-excursions), naključni sprehod osemnajstih testov (random-excursions-variant), sovpadanje 16 bitnih vzorcev (serial), Lempel-Ziv test za stiskanje (lempel-ziv) in linearna kompleksnost (dolžina LFSR) za bloke dolžine 1024 (linear-complexity).

Za testiranje bitnih podnizov smo zaradi prekratkih podnizov morali izpustiti 9 testov: najdaljši podniz enic, linearna odvisnost podnizov, iskanje 9 bitnih predefinirah vzorcev, Maurer-ov test, oba naključna sprehoda, sovpadanje 16 bitnih vzorcev, test Lempel-Ziv in test linearne kompleksnosti.

Pri testu deleža enic v blokih smo spremenili dolžino bloka na 128, pri oceni entropije pa smo dolžino blokov spremenili na 8.

Pri nastavitvah parametrov za teste smo upoštevali NIST [8] navodila. Vsi testi so izvedeni na stopnji značilnosti $\alpha = 0.01$. Testiramo hipotezo, da je testna sekvenca bitov naključna. Torej imamo verjetnost 1%, da zavrnemo res naključno sekvenco oz., da naredimo napako prve vrste. Ker ne poznamo verjetnosti za napako druge vrste, ne vemo kolikšna je verjetnost, da sprejmemo nenaključno sekvenco za pravilno naključno. Rezultati posameznih NIST testov so verjetnosti, da

32 Določeni NIST testi zahtevajo minimalno $10^6$ dolge bitne nize.

33 Za relevantne rezultate NIST testov je priporočljivo testiranje več različnih neodvisnih bitnih nizov in skupna analiza rezultatov vseh testov. Ker je pridobivanje oz. črpanje bitov iz */dev/random* na Linux jedru 2.6.24 zelo zamudno, še posebej v stacionarnih sistemih, smo pridobljene bitne nize raje razkosali kot pa pridobivali nove.

popolnoma naključni generator generira statistično slabšo naključno sekvenco bitov kot je testirana[34]. Pri seriji 100 testov smo izvajali še test uniformnosti izračunanih verjetnosti[35].

## 4.2. Rezultati testov

Posebej bomo obravnavali rezultate testov za celotne 1M bitne nize in posebej rezultate za teste podnizov.

**Tabela 3: NIST testi na nizih dolžine 1 048 576 bitov**

| **Test (število testov)** | **jedro 2.6.10 število uspešnih testov, število neuspešnih testov** | | **jedro 2.6.24 število uspešnih testov, število neuspešnih testov** | |
|---|---|---|---|---|
| | /dev/urandom | /dev/random | /dev/urandom | /dev/random |
| frequency (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| block-frequency (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| cumulative-sums (2) | 2, 0 | 2, 0 | 2, 0 | 2, 0 |
| runs (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| longest-run (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| rank (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| fft (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| nonperodic-templates (148) | 145, 3 | 145, 3 | 146, 2 | 145, 3 |
| overlapping-templates (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| universal (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| apen (1) | 1, 0 | 0, 1 | 0, 1 | 1, 0 |
| random-excursions (8)[36] | 8, 0 | 8, 0 | - | - |
| random-excursions-variant (18)[37] | 18, 0 | 18, 0 | - | - |
| serial (2) | 2, 0 | 2, 0 | 2, 0 | 2, 0 |
| lempel-ziv (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |
| linear-complexity (1) | 1, 0 | 1, 0 | 1, 0 | 1, 0 |

34 Izračunana verjetnost $P$=1 pomeni, da imamo popolnoma naključno sekvenco bitov. Izračunana verjetnost $P$=0 pa pomeni, da imamo nenaključno sekvenco bitov.

35 Test uniformnosti se izvaja na izračunanih verjetnostih in z njim preverjamo uniformnost oz. neodvisnost posameznih testnih podnizov. Če ta test pade, lahko sklepamo, da nimamo neodvisnih podnizov in lahko dvomimo v zanesljivost dobljenih rezultatov.

36 Testov random-excursions za jedro 2.6.24 nismo mogli izvajati, ker imamo v testnih podatkih premalo ciklov $J$ [8] (361 za */dev/urandom* in 212 za */dev/random*, minimalno zahtevano število teh ciklov za izvedbo testov pa je 500).

37 Testov random-excursions-variant za jedro 2.6.24 nismo mogli izvajati, ker imamo v testnih podatkih premalo ciklov $J$ [8] (361 za */dev/urandom* in 212 za */dev/random*, minimalno zahtevano število teh ciklov za izvedbo testov pa je 500).

Iz tabele razberemo, da imamo približno 2% neuspešnih testov za vse testirane bitne vire pri testih nonperiodic-templates. Opazimo tudi, da dva vira padeta na testih ocene entropije (apen). Pri tem testu doseže najslabši rezultat bitni vir za jedro 2.6.24 */dev/urandom*. Ta bitni vir pri vseh ostalih testih doseže zadovoljive rezultate, torej mu manjka le entropija (podobno velja za bitni vir jedra 2.6.10 */dev/random*)[38].

**Tabela 4: NIST testi na nizih dolžine 10 240 bitov**[39]

| **Test (število testov)** | **jedro 2.6.10 test uniformnosti prestan, število uspešnih testov, število neuspešnih testov** | | **jedro 2.6.24 test uniformnosti prestan, število uspešnih testov, število neuspešnih testov** | |
|---|---|---|---|---|
| | /dev/urandom | /dev/random | /dev/urandom | /dev/random |
| frequency (100) | ✔, 97, 3 | ✔, 98, 2 | ✔, 100, 0 | ✔, 100, 0 |
| block-frequency (100) | ✔, 100, 0 | ✔, 99, 1 | ✔, 100, 0 | ✔, 99, 1 |
| cumulative-sums (200) | ✔, 192, 8 | ✔, 198, 2 | ✔, 199, 1 | ✔, 200, 0 |
| runs (100) | ✔, 100, 0 | ✔, 97, 3 | ✔, 99, 1 | ✔, 98, 2 |
| fft (100) | ✔, 100, 0 | —[40], 0, 100 | ✔, 100, 0 | —[41], 0, 100 |
| nonperodic-templates (14 800) | ✔[42], 14 360, 440 | ✔[43], 14 447, 353 | ✔[44], 14 352, 448 | ✔[45], 14 256, 544 |
| apen (100) | ✔, 100, 0 | ✔, 99, 1 | ✔, 95, 5 | ✔, 98, 2 |

Iz tabele razberemo, da pri testih fft za oba bitna vira */dev/random* nimamo uniformnosti izračunanih verjetnosti. Če test uniformnosti zanemarimo, dobimo pri obeh virih 100% uspešnost fft testov. Pri testih nonperiodic-templates imamo pri bitnih virih približno 3% neuspešnih testov. Sicer pa imamo visok[46] odstotek neuspešnih testov za teste cumulative-sums za vir */dev/urandom* za jedro 2.6.10 in za teste apen za vir */dev/urandom* za jedro 2.6.24.

Iz rezultatov NIST testov lahko sklepamo, da bitni vir */dev/urandom* za jedro 2.6.10 ne izpolnjuje kriterija uravnoteženosti enic in ničel v naključno izbranih podnizih. Bitni vir */dev/urandom* za jedro 2.6.24 pa ima premajhno entropijo. Če zanemarimo neuniformnost rezultatov testov spektralne analize, imata bitna vira */dev/random* za obe Linux jedri dobre statistične značilnosti za naključne vire.

38 Imamo naslednje rezultate testa apen: jedro 2.6.10 */dev/urandom* ApEn(15)=0.677384, jedro 2.6.10 */dev/random* ApEn(15)=0.677185, jedro 2.6.24 */dev/urandom* ApEn(15)=0.677072 in jedro 2.6.24 */dev/random* ApEn(15)=0.677351. Pri NIST testih je maksimalna možna ocenjena entropija lahko $\ln(2) \approx 0.693147$ [8]. Tukaj je treba omeniti, da je en sam test z enim argumentom za velikost blokov premalo za relevantno oceno naključnosti bitnega niza.

39 Za teste, ki niso prestali testa uniformnosti, smo rezultate šteli med neuspešne.

40 Testi fft niso prestali testa uniformnosti.

41 Testi fft niso prestali testa uniformnosti.

42 Pri testih nonperiodic-templates 200 testov (1,35%) ni prestalo testa uniformnosti.

43 Pri testih nonperiodic-templates 100 testov (0,68%) ni prestalo testa uniformnosti.

44 Pri testih nonperiodic-templates 200 testov (1,35%) ni prestalo testa uniformnosti.

45 Pri testih nonperiodic-templates 300 testov (2,03%) ni prestalo testa uniformnosti.

46 Ocena “visok odstotek” je relativna in je odvisna od metodologije testov. Po naši metodi to pomeni neuspešen rezultat testov.

# 5. Pregled nekaterih predlogov za izboljšavo Linux generatorja psevdo-naključnih števil

## 5.1. Model Barak-Halevi

Avtorji članka [4] priporočajo uporabo modela Barak-Halevi [10], ker kompleksnost strukture obstoječega generatorja skriva slabosti generatorja, ne poveča pa njegove kriptografske varnosti.

Barak in Halevi sta l. 2005 predstavila formalni model generatorja psevdo-naključnih števil, ki je enostaven in robusten. Model deluje na osnovi zbiranja entropije in je odporen na direktni kriptoanalitični napad na napad z izbranim vhodom in na napad na notranje stanje generatorja.

Predstavljeni model se lahko praktično uporabi v poljubnem okolju in ne samo za operacijski sistem Linux.

## 5.2. Dual_EC_DRBG

Generator Dual_EC_DRBG[47] je l. 2005 predstavila agencija NIST kot priporočila za generacijo psevdo-naključnih števil [15]. L. 2007 je agencija NIST objavila revizijo standarda, ki velja za aktualni Dual_EC_DRBG generator. Generator deluje na osnovi diskretnega logaritma za grupo na eliptični krivulji.

Generator je kontraverzen, ker velja kot standard, čeprav so mu dokazali pomanjkljivosti. Microsoftovi strokovnjaki za varnost so dokazali, da algoritem uporablja za svoje delovanje konstante, ki so povezane z naborom skritih številk in ki omogočajo tistemu, ki ta nabor pozna, enostaven vlom v praktično vsak kriptografski sistem, ki bazira na tem algoritmu. Ta generator velja za neke vrste NSA[48] podtaknjenca, čeprav ni dokazov, da NSA pozna ta nabor skritih števil.

Tudi ta generator je namenjen uporabi v poljubnem okolju in ne samo za operacijski sistem Linux.

## 5.3. Frandom

Frandom pomeni Fast Random in je bil l. 2003 objavljen kot programski popravek (patch) za Linux jedro v smislu, da dobimo hiter generator psevdo-naključnih števil.

Paket je objavljen na Sourceforge-u[49]. Avtor Billauer želi, da bi paket postal standardni del Linux jedra, vendar pa skrbniki Linux jedra tega ne dovolijo. Torej moramo paket samostojno itegrirati v jedro, če ga želimo uporabljati.

Frandom je mišljen kot nadomestek oz. kot dopolnilo vmesnika */dev/urandom*. Temelji na algoritmu RC4. Avtor trdi, da paket ni namenjen uporabi v kriptografiji in da je najmanj 10 krat

47 Dual Elliptic Curve Deterministic Random Bit Generator.
48 National Security Agency.
49 Sourceforge.net je portal za odprto kodne projekte.

hitrejši v generaciji psevdo-naključnih števil kot je generacija preko vmesnika */dev/urandom*.

Nasprotniki algoritma trdijo, da na račun hitrosti izgublja kriptografsko varnost in to je tudi eden od razlogov zakaj ne more postati standardni del Linux jedra.

# 6. Sklep

Z analizo generatorja psevdo-naključnih števil v operacijskem sistemu Linux smo ugotovili, da ima generator v splošnem dobre značilnosti za kriptografsko varen generator, ima pa tudi nekaj pomanjkljivosti. Kot izhodišče za analizo smo uporabili članek izraelskih znanstvenikov iz l. 2006 [4], ki analizira generator v Linux jedru 2.6.10, ki je bilo objavljeno l. 2004. Mi smo primerjali njihove ugotovitve z delovanjem generatorja v Linux jedru 2.6.24, ki je bilo objavljeno l. 2008.

Izkazalo se je, da razvijalci generatorja delno upoštevajo ugotovitve in priporočila izraelskih znanstvenikov glede možnih izboljšav generatorja. Zraven popravkov nekaterih napak v kodi[50] se delovanje generatorja spreminja nekako po priporočilih. S posegi v generator pa so se pridelale nekatere druge pomanjkljivosti. Najbolj izrazita nova pomanjkljivost je arhitekturno odvisno dodajanje vsebine entropije v primarno zalogo entropije, ki enkrat doda 4, drugič pa 3 32 bitne besede v zalogo, na nekaterih arhitekturah pa je del te vsebine konstantno 0[51].

NIST statistični testi potrjujejo dobre statistične značilnosti generatorja še posebej za vmesnik */dev/random*.[52]

Ostajata pa dve izraziti pomanjkljivosti generatorja. Prvič, da imamo v vgrajenih sistemih izrazito kriptografsko šibek generator zaradi pomanjkanja virov entropije. Ker je Linux zelo razširjen operacijski sistem v vgrajenih sistemih, je zanimivo, da se na izboljšanju te pomanjkljivosti ničesar ne naredi?[53] In drugič, da imamo zaradi zelo počasnega naraščanja števcev entropije, pridobivanje entropije preko kriptografsko močnejšega vmesnika */dev/random* relativno pogosto blokirano.[54] Aplikacije zaradi tega pridobivajo psevdo-naključne nize števil predvsem preko vmesnika */dev/urandom*, ki ne blokira delovanja, je pa kriptografsko šibkejši.[55]

Med priporočili za izboljšanje značilnosti generatorja je tudi uporaba modela Barak-Halevi [10]. Zaenkrat pri Linux generatorju psevdo-naključnih števil ni aktivnosti (vsaj javno znanih ne) v smeri uporabe tega modela.

50 Kot napako v kodi razumemo npr. nepotrebno deljenje z 2 v oceni entropije (opombi 8 in 25) in povratno polnjenje zalog entropije ob praznjenju z vsako drugo besedo rezultata SHA-1 (poglavje 3.1.1., razlike v uporabi SHA-1).

51 Poglavje 3.1.1., dodajanje entropije v primarno zalogo entropije.

52 Testirali smo samo psevdo-naključne nize pridobljene na PC arhitekturi.

53 V vgrajenih sistemih bi morali najti dodatne vire entropije. Arhitekti vgrajenih sistemov lahko poskrbijo za inicializacijo generatorja psevdo-naključnih števil.

54 To je še posebej problematično v strežniških sistemih, ki imajo edini vir entropije v dostopih do diskov in pogojno še v mrežnih prekinitvah (odvisno od gonilnika, opomba 4). V strežniških sistemih lahko uporabimo zunanje strojne generatoje [13] naključnih števil.

55 Tudi za potrebe TLS/SSL [7] se uporablja predvsem vmesnik */dev/urandom*.

## Literatura


[1] D. R. Stinson, Cryptography, Theory and Practice, Third Edition, Chapman & Hall/CRC, Taylor & Francis Group, ISBN-10: 1-58488-508-4, ISBN-13: 978-1-58488-508-5, 2006.

[2] A. J. Menezes, P. C. van Oorschot, S. A. Vanstone, Handbook of Applied Cryptography, CRC Press, ISBN: 0-8493-8523-7, 1996.

[3] Tim Matthews, Suggestions for Random Number Generation in Software, RSA Laboratories Security Bulletins, RSA Laboratories, a division of RSA Data Security, 1996.

[4] Z. Gutterman, B. Pinkas, T. Reinman, Analysis of the Linux Random Number Generator, IEEE Symposium on Security and Privacy (S&P Conference 2006), pp. 371-385, ISBN: 0-7695-2574-1, 2006.

[5] Žiga Mahkovec, Analiza generatorjev psevdo-naključnih števil v operacijskem sistemu Linux, Seminarska naloga, Univerza v Ljubljani, Fakulteta za računalništvo in informatiko, 2004.

[6] M. Mackall, T. Ts'o, random.c, The Linux Kernel Archives, 2004, 2008.

[7] Eric Young, rand_unix.c, OpenSSL Project, 2007.

[8] A. Rukhin, J. Soto, J. Nechvatal, M. Smid, E. Barker, S. Leigh, M. Levenson, M. Vangel, D. Banks, A. Heckert, J. Dray, S. Vo, A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications, NIST Special Publications 800-22, 2001.

[9] Rajko Jamnik, Verjetnostni račun in statistika, Društvo matematikov, fizikov in astronomov SRS, 1986.

[10] B. Barak, S. Halevi, An architecture for robust pseudo-random generation and applications to /dev/random, 12th ACM-CCS (Conference on Computer and Communications Security, 2005), pp. 203-212, ISBN 1-59593-226-7, 2005.

[11] Charmaine Kenny, Random Number Generators: An Evaluation and Comparison of Random.org and Some Commonly Used Generators, The Distributed Systems Group, Computer Science Department, Trinity College Dublin, 2005.

[12] Alessandro Rubini, Linux Device Drivers, O'Reilly & Associates, ISBN: 1-56592-292-1, 1998.

[13] R. Davies, Hardware random number generators, 15th Australian Statistics Conference, 2000.

[14] http://en.wikipedia.org/wiki/Mersenne_Twister, 2007.

[15] http://en.wikipedia.org/wiki/Dual_EC_DRBG, 2008.

[16] Frandom, http://billauer.co.il/frandom.html.