Masarykova univerzita
Fakulta informatiky

Semínko generátoru náhodných čísel OS Linux při bootu

Bakalářská práce

Filip Škola
Brno, 2012

Prohlášení

Prohlašuji, že tato práce je mým původním autorským dílem, které jsem vypracoval samostatně. Všechny zdroje, prameny a literaturu, které jsem při vypracování používal nebo z nich čerpal, v práci řádně cituji s uvedením úplného odkazu na příslušný zdroj.

Vedoucí práce: Ing. Mgr. Zdeněk Říha, Ph.D.

Poděkování

Chtěl bych poděkovat Ing. Mgr. Zdeňku Říhovi, Ph.D. za odborné vedení práce a poskytnuté cenné rady. Dále děkuji společnosti Red Hat za možnost využití jejich počítačů pro potřeby této práce.

Shrnutí

Cílem práce je kvantitativní analýza semínka linuxového generátoru pseudonáhodných čísel se zaměřením na boot systému. Analýza je provedena na vybraných počítačových systémech v různých konfiguracích, zahrnuto je i prostředí virtuálních strojů.

Klíčová slova

Generátor pseudonáhodných čísel, semínko, boot, pool, pseudorandom number generator, random seed.

Obsah

1. Úvod

Náhodným číslem se nazývá číslo, jehož hodnotu nedokážeme předvídat ani vyvodit. Taková čísla nachází využití v širokém spektru vědeckých i nevědeckých odvětví lidské činnosti. Ve světě informačních technologií se náhodnosti využívá především v kryptografii, ale také v oblasti simulací nebo zábavy [1].

Generování náhodných čísel je netriviální problém, protože většina jevů kolem nás je do jisté míry předvídatelná. Tato míra je otázkou hlavně filozofickou. Připustíme-li ale, že podstatná část světa se řídí nějakými zákonitostmi, je vhodné dobře promyslet, který jev použijeme jako zdroj pro náhodná data. Předvídatelnost nacházíme především ve výpočetní technice, kde je determinismus velmi žádoucí. Není snadné „přesvědčit“ ten samý počítač, po kterém bylo vyžadováno přesné provedení nějakého kódu, aby vygeneroval něco zcela náhodného. Ve světě počítačů je generování náhodných čísel extrémně náročný problém.

Důležité je rozlišovat mezi výrazy „náhodné číslo“ a „pseudonáhodné číslo“. Náhodná čísla vznikají důsledkem jevů, které jsou nepředvídatelné. Mezi takové jevy patří například rozpad subatomických částic (za určitých podmínek) nebo vznik atmosférického šumu [2]. Posloupnost (binárních) náhodných čísel můžeme vyprodukovat i opakovanými hody mincí [3]. Naopak pseudonáhodná čísla vznikají jako důsledek událostí, jejichž přesný postup neznáme, ale nejedná se o objektivně náhodné jevy. Na deterministickém počítači dokážeme vytvořit pouze pseudonáhodná čísla (pokud nepoužijeme hardwarového generátoru náhodných čísel [4]). Pseudonáhodná čísla jsou generována algoritmem [5].

Tato práce se zabývá generátorem pseudonáhodných čísel. Pokud bude odkazováno na náhodná data či linuxový generátor náhodných čísel, budou tím myšlena pseudonáhodná data a linuxový generátor pseudonáhodných čísel.

Aby bylo možno tvořit pseudonáhodná čísla, je potřeba generátor inicializovat nějakou hodnotou. Právě tato hodnota spolu s algoritmem generátoru ovlivňují sílu (kvalitu) získaného pseudonáhodného čísla. Hodnota se nazývá semínko generátoru pseudonáhodných čísel (random seed). Data, která slouží jako semínko, jsou zpracována algoritmem generátoru a přidávána do poolu. Pool je

úložiště náhodnosti, ze kterého získáváme náhodná čísla. Ačkoliv algoritmus generátoru pseudonáhodných čísel je deterministický, je možné použít jako semínko hodnoty, které nejsou úplně předvídatelné (typicky jde o šum z prostředí nebo akce, ke kterým dá podnět uživatel). Získané pseudonáhodné číslo se pak jeví jako náhodné.

Zadáním této práce bylo provést kvantitativní analýzu dat vstupujících do generátoru pseudonáhodných čísel při zavádění operačního systému GNU/Linux, jedná se tedy o analýzu semínka. GNU/Linux je otevřený operační systém, který poskytuje funkce pro generování náhodných hodnot na úrovni jádra. Otevřenost systému byla velmi důležitá pro zadání práce, jejíž součástí bylo upravit jádro systému tak, aby zaznamenávalo informace o náhodných datech proudících do generátoru.

Tato práce je rozdělena na teoretickou část, ve které je zpracován stručný úvod do problematiky generování pseudonáhodných čísel. Dále je zde popsán linuxový generátor pseudonáhodných čísel spolu s jeho vstupním a výstupním rozhraním. Praktická část zahrnuje postupy použité pro úpravu linuxového jádra a následnou vlastní analýzu semínka v různých kombinacích hardwarových architektur a uživatelských prostředí. Na závěr jsou uvedeny speciální netestované případy a následuje shrnutí zjištěných výsledků.

Pro analýzu bylo použito prostředí komunitní linuxové distribuce Fedora 15 a systému z ní vycházejícího, Red Hat Enterprise Linux (RHEL) 6, spolu s upraveným jádrem verze 3.0.3. Výstupem této práce jsou data, která ovlivňují proces tvorby náhodných čísel v OS Linux, a jejich analýza. Součástí je i popis chování v prostředí virtuálních strojů (virtual machine, VM). Význam výstupních dat je především v možnosti jejich dalšího zkoumání. Jedním z možných navázání na kvantitativní analýzu, která byla provedena v této práci, by mohla být kvalitativní analýza s cílem určení kvality těchto dat. Bylo by možno posoudit vhodnost použití linuxového generátoru pseudonáhodných čísel ke specifickým účelům, zvláště pak v již zmíněné kryptografii.

2. Počítače a náhodnost

2.1 Úvod do problematiky

Počítače jsou velmi předvídatelná zařízení. Je tedy extrémně těžké vyprodukovat opravdu náhodná čísla na počítači, na rozdíl od pseudonáhodných čísel, která získáme algoritmem. Pro útočníka může být ale jednoduché zjistit sekvence čísel, která pseudonáhodný generátor produkuje. To dělá tato čísla nevhodná pro mnoho aplikací. Aby se útočníkovi co nejvíce znesnadnila cesta k uhodnutí čísel, která generátor produkuje, využívá se zdrojů náhodnosti z prostředí [5].

V následujícím textu je používáno výrazu **entropie**. Entropie v informatice není nepodobná fyzikální entropii. Klasické pojetí entropie v informatice je to, které formuloval Shannon. Podle něj je entropie statistická hodnota, která vyjadřuje, kolik informace je průměrně vyjádřeno v každém písmenu textu, psaném v přirozeném jazyce [6]. Častěji se toto interpretuje volně, jako míra překvapení či neurčitosti určité informace.

V této práci bude výrazu entropie využíváno ve spojitosti s nepředvídatelností pseudonáhodných čísel. Stejná terminologie je zavedena v komentářích kódu linuxového generátoru pseudonáhodných čísel [5]. Čím větší počet bitů entropie máme k dispozici, tím více čísel můžeme vytvořit. Pokud nemáme k dispozici entropii, nemůžeme vytvořit číslo, které by se podobalo náhodnému – o těchto číslech bude hovořeno jako o slabých či nekvalitních, o zbytku jako silných či kvalitních.

2.2 Generátory pseudonáhodných čísel

Generátor pseudonáhodných čísel je algoritmus, který se inicializuje počátečním stavem, semínkem. Pro každé semínko je generována vždy stejná sekvence čísel. Obecně platí, že generátor pseudonáhodných čísel je schopen generovat číselné sekvence, které jsou násobně delší, než vložené semínko [1] [3]. Pokud je zapotřebí mít co největší množství náhodných dat k dispozici, je třeba dodávat nová semínka. Náhodná data se pak generují do úložiště, takzvaného **poolu**. Čísla, která vybíráme z poolu, by měla mít rovnoměrné rozložení, aby byla stejná pravděpodobnost získání jakéhokoliv čísla z výstupního intervalu generátoru.

Jak již bylo zmíněno v úvodu, pseudonáhodná čísla nacházejí uplatnění v širokém rozsahu činností. Krom nejčastěji zmiňované kryptografie mezi ně patří vzorkování, numerická simulace, testování počítačových čipů na výrobní vady, rozhodování a mnohé další [1]. Podle typu činnosti je možné určit, jak kvalitní pseudonáhodná čísla potřebujeme. Často postačí pseudonáhodné hodnoty, které byly vytvořeny jednoduchým procesem. Například běžný algoritmus, který používá náhodná rozhodnutí k volbě dalších kroků. Zvláště v kryptografii je ale důležité, aby nešlo zpětně zjistit, jaká pseudonáhodná čísla byla použita v určitém okamžiku.

Na pseudonáhodná čísla, která jsou zamýšlena k použití v kryptografii, se klade více požadavků. Nejobecnější je požadavek na nerozlišitelnost od pravého náhodného čísla. Pseudonáhodné číslo by nemělo být rozlišitelné od náhodného čísla v menším, než polynomiálním čase [3].

S kvalitou generovaných čísel těsně souvisí ostatní prvky bezpečnosti generátoru. Nejběžnější způsoby, jakými je zajištěna bezpečnost generátorů, je použití šifer nebo hašovacích funkcí. Bezpečnost je pak závislá na bezpečnosti zvolené metody (hašovací či šifrovací funkce). Bezpečností je zde myšlena ochrana proti vyzrazení vnitřního stavu generátoru.

2.3 Problémy generování pseudonáhodných čísel

Jedním z typických (prvních) generátorů pseudonáhodných čísel je algoritmus Johna von Neumanna z roku 1946. Jeho funkčnost je zajištěna tak, že vstupní číslo, sloužící jako semínko, se umocní na druhou a následně se použije prostředních čísel výsledku jako výstupní náhodné hodnoty [7]. Mezi problémy tohoto přístupu se řadí časté opakování výstupních číselných sekvencí, některých hodnot velmi často (např. nulových). Tato negativní vlastnost generátorů již byla zmíněna jako nerovnoměrnost rozložení náhodné veličiny. Pokud pseudonáhodná čísla nejsou rozložena rovnoměrně, pak je již intuitivně patrné, že klesá podobnost s opravdu náhodnými čísly, která se snažíme simulovat.

Používání deterministických algoritmů pro vytvoření pseudonáhodných číselných sekvencí tam, kde potřebujeme náhodná čísla, má řadu úskalí. Nejběžnějším problémem, se kterým se setkáme, pokud potřebujeme kvalitní pseudonáhodná čísla, je nedostatek entropie v poolu. Generátor pak může dát

zpravidla na výběr buď méně kvalitní náhodné číslo, nebo uživatel musí počkat, než se entropie v poolu doplní.

Existuje mnoho různých možných problémů s výstupem generátoru. Lze vypozorovat charakteristiky (ve výstupních datech), jako například opakování čísel, předvídatelnost. Jakékoliv podobné chování má negativní důsledky.

3. Generování náhodných čísel v OS Linux

3.1 Operační systém s linuxovým jádrem

Linux je název jádra operačního systému, jehož vývoj započal v roce 1991 Linus Torvalds. Ačkoliv se původně jednalo o studentský projekt, v současné době je Linux jádro, které nachází široké uplatnění v nejrůznějších odvětvích informačních technologií a lidského života, nevyjímaje komerční využití. Linuxové jádro je kompletně otevřené, distribuuje se ve formě zdrojových kódů. Uživatel má možnost použít předkompilované jádro, které má typicky podporu běžného aktuálně používaného hardwaru. Druhá možnost je vytvořit jádro na míru vlastních potřeb, které bude obsahovat pouze vybrané funkce a jehož nastavení bude optimální pro zamýšlené využití. Z otevřenosti vyplývají ale i jiná pozitiva. Kdokoliv má možnost provést kontrolu či úpravu stávajícího kódu.

Linuxové jádro se používá jako základní stavební kámen takzvaných **linuxových distribucí**. Distribuce je kolekce softwaru, která tvoří operační systém. Ačkoliv hovorově se často mluví o „operačním systému Linux“, Linux je opravdu pouze jádro OS a přidružené aplikace nemají s tímto projektem mnoho společného. Zpravidla se jedná o softwarovou výbavu z projektu **GNU**, založeného v roce 1983 Richardem Stallmanem. Správný název OS s jádrem Linuxu a aplikacemi GNU je **GNU/Linux** [8].

3.2 Linuxový generátor pseudonáhodných čísel

Generování náhodných čísel v systémech GNU/Linuxu je prováděno na úrovni jádra. Z tohoto důvodu je v této práci hovořeno o linuxovém generátoru pseudonáhodných čísel. Zdrojem informací pro kapitoly zabývající se linuxovým generátorem náhodných čísel jsou převážně komentáře vložené do jeho zdrojového kódu. Zdrojový kód je dostupný online [5].

První implementace linuxového generátoru náhodných čísel pochází z roku 1994 a její autor je Theodor Ts'o. Jaderná implementace přináší několik výhod. Je jednoduché odchytávat uživatelské akce a akce hardwaru na nejnižší úrovni v systému. Je možné využít funkce generátoru v kódu jádra. Jaderný kód se provádí rychleji než aplikace běžící v uživatelském režimu. Nevýhodou stávající implementace je prakticky neexistující dokumentace, s výjimkou komentářů přímo v kódu. Těch je dostatek, nicméně velké množství patchů aplikovaných za poslední léta zhoršuje orientaci v něm.

Využívané zdroje náhodnosti jsou intervaly mezi stisky kláves, intervaly mezi přerušeními hardwarových zařízení a jiné události, jejichž chování je pro potenciálního útočníka obtížně vypozorovatelné. Náhodnost je přidávána do poolu, ve kterém jsou data promíchávána funkcí podobnou CRC (Cyclic Redudancy Check). Tento přístup byl zvolen hlavně kvůli celkové rychlosti. Po přidání náhodných hodnot jádro udržuje informaci o přibližném počtu bitů náhodnosti v poolu.

Při požadavku na náhodné číslo je vytvořen SHA (Secure Hash Algorithm) haš obsahu poolu. Hašovací funkce je použito, aby nedošlo k vyzrazení vnitřního stavu poolu. I když se má za to, že je výpočetně nemožné odvodit z výstupu SHA nějakou informaci o vstupu, tím že se pro hašování nepoužívá celý obsah poolu, je výstup zcela nepředvídatelný. Jako další opatření je snížen odhad o počtu bitů náhodnosti.

I když se počet bitů náhodnosti blíží k nule, je stále možné generovat náhodná čísla. V tomto případě je ale teoreticky možné, aby útočník odvodil další výstupy generátoru v závislosti na předchozích výstupech. I toto by však vyžadovalo úspěšnou kryptoanalýzu funkce SHA.

3.3 Výstupní rozhraní generátoru

Nejběžnější postup pro získání náhodného čísla v linuxovém systému je čtení souborů */dev/random* a */dev/urandom*. Jedná se o speciální soubory typu znakové zařízení, vytvořené v adresáři */dev* jádrem (respektive generátorem náhodných čísel).

Unixové operační systémy jsou založeny na pojetí souboru, který je schránkou pro informace, které jsou strukturovány jako sekvence bytů. Podle tohoto přístupu se zachází i se vstupně-výstupními zařízeními, které jsou v systému reprezentovány soubory [9].

Speciální soubory zařízení (device files) reprezentují většinu vstupně-výstupních zařízení v systému. Kromě jména mají ještě tři hlavní atributy: typ, major číslo a minor číslo. Typ určuje, jestli je soubor znakové nebo blokové zařízení. Major (hlavní číslo, většinou se nepřekládá) je číslo z rozsahu 1 až 255, které identifikuje typ zařízení. Většinou platí, že všechny soubory, které mají stejný major a stejný typ, disponují stejnou sadou operací nad souborem (*file_operations*, viz 4.3), protože je vytváří jeden ovladač [9].

Zařízení */dev/random* a */dev/urandom* reprezentují logická zařízení, protože se nejedná o fyzická zařízení (to by byl například hardwarový generátor náhodných čísel). Jde o znaková zařízení, což znamená, že přenáší sekvenčně. Obecně pro znaková zařízení platí, že přenáší v jedné operaci čtení/zápisu libovolně velké bloky dat [9].

Soubor */dev/random* slouží k získání kvalitních náhodných čísel. Při volání vrací nejvýše tolik bitů náhodnosti, kolik generátor odhaduje, že se nachází v poolu. Pokud se vyžadují další hodnoty, je prováděno blokující čekání do té doby, než jádro posbírá další data pro nová pseudonáhodná čísla.

Čtení ze souboru */dev/urandom* je neblokující, vždy okamžitě poskytne nějaké náhodné hodnoty. Jejich kvalita ale nemusí být taková, jako u zařízení */dev/random*. To ale neznamená, že by byl *urandom* nevyužitelný v praxi. Právě naopak, pro běžné využití nám postačí i jím poskytnuté náhodné hodnoty. Zařízení */dev/random* je doporučeno používat u kryptografických aplikací, kdy je míra náhodnosti relevantní. Pro ostatní účely se doporučuje číst z */dev/urandom*.

Zařízení */dev/random* používá minor 8 a */dev/urandom* minor 9 major čísla 1 (*/dev/mem*). Pokud nejsou přítomny v sytému, lze jej vytvořit pomocí příkazů:
mknod /dev/random c 1 8
mknod /dev/urandom c 1 9.

Mimo uvedené speciální soubory je možno využít rozhraní generátoru i na úrovni jádra. Pro tyto potřeby slouží funkce *get_random_bytes*.

*void get_random_bytes(void *buf, int nbytes);*

Počet potřebných bytů specifikovaných v proměnné *nbytes* bude umístěno do bufferu *buf*.

Z definice funkce je patrné, že si žádáme o určitý počet náhodných bytů. Žádaný počet ale nemusí být v poolu k dispozici – funkce pak neblokuje. Je tedy analogická k uživatelskému rozhraní *urandom*.

3.4 Vstupní rozhraní generátoru

Pro sběr nahodilých událostí v systému je linuxovým generátorem náhodných čísel využívána trojice funkcí:
void add_input_randomness(unsigned int type, unsigned int code, unsigned int value);
void add_interrupt_randomness(int irq);
*void add_disk_randomness(struct gendisk *disk);*.

Funkce *add_input_randomness* vyžívá časování přerušení vstupní vrstvy spolu s údajem o typu události. Tento zdroj přidává do poolu data proudící typicky z klávesnice a myši (nebo jiného polohovacího zařízení např. u notebooků).

Dalším zdrojem náhodnosti jsou ostatní hardwarová přerušení. Ty lze přidat pomocí funkce *add_interrupt_randomness*. Je důležité si uvědomit, že ne každé přerušení v sytému je dobrým zdrojem náhodných dat. Například přerušení časovače jsou velmi pravidelná a tudíž předvídatelná a nevhodná. Příklad vhodnějšího přerušení je třeba to, které pochází ze síťové karty. Přidání přerušení do poolu je podmíněno speciálním příznakem u definice přerušení. Takto označené přerušení by mělo zajišťovat vhodnost použití jako zdroj entropie.

Zdroj *add_disk_randomness* používá časové rozvrstvení práce disku. Ani tady nelze zaručit, že budou poskytnuty vhodné hodnoty. Kupříkladu vysokorychlostní SSD (Solid-state Drive) disky mají velmi nízké a konzistentní hodnoty přístupového času (seek time) k datům. Takto poskytnutá data pro semínko mohou být pravidelná a nevhodná.

Zápis do speciálního souboru generátoru v systému *dev* způsobí přidání hodnot do poolu, nicméně nezvýší se odhad celkového počtu bitů v poolu. Zabraňuje se tak znehodnocení poolu předvídatelnými hodnotami (bezpečnostní opatření). Počet bitů lze zvýšit pomocí systémového volání *ioctl*, ke kterému je potřeba privilegovaného režimu oprávnění.

Volání *ioctl* bylo zavedeno pro případy, kdy nestačí základních funkcí (otevření, čtení, zápis) pro práci se zařízením. Toto systémové volání je specifické vzhledem k zařízení, se kterým je svázáno. Například u CD-ROM mechanik je možno s ním nastavovat hlasitost zvuku nebo vysunout médium [9].

3.5 Zajištění náhodnosti po startu

Iniciální stav každého generátoru pseudonáhodných čísel je pevně nastavený v jeho kódu. Linuxový generátor sbírá náhodné hodnoty již v raných fázích bootu, kdy se provádí velké množství operací. Ty jsou ale většinou předvídatelné, zvláště pokud systém startuje bez účasti člověka. To snižuje hodnotu skutečné nepředvídatelnosti bitů v poolu pod odhad generátoru. Aby se zamezilo tomuto jevu, je silně doporučeno ukládat stav poolu mezi starty systému. Toho je možno docílit jednoduchými skripty, který jsou v dnešní době součástí všech běžných distribucí. Kódy skriptů jsou k dispozici v příloze A (převzaty ze zdrojového kódu [5]).

Tyto skripty by měly být součástí startu a vypínání každého systému. V nejběžnější implementaci initu, System V, jsou k nalezení v adresářích /etc/rc.d/init.d. Existují ale i systémy, ve kterých tyto skripty nejsou přítomny (protože není k dispozici disk, na který by se uložil stav poolu). Mezi ně se řadí například routery běžící na linuxovém systému.

Hlubší analýzu vnitřní funkcionality linuxového generátoru, stejně tak i popis používaných algoritmů a rozbor bezpečnostních problémů, lze najít v práci [10].

4. Úprava jádra pro analýzu

4.1 Provedené úpravy

Aby mohla být provedena analýza hodnot, které slouží jako semínko generátoru náhodných čísel, bylo potřeba modifikovat jádro tak, aby data mohla být průběžně zaznamenávána. Implementace potřebné úpravy sestávala z návrhu datové struktury, do které se data ukládají (tato struktura se nachází v paměťovém prostoru jádra) a rozhraní mezi jádrem a uživatelským paměťovým prostorem, aby data mohla být přečtena a uložena na permanentní datové úložiště.

Relativní cesty použité v této kapitole (např. *drivers/char/random.c*) odkazují na soubory ve zdrojových kódech jádra.

4.2 Datová struktura pro záznam dat

Jelikož data, která je třeba uchovávat, tvoří jen posloupnost čísel spolu s textovým popisem, bylo zvoleno obyčejného spojovaného seznamu (linked-list). Jeho deklarace byla umístěna do hlavičkového souboru *include/linux/kernel.h*.

```
typedef struct randlistmsg {
    int id;
    char read;
    char * text;
    struct randlistmsg * next;
} randlistmsg;
```

Položka *id* označuje pořadové číslo zprávy, *read* je příznak přečtenosti, text je samotný text položky. Ukazatel na další položku seznamu se nazývá *next*.

Struktura je inicializována v souboru *drivers/char/random.c*. V tomto souboru se nachází i funkce pro zápis do seznamu tvořeného strukturou *randlistmsg*. Tato funkce, nazvaná *write_to_randlist*, je volána pokaždé, když se přidávají nějaká data do poolu.

Funkce *write_to_randlist* přijímá pouze jeden parametr, a to zprávu, která se bude zapisovat do seznamu. Před tuto zprávu se umístí informace o času

uplynulého od bootu systému (uptime) a pořadové číslo zprávy. Uptime se vypočítává z počtu uplynulých jednotek *jiffies* od startu systému. *Jiffies* označují časový úsek mezi dvěma tiky jádra. Vynásobení s interní frekvencí jádra je převádí na sekundy. Informace o času, pořadové číslo a vlastní text záznamu se zformátují pomocí funkce *snprintf*, která vrací počet znaků výsledného řetězce. Tento údaj umožní dynamicky alokovat přesně tak velkou oblast paměti, jaká je potřeba pro uložení formátované zprávy. Před zápisem nové položky do seznamu je ještě vynulován příznak přečtenosti záznamu.

Aby nedocházelo ke konfliktu mezi čteními a zápisy do seznamu, používá se proměnná *randlist_open* s počáteční hodnotou 0, která je inkrementována rutinou, která se seznamem právě pracuje. V případě, že nastane stav, kdy se ze seznamu čte/zapisuje a zároveň vznikl požadavek na další čtení/zápis, provádí se čekání.

4.3 Rozhraní pro kontakt s uživatelským paměťovým prostorem

Pro implementaci rozhraní mezi paměťovým prostorem jádra a uživatelských aplikací se nabízelo několik možností, jejichž společným znakem bylo vytvoření souboru v některém ze souborových systémů pro speciální účely (pseudo-filesystem). V současných linuxových distribucích se typicky nacházejí tři takové systémy: *proc*, *sys* a *dev*. Souborový systém *proc* se ukázal být nevhodný pro účely této práce. Systému *proc* se doporučuje používat pouze pro účely, které mají souvislost s běžícími procesy [11]. Systém *sys* je pro tento účel vhodnější. Motivací pro vznik *sys* byl mimo jiné fakt, že v *proc* se nacházelo příliš mnoho s procesy nesouvisejících informací. Třetí možností byl systém *dev*. Tento souborový systém se v linuxových (a jiných unix-like) operačních systémech používá pro reprezentaci zařízení [9][11]. Krom souborů, které zastupují reálný hardware, se zde nacházejí virtuální zařízení, mezi které patří i rozhraní generátoru náhodných čísel */dev/random* a */dev/urandom*. Pro implementaci byla zvolena třetí možnost, systém *dev*.

Zařízení v systému *dev* se definuje pomocí struktury *file_operations*, definované v souboru *include/linux/fs.h*. Tato struktura uchovává ukazatele na funkce definující základní operace nad novým souborem. Typicky se jedná o otevření a zavření souboru, čtení a zápis. Pro vytvoření rozhraní mezi strukturou náhodných dat a uživatelským prostorem bylo třeba implementovat především čtení ze souboru. Zápis do souboru v *dev* má za důsledek přenos dat

z uživatelského paměťového prostoru do prostoru jádra. Tato funkcionalita je v navrhovaném rozhraní použita pro rozdělení seznamu náhodných dat podnětem zvenčí (například zpráva o úspěšném ukončení bootu systému).

Zařízení zadefinované ve *file_operations* je potřeba zaregistrovat do *dev*. Ačkoliv pro vývoj a testování prvních fází implementace bylo použito jaderných modulů, finální zhotovení je součástí kódu jádra. Zdrojový kód rozhraní v *dev* byl proto umístěn do zdrojového souboru, ve kterém jádro sdružuje i další znaková zařízení, která jsou umístěna v *dev – drivers/char/mem.c*. Tento soubor obsahuje strukturu *memdev*, ve které jsou tato zařízení definována. Do ní bylo přidáno zařízení reprezentující rozhraní pro datovou strukturu generátoru a toto zařízení bylo pojmenováno *randlist* (ze slov random – náhodný, a list – seznam).

[13] = { "randlist", 0666, &randlist_fops, NULL },

První položkou je název souboru, druhou tvoří implicitní přístupová práva k objektu. Předposlední je ukazatel na strukturu *file_operations*. Poslední položka je ukazatel na strukturu rozšířené funkcionality zařízení, kterou nebylo potřeba pro *randlist* definovat.

Funkce obstarávající otevření a uvolnění souboru jsou triviální, jediná jejich úloha je řídit přístup k souboru a zabránit konfliktu čtení a zápisů.

Čtení ze souboru zprostředkovává uživateli kompletní výpis (zatím nepřečtených) položek seznamu posbíraných náhodných dat. Struktura se čte záznam po záznamu tak, jak je obvyklé u spojovaných seznamů. Data je třeba nejdříve přesunout z paměti jádra do paměti aplikací, odkud jsou teprve zobrazena. To obstarává funkce *put_user*, která znak po znaku kopíruje text ze záznamu položky seznamu do pole typu *char*. Ukazatel na toto pole (který se nachází mimo paměť jádra) poskytuje jádro při volání funkce pro čtení speciálního souboru. Toto pole (označené jako *buffer*) má omezenou kapacitu, po jejímž překročení je třeba čtení pozastavit. Pokud není čtení ukončeno, jádro volá funkci znovu, až do konce procesu čtení. Z tohoto důvodu má každý záznam v seznamu položku, která označuje příznak přečtenosti. Pro každou operaci čtení se vypisuje část seznamu od poslední přečtené položky do konce seznamu.

Zápis do souboru sestává především z kopírování dat, která jsou určeny k zápisu, do paměťového prostoru jádra. Poté je provedeno dodatečné formátování, které je velmi podobné formátování při vkládání dat o náhodných událostech. O kopírování dat se stará funkce analogická k *put_user*, *get_user*. Jádro opět poskytuje *buffer* (pole znaků), ve kterém je nyní uložen uživatelský vstup. Vložená data jsou uvozeny údajem o čase a klíčovým slovem *msg*, až pak je přítomna samotná zpráva, která měla být vložena. Tohoto přístupu bylo zvoleno především pro zjednodušení pozdějšího zpracování.

4.4 Zdroje náhodných dat

Klíčové pro sběr informací o náhodných datech proudících do generátoru bylo rozmístění volání funkce pro zápis do seznamu hodnot. Linuxový náhodný generátor poskytuje na úrovni jádra tři funkce pro přidávání dat do poolu, *add_input_randomness*, *add_interrupt_randomness* a *add_disk_randomness*.

Vzhledem k tomu, že všechny tyto zdroje posílají do poolu přidávanou hodnotu dál funkci *add_timer_randomness*, byla upravena až tato funkce. Konkrétně tak, že jí byl přidán další parametr, ve kterém se specifikuje zdroj do poolu přidávaného čísla. Toto opatření mimo jiné zajišťuje kontrolu úplnosti zaznamenávaných dat na úrovni kompilace. Kdyby v jádře docházelo k volání této funkce, které nebylo modifikováno (přidaným parametrem), kompilátor by na to upozornil.

Samotná *add_timer_randomness*, která sjednocuje tři jmenované zdroje náhodných dat, nejprve volá funkci, ve které se přidávaná data promíchají s informacemi o aktuálním čase. Až vzniklé číslo se přidává do poolu. Po přidání jádro určuje, kolik bitů entropie nové hodnoty generují. Tento odhad je klíčový, pokud by číslo nemělo způsobit nárůst entropie, není zvýšen údaj o počtu bitů v poolu. Z toho důvodu se nepřidává ani do seznamu *randlist*. Jeho hodnota je pro celkovou entropii poolu bezvýznamná.

Mimo funkce na úrovni jádra je možné zvýšit počet bitů entropie náhodného generátoru pomocí zápisu do */dev/random* (resp. *urandom*) a volání funkce *ioctl*. Tohoto přístupu lze využít například v případech, kdy běžně používané zdroje entropie nepřinášejí uspokojivý počet bitů – například u bezdiskových stanic. Této problematice je věnována kapitola 6.2. Počty bitů přidávaných do poolu pomocí zápisu do */dev/random* jsou také monitorovány a přidávány do seznamu.

4.5 Formát výstupu

Na výstup byl kladen především požadavek snadné zpracovatelnosti dalšími nástroji. Každý řádek výstupu má pevnou strukturu s následujícím formátem.

čas id [zdroj] num *hodnota* bits *bity*
Čas reprezentuje sekundy a mikrosekundy od bootu jádra, *id* je pořadové číslo zprávy, položka *zdroj* je právě jedna ze tří *disk/input/irq*. *Hodnota* je údaj, který se bude vkládát do poolu a pod položkou *bity* nalezneme odhad nárůstu počtu bitů entropie po přidání hodnoty.

Výpis seznamu *randlist* získáme čtením souboru */dev/randlist*.

5. Analýza výstupu

5.1 Podmínky testování

Zadáním práce bylo provést kvantitativní analýzu dat, která slouží jako semínko generátoru. Data byla zkoumána na různých hardwarových architekturách, konkrétně x86-64 (32/64-bitová AMD architektura), x86 (klasická 32-bitová od Intelu), PPC64 (Power PC 64-bitová) a S390 (IBM architektura). Na vybraných systémech se prováděla i analýza v prostředí virtuálních strojů.

Systémy byly konfigurovány pro boot do textového prostředí. Mimo srovnávací testy architektur proběhlo testování různých příkladů užití systému (tyto testy byly provedeny na architektuře x86-64). Cílem bylo odhalit změny ve stavu entropie při typických scénářích využití počítače.

Aby analýza věrněji reflektovala využití takových systémů v praxi, nebylo zkoumání zaměřeno striktně na čas bootu. Začátek souboru zkoumaných hodnot je vymezen začátkem startu systému. První zkoumaná hodnota je tedy první hodnota směřující do poolu. Hodnota, která je uvažována jako poslední, je ta, která je přidána nejvýše minutu po ukončení zaváděcích skriptů distribuce.

Rozšíření časového úseku analýzy o první okamžiky běhu systému pomáhá vzít v úvahu i typické akce na různých systémech. Například bity entropie, které mají zdroj v uživatelském vstupu (klávesnice, myš), by hrály daleko menší roli, kdyby byl časový úsek analýzy omezen výzvou k přihlášení do systému. V praxi není obvyklé využít náhodných hodnot dříve, než jsou provedeny jisté akce, pří kterých se použijí vstupní zařízení (přihlášení do systému, start potřebných programů, zadání vstupních hodnot programu).

Distribuce Fedora využívá doporučeného postupu ukládání (při vypínání systému) a následného obnovení (při zapínání) hodnot poolu do/z souboru. Ve výsledném seznamu hodnot je toto obnovení označeno jako zdroj *file*. Součástí mechanismu ale není zvýšení hodnoty počtu bitů entropie v jádře.

Počty zjištěných náhodných bitů je třeba brát orientačně. Ačkoliv běh systému do ukončení bootu je do značné míry předvídatelný, protože se startují stále ty stejné služby, záleží na mnoha proměnných, jako například velikosti a počty

souborů, se kterými zaváděcí skripty pracují. Po určitém časovém úseku používání některého z testovaných systémů by se počet bitů náhodnosti přidávaný do poolu lišil.

Základní informace o jednotlivých hardwarových architekturách (výrobce, rok uvedení na trh a podobné) jsou čerpány z anglické Wikipedie. Informace o hardwaru a konfiguraci testovaných systémů společně s výstupním souborem *randlist* lze nalézt mezi elektronickými přílohami.

5.2 x86-64

Architektura, označovaná jako x86-64 nebo amd64, je patrně nejrozšířenější architekturou mezi osobními počítači a notebooky. Jedná se o přímého nástupce intelovské 32-bitové architektury procesorů, která však byla vytvořena konkurenční firmou AMD. Později byla adaptována i ostatními výrobci CPU.

Testování na této architektuře proběhlo s jádrem kompilovaným právě pro tuto platformu, ne s jádrem pro obecnou architekturu 386. Bylo provedeno několik zkoumání na třech různých systémech. První analyzovaný systém je provozován na laptopu s grafickým prostředím. Druhý systém běží na serverovém počítači bez klávesnice, bylo použito vzdáleného přístupu. Třetí systém je virtualizovaný.

5.2.1 x86-64 notebook

Tento systém je typický zástupce počítačového systému v domácím uživatelském prostředí. Jedná se o notebook značky Lenovo s klasickým rotačním diskem a zabudovaným touchpadem. Byla provedena analýza běhu při bootu do grafického prostředí. V grafickém prostředí je typicky používáno myši a jiných polohovacích zařízení častěji, než v prostředí textovém. Vstupní zařízení byla nezanedbatelným zdrojem náhodných bitů.

Poměr získaných náhodných bitů z disku a ze vstupních zařízení je zhruba 68 % ku 32 %. Počet získaných bitů je 19650. Samotný uživatelský vstup měl za následek 6386 bitů do poolu, lze tedy předpokládat, že při standardním využití klávesnice a myši na tradičním desktopovém systému se bude do poolu přidávat dostatečné množství hodnot, aby pokrylo potřeby uživatele (systému). V tomto analyzovaném běhu nedošlo k přidání náhodných bitů ze zdroje přerušení. Nicméně i tak lze hovořit o nadprůměrně velkém počtu hodnot.

5.2.2 x86-64 server

Nejvýraznější charakteristika dat získaných ze serverového systému je, že zdrojem veškerých náhodných bitů je práce disku. Systém byl ovládán přes sériovou konzoli, ze které se náhodné hodnoty nezískávají, stejně jako se nezískávají pomocí práce přes protokol *ssh*. Celkový počet bitů přidaných během bootu a první minuty po něm je 2320. Z toho 1789 bitů bylo přidáno přímo během bootu, zbývajících 531 po něm. V tomto případě nemá příliš smysl z toho vyvozovat nějaký důsledek. Po přihlášení nebyly explicitně spuštěny žádné programy (jako například grafické desktopové prostředí) a v potaz nebyly brány ani znaky napsané při procesu přihlašování se do systému.

Sériová konzole určená k ovládání systému je používána stejně jako klávesnice u lokálního přístupu. Bylo by dobré zjistit možnosti přidání tohoto vstupního zařízení jako zdroje entropie pro generátor.

5.2.3 x86-64 virtualizovaná

Podmínky analýzy prováděné na architektuře x86-64 ve virtualizovaném prostředí byly velmi podobné těm u serveru na stejné architektuře. Jediný podstatný rozdíl tvoří klávesnice přímo „připojená“ k virtuálnímu stroji. A pak samozřejmě rozdíl mezi reálným hardwarem a virtualizovaným prostředím. Byla zvolena metoda KVM (Kernel-based Virtual Machine) emulovaná softwarovým nástrojem QEMU.

Počet celkově přidaných bitů je nižší než u reálného hardwaru. Celkově jich během celého testování bylo do poolu přidáno 1225. 1078 bylo způsobeno diskovými přerušeními, zbývajících 47 tvoří kódy kláves stisknutých při přihlašování (záleží pochopitelně na délce přihlašovacích údajů, v tomto případě jméno a heslo tvořilo 10 znaků celkem).

Vzhledem k tomu, že virtuální prostředí je ideální pro experimenty s modifikací hardwaru, byl k virtuálnímu počítači připojen druhý disk, s částečně zaplněným *ext3* oddílem. Ten byl připojen jako *home* oddíl. Takový disk vyprodukoval 27 přerušení o celkové hodnotě 76 bitů. V případě že bychom tedy měli připojený druhý disk jako *home*, na který by se během bootu příliš nepřistupovalo, pomoc s generováním náhodných bitů by byla zanedbatelná.

5.3 x86

Tato architektura zahrnuje procesory založené na Intelu 8086. Ten byl vyroben roku 1978. Jedná se o předchůdce kategorie x86-64, do nedávna typického zástupce na poli desktopových a notebookových systémů. Vznikl jako 16-bitové rozšíření procesoru Intel 8080. Architektura byla adaptována řadou dalších výrobců a stal se z ní průmyslový standard. Jedná se většinou o 32-bitové procesory (počínaje CPU Intel 80486DX).

Jako zástupce systému s klasickým 32-bitovým procesorem od Intelu byl vybrán server s dvěma procesory třídy Intel Pentium III. Systém disponuje dvěma disky. Přímo během bootu bylo vygenerováno 2671 bitů, během první minuty běhu systému dalších 623. Celkově se tedy jedná o 3294 bitů náhodnosti, všechny byly přidány jako důsledek práce disku, protože k systému bylo přistupováno vzdáleně.

Jednalo o starší počítač, nicméně ani v tomto případě nebylo využito přerušení jako zdroje náhodnosti. U architektur x86 a x86-64 lze očekávat podobné výsledky, protože portfolio hardwaru vyskytujících se u těchto architektur je velmi podobné.

5.4 S390

Architektura System/390, zkracovaná jako S390 (plným názvem Enterprise System Architecture/390), je poslední 32-bitová mainframová architektura od IBM, která byla představena v roce 1990. Ačkoliv je možno provozovat linuxové systémy na této platformě přímo, není to příliš obvyklé. Tento přístup nedokáže plně využít potenciál strojů, které jsou založeny na této architektuře. Z tohoto důvodu se často využívá virtualizace.

Stroj, na kterém probíhalo testování v rámci architektury s390x, je mainframe IBM System z10. Linuxový systém na něm provozovaný je konfigurován tak, aby nabídl více nezávislých virtuálních počítačů na jednom reálném zařízení. Zde bylo nutné instalovat systém Red Hat Enterprise Linux, Fedora nemohla být použita. Tato architektura není běžná, ale vzhledem k tomu, že byla možnost na ní analýzu provést, bylo tak učiněno. Výsledky mohou sloužit pro srovnání s „běžným“ hardwarem.

K tomuto virtuálnímu stroji bylo přistupováno pomocí *ssh* konzole, nebylo tedy generováno žádných *input* přerušení. Veškeré zdroje náhodných dat jsou

opět z pevného disku. Systém disponuje čtyřmi disky, které byly zapojeny pomocí LVM (Logical Volume Management). Všechny disky generovaly nějaké nezanedbatelné hodnoty (zastoupení jednotlivých disků v celkovém počtu hodnot bylo 20 %, 43 %, 11 % a 26 %).

Celkově bylo vygenerováno 894 bitů entropie, což není příliš mnoho. Jedná se o vcelku exotický systém, který je navíc provozován v netradičním virtuálním prostředí. Nebylo možno zkoumat tento systém z pohledu reálného hardwaru, k dispozici byl pouze pohled „zevnitř“ virtuálního stroje. Příčina nedostatečné náhodnosti v systému může být nevhodně zvolená konfigurace disků. Další zkoumání tohoto systému by mohlo přinést zajímavé výsledky.

5.5 PPC64

Architektura PowerPC (plným názvem Performance Optimization With Enhanced RISC – Performance Computing) je architektura z roku 1991, která vznikla společnou prací firem Apple, IBM a Motorola. Jedná se o procesory typu RISC (Reduced Instruction Set Computing), které jsou vyráběny jako 32 i 64-bitové. Pro tuto architekturu bylo portováno více distribucí, včetně Fedory do verze 12. Avšak připravuje se i port pro verzi 16. Pro testování byla použita distribuce RHEL 6, instalován byl na systému s 64-bitovým procesorem PPC.

Testovaný systém je IBM server s jedním pevným diskem a osmi procesory typu POWER6. K systému byl přístup pomocí sériové konzole a v systému se nenacházel žádný hardware generující vhodná přerušení, všechny hodnoty jsou proto opět důsledkem práce disku. Náhodných bitů bylo přidáno pouze 239. Vzhledem k tomu, že se jedná o další systém, ve kterém hrálo roli jen použití disku jako zdroje entropie, je těžké vytvořit nějaký závěr. Malá hodnota počtu bitů může být způsobena čistou instalací systému (méně dat na disku, menší fragmentace) nebo použitým diskem. Podstatné je, že systém nedisponuje vhodnými přerušeními pro zdroj náhodnosti. Při vhodné konfiguraci disku lze předpokládat srovnatelné množství bitů jako u ostatních architektur.

5.6 Boot z flash disku

Speciální případ, kdy bylo bootováno z flash disku, byl proveden na notebooku HP Compaq s procesorem architektury x86-64. Pro účely testu byl vyjmut pevný disk, aby případ simuloval použití stanic bez rotačního disku. Boot

systému byl značně pomalý (bootovalo se jen do textového režimu), což mělo za následek vygenerování podstatně více náhodných hodnot, než u systémů s tradičními disky. Celkově bylo během bootu nasbíráno 8964 náhodných bitů, z toho 1004 z nich po ukončení zaváděcích skriptů distribuce. 98 % tvořila práce (flash) disku.

U tohoto případu by bylo vhodné provést kvalitativní analýzu. Ačkoliv náhodných hodnot bylo posbíráno mnohonásobně (téměř 10x) více, než u systémů s tradičním diskem (při bootu do textového prostředí), nelze vyloučit, že se jedná o nevhodná data. Přístupové doby na flash paměti jsou konstantní a to by mohlo způsobit pravidelnost rozložení takto generovaných náhodných hodnot.

5.7 Shrnutí

Následující tabulka shrnuje výsledky měření. Na systémech, kde nebylo možno určit čas ukončení bootu (nemožnost konfigurace), či kde to nemělo smysl (boot do grafického prostředí), je místo hodnoty pomlčka.

	bity přidané	během bootu	eventů	input	disk
X86-64 server	2320	1789	1099	0	1099
X86-64 notebook	19650	-	8935	1972	6963
X86-64 KVM	1225	700	490	27	463
X86-64 KVM /home	1192	666	471	22	449
X86	3294	2671	1431	0	1431
S390	894	-	270	0	270
PPC64	239	-	95	0	95
Flashdisk	8964	7960	2211	24	2187

Tabulka 1 – přehled naměřených hodnot

Položka „bity přidané“ shrnuje celkový počet bitů entropie přidaných do poolu. Počet bitů z celkového počtu, který byl přidán před ukončením procesu bootování, je uveden v položce „během bootu“. Pod označením „eventů“ je uveden počet jednotlivých událostí, které měly za následek přidávání hodnot do poolu. Tyto hodnoty jsou pak rozděleny na „input“ (přerušení vstupní vrstvy) a „disk“ (bity přidané prací disku). V průběhu testování se nevyskytlo žádné přerušení z jiného hardwaru.

6. Netestované případy

6.1 Příčiny nedostatku entropie

Celá analýza byla zaměřena na tradiční linuxový systém s obyčejnou distribucí. Linuxové jádro se ale stává základem mnoha jiných systémů, zvláště pak embedded a mobilních zařízení (embedded systémy, které jsou někdy překládány jako vestavěné systémy, jsou jednoúčelové systémy, které slouží k ovládání nějakých zařízení). V jiných podmínkách je možné, že jádro neposbírá dost dat k naplnění poolu. Může to být způsobeno například absencí vstupu nebo tradičního diskového zařízení. V případě, že je zapotřebí použít generátor pseudonáhodných čísel a není použita vlastní implementace, je několik možností, jak doplnit pool generátoru.

6.2 Alternativní zdroje získávání entropie

Jednou z možností je modifikace jádra. Z provedené analýzy je patrné, že většina náhodných bitů je závislá na uživatelském vstupu a práci disku. Pokud potřebujeme náhodná čísla v zařízení bez možnosti získat data takto, může se provést ad-hoc úprava jádra, která bude využívat ověřeného zdroje náhodnosti. Implementace přímo do jádra je sice nejúčinnější metoda, jak náhodné jevy využít, ale poměrně neflexibilní. Zvláště pak z toho důvodu, že mezi linuxovou komunitou je běžné kompilovat jádro podle vlastní potřeby. Uživatel by pak musel na daném systému zajišťovat úpravu jádra sám.

Druhá možnost je zajistit sběr na úrovni userspace. Je možné použít souboru */dev/random*, do kterého lze zapsat soubor náhodných hodnot, které se pak přidají do poolu, a voláním *ioctl* následně zvýšit hodnotu počtu bitů v něm. Tato úprava by byla implementována nejspíš jako démon, což je program, který běží na pozadí. Nebylo by pak obtížné programovat jej flexibilně, pro více různých zařízení.

Jako příklad poslední uvedené možnosti může sloužit distribuce pro routery. Ačkoliv router běžně nemá vstupní zařízení ani disk, disponuje jiným zdrojem entropie – síťovým provozem. Zvláště bezdrátová zařízení mohou posloužit jako dobrý zdroj náhodnosti. Praktická aplikace toho řešení byla provedena například v programu *iwrandom* [12].

Další zdroj by mohl být zvukový šum přijímaný mikrofonem (který se nachází

na většině notebooků a mobilních zařízeních). Zde by bylo nejspíš vhodné zvuk nejprve (alespoň na základní úrovni) analyzovat, aby se zabránilo přidávání podobných bitových vzorů. Nicméně i tak by se jednalo o užitečný zdroj entropie, protože by zprostředkovával velké množství dat. Podobně by mohla sloužit i webkamera, která se stává tradiční součástí nových notebooků.

7. Závěr

Z uvedených příkladů je patrné, že napříč různými systémy s tradiční linuxovou distribucí není nouze o náhodné bity. Primární pool má velikost 512 bytů, sekundární 128 bytů. Celková kapacita je tak 5120 bitů. Na systémech, ke kterým se přistupuje lokálně, je již během bootu a přihlášení do systému do poolu přidáno mnohonásobně více, než je tato hodnota.

Problém by mohl eventuálně nastat u serverových a jiných vzdálených systémů, kde se bity generují pouze z práce disku. Není ale obvyklé takové systémy vybavovat pouze SSD disky, takže lze očekávat, že ani u těchto systému nebude nouze o náhodné hodnoty. S rostoucí dobou běhu systému jsou navíc zajišťovány stále nové hodnoty, jejichž počet bude růst s prací disku.

Reálná hrozba nedostatečně naplněného poolu by mohla nastat u mobilních a embedded zařízení (například routerů) bez rotačního disku a/nebo vstupního zařízení. Zde je potřeba řešit nedostatek individuálně, podle typu zařízení, jak bylo nastíněno v kapitole 6.2.

Volba architektury nemá velký vliv na počet dat sloužících jako semínko. U systémů záleží především na tom, jestli je k dispozici nějaké vstupní zařízení, které poslouží jako zdroj pro semínko. U žádného z testovaných systémů se pro semínko nepoužilo hardwarových přerušení. Pokud tedy není k dispozici vstupních zařízení, nejvíce ovlivní počet náhodných dat, sloužících jako semínko, konfigurace pevných disků.

Je třeba brát v úvahu i obnovení obsahu poolu ze souboru, které se provádí během každého bootu. Odhad počtu bitů entropie v poolu se tím nezvedne, ale hodnoty jsou reálně přidány. Pokud je čteno */dev/urandom*, jsou k dispozici kvalitnější náhodná čísla, než by se dalo podle stavu poolu čekat. Nijak to ale neovlivní čtení z */dev/random*, které se řídí odhadem entropie v poolu (když jí není dostatek, čeká se). Není-li potřeba opravdu kvalitních náhodných čísel, je dobré zvážit použití *urandom* místo *random*.

Z analýzy dále plyne, že největší množství dat, sloužících jako semínko, pochází z práce disku a vstupních zařízení. Jádro však disponuje funkcí *add_interrupt_randomness*, která slouží k přidávání náhodnosti z hardwarových přerušení. Je otázkou, proč se tato funkce nevolá, resp. proč se

neklasifikují přerušení jako vhodný zdroj náhodnosti. V testovaných systémech se vyskytovalo portfolio různého hardwaru, ale ani tak se žádného z nich nevyužilo. Logickým pokračováním této práce by mohla být revize některých jádrem podporovaných zařízení a úprava stávajícího kódu tak, aby se použilo přerušení z vhodných ovladačů. Především hardware často nasazovaný do současných počítačových sestav by mohl posloužit jako nový zdroj cenných bitů entropie.

Dalším vhodným pokračováním této práce by byla kvalitativní analýza semínka, jak u typických analyzovaných případů, tak i u méně tradičních počítačových systémů. Při bootu z flash disku bylo do poolu přidáno velké množství hodnot, ovšem semínko bylo z podstatné části tvořeno prací flash disku. Přístupové doby u nerotačních disků bývají konzistentní a lehce předvídatelné, je tedy možné, že je těmito hodnotami obsah poolu znehodnocen. Vhodné by bylo také analyzovat hodnoty pocházející z virtuálního prostředí.

Seznam literatury

[1]
LAGARIAS, J.C. Pseudorandom Number Generators in Cryptography and Number Theory. In POMERANCE, C., GOLDWASSER, S., et al. *Cryptology and computational number theory* [online]. Boulder, Colorado: American Mathematical Society, 1990 [cit. 29. prosinec 2011]. Dostupné na: <http://www.worldcat.org/title/cryptology-and-computational-number-theory-held-in-boulder-colorado-august-6-7-1989/oclc/174465175>.

[2]
HAAHR, Mads. *Introduction to Randomness and Random Numbers* [online]. 2011 [cit. 27. 4. 2011]. Dostupné na: <http://www.random.org/randomness/>.

[3]
GOLDREICH, O., KRAWCZYK, H., et al. On the Existence of Pseudorandom Generators. *Lecture Notes in Computer Science*. 1990, roč. 17, č. 403, s. 146—162.

[4]
JUN, B., KOCHER, P. *The Intel Random Number Generator* [online]. Cryptography Research, Intel, 22. duben 1999 [cit. 29. prosinec 2011]. Dostupné na: <http://www.cryptography.com/public/pdf/IntelRNG.pdf>.

[5]
TS'O, Theodore. *random.c: A strong random number generator* [online]. 2011 [cit. 29. prosinec 2011]. Dostupné na: <http://www.kernel.org/pub/linux/kernel/v3.x/linux-3.0.3.tar.gz>.

[6]
SHANNON, C.E. Prediction and Entropy of Printed English. *The Bell System Technical Journal*, 1951, roč. 30, č. 1, s. 50—64.

[7]
Pseudorandom number generator [online], aktualizace 26. 12. 2011, [cit. 29. 12. 2011], Wikipedie. Dostupné z: <http://en.wikipedia.org/wiki/Pseudorandom_number_generator>.

[8]
STALLMAN, R. *Linux a projekt GNU* [online]. 5. 5. 2005 [cit. 29. 12. 2011]. Dostupné na: <http://www.gnu.org/gnu/linux-and-gnu.html>.

[9]
BOVET, D. P., CESATI, M. *Understanding the Linux Kernel.* 1. vyd. Sebastopol: O'Reilly, 2001. 684 s. ISBN 0-596-00002-2.

[10]
GUTTERMAN, Z., PINKAS, B., et al. *Analysis of the Linux Random Number Generator* [online]. The Hebrew University of Jerusalem, 6. březen 2006 [cit. 29. prosinec 2011]. Dostupné na: <http://eprint.iacr.org/2006/086.pdf>.

[11]
Kolektiv autorů. *Linux: Dokumentační projekt*. Přel. DRESLEROVÁ, D., et al. 1. vyd. Praha: Computer Press, 1998. 1175 s. ISBN 80-7226-114-2.

[12]
MUTAF, P. *True random number from Wi-Fi background noise* [online]. Únor 2006 [cit. 29. 12. 2011]. Dostupné na: <http://www.freewebs.com/pmutaf/iwrandom.html>.

Přílohy

A. Skripty pro zajištění náhodnosti mezi starty systému

Skript zajišťující obnovení poolu při zavádění systému:

```
echo "Initializing random number generator..."
random_seed=/var/run/random-seed
# Carry a random seed from start-up to start-up
# Load and then save the whole entropy pool
if [ -f $random_seed ]; then
        cat $random_seed >/dev/urandom
else
        touch $random_seed
fi
chmod 600 $random_seed
dd if=/dev/urandom of=$random_seed count=1 bs=512
```

Skript zajišťující ukládání poolu při vypínání systému:

```
# Carry a random seed from shut-down to start-up
# Save the whole entropy pool
echo "Saving random seed..."
random_seed=/var/run/random-seed
touch $random_seed
chmod 600 $random_seed
dd if=/dev/urandom of=$random_seed count=1 bs=512
```

B. Seznam elektronických příloh

Součástí práce jsou elektronické přílohy. Ty obsahují:

- zdrojové kódy provedených změn v jádře ve formátu souboru „patch“
- výstup seznamu *randlist* na analyzovaných systémech
- systémové soubory se základními informacemi o hardwaru a konfiguraci analyzovaných systémů
  - konfigurační soubor jádra *.config*
  - informace o CPU *cpuinfo*
  - výpis ze záznamů jádra *dmesg*
  - jméno testovaného počítače *hostname*
  - informace o paměti *meminfo*
  - informace o rozdělení disku *partitions*