

MASARYKOVA UNIVERZITA
FAKULTA INFORMATIKY



Analýza entropie dat sloužících jako semínko generátoru náhodných čísel OS Linux při bootu

DIPLOMOVÁ PRÁCE

Ondřej Mokoš

Brno, podzim 2012

Prohlášení

Prohlašuji, že tato diplomová 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.

Ondřej Mokoš

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

Poděkování

Rád bych poděkoval Ing. Mgr. Zdeňku Říhovi, Ph.D. za vedení mé práce, jeho trpělivost a cenné rady, které mi poskytl.

Shrnutí

Cílem této diplomové práce je analýza semínka linuxového generátoru náhodných čísel při bootu. První část se věnuje teorii potřebné pro následnou analýzu. Zmiňuje některé často používané definice entropie, podává přehled linuxového generátoru náhodných čísel a jeho rozhraní. Také popisuje použitou implementaci. V druhé části je rozebrána analýza posbíraných dat. Jsou představeny výsledky analýz se zaměřením na kvantitu i kvalitu dat. Na závěr jsou diskutovány výsledky analýzy a některé možnosti, jak zlepšit aktuální stav.

Klíčová slova

Generátor náhodných čísel, RNG, Linux, semínko, boot, kvalitativní analýza, kvantitativní analýza, entropie

Obsah

1	Úvod	1
2	Entropie	3
2.1	<i>Shannonova entropie</i>	3
2.2	<i>Min-entropie</i>	3
2.3	<i>Rényiho entropie</i>	4
2.4	<i>Entropie podle random.c</i>	4
3	Linux RNG	7
3.1	<i>Struktura</i>	7
3.2	<i>Vstupní rozhraní</i>	9
3.2.1	<i>add_device_randomness()</i>	9
3.2.2	<i>add_input_randomness()</i>	9
3.2.3	<i>add_disk_randomness()</i>	9
3.2.4	<i>add_interrupt_randomness()</i>	10
3.2.5	<i>/dev/random</i>	11
3.3	<i>Výstupní rozhraní</i>	11
3.3.1	<i>Zařízení v /dev</i>	12
3.3.2	<i>Rozhraní /proc</i>	13
3.3.3	<i>Funkce na úrovni jádra</i>	14
4	Implementace	17
4.1	<i>Patch jádra</i>	17
4.2	<i>Provedené úpravy</i>	18
4.3	<i>Výstup a ukládání</i>	19
4.4	<i>Problémy</i>	21
5	Analýza	23
5.1	<i>Sběr dat</i>	23
5.2	<i>Kvantitativní analýza</i>	24
5.2.1	<i>Celkový počet událostí</i>	24
5.2.2	<i>Počet událostí, které přidávají náhodnost</i>	25
5.2.3	<i>Počet bitů náhodnosti</i>	26
5.2.4	<i>Ostatní</i>	30
5.3	<i>Kvalitativní analýza</i>	31
5.3.1	<i>Init</i>	32
5.3.2	<i>Irq</i>	32
5.3.3	<i>Input</i>	36
5.3.4	<i>Disk</i>	37

5.3.5	Časování	38
5.3.6	Data v input_pool	40
6	Závěr	43
	Literatura	45
A	Výstup ENT pro data v input_pool	47
A.1	<i>HW 1</i>	47
A.2	<i>HW 2</i>	47
A.3	<i>HW 3</i>	48
A.4	<i>HW 4</i>	49
A.5	<i>HW 5</i>	49
A.6	<i>Virt 1</i>	50
A.7	<i>Virt 2</i>	51
A.8	<i>Virt 3</i>	51
A.9	<i>Virt 4</i>	52
A.10	<i>Virt 5</i>	53
B	Obsah přiloženého CD	55

1 Úvod

Některé aplikace pro svou práci potřebují náhodná data. Skutečně náhodná data počítačům poskytují hardwarové generátory náhodných čísel nebo služby jako například RANDOM.ORG¹. Tato zařízení generují náhodná data měřením nepředvídatelných fyzikálních jevů. Z různých důvodů nemohou být všechny počítače vybavené hardwarovým generátorem náhodných čísel a služby jako RANDOM.ORG a ostatní nejsou vždy dostupné. V takových případech je odpovědnost za generování náhodných dat přenesena na operační systém.

V deterministickém prostředí počítačů je prakticky nemožné generovat skutečně náhodná data, proto se musíme spokojit s pseudonáhodnými daty. Pseudonáhodná data jsou generována algoritmem, který pro svou funkci potřebuje nějakou počáteční hodnotu zvanou semínko. Tato počáteční hodnota přímo ovlivňuje, jaká čísla algoritmus vrátí na výstup. Je tedy nutné zajistit, aby tato hodnota nebyla předvídatelná a také aby pro různé systémy ve stejných nebo podobných podmírkách nebyla stejná. To může být velmi náročný úkol podle toho, v jaké fázi se běh systému nachází.

Předmětem této práce je právě zkoumání semínka během startu systému. Boot systému je předem stanovená sekvence akcí, která má za cíl inicializovat a nastavit systém a spustit služby, které následně umožní uživatelům přihlásit se do systému a dále s ním pracovat. Tato sekvence je v podstatě neměnná a podnětů zvenčí, které by mohly být použity jako data pro semínko, je velmi málo, pokud nějaké jsou. Systém tak musí data pro semínko sbírat z událostí, ke kterým dochází uvnitř počítače. Paradoxně mohou být v této fázi prováděny operace, které mají velmi vysoké požadavky na objem a kvalitu náhodných dat, příkladem může být generování šifrovacích klíčů. Práce se pokusí ukázat, kolik bitů náhodnosti je k dispozici během startu systému, jak kvalitní tato náhodná data jsou a zda má systém k dispozici dostatečný objem dat.

Testován bude generátor v operačním systému GNU/Linux. Li-

1. Webová služba poskytující náhodná čísla pro mnoho různých účelů. Pro přístup je možné použít webovou stránku nebo rozhraní pro automatizované HTTP požadavky. Dostupné online na <http://www.random.org/>.

1. ÚVOD

nuxový generátor náhodných čísel je algoritmus, který pro dané semínko vrátí určitý výstup, jedná se tedy o generátor pseudonáhodných čísel. Pokud se v práci píše o generátoru náhodných čísel a náhodných číslech a datech, je myšlen generátor pseudonáhodných čísel a pseudonáhodná data a čísla.

Rozvržení kapitol

V první části je krátce popsána entropie. Zmíněny jsou některé často používané způsoby definice této veličiny.

Druhá část je přehled generátoru pseudonáhodných čísel, jak je implementován v linuxovém jádře, a vstupní a výstupní rozhraní pro práci s ním.

Třetí část představí implementaci, která byla použita pro sbírání dat pro následnou analýzu.

Čtvrtá část, která tvoří téměř polovinu práce, se zabývá vlastní analýzou posbíraných dat. Kvantitativní analýza zkoumá kolik dat je posbíráno během startu systému. Kvalitativní analýza se snaží objasnit, kolik náhodnosti je obsažené v těchto datech.

2 Entropie

Tato práce se převážně zabývá náhodností vyjádřenou jako entropie. Entropie je veličina, která má původ ve fyzice v oblasti termodynamiky. Zde entropie vyjadřuje do jaké míry je energie rovnoměrně rozdělena v nějakém systému, tedy kolik energie není možné použít pro výkon užitečné práce.[1] Termín entropie se postupně přenesl i do dalších oblastí, mezi nimi je i teorie informace. V teorii informace je entropie obecně chápána jako míra neurčitosti náhodné proměnné.

2.1 Shannonova entropie

Pod pojmem entropie je v teorii informace většinou chápána entropie, jak ji definoval Claude E. Shannon v článku „A Mathematical Theory of Communication“ z roku 1948, tzv. Shannonova entropie. Takto definovaná entropie udává průměrnou nepředvídatelnost náhodné proměnné, což je ekvivalentní hodnotě informace v ní obsažené.[3]

Entropii H diskrétní náhodné proměnné X je možné vypočítat následujícím vzorcem.

$$H(X) = - \sum_{i=1}^n p_i \log_b p_i \quad (2.1)$$

x_i jsou postupně možné hodnoty náhodné proměnné X , p_i pak pravděpodobnosti jejich výskytu, $p_i = \Pr(X = x_i)$. V závislosti na bázi b logaritmu lze entropii vyjádřit v různých jednotkách. Běžné používané jednotky jsou bity pro $b = 2$, naty pro $b = e$ nebo dity pro $b = 10$.

Pojmem entropie v této práci je myšlena právě Shannonova entropie, jednotky, ve kterých je entropie uváděna, jsou vždy bity.

2.2 Min-entropie

Shannonova entropie není jediný způsob, jak vyjádřit neurčitost nějaké proměnné. Jednou z dalších možných definic je tzv. min-entropie. Min-entropie udává míru náhodnosti v závislosti na čet-

2. ENTROPIE

nosti výskytu nejčastější hodnoty nějaké náhodné proměnné jako logaritmus pravděpodobnosti výskytu této hodnoty. Ekvivalentně min-entropie je reálné číslo c takové, že všechny hodnoty náhodné proměnné se vyskytují s pravděpodobností nejvýše 2^{-c} .[2]

$$H_\infty(X) = -\log_b(\max_{i=1}^n p_i) \quad (2.2)$$

Výše je vzorec pro výpočet min-entropie H_∞ náhodné proměnné X . Stejně jako u Shannonovy entropie báze logaritmu b udává, v jakých jednotkách je entropie vyjádřena.

2.3 Rényiho entropie

Rényiho entropii lze chápat jako zobecnění pojmů Shannonova entropie, min-entropie a některých dalších.[2] Rényiho entropie nese jméno maďarského matematika Alfréda Rényiho. Hodnotu Rényiho entropie řádu α je možné vypočítat podle následujícího vzorce.

$$H_\alpha(X) = \frac{1}{1-\alpha} \log_b \left(\sum_{i=1}^n p_i^\alpha \right) \quad (2.3)$$

Kde $\alpha \geq 0$ a $\alpha \neq 1$. S α blížící se k 0, Rényiho entropie započítává všechny hodnoty se stejnou váhou nezávisle na jejich pravděpodobnosti výskytu. V limitě $\alpha \rightarrow 0$ je Rényiho entropie rovná logaritmu mohutnosti náhodné proměnné X .

$$H_0(X) = \log |X| \quad (2.4)$$

Tento případ je někdy nazýván Hartleyova entropie. Naproti tomu s α blížící se nekonečnu je Rényiho entropie stále více určena pouze událostmi s největší pravděpodobností výskytu. Pro $\alpha \rightarrow \infty$ Rényiho entropie konverguje k min-entropii. Dále pro limitu $\alpha \rightarrow 1$ Rényiho entropie konverguje k Shannonově entropii.[7]

2.4 Entropie podle random.c

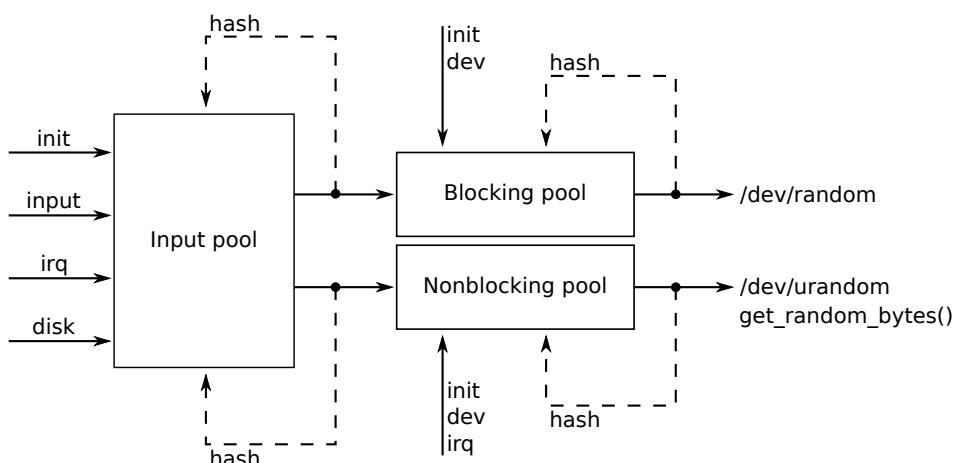
S pojmem entropie je možné se v této práci setkat v ještě jednom kontextu. Generátor náhodných čísel používá termín entropie pro ozna-

2. ENTROPIE

čení odhadu počtu náhodných bitů obsažených v nějakých datech. Tento odhad je udržován zvlášť pro všechny struktury, které ukládají data. Aktuální hodnota v určitém čase není výsledkem výpočtu v tomto čase, ale výsledkem série zvyšování nebo snižování hodnoty čítače od spuštění systému.

3 Linux RNG

V této části je popsán generátor náhodných čísel, který je k dispozici jako část linuxového jádra. Zdrojový kód generátoru je součástí zdrojového kódu jádra a nachází se v *drivers/char/random.c*, deklarace typů je možné nalézt v *include/linux/random.h*. V sekci Struktura jsou popsány jednotlivé části, ze kterých se generátor skládá. V sekcích o rozhraních jsou popsány způsoby sběru dat generátorem a další práce s ním a operace s tím spojené.



Obrázek 3.1: Struktura generátoru náhodných čísel.

3.1 Struktura

Data posílaná generátorem náhodných čísel jsou uchována ve třech datových strukturách typu `entropy_store`. `Input_pool`¹ slouží pro sběr dat, která jsou generována ideálně nepředvídatelnými událostmi jako je například vstup zadáný uživatelem. `Blocking_pool`² je určen pro většinou dočasné uložení dat při přístupu ke generátoru prostřednictvím zařízení `random`. `Nonblocking_pool`² má stejný

1. Nazýván také jako primární.
2. Nazýván také jako sekundární.

3. LINUX RNG

účel jako `blocking_pool` při přístupu s využitím zařízení `urandom` nebo funkce `get_random_bytes()`.

Datová struktura `entropy_store` obsahuje několik proměnných, zmíněné jsou pouze ty, které jsou relevantní pro obsah práce:

- **Pool** – Pole, které uchovává vlastní data. Data jsou reprezentována jako 4 byty dlouhá slova. Pool obsahuje 128 slov v případě `input_pool`, velikost je tak 512 bytů. `Blocking_pool` a `nonblocking_pool` obsahují po 32 slovech, jejich velikost je tedy 128 bytů.
- **Ukazatel na entropy_store strukturu** – Ukazatel na strukturu, ze které budou vybrána data v případě, že dostatečný objem nebude v příslušné struktuře k dispozici. V případě `input_pool` je tento ukazatel nastaven na `NULL`, `u blocking_pool` a `nonblocking_pool` ukazuje na `input_pool`.
- **Čítač dostupné entropie** – Obsahuje odhad bitů entropie aktuálně dostupných v dané struktuře.
- **Čítač celkově přidané entropie** – Obsahuje počet bitů entropie přidaných od inicializace struktury. Po přidání 128 bitů entropie se tento čítač přestane dále aktualizovat a příznak inicializace dané struktury je nastaven na 1.
- **Inicializace** – Příznak, který označuje, zda je daná struktura inicializovaná (bylo přidáno alespoň 128 bitů dat).
- **Limit** – Příznak určující, jestli je nutné ponechat dvojnásobek minimálního objemu dat, který může být přesunut, ve struktuře, ze které se vybírají data.

Během inicializace není obsah polí uchovávajících data vynulován. Toto by vedlo pouze ke snížení entropie.[14] Do všech struktur jsou běžným způsobem přimíchána systémová data funkcí `rand_initialize()`. V době inicializace generátoru mohou být v polích struktur již přítomná data přidaná funkcí `add_device_randomness()`. Tyto funkce jsou na obrázku 3.1 znázorněny jako šipky s označením init.

3.2 Vstupní rozhraní

Vkládat data do generátoru je možné pomocí několika funkcí. Nová data jsou do struktur generátoru postupně přimíchávána po jednotlivých bytech. Mechanismus pro přidávání dat využívá Twisted Generalized Feedback Shift Register. Více informací je možné nalézt v [10], [12] a [13].

3.2.1 add_device_randomness()

Funkce `add_device_randomness()`, která je součástí exportovaného vstupního rozhraní, slouží k vkládání dat do generátoru zejména před a také během jeho inicializace. Úkolem této funkce není přidávat reálnou entropii, ale odlišit počáteční hodnoty datových struktur u zařízení, kde jinak hrozí, že budou inicializovány stejnými daty.[14] Data přidaná touto funkcí jsou vložena do `input_pool` a `nonblocking_pool` a nezpůsobí zvýšení odhadu entropie. Touto funkcí mohou být vkládány například MAC adresy síťových karet nebo sériová čísla. `Add_device_randomness()` je možné použít také kdykoliv po inicializaci generátoru, ale takové použití nemá velký význam.

3.2.2 add_input_randomness()

Tato funkce slouží ke sbírání dat od událostí generovaných jako následek interakce uživatele s počítačem. Data jsou vkládána do `input_pool`. Pro odhad přidané entropie se používá časování těchto událostí, v dalších argumentech je kódována událost, ke které došlo. Na obrázku 3.1 odpovídá šipkám označeným jako `input`.

3.2.3 add_disk_randomness()

Pro vkládání dat pocházejících z práce disků slouží funkce `add_disk_randomness()`. Tato funkce je volána po dokončení každého vstupního/výstupního požadavku na disk. Data jsou vkládána do `input_pool` a pro odhad entropie se opět používá časování událostí. Funkce `add_disk_randomness()` je na obrázku 3.1 označena jako šipka s popisem `disk`.

3. LINUX RNG

Data pocházející z práce SSD³ disků nejsou na rozdíl od klasických rotačních disků dobrým zdrojem náhodnosti kvůli jejich velmi nízkým a pravidelným přístupovým časům. Předpokládám, že funkce, která provádí odhad počtu náhodných bitů v přidaných datech, by v takovém případě stanovila pro většinu událostí odhad na 0 bitů. Tedy přidávaná data by způsobila pouze minimální nárůst odhadu dostupné entropie. Bohužel nemám k dispozici SSD disk, abych mohl toto tvrzení ověřit.

3.2.4 add_interrupt_randomness()

`Add_interrupt_randomness()` je funkce, která v posledních verzích jádra prošla velkými změnami. Slouží ke sbírání náhodnosti generované přerušeními. V aktuální podobě vkládá do struktur generátoru údaje o časování, identifikátor přerušení a hodnotu obsaženou v čítači instrukcí⁴. Data jsou vkládána do `input_pool` a `nonblocking_pool`. Na obrázku 3.1 odpovídá šípkám s označením `irq`.

V předechozích verzích jádra byla tato funkce volána během obsluhy přerušení pouze u těch přerušení, která měla nastaven příznak `IRQF_SAMPLE_RANDOM` (dříve pojmenovaný `SA_SAMPLE_RANDOM`). Tento příznak označoval přerušení jako vhodný zdroj náhodnosti pro generátor náhodných čísel. Vhodná přerušení jsou taková, která jsou nepředvídatelná a zvenčí neovlivnitelná. Tento příznak byl postupně přerušením odebírána, až bylo rozhodnuto data od přerušení dále nesbírat a příznak byl označen jako `deprecated`. Příkladem diskuzí spojených s odstraněním `IRQF_SAMPLE_RANDOM` může být přerušení generované síťovou kartou. Jedna strana se přikláněla k názoru, že potenciální útočník dokáže úspěšně předvídat doručení paketů pouze v případě, kdy má fyzický přístup ke stroji, protože aktivní prvky v síti přidávají zpoždění, které je závislé na mnoha faktorech a tedy velmi obtížně předvídatelné s potřebnou přesností. Druhá strana zastávala názor, že přerušení generovaná síťovou kartou mohou být sledována i ovlivněna útočníkem. Tedy takové přerušení je zvenčí ovlivnitelné a ne-

3. Solid-state drive.

4. Speciální registr procesoru, také označován jako PC (Program Counter) nebo IP (Instruction Pointer).

vhodné jako zdroj náhodnosti.[4] Data generovaná přerušeními se do generátoru vrátila jako reakce na studii publikovanou v Mining Your Ps and Qs: Detection of Widespread Weak Keys in Network Devices, která ukázala, že přerušení jsou příliš cenným zdrojem, než aby byla ignorována.[11] Aktuálně jsou tedy sbírána data od všech přerušení, tato data jsou ukládána ve speciální struktuře, která je v daných intervalech přimíchána do generátoru. Vždy, když je tato struktura přidána k datům v generátoru, je odhad dostupné entropie zvýšen o 1 bit.

3.2.5 /dev/random

Dalším způsobem, jak přidat data do datových struktur je zápis do speciálních souborů `/dev/random` a `/dev/urandom` reprezentujících generátor náhodných čísel. V případě zápisu v obou souborech není rozdíl, protože zápis do obou má za následek volání funkce `random_write()`. Tato funkce má jako jeden z argumentů buffer s daty, který přimíchá zároveň do obou sekundárních úložišť. Zápisem do těchto souborů nedojde ke zvýšení odhadu dostupné entropie, to je potřeba udělat voláním funkce `ioctl()` s argumentem `RNDADDTOENTCNT`, které může provést pouze privilegovaný uživatel. Zápisem dat, která nejsou náhodná, nedojde k tomu, že výstupní data generátoru nebudou dostatečně náhodná. Toto by se mohlo stát pouze v případě, pokud by byl odhad dostupné entropie zvýšen bez přidání odpovídajícího objemu náhodných dat. Odpovídá šipkám označeným dev na obrázku 3.1.

3.3 Výstupní rozhraní

Aby bylo možné získat náhodná data z generátoru, musí být tato data nejprve přesunuta z primárního úložiště, kde jsou sbírána, do jednoho ze sekundárních úložišť. Při vybírání dat je nejprve spočítán SHA-1 hash ze všech dat v `input_pool`. Tento hash je přimíchán zpět do `input_pool` a po aplikaci funkce, která hash „přeloží napůl“, je také vrácen jako výstup. Tato operace je opakována, dokud nebylo vráceno tolik dat, kolik bylo požadováno nebo kolik je aktuálně dostupných. „Překládání napůl“ znamená, že prvních 10

3. LINUX RNG

bytů je pomocí operace XOR složeno s druhými 10 byty. Tato operace se provádí pro případ, že by hashovací funkce měla rozpoznatelný pravidelný výstup, zároveň se takto vždy vloží zpět dvakrát více dat, než je předáno na výstup.[14] Data jsou vložena do sekundárního úložiště opět pomocí Twisted Generalized Feedback Shift Register mechanismu. Odhad dostupné entropie je v `input_pool` snížen a v sekundární úložišti zvýšen o příslušný počet bitů. Minimální objem dat, který je možné z `input_pool` vybrat je stanoven na 8 bytů.

Zamíchání SHA-1 hashe zpět do struktury, ze které bylo vybíráno, má za úkol zamezit útokům, kdy útočník zná současný stav úložiště a výstupy a snaží se najít předchozí výstupy. Zamíchání hashe zpět činí tento útok alespoň tak náročný jako obrácení výpočtu SHA-1, tedy hledání vstupních dat pro známý SHA-1 hash. Odolnost proti tomuto typu útoků se nazývá Forward security⁵.

Vybírání dat ze sekundárních úložišť se skládá ze stejných operací, ale nemá omezení na minimální počet bytů, které je možné vybrat. Minimum v tomto případě je 1 byte ze zřejmých důvodů.

3.3.1 Zařízení v `/dev`

Nejběžnějším způsobem pro práci s generátorem náhodných čísel je čtení znakových zařízení `random` a `urandom` nacházející se v `/dev`.

random

Čtení tohoto zařízení je realizováno jako funkce, která vybere data z `blocking_pool` a předá je na určený výstup. V případě, že požadovaný počet bytů náhodnosti není v `blocking_pool` k dispozici, jsou tato data prvně vybrána z `input_pool`, vložena do `blocking_pool` a až potom vrácena uživateli. Pokud požadovaná data nejsou k dispozici ani v `input_pool`, funkce blokuje a žadatel musí čekat, až systém posbírá více náhodných dat.

Takto získaná data jsou vhodná pro použití v situacích, kdy jsou potřeba velmi kvalitní náhodná data, například při generování kryp-

5. Dopředná bezpečnost, většinou se nepřekládá.

tografického materiálu. Čtení `/dev/random` zaručuje, že vrácená data byla vygenerována s použitím čerstvě posbírané entropie.

urandom

Čtení zařízení `urandom` je velmi podobné `random`. Data jsou vybrána ze struktury `nonblocking_pool` a vrácena uživateli. Pokud požadovaná data nejsou v `nonblocking_pool` k dispozici, jsou nejprve vybrána z `input_pool`. Pokud nejsou data k dispozici ani v `input_pool`, jsou vrácena eventuálně méně kvalitní náhodná data bez blokování.

Aby mohl být požadovaný počet náhodných bytů přesunut do `nonblocking_pool`, musí být v `input_pool` dostatečný počet bytů náhodnosti (požadovaný počet + 16 bytů). Důvodem je, že `nonblocking_pool` má nastaveno omezení při vybírání dat – příznak limit. Toto omezení způsobí, že v `input_pool` zůstane vždy dostatek náhodných dat pro dva přenosy minimálního možného počtu bytů do `blocking_pool`. Pokud v `input_pool` není dostatečný počet bytů k dispozici, je přenesen nejvyšší možný počet bytů tak, aby zůstala rezerva 16 bytů. V případě, že počet náhodných bytů v `input_pool` je menší než 24, nejsou přenesena žádná data, protože není možné přenést stanovené minimum 8 bytů a zároveň zachovat požadovanou rezervu. Tento mechanismus zajistuje, že čtení `random` nebude zbytečně blokovat, zatímco `urandom` poskytuje data s využitím čerstvě posbírané entropie.

V ideálním případě je výstup `urandom` i `random` stejný v ohledu na kvalitu získaných dat. V horším případě, kdy `input_pool` neobsahuje dostatečnou entropii, mohou mít data získaná čtením `urandom` horší kvalitu.

Takto získaná data by měla být dostatečně kvalitní pro většinu použití s výjimkou použití v kryptografii.

3.3.2 Rozhraní /proc

`Procfs` je speciální souborový systém, který poskytuje informace o systému, běžících procesech a další hierarchicky rozdělené do adresářů. Během startu bývá typicky připojen do `/proc`.

Informace o generátoru náhodných čísel jsou umístěny

3. LINUX RNG

v `/proc/sys/kernel/random/`. Toto rozhraní neslouží pro získávání náhodných dat, ale pro získání informací o stavu generátoru. K dispozici je několik souborů.

- **boot_id** – Náhodný řetězec vygenerovaný během startu systému.
- **entropy_avail** – Dostupná entropie v bitech.
- **poolsizes** – Velikost pole pro ukládání náhodných dat v `input_pool` v bytech⁶.
- **read_wakeup_threshold** – Počet bitů, které musí být dostupné, aby byly probuzeny procesy čekající na novou entropii při čtení `/dev/random`. Hodnota může být změněna zápisem do souboru.
- **uuid** – Náhodný řetězec vygenerovaný při čtení tohoto souboru.
- **write_wakeup_threshold** – Pokud je dostupných méně bitů náhodnosti než tato hranice, jsou probuzeny procesy čekající na zápis do `/dev/random`. Hodnota může být změněna zápisem do souboru.

3.3.3 Funkce na úrovni jádra

Jako rozhraní pro přístup ke generátoru na úrovni jádra je poskytnuta funkce `get_random_bytes()`. Funkce vždy vrátí požadovaný počet bytů bez blokování. Volání `get_random_bytes()` je tedy obdobné čtení `/dev/urandom`.

Další exportovanou funkcí, která je definovaná v `random.c` je `get_random_bytes_arch()`. Volání této funkce vrátí náhodná data od hardwarového generátoru, pokud je k dispozici. Komentář této funkce zmiňuje, že hardwarový generátor je téměř jistě schopný generovat náhodná data rychleji než softwarový. Nevýhodou je nutnost věřit, že implementace je bezpečná. Jako příklad implementace,

6. Pouze v jádrech verze 2.6 a novější. Do verze 2.4 bylo do tohoto souboru možné zapisovat a obsahoval velikost pole v bytech. Zápisem bylo možné velikost pole změnit.[5]

3. LINUX RNG

která není bezpečná, je uvedeno generování náhodných dat šifrováním sekvenčních čísel za použití AES algoritmu, kdy NSA⁷ zná použitý klíč.[14]

7. National Security Agency, vládní agentura USA.

4 Implementace

Bez zásahu do kódu generátoru je možné číst pouze výstupní data, pro získání dat o semínku generátoru je nutné upravit kód, tedy vytvořit patch pro jádro. Data je potřeba sbírat v době startu systému, kdy zejména není k dispozici souborový systém a data tak nemohou být ukládána přímo na disk. Zápis na disk by navíc ovlivnil práci generátoru a způsobil by přidání nových náhodných dat. Data proto musí být ukládána do paměti a na disk zapsána až po startu systému.

4.1 Patch jádra

Pro účely práce byl upraven patch, který vytvořil a popsal Filip Škola ve své bakalářské práci.[16] Patch se týká těchto souborů:

- `include/linux/kernel.h`,
- `include/linux/random.h`,
- `drivers/char/mem.c`,
- `drivers/char/random.c`.

Patch zaregistrouje nové zařízení `randlist` do `/dev`. Zařízení `randlist` slouží jako uživatelské rozhraní pro přístup ke struktuře `randlistmsg`, která uchovává posílaná data. Data jsou ukládaná jako spojovaný seznam, jednou z proměnných v `randlistmsg` je odkaz na následující objekt. Struktura `randlistmsg` je definovaná v souboru `kernel.h`.

Funkce pro obsluhu zařízení `randlist` jsou definované v `mem.c`, konkrétně jsou to `open_randlist()`, `release_randlist()`, `read_randlist()` a `write_randlist()`. Funkce `read_randlist()` umožňuje číst posílaná data a funkce `write_randlist()` slouží pro zápis dat do `randlistmsg`.

Funkce `write_randlist()` umožňuje zápis do `randlistmsg` z uživatelského prostoru, ale sbírání dat v generátoru náhodných čísel je realizováno funkcí `write_to_randlist()`, která je definovaná v `random.c`. Tato funkce je volána při obsluze

4. IMPLEMENTACE

vstupů z jednotlivých zdrojů náhodných dat. Pro data pocházející z práce disků, přerušení a vstupních událostí je tato funkce volaná v `add_timer_randomness()`, která je společná pro zmiňované zdroje. Data vložená zápisem do `/dev/random` jsou odchycena ve funkci `random_write()`, ioctl volání přidávající data pak v `random_ioctl()` s příslušnými příznaky.

Ukládané události jsou pouze takové, které mají nenulový odhad přidané entropie. Události, které by nepřidaly žádnou novou náhodnost, jsou ignorovány.

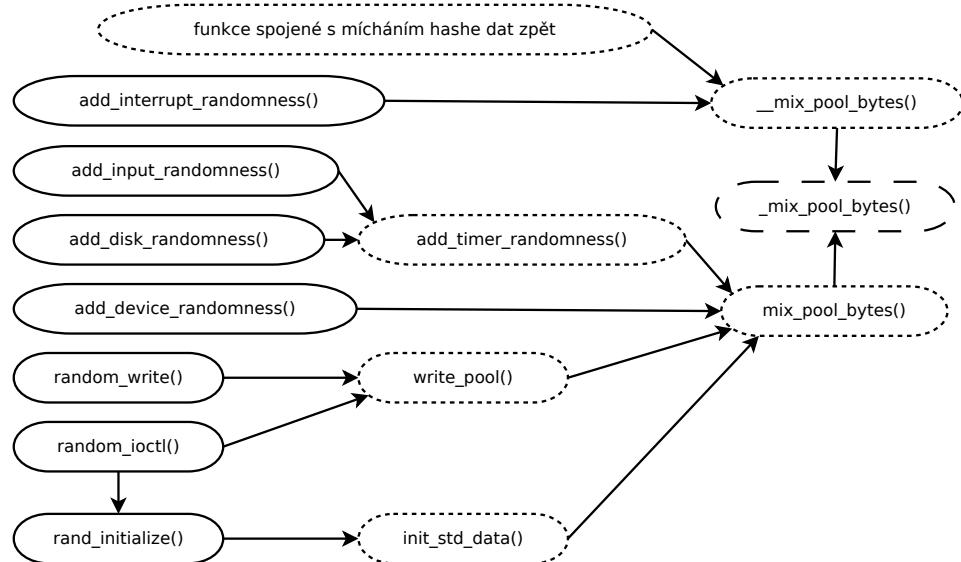
4.2 Provedené úpravy

V práci Semínko generátoru náhodných čísel OS Linux při bootu byl patch vytvořen a použit v jádrech verze 3.0.3 a 3.1.1. Mezi těmito verzemi a verzí 3.5.4 jádra, použité v této práci, došlo v kódu generátoru ke změnám ve funkcích, které přidávají náhodná data. Patch tedy musel být přizpůsoben pro aktuálnímu stavu jádra a potřebám práce.

Funkce `add_interrupt_randomness()` byla kompletně změněna. Ve svém těle dále nevolá `add_timer_randomness()` a tato funkce tak nemohla být použita pro sbírání dat jako společná pro všechny zdroje. Jedinou funkcí společnou pro všechny zdroje je `_mix_pool_bytes()`. Výhodou této funkce je, že se jedná o funkci na nejnižší úrovni, která zajišťuje míchání dat do struktur generátoru a je tedy možné zachytit data v podobě, v jaké jsou přidávána. Nevýhodou je, že úpravy odhadu entropie dostupné v generátoru provádí funkce na vyšších úrovních a je tak nutné předávat argumenty mezi několika funkcemi, aby mohly být v `_mix_pool_bytes()` konečně zapsány do `randlist`, postupné volání funkcí je zobrazeno na obrázku 4.1. Také byla nově přidána funkce `add_device_randomness()`.

Další změnou je, že jsou ukládány i události, které nezpůsobí zvýšení odhadu dostupné entropie. Tato změna by mohla pomoci lépe porozumět způsobu, jakým je odhadován počet nově přidaných náhodných bitů z informací o časování událostí. Protože předmětem analýzy je semínko generátoru náhodných čísel, jsou ukládány pouze takové události, které vkládají data do `input_pool`.

4. IMPLEMENTACE



Obrázek 4.1: Znázornění všech funkcí, které při provádění eventuálně volají `_mix_pool_bytes()`. Plnou čarou jsou označeny funkce, které slouží jako rozhraní pro práci s generátorem náhodných čísel popsané v kapitole 3, pírušovanou čarou je označena funkce, kde jsou data zapisována do *randlist*. Šipka z A do B vyjádřuje, že funkce A ve svém těle volá funkci B.

4.3 Výstup a ukládání

Zařízení *randlist* každý záznam ukládá jako textový řetězec. Formát tohoto řetězce je:

`čas id [zdroj] %d %d %d -data data -pool pool`

- **čas** – Čas od startu systému uváděn v sekundách a milisekundách.
- **id** – Pořadové číslo záznamu.
- **[zdroj]** – Zdroj události, možné hodnoty jsou init, input, irq, disk, dev, device a msg. Init a device označuje události spojené s inicializací generátoru, input jsou události způsobené

4. IMPLEMENTACE

interakcí s počítačem, irq označuje události generované přerušeními, disk události vzniklé následkem práce disků a dev události způsobené zápisem do zařízení v `/dev`.

- **%d** – Počet bitů entropie přidaných touto událostí.
- **%d** – Počet bitů entropie aktuálně dostupných v `input_pool`.
- **%d** – Celkový počet bitů entropie přidaných do `input_pool`, odpovídá proměnné čítač celkově přidané entropie popsané v kapitole 3.1.
- **data** – Data, která byla přidána touto událostí. 16 bytů dat, zapsaná hexadecimálně.
- **pool** – Obsah `input_pool`, konkrétně obsah pole, které udržuje vložená data. Data jsou zapsaná hexadecimálně jako 128 4bytových slov.

Data posbíraná v zařízení `randlist` byla na disk ukládána skriptem spouštěným na konci startu systému. Tento skript nejprve do `randlist` zapíše novou zprávu.

```
echo "rc.local -- konec bootu" > /dev/randlist
```

Tato zpráva se ve výpisu `randlist` zobrazí se zdrojem nastaveným jako msg, který označuje zprávy přidané do `/dev/randlist` právě zápisem do tohoto zařízení. Tato zpráva označuje místo, které je pro účely práce považováno za konec startu systému, všechny záznamy následující tuto zprávu budou v následné analýze ignorovány. Skript dále uloží některé soubory s logy, které obsahují informace o startu systému, a také verzi jádra.

Skript pro ukládání je spouštěný z `rc.local`. Skript `rc.local` je součástí Initscripts a je určený právě pro spouštění příkazů na konci bootovací sekvence. Initscripts je sada skriptů, které daemon init spouští během startu a inicializace systému. System V init bývá v novějších systémech nahrazen jinými implementacemi, které například lépe využívají paralelismus. Linuxová distribuce Fedora 17, na které byla data sbírána, využívá implementaci systemd. Pro zachování zpětné

kompatibility systemd spouští skript *rc.local* prostřednictvím služby *rc-local.service*.

4.4 Problémy

Protože patch určuje čas výpočtem zvlášť sekund a milisekund, které jsou pak ve výpisu spojeny jako sekundy.milisekundy, může dojít k tomu, že počet milisekund vypočítaný pro časový údaj dané události je menší než 100. Tedy ve výpisu času by číslo udávající milisekundy mělo být zleva doplněno jednou nebo dvěma nulami. Patch toto nijak nevynucuje, proto se může stát, že ve výpisu událostí z *randlist* je čas udán jako 1.5 nebo 1.50. To odpovídá 1 sekundě a 5 milisekundám, respektive 50 milisekundám, správně by tedy mělo být zobrazeno 1.005 a 1.050. Tato chyba je opravena a číslo je vždy doplněno příslušným počtem nul. Bohužel jsem tuto chybu objevil až po provedení větší části analýzy a rozhodl se tak nechat data v původní podobě.

Dalším problémem je sbírání dat, přesněji jejich nesbírání, ve funkci *add_device_randomness()*. Pokud je tento zdroj nastaven, aby byl ignorován, boot s upraveným jádrem proběhne v pořádku. Pokud se ale v této funkci zapisuje do *randlist*, start systému se zastaví krátce po začátku a už se znova nerozběhne. Nejsem schopen podat konkrétní vysvětlení, proč tomu tak je. Domnívám se, že funkce *write_to_randlist()* pro svoje vykonání potřebuje něco, co v této fázi ještě není dostupné, protože funkce *add_device_randomness()* je pravděpodobně volána velmi brzy po začátku bootovací sekvence, dříve než *rand_initialize()*, jak bylo popsáno v sekci 3.2.1. Tento problém se mi nepovedlo vyřešit jinak, než ignorováním *add_device_randomness()* jako zdroje dat, která jsou zapisována do *randlist*.

V některých případech může boot systému s takto upraveným jádrem skončit neúspěchem s chybami `bad: scheduling from the idle thread!` a `BUG: scheduling while atomic: swapper /0/0/0x00010000`. Pokusy s různými verzemi patche jádra ukázaly, že tyto chyby jsou způsobeny polem *randpool*, které dočasně ukládá obsah *input_pool*. Během alokování paměti pro toto pole vlákno pravděpodobně může čekat na výsledek,

4. IMPLEMENTACE

tedy stane se neaktivním a plánovač se pokusí přidělit výpočetní čas jinému vláknu. Během obsluhy přerušení je volána funkce `add_interrupt_randomness()`, která dále volá funkce určené pro sbírání dat, ty se pak snaží alokovat paměť pro toto pole. Pokud v tomto případě dojde k čekání na přidělení paměti a tedy snaze plánovače přidělit čas jinému vláknu, vzniknou tyto chyby, protože během obsluhy přerušení by zejména k tomuto nemělo dojít. Výše je, dle mého názoru, vysvětlení tohoto chování pro boot systému na reálném počítači. Zajímavé ale je, že se stejnými chybami je možné se setkat i ve virtuálním prostředí. Boot systému ve VirtualBoxu vždy proběhl úspěšně. Zmíněné dvě chyby se objevily až při pokusu o čtení `/dev/randlist`, a podobně jako u reálného počítače, i zde pouze v některých případech. V tomto případě nejsem schopen podat jakékoliv vysvětlení.

5 Analýza

Tato kapitola se věnuje sbírání dat a jejich následné analýze. Analýza je rozdělena na kvantitativní a kvalitativní část. Kvantitativní analýza odpovídá na otázku, kolik si systém myslí, že má k dispozici bitů náhodnosti, kvalitativní analýza se pak snaží ukázat, kolik bitů náhodnosti je skutečně k dispozici.

5.1 Sběr dat

Data byla sbírána upraveným jádrem, jak bylo popsáno ve 4. části. Jedním z cílů práce je porovnání dat získaných na reálném počítači s daty z virtuálního prostředí. V obou případech byla využita dnes nejrozšířenější architektura x86-64 s připojeným jedním pevným diskem a jednou mechanikou pro optické disky. Pro virtualizaci byl zvolen softwarový balík VirtualBox. Na obou počítačích byla nainstalována 64bitová verze linuxové distribuce Fedora ve verzi 17. Bylo použito standardní jádro verze 3.5.4 z repozitářů distribuce, na které byl aplikován dříve zmíněný patch.

Boot s tímto jádrem byl opakován celkem pětkrát na obou systémech. Tímto způsobem bylo celkem získáno více než 60 MB dat v podobě deseti textových souborů. Tyto soubory celkem obsahují 51877 událostí, které přidávají nějaká data do `input_pool`, 31724 událostí na reálném PC a 20153 na virtuálním PC. 12165 událostí má nenulový odhad entropie obsažené v jimi přidávaných datech, 6662 pro reálné PC a 5503 pro virtuální PC.

Sbírána byla pouze data přidávaná do `input_pool`, protože právě data z `input_pool` jsou použita pro `reseed blocking_pool` i `nonblocking_pool`, když generátor předává náhodná data na výstup, tato data je tedy možné chápat jako semínko generátoru.

Jak již bylo zmíněno dříve, data jsou přimíchávána do struktur generátoru pomocí mechanismu na principu Twisted Generalized Feedback Shift Register. Velmi zjednodušeně lze říci, že se tento mechanismus skládá z operací XOR a bitový posun. Entropie přítomná v přidávaných datech je zachována a přenesena do generátoru díky použití operace XOR. Jednou z vlastností XOR je, že vždy zachovává náhodnost. To znamená, že nepředvídatelnost výstupu této operace

5. ANALÝZA

	init	input	irq	disk	celkem
HW 1	2	7	102	6327	6438
HW 2	2	7	90	5732	5831
HW 3	2	7	99	6422	6530
HW 4	2	7	99	6254	6362
HW 5	2	7	101	6453	6563
Virt 1	2	11	71	3943	4027
Virt 2	2	11	29	3942	3984
Virt 3	2	11	28	3893	3934
Virt 4	2	11	29	4032	4074
Virt 5	2	11	29	4092	4134
celkem	20	90	677	51090	51877

Tabulka 5.1: Celkový počet událostí rozdelený podle zdrojů pro každý boot. HW jsou označeny jednotlivé booty na reálném počítači, Virt značí boot ve virtuálním prostředí.

je vždy alespoň tak vysoká, jako nepředvídatelnost lepšího z obou vstupů.[8]

5.2 Kvantitativní analýza

Tato část se pokusí odpovědět na otázku, kolik si systém myslí, že má k dispozici bitů náhodnosti. Předmětem zkoumání v tomto případě je pouze množství posbíraných dat a bitů, které jsou pokládány za byty entropie. Kvalitu těchto dat rozebere část 5.3.

5.2.1 Celkový počet událostí

Tabulka 5.1 zobrazuje celkový počet událostí, které byly uloženy v zařízení *randlist*. Události jsou pro každý boot rozděleny podle zdrojů, ze kterých pocházely. V každém startu systému byla přidávána pouze data ze zdrojů uvedených v tabulce, zbylé zdroje dev a device se nevyskytly. Device není v seznamu přítomný z toho důvodu, že data z funkce `add_device_randomness()` musela být při sbírání ignorována, jak bylo vysvětleno v části 4.4. Zdroj dev označuje data přidaná zápisem do souborů `/dev/random`.

5. ANALÝZA

a `/dev/urandom`. Protože zápis do zařízení v `/dev` přidává data přímo do obou sekundárních struktur, události od tohoto zdroje se nemohou objevit v seznamu. Jediný případ, kdy nějaký proces zapisuje do těchto souborů, je při zapisování semínka, které bylo uloženo před vypnutím počítače. K tomu dojde přibližně po 8 sekundách po zahájení bootu.

Ve všech případech nejvíce událostí pocházelo z práce disků. Na HW stroji podíl událostí z práce disku činil 98 %, ve virtuálním prostředí byl podíl práce disků nepatrně vyšší až 99 %. Čísla se pro daný počítač liší vždy maximálně o několik desetin, s výjimkou Virt 1, tedy prvního bootu ve virtuálním prostředí. Počet událostí pocházejících ze vstupních zařízení počítače je vždy stejný.

První boot virtuálního počítače se od ostatních výrazně lišil dobou trvání, která byla více než třikrát delší. Toto dále ovlivnilo i většinu ostatních hodnot. V tabulce 5.1 je patrný rozdíl mezi počty událostí vygenerovanými následkem přerušení, 71 v případě Virt 1 oproti méně než 30 v ostatních případech. Další rozdíl je, že s vyšším počtem přerušení se mírně snížil podíl událostí z práce disků na celkovém počtu zaznamenaných událostí.

5.2.2 Počet událostí, které přidávají náhodnost

Všechny události vkládají nová data do struktur generátoru, ale jenom některé současně zvyšují i odhad dostupné entropie. Počty událostí s pozitivním odhadem entropie obsažené v přidávaných datech jsou obsahem tabulky 5.2. Jednotlivé události jsou opět rozděleny podle zdrojů, ze kterých pochází.

Z tabulek 5.1 a 5.2 je patrné, že všechny události ze zdroje irq přidávají nějaké byty náhodnosti. Důvod je ten, že ve zdrojovém kódu generátoru je pevně určeno, že každá taková událost zvýší čítač dostupné entropie právě o 1 bit. Naproti tomu události pocházející z práce disků nebo vstupů od uživatele mohou, ale také nemusí, zvyšovat tuto hodnotu, to závisí na odhadu vycházejícím z časování těchto událostí. Protože hrají roli i velmi malé časové intervaly, počet událostí, které přidávají entropii, není vždy stejný. Způsob, jakým je odhad prováděn, bude popsán dále.

Dlouhá doba trvání prvního startu virtuálního počítače se zde projevuje vyšším počtem událostí ze zdrojů input a disk, které přidá-

5. ANALÝZA

	init	input	irq	disk	celkem
HW 1	0	3	102	1355	1460
HW 2	0	3	90	1139	1232
HW 3	0	3	99	1171	1273
HW 4	0	3	99	1174	1276
HW 5	0	3	101	1317	1421
Virt 1	0	3	71	1695	1769
Virt 2	0	2	29	874	905
Virt 3	0	2	28	856	886
Virt 4	0	2	29	898	929
Virt 5	0	2	29	983	1014
celkem	0	26	677	11462	12165

Tabulka 5.2: Celkový počet událostí, které přidávají nějaké byty náhodnosti, rozdelený podle zdrojů pro každý boot. HW jsou označeny jednotlivé booty na reálném počítači, Virt značí boot ve virtuálním prostředí.

vají nějakou náhodnost. Tyto události byly rozprostřeny do delšího časového úseku a intervaly mezi nimi jsou tak méně pravidelné, tedy více těchto událostí má nenulový odhad přidané entropie.

5.2.3 Počet bitů náhodnosti

Tabulka 5.3 ukazuje, kolik bitů entropie bylo obsaženo v datech vložených do generátoru během jednotlivých bootů. Opět je možné vidět, že práce disků je zdrojem většiny bitů náhodnosti přidaných během startu počítače. Více než 95 % bitů přidaných během startu reálného počítače pochází z tohoto zdroje, ve virtuálním prostředí je to pak 98 %.

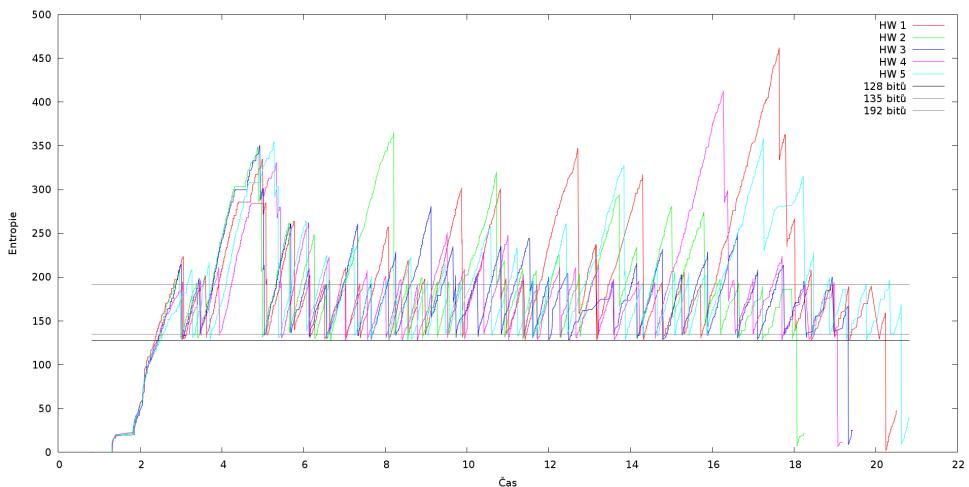
V případě reálného počítače bylo vždy přidáno alespoň 2600 bitů, tedy generátor by měl být během startu počítače schopen poskytnout více než 325 bytů náhodných dat. V případě virtuálního počítače bylo vždy přidáno více než 2100 bitů, což odpovídá více než 262 bytům náhodných dat. Tato čísla by se mohla zdát jako dostatečně vysoká, nicméně tabulka je pouze jeden způsob, jak reprezentovat data.

Následující grafy ukazují, kolik bitů entropie je v generátoru do-

5. ANALÝZA

	init	input	irq	disk	celkem
HW 1	0	27	102	2953	3082
HW 2	0	28	90	2538	2656
HW 3	0	28	99	2565	2692
HW 4	0	28	99	2622	2749
HW 5	0	27	101	2866	2994
Virt 1	0	18	71	5284	5373
Virt 2	0	17	29	2201	2247
Virt 3	0	16	28	2116	2160
Virt 4	0	17	29	2111	2157
Virt 5	0	16	29	2265	2310
celkem	0	222	677	27521	28420

Tabulka 5.3: Celkový počet přidaných bitů náhodnosti rozdělený podle zdrojů pro každý boot. HW jsou označeny jednotlivé booty na reálném počítači, Virt značí boot ve virtuálním prostředí.



Obrázek 5.1: Počet dostupných bitů náhodnosti v závislosti na čase od zahájení bootu systému na reálném počítači. Graf je v lepší kvalitě na přiloženém CD.

stupných v závislosti na uplynulém času od spuštění systému. Graf na obrázku 5.1 ukazuje vývoj počtu dostupných bitů entropie bě-

5. ANALÝZA

hem startu systému na reálném počítači. Části, kde křivky v grafu stoupají, odpovídají přidávání dat do `input_pool` a zvyšování odhadu dostupné entropie. Naopak části, kde křivky klesají značí vybírání dat z `input_pool` a vkládání těchto dat do některého ze sekundárních poolů, tzv. `reseed`. V grafu je dále označen limit 192 bitů. `Reseed nonblocking_pool` je možné provádět pouze v těch případech, kdy odhad dostupné náhodnosti v `input_pool` je 192 bitů nebo vyšší. Protože vybírání dat ve většině případů sníží odhad dostupných bitů na 128, je možné předpokládat, že data vybírá `nonblocking_pool` pro svůj `reseed`. Z grafu je také možné odhadnout, co se v takových případech děje uvnitř generátoru.

V nejpravděpodobnějším scénáři nějaký proces žádá od generátoru náhodná data použitím jednoho z neblokujících výstupních rozhraní. V tomto případě se jako výstup vrátí data z `nonblocking_pool`. `Nonblocking_pool` ale nemá dostatek entropie, aby vyhověl tomuto požadavku, a tak se pokusí provést `reseed`, tedy pokusí se z `input_pool` přenést požadovaný objem dat. Ani v `input_pool` není dostatečný počet bitů entropie a do `nonblocking_pool` je přeneseno jenom kolik bitů, kolik je aktuálně dostupných tak, aby byl zachován limit 128 bitů popsáný v 3.3.1 v části o `urandom`.

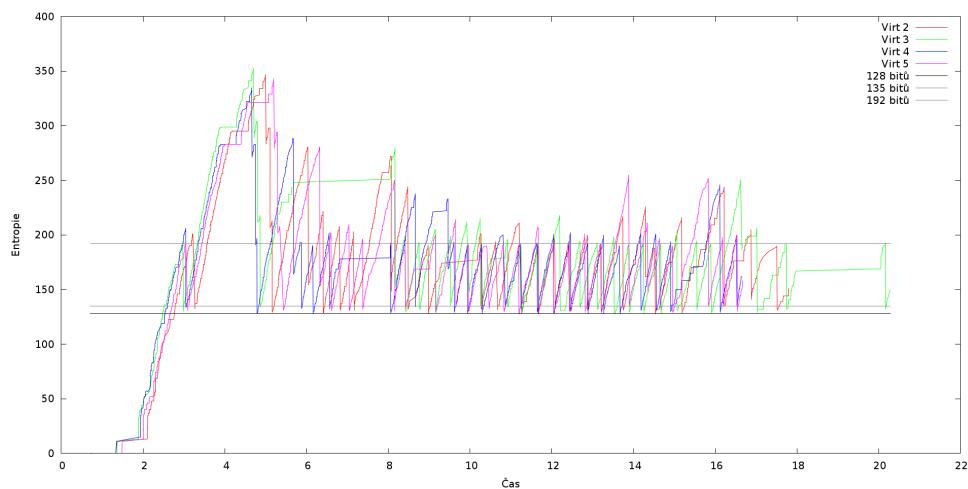
Počet bitů, který zůstane v `input_pool` nemusí být přesně 128. Funkce, která zajišťuje `reseed` sekundárních poolů má jako jeden z argumentů počet požadovaných bytů. Je tedy možné, že v `input_pool` zůstane až 135 bitů, protože při pokusu o vybrání dalšího 1 bytu dat by nebyla dodržena povinná rezerva 128 bitů. Z toho vyplývá, že pokud odhad po vybrání dat klesne mezi 128 a 135 bitů, generátor neměl k dispozici dostatek bitů náhodnosti, aby provedl úplný `reseed` před vrácením náhodných dat, a vrácená data potenciálně mohou být méně kvalitní.

V některých případech se mohlo stát, že byl vybrán požadovaný objem dat a rezerva, která zůstala v `input_pool`, spadla do tohoto intervalu pouze dílem náhody. K vybírání dat z `input_pool` vůbec nedošlo a v grafu se neprojeví, že generátor vrátil náhodná data procesu, který o ně žádal, pokud `nonblocking_pool` obsahoval dostatek entropie. V takovém případě generátor vrátil data na výstup a pouze snížil odhad entropie v `nonblocking_pool`. Druhý případ

5. ANALÝZA

je, pokud v `input_pool` bylo dostupných méně než 192 bitů náhodnosti. Pak reseed vůbec nebylo možné provést a výstupní data byla vygenerována bez použití čerstvě posbírané entropie.

Krátce před koncem startu systému je možné vidět přístup ke generátoru přes rozhraní `/dev/random` a následkem toho reseed `blocking_pool`, který z `input_pool` vybere 20 bytů dat. Při druhém a čtvrtém startu generátor neměl dostatek entropie, aby požadavku vyhověl a tak proces byl zablokován při čtení `/dev/random`. Toto v grafu vidět není, je možné to odvodit z informací posbíraných v `randlist`.

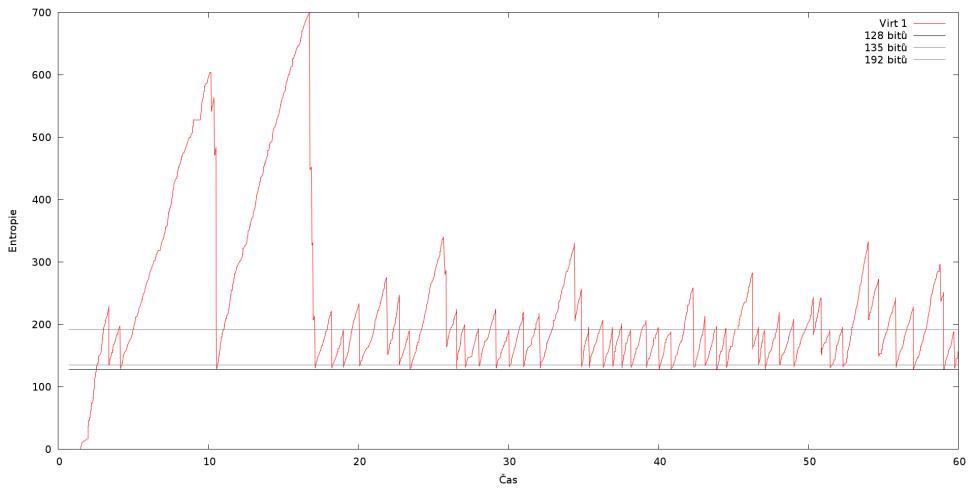


Obrázek 5.2: Počet dostupných bitů náhodnosti v závislosti na čase od zahájení bootu systému ve virtuálním prostředí. Graf je v lepší kvalitě na přiloženém CD.

Obrázky 5.2 a 5.3 ukazují tyto grafy pro start systému ve virtuálním prostředí. Na těchto grafech je opět možné vidět chování popsané výše. Až na několik výjimek každý reseed sníží odhad bitů náhodnosti na možné minimum, tedy generátor opět nemá dostatek entropie.

Fakt, že toto chování je možné pozorovat i při prvním startu virtuálního počítače (obrázek 5.3), během kterého generátor posbíral nejvíce bitů ze všech pokusů, naznačuje, že entropie je potřeba řádově více, než kolik je dostupné v aktuální implementaci.

5. ANALÝZA



Obrázek 5.3: Počet dostupných bitů náhodnosti v závislosti na čase od zahájení bootu systému ve virtuálním prostředí. Speciální případ prvního startu virtuálního počítače, který trval výrazně déle. Graf je v lepší kvalitě na přiloženém CD.

5.2.4 Ostatní

Tabulka 5.4 ukazuje některé další hodnoty. Je možné vidět, že dříve zmiňovaný limit 192 bitů byl poprvé dosažen 3 sekundy po zahájení bootu systému. To znamená, že během prvních 3 sekund běhu počítače není možné provést reseed před vrácením náhodných dat z nonblocking_pool. Pro srovnání souborový systém je k dispozici přibližně po 7 sekundách od začátku bootu.

Další sloupec uvádí celkový čas od startu systému po místo, kde byla sbírána data. Zde je dobře viditelný rozdíl mezi Virt 1 a ostatními starty. První start virtuálního stroje trval třikrát déle než ostatní, včetně těch prováděných na hardwaru místo virtuálního prostředí. To umožnilo posbírat přes dvakrát více bitů náhodnosti, jak ukazuje tabulka 5.3, a také dosáhnout až dvojnásobně vyššího maximálního počtu bitů entropie dostupných během startu.

Maximální počet dostupných bitů entropie je zejména ve virtuálním prostředí konzistentní pro všechny pokusy. Po vrcholu na začátku bootu se dostupná entropie rychle vyčerpala a odhad dále pře-

	192 bitů	celkový čas	max entropie	čas
HW 1	2,811	20,516	462	17,634
HW 2	2,845	18,258	366	8,209
HW 3	2,879	19,432	352	4,932
HW 4	3,041	19,191	413	16,271
HW 5	3,154	20,823	358	17,252
Virt 1	2,996	59,992	703	16,733
Virt 2	3,183	17,804	347	4,994
Virt 3	2,967	20,296	353	4,701
Virt 4	2,977	16,649	335	4,660
Virt 5	3,028	16,678	343	5,196

Tabulka 5.4: Některé další údaje. První sloupec obsahuje čas, kdy byla dosažena hranice 192 bitů, v druhém sloupci je celkový čas startu systému. Třetí a čtvrtý sloupec obsahují maximální počet bitů přítomný v `input_pool` a čas, kdy se tak stalo. HW jsou označeny jednotlivé booty na reálném počítači, Virt značí boot ve virtuálním prostředí.

kračoval hranici 192 bitů jen zřídka a o relativně malý počet bitů. To může poukazovat na nedostatek entropie v ještě větší míře než je tomu u reálného počítače. Krátce po začátku bootu na reálném počítači generátor také posbírá hodně bitů entropie, ale většinou během dalšího běhu tuto hranici ještě překročí.

5.3 Kvalitativní analýza

Kvantitativní analýza odhalila, že během startu systému generátor nemá k dispozici dostatek náhodných bitů. Linuxový generátor náhodných čísel je obecně považován za konzervativní při odhadování, kolik náhodnosti je obsaženo v přidaných datech.[10] V souvislosti s tím se nabízí hned několik otázek.

Pokud generátor skutečně odhad entropie podhodnocuje, bylo by možné výpočet upravit, aby byl trochu více optimistický a přesněji odpovídal entropii obsažené v těchto datech? Každá událost přidává 16 bytů dat, ale pouze polovina je použita při odhadu přidané entropie. Mohla by být použita i druhá polovina dat pro odhad entropie.

5. ANALÝZA

ropie? Kolik entropie tato data obsahuje?

Data budou při zkoumání rozdělena podle zdrojů, ze kterých pocházejí.

5.3.1 Init

Každý boot obsahoval právě dva záznamy od tohoto zdroje přidané jako první méně než sekundu po startu systému. Tyto záznamy byly přidány z funkce `rand_initialize()`. První záznam obsahuje aktuální čas s přesností v nanosekundách zapsaný hexadecimálně. Následující příklad lépe vysvětlí formát, v jakém je čas uvedený.

$$(0x12c94dfcb890a120)_{10} = 1353698910\mathbf{825128200} \quad (5.1)$$

- `0x12c94dfcb890a120` – data, jak byla uložena v `randlist`
- `1353698910` – standardní unixový čas v sekundách, po převodu na běžně používaný formát odpovídá 23.11.2012, 19:28:30 GMT
- `825128200` – počet nanosekund

Těchto 8 byte je doplněno `0x00000000FFFFFFFF` na celkovou délku 16 byteů.

Druhý záznam vloží do generátoru jméno systému doplněné nulami na 16 byteů. Jméno systému vložené touto událostí neodpovídá hostname počítače, ale pouze „Linux“ v ASCII kódované jako `0x4c696e7578`, protože hostname v této fázi ještě není k dispozici.

Odhad náhodnosti přidané v `rand_initialize()` je ve zdrojovém kódu generátoru nastaven na 0. Data přidaná ve všech startech na obou testovaných systémech byla vždy stejná, pouze čas se zvyšoval, protože počítač má k dispozici funkční hodiny a VirtualBox využívá systémový čas.

5.3.2 IRQ

IRQ jsou označeny události, které byly přidány funkcí `add_interrupt_randomness()`. Tato funkce je volána při obsluze každého přerušení, ale data do generátoru přidává pouze

5. ANALÝZA

v pevně stanovených intervalech. Data pocházející od přerušení jsou přimíchána do speciální struktury `fast_pool` mechanismem podobným tomu, který vkládá data do generátoru. Struktura `fast_pool` je standardním způsobem vložena do dat generátoru po 64 přerušení nebo uplynutí 1 sekundy od předchozího přidání podle toho, co nastane dříve, a odhad dostupné entropie je zvýšen o 1 bit.

Funkce `add_interrupt_randomness()` při každém volání přidává 16 bytů dat do `fast_pool`.

- 4 byty – **cycles XOR jiffies** – Položka `cycles` obsahuje dolních 32 bitů čítače cyklů. Jiffies obsahuje počet tiků, tedy přerušení časovače, od startu systému.
- 4 byty – **irq** – Číslo přerušení, každý zdroj přerušení má přiřazeno unikátní číslo.
- 8 bytů – **instruction pointer** – Hodnota uložená v čítači instrukcí. Typicky je to adresa v paměti, kde se nachází instrukce, která bude vykonána v příštím cyklu.

Odhad entropie je ve zdrojovém kódu nastaven na 1 bit. Tedy v případě reálného počítače bylo touto funkcí přidáno 100 bitů náhodnosti, u virtuálního počítače dokonce jenom 30 bitů. Přerušení byla v minulosti postupně odstraněna ze zdrojů entropie pro generátor náhodných čísel a zpět byla přidána až poté, co byl prokázán kritický nedostatek entropie v některých embedded zařízeních. Velmi malý podíl přerušení na celkovém počtu přidaných bitů v aktuální implementaci znamená, že přerušení jako zdroj entropie jsou stále v podstatě ignorována.

Záznamy v `randlist` s označením `irq` obsahují data z `fast_pool` ve chvíli, kdy byla přimíchána do generátoru. Na první pohled tato data vypadají jako náhodná sekvence bytů. Rozhodl jsem se použít program ENT¹ pro otestování, zda jsou data opravdu (pseudo)náhodná a kolik entropie obsahují. Nejprve bylo data nutné převést do formátu vhodného jako vstup pro ENT. Pro každý boot

1. ENT je program pro testování náhodnosti obsažené v souboru, se kterým pracuje jako s proudem bytů. Dostupné na <http://www.fourmilab.ch/random/>.

5. ANALÝZA

	HW 1	HW 2	HW 3	HW 4	HW 5
Entropie	7,870207	7,873353	7,869666	7,896836	7,874780
Komprese	1 %	1 %	1 %	1 %	1 %
Chi-square	283,92	251,02	289,45	222,55	265,98
	10,30 %	55,86 %	6,80 %	92,97 %	30,54 %
Stř. hodnota	128,2010	128,0264	130,5120	126,8750	127,1745
Hodnota π	3,073529	2,883333	3,015151	3,090909	3,092936
Chyba π	2,17 %	8,22 %	4,02 %	1,61 %	1,55 %
Korelace	0,009140	0,019122	0,002900	0,029792	0,016188

Tabulka 5.5: Výstup programu ENT pro data posbíraná na reálném počítači.

	Virt 1	Virt 2	Virt 3	Virt 4	Virt 5
Entropie	7,826112	7,480690	7,544851	7,501435	7,482177
Komprese	2 %	6 %	5 %	6 %	6 %
Chi-square	259,83	287,45	241,14	281,93	288,55
	40,44 %	7,94 %	72,42 %	11,86 %	7,29 %
S. hodnota	127,1567	126,3556	126,5223	131,2371	130,7996
Hodnota π	3,132275	3,116883	2,972972	2,805194	3,012987
Chyba π	0,30 %	0,79 %	5,37 %	10,71 %	4,09 %
Korelace	-0,00577	-0,091378	-0,04169	-0,00209	-0,003455

Tabulka 5.6: Výstup programu ENT pro data posbíraná na virtuálním počítači.

jsem vytvořil jeden proud bytů zřetězením všech dat pocházejících ze zdroje irq za sebe. Výstupy programu ENT jsou v tabulkách 5.5 a 5.6.

Ve všech případech obsažená entropie překračuje 7 bitů na 1 byte a kompresí by se velikost souborů s daty zmenšila pouze minimálně. Stejně tak střední hodnota je vždy poměrně blízko 127,5.

Test, který odhaduje hodnotu π pomocí generování náhodných bodů a následně určování, jaká část bodů spadá do určité oblasti, ukázal poměrně velké odchylinky od skutečné hodnoty π . To je do jisté míry způsobeno tím, že odhad π tímto způsobem konverguje ke skutečné hodnotě velmi pomalu. Tedy pro získání přesného odhadu π je potřeba velký objem náhodných dat.[15]

5. ANALÝZA

Chi-square test spočítá Chi-square rozdělení jako absolutní hodnotu² a jak často by skutečně náhodná sekvence tuto hodnotu překročila vyjádřeno v procentech³. Tento test je velmi citlivý a odhalí i drobné chyby při generování pseudonáhodných dat.[15] V 6 případech z 10 je i tento test úspěšný. Ve zbylých 4 případech pak spadají výsledky testu do kategorie „almost suspect“, tedy data ještě nejsou podezřelá, že by nebyla pseudonáhodná, ale není to možné vyloučit. Jedná se o případy HW 3 a 4 a Virt 2 a 5.

Poslední hodnotou v tabulkách je korelační koeficient. Ve všech případech je hodnota velmi blízko 0, tedy každý byte dat je prakticky nezávislý na předchozím bytu.

Z výsledků testů je vidět, že náhodnost dat přidaných do generátoru je poměrně dobrá. Tato přidaná data obsahují více než 112 bitů entropie v každých 16 bytech, které jsou přidány. Při přimíchání dat z `fast_pool` by tedy bylo možné odhad dostupné náhodnosti zvýšit podstatně více než pouze o jeden bit. Protože některá data pocházejí z čítačů, tzn. proměnných, které jsou pouze inkrementovány, dokud nedojde k přetečení, nelze vyloučit, že v datech zůstávají závislosti skryté aplikací míchací funkce. Skutečná hodnota entropie bude s největší pravděpodobností nižší než spočítaných 112 bitů na jeden záznam.

Dalo by se navrhnout, že tento zdroj je potenciálně ovlivnitelný útočníkem, ale pokud je útočník schopen přerušení v systému ovlivnit natolik, že dokáže data jimi přidaná odhadnout s dostatečnou přesností, tak má systém pravděpodobně větší problémy než jeden předvídatelný zdroj náhodnosti.

Přerušení jsou velmi důležitým zdrojem, protože jako první plní `nonblocking_pool` náhodnými daty. Data jsou vkládána před tím, než je do generátoru přidáno semínko uložené před posledním vypnutím systému i než je poprvé dosažen limit 128 bitů v `input_pool` pro `reseed nonblocking_pool`.

2. První číslo v tabulce v řádku Chi-square.
3. Druhé číslo v tabulce v řádku Chi-square.

5. ANALÝZA

5.3.3 Input

Data generovaná vstupními událostmi jsou sbírána ve funkci `add_input_randomness`. Tato funkce do generátoru přidá čas, kdy k události došlo, a popis této události.

- 8 bytů – **jiffies**
- 4 byty – **cycles**
- 4 byty – **num** – Popis události, ke které došlo.

$$num = (type \ll 4) \oplus code \oplus (code \gg 4) \oplus value \quad (5.2)$$

- **type** – Vstupní zařízení, které je zdrojem události.
- **code** – Typ události, ke které došlo.
- **value** – Vlastní hodnota.

Type a code jsou konstanty, které identifikují zařízení a typ události. Tyto konstanty jsou definovány v `include/linux/input.h`. Value je nějaká konkrétní hodnota, true (false) při stisknutí (uvolnění) klávesy nebo tlačítka myši, apod.

Během startu systému se vyskytlo 7 událostí pocházejících od zdroje input na HW počítači a 11 událostí na virtuálním. Z těchto událostí bylo 5 vždy následkem stisknutí klávesy Esc pro zobrazení konzole, ostatní události byly synchronizační. Stisk Esc je v podstatě jediná možnost interakce s počítačem během jeho startu. Tedy během startu systému jsou vstupní události zdrojem pouze velmi malého počtu bitů náhodnosti. Naopak, když uživatel pracuje s běžícím počítačem, proměnná num by mohla být velmi významným zdrojem entropie. Například pohyb myši je kódován v 10 bitech a stisk klávesy v 8 bitech. To je potenciálně 10, respektive 8 bitů entropie.[10] Proměnná num je ale při odhadu počtu přidaných bitů náhodnosti ignorována. Pro odhad se nehodí, protože je snadné její hodnotu přímo ovlivnit, použije se tedy pouze informace o časování dané události.

5.3.4 Disk

Diskové operace jsou zdrojem nejvíce bitů náhodnosti. Data jsou do generátoru přidávána pomocí funkce `add_disk_randomness()`.

- 8 bytů – **jiffies**
- 4 byty – **cycles**
- 4 byty – **num** – Disk, na který se přistupovalo.

$$\begin{aligned} num &= 0x100 + disk_devt(disk) \\ disk_devt(disk) &= ((major \ll 20) | minor) \end{aligned} \quad (5.3)$$

Přidaná data jsou stejná jako data ze zdroje input. Rozdíl je pouze v proměnné num, která v tomto případě slouží jako identifikátor disku, ke kterému se v dané diskové operaci přistupovalo. Disk je identifikován pomocí jeho major a minor čísel.

V datech posbíraných v randlist se objevily pouze dvě různé hodnoty proměnné num.

- **0x00800100** – Major 8, minor 0, to odpovídá /dev/sda, tedy pevnému disku.
- **0x00B00100** – Major 11, minor 0, optická mechanika reprezentovaná zařízením /dev/sr0.

K oběma systémům, na kterých byla data sbírána, byl připojený pouze jeden disk a jedna optická mechanika. Pevný disk byl rozdělen na několik oddílů. Různé oddíly na jednom pevném disku jsou odlišeny minor číslem a sdílejí stejné major číslo. Minor číslo 0 značí zařízení, které reprezentuje celý disk, jednotlivé oddíly mají minor čísla 1 a vyšší.[6]

Je tedy zřejmé, že proměnná num má hodnotu podle toho, ke kterému disku se přistupovalo, jednotlivé oddíly nehrají roli. Lze očekávat, že běžný počítač nebude mít připojeno více než 8 pevných disků nebo optických mechanik. Maximální entropie hodnoty proměnné num by v takovém případě byla 3 bity a je velmi nepravděpodobné, že by rozdělení přístupů k jednotlivým diskům bylo uniformní, tedy entropie by ve skutečnosti byla ještě nižší. Proměnná

5. ANALÝZA

num je stejně jako u zdroje input ignorovaná při odhadu počtu přidaných náhodných bitů, nicméně započítáním této položky by se odhad nijak výrazně nezlepšil, protože tato proměnná obsahuje jenom zanedbatelné množství entropie.

5.3.5 Časování

Kapitoly 5.3.3 a 5.3.4 popsaly, která data se pro odhad náhodnosti nepoužívají, tato část se bude věnovat tomu, z čeho se skutečně přidaná náhodnost odhaduje, a také, jak se to dělá. Jak již bylo řečeno odhad počtu přidaných bitů náhodnosti je určen z informace o časování dané události. K určení času události slouží proměnné jiffies a cycles. Tyto proměnné jsou k datům, která popisují vstupní událost nebo disk, přidána ve funkci `add_timer_randomness()`, která je volaná z `add_input_randomness()` i `add_disk_randomness()`.

Odhad počtu bitů záleží na rozdílu v jiffies mezi časy aktuální události a minulé události. Tento rozdíl je označen jako delta, dále je vypočítána delta druhého rádu jako rozdíl aktuálně vypočítané delta a delta z předchozího volání funkce `add_timer_randomness()` a delta třetího rádu jako rozdíl delty druhého rádu. Z vypočítaných delty se vezme nejmenší absolutní hodnota, která je dále podělena dvěma. Logaritmus tohoto čísla je odhad počtu přidaných bitů náhodnosti, který je dále omezen na maximálně 11 bitů.

Na první pohled nemusí být zřejmé, co zmíněné delty znamenají. Pro lepší vysvětlení, nechť T_i značí čas, kdy nastala událost i , potom:

- $\text{delta} = 0$ znamená, že $T_{i-1} = T_i$, tedy obě události se vyskytly ve stejném čase.
- $\text{delta2} = 0$ znamená, že $T_i - T_{i-1} = T_{i-1} - T_{i-2}$, tedy rozdíl času událostí byl stejný jako v předchozím volání funkce.
- $\text{delta3} = 0$ znamená, že $(T_i - T_{i-1}) - (T_{i-1} - T_{i-2}) = (T_{i-1} - T_{i-2}) - (T_{i-2} - T_{i-3})$, tedy rozdíl rozdílů časů událostí byl stejný jako v předchozím volání funkce.

Zjednodušeně lze říci, že odhad náhodnosti závisí na rozdílu času událostí a aby byl odhad náhodnosti nenulový, nesmí být rozdíly času mezi událostmi konstantní. Tento odhad je poměrně nízký. Pro

	entropie	počet událostí	náhodné bity
HW 1	2,58538	6334	2980
HW 2	2,55261	5739	2566
HW 3	2,47463	6429	2593
HW 4	2,45828	6261	2650
HW 5	2,58469	6460	2893
Virt 1	4,12278	3954	5302
Virt 2	2,66611	3953	2218
Virt 3	2,63116	3904	2132
Virt 4	2,61846	4043	2128
Virt 5	2,77293	4103	2281

Tabulka 5.7: Entropie rozdílů časů událostí pocházejících ze zdrojů input a disk a počet bitů náhodnosti, které tyto události přidaly. Entropie je uvedena v bitech na jeden záznam.

srovnání tabulka 5.7 ukazuje skutečnou entropii rozdílů časů událostí input a disk, počet těchto událostí a také počet odhadnutých bitů náhodnosti, o které byl zvýšen čítač.

Dalším údajem o času události je proměnná cycles. Tato proměnná je při odhadu entropie ignorována. Na rozdíl od čítače jiffies se čítač cyklů mezi dvěma událostmi vždy zvýší. Rozdíl časů vyjádřený jako počet cyklů od spuštění systému by tedy mohl obsahovat více entropie než rozdíl jiffies. Tabulka 5.8 obsahuje entropii rozdílů počtu cyklů událostí. V některých případech je tato entropie až o 10 bitů na jednu událost vyšší než pro jiffies.

V datech přidaných ze zdrojů input a disk je obsaženo více entropie, než o kolik je zvýšen odhad při přidání těchto dat. Z uvedených hodnot vyplývá, že odhad přidaných bitů je skutečně velmi konzervativní a pravděpodobně by mohl být upraven, aby se více blížil skutečné entropii obsažené v těchto datech.

Je také důležité zmínit, že tato data pocházejí z čítačů, navíc je vložena vlastní hodnota čítače, ale entropie je odhadována z rozdílu časů dvou událostí. Skutečná hodnota obsažené entropie tedy pravděpodobně leží mezi aktuálním odhadem a spočítanou hodnotou, kterou lze chápout jako teoretické maximum.

5. ANALÝZA

	entropie
HW 1	12,6002
HW 2	12,4629
HW 3	12,6268
HW 4	12,5778
HW 5	12,6295
Virt 1	11,9413
Virt 2	11,9452
Virt 3	11,9225
Virt 4	11,9777
Virt 5	11,9976

Tabulka 5.8: Entropie rozdílů čítače cyklů v bitech na jednu událost.

5.3.6 Data v `input_pool`

Může se stát, že data v `input_pool` jsou inicializována z větší části stejnými daty. To bylo možné pozorovat v případě, kdy jednotlivé starty počítače byly prováděny v krátkém sledu po sobě, konkrétně se jedná o HW 1 a 2 a také Virt 2, 3, 4 a 5. Většina bytů byla stejná, ale vždy byla přítomna sekvence dlouhá 120 bytů, ve které se od sebe data lišila. V době, kdy jsou tato data poprvé použita jako semínko pro jiný pool, se v datech už žádné sdílené sekvence bytů nenacházejí.

Opakující se sekvence bytů nepředstavují problém při běhu generátoru, pokud se data liší alespoň v některých bytech, protože výstup je výsledkem aplikace hashovací funkce na všechna data v příslušné struktuře. Díky vlastnosti hashovacích funkcí, která se nazývá lavinový efekt, byty, ve kterých se data liší, způsobí, že jednotlivé výstupy hashovací funkce se budou lišit ve velké části bytů.[9] Nicméně pokud by se takové sekvence pravidelně objevovaly za běhu systému, mohlo by to poukazovat na problémy při přidávání dat do struktur generátoru.

Zajímavé je, že z pohledu kvality náhodnosti dat v `input_pool` nehraje odhad, kolik bitů náhodnosti je aktuálně dostupných, žádnou roli. Pro srovnání je v příloze A výstup programu ENT pro data nacházející se v `input_pool` v různých fázích bootovacího procesu. První je během inicializace struktury po přidání aktuálního času

5. ANALÝZA

zdrojem init, druhý v době, kdy byl odhad dostupných náhodnosti nejvyšší (viz tabulka 5.4). Podle výsledků testů je ve všech případech kvalita dat na velmi dobré úrovni a výsledky pro první záznam s odhadem entropie 0 a druhý záznam s odhadem překračujícím 300 bitů jsou prakticky stejné.

6 Závěr

Tato práce popsalala provedenou analýzu semínka linuxového generátoru během startu systému. V první části byl popsán vlastní generátor, jeho struktura, dostupná vstupní a výstupní rozhraní a operace, které generátor provádí. Dále byla popsána implementace v podobě patche jádra systému, která byla použita pro sběr dat. Byl popsán způsob, jakým jsou data sbírána a ukládána, význam těchto dat a také některá omezení při použití tohoto patche. Druhá část se věnovala analýze těchto dat. Kvantitativní analýza odhalila, kolik dat generátor posbírá. Kvalitativní analýza se zabývala kvalitou těchto dat a zejména entropií obsaženou v těchto datech.

Na základě výsledků analýzy je možné konstatovat, že generátor posbírá poměrně velký objem dat. Při přimíchávání těchto dat do generátoru je stanoven odhad, kolik bitů náhodnosti tato data obsahují. Tento odhad udává, kolik bitů náhodných dat může generátor poskytnout procesům, které o tato data žádají použitím jednoho z výstupních rozhraní. Kvantitativní analýza dále odhalila, že generátor nedokáže posbírat dostatek bitů náhodnosti, aby vždy provedl plný reseed sekundárních struktur před tím, než z nich vybere data, která jsou vrácena uživateli jako výstup. Plný reseed znamená přesunutí požadovaného počtu bytů z primární do sekundární struktury. V některých případech, pokud je odhad náhodných bitů v `input_pool` nižší než 192 bitů a data jsou z generátoru vybírána pomocí jednoho z neblokujících rozhraní, dochází k tomu, že reseed sekundární struktury vůbec není možné provést a data jsou vygenerována bez použití čerstvě posbírané entropie. Při jednom z pokusů trval start systému na virtuálním počítači nezvykle dlouhou dobu, to umožnilo generátoru posbírat více než dvojnásobný počet bitů. Ani v tomto případě neměl generátor k dispozici dostatek bitů náhodnosti pro plný reseed, většinou bylo možné pozorovat, že byl vybrán maximální možný objem dat dostupný pro aktuální reseed.

Výsledky kvalitativní analýzy potvrdily, že generátor je velmi konzervativní při odhadování náhodnosti obsažené v přidávaných datech. Vstupní data obsahují více entropie než, kolik je stanoveno odhadem. Následkem toho data získaná čtením `/dev/random` mají zajištěno, že k jejich vygenerování bylo použito velkého počtu ná-

6. ZÁVĚR

hodných bitů a tedy kvalita těchto dat je na dobré úrovni. Na druhé straně toto podhodnocování náhodnosti může způsobit, že při přístupu přes neblokující rozhraní jsou data vygenerována bez použití nově posbírané náhodnosti, protože odhad je příliš nízký, než aby mohl být proveden reseed `nonblocking_pool`.

Tento problém je možné řešit několika způsoby. Kvalitativní analýza ukázala, že by odhad náhodnosti v aktuální podobě mohl být zvýšen, aby se více blížil hodnotě entropie obsažené v datech. Zejména u dat přidaných následkem výskytu přerušení při běhu systému je velká rezerva, ale i odhad na základě času vstupních událostí a událostí generovaných prací disků by mohl lépe odpovídat entropii. Další možnost je zahrnout i proměnné, které jsou při odhadování náhodnosti aktuálně ignorované, například proměnná *cycles*. Proměnná *cycles* obsahuje výrazně více entropie než proměnná *jiffies*. Jedním z možných způsobů, jak *cycles* do odhadu náhodnosti zahrnout je spočítání dvou různých odhadů standardním způsobem. Jeden z *jiffies* a druhý z *cycles* a jako odhad počtu přidaných bitů pak bude použita vyšší z obou hodnot. Za zvážení by možná stalo zahrnout i jiné zdroje náhodnosti. Audio entropy daemon¹ je daemon, který přidává entropii ze statického šumu audio dat, ale je možné najít i daemony využívající jiné zdroje. Aby tyto zdroje mohly být využity během bootu systému, musely by jejich implementace nejspíše být součástí kódu jádra. To může představovat problém a nejprve by musely být podrobeny důkladné analýze.

Zvýšením odhadu náhodnosti by došlo k tomu, že při čtení `/dev/random` by žádající proces nebyl blokován nebo alespoň na kratší dobu než dosud. V aktuální implementaci může čtení `random` blokovat velmi dlouhou dobu zejména na systémech, které nemají k dispozici dostatečně dobré zdroje náhodnosti. Vývojáři aplikací toto často obchází tím, že i při generování kryptografického materiálu používají *urandom*, který potom vrací méně kvalitní náhodná data. Ke zkrácení doby, po kterou `random` blokuje, by mohlo dojít i úpravou limitů. Pokud při čtení `random` chybí pro uspokojení požadavku pouhý 1 bit, `random` blokuje až do posbírání 8 bytů náhodných dat, protože méně není možné přenést. Aplikací výše zmíněných opatření by mohlo dojít ke zvýšení výkonu generátoru.

1. Dostupné online na <http://www.vanheusden.com/aed/>.

Literatura

- [1] Wikipedia, the free encyclopedia: Entropy. Dostupné online na <http://en.wikipedia.org/wiki/Entropy> (prosinec 2012).
- [2] Wikipedia, the free encyclopedia: Entropy (information theory). Dostupné online na http://en.wikipedia.org/wiki/Shannon_entropy (prosinec 2012).
- [3] Wikipedia, the free encyclopedia: Rényi entropy. Dostupné online na http://en.wikipedia.org/wiki/R%C3%A9nyi_entropy (prosinec 2012).
- [4] Linux kernel mailinglist – random: Remove SA_SAMPLE_RANDOM from network drivers, 2006. Dostupné online na <http://thread.gmane.org/gmane.linux.kernel/404007> (prosinec 2012).
- [5] *random(4) – Linux manual page*, 2010. Dostupné online na <http://www.kernel.org/doc/man-pages/online/pages/man4/random.4.html> (prosinec 2012).
- [6] Daniel Bovet and Marco Cesati. *Understanding The Linux Kernel*. O'reilly & Associates Inc, 3rd edition, 2005.
- [7] P.A. Bromiley, N.A. Thacker, and E. Bouhova-Thacker. Shannon entropy, renyi entropy, and information. Technical report, The University of Manchester, 2004. Dostupné online na <http://www.tina-vision.net/docs/memos/2004-004.pdf> (prosinec 2012).
- [8] Robert B Davies. Exclusive OR (XOR) and hardware random number generators. Dostupné online na <http://www.robertnz.net/pdf/xor2.pdf> (prosinec 2012), 2002.
- [9] D. Eastlake 3rd and P. Jones. Us secure hash algorithm 1 (sha1). RFC 3174 (Informational), September 2001. Dostupné online na <https://www.ietf.org/rfc/rfc3174.txt> (prosinec 2012).

- [10] Zvi Guterman, Benny Pinkas, and Tzachi Reinman. Analysis of the linux random number generator. In *Proceedings of the 2006 IEEE Symposium on Security and Privacy*, SP '06, pages 371–385, Washington, DC, USA, 2006. IEEE Computer Society. Dostupné online na <http://eprint.iacr.org/2006/086.pdf> (prosinec 2012).
- [11] Nadia Heninger, Zakir Durumeric, Eric Wustrow, and J. Alex Halderman. Mining your Ps and Qs: Detection of widespread weak keys in network devices. In *Proceedings of the 21st USENIX Security Symposium*, August 2012.
- [12] Makoto Matsumoto and Yoshiharu Kurita. Twisted gfsr generators. *ACM Trans. Model. Comput. Simul.*, 2(3):179–194, July 1992.
- [13] Makoto Matsumoto and Yoshiharu Kurita. Twisted gfsr generators ii. *ACM Trans. Model. Comput. Simul.*, 4(3):254–266, July 1994.
- [14] Theodore Ts'o and Matt Mackall. *random.c – A strong random number generator*. Dostupné online na <https://www.kernel.org/> (prosinec 2012).
- [15] John Walker. *ENT – A Pseudorandom Number Sequence Test Program*, 2008. Dostupné online na <http://www.fourmilab.ch/random/> (prosinec 2012).
- [16] Filip Škola. Semínko generátoru náhodných čísel OS Linux při bootu. Bakalářská práce, Fakulta informatiky Masarykovy univerzity, 2012. Dostupné online na https://is.muni.cz/th/325197/fi_b (prosinec 2012).

A Výstup ENT pro data v input_pool

A.1 HW 1

První záznam

Entropy = 7.567643 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 269.00, and randomly would exceed this value 26.16 percent of the times.

Arithmetic mean value of data bytes is 128.1328 (127.5 = random). Monte Carlo value for Pi is 3.294117647 (error 4.86 percent). Serial correlation coefficient is 0.027021 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.630319 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 241.00, and randomly would exceed this value 72.64 percent of the times.

Arithmetic mean value of data bytes is 122.8086 (127.5 = random). Monte Carlo value for Pi is 3.388235294 (error 7.85 percent). Serial correlation coefficient is -0.009371 (totally uncorrelated = 0.0).

A.2 HW 2

První záznam

Entropy = 7.537396 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 289.00, and randomly would exceed this value 7.04 percent of the times.

Arithmetic mean value of data bytes is 124.6875 (127.5 = random). Monte Carlo value for Pi is 3.294117647 (error 4.86 percent). Serial

A. VÝSTUP ENT PRO DATA V INPUT_POOL

correlation coefficient is 0.040753 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.626653 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 244.00, and randomly would exceed this value 67.89 percent of the times.

Arithmetic mean value of data bytes is 130.5059 (127.5 = random).

Monte Carlo value for Pi is 2.917647059 (error 7.13 percent). Serial correlation coefficient is -0.000250 (totally uncorrelated = 0.0).

A.3 HW 3

První záznam

Entropy = 7.611506 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 238.00, and randomly would exceed this value 77.05 percent of the times.

Arithmetic mean value of data bytes is 124.7051 (127.5 = random).

Monte Carlo value for Pi is 3.247058824 (error 3.36 percent). Serial correlation coefficient is -0.013240 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.578391 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 267.00, and randomly would exceed this value 29.02 percent of the times.

Arithmetic mean value of data bytes is 123.0254 (127.5 = random).

Monte Carlo value for Pi is 3.247058824 (error 3.36 percent). Serial correlation coefficient is -0.003102 (totally uncorrelated = 0.0).

A.4 HW 4

První záznam

Entropy = 7.563963 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 259.00, and randomly would exceed this value 41.85 percent of the times.

Arithmetic mean value of data bytes is 130.2051 (127.5 = random). Monte Carlo value for Pi is 2.917647059 (error 7.13 percent). Serial correlation coefficient is -0.008857 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.547291 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 281.00, and randomly would exceed this value 12.64 percent of the times.

Arithmetic mean value of data bytes is 133.4102 (127.5 = random). Monte Carlo value for Pi is 3.011764706 (error 4.13 percent). Serial correlation coefficient is -0.007479 (totally uncorrelated = 0.0).

A.5 HW 5

První záznam

Entropy = 7.605458 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 243.00, and randomly would exceed this value 69.51 percent of the times.

Arithmetic mean value of data bytes is 126.8926 (127.5 = random). Monte Carlo value for Pi is 3.105882353 (error 1.14 percent). Serial correlation coefficient is 0.077580 (totally uncorrelated = 0.0).

A. VÝSTUP ENT PRO DATA V INPUT_POOL

Maximální entropie

Entropy = 7.623219 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 228.00, and randomly would exceed this value 88.70 percent of the times.

Arithmetic mean value of data bytes is 128.2012 (127.5 = random). Monte Carlo value for Pi is 3.294117647 (error 4.86 percent). Serial correlation coefficient is -0.032267 (totally uncorrelated = 0.0).

A.6 Virt 1

První záznam

Entropy = 7.620566 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 249.00, and randomly would exceed this value 59.41 percent of the times.

Arithmetic mean value of data bytes is 129.1367 (127.5 = random). Monte Carlo value for Pi is 3.105882353 (error 1.14 percent). Serial correlation coefficient is -0.005596 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.566636 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 265.00, and randomly would exceed this value 32.04 percent of the times.

Arithmetic mean value of data bytes is 131.4512 (127.5 = random). Monte Carlo value for Pi is 3.011764706 (error 4.13 percent). Serial correlation coefficient is -0.040927 (totally uncorrelated = 0.0).

A.7 Virt 2

První záznam

Entropy = 7.603171 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 243.00, and randomly would exceed this value 69.51 percent of the times.

Arithmetic mean value of data bytes is 123.5996 (127.5 = random). Monte Carlo value for Pi is 3.058823529 (error 2.63 percent). Serial correlation coefficient is 0.041312 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.583854 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 266.00, and randomly would exceed this value 30.51 percent of the times.

Arithmetic mean value of data bytes is 119.3691 (127.5 = random). Monte Carlo value for Pi is 3.152941176 (error 0.36 percent). Serial correlation coefficient is 0.053501 (totally uncorrelated = 0.0).

A.8 Virt 3

První záznam

Entropy = 7.610712 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 4 percent.

Chi square distribution for 512 samples is 241.00, and randomly would exceed this value 72.64 percent of the times.

Arithmetic mean value of data bytes is 123.2500 (127.5 = random). Monte Carlo value for Pi is 3.152941176 (error 0.36 percent). Serial correlation coefficient is -0.000618 (totally uncorrelated = 0.0).

A. VÝSTUP ENT PRO DATA V INPUT_POOL

Maximální entropie

Entropy = 7.585587 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 243.00, and randomly would exceed this value 69.51 percent of the times.

Arithmetic mean value of data bytes is 125.0996 (127.5 = random). Monte Carlo value for Pi is 3.011764706 (error 4.13 percent). Serial correlation coefficient is 0.050086 (totally uncorrelated = 0.0).

A.9 Virt 4

První záznam

Entropy = 7.589506 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 254.00, and randomly would exceed this value 50.59 percent of the times.

Arithmetic mean value of data bytes is 125.0449 (127.5 = random). Monte Carlo value for Pi is 3.105882353 (error 1.14 percent). Serial correlation coefficient is 0.028038 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.563864 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 282.00, and randomly would exceed this value 11.80 percent of the times.

Arithmetic mean value of data bytes is 127.7715 (127.5 = random). Monte Carlo value for Pi is 3.011764706 (error 4.13 percent). Serial correlation coefficient is 0.002869 (totally uncorrelated = 0.0).

A.10 Virt 5

První záznam

Entropy = 7.587646 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 265.00, and randomly would exceed this value 32.04 percent of the times.

Arithmetic mean value of data bytes is 126.0820 (127.5 = random). Monte Carlo value for Pi is 3.105882353 (error 1.14 percent). Serial correlation coefficient is -0.011813 (totally uncorrelated = 0.0).

Maximální entropie

Entropy = 7.599355 bits per byte.

Optimum compression would reduce the size of this 512 byte file by 5 percent.

Chi square distribution for 512 samples is 250.00, and randomly would exceed this value 57.66 percent of the times.

Arithmetic mean value of data bytes is 127.6777 (127.5 = random). Monte Carlo value for Pi is 2.917647059 (error 7.13 percent). Serial correlation coefficient is 0.003422 (totally uncorrelated = 0.0).

B Obsah přiloženého CD

- text práce
- patch linuxového jádra
- data pro analýzu
 - výstup randlist
 - systémové logy
- grafy 5.1, 5.2 a 5.3