Yarrow

NÁHODNÝ GENERÁTOR YARROW

# Rukavice hozená hackerům

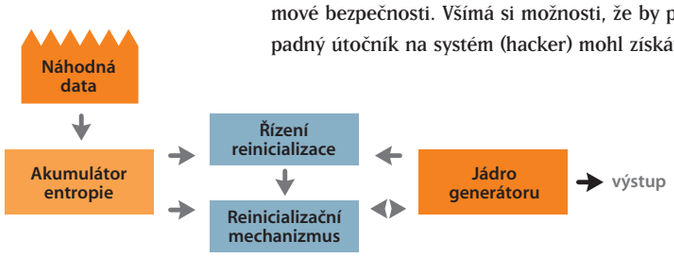
**Implementace generátoru náhodných čísel v osobním počítači je na první pohled snadno řešitelná. Chceme-li však takový generátor použít v bezpečnostních aplikacích, jako je například autentizace, šifrování a elektronický obchod, pak je to docela slušný oříšek. Seznámíme vás s rodinou generátorů Yarrow, jejichž nespornou předností je přímá orientace nejen na kryptografickou kvalitu, ale i na systémově-počítačovou bezpečnost.**

V minulém Chipu (str. 54–56) jsme uvedli základní principy moderní kryptografie pro tvorbu kvalitních generátorů náhodných čísel. Ukázali jsme si i příklady generátorů založených na radioaktivním rozpadu či na napěťově-proudových změnách způsobených tepelným šumem a kvantovými jevy v polovodičových strukturách. Pokud takové zdroje v počítači nemáme, můžeme slevit z požadavku náhodnosti a nahradit ho požadavkem nepredikovatelnosti – ostatně právě to od náhodných generátorů vlastně očekáváme. Proto jsme také mohli generování náhodných čísel přesunout na kryptografické generátory a zůstal nám jen úkol získat zdroj entropie pro jejich počáteční nastavení. V tomto článku budeme používat pojmy hašovací funkce, blokové šifry apod. Pokud byste si je chtěli osvěžit, v infotipech na ně naleznete příslušné odkazy.

## KRYPTOGRAFICKY BEZPEČNÝ

Generátor ***Yarrow*** si klade za cíl generovat na počítači **kryptograficky bezpečná náhodná čísla**. Jeho autoři (Kelsey, Schneier a Ferguson) se dlouhodobě zabývají počítačovou bezpečností a aplikovanou kryptografií. Rodinu generátorů Yarrow (a jejího konkrétního představitele, *Yarrow-160*) navrhli poté, co ve své analýze existujících pseudonáhodných generátorů zjistili různé chyby. V infotipech naleznete dokumenty, které obsahují jak podrobný popis rodiny generátorů Yarrow, tak analýzu existujících generátorů.

Yarrow (viz obr. 1) realizuje obecný myšlenkový postup tvorby náhodných generátorů, který jsme uvedli v minulém čísle (sběr entropie, její destilace, nastavení kryptografického generátoru do neurčitého počátečního stavu, generování náhodných čísel), ale jde dále v úvahách o systémové bezpečnosti. Všímá si možnosti, že by případný útočník na systém (hacker) mohl získávat důležité informace o činnosti generátoru, jako například zjišťovat mezistavy, ovlivňovat zdroje neurčitosti a podobně. Yarrow je proto něco více než jen teoretický stroj na produkci náhodných čísel. Má některé zajímavé zvláštnosti, pro něž stojí za to se s ním seznámit. Na druhé straně otázku systémové bezpečnosti nemůže jednou provždy uspokojivě vyřešit, protože se každý systém chová jinak, a proto je zde ještě volné pole pro další myšlenky a zdokonalení.




Obr. 1. Obecná struktura PRNG typu Yarrow

## YARROW PROTI HACKERŮM

Ostatní modely chápaly PRNG jako černou skříňku, do níž není příliš vidět. Tím nijak nepopíráme principy, které jsme uvedli minule – například, že útočník má úplný popis PRNG, má k dispozici i stroj, na němž běží, apod., ale myslíme tím, že řada PRNG nepřipouští, že by do jejich černé skříňky mohlo být občas vidět za chodu apod. Konkrétně je z hlediska praktické bezpečnosti vhodné zvažovat i situace, kdy útočník

- může částečně znát nebo částečně ovlivňovat vstupy do generátoru,
- občas může zjistit některý minulý vnitřní stav PRNG.

Tyto předpoklady by pro některé existující generátory byly z bezpečnostního hlediska nepřípustné (kompromitovalo by to jejich další produkci). Na Yarrow je sympatické, že bere v úvahu i tyto situace.

Zároveň se Yarrow snaží metodikou svého návrhu předcházet nejčastějším nedostatkům softwarových PRNG. Máme tím na mysli zejména:

- přecenění entropie vstupu;
- nedostatečnou ochranu klíče (vnitřního stavu generátoru);
- implementační chyby (vzniklé složitostí a nepřehledností matematického modelu);
- umožnění nepřímých útoků (útoky na bázi analýzy spotřeby času, energie, diferenční útoky ap.).

## GENEROVÁNÍ NÁHODNÝCH ČÍSEL

Těžiště Yarrow spočívá v tvorbě náhodného vektoru SEED a **nastavení** a **obnově nastavení** kryptografického generátoru. Tyto činnosti po-

píšeme později. Nyní předpokládejme, že už máme vygenerován náhodný vektor SEED o 160 bitech a popíšeme si proces tvorby náhodných čísel. Yarrow-160 k tomu používá kryptografický generátor založený na blokové šifře *TripleDES* (viz infotipy) se třemi klíči. Je tedy potřeba celkem $3 \times 56 = 168$ bitů klíče K, z nichž 160 tvoří právě „dodaná" hodnota SEED a zbylých osm se stanoveným způsobem dopočítá. (Poznamenejme, že ve skutečnosti se z hodnoty SEED odvozuje 192 bitů klíče, aby se tak pokryly i paritní bity klíče pro DES, které by se jinak musely „ručně" doplňovat.)

Yarrow používá TripleDES v tzv. *counter modu* (viz obrázek 2). Počáteční hodnota čítače je určena jako Counter = $E_K(0)$, kde $E_K(x)$ označuje zašifrování bloku x. Poté se už podle schématu na obrázku generují náhodné bloky PRNG(i), které tvoří výstup generátoru. Těchto bloků je možné vygenerovat maximálně Pg; tzv. *systémově bezpečnostní parametr* **Pg** by měl splňovat podmínku $1 \leq Pg \leq 2^{n/3}$, kde **n** je délka bloku použité blokové šifry, což má zaručit, aby produkce generátoru nebyla statistickými testy odlišitelná od náhodné posloupnosti.

Zde se na okamžik zastavme. Režim blokové šifry, který je zde využit, má určitou výhodu. Jde totiž o to, že při procházení čítače (Counter) od 0 do $2^{64}$-1 bude jeho obraz $E_K$(Counter) procházet také všechny možné (a vždy různé) 64tice bitů neboli čísla od 0 do $2^{64}$-1; tak je realizováno bijektivní zobrazení množiny těchto 64bitových čísel na sebe. Z tohoto hlediska má výborné statistické vlastnosti a zaručenou periodu. Oproti skutečně náhodné posloupnosti se ale právě liší svou „přílišnou dokonalostí". Nikdy se u něj totiž nestane, že by vyprodukoval dva stejné bloky, neboť Counter není nikdy stejný! (U náhodných posloupností čas od času dva 64bitové bloky shodné být mohou.)

Aby tedy v counter modu nebyla jeho produkce odlišitelná od náhodné posloupnosti, je nutné využít jen část cyklu, což je oněch zmíněných Pg bloků. U Yarrow-160 je n = 64 a horní hranice pro Pg by tak byla přes 2 miliony. Z důvodů bezpečnosti systému se ale volí pouze Pg = 10 a poté dochází ke změně klíče!

Oč přitom jde? Pokud by se útočník jakýmkoliv způsobem někdy dostal k právě používané hodnotě K nebo k její předchozí hodnotě, mohl by trasovat veškerou činnost generátoru po dobu Pg bloků. Proto se Pg volí tak nízké a po využití každých Pg bloků dochází vynuceně ke změně klíče. Tuto tzv. *redefinici* (v originále „zarážku") obstarává jednosměrná funkce, což útočníkovi znemožní trasovat činnost generátoru do minulosti.

Redefinice se provede tak, že po využití Pg = 10 bloků se následující tři hodnoty výstupu, tj. PRNG(Pg+1), PRNG(Pg+2) a PRNG(Pg+3), použijí vnitřně (nevystupují z generátoru) pro definici nové hodnoty klíče K. Hodnotu čítače přitom není potřeba měnit, protože se změní klíč, takže hodnota Counter se kontinuálně zvyšuje i při redefinici klíče. Aby se útočník v případě, že se dostane k hodnotě klíče K, neradoval příliš dlouho (umožnilo by mu to trasovat činnost generátoru dopředu), zavádí se další opatření. Je to tzv. *periodická reinicializace generátoru*, při níž je nový klíč tvořen také s využitím nového vstupu entropie.

### Inicializace a reinicializace generátoru

V tomto odstavci budeme hovořit o reinicializaci, ale zároveň tím definujeme i počáteční inicializaci generátoru, která je vyvolána úplně stejně. Jde o to, že generátor je periodicky přerušován dvěma procesy sběru entropie, tzv. *rychlou a pomalou bankou*. Jakmile jedna banka nasbírá dostatečné množství entropie (řekněme, že je reprezentované řetězcem SEED), vyvolá přerušení činnosti generátoru a reinicializuje ho. Předtím se stará hodnota klíče K smísí pomocí hašovací funkce s řetězcem SEED. Výsledek označme jako SEED'. Při reinicializaci dochází k nové definici klíče K i čítače Counter na základě nově získané entropie a staré hodnoty klíče – viz obrázek 3.

### Řízení reinicializace a činnost bank entropie

Předně je třeba říci, že označení „rychlá" a „pomalá" banka v tomto případě nesouvisí s rychlostí, jakou do těchto akumulá-

**Požadavek náhodnosti lze u PRNG bez velké újmy na bezpečnostní kvalitě nahradit požadavkem nepredikovatelnosti.**

```
Dokud nedojde k přerušení od bank
entropie, opakuj následující:
{
   //generování výstupu - PRNG(*)
   for i = 1 to Pg do
   {
      Counter = (Counter+1) mod 2^64
      PRNG(i) = E_K(Counter)
   }
   // redefinice klíče
   K ← ( PRNG(Pg+1), PRNG(Pg+2),
         PRNG(Pg+3) )
}
```

Obr. 2. Jádro generátoru

| Reinicializace generátoru: | Výpočet hodnoty SEED': |
|---|---|
| K = SEED'<br>Counter = $E_K(0)$ | SEED' = h'( SHA-1( SEED \|\| K ), 192 )<br>*( \|\| označuje zřetězení ),*<br><br>kde h' definujeme takto:<br>h'(m, k) = prvních k bitů z hodnoty ($s_0$ \|\| $s_1$ \|\| ...),<br><br>kde pro posloupnost hodnot s platí:<br>$s_0$ = m<br>$s_i$ = SHA-1( $s_0$ \|\| ... \|\| $s_{i-1}$ ), pro $i \geq 1$ |

Obr. 3. Reinicializace generátoru

torů entropie přicházejí sbíraná data. Všechny zdroje entropie totiž svůj výstup periodicky alternují mezi oběma bankami. Tyto termíny zde pouze označují, jak často se výstup příslušné banky pro reinicializaci jádra generátoru použije. Častěji se tedy používá výstup rychlé banky, což je dáno způsobem jejího řízení (viz dále).

Předpokládejme, že v systému máme několik (n) zdrojů náhodných veličin. Příkladem může být pohyb myši, psaní na klávesnici, systémová data ap. (více v minulém článku). Z pohledu těchto zdrojů představují obě banky otevřené kontexty hašovací funkce, do kterých se získané náhodné veličiny přidávají způsobem obvyklým pro použitý typ hašovací funkce (zde SHA-1). V okamžiku, kdy má dojít k reinicializaci jádra generátoru, se kontext dané banky uzavře a výsledek je (jako hodnota SEED) výše popsaným způsobem použit pro výpočet nového klíče a registru čítače. V případě, že má dojít k reinicializaci z pomalé banky, je tento scénář ještě mírně modifikován tím, že výsledek z této banky se přidá (jako by to byl běžný zdroj náhody) do rychlé banky, ta se uzavře a výsledek této operace se pak použije jako hodnota SEED.

Filozofie řízení činnosti jednotlivých bank je stejná. U obou bank jsou stanoveny limity entropie (Yarrow-160 má limit 100 bitů pro rychlou a 160 pro pomalou banku) a nad zdroji pracují měřiče. Jakmile *k* zdrojů (z celkového počtu *n* zdrojů) už překročí limit, vyvolá se reinicializace. Pro rychlou banku je u Yarrow-160 stanoveno $k = 1$, pro pomalou $k = 2$. Poté, co je daná banka použita, vytvoří se nový kontext její hašovací funkce a sběr entropie začíná nanovo s vynulovanými měřiči entropie.

Úkolem rychlé banky je umožnit co možná nejrychlejší **nedeterministickou** změnu klíče K (na rozdíl od deterministické zarážky), a tím co nejvíce omezit následky jeho kompromitace. Předpokládá se, že z rychlé banky bude vyvoláno přerušení mnohokrát za hodinu. Pomalá banka zase má svým konzervativním přístupem k entropii jistit její kvalitu a čas od času generátor „betonově" znáhodnit.

Maximální počet bloků, které je možné generovat bez přerušení, je $\min(2^n, 2^{k/3}Pg)$, což pro Yarrow-160 představuje $10*2^{56}$ bloků. Po této hodnotě už musí bezpodmínečně dojít k reinicializaci. Návrhář systému proto musí vyřešit sběr entropie a na něj navazující systém řízení obou bank tak, aby se do tohoto limitu vešel. Samo odvození této hodnoty je poměrně snadné a vychází ze dvou zásad:

- nevyčerpat celou množinu hodnot pro čítač (zde $2^n$);
- předejít kolizím klíčů při aplikaci „zarážky" (na množině o velikosti $2^{k/3}$ je tato kolize nepravděpodobná).

### Bezpečnost v praxi

Právě popsaný typ generátoru sice vypadá na první pohled velmi odolně, zejména z úhlu pohledu systémového hackera, avšak realita je v tomto případě poněkud střízlivější. Pokud se rozhodnete takový generátor aplikovat v praxi a zamyslíte se hlouběji nad účinností jeho ochran, zjistíte, že ve skutečnosti držíte v ruce pouze líbivé, teoreticko-alibistické povídání o tom, jak by to vše mohlo fungovat, kdyby...

To jsou poměrně silná slova a jistě je z nich cítit spor s tím, co jsme řekli na začátku. Stala se snad někde chyba? Nestala, ale mohla by! Disciplína označovaná jako aplikovaná kryptografie je totiž komplexní záležitost, a to je třeba mít stále na mysli.

Představme si například situaci, kdy dojde ke kompromitaci klíče. Vzhledem k tomu, že návrh Yarrow uvažoval obecné prostředí pro jeho nasazení, nemohli autoři při nejlepší vůli udělat víc, než vymyslet rychlou banku a operaci časté reinicializace klíče K. A výsledek? Diskutabilní! Pro architekturu většiny praktických zařízení a druhy možných útoků totiž platí, že pokud byl útočník schopen jednou prolomit ochranu vnitřních datových struktur PRNG a přečíst si hodnotu klíče K, může to po jeho reinicializaci udělat znovu. Na rozdíl od ryze teoretických úvah autorů, kteří o kompromitaci klíče uvažují jako o náhodném jevu, který s nějakou (velmi malou) pravděpodobností může nastat, je praktická situace většinou mnohem prozaičtější. Buďto něco jde, nebo to nejde. A když to jde, tak to jde pokaždé. Situace, kdy by hacker nemohl svůj útok opakovat, jsou velmi řídké!

Existuje tedy vůbec řešení této na první pohled patové situace? Ano, existuje. Abychom je našli, musíme vzít v úvahu všechny bezpečnostní mechanismy, které nám daná architektura nabízí. Teprve jejich vzájemným propojením můžeme nakonec dosáhnout uspokojivé úrovně zabezpečení. Pro ilustraci se vraťme k příkladu s kompromitací klíče K. Je logické, že paměťové oblasti, v níž je tato hodnota uložena, budeme muset poskytnout jistou úroveň systémové ochrany. Pokud bychom ji totiž nechali volně přístupnou, samozřejmě nemůžeme očekávat, že takto implementovaný PRNG bude vykazovat slušnou úroveň zabezpečení.

Jakou konkrétní úroveň ochrany tedy klíči K poskytneme? Budeme-li na ní šetřit, zbývá jen doufat, že k jejímu prolomení nedojde tak často, aby reinicializace z pomalé banky nestačila tyto incidenty pokrýt. To je v souvislosti s tím, co jsme si o charakteru běžných průniků (všechno, nebo nic) řekli, přinejmenším odvážné. Mnohem lepší je poskytnout klíči K tu nejvyšší možnou úroveň systémové ochrany, jaká je v daném zařízení k dispozici.

Jako příklad můžeme uvést implementaci PRNG přímo do jádra daného operačního systému. Všechny použitelné operační systémy dnes nabízejí alespoň dvojúrovňový režim práce mikroprocesoru, přičemž rozeznáváme režim jádra a režim uživatelských aplikací. Procesy běžící v jádru operačního systému jsou přitom maximálním možným způsobem odděleny od důsledků potenciálně nebezpečného chování procesů na aplikační úrovni.

Umístíme-li tedy PRNG v podobě nějaké rozšiřující služby do jádra daného operačního systému, můžeme právem očekávat, že k incidentům kompromitace klíče s největší dosažitelnou pravděpodobností

### infotipy

**Předběžný popis Yarrow:**
*www.counterpane.com/yarrow.html*
**Bezpečnostní analýza existujících generátorů:**
Kelsey, Schneier, Wagner, Hall: Cryptanalytic Attacks on Pseudorandom Number Generators, Fast Software Encryption, 5th Int. Workshop Proceedings, Springer-Verlag, 1998, str. 168 – 188.
**Další pseudonáhodné generátory:**
ANSI X9.17, American National Standard for Finantial Institution Key Management (Wholesale), ABA, 1985.
Gutmann P.: Software Generation of Random Numbers for Cryptographic Purposes, Proceedings of the 1998 Usenix security Symposium, 1998, str. 243 – 257.
**Hašovací funkce SHA-1:**
Chip 3/99, str. 40 – 43.
**DES (TripleDES):**
Chip 5/93, str. 52 – 58.

nedojde. Navíc stále platí zmíněný teorém „všechno, nebo nic“, který nám dovoluje poněkud slevit z nároků na řízení akumulátorů entropie a neprovádět reinicializaci tak často. Proč? Jednoduše proto, že dokud nedojde k narušení ochrany jádra systému, není důvod předpokládat, že by klíč K byl odchycen, a tudíž je zběsilé provádění reinicializace jen zpomalujícím faktorem. Pokud naopak k porušení ochrany jádra dojde, potom sice pravděpodobně dojde také ke kompromitaci klíče K (pokud to bylo cílem útočníka), avšak ani zde nám častější reinicializace nepomůže. Útočník, který je nyní neomezeným vládcem celého systému, totiž může reinicializace velmi jednoduše monitorovat nebo vyřadit z činnosti, takže jejich přínos je nulový.

Právě jsme si ukázali, že pro účely praktického nasazení generátoru je ve většině případů vhodné nahradit operaci časté (je třeba pouze dodržet výše uvedený maximální počet bloků) reinicializace PRNG dostatečnou úrovní systémové ochrany, která je jednak lepší, jednak už v systému přirozeně existuje, takže neznamená žádnou další spotřebu výkonu (sběr kvalitních náhodných dat přece jen nějaký čas zabere).

Jestliže jste však právě nabyli dojmu, že veškerá bezpečnost se dá vyřešit pouze systémovými prostředky, byl by to zase opačný extrém (což je také špatně), a proto máme na závěr jeden protipříklad. Předpokládejme implementaci PRNG právě popsaným způsobem (tj. v jádru OS). Řekli jsme již, že nemá valný smysl ptát se, jaká bude budoucnost počítače, u kterého se hackerovi podaří prolomit ochranný mechanismus jádra systému. Neptejme se proto po ohrožení budoucnosti, nýbrž minulosti. Jaký bude mít tento útok vliv na předchozí produkci generátoru? Pokud by například nebyla aplikována výše popsaná „zarážka“, potom by útočník mohl zjistit celou minulou produkci PRNG a zjistit tak například všechny vygenerované a použité klíče (ty se totiž většinou odvozují od výstupu PRNG). Žádná systémová ochrana by mu v tom nemohla zabránit. Protože je však „zarážka“ použita, nemůže útočník takový zpětný výpočet provést – minulost je tak zachráněna.

Podobně jako v prvním případě pomohla systémová ochrana kryptografii, zde zase pomohla kryptografie eliminovat důsledky porušení ochrany operačního systému – a přesně tak to má ve správně navržené bezpečnostní architektuře být.

## Závěr

Yarrow je představitelem moderních generátorů náhodných čísel, které jsou založeny na kryptografických metodách. Jako jeden z prvních bere vážně v úvahu i bezpečnostní vlastnosti systému a možné hrozby útočníků. Z tohoto hlediska je to jeden z prvních kvalitních generátorů náhodných čísel v osobním počítači, a jako takový rozhodně stojí za pozornost. Nesmíme ale zapomenout, že návrh kvalitního generátoru tvoří pouze část výsledného systému a že leckdy můžeme stejný druh obranného mechanismu realizovat za jiných okolností mnohem lépe jinými prostředky. Musíme proto vzít v úvahu vlastnosti všech spolupracujících celků a vhodně je mezi sebou propojit. Jedině tak nakonec obdržíme kvalitně zabezpečený systém.

A snad ještě malou pozoruhodnost, pokud vás zajímá, jak tento generátor přišel ke svému jménu: ve staré Číně se k věštění jako „randomizér“ (komplikovaně a s bídnými statistickými vlastnostmi) používaly stonky byliny zvané řebříček, anglicky *yarrow*....

Vlastimil Klíma (v.klima@decros.cz),
Tomáš Rosa (t.rosa@decros.cz)