Um estudo empírico de PRNGs utilizados em *shell scripts*

João Otávio Massari Chervinski[1], **Vinicius Nunez**[1], **Diego Kreutz**[1]

[1]Ciência da Computação (CC) e Laboratório de Estudos Avançados (LEA)
Universidade Federal do Pampa (UNIPAMPA)

{joaootaviors,viniciuslopes.vn}@gmail.com,kreutz@unipampa.edu.br

Abstract. *Pseudorandom number generators, a.k.a. PRNGs, are essential to the inner workings of many systems such as games, simulation algorithms, and security mechanisms. As a way to contribute to the use of PRNGs in practice, this work aims to investigate the quality and performance of the PRNGs most commonly used by shell scripts developers on GNU/Linux systems. Our results indicate a lack of understanding and technical knowledge in security (and the importance of PRNGs) on the part of programmers and developers who are responsible for systems quality and security.*

Resumo. *Geradores de números pseudo-aleatórios, mais conhecidos como PRNGs, são essenciais para o bom funcionamento de sistemas como jogos, ambientes de simulação e mecanismos de segurança. Como forma de contribuir na conscientização do uso de PRNGs na prática, este trabalho tem como objetivo investigar a qualidade e o desempenho dos PRNGs mais utilizados por desenvolvedores de shell scripts em ambientes GNU/Linux. As análises realizadas apontam uma falta de informação e conhecimento técnico sobre segurança (e a importância dos PRNGs) por parte dos programadores e responsáveis pela qualidade ou segurança dos sistemas.*

1. Introdução

Os PRNGs (*Pseudo-Random Number Generators*), ou geradores de números pseudo-aleatórios, são ferramentas essenciais para o funcionamento de uma grande gama de sistemas, como jogos, sistemas de sorteio, sistemas de simulação como o algoritmo de Monte Carlo e os algoritmos genéticos, urnas eletrônicas, sistemas de criptografia, geração de chaves, entre outros. Na maioria dos casos de uso, uma das características mais imprescindíveis para um PRNG é apresentar uma distribuição dos valores pseudo-aleatórios o mais próxima possível de uma distribuição uniforme.

Apesar de a distribuição uniforme ser um dos principais parâmetros para aplicações mais simples, como sistemas de sorteio e simulação, existem outros métodos estatísticos que são aplicados a PRNGs para determinar a qualidade do gerador para casos mais críticos, como mecanismos e protocolos de criptografia. Para que um PRNG possa ser considerado apropriado para o uso em aplicações onde os mecanismos de segurança são um aspecto essencial, ele deve passar por uma bateria de testes empíricos que servem de subsídio para determinar a qualidade do gerador [Luizi et al. 2010, Maksutov et al. 2018, Prokofiev et al. 2018, Kreutz et al. 2017].

Para a realização de sorteios de brindes, alguns eventos utilizam *shell scripts* que realizam a seleção dos ganhadores a partir de uma lista de inscritos e da geração de

números pseudo-aleatórios. A utilização de linguagens de *scripting* é interessante pelo fato de existirem várias opções nativas aos sistemas GNU/Linux, potencializarem a alta produtividade, serem práticas para a automação de tarefas e oferecerem várias alternativas práticas de PRNGs. Este é o caso do sistema de sorteio de brindes, baseado em Bourne-Again Shell (BASH), do clube Parceria Indoor de Alegrete-RS, o estudo de caso deste trabalho (a versão 0.1 do código está disponível em [Chervinski et al. 2018]).

Ao selecionar um PRNG sem uma análise prévia da qualidade do mesmo, é difícil garantir que o sorteio será justo e confiável. É importante que o desenvolvedor do *shell script* utilize um PRNG de boa qualidade para não favorecer um sub-conjunto de participantes do sorteio. Na prática, um PRNG de baixa qualidade pode gerar números pseudo-aleatórios distantes de uma distribuição uniforme e/ou tendenciosos, ou seja, números que irão beneficiar determinados participantes do sorteio (exemplo: os participantes com os maiores números do intervalo de sorteio).

Dado o contexto apresentado, os objetivos principais deste trabalho são: pesquisar os PRNGs mais utilizados na prática por desenvolvedores de *shell scripts*; reproduzir e avaliar os geradores utilizando métodos estatísticos recomendados na literatura; auxiliar na escolha adequada de PRNGs para aplicações que necessitem de resultados transparentes e confiáveis, como sistemas de sorteios de brindes; classificar e recomendar geradores de acordo com o cenário de aplicação. Num primeiro momento, foram realizadas pesquisas na *web* para levantar informações sobre os geradores mais utilizados. Na parte de análise, os números pseudo-aleatórios dos PRNGs foram avaliados utilizando métodos estatísticos (e.g. distribuição uniforme) e a ferramenta *Dieharder*.

As principais contribuições deste trabalho podem ser resumidas em: (i) identificação dos PRNGs mais utilizados na prática por desenvolvedores de *shell scripts*; (ii) classificação dos PRNGs de acordo com a qualidade segundo diferentes critérios, métodos e ferramentas estatísticas disponíveis na literatura; (iii) identificação e caracterização de uma análise de baixa confiabilidade de PRNGs; (iv) discussão da importância da avaliação e escolha cuidadosa de PRNGs de qualidade; e (v) identificação do PRNG mais adequando para o *shell script* de sorteio utilizado pelo clube Parceria Indoor.

O restante do trabalho está dividido da seguinte maneira. Nas Seções 2 e 3 são apresentados os trabalhos relacionados e os métodos de seleção e os PRNGs avaliados. Nas Seções 4 e 5 são discutidas a implementação e os resultados obtidos, respectivamente. Por fim, as considerações finais são apresentadas na Seção 6.

2. Trabalhos Relacionados

Existem diferentes estudos que realizam análises da qualidade de PRNGs utilizando técnicas variadas, como distribuição binomial [Luizi et al. 2010], visualização multidimensional da distribuição de números gerados [Prokofiev et al. 2018], diferentes métodos gráficos [Prokofiev et al. 2017, Chugunkov and Muleys 2014], entre outras análises baseadas em diferentes métodos estatísticos [Dodis et al. 2013, Kreutz et al. 2017, Schaathun 2015, Haramoto 2009]. Apesar de existirem métodos baseados em visualização e outras técnicas gráficas, os métodos mais amplamente utilizados e aceitos são os estatísticos. O próprio *National Institute of Standards and Technology* (NIST), que é um das principais entidades internacionalmente reconhecidas pelas recomendações técnicas na área de segurança, disponibiliza um pacote de testes

estatísticos, conhecido como *NIST Statistical Test Suite* [NIST 2017], para avaliar a qualidade de PRNGs.

A maioria dos trabalhos encontrados na literatura investiga aspectos qualitativos de um ou um subconjunto específicos de PRNGs [Luizi et al. 2010, Schaathun 2015, Chugunkov and Muleys 2014, Kreutz et al. 2017, Haramoto 2009, Dodis et al. 2013]. Como exemplo, há trabalhos que avaliam PRNGs denominados *splittable* (S-PRNG) [Schaathun 2015]. O foco desses PRNGs é o desempenho, explorando paralelismo em nível de algoritmo e hardware, e o desafio é garantir as propriedades do gerador nesse contexto. Outros trabalhos avaliam algoritmos específicos, como BBS, ANSI X 9.17, LCG e QCG-II [Luizi et al. 2010]. Em resumo, nenhum dos trabalhos encontrados na literatura realiza uma análise da qualidade dos PRNGs utilizados na prática, em especial no que diz respeito à programação *shell scripting*, que é o foco deste trabalho.

3. Seleção dos PRNGs

Como método de selecionar as principais soluções utilizadas na prática, por desenvolvedores de *shell scripts*, para a geração de números pseudo-aleatórios, foram realizadas pesquisas em três plataformas de busca distintas, Google Search, DuckDuckGo e Bing. Dos resultados obtidos nas buscas, foram selecionados os 44 primeiros sites e fóruns de discussão contendo código de geradores de números pseudo-aleatórios. Nesses 44 sites foram identificados 4 geradores, sendo que em alguns sites haviam discussões e exemplos de código de múltiplos geradores. Os 4 geradores e as respectivas frequências de utilização, segundo os 44 sites, são: RANDOM (81.81%), URANDOM (25%), SHUF (11.36%) e DATE_N (2.27%). Além dos 4 geradores, foram implementados outros 3 geradores (RANDOM_DATE, DATE_AMD, SHUF_URANDOM) com base em recomendações de desenvolvedores locais e combinação de geradores encontrados nos sites. Devido a limitação de espaço, os detalhes técnicos dos 7 geradores estão disponíveis em [Chervinski et al. 2018].

4. Implementação

Os 7 geradores selecionados foram codificados em um *shell script* parametrizável, utilizado para geração e posterior análise dos números pseudo-aleatórios. O *shell script* desenvolvido permite a seleção da quantidade de números a serem gerados, o intervalo ao qual os números gerados pertencerão e o PRNG que será utilizado para gerá-los.

Para a análise dos dados, foi utilizada a linguagem de programação Python em sua versão 3.6.3 e a biblioteca *matplotlib*. Após a geração dos números pseudo-aleatórios, o programa Python organiza o conjunto de dados gerado por cada um dos PRNGs em intervalos (também conhecidos como *bins*) para exibição em um histograma e realiza o cálculo do desvio padrão e do erro padrão dos intervalos. Além disso, o programa desenvolvido também oferece a opção de realizar um *benchmark* dos PRNGs. Um *benchmark* consiste em executar o PRNG selecionado por um número de vezes definido pelo usuário e calcular a média do desvio padrão e do erro padrão considerando os resultados de todas as iterações. O código utilizado para gerar os gráficos e analisar os dados e o *shell script* dos geradores estão disponíveis em [Chervinski et al. 2018].

5. Resultados

Com o intuito de avaliar e classificar os PRNGs, foram realizadas quatro análises distintas. (a_1) Visualização e discussão da distribuição dos valores de cada PRNG em um histograma e análise dos valores de desvio e erro padrão em cada caso. Para esta análise foram gerados 10.000 números em um intervalo fechado de 0 à 1.000. (a_2) Análise da média do desvio padrão e do erro padrão para 40 execuções de cada PRNG. Os geradores foram executados utilizando o *benchmark* do programa Python descrito na Seção 4 com os mesmos parâmetros da análise (a_1). (a_3) Análise de desempenho dos PRNGs. Aferição do tempo de execução de cada gerador para 100.000 números em um intervalo fechado de 0 à 10.000. (a_4) Avaliação da qualidade dos geradores, para aplicação em sistemas criptográficos, utilizando a bateria de testes padrão da ferramenta *Dieharder*, que executa todos os testes disponíveis utilizando parâmetros pré-definidos. Os mesmos 100.000 números do item (a_3) foram utilizados como entrada para a avaliação dos geradores com a ferramenta *Dieharder*. Os resultados apresentados a seguir foram coletados em uma máquina com o sistema operacional Ubuntu 16.04.5 LTS, processador Core i5-7500 Quad-Core 3.4 Ghz, 16 GB de RAM DDR4 2133Mhz e disco rígido de 1TB e 7200RPM.

As Tabelas 1 e 2 resumem os resultados das análises (a_1), (a_2) e (a_3). Como pode ser observado na primeira tabela, os PRNGs SHUF e RANDOM apresentam os melhores resultados em termos de desvio padrão e erro padrão. A análise (a_1) (Tabela 1) mostra a dispersão dos resultados dos PRNGs através dos histogramas e do desvio padrão. Para o desvio, são consideradas como amostras as barras do histograma (os histogramas estão disponíveis em [Chervinski et al. 2018]), onde cada uma representa a quantidade de valores gerados em um intervalo de 50 números. Entretanto, foi observado que a análise é de baixa confiabilidade uma vez que ocorrem oscilações a cada execução dos PRNGs. A análise (a_2), Tabela 2, tem como objetivo apresentar uma avaliação de maior confiabilidade através da utilização de um *benchmark* mais robusto para avaliar os PRNGs. A ordem de classificação dos geradores muda completamente quando comparada com os resultados apresentados na Tabela 1. Tomando como exemplo o RANDOM, ele passa da segunda para a penúltima posição.

Tabela 1. Desvio padrão e erro padrão dos PRNGs segundo a análise (a_1).

PRNG	Desvio Padrão	Erro Padrão
SHUF	17,55	3,82
RANDOM	19,27	4,31
RANDOM_DATE	21,88	4,88
DATE_AMD	22,23	4,97
DATE_N	22,38	5,0
URANDOM	23,0	5,1
SHUF_URANDOM	23,18	5,18

A análise (a_3) mede o desempenho dos PRNGs considerando a geração de 100.000 números pseudo-aleatórios. Os resultados da (a_2) e (a_3) são apresentados na Tabela 2. Como pode ser observado, há uma diferença significativa entre os geradores. O SHUF,

por exemplo, apresenta boa qualidade e o menor tempo de execução, sendo um candidato forte para sistemas que precisam gerar milhares de números pseudo-aleatórios.

Tabela 2. Desvio padrão, erro padrão e tempo de execução dos PRNGs

PRNG	Desvio Padrão Médio	Erro Padrão Médio	Tempo (100.000 números)
RANDOM_DATE	20,654	4,721	2m59,990s
DATE_AMD	21,672	4,846	9m49,676s
SHUF	21,915	4,900	0,066s
URANDOM	22,141	4,951	2m20,869s
SHUF_URANDOM	22,362	5,000	0,045s
RANDOM	22,854	5,110	0,997s
DATE_N	23,232	5,194	2m56,915s

Outro ponto a ser observado na análise (a_3) é o fato do gerador SHUF_URANDOM ser muito mais rápido que o gerador URANDOM, embora ambos utilizem a mesma fonte de entropia. Isso deve-se a quantidade de entropia a qual cada um dos geradores precisa coletar. Enquanto o gerador SHUF_URANDOM acessa o arquivo `/dev/urandom` apenas uma vez, o gerador URANDOM acessa o arquivo para cada número gerado, impactando significativamente no tempo de execução.

Tabela 3. Resultados da análise com a ferramenta *Dieharder*

PRNG	Testes aprovados (PASS)	Testes indecisivos (WEAK)
URANDOM	113	1
DATE_AMD	113	1
DATE_N	113	1
SHUF_URANDOM	112	2
RANDOM	111	3
SHUF	109	5
RANDOM_DATE	108	6

Finalmente, a análise (a_4) visa explorar a qualidade dos PRNGs com base na bateria de testes padrão da ferramenta *Dieharder*. A ferramenta executa 114 testes estatísticos sobre o conjunto de números pseudo-aleatórios gerados por um PRNG. Para cada um dos 114 testes, o gerador recebe um veredito, *PASS* indicando sucesso, *WEAK* indicando que podem haver problemas, mas não necessariamente uma falha devido a natureza mutável dos valores de entrada, e *FAILED* indicando uma falha.

Como pode ser observado na Tabela 3, para sistemas criptográficos, o gerador URANDOM é o mais recomendado uma vez que passou 113 testes e possui um tempo de execução menor que o DATE_AMD e o DATE_N (como mostra a Tabela 2). Apesar de o RANDOM ter se saído ligeiramente melhor que o SHUF, é importante ressaltar que o RANDOM gera números pseudo-aleatórios em um intervalo pequeno (0 à 32767), o que o torna uma opção limitada para sistemas que necessitam intervalos maiores, como

sistemas criptográficos. Neste último caso, o RANDOM é desencorajado, uma vez que pode comprometer toda a segurança do sistema devido a limitação do intervalo de número.

Os histogramas dos números pseudo-aleatórios gerados pelos PRNGs, bem como uma análise mais extensa e detalhada, estão disponíveis em [Chervinski et al. 2018].

6. Considerações Finais

Os resultados e discussões mostram que há diferentes *trade-offs* (e.g. qualidade versus desempenho) que devem ser observados pelos programadores na hora de desenvolver uma aplicação. Por exemplo, segundo as análises, os PRNGs SHUF e URANDOM são os mais recomendados para aplicações gerais e sistemas criptográficos, respectivamente. O SHUF possui uma boa qualidade e um ótimo desempenho quando comparado aos demais geradores. Apesar de ser pouco utilizado, o SHUF é um forte candidato a substituir o RANDOM, o PRNG mais utilizado na prática. Entretanto, vale ressaltar que o RANDOM pode ser utilizado na prática, para coisas simples como selecionar aleatoriamente um caracter de uma string, sem nenhuma restrição. Caso a aplicação necessite de números dentro de um intervalo grande ou valores pseudo-aleatórios de qualidade para serem utilizados em mecanismos criptográficos, o RANDOM é desencorajado. Além disso, os resultados indicam que é recomendado substituir o RANDOM, utilizado no *shell script* de sorteio de brindes (ver código em [Chervinski et al. 2018]) do clube Parceria Indoor, pelo SHUF, uma vez que este último gera distribuições mais uniformes de números pseudo-aleatórios.

Referências

Chervinski, J. O. M., Nunez, V., and Kreutz, D. (2018). Um estudo empírico de PRNGs utilizados em shell scripts. `https://goo.gl/W4Gwkq`.

Chugunkov, I. and Muleys, R. (2014). Pseudorandom numbers generators quality assessment using graphic tests. In *IEEE EIConRus*, pages 8–13.

Dodis, Y., Pointcheval, D., Ruhault, S., Vergniaud, D., and Wichs, D. (2013). Security analysis of pseudo-random number generators with input:/dev/random is not robust. In *ACM SIGSAC CCS*, pages 647–658.

Haramoto, H. (2009). Automation of statistical tests on randomness to obtain clearer conclusion. In *Monte Carlo and Quasi-Monte Carlo Methods*, pages 411–421. Springer.

Kreutz, D., Yu, J., Ramos, F. M. V., and Esteves-Verissimo, P. (2017). ANCHOR: logically-centralized security for Software-Defined Networks. *ArXiv e-prints*.

Luizi, P., Cruz, F., and van de Graaf, J. (2010). Assessing the quality of pseudo-random number generators. *Computational Economics*, 36(1):57–67.

Maksutov, A. A., Goryushkin, P. N., Gerasimov, A. A., and Orlov, A. A. (2018). Prng assessment tests based on neural networks. In *IEEE EIConRus*, pages 339–341.

NIST (2017). NIST statistical test suite. `https://goo.gl/qmdPGK`.

Prokofiev, A. O., Chirkin, A. V., and Bukharov, V. A. (2018). Methodology for quality evaluation of prng, by investigating distribution in a multidimensional space. In *IEEE EIConRus*, pages 355–357.

Prokofiev, A. O., Denisov, D. V., and Chirkin, A. V. (2017). The distribution in space test for quality evaluation of pseudorandom numbers generators. In *IEEE EIConRus*.

Schaathun, H. G. (2015). Evaluation of splittable pseudo-random generators. *Journal of Functional Programming*, 25:e6.