

Randomly Failed!

The State of Randomness in Current Java Implementations

Kai Michaelis, Christopher Meyer, and Jörg Schwenk

Horst Görtz Institute for IT-Security, Ruhr-University Bochum
{kai.michaelis,christopher.meyer,joerg.schwenk}@rub.de

Abstract. This paper investigates the Randomness of several Java Runtime Libraries by inspecting the integrated Pseudo Random Number Generators. Significant weaknesses in different libraries including Android, are uncovered.

1 Introduction

With a market share of 33-50% [1], [2], [3] Android is currently the most popular mobile OS. Each of the 331-400 million sold devices (cf. [3]) is able to run Java applications written in Java. Java provides interfaces for different Pseudo Random Number Generators (PRNGs), such as `SecureRandom`, which is intended for use by cryptographic schemes. But, only the API is specified - each library implements own algorithms. Designing secure and reliable PRNGs is a hard and complicated task [4], as well as implementing these algorithms correctly. One of the worst ideas is to implement own unproved PRNG constructs.

This paper examines the quality of the random numbers generated by common Java libraries. In detail, the PRNGs, intended for use in cryptographic environments, of Apache Harmony¹, GNU Classpath², OpenJDK³ and BouncyCastle⁴ are inspected. It is shown that the over-all entropy of the Android PRNG can be reduced to 64 bits. Beyond this, multiple weaknesses of entropy collectors are revealed. However, some of these weaknesses only occur under special conditions (e.g., unavailable `/dev/{u}random` device). We clearly point out that we are not going to discuss the quality of random numbers generated by PRNGs shipped with Operating Systems (OS). Discussions of OS provided (P)RNG facilities can be found at e.g., [5], [6], [7].

2 Related Work

Problems related to weak pseudo random number generators (PRNGs) have been topic of several previously published papers. In 2012 Argyros and Kiayias

¹ <http://harmony.apache.org/>

² <http://www.gnu.org/software/classpath/>

³ <http://openjdk.java.net/>

⁴ <http://www.bouncycastle.org/>

investigated in [8] the state of PRNGs in PHP⁵ and outlined flaws leading to attack vectors. Their results are based on insecure constructions of PRNGs introduced by custom algorithms.

Kopf outlined in [9] multiple cryptographic and implementational flaws in widespread Content Management Systems (CMS). These observations focused weak cryptographic constructs and peculiarities of PHP. The resulting bugs are not caused by weak PRNGs, but by vulnerable custom algorithms in combination with implementational flaws.

Meyer and Somorovsky [10] uncovered a vulnerable use of `SecureRandom` in WSS4J⁶. A function responsible for nonce generation, used at various places in the framework, suffered from weak PRNG seeding.

Problems related to PRNGs are also topic of multiple CWEs (Common Weakness Enumeration)⁷ that deal with the misuse or use of weak pseudo random number generators (cf. CWE 330-343).

In [11] Lenstra et al. inspected millions of public keys of different types (RSA, DSA, ElGamal and ECDSA) and found keys violating basic principles for secure cryptographic parameters. According to the authors these weak keys could be a result of poorly seeded PRNGs.

More alarming results concerning key quality are presented by Heninger et al. in [7] pointing out that *"Randomness is essential for modern cryptography"*. Missing entropy was identified as a root cause for many weak and vulnerable keys. Additionally, the authors identified weak entropy problems under certain conditions in the Linux RNG.

A more theory based cryptanalytic set of attacks on PRNGs can be found at e.g., [12].

2.1 Contribution

The results of this paper focus on PRNG implementations of Java core libraries. Even thus all libraries implement the same functionality - generating (pseudo) *random* numbers - based on the same API, the algorithms for random number generation differ from library to library.

The main contribution is an analysis of algorithms implemented in most commonly used `SecureRandom` generators. For this analysis tests provided by the Dieharder [13] and STS [14] test suites are used to check for cryptographic weaknesses. Additionally manual code analysis uncovered algorithmical and implementational vulnerabilities.

3 Implementations and Algorithms

In Java, random numbers are derived from an initial seed. Two instances of a PRNG seeded with equal values always generate equal *random* sequences. The

⁵ <http://www.php.net>

⁶ <http://ws.apache.org/wss4j/>

⁷ <http://cwe.mitre.org>