Software Assurance Tips

A product of the Software Assurance Tips Team[4]

Generated Monday 17th May, 2021

James Shelton

Monday 17th May, 2021

1 Secure Pseudo-Random Number Generation

1.1 Definition

As the name implies, pseudo-random number generation (PRNG) does not produce a truly random, indeterminate value. The exact nature of this pseudo-randomness varies between languages and implementations, but essentially, PRNG is accomplished via an algorithm that uses mathematical formulae or precalculated tables to produce numerical values that appear random.[3]

However, a common characteristic of all methods of PRNG is that they are deterministic, meaning that they require an initial seed in the form of another numerical value, and that, if the seed is known and replicated exactly, the PRNG will produce the exact same results multiple times. These seeds can take the form of literal values or variable representation of other, unrelated values, such as system time.

Another property of PRNG is that the values it generates are periodic, meaning that, due to the finite amount of pseudo-random values a given implementation can generate, the output will eventually begin to repeat.

1.2 Examples

Most common programming languages have built-in PRNG implementations. The method `rand()`, defined in the C Standard General Utilities Library, is perhaps the most basic PRNG function in the C family, using a seed, provided by the developer as an argument to an earlier call to the method `srand()`, to return an integer value between zero and the language-defined constant `RAND_MAX`.[8] C++11 expands upon these PRNG capabilities in its `<random>` header, which introduces numerous PRNG implementations, from uniform distributors to algorithmic implementations of various mathematical distributions.[6]

Likewise, Java implements a *random* series of methods, found in the `Random` class of the `java.util.Random` object, which function similarly to the C `rand()` function but can be specified to accept as a seeds and output values of other numeric data types.[7]

1.3 Issues

Because of the deterministic and periodic factors, the use of PRNG in a security context is typically discouraged, as the requirement of a defined seed and the finite amount of possible outcomes means that, even if an attacker is not aware of the initial condition, a supposedly random value can be realistically guessed by the use of brute force alone. Still, PRNG has its place in such contexts, as it is generally considered significantly more efficient and practical to implement than true random number generation (TRNG), as TRNG extracts randomness from captured aspects of physical phenomena, such as radioactive decay or atmospheric noise, which most commercially available computer hardware is simply not equipped to do.

An example of a use of PRNG in an identification and authentication scenario is the creation of a temporary session ID for a user. The viability of this implementation is entirely dependent on the source of the seed value used in the PRNG algorithm. One hypothetical source is a given user's authentication ID, but this value is the same each time the user logs in. Another option is that of the system time in Unix Time format, which, on one hand, is guaranteed to change each second. On the other hand, an attacker could replicate a Unix Time string corresponding with a date and time at which any given user is likely to be authenticating (say, a weekday at around 9:00AM), and, with enough luck, could potentially spoof the session ID of an active, verified user simply by correctly guessing what time they logged on, a tactic that only increases in viability the more users a system supports.

From a software assurance perspective, in the event of scenario similar in scope to the previous example, reported issues will usually adhere to a vulnerability description enumerated as CWE-337 or CWE-338,[1, 2] the former with respect to the use of the seeded value and the latter with respect to the PRNG function itself. Occasionally, both will occur concurrently for the same instance.

1.4 Mitigations

In general, the larger and more arbitrary the seed, the less likely the likely the PRNG implementation can be exploited by an attacker. Or, at least, it would result in exploitation attempts taking longer to accomplish, thus increasing the chance of the attack being noticed before the attacker succeeds. Regardless, the very presence of a PRNG implementation is overwhelmingly likely to be documented as an issue by a scan tool, but the security of such of an implementation can still be ensured through a multilayered approach tailored to fit the specific context in which the pseudo-random value is being used.

In an authentication context, imposing a limit to the number of times a user can unsuccessfully supply valid credentials and/or placing a restriction on subsequent attempts can mitigate attacks based around exploiting PRNG to replicate a valid set of credentials or the privileges attached to a valid set of credentials.

In a cryptographic context, the use of algorithms whose PRNG has been sufficiently tested and independently verified is the best way to mitigate potential issues. A list of random number generators approved for use in cryptographic modules can be found in Annex C of the FIPS 140-2 publication from the Information Technology Laboratory of the National Institute of Standards and Technology.[5]

References

[1] CWE Content Team. “CWE-337: Predictable Seed in Pseudo-Random Number Generator (PRNG)”. In: (2020). URL: https://cwe.mitre.org/data/definitions/337.html.

[2] CWE Content Team. “CWE-338: Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)”. In: (2021). URL: https://cwe.mitre.org/data/definitions/338.html.

[3] Mads Haahr. “Introduction to Randomness and Random Numbers”. In: *RANDOM.ORG* (2021). URL: https://www.random.org/randomness/.

[4] Jon Hood, ed. *SwATips*. https://www.SwATips.com/.

[5] National Institute of Standards and Technology. *Annex C: Approved Random Number Generators for FIPS PUB 140-2. Security Requirements for Cryptographic Modules*. Department of Commerce. 2019. URL: https://csrc.nist.gov/CSRC/media/Publications/fips/140/2/final/documents/fips1402annexc.pdf.

[6] Various. “<random>”. In: *cplusplus.com* (2021). URL: https://www.cplusplus.com/reference/random/.

[7] Various. “Class Random”. In: *Java 8 Documentation* (2021). URL: https://docs.oracle.com/javase/8/docs/api/java/util/Random.html.

[8] Various. “rand”. In: *cplusplus.com* (2021). URL: https://www.cplusplus.com/reference/cstdlib/rand/.