# CWE Detail – CWE-335

## Description

The product uses a Pseudo-Random Number Generator (PRNG) but does not correctly manage seeds.

## Extended Description

PRNGs are deterministic and, while their output appears  
 random, they cannot actually create entropy. They rely on  
 cryptographically secure and unique seeds for entropy so  
 proper seeding is critical to the secure operation of the  
 PRNG. Management of seeds could be broken down into two main areas: (1) protecting seeds as cryptographic material (such as a cryptographic key); (2) whenever possible, using a uniquely generated seed from  
 a cryptographically secure source PRNGs require a seed as input to generate a stream of  
 numbers that are functionally indistinguishable from  
 random numbers. While the output is, in many cases,  
 sufficient for cryptographic uses, the output of any  
 PRNG is directly determined by the seed provided as  
 input. If the seed can be ascertained by a third party,  
 the entire output of the PRNG can be made known to  
 them. As such, the seed should be kept secret and  
 should ideally not be able to be guessed. For example,  
 the current time may be a poor seed. Knowing the  
 approximate time the PRNG was seeded greatly reduces  
 the possible key space. Seeds do not necessarily need to be unique, but reusing seeds may open up attacks if the seed is discovered.

## Threat-Mapped Scoring

Score: 1.8

Priority: P4 - Informational (Low)

## Observed Examples (CVEs)

**•** CVE-2020-7010: Cloud application on Kubernetes generates passwords using a weak random number generator based on deployment time.

**•** CVE-2019-11495: server uses erlang:now() to seed the PRNG, which  
 results in a small search space for potential random  
 seeds

**•** CVE-2018-12520: Product's PRNG is not seeded for the generation of session IDs

**•** CVE-2016-10180: Router's PIN generation is based on rand(time(0)) seeding.

## Modes of Introduction

**•** Implementation: REALIZATION: This weakness is caused during implementation of an architectural security tactic.

## Common Consequences

**•** Impact: Bypass Protection Mechanism, Other — Notes: If a PRNG is used incorrectly, such as using the same seed for each initialization or using a predictable seed, then an attacker may be able to easily guess the seed and thus the random numbers. This could lead to unauthorized access to a system if the seed is used for authentication and authorization.

## Applicable Platforms

**•** None (Class: Not Language-Specific, Prevalence: Undetermined)

## Demonstrative Examples

**•** Because the program uses the same seed value for every invocation of the PRNG, its values are predictable, making the system vulnerable to attack.

**•** An attacker can easily predict the seed used by these PRNGs, and so also predict the stream of random numbers generated. Note these examples also exhibit CWE-338 (Use of Cryptographically Weak PRNG).

**•** Since only 2 bytes are used as a seed, an attacker will only need to guess 2^16 (65,536) values before being able to replicate the state of the PRNG.

## Notes

**•** Maintenance: As of CWE 4.5, terminology related to randomness, entropy, and  
 predictability can vary widely. Within the developer and other  
 communities, "randomness" is used heavily. However, within  
 cryptography, "entropy" is distinct, typically implied as a  
 measurement. There are no commonly-used definitions, even within  
 standards documents and cryptography papers. Future versions of  
 CWE will attempt to define these terms and, if necessary,  
 distinguish between them in ways that are appropriate for  
 different communities but do not reduce the usability of CWE for  
 mapping, understanding, or other scenarios.