C++ static code analysis: Facilities in should be used instead of "srand", "rand" and "random_shuffle"

2-3 minutes

The use of `srand` together with `rand` to seed the random number generator and then generate numbers usually produces low-quality randomness. Further, `rand` can only provide a number between `0` and `RAND_MAX`, and it is left to the caller to transform the result into what is actually required (E.G. a `float` between 0 and 1 for a random percentage, an `int` between 1 and 6 for a dice game, ...), and that transformation might introduce additional biases.

C++11 introduced the `<random>` library, which contains several high quality random value generators as well as statistical distributions you

can use to put the results in the form you need. Those mechanisms should be used instead of `rand` and `srand`.

Additionally, `std::random_shuffle`, which is deprecated in C++14 and removed in C++17, uses `rand` and should be replaced by `std::shuffle`, which uses the random number generators provided by `<random>`.

Noncompliant Code Example

```
#include <stdlib.h>
#include <algorithm>
// ...

void f() {
  srand(time(nullptr)); // Noncompliant
  vector<int> v;
  int size = rand() % 1000 + 1000; // Noncompliant, note that this way of coercing the result introduces extra bias
  for (auto i = 0; i < size; ++i) {
    v.push_back(i);
  }
```

```
  random_shuffle(v.begin(), v.end()); //
Noncompliant
  for (auto i : v) { cout << i << " "; }
}
```

Compliant Solution

```
#include <algorithm>
#include <random>
// ...

void f() {
  random_device rd;  // Will be used to obtain a
seed for the random number engine
  mt19937 gen(rd()); // Standard
mersenne_twister_engine seeded with rd()
  uniform_int_distribution<> dis(1000, 1999); //
Same distribution as before, but explicit and
without bias
  vector<int> v;
  for (auto i = 0; i < dis(gen); ++i) {
    v.push_back(i);
  }
  shuffle(v.begin(), v.end(), gen);
  for (auto i : v) { cout << i << " "; }
```

```
}
```