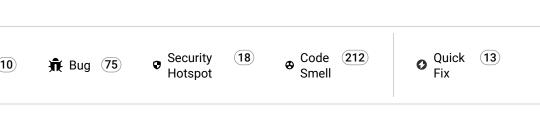
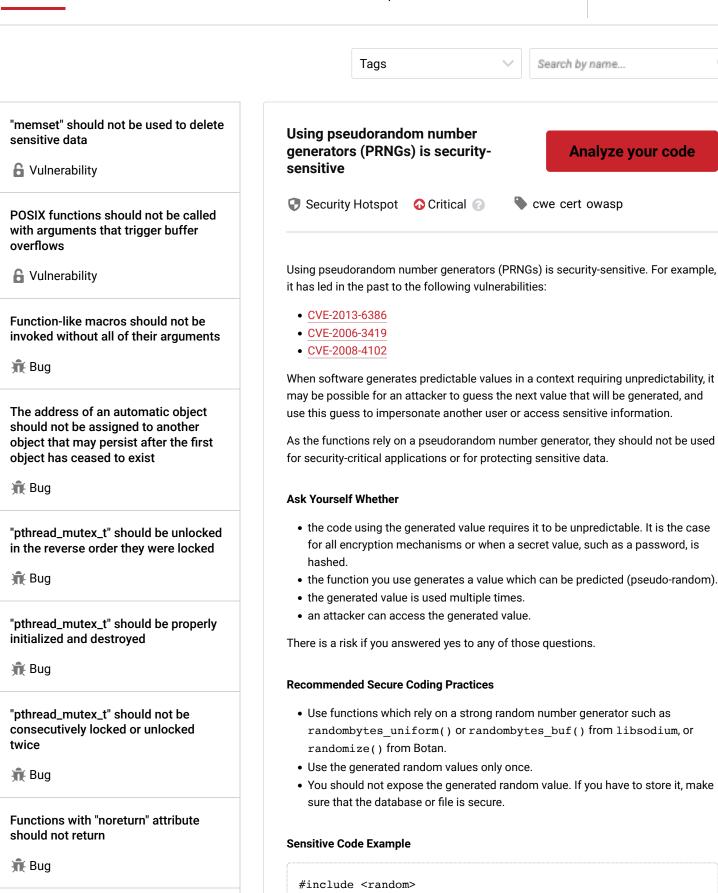






Unique rules to find Bugs, Vulnerabilities, Security Hotspots, and Code Smells in your OBJECTIVE C code





```
void f() {
 int random_int = std::rand(); // Sensitive
```

Compliant Solution

Freed memory should not be used

📆 Bug

Memory locations should not be released more than once

📆 Bug

Memory access should be explicitly bounded to prevent buffer overflows

📆 Bug

Printf-style format strings should not lead to unexpected behavior at runtime

📆 Bug

Recursion should not be infinite

👬 Bug

Resources should be closed

👬 Bug

Hard-coded credentials are securitysensitive

Security Hotspot

"goto" should jump to labels declared later in the same function

Code Smell

Only standard forms of the "defined" directive should be used

Code Smell

Switch labels should not be nested inside non-switch blocks

Code Smell

```
#include <sodium.h>
#include <botan/system_rng.h>
// ...

void f() {
   char random_chars[10];
   randombytes_buf(random_chars, 10); // Compliant
   uint32_t random_int = randombytes_uniform(10); // Compliant

uint8_t random_chars[10];
   Botan::System_RNG system;
   system.randomize(random_chars, 10); // Compliant
}
```

See

- OWASP Top 10 2021 Category A2 Cryptographic Failures
- OWASP Top 10 2017 Category A3 Sensitive Data Exposure
- Mobile AppSec Verification Standard Cryptography Requirements
- OWASP Mobile Top 10 2016 Category M5 Insufficient Cryptography
- MITRE, CWE-338 Use of Cryptographically Weak Pseudo-Random Number Generator (PRNG)
- MITRE, CWE-330 Use of Insufficiently Random Values
- MITRE, CWE-326 Inadequate Encryption Strength
- - <u>CERT, MSC50-CPP.</u> Do not use still for generating pseudorandom numbers
 - Derived from FindSecBugs rule Predictable Pseudo Random Number Generator

Available In:

sonarcloud **Sonarcloud** sonarcloud obe