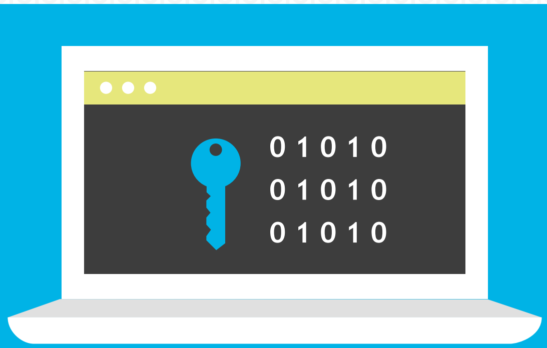
VULNERABILITY DECODER

Insecure Crypto

The Vulnerability

Proper encryption is notorious for its difficult implementation, but it's essential to strong application security practices. Cryptographic flaws include using broken crypto algorithms, improperly validating certificates, storing sensitive information in cleartext, and employing inadequate encryption strength.



63.7% **of applications have cryptographic issues on initial scan.**

Source: SOSS v11

The Risks

Insecure crypto can lead to stolen or destroyed data, including some of your most sensitive information, such as personally identifiable information of customers or employees (e.g., Social Security numbers, or bank or credit card details).

Example Breach

A malicious hacktivist leaked The Panama Papers from Panamanian law firm Mossack Fonseca. The document dump was made possible by an SSL cryptographic flaw, known as the DROWN attack, in the firm's customer-facing website.

Prevention & Remediation

Cryptographic vulnerabilities are preventable with secure coding practices. Most major languages inherently support good cryptographic practices, and concerns over incorrect implementation typically arise only on a case-by-case basis.

Example: In Java 8, the **SecureRandom** class provides CSPRNG functionality. The most OS-agnostic way to generate pseudo-random data that's suitable for general cryptographic use is to rely on the OS implementation's defaults, and never to explicitly seed it (i.e., don't use the setSeed method before a call to next* methods). You'll find more information **here**.

How to generate pseudo-random data for general cryptographic use

```
// returns an unseeded instance of default RNG algorithm based on most preferred provider from list of providers
configured in java.security
// On Unix like system, NativePRNG algorithm, configured with seeding from non-blocking entropy source, is returned.
// On Windows, SHA1PRNG algorithm, which can be self-seeded or explicitly seeded is returned.
        SecureRandom secRan = new SecureRandom();
        byte[] ranBytes = new bytes[20];
        secRan.nextBytes(ranBytes);
// since, there is no setSeed method called before a call to next* method, self-seeding occurs
```

Recommendations

☑ Get training in secure coding best practices, through on-demand eLearning courses, in person security consultations, and professional development certifications and conferences.

☑ Scan early and often to detect flaws while you code. Use application security tools that allow you to scan small batches of code instantaneously, and can provide remediation guidance within your development workflow.

Download the Secure Coding Best Practices Handbook

Join the Veracode Community

community.veracode.com

VERACODE