rules.sonarsource.com

Kotlin static code analysis: Using weak hashing algorithms is security-sensitive

3-4 minutes

The MD5 algorithm and its successor, SHA-1, are no longer considered secure, because it is too easy to create hash collisions with them. That is, it takes too little computational effort to come up with a different input that produces the same MD5 or SHA-1 hash, and using the new, same-hash value gives an attacker the same access as if he had the originally-hashed value. This applies as well to the other Message-Digest algorithms: MD2, MD4, MD6, HAVAL-128, HMAC-MD5, DSA (which uses SHA-1), RIPEMD, RIPEMD-128, RIPEMD-160, HMACRIPEMD160.

The following APIs are tracked for use of obsolete crypto algorithms:

- `java.security.AlgorithmParameters` (JDK)
- `java.security.AlgorithmParameterGenerator` (JDK)
- `java.security.MessageDigest` (JDK)
- `java.security.KeyFactory` (JDK)
- `java.security.KeyPairGenerator` (JDK)
- `java.security.Signature` (JDK)
- `javax.crypto.Mac` (JDK)
- `javax.crypto.KeyGenerator` (JDK)
- `org.apache.commons.codec.digest.DigestUtils` (Apache Commons Codec)
- `org.springframework.util.DigestUtils`
- `com.google.common.hash.Hashing` (Guava)
- `org.springframework.security.authentication.encoding.ShaPasswordEncoder` (Spring Security 4.2.x)
- `org.springframework.security.authentication.encoding.Md5PasswordEncoder` (Spring Security 4.2.x)
- `org.springframework.security.crypto.password.LdapShaPasswordEncoder` (Spring Security 5.0.x)
- `org.springframework.security.crypto.password.Md4PasswordEncoder` (Spring Security 5.0.x)
- `org.springframework.security.crypto.password.MessageDigestPasswordEncoder` (Spring Security 5.0.x)
- `org.springframework.security.crypto.password.NoOpPasswordEncoder` (Spring Security 5.0.x)
- `org.springframework.security.crypto.password.StandardPasswordEncoder` (Spring Security 5.0.x)

Ask Yourself Whether

The hashed value is used in a security context like:

- User-password storage.
- Security token generation (used to confirm e-mail when registering on a website, reset password, etc …).
- To compute some message integrity.

There is a risk if you answered yes to any of those questions.

Recommended Secure Coding Practices

Safer alternatives, such as `SHA-256`, `SHA-512`, `SHA-3` are recommended, and for password hashing, it's even better to use algorithms that do not compute too "quickly", like `bcrypt`, `scrypt`, `argon2` or `pbkdf2` because it slows down `brute force`

attacks.

Sensitive Code Example

```
val md1: MessageDigest = MessageDigest.getInstance("SHA");  // Sensitive:  SHA is not a standard name, for most security providers it's an alias of SHA-1
val md2: MessageDigest = MessageDigest.getInstance("SHA1");  // Sensitive
```

Compliant Solution

```
val md1: MessageDigest = MessageDigest.getInstance("SHA-512"); // Compliant
```

See

- OWASP Top 10 2021 Category A2 - Cryptographic Failures
- OWASP Top 10 2017 Category A3 - Sensitive Data Exposure
- OWASP Top 10 2017 Category A6 - Security Misconfiguration
- Mobile AppSec Verification Standard - Cryptography Requirements
- OWASP Mobile Top 10 2016 Category M5 - Insufficient Cryptography
- MITRE, CWE-1240 - Use of a Risky Cryptographic Primitive
- SANS Top 25 - Porous Defenses