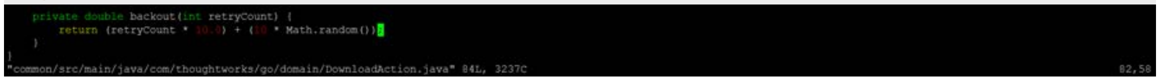
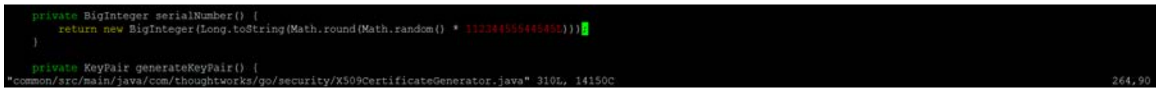
com/thoughtworks/go/domain/DownloadAction.java	82	2	SECURITY	PREDICTABLE_RANDOM	ℹ	💡
com/thoughtworks/go/security/X509CertificateGenerator.java	264	2	SECURITY	PREDICTABLE_RANDOM	ℹ	💡
com/thoughtworks/go/util/SystemUtil.java	55	2	SECURITY	PREDICTABLE_RANDOM	ℹ	💡

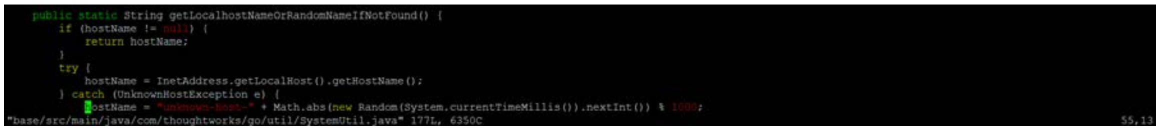


```
    private double backout(int retryCount) {
        return (retryCount * 10.0) + (10 * Math.random());
    }
}
"common/src/main/java/com/thoughtworks/go/domain/DownloadAction.java" 84L, 3237C                                   82,58
```



```
    private BigInteger serialNumber() {
        return new BigInteger(Long.toString(Math.round(Math.random() * 11234455544545L)));
    }

    private KeyPair generateKeyPair() {
"common/src/main/java/com/thoughtworks/go/security/X509CertificateGenerator.java" 310L, 14150C                       264,90
```



```
    public static String getLocalhostNameOrRandomNameIfNotFound() {
        if (hostName != null) {
            return hostName;
        }
        try {
            hostName = InetAddress.getLocalHost().getHostName();
        } catch (UnknownHostException e) {
            hostName = "unknown-host-" + Math.abs(new Random(System.currentTimeMillis()).nextInt()) % 1000;
"base/src/main/java/com/thoughtworks/go/util/SystemUtil.java" 177L, 6350C                                            55,13
```

The use of java.lang.Math.random() is predictable

The use of java.util.Random is predictable

The use of a predictable random value can lead to vulnerabilities when used in certain security critical contexts.

For example, when the value is used as:

a CSRF token: a predictable token can lead to a CSRF attack as an attacker will know the value of the token

a password reset token (sent by email): a predictable password token can lead to an account takeover, since an attacker will guess the URL of the "change password" form

any other secret value

A quick fix could be to replace the use of java.util.Random with something stronger, such as java.security.SecureRandom.

Vulnerable Code:

String generateSecretToken() { Random r = new Random(); return Long.toHexString(r.nextLong()); }

Solution:

import org.apache.commons.codec.binary.Hex; String generateSecretToken() { SecureRandom secRandom = new SecureRandom(); byte[] result = new byte[32]; secRandom.nextBytes(result); return Hex.encodeHexString(result); }

References

Cracking Random Number Generators - Part 1 (http://jazzy.id.au)

CERT: MSC02-J. Generate strong random numbers

CWE-330: Use of Insufficiently Random Values

Predicting Struts CSRF Token (Example of real-life vulnerability and exploitation)