**Use and Misuse Cases**

The **Use Cases** need to be from a perspective of us (as a software writing business) pitching our service to a potential customer:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **User Interaction** | **Enabling Step** | **Protection Mech** | **Attacker Vector** |
| 1 | Create customer account | Save payment information | Require CCV number | Purchase using CC info |
|  |  | Save payment information | Encrypt CC number | Retrieve CC info |
|  |  | Save address information | Protect DB with Auth mech | Use account  Priv-escalation |
|  |  |  |  |  |
| 2 | Create seller account | Access 3rd Party Inventory DB | Protect DB with Auth mech | Use account  Priv-escalation |
|  |  | Access Payment API | Encrypt payment info | Retrieve CC info |
|  |  | Access Shipping API | Protect with Auth mech | Use account  Priv-escalation |
|  |  | Access Customer Address DB | Protect DB with Auth mech | Use account  Priv-escalation |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |
|  |  |  |  |  |

The app uses **BCryptPasswordEncoder.java** to encrypt the user’s password prior to transmission to the <\_\_\_> (name of next step in the process flow).

**Update the default strength for password encoders #7411**

(<https://github.com/spring-projects/spring-security/issues/7411>)

|  |
| --- |
| **I propose to increase the default strength for some password encoders.**  **The BCrypt encoder currently uses 10 as default strength, with which it only takes ~80ms to encode a password (on my Early 2015 MacBook Pro). I think the default should be increased to (at least) 12.**  **The Argon2 encoder only needs ~40ms, so its default should be increased too.**  **The SCrypt encoder needs ~80ms with its default parameters.**  [**larsgrefer**](https://github.com/larsgrefer) **commented** [**on Sep 9, 2019**](https://github.com/spring-projects/spring-security/issues/7411#issue-491168547) |

**BCryptPasswordEncoder.java**

(<https://github.com/spring-projects/spring-security/blob/2abf59b695b3ad14719299ed17ff47b181eed802/crypto/src/main/java/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoder.java>)

/\*

\* Copyright 2002-2011 the original author or authors.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* <https://www.apache.org/licenses/LICENSE-2.0>

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package org.springframework.security.crypto.bcrypt;

import java.security.SecureRandom;

import java.util.regex.Matcher;

import java.util.regex.Pattern;

import org.apache.commons.logging.Log;

import org.apache.commons.logging.LogFactory;

import org.springframework.security.crypto.password.PasswordEncoder;

/\*

\* Implementation of PasswordEncoder that uses the BCrypt strong hashing

\* function. Clients can optionally supply a "version" ($2a, $2b, $2y) and a

\* "strength" (a.k.a. log rounds in BCrypt) and a SecureRandom instance. The

\* larger the strength parameter the more work will have to be done

\* (exponentially) to hash the passwords. The default value is 10.

\*

\* @author Dave Syer

\*/

public class BCryptPasswordEncoder implements PasswordEncoder {

private Pattern BCRYPT\_PATTERN = Pattern.compile("\\A\\$2(a|y|b)?\\$(\\d\\d)\\$[./0-9A-Za-z]{53}");

private final Log logger = LogFactory.getLog(getClass());

private final int strength;

private final BCryptVersion version;

private final SecureRandom random;

public BCryptPasswordEncoder() {

this(-1);

}

/\*\*

\* @param strength the log rounds to use, between 4 and 31

\*/

public BCryptPasswordEncoder(int strength) {

this(strength, null);

}

/\*\*

\* @param version the version of bcrypt, can be 2a,2b,2y

\*/

public BCryptPasswordEncoder(BCryptVersion version) {

this(version, null);

}

/\*\*

\* @param version the version of bcrypt, can be 2a,2b,2y

\* @param random the secure random instance to use

\*/

public BCryptPasswordEncoder(BCryptVersion version, SecureRandom random) {

this(version, -1, random);

}

/\*\*

\* @param strength the log rounds to use, between 4 and 31

\* @param random the secure random instance to use

\*/

public BCryptPasswordEncoder(int strength, SecureRandom random) {

this(BCryptVersion.$2A, strength, random);

}

/\*\*

\* @param version the version of bcrypt, can be 2a,2b,2y

\* @param strength the log rounds to use, between 4 and 31

\*/

public BCryptPasswordEncoder(BCryptVersion version, int strength) {

this(version, strength, null);

}

/\*\*

\* @param version the version of bcrypt, can be 2a,2b,2y

\* @param strength the log rounds to use, between 4 and 31

\* @param random the secure random instance to use

\*/

public BCryptPasswordEncoder(BCryptVersion version, int strength, SecureRandom random) {

if (strength != -1 && (strength < BCrypt.MIN\_LOG\_ROUNDS || strength > BCrypt.MAX\_LOG\_ROUNDS)) {

throw new IllegalArgumentException("Bad strength");

}

this.version = version;

this.strength = (strength == -1) ? 10 : strength;

this.random = random;

}

@Override

public String encode(CharSequence rawPassword) {

if (rawPassword == null) {

throw new IllegalArgumentException("rawPassword cannot be null");

}

String salt = getSalt();

return BCrypt.hashpw(rawPassword.toString(), salt);

}

private String getSalt() {

if (this.random != null) {

return BCrypt.gensalt(this.version.getVersion(), this.strength, this.random);

}

return BCrypt.gensalt(this.version.getVersion(), this.strength);

}

@Override

public boolean matches(CharSequence rawPassword, String encodedPassword) {

if (rawPassword == null) {

throw new IllegalArgumentException("rawPassword cannot be null");

}

if (encodedPassword == null || encodedPassword.length() == 0) {

this.logger.warn("Empty encoded password");

return false;

}

if (!this.BCRYPT\_PATTERN.matcher(encodedPassword).matches()) {

this.logger.warn("Encoded password does not look like BCrypt");

return false;

}

return BCrypt.checkpw(rawPassword.toString(), encodedPassword);

}

@Override

public boolean upgradeEncoding(String encodedPassword) {

if (encodedPassword == null || encodedPassword.length() == 0) {

this.logger.warn("Empty encoded password");

return false;

}

Matcher matcher = this.BCRYPT\_PATTERN.matcher(encodedPassword);

if (!matcher.matches()) {

throw new IllegalArgumentException("Encoded password does not look like BCrypt: " + encodedPassword);

}

int strength = Integer.parseInt(matcher.group(2));

return strength < this.strength;

}

/\*\*

\* Stores the default bcrypt version for use in configuration.

\*

\* @author Lin Feng

\*/

public enum BCryptVersion {

$2A("$2a"),

$2Y("$2y"),

$2B("$2b");

private final String version;

BCryptVersion(String version) {

this.version = version;

}

public String getVersion() {

return this.version;

}

}

}

**BCryptPasswordEncoderTests.java**

(<https://github.com/spring-projects/spring-security/blob/c5023127190c8f80bf83bafabcbea4df5cf44898/crypto/src/test/java/org/springframework/security/crypto/bcrypt/BCryptPasswordEncoderTests.java>)

/\*

\* Copyright 2002-2011 the original author or authors.

\*

\* Licensed under the Apache License, Version 2.0 (the "License");

\* you may not use this file except in compliance with the License.

\* You may obtain a copy of the License at

\*

\* <https://www.apache.org/licenses/LICENSE-2.0>

\*

\* Unless required by applicable law or agreed to in writing, software

\* distributed under the License is distributed on an "AS IS" BASIS,

\* WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.

\* See the License for the specific language governing permissions and

\* limitations under the License.

\*/

package org.springframework.security.crypto.bcrypt;

import java.security.SecureRandom;

import org.junit.Test;

import static org.assertj.core.api.Assertions.assertThat;

import static org.assertj.core.api.Assertions.assertThatIllegalArgumentException;

/\*\*

\* @author Dave Syer

\*

\*/

public class BCryptPasswordEncoderTests {

@Test

// gh-5548

public void emptyRawPasswordDoesNotMatchPassword() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

String result = encoder.encode("password");

assertThat(encoder.matches("", result)).isFalse();

}

@Test

public void $2yMatches() {

// $2y is default version

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

String result = encoder.encode("password");

assertThat(result.equals("password")).isFalse();

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void $2aMatches() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2A);

String result = encoder.encode("password");

assertThat(result.equals("password")).isFalse();

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void $2bMatches() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2B);

String result = encoder.encode("password");

assertThat(result.equals("password")).isFalse();

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void $2yUnicode() {

// $2y is default version

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

String result = encoder.encode("passw\u9292rd");

assertThat(encoder.matches("pass\u9292\u9292rd", result)).isFalse();

assertThat(encoder.matches("passw\u9292rd", result)).isTrue();

}

@Test

public void $2aUnicode() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2A);

String result = encoder.encode("passw\u9292rd");

assertThat(encoder.matches("pass\u9292\u9292rd", result)).isFalse();

assertThat(encoder.matches("passw\u9292rd", result)).isTrue();

}

@Test

public void $2bUnicode() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2B);

String result = encoder.encode("passw\u9292rd");

assertThat(encoder.matches("pass\u9292\u9292rd", result)).isFalse();

assertThat(encoder.matches("passw\u9292rd", result)).isTrue();

}

@Test

public void $2yNotMatches() {

// $2y is default version

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

String result = encoder.encode("password");

assertThat(encoder.matches("bogus", result)).isFalse();

}

@Test

public void $2aNotMatches() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2A);

String result = encoder.encode("password");

assertThat(encoder.matches("bogus", result)).isFalse();

}

@Test

public void $2bNotMatches() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2B);

String result = encoder.encode("password");

assertThat(encoder.matches("bogus", result)).isFalse();

}

@Test

public void $2yCustomStrength() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(8);

String result = encoder.encode("password");

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void $2aCustomStrength() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2A, 8);

String result = encoder.encode("password");

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void $2bCustomStrength() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(BCryptPasswordEncoder.BCryptVersion.$2B, 8);

String result = encoder.encode("password");

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void badLowCustomStrength() {

assertThatIllegalArgumentException().isThrownBy(() -> new BCryptPasswordEncoder(3));

}

@Test

public void badHighCustomStrength() {

assertThatIllegalArgumentException().isThrownBy(() -> new BCryptPasswordEncoder(32));

}

@Test

public void customRandom() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder(8, new SecureRandom());

String result = encoder.encode("password");

assertThat(encoder.matches("password", result)).isTrue();

}

@Test

public void doesntMatchNullEncodedValue() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThat(encoder.matches("password", null)).isFalse();

}

@Test

public void doesntMatchEmptyEncodedValue() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThat(encoder.matches("password", "")).isFalse();

}

@Test

public void doesntMatchBogusEncodedValue() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThat(encoder.matches("password", "012345678901234567890123456789")).isFalse();

}

@Test

public void upgradeFromLowerStrength() {

BCryptPasswordEncoder weakEncoder = new BCryptPasswordEncoder(5);

BCryptPasswordEncoder strongEncoder = new BCryptPasswordEncoder(15);

String weakPassword = weakEncoder.encode("password");

String strongPassword = strongEncoder.encode("password");

assertThat(weakEncoder.upgradeEncoding(strongPassword)).isFalse();

assertThat(strongEncoder.upgradeEncoding(weakPassword)).isTrue();

}

/\*\*

\* @see <a href=

\* "https://github.com/spring-projects/spring-security/pull/7042#issuecomment-506755496">https://github.com/spring-projects/spring-security/pull/7042#issuecomment-506755496</a>

\*/

@Test

public void upgradeFromNullOrEmpty() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThat(encoder.upgradeEncoding(null)).isFalse();

assertThat(encoder.upgradeEncoding("")).isFalse();

}

/\*\*

\* @see <a href=

\* "https://github.com/spring-projects/spring-security/pull/7042#issuecomment-506755496">https://github.com/spring-projects/spring-security/pull/7042#issuecomment-506755496</a>

\*/

@Test

public void upgradeFromNonBCrypt() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThatIllegalArgumentException().isThrownBy(() -> encoder.upgradeEncoding("not-a-bcrypt-password"));

}

@Test

public void encodeNullRawPassword() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThatIllegalArgumentException().isThrownBy(() -> encoder.encode(null));

}

@Test

public void matchNullRawPassword() {

BCryptPasswordEncoder encoder = new BCryptPasswordEncoder();

assertThatIllegalArgumentException().isThrownBy(() -> encoder.matches(null, "does-not-matter"));

}

}