

wildfly-security / wildfly-elytron Public

<> **Code** Pull requests 34 Actions Projects Wiki Security and quality 1

Commit d7754f5



rsearls committed on Mar 14, 2025 · 9 / 9

[ELY-2887] Add a nonce to OIDC requests for [CVE-2024-12369](#)


[2.x](#) (#2261) · 3.0.0.Alpha1 ... 2.2.9.Final

1 parent [a705c00](#) commit d7754f5

13 files changed +175 -66 lines changed

Top

- main/java/org/wildfly/security/http/oidc
 - AuthenticationError.java
 - ElytronMessages.java
 - IDToken.java
 - Oidc.java
 - OidcCookieTokenStore.java
 - OidcRequestAuthenticator.java
 - RefreshableOidcSecurityContext.java
 - RequestAuthenticator.java
 - TokenValidator.java
 - test/java/org/wildfly/security/http/oidc
 - OidcBaseTest.java
 - OidcSecurityRealmTest.java
 - OidcTest.java
 - tests/base/src/test/java/org/wildfly/security/http/impl

 AbstractBaseHttpTest.java

 **13 files changed** +175 -66 lines changed


 ▾ ...security/http/oidc/AuthenticationError.java ⋮

```

↑⋮      @@ -36,7 +36,8 @@ public enum Reason {
36 36          INVALID_TOKEN,
37 37          STALE_TOKEN,
38 38          NO_AUTHORIZATION_HEADER,
39 -      NO_QUERY_PARAMETER_ACCESS_TOKEN
39 +      NO_QUERY_PARAMETER_ACCESS_TOKEN,
40 +      INVALID_NONCE
40 41      }
41 42
42 43      private Reason reason;
  
```

 ▾ ...fly/security/http/oidc/ElytronMessages.java ⋮

```

↑⋮      @@ -238,5 +238,13 @@ interface ElytronMessages extends BasicLogger {
238 238          @Message(id = 23057, value = "principal-attribute '%s' claim does not
          exist, falling back to 'sub'")
239 239          void principalAttributeClaimDoesNotExist(String principalAttributeClaim);
240 240
241 +      @Message(id = 23071, value = "Invalid ID token nonce: %s")
242 +      String invalidNonceValue(String name);
243 +
244 +      @Message(id = 23072, value = "No such algorithm: '%s'")
245 +      IllegalArgumentException noSuchAlgorithm(String algorithm);
246 +
247 +      @Message(id = 23073, value = "Nonce cookie does not exist")
248 +      String nonceCookieDoesNotExist();
241 249      }
242 250
  
```

 ▾ ...org/wildfly/security/http/oidc/IDToken.java ⋮

```

↑⋮      @@ -51,6 +51,7 @@ public class IDToken extends JsonWebToken {
51 51          public static final String CLAIMS_LOCALES = "claims_locales";
52 52          public static final String ACR = "acr";
  
```

```

53 53      public static final String S_HASH = "s_hash";
54 54  +    public static final String NONCE = "nonce";
54 55
55 56      /**
56 57      * Construct a new instance.
57 57
58 58  @@ -220,4 +221,7 @@ public String getAcr() {
59 59
220 221      return getClaimValueAsString(ACR);
221 222    }
222 223
224 224  +    public String getNonce() {
225 225  +        return getClaimValueAsString(NONCE);
226 226  +    }
227 227    }

```

```

...va/org/wildfly/security/http/oidc/Oidc.java
19 19  package org.wildfly.security.http.oidc;
20 20
21 21  import static org.wildfly.security.http.oidc.ElytronMessages.log;
22 22  + import static org.wildfly.security.jose.jwk.JWKUtil.BASE64_URL;
23 23
24 24  import java.io.IOException;
25 25  import java.io.InputStream;
26 26  import java.net.InetAddress;
27 27  import java.net.UnknownHostException;
28 28  + import java.nio.charset.StandardCharsets;
29 29  + import java.security.MessageDigest;
30 30  + import java.security.NoSuchAlgorithmException;
31 31  import java.util.Map;
32 32  import java.util.StringTokenizer;
33 33  import java.util.UUID;
34 34  @@ -34,6 +38,7 @@
35 35  import org.apache.http.HttpResponse;
36 36  import org.apache.http.client.methods.HttpRequestBase;
37 37  import org.jose4j.jws.AlgorithmIdentifiers;
38 38  + import org.wildfly.common.iteration.ByteIterator;
39 39  import org.wildfly.security.jose.util.JsonSerialization;
40 40
41 41  /**

```

```

@@ -81,13 +86,15 @@ public class Oidc {
81 86     public static final String REDIRECT_URI = "redirect_uri";
82 87     public static final String REFRESH_TOKEN = "refresh_token";
83 88     public static final String RESPONSE_TYPE = "response_type";
89 +     public static final String SESSION_RANDOM_VALUE="session_random_value";
84 90     public static final String SESSION_STATE = "session_state";
85 91     public static final String SOAP_ACTION = "SOAPAction";
86 92     public static final String STALE_TOKEN = "Stale token";
87 93     public static final String STATE = "state";
88 94     public static final int INVALID_ISSUED_FOR_CLAIM = -1;
89 95     public static final int INVALID_AT_HASH_CLAIM = -2;
90 96     public static final int INVALID_TYPE_CLAIM = -3;
97 +     public static final int INVALID_SESSION_RANDOM_VALUE = -4;
91 98     static final String OIDC_CLIENT_CONFIG_RESOLVER = "oidc.config.resolver";
92 99     static final String OIDC_CONFIG_FILE_LOCATION = "oidc.config.file";
93 100    static final String OIDC_JSON_FILE = "/WEB-INF/oidc.json";
@@ -368,4 +375,14 @@ protected static boolean
checkCachedAccountMatchesRequest(OidcAccount account, 0
368 375         return true;
369 376     }
370 377
378 +     protected static String getCryptographicValue(final String src) {
379 +         try {
380 +             MessageDigest md = MessageDigest.getInstance(SHA256);
381 +             md.update(src.getBytes(StandardCharsets.UTF_8));
382 +             return ByteIterator.ofBytes(md.digest())
383 +                 .base64Encode(BASE64_URL, false).drainToString();
384 +         } catch (NoSuchAlgorithmException e) {
385 +             throw log.noSuchAlgorithm(e.getMessage());
386 +         }
387 +     }
371 388 }

```

```

...ecurity/http/oidc/OidcCookieTokenStore.java
@@ -21,6 +21,7 @@
21 21     import static org.wildfly.security.http.oidc.ElytronMessages.log;
22 22     import static org.wildfly.security.http.oidc.Oidc.OIDC_STATE_COOKIE;
23 23     import static
        org.wildfly.security.http.oidc.Oidc.checkCachedAccountMatchesRequest;

```

```

24 + import static org.wildfly.security.http.oidc.Oidc.SESSION_RANDOM_VALUE;
24 25
25 26 import java.net.URISyntaxException;
26 27 import java.util.List;
  ↓
  ↑
@@ -227,7 +228,7 @@ public static
OidcPrincipal<RefreshableOidcSecurityContext> getPrincipalFromCook
227 228         idToken = new IDToken(new
        JwtConsumerBuilder().setSkipSignatureVerification().setSkipAllValidators().build().processToClaims(idTokenString));
228 229     }
229 230     log.debug("Token obtained from cookie");
230 -         RefreshableOidcSecurityContext secContext = new
        RefreshableOidcSecurityContext(deployment, tokenStore, accessTokenString,
        accessToken, idTokenString, idToken, refreshTokenString);
231 +         RefreshableOidcSecurityContext secContext = new
        RefreshableOidcSecurityContext(deployment,
        facade.getRequest().getCookie(SESSION_RANDOM_VALUE), tokenStore,
        accessTokenString, accessToken, idTokenString, idToken, refreshTokenString);
231 232         return new OidcPrincipal<>(idToken.getPrincipalName(deployment),
        secContext);
232 233     } catch (InvalidJwtException e) {
233 234         log.failedToParseTokenFromCookie(e);
  ↓

```

```

  ...ity/http/oidc/OidcRequestAuthenticator.java
  ↑
@@ -19,6 +19,7 @@
19 19 package org.wildfly.security.http.oidc;
20 20
21 21 import static org.wildfly.security.http.oidc.ElytronMessages.log;
22 + import static org.wildfly.security.http.oidc.IDToken.NONCE;
22 23 import static
        org.wildfly.security.http.oidc.Oidc.ALLOW_QUERY_PARAMS_PROPERTY_NAME;
23 24 import static org.wildfly.security.http.oidc.Oidc.CLIENT_ID;
24 25 import static org.wildfly.security.http.oidc.Oidc.CODE;
  ↓
@@ -33,6 +34,7 @@
33 34 import static org.wildfly.security.http.oidc.Oidc.REDIRECT_URI;
34 35 import static org.wildfly.security.http.oidc.Oidc.RESPONSE_TYPE;
35 36 import static org.wildfly.security.http.oidc.Oidc.SCOPE;
37 + import static org.wildfly.security.http.oidc.Oidc.SESSION_RANDOM_VALUE;
36 38 import static org.wildfly.security.http.oidc.Oidc.SESSION_STATE;

```

```

37 39 import static org.wildfly.security.http.oidc.Oidc.STATE;
38 40 import static org.wildfly.security.http.oidc.Oidc.UI_LOCALES;
@@ -47,6 +49,7 @@
47 49 import java.net.URL;
48 50 import java.security.AccessController;
49 51 import java.security.PrivilegedAction;
52 + import java.security.SecureRandom;
50 53 import java.util.ArrayList;
51 54 import java.util.Arrays;
52 55 import java.util.List;
@@ -56,6 +59,7 @@
56 59 import org.apache.http.NameValuePair;
57 60 import org.apache.http.client.utils.URIBuilder;
58 61 import org.apache.http.message.BasicNameValuePair;
62 + import org.wildfly.common.iteration.ByteIterator;
59 63 import org.wildfly.security.http.HttpConstants;
60 64
61 65 /**
@@ -75,7 +79,7 @@ public class OidcRequestAuthenticator {
75 79     protected AuthChallenge challenge;
76 80     protected String refreshToken;
77 81     protected String strippedOauthParametersRequestUri;
78 -
82 +     private int NONCE_SIZE = 36;
79 83     static final boolean ALLOW_QUERY_PARAMS_PROPERTY;
80 84
81 85     static {
@@ -161,7 +165,7 @@ protected String getCode() {
161 165         return getQueryParamValue(facade, CODE);
162 166     }
163 167
164 -     protected String getRedirectUri(String state) {
168 +     protected String getRedirectUri(String state, String
sessionRandomValueHash) {
165 169         String url = getRequestUrl();
166 170         log.debugf("callback uri: %s", url);
167 171
@@ -199,7 +203,8 @@ protected String getRedirectUri(String state) {

```

```

199 203         .addParameter(RESPONSE_TYPE, CODE)
200 204         .addParameter(CLIENT_ID, deployment.getResourceName())
201 205         .addParameter(REDIRECT_URI, rewrittenRedirectUri(url))
202 -         .addParameter(STATE, state);
206 +         .addParameter(STATE, state)
207 +         .addParameter(NONCE, sessionRandomValueHash);
203 208         redirectUriBuilder.addParameters(forwardedQueryParams);
204 209         return redirectUriBuilder.build().toString();
205 210     } catch (URISyntaxException e) {
@@ -217,7 +222,8 @@ protected String getStateCode() {
217 222
218 223         protected AuthChallenge loginRedirect() {
219 224             final String state = getStateCode();
220 -             final String redirect = getRedirectUri(state);
225 +             final String sessionRandomValue = generateSessionRandomValue();
226 +             final String redirect = getRedirectUri(state,
Oidc.getCryptographicValue(sessionRandomValue));
221 227             if (redirect == null) {
222 228                 return challenge(HttpStatus.SC_FORBIDDEN,
AuthenticationError.Reason.NO_REDIRECT_URI, null);
223 229             }
@@ -235,6 +241,7 @@ public boolean challenge(OidcHttpFacade exchange) {
235 241
exchange.getResponse().setStatus(HttpStatus.SC_MOVED_TEMPORARILY);
236 242
exchange.getResponse().setCookie(deployment.getStateCookieName(), state, "/",
null, -1,
deployment.getSSLRequired().isRequired(facade.getRequest().getRemoteAddr()),
true);
237 243         exchange.getResponse().setHeader(HttpConstants.LOCATION,
redirect);
244 +         exchange.getResponse().setCookie(SESSION_RANDOM_VALUE,
sessionRandomValue, "/", null, -1,
deployment.getSSLRequired().isRequired(facade.getRequest().getRemoteAddr()),
true);
238 245         return true;
239 246     }
240 247 };
@@ -362,7 +369,8 @@ protected AuthChallenge resolveCode(String code) {

```

```

362 369
363 370         try {
364 371             TokenValidator tokenValidator =
TokenValidator.builder(deployment).build();
365 -             TokenValidator.VerifiedTokens verifiedTokens =
tokenValidator.parseAndVerifyToken(idTokenString, tokenString);
372 +             TokenValidator.VerifiedTokens verifiedTokens =
tokenValidator.parseAndVerifyToken(idTokenString, tokenString,
373 +             facade.getRequest().getCookie(SESSION_RANDOM_VALUE));
366 374             idToken = verifiedTokens.getIdToken();
367 375             token = verifiedTokens.getAccessToken();
368 376             log.debug("Token Verification succeeded!");
⌵
@@ -435,4 +443,11 @@ private static boolean hasScope(String scopeParam,
String targetScope) {
435 443         }
436 444         return false;
437 445     }
446 +
447 +     private String generateSessionRandomValue() {
448 +         SecureRandom random = new SecureRandom();
449 +         byte[] nonceData = new byte[NONCE_SIZE];
450 +         random.nextBytes(nonceData);
451 +         return ByteIterator.ofBytes(nonceData).base64Encode().drainToString();
452 +     }
438 453     }

```

```

▼ ...tp/oidc/RefreshableOidcSecurityContext.java
⌵
@@ -34,16 +34,18 @@ public class RefreshableOidcSecurityContext extends
OidcSecurityContext {
34 34         protected transient OidcClientConfiguration clientConfiguration;
35 35         protected transient OidcTokenStore tokenStore;
36 36         protected String refreshToken;
37 +         protected transient OidcHttpFacade.Cookie cookie;
37 38
38 39         public RefreshableOidcSecurityContext() {
39 40     }
40 41
41 -         public RefreshableOidcSecurityContext(OidcClientConfiguration
clientConfiguration, OidcTokenStore tokenStore, String tokenString,

```

```

42  +      public RefreshableOidcSecurityContext(OidcClientConfiguration
      clientConfiguration, OidcHttpFacade.Cookie cookie, OidcTokenStore tokenStore,
      String tokenString,
42  43          AccessToken token, String
      idTokenString, IDToken idToken, String refreshToken) {
43  44          super(tokenString, token, idTokenString, idToken);
44  45          this.clientConfiguration = clientConfiguration;
45  46          this.tokenStore = tokenStore;
46  47          this.refreshToken = refreshToken;
48  +      this.cookie = cookie;
47  49      }
48  50
49  51      @Override
      @@ -149,7 +151,7 @@ public boolean refreshToken(boolean checkActive) {
149  151          IDToken idToken;
150  152          try {
151  153              TokenValidator tokenValidator =
      TokenValidator.builder(clientConfiguration).build();
152  -      TokenValidator.VerifiedTokens verifiedTokens =
      tokenValidator.parseAndVerifyToken(idTokenString, accessTokenString);
154  +      TokenValidator.VerifiedTokens verifiedTokens =
      tokenValidator.parseAndVerifyToken(idTokenString, accessTokenString, cookie);
153  155          idToken = verifiedTokens.getIdToken();
154  156          accessToken = verifiedTokens.getAccessToken();
155  157          log.debug("Token Verification succeeded!");
      @@ -202,14 +203,14 @@ protected boolean verifySSL() {
202  203      }

```

```

  ...ecurity/http/oidc/RequestAuthenticator.java
      @@ -24,6 +24,7 @@
24  24      import static org.wildfly.security.http.oidc.Oidc.FACES_REQUEST;
25  25      import static org.wildfly.security.http.oidc.Oidc.HTML_CONTENT_TYPE;
26  26      import static org.wildfly.security.http.oidc.Oidc.PARTIAL;
27  +      import static org.wildfly.security.http.oidc.Oidc.SESSION_RANDOM_VALUE;
27  28      import static org.wildfly.security.http.oidc.Oidc.SOAP_ACTION;
28  29      import static org.wildfly.security.http.oidc.Oidc.TEXT_CONTENT_TYPE;
29  30      import static org.wildfly.security.http.oidc.Oidc.WILDCARD_CONTENT_TYPE;
      @@ -202,14 +203,14 @@ protected boolean verifySSL() {
202  203      }

```

```

203 204
204 205     protected void completeAuthentication(OidcRequestAuthenticator oidc) {
205 -         RefreshableOidcSecurityContext session = new
RefreshableOidcSecurityContext(deployment, facade.getTokenStore(),
oidc.getTokenString(), oidc.getToken(), oidc.getIDTokenString(),
oidc.getIDToken(), oidc.getRefreshToken());
206 +         RefreshableOidcSecurityContext session = new
RefreshableOidcSecurityContext(deployment,
facade.getRequest().getCookie(SESSION_RANDOM_VALUE), facade.getTokenStore(),
oidc.getTokenString(), oidc.getToken(), oidc.getIDTokenString(),
oidc.getIDToken(), oidc.getRefreshToken());
206 207         final OidcPrincipal<RefreshableOidcSecurityContext> principal = new
OidcPrincipal<>(oidc.getIDToken().getPrincipalName(deployment), session);
207 208         completeOidcAuthentication(principal);
208 209         log.debugv("User '{0}' invoking '{1}' on client '{2}'",
principal.getName(), facade.getRequest().getURI(),
deployment.getResourceName());
209 210     }
210 211
211 212     protected void completeAuthentication(BearerTokenRequestAuthenticator
bearer) {
212 -         RefreshableOidcSecurityContext session = new
RefreshableOidcSecurityContext(deployment, null, bearer.getTokenString(),
bearer.getToken(), null, null, null);
213 +         RefreshableOidcSecurityContext session = new
RefreshableOidcSecurityContext(deployment,
facade.getRequest().getCookie(SESSION_RANDOM_VALUE), null,
bearer.getTokenString(), bearer.getToken(), null, null, null);
213 214         final OidcPrincipal<RefreshableOidcSecurityContext> principal = new
OidcPrincipal<>(bearer.getToken().getPrincipalName(deployment), session);
214 215         completeBearerAuthentication(principal);
215 216         log.debugv("User '{0}' invoking '{1}' on client '{2}'",
principal.getName(), facade.getRequest().getURI(),
deployment.getResourceName());

```



▼ ...dfly/security/http/oidc/TokenValidator.java ...



@@ -24,6 +24,7 @@

```

24 24     import static org.wildfly.security.http.oidc.Oidc.INVALID_AT_HASH_CLAIM;
25 25     import static org.wildfly.security.http.oidc.Oidc.INVALID_ISSUED_FOR_CLAIM;

```

```

26 26 import static org.wildfly.security.http.oidc.Oidc.INVALID_TYPE_CLAIM;
27 + import static org.wildfly.security.http.oidc.Oidc.INVALID_SESSION_RANDOM_VALUE;
27 28 import static org.wildfly.security.http.oidc.Oidc.getJavaAlgorithmForHash;
28 29 import static org.wildfly.security.jose.jwk.JWKUtil.BASE64_URL;
29 30
    ↓
    ↑
@@ -82,12 +83,14 @@ private TokenValidator(Builder builder) {
82 83     * @return the {@code VerifiedTokens} if the ID token was valid
83 84     * @throws OidcException if the ID token is invalid
84 85     */
85 - public VerifiedTokens parseAndVerifyToken(final String idToken, final
String accessToken) throws OidcException {
86 + public VerifiedTokens parseAndVerifyToken(final String idToken, final
String accessToken, OidcHttpFacade.Cookie cookie) throws OidcException {
86 87     try {
87 88         JwtContext idJwtContext = setVerificationKey(idToken,
jwtConsumerBuilder);
88 89
jwtConsumerBuilder.setExpectedAudience(clientConfiguration.getResourceName());
89 90         jwtConsumerBuilder.registerValidator(new
AzpValidator(clientConfiguration.getResourceName()));
90 91         jwtConsumerBuilder.registerValidator(new
AtHashValidator(accessToken,
clientConfiguration.getTokenSignatureAlgorithm()));
92 +         jwtConsumerBuilder.registerValidator(new NonceValidator(cookie));
93 +
91 94         // second pass to validate
92 95         jwtConsumerBuilder.build().processContext(idJwtContext);
93 96         JwtClaims idJwtClaims = idJwtContext.getJwtClaims();
    ↓
    ↑
@@ -276,6 +279,32 @@ public ErrorCodesValidator.Error validate(JwtContext
jwtContext) throws Malformed
276 279     }
277 280     }
278 281
282 + private static class NonceValidator implements ErrorCodesValidator {
283 +     private OidcHttpFacade.Cookie cookie;
284 +
285 +     public NonceValidator(OidcHttpFacade.Cookie cookie) {
286 +         this.cookie = cookie;
287 +     }

```

```

288 +
289 +     public ErrorCodeValidator.Error validate(JwtContext jwtContext) throws
    MalformedClaimException {
290 +         JwtClaims idJwtClaims = jwtContext.getJwtClaims();
291 +         IDToken idToken = new IDToken(idJwtClaims);
292 +
293 +         if (cookie != null) {
294 +             String sessionRandomValue =
Oidc.getCryptographicValue(cookie.getValue());
295 +             String nonceValue = idToken.getNonce();
296 +             if (!sessionRandomValue.equals(nonceValue)) {
297 +                 return new
    ErrorCodeValidator.Error(INVALID_SESSION_RANDOM_VALUE,
298 +                     log.invalidNonceValue(nonceValue));
299 +             }
300 +         } else {
301 +             return new
    ErrorCodeValidator.Error(INVALID_SESSION_RANDOM_VALUE,
302 +                     log.nonceCookieDoesNotExist());
303 +         }
304 +         return null;
305 +     }
306 + }
307 +

```

```

279 308     private static class TypeValidator implements ErrorCodeValidator {
280 309         public static final String TYPE = "typ";
281 310         private final String expectedType;

```



...ildfly/security/http/oidc/OidcBaseTest.java



@@ -22,6 +22,7 @@

```

22 22     import static org.junit.Assert.assertNotNull;
23 23     import static org.junit.Assert.assertTrue;
24 24     import static org.wildfly.security.http.oidc.Oidc.OIDC_NAME;
25 + import static org.wildfly.security.http.oidc.Oidc.SESSION_RANDOM_VALUE;
25 26
26 27     import java.io.ByteArrayInputStream;
27 28     import java.io.IOException;

```



@@ -185,6 +186,32 @@ public MockResponse dispatch(RecordedRequest



recordedRequest) throws InterruptedException

```
185 186         };
186 187     }
187 188
189 +     protected static Dispatcher
190 +     createAppResponse(HttpServerAuthenticationMechanism mechanism,
191 +                       int expectedStatusCode,
192 +                       String expectedLocation,
193 +                       String clientPageText,
194 +                       List<HttpServerCookie> cookies) {
195 +         return new Dispatcher() {
196 +             @Override
197 +             public MockResponse dispatch(RecordedRequest recordedRequest)
198 +             throws InterruptedException {
199 +                 String path = recordedRequest.getPath();
200 +                 if (path.contains("/") + CLIENT_APP) && path.contains("&code="))
201 +                 {
202 +                     try {
203 +                         TestingHttpRequest request = new
204 +                         TestingHttpRequest(new String[0],
205 +                         new
206 +                         URI(recordedRequest.getRequestUrl().toString()), cookies);
207 +                         mechanism.evaluateRequest(request);
208 +                         TestingHttpResponse response =
209 +                         request.getResponse();
210 +                         assertEquals(expectedStatusCode,
211 +                         response.getStatusCode());
212 +                         assertEquals(expectedLocation, response.getLocation());
213 +                         return new MockResponse().setBody(clientPageText);
214 +                     } catch (Exception e) {
215 +                         throw new RuntimeException(e);
216 +                     }
217 +                 }
218 +                 return new MockResponse()
219 +                 .setBody("");
220 +             }
221 +         };
222 +     }
223 + }
224 +
225 +     protected static Dispatcher
226 +     createAppResponse(HttpServerAuthenticationMechanism mechanism, int
```

```

expectedStatusCode, String expectedLocation, String clientPageText,
189 216                                     Map<String, Object>
    sessionScopeAttachments) {
190 217         return new Dispatcher() {
    @@ -305,27 +332,43 @@ protected String getCookieString(HttpServerCookie
    cookie) {
305 332         return header.toString();
306 333     }
307 334
335 +     protected void performAuthentication(InputStream oidcConfig, String
    username, String password, boolean loginToKeycloak,
336 +         int expectedDispatcherStatusCode,
    String expectedLocation, String clientPageText,
337 +         boolean changeSessionId) throws
    Exception {
338 +         performAuthentication(oidcConfig, username, password, loginToKeycloak,
    expectedDispatcherStatusCode, getClientUrl(),
339 +         expectedLocation, clientPageText, changeSessionId);
340 +     }
341 +
308 342     protected void performAuthentication(InputStream oidcConfig, String
    username, String password, boolean loginToKeycloak,
309 343         int expectedDispatcherStatusCode,
    String expectedLocation, String clientPageText) throws Exception {
310 -         performAuthentication(oidcConfig, username, password, loginToKeycloak,
    expectedDispatcherStatusCode, getClientUrl(), expectedLocation,
    clientPageText);
344 +         performAuthentication(oidcConfig, username, password, loginToKeycloak,
    expectedDispatcherStatusCode, getClientUrl(), expectedLocation, clientPageText,
    false);
311 345     }
312 346
313 347     protected void performAuthentication(InputStream oidcConfig, String
    username, String password, boolean loginToKeycloak,
314 348         int expectedDispatcherStatusCode,
    String expectedLocation, String clientPageText,
315 349         CallbackHandler callbackHandler)
    throws Exception {
316 350         performAuthentication(oidcConfig, username, password, loginToKeycloak,
    expectedDispatcherStatusCode, getClientUrl(), expectedLocation, clientPageText,

```

```

317 -         callbackHandler);
351 +         callbackHandler, false);
352 +     }
353 +
354 +     protected void performAuthentication(InputStream oidcConfig, String
        username, String password, boolean loginToKeycloak,
355 +         int expectedDispatcherStatusCode,
        String clientUrl, String expectedLocation,
356 +         String clientPageText) throws
        Exception {
357 +         performAuthentication(oidcConfig, username, password, loginToKeycloak,
358 +             expectedDispatcherStatusCode, clientUrl, expectedLocation,
359 +             clientPageText, false);
318 360     }
319 361
320 362     protected void performAuthentication(InputStream oidcConfig, String
        username, String password, boolean loginToKeycloak,
321 -         int expectedDispatcherStatusCode,
        String clientUrl, String expectedLocation, String clientPageText) throws
        Exception {
363 +         int expectedDispatcherStatusCode,
        String clientUrl, String expectedLocation,
364 +         String clientPageText, boolean
        changeSessionId) throws Exception {
322 365         performAuthentication(oidcConfig, username, password, loginToKeycloak,
        expectedDispatcherStatusCode, clientUrl, expectedLocation, clientPageText,
323 -         getCallbackHandler());
366 +         getCallbackHandler(), changeSessionId);
324 367     }
325 368
326 369     protected void performAuthentication(InputStream oidcConfig, String
        username, String password, boolean loginToKeycloak,
327 370         int expectedDispatcherStatusCode,
        String clientUrl, String expectedLocation, String clientPageText,
328 -         CallbackHandler callbackHandler)
        throws Exception {
371 +         CallbackHandler callbackHandler,
        boolean changeSessionId) throws Exception {
329 372         try {
330 373             Map<String, Object> props = new HashMap<>();

```

```

331 374         OidcClientConfiguration oidcClientConfiguration =
OidcClientConfigurationBuilder.build(oidcConfig);
@@ -343,7 +386,20 @@ protected void performAuthentication(InputStream
oidcConfig, String username, St
343 386         assertEquals(Status.NO_AUTH, request.getResult());
344 387
345 388         if (loginToKeycloak) {
346 -         client.setDispatcher(createAppResponse(mechanism,
expectedDispatcherStatusCode, expectedLocation, clientPageText));
389 +         // change the sessionRandomValue value so that the compare to
nonce will fail.
390 +         List<HttpServerCookie> tmpCookies = response.getCookies();
391 +         if (changeSessionId) {
392 +             for (HttpServerCookie c : tmpCookies) {
393 +                 if (c.getName().equals(SESSION_RANDOM_VALUE)) {
394 +                     HttpServerCookie tmpCookie =
HttpServerCookie.getInstance(c.getName(),
395 +                             "9999" + c.getValue(), c.getDomain(),
c.getMaxAge(), c.getPath(),
396 +                             c.isSecure(), c.getVersion(),
c.isHttpOnly());
397 +                     tmpCookies.remove(c);
398 +                     tmpCookies.add(tmpCookie);
399 +                 }
400 +             }
401 +         }
402 +         client.setDispatcher(createAppResponse(mechanism,
expectedDispatcherStatusCode, expectedLocation, clientPageText, tmpCookies));
347 403         TextPage page = loginToKeycloak(username, password, requestUri,
response.getLocation(),
348 404         response.getCookies()).click();
349 405         assertTrue(page.getContent().contains(clientPageText));

```

```

...curity/http/oidc/OidcSecurityRealmTest.java
@@ -72,7 +72,7 @@ public void testGetRealmIdentityWithNonOidcPrincipal()
throws RealmUnavailableEx
72 72         @Test
73 73         public void testGetRealmIdentityNoRoles() throws RealmUnavailableException
{

```

```

74 74 // setup
75 - RefreshableOidcSecurityContext securityContext = new
RefreshableOidcSecurityContext(new OidcClientConfiguration(),
75 + RefreshableOidcSecurityContext securityContext = new
RefreshableOidcSecurityContext(new OidcClientConfiguration(), null,
76 76 null, null, new AccessToken(new JwtClaims()), null, null,
null);
77 77 OidcPrincipal principal = new OidcPrincipal("john", securityContext);
78 78
⌵ @@ -108,7 +108,7 @@ public void testGetRealmIdentityRolesCombined() throws
RealmUnavailableException
⌴
108 108 jwtClaims.setClaim("resource_access", resourceAccess);
109 109 jwtClaims.setClaim("realm_access", createRoles("roleC", "roleD"));
110 110
111 - RefreshableOidcSecurityContext securityContext = new
RefreshableOidcSecurityContext(clientConfiguration,
111 + RefreshableOidcSecurityContext securityContext = new
RefreshableOidcSecurityContext(clientConfiguration, null,
112 112 null, null, new AccessToken(jwtClaims), null, null, null);
113 113 OidcPrincipal principal = new OidcPrincipal("john", securityContext);
114 114
⌵ @@ -137,7 +137,7 @@ public void testGetRealmIdentityOnlyRealmRoles() throws
RealmUnavailableExceptio
⌴
137 137 jwtClaims.setClaim("resource_access", resourceAccess);
138 138 jwtClaims.setClaim("realm_access", createRoles("roleC", "roleD"));
139 139
140 - RefreshableOidcSecurityContext securityContext = new
RefreshableOidcSecurityContext(clientConfiguration,
140 + RefreshableOidcSecurityContext securityContext = new
RefreshableOidcSecurityContext(clientConfiguration, null,
141 141 null, null, new AccessToken(jwtClaims), null, null, null);
142 142 OidcPrincipal principal = new OidcPrincipal("john", securityContext);
143 143
⌵ @@ -165,7 +165,7 @@ public void testGetRealmIdentityOnlyResourceRoles()
throws RealmUnavailableExcep
⌴
165 165 jwtClaims.setClaim("resource_access", resourceAccess);
166 166 jwtClaims.setClaim("", new RealmAccessClaim(createRoles("roleC",
"roleD"))));
167 167

```

168	-	RefreshableOidcSecurityContext securityContext = new RefreshableOidcSecurityContext(clientConfiguration,
168	+	RefreshableOidcSecurityContext securityContext = new RefreshableOidcSecurityContext(clientConfiguration, null,
169	169	null, null, new AccessToken(jwtClaims), null, null, null);
170	170	OidcPrincipal principal = new OidcPrincipal("john", securityContext);
171	171	
⌵		@@ -193,7 +193,7 @@ public void testGetRealmIdentityNoMappings() throws RealmUnavailableException {
193	193	jwtClaims.setClaim("resource_access", resourceAccess);
194	194	jwtClaims.setClaim("", new RealmAccessClaim(createRoles("roleC", "roleD")));
195	195	
196	-	RefreshableOidcSecurityContext securityContext = new RefreshableOidcSecurityContext(clientConfiguration,
196	+	RefreshableOidcSecurityContext securityContext = new RefreshableOidcSecurityContext(clientConfiguration, null,
197	197	null, null, new AccessToken(jwtClaims), null, null, null);
198	198	OidcPrincipal principal = new OidcPrincipal("john", securityContext);
199	199	
⌵		@@ -390,7 +390,7 @@ private static String getRealmAndResourceRolesClaims(boolean includeRealmAndReso
390	390	}
391	391	
392	392	private Attributes.Entry getRealmIdentityRoles(OidcClientConfiguration clientConfiguration, JwtClaims jwtClaims) throws RealmUnavailableException {
393	-	RefreshableOidcSecurityContext securityContext = new RefreshableOidcSecurityContext(clientConfiguration,
393	+	RefreshableOidcSecurityContext securityContext = new RefreshableOidcSecurityContext(clientConfiguration, null,
394	394	null, null, new AccessToken(jwtClaims), null, null, null);
395	395	OidcPrincipal principal = new OidcPrincipal("john", securityContext);
396	396	
⌵		

⌵		...rg/wildfly/security/http/oidc/OidcTest.java
⌵		@@ -206,6 +206,13 @@ KeycloakConfiguration.ALICE_PASSWORD, true, HttpStatus.SC_MOVED_TEMPORARILY, get
206	206	getCallbackHandler());
207	207	}

```

208 208
209 + @Test
210 + public void testStandardConfigWithNonceMismatch() throws Exception {
211 +     performAuthentication(getOidcConfigurationInputStreamWithProviderUrl(),
212 +         KeycloakConfiguration.ALICE,
213 +         KeycloakConfiguration.ALICE_PASSWORD, true,
214 +         HttpStatus.SC_FORBIDDEN, null, CLIENT_PAGE_TEXT, true);
215 + }
216
217 /*****
218 *****/
219 * Tests for multi-tenancy.
220 *

```

▼ ...curity/http/impl/AbstractBaseHttpTest.java

```

@@ -191,49 +191,17 @@ public TestingHttpRequest(String[]
authorization, URI requestURI, String c
191 191         this.requestURI = requestURI;
192 192         this.cookies = new ArrayList<>();
193 193         if (cookie != null) {
194 -             final String cookieName = cookie.substring(0,
195 -                 cookie.indexOf('='));
196 -             final String cookieValue = cookie.substring(cookie.indexOf('=')
197 -                 + 1);
198 -             cookies.add(new HttpServerCookie() {
199 -                 @Override
200 -                 public String getName() {
201 -                     return cookieName;
202 -                 }
203 -                 @Override
204 -                 public String getValue() {
205 -                     return cookieValue;
206 -                 }
207 -                 @Override
208 -                 public String getDomain() {
209 -                     return null;

```

```
210 - }
211 -
212 - @Override
213 - public int getMaxAge() {
214 -     return -1;
215 - }
216 -
217 - @Override
218 - public String getPath() {
219 -     return "/";
220 - }
221 -
222 - @Override
223 - public boolean isSecure() {
224 -     return false;
225 - }
226 -
227 - @Override
228 - public int getVersion() {
229 -     return 0;
230 - }
231 -
232 - @Override
233 - public boolean isHttpOnly() {
234 -     return true;
235 +
236 + String[] cookiesArr = cookie.split(";");
237 +
238 + if (cookiesArr.length == 0) {
239 +     cookiesArr[0] = cookie;
240 + }
241 +
242 + for (int i = 0; i < cookiesArr.length; i++) {
243 +     String[] cookiePair = cookiesArr[i].trim().split("=");
244 +     if (cookiePair.length == 2) {
245 +         this.cookies.add(HttpServerCookie.getInstance(
246 +             cookiePair[0], cookiePair[1], null, -1, "/",
247 +             false, 0, true));
248 +     }
249 + }
250 +
251 + }
252 +
253 + }
254 + }
```

239 207



Comments 0



Please [sign in](#) to comment.