wildfly-security / **wildfly-elytron** Public

Code | Pull requests 32 | Actions | Projects | Wiki | Security and quality 1

Commit 5ac5e6b

rsearls authored and **darranl** committed on Mar 18, 2025

```
[ELY-2887] Add a nonce to OIDC requests for CVE-2024-12369
```

2.x · 3.0.0.Alpha1 ••• 2.6.2.Final

1 parent 8b40fbd commit 5ac5e6b

13 files changed +209 -34

Top

Filter files…

- http/oidc/src
  - 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

Search within code

...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

```
@@ -278,5 +278,14 @@ interface ElytronMessages extends BasicLogger {
278 278
279 279      @Message(id = 23070, value = "Authentication request format must be one of
        the following: oauth2, request, request_uri.")
280 280      RuntimeException invalidAuthenticationRequestFormat();
    281 +
    282 +    @Message(id = 23071, value = "Invalid ID token nonce: %s")
    283 +    String invalidNonceValue(String name);
    284 +
    285 +    @Message(id = 23072, value = "No such algorithm: '%s'")
    286 +    IllegalArgumentException noSuchAlgorithm(String algorithm);
    287 +
    288 +    @Message(id = 23073, value = "Nonce cookie does not exist")
    289 +    String nonceCookieDoesNotExist();
281 290  }
282 291
```

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

```
@@ -53,6 +53,7 @@ public class IDToken extends JsonWebToken {
53  53      public static final String CLAIMS_LOCALES = "claims_locales";
54  54      public static final String ACR = "acr";
55  55      public static final String S_HASH = "s_hash";
    56  +   public static final String NONCE = "nonce";
56  57
57  58      /**
58  59       * Construct a new instance.
@@ -228,4 +229,7 @@ public String getAcr() {
228 229          return getClaimValueAsString(ACR);
229 230      }
230 231
    232 +   public String getNonce() {
    233 +       return getClaimValueAsString(NONCE);
    234 +   }
231 235  }
```

...va/org/wildfly/security/http/oidc/Oidc.java

```
@@ -19,11 +19,15 @@
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.jose.jwk.JWKUtil.BASE64_URL;
22  23
23  24  import java.io.IOException;
24  25  import java.io.InputStream;
25  26  import java.net.InetAddress;
26  27  import java.net.UnknownHostException;
    28  + import java.nio.charset.StandardCharsets;
    29  + import java.security.MessageDigest;
    30  + import java.security.NoSuchAlgorithmException;
27  31  import java.util.Map;
28  32  import java.util.StringTokenizer;
29  33  import java.util.UUID;
@@ -34,6 +38,7 @@
34  38  import org.apache.http.HttpResponse;
35  39  import org.apache.http.client.methods.HttpRequestBase;
36  40  import org.jose4j.jws.AlgorithmIdentifiers;
    41  + import org.wildfly.common.iteration.ByteIterator;
```

```
37  42    import org.wildfly.security.jose.util.JsonSerialization;
38  43
39  44    /**
          @@ -111,6 +116,7 @@ public class Oidc {
111 116       public static final String REDIRECT_URI = "redirect_uri";
112 117       public static final String REFRESH_TOKEN = "refresh_token";
113 118       public static final String RESPONSE_TYPE = "response_type";
    119 +     public static final String SESSION_RANDOM_VALUE="session_random_value";
114 120       public static final String SESSION_STATE = "session_state";
115 121       public static final String SOAP_ACTION = "SOAPAction";
116 122       public static final String SSL_REQUIRED = "ssl-required";
          @@ -119,6 +125,7 @@ public class Oidc {
119 125       public static final int INVALID_ISSUED_FOR_CLAIM = -1;
120 126       public static final int INVALID_AT_HASH_CLAIM = -2;
121 127       public static final int INVALID_TYPE_CLAIM = -3;
    128 +     public static final int INVALID_SESSION_RANDOM_VALUE = -4;
122 129       static final String OIDC_CLIENT_CONFIG_RESOLVER = "oidc.config.resolver";
123 130       static final String OIDC_CONFIG_FILE_LOCATION = "oidc.config.file";
124 131       static final String OIDC_JSON_FILE = "/WEB-INF/oidc.json";
          @@ -445,4 +452,14 @@ protected static boolean
          checkCachedAccountMatchesRequest(OidcAccount account, O
445 452           return true;
446 453       }
447 454
    455 +     protected static String getCryptographicValue(final String src) {
    456 +         try {
    457 +             MessageDigest md = MessageDigest.getInstance(SHA256);
    458 +             md.update(src.getBytes(StandardCharsets.UTF_8));
    459 +             return ByteIterator.ofBytes(md.digest())
    460 +                     .base64Encode(BASE64_URL, false).drainToString();
    461 +         } catch (NoSuchAlgorithmException e) {
    462 +             throw log.noSuchAlgorithm(e.getMessage());
    463 +         }
    464 +     }
448 465   }
```

...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().buil
            d().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

```
            @@ -23,6 +23,7 @@
23  23      import static org.jose4j.jws.AlgorithmIdentifiers.HMAC_SHA512;
24  24      import static org.jose4j.jws.AlgorithmIdentifiers.NONE;
25  25      import static org.wildfly.security.http.oidc.ElytronMessages.log;
        26  + import static org.wildfly.security.http.oidc.IDToken.NONCE;
26  27      import static
            org.wildfly.security.http.oidc.Oidc.ALLOW_QUERY_PARAMS_PROPERTY_NAME;
27  28      import static org.wildfly.security.http.oidc.Oidc.CLIENT_ID;
28  29      import static org.wildfly.security.http.oidc.Oidc.CODE;
            @@ -39,6 +40,7 @@
39  40      import static org.wildfly.security.http.oidc.Oidc.REQUEST;
40  41      import static org.wildfly.security.http.oidc.Oidc.REQUEST_URI;
```

```
41  42    import static org.wildfly.security.http.oidc.Oidc.SCOPE;
    43  + import static org.wildfly.security.http.oidc.Oidc.SESSION_RANDOM_VALUE;
42  44    import static org.wildfly.security.http.oidc.Oidc.SESSION_STATE;
43  45    import static org.wildfly.security.http.oidc.Oidc.STATE;
44  46    import static org.wildfly.security.http.oidc.Oidc.UI_LOCALES;
        ...  @@ -59,6 +61,7 @@
59  61    import java.security.Key;
60  62    import java.security.KeyPair;
61  63    import java.security.PublicKey;
    64  + import java.security.SecureRandom;
62  65    import java.util.ArrayList;
63  66    import java.util.Arrays;
64  67    import java.util.HashSet;
        ...  @@ -76,6 +79,7 @@
76  79    import org.jose4j.jwt.JwtClaims;
77  80    import org.jose4j.keys.HmacKey;
78  81    import org.jose4j.lang.JoseException;
    82  + import org.wildfly.common.iteration.ByteIterator;
79  83    import org.wildfly.security.http.HttpConstants;
80  84
81  85    /**
        ...  @@ -96,6 +100,8 @@ public class OidcRequestAuthenticator {
96  100       protected String refreshToken;
97  101       protected String strippedOauthParametersRequestUri;
98  102
    103 +     private int NONCE_SIZE = 36;
    104 +
99  105       static final boolean ALLOW_QUERY_PARAMS_PROPERTY;
100 106
101 107       static {
        ...  @@ -181,7 +187,7 @@ protected String getCode() {
181 187           return getQueryParamValue(facade, CODE);
182 188       }
183 189
184     -     protected String getRedirectUri(String state) {
    190 +     protected String getRedirectUri(String state, String
        sessionRandomValueHash) {
185 191           String url = getRequestUrl();
186 192           log.debugf("callback uri: %s", url);
```

```
187  193
     @@ -229,31 +235,31 @@ protected String getRedirectUri(String state) {
229  235                        if (deployment.getRequestParameterSupported()) {
230  236                            // add request objects into request parameter
231  237                            try {
232       -                            createRequestWithRequestParameter(REQUEST, redirectUriBuilder, redirectUri, state, forwardedQueryParams);
     238  +                            createRequestWithRequestParameter(REQUEST, redirectUriBuilder, redirectUri, state, forwardedQueryParams, sessionRandomValueHash);
233  239                            } catch (IOException | JoseException e) {
234  240                                throw log.unableToCreateRequestWithRequestParameter(e);
235  241                            }
236  242                        } else {
237  243                            // send request as usual
238       -                            createOAuthRequest(redirectUriBuilder, redirectUri, state, forwardedQueryParams);
     244  +                            createOAuthRequest(redirectUriBuilder, redirectUri, state, forwardedQueryParams, sessionRandomValueHash);
239  245                            log.requestParameterNotSupported();
240  246                        }
241  247                        break;
242  248                    case REQUEST_URI:
243  249                        if (deployment.getRequestUriParameterSupported()) {
244  250                            try {
245       -                            createRequestWithRequestParameter(REQUEST_URI, redirectUriBuilder, redirectUri, state, forwardedQueryParams);
     251  +                            createRequestWithRequestParameter(REQUEST_URI, redirectUriBuilder, redirectUri, state, forwardedQueryParams, sessionRandomValueHash);
246  252                            } catch (IOException | JoseException e) {
247  253                                throw log.unableToCreateRequestUriWithRequestParameter(e);
248  254                            }
249  255                        } else {
250  256                            // send request as usual
251       -                            createOAuthRequest(redirectUriBuilder, redirectUri, state, forwardedQueryParams);
```

```
+                        createOAuthRequest(redirectUriBuilder, redirectUri, state, forwardedQueryParams, sessionRandomValueHash);
                         log.requestParameterNotSupported();
                     }
                     break;
                 default:
-                    createOAuthRequest(redirectUriBuilder, redirectUri, state, forwardedQueryParams);
+                    createOAuthRequest(redirectUriBuilder, redirectUri, state, forwardedQueryParams, sessionRandomValueHash);
                     break;
             }
             return redirectUriBuilder.build().toString();
@@ -262,15 +268,16 @@ protected String getRedirectUri(String state) {
         }
     }

-    protected URIBuilder createOAuthRequest(URIBuilder redirectUriBuilder, String redirectUri, String state, List<NameValuePair> forwardedQueryParams) {
+    protected URIBuilder createOAuthRequest(URIBuilder redirectUriBuilder, String redirectUri, String state, List<NameValuePair> forwardedQueryParams, String sessionRandomValueHash) {
         redirectUriBuilder.addParameter(REDIRECT_URI, redirectUri)
                 .addParameter(STATE, state)
-                .addParameters(forwardedQueryParams);
+                .addParameters(forwardedQueryParams)
+                .addParameter(NONCE, sessionRandomValueHash);
         return redirectUriBuilder;
     }

-    protected URIBuilder createRequestWithRequestParameter(String requestFormat, URIBuilder redirectUriBuilder, String redirectUri, String state, List<NameValuePair> forwardedQueryParams) throws JoseException, IOException {
-        String request = convertToRequestParameter(redirectUriBuilder, redirectUri, state, forwardedQueryParams);
+    protected URIBuilder createRequestWithRequestParameter(String requestFormat, URIBuilder redirectUriBuilder, String redirectUri, String state, List<NameValuePair> forwardedQueryParams, String sessionRandomValueHash) throws JoseException, IOException {
```

```
                280 +           String request = convertToRequestParameter(redirectUriBuilder,
                    redirectUri, state, forwardedQueryParams, sessionRandomValueHash);
274             281
275             282             switch (requestFormat) {
276             283                 case REQUEST:
                    @@ -296,7 +303,8 @@ protected String getStateCode() {
296             303
297             304         protected AuthChallenge loginRedirect() {
298             305             final String state = getStateCode();
299             -           final String redirect = getRedirectUri(state);
                306 +           final String sessionRandomValue = generateSessionRandomValue();
                307 +           final String redirect = getRedirectUri(state,
                    Oidc.getCryptographicValue(sessionRandomValue));
300             308             if (redirect == null) {
301             309                 return challenge(HttpStatus.SC_FORBIDDEN,
                    AuthenticationError.Reason.NO_REDIRECT_URI, null);
302             310             }
                    @@ -314,6 +322,8 @@ public boolean challenge(OidcHttpFacade exchange) {
314             322
                    exchange.getResponse().setStatus(HttpStatus.SC_MOVED_TEMPORARILY);
315             323
                    exchange.getResponse().setCookie(deployment.getStateCookieName(), state, "/",
                    null, -1,
                    deployment.getSSLRequired().isRequired(facade.getRequest().getRemoteAddr()),
                    true);
316             324                 exchange.getResponse().setHeader(HttpConstants.LOCATION,
                    redirect);
                325 +               exchange.getResponse().setCookie(SESSION_RANDOM_VALUE,
                    sessionRandomValue, "/", null, -1,
                    deployment.getSSLRequired().isRequired(facade.getRequest().getRemoteAddr()),
                    true);
                326 +
317             327                 return true;
318             328             }
319             329         };
                    @@ -336,6 +346,7 @@ protected AuthChallenge checkStateCookie() {
336             346             log.warn("state parameter was null");
337             347             return challenge(HttpStatus.SC_BAD_REQUEST,
                    AuthenticationError.Reason.INVALID_STATE_COOKIE, null);
338             348         }
```

```
                349 +
339             350             if (!state.equals(stateCookieValue)) {
340             351                 log.warn("state parameter invalid");
341             352                 log.warn("cookie: " + stateCookieValue);
                    @@ -441,9 +452,12 @@ protected AuthChallenge resolveCode(String code) {
441             452
442             453             try {
443             454                 TokenValidator tokenValidator =
                    TokenValidator.builder(deployment).build();
444             -                 TokenValidator.VerifiedTokens verifiedTokens =
                    tokenValidator.parseAndVerifyToken(idTokenString, tokenString);
                455 +
                456 +                 TokenValidator.VerifiedTokens verifiedTokens =
                    tokenValidator.parseAndVerifyToken(idTokenString, tokenString,
                457 +                         facade.getRequest().getCookie(SESSION_RANDOM_VALUE));
445             458                 idToken = verifiedTokens.getIdToken();
446             459                 token = verifiedTokens.getAccessToken();
                460 +
447             461                 log.debug("Token Verification succeeded!");
448             462             } catch (OidcException e) {
449             463                 log.failedVerificationOfToken(e.getMessage());
                    @@ -456,6 +470,7 @@ protected AuthChallenge resolveCode(String code) {
456             470                 log.error("Stale token");
457             471                 return challenge(HttpStatus.SC_FORBIDDEN,
                    AuthenticationError.Reason.STALE_TOKEN, null);
458             472             }
                473 +
459             474             log.debug("successfully authenticated");
460             475             return null;
461             476         }
                    @@ -535,7 +550,7 @@ private void addScopes(String scopes, Set<String>
                    allScopes) {
535             550             }
536             551         }
537             552
538             -     private String convertToRequestParameter(URIBuilder redirectUriBuilder,
                    String redirectUri, String state, List<NameValuePair> forwardedQueryParams)
                    throws JoseException, IOException {
```

```
                553 +     private String convertToRequestParameter(URIBuilder redirectUriBuilder,
                    String redirectUri, String state, List<NameValuePair> forwardedQueryParams,
                    String sessionRandomValueHash) throws JoseException, IOException {
539             554           redirectUriBuilder.addParameter(SCOPE, OIDC_SCOPE);
540             555
541             556           JwtClaims jwtClaims = new JwtClaims();
                    @@ -545,10 +560,12 @@ private String convertToRequestParameter(URIBuilder
                    redirectUriBuilder, String r
545             560           for ( NameValuePair parameter: forwardedQueryParams) {
546             561               jwtClaims.setClaim(parameter.getName(), parameter.getValue());
547             562           }
                563 +
548             564           jwtClaims.setClaim(STATE, state);
549             565           jwtClaims.setClaim(REDIRECT_URI, redirectUri);
550             566           jwtClaims.setClaim(RESPONSE_TYPE, CODE);
551             567           jwtClaims.setClaim(CLIENT_ID, deployment.getResourceName());
                568 +         jwtClaims.setClaim(NONCE, sessionRandomValueHash);
552             569
553             570           // sign JWT first before encrypting
554             571           JsonWebSignature signedRequest = signRequest(jwtClaims, deployment);
                    @@ -622,4 +639,11 @@ private JsonWebEncryption
                    encryptRequest(JsonWebSignature signedRequest) throws
622             639               return jsonEncryption;
623             640           }
624             641       }
                642 +
                643 +     private String generateSessionRandomValue() {
                644 +         SecureRandom random = new SecureRandom();
                645 +         byte[] nonceData = new byte[NONCE_SIZE];
                646 +         random.nextBytes(nonceData);
                647 +         return ByteIterator.ofBytes(nonceData).base64Encode().drainToString();
                648 +     }
625             649   }
```

...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!");
         ↓
```

…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;
          @@ -200,14 +201,14 @@ protected boolean verifySSL() {
200  201       }
201  202
202  203       protected void completeAuthentication(OidcRequestAuthenticator oidc) {
203      -         RefreshableOidcSecurityContext session = new
                  RefreshableOidcSecurityContext(deployment, facade.getTokenStore(),
                  oidc.getTokenString(), oidc.getToken(), oidc.getIDTokenString(),
                  oidc.getIDToken(), oidc.getRefreshToken());
     204 +         RefreshableOidcSecurityContext session = new
                  RefreshableOidcSecurityContext(deployment,
                  facade.getRequest().getCookie(SESSION_RANDOM_VALUE), facade.getTokenStore(),
                  oidc.getTokenString(), oidc.getToken(), oidc.getIDTokenString(),
                  oidc.getIDToken(), oidc.getRefreshToken());
204  205           final OidcPrincipal<RefreshableOidcSecurityContext> principal = new
                  OidcPrincipal<>(oidc.getIDToken().getPrincipalName(deployment), session);
205  206           completeOidcAuthentication(principal);
206  207           log.debugv("User ''{0}'' invoking ''{1}'' on client ''{2}''",
                  principal.getName(), facade.getRequest().getURI(),
                  deployment.getResourceName());
207  208       }
208  209
209  210       protected void completeAuthentication(BearerTokenRequestAuthenticator
                  bearer) {
210      -         RefreshableOidcSecurityContext session = new
                  RefreshableOidcSecurityContext(deployment, null, bearer.getTokenString(),
                  bearer.getToken(), null, null, null);
     211 +         RefreshableOidcSecurityContext session = new
                  RefreshableOidcSecurityContext(deployment,
                  facade.getRequest().getCookie(SESSION_RANDOM_VALUE), null,
                  bearer.getTokenString(), bearer.getToken(), null, null, null);
211  212           final OidcPrincipal<RefreshableOidcSecurityContext> principal = new
                  OidcPrincipal<>(bearer.getToken().getPrincipalName(deployment), session);
212  213           completeBearerAuthentication(principal);
213  214           log.debugv("User ''{0}'' invoking ''{1}'' on client ''{2}''",
                  principal.getName(), facade.getRequest().getURI(),
```

```
                deployment.getResourceName());
```

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

```
@@ -23,6 +23,7 @@
23  23  import static
        org.wildfly.security.http.oidc.Oidc.DISABLE_TYP_CLAIM_VALIDATION_PROPERTY_NAME;
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  + import static org.wildfly.security.http.oidc.Oidc.INVALID_SESSION_RANDOM_VALUE;
26  27  import static org.wildfly.security.http.oidc.Oidc.INVALID_TYPE_CLAIM;
27  28  import static org.wildfly.security.http.oidc.Oidc.getJavaAlgorithmForHash;
28  29  import static org.wildfly.security.jose.jwk.JWKUtil.BASE64_URL;
@@ -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();
@@ -288,6 +291,33 @@ private static String getAccessTokenHash(String
            accessTokenString, String jwsAlg
288 291
```

```
289  292         }
290  293
     294 +
     295 +     private static class NonceValidator implements ErrorCodeValidator {
     296 +         private OidcHttpFacade.Cookie cookie;
     297 +
     298 +         public NonceValidator(OidcHttpFacade.Cookie cookie) {
     299 +             this.cookie = cookie;
     300 +         }
     301 +
     302 +         public ErrorCodeValidator.Error validate(JwtContext jwtContext) throws
           MalformedClaimException {
     303 +             JwtClaims idJwtClaims = jwtContext.getJwtClaims();
     304 +             IDToken idToken = new IDToken(idJwtClaims);
     305 +
     306 +             if (cookie != null) {
     307 +                 String sessionRandomValue =
           Oidc.getCryptographicValue(cookie.getValue());
     308 +                 String nonceValue = idToken.getNonce();
     309 +                 if (!sessionRandomValue.equals(nonceValue)) {
     310 +                     return new
           ErrorCodeValidator.Error(INVALID_SESSION_RANDOM_VALUE,
     311 +                             log.invalidNonceValue(nonceValue));
     312 +                 }
     313 +             } else {
     314 +                 return new
           ErrorCodeValidator.Error(INVALID_SESSION_RANDOM_VALUE,
     315 +                         log.nonceCookieDoesNotExist());
     316 +             }
     317 +             return null;
     318 +         }
     319 +     }
     320 +
291  321         private static class TypeValidator implements ErrorCodeValidator {
292  322             public static final String TYPE = "typ";
293  323             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;
          @@ -211,6 +212,33 @@ public MockResponse dispatch(RecordedRequest
          recordedRequest) throws Interrupted
211 212           };
212 213       }
213 214
      215 +     protected static Dispatcher
          createAppResponse(HttpServerAuthenticationMechanism mechanism,
      216 +                                                   int expectedStatusCode,
          String expectedLocation,
      217 +                                                   String clientPageText,
          List<HttpServerCookie> cookies) {
      218 +         return new Dispatcher() {
      219 +             @Override
      220 +             public MockResponse dispatch(RecordedRequest recordedRequest)
          throws InterruptedException {
      221 +                 String path = recordedRequest.getPath();
      222 +                 if (path.contains("/" + CLIENT_APP) && path.contains("&code="))
          {
      223 +                     try {
      224 +
      225 +                         TestingHttpServerRequest request = new
          TestingHttpServerRequest(new String[0],
      226 +                                 new
          URI(recordedRequest.getRequestUrl().toString()), cookies);
      227 +                         mechanism.evaluateRequest(request);
      228 +                         TestingHttpServerResponse response =
          request.getResponse();
      229 +                         assertEquals(expectedStatusCode,
          response.getStatusCode());
      230 +                         assertEquals(expectedLocation, response.getLocation());
      231 +                         return new MockResponse().setBody(clientPageText);
      232 +                     } catch (Exception e) {
      233 +                         throw new RuntimeException(e);
```

```
                    }
                }
                return new MockResponse()
                        .setBody("");
            }
        };
    }

    protected static Dispatcher createAppResponse(HttpServerAuthenticationMechanism mechanism, int expectedStatusCode, String expectedLocation, String clientPageText,
                                                  Map<String, Object> sessionScopeAttachments) {
        return new Dispatcher() {
@@ -351,24 +379,37 @@ protected void checkForScopeClaims(Callback callback, String expectedScopes) thr
    protected void performAuthentication(InputStream oidcConfig, String username, String password, boolean loginToKeycloak,
                                         int expectedDispatcherStatusCode, String expectedLocation, String clientPageText) throws Exception {
        performAuthentication(oidcConfig, username, password, loginToKeycloak, expectedDispatcherStatusCode, getClientUrl(), expectedLocation,
-                clientPageText, null, false);
+                clientPageText, null, false, false );
    }

    protected void performAuthentication(InputStream oidcConfig, String username, String password, boolean loginToKeycloak,
                                         int expectedDispatcherStatusCode, String expectedLocation, String clientPageText,
                                         boolean changeSessionId) throws Exception {
        performAuthentication(oidcConfig, username, password, loginToKeycloak, expectedDispatcherStatusCode, getClientUrl(), expectedLocation,
                clientPageText, null, false, changeSessionId);
    }

    protected void performAuthentication(InputStream oidcConfig, String username, String password, boolean loginToKeycloak,
```

```
358  393                                          int expectedDispatcherStatusCode,
     String clientUrl, String expectedLocation, String clientPageText) throws
     Exception {
359  394          performAuthentication(oidcConfig, username, password, loginToKeycloak,
     expectedDispatcherStatusCode, clientUrl, expectedLocation,
360       -                clientPageText, null, false);
          395  +                clientPageText, null, false, false);
361  396      }
362  397
          398  +    protected void performAuthentication(InputStream oidcConfig, String
     username, String password, boolean loginToKeycloak, int
     expectedDispatcherStatusCode,
          399  +                                         String expectedLocation, String
     clientPageText, String expectedScope, boolean checkInvalidScopeError,
          400  +                                         boolean changeSessionId) throws
     Exception {
          401  +         performAuthentication(oidcConfig, username, password, loginToKeycloak,
     expectedDispatcherStatusCode, getClientUrl(), expectedLocation, clientPageText,
          402  +                 expectedScope, checkInvalidScopeError, changeSessionId);
          403  +     }
363  404      protected void performAuthentication(InputStream oidcConfig, String
     username, String password, boolean loginToKeycloak, int
     expectedDispatcherStatusCode,
364  405                                          String expectedLocation, String
     clientPageText, String expectedScope, boolean checkInvalidScopeError) throws
     Exception {
365  406          performAuthentication(oidcConfig, username, password, loginToKeycloak,
     expectedDispatcherStatusCode, getClientUrl(), expectedLocation, clientPageText,
366       -                expectedScope, checkInvalidScopeError);
          407  +                expectedScope, checkInvalidScopeError, false);
367  408      }
368  409
369  410      private void performAuthentication(InputStream oidcConfig, String username,
     String password, boolean loginToKeycloak,
370  411                                         int expectedDispatcherStatusCode, String
     clientUrl, String expectedLocation, String clientPageText,
371       -                                         String expectedScope, boolean
     checkInvalidScopeError) throws Exception {
          412  +                                         String expectedScope, boolean
     checkInvalidScopeError, boolean changeSessionId) throws Exception {
```

```
372  413            try {
373  414                Map<String, Object> props = new HashMap<>();
374  415                OidcClientConfiguration oidcClientConfiguration =
     OidcClientConfigurationBuilder.build(oidcConfig);
     @@ -394,7 +435,21 @@ private void performAuthentication(InputStream
     oidcConfig, String username, Stri
394  435                }
395  436
396  437                if (loginToKeycloak) {
397       -                client.setDispatcher(createAppResponse(mechanism,
     expectedDispatcherStatusCode, expectedLocation, clientPageText));
          438 +                // change the sessionRandomValue value so that the compare to
     nonce will fail.
          439 +                List<HttpServerCookie> tmpCookies = response.getCookies();
          440 +                if (changeSessionId) {
          441 +                    for (HttpServerCookie c : tmpCookies) {
          442 +                        if (c.getName().equals(SESSION_RANDOM_VALUE)) {
          443 +                            HttpServerCookie tmpCookie =
     HttpServerCookie.getInstance(c.getName(),
          444 +                                    "9999" + c.getValue(), c.getDomain(),
     c.getMaxAge(), c.getPath(),
          445 +                                    c.isSecure(), c.getVersion(),
     c.isHttpOnly());
          446 +                            tmpCookies.remove(c);
          447 +                            tmpCookies.add(tmpCookie);
          448 +                        }
          449 +                    }
          450 +                }
          451 +                client.setDispatcher(createAppResponse(mechanism,
     expectedDispatcherStatusCode,
          452 +                        expectedLocation, clientPageText, tmpCookies));
398  453
399  454                if (checkInvalidScopeError) {
400  455                    WebClient webClient = getWebClient();
```

...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

```
     @@ -367,6 +367,28 @@ public void testRequestObjectConfigMissingENCValue() throws Exception {
```

```
367  367
                testRequestObjectInvalidConfiguration(getOidcConfigurationInputStreamWithoutEnc
                Value(REQUEST.getValue(), RSA_OAEP), RequestObjectErrorType.MISSING_ENC_VALUE);
368  368        }
369  369
     370    +     @Test
     371    +     // Generate an invalid sessionRandomValue so that the nonce check fails
     372    +     public void testRequestParameterNonceMismatch() throws Exception {
     373    +         performAuthentication(
     374    +
                  getOidcConfigurationInputStreamWithRequestParameter(REQUEST.getValue(), NONE,
                  "", ""),
     375    +                 KeycloakConfiguration.ALICE,
                  KeycloakConfiguration.ALICE_PASSWORD,true,
     376    +                 HttpStatus.SC_FORBIDDEN,null, CLIENT_PAGE_TEXT, true);
     377    +     }
     378    +
     379    +     @Test
     380    +     public void testRequestUriParameterNonceMismatch() throws Exception {
     381    +
                  performAuthentication(getOidcConfigurationInputStreamWithRequestParameter(REQUE
                  ST_URI.getValue(), NONE, "", ""),
     382    +                 KeycloakConfiguration.ALICE,
                  KeycloakConfiguration.ALICE_PASSWORD, true,
     383    +                 HttpStatus.SC_FORBIDDEN, null, CLIENT_PAGE_TEXT, true);
     384    +     }
     385    +
     386    +     @Test
     387    +     public void testStandardConfigWithNonceMismatch() throws Exception {
     388    +         performAuthentication(getOidcConfigurationInputStreamWithProviderUrl(),
                  KeycloakConfiguration.ALICE, KeycloakConfiguration.ALICE_PASSWORD,
     389    +                 true, HttpStatus.SC_FORBIDDEN, null, CLIENT_PAGE_TEXT, true);
     390    +     }
     391    +
370  392
                /**********************************************************************
                ************************************************
371  393         * Tests for multi-tenancy.
372  394         *
```

...ecurity/http/impl/AbstractBaseHttpTest.java

```
@@ -213,11 +213,20 @@ public TestingHttpServerRequest(String[] authorization, URI requestURI, String c
213 213             }
214 214             this.remoteUser = null;
215 215             this.requestURI = requestURI;
    216 +
216 217             this.cookies = new ArrayList<>();
217 218             if (cookie != null) {
218     -               final String cookieName = cookie.substring(0, cookie.indexOf('='));
219     -               final String cookieValue = cookie.substring(cookie.indexOf('=') + 1);
220     -               cookies.add(HttpServerCookie.getInstance(cookieName, cookieValue, null, -1, "/", false, 0, true));
    219 +               String[] cookiesArr = cookie.split(";");
    220 +               if (cookiesArr.length == 0) {
    221 +                   cookiesArr[0] = cookie;
    222 +               }
    223 +               for (int i = 0; i < cookiesArr.length; i++) {
    224 +                   String[] cookiePair = cookiesArr[i].trim().split("=");
    225 +                   if (cookiePair.length == 2) {
    226 +                       this.cookies.add(HttpServerCookie.getInstance(
    227 +                               cookiePair[0], cookiePair[1], null, -1, "/", false, 0, true));
    228 +                   }
    229 +               }
221 230             }
222 231         }
223 232
```

Comments 0

Please sign in to comment.