Commit 49033a6

u3s committed on Mar 18 Verified

```
public_key: Verify designated OCSP responder certificate signature

A designated OCSP responder (Case 2 in RFC 6960 §4.2.2.2) must hold
a certificate issued directly by the CA. The existing check verified
the issuer DN match and id-kp-OCSPSigning EKU but did not verify the
CA's cryptographic signature on the responder certificate.

An attacker could forge a self-signed certificate with the CA's
subject DN and OCSPSigning EKU, and it would be accepted as a valid
designated responder.

Add public_key:pkix_verify/2 call to Case 2 in is_authorized_responder/3
to verify the responder certificate was actually signed by the CA.

Add designated_responder test case covering both legitimate (CA-signed)
and forged (self-signed) designated responder certificates.

GHSA-gxrm-pf64-99xm
```

master · OTP-29.0-rc3 ••• OTP-27.3.4.10

1 parent 601a012 commit 49033a6

2 files changed +149 -2

Top

Filter files…

- lib/public_key
  - src
    - pubkey_ocsp.erl
  - test
    - pubkey_ocsp_SUITE.erl

Search within code

lib/public_key/src/pubkey_ocsp.erl

```
@@ -229,7 +229,9 @@ is_authorized_responder(CombinedResponderCert =
#cert{otp = ResponderCert},
229 229          %%      issue OCSP responses for that CA (id-kp-OCSPSigning)
230 230          fun() ->
231 231                  public_key:pkix_is_issuer(ResponderCert, IssuerCert) andalso
232     -                                 designated_for_ocsp_signing(ResponderCert)
    232 +                     designated_for_ocsp_signing(ResponderCert) andalso
    233 +                     public_key:pkix_verify(CombinedResponderCert#cert.der,
    234 +                                            get_public_key_rec(IssuerCert))
233 235          end,
234 236      Case3 =
235 237          %% a Trusted Responder whose public key is trusted by the requestor
```

lib/public_key/test/pubkey_ocsp_SUITE.erl

```
@@ -105,7 +105,7 @@
105 105  %% Common Test interface functions -----------------------------------
106 106  %%--------------------------------------------------------------------
107 107  all() ->
108     -     [ocsp_test].
    108 +     [ocsp_test, designated_responder].
109 109
110 110  groups() ->
111 111      [].
@@ -211,3 +211,148 @@ ocsp_test(Config) when is_list(Config) ->
211 211                                      ?ISSUER_CERT,
212 212                                      IsTrustedReponderFun),
213 213      ok.
    214 +
    215 + %%--------------------------------------------------------------------
    216 + designated_responder() ->
    217 +     [{doc, "Test Case2 (designated responder) in is_authorized_responder/3. "
    218 +       "Verifies that a legitimate designated responder cert signed by the CA "
    219 +       "is accepted, and a forged self-signed cert with the same subject DN "
```

```
220 +       "is rejected (GHSA-gxrm-pf64-99xm)."}].
221 + designated_responder(Config) when is_list(Config) ->
222 +     %% EC keys are used for fast key generation (vs RSA-2048).
223 +     %% is_authorized_responder/3 Case 2 only checks DN match,
224 +     %% OCSPSigning EKU, and pkix_verify — no chain validation.
225 +     CAKey = public_key:generate_key({namedCurve, ?'secp256r1'}),
226 +     CAPubKey = ec_public_key(CAKey),
227 +     CASubject = cn_subject(<<"Test CA">>),
228 +     CACertDer = public_key:pkix_sign(ca_tbs(CASubject, CAPubKey), CAKey),
229 +     CACert = public_key:pkix_decode_cert(CACertDer, otp),
230 +
231 +     %% Legitimate designated responder cert (signed by CA)
232 +     ResponderKey = public_key:generate_key({namedCurve, ?'secp256r1'}),
233 +     {ResponderCertDer, ResponderCert} =
234 +         sign_responder_cert(2, CASubject, ec_public_key(ResponderKey), CAKey),
235 +
236 +     %% Forged designated responder (self-signed, same subject DN)
237 +     ForgedKey = public_key:generate_key({namedCurve, ?'secp256r1'}),
238 +     {ForgedCertDer, ForgedCert} =
239 +         sign_responder_cert(9999, CASubject, ec_public_key(ForgedKey),
    ForgedKey),
240 +
241 +     %% Build OCSP responses and verify
242 +     Nonce = crypto:strong_rand_bytes(8),
243 +     NonceExt = <<4, 8, Nonce/binary>>,
244 +     IsNotTrustedFun = fun(_) -> false end,
245 +
246 +     %% Positive: legitimate designated responder accepted
247 +     LegitResponse = build_ocsp_response(CASubject, CAKey, NonceExt,
    ResponderKey),
248 +     {ok, [_], _} =
249 +         pubkey_ocsp:verify_response(
250 +             LegitResponse,
251 +             [#cert{otp = ResponderCert, der = ResponderCertDer}],
252 +             NonceExt, CACert, IsNotTrustedFun),
253 +
254 +     %% Negative: forged responder (same DN, not signed by CA) rejected
255 +     ForgedResponse = build_ocsp_response(CASubject, CAKey, NonceExt,
    ForgedKey),
256 +     {error, ocsp_responder_cert_not_found} =
```

```
257 +         pubkey_ocsp:verify_response(
258 +             ForgedResponse,
259 +             [#cert{otp = ForgedCert, der = ForgedCertDer}],
260 +             NonceExt, CACert, IsNotTrustedFun),
261 +     ok.
262 +
263 + %%--------------------------------------------------------------------
264 + %% Helpers -----------------------------------------------------------
265 + %%--------------------------------------------------------------------
266 +
267 + ec_public_key(#'ECPrivateKey'{publicKey = PubKey}) ->
268 +     #'ECPoint'{point = PubKey}.
269 +
270 + cn_subject(CN) ->
271 +     {rdnSequence,
272 +      [[#'AttributeTypeAndValue'{
273 +           type = ?'id-at-commonName',
274 +           value = {utf8String, CN}}]]}.
275 +
276 + ec_subject_pubkey_info(PubKey) ->
277 +     #'OTPSubjectPublicKeyInfo'{
278 +        algorithm = #'PublicKeyAlgorithm'{
279 +                       algorithm = ?'id-ecPublicKey',
280 +                       parameters = {namedCurve, ?'secp256r1'}},
281 +        subjectPublicKey = PubKey}.
282 +
283 + ca_tbs(Subject, PubKey) ->
284 +     make_tbs(1, Subject, PubKey,
285 +              [#'Extension'{extnID = ?'id-ce-basicConstraints',
286 +                            critical = true,
287 +                            extnValue = #'BasicConstraints'{cA = true}},
288 +               #'Extension'{extnID = ?'id-ce-keyUsage',
289 +                            critical = false,
290 +                            extnValue = [keyCertSign, cRLSign]}]).
291 +
292 + ocsp_responder_tbs(Serial, Issuer, PubKey) ->
293 +     make_tbs(Serial, Issuer, PubKey,
294 +              [#'Extension'{extnID = ?'id-ce-basicConstraints',
295 +                            critical = false,
296 +                            extnValue = #'BasicConstraints'{cA = false}},
```

```
297 +               #'Extension'{extnID = ?'id-ce-keyUsage',
298 +                            critical = false,
299 +                            extnValue = [digitalSignature]},
300 +               #'Extension'{extnID = ?'id-ce-extKeyUsage',
301 +                            critical = false,
302 +                            extnValue = [?'id-kp-OCSPSigning']}]).
303 +
304 + sign_responder_cert(Serial, Issuer, PubKey, SigningKey) ->
305 +     Der = public_key:pkix_sign(ocsp_responder_tbs(Serial, Issuer, PubKey),
    SigningKey),
306 +     {Der, public_key:pkix_decode_cert(Der, otp)}.
307 +
308 + make_tbs(Serial, Subject, PubKey, Extensions) ->
309 +     #'OTPTBSCertificate'{
310 +        version = v3,
311 +        serialNumber = Serial,
312 +        signature = #'SignatureAlgorithm'{
313 +                      algorithm = ?'ecdsa-with-SHA256',
314 +                      parameters = asn1_NOVALUE},
315 +        issuer = Subject,
316 +        validity = #'Validity'{
317 +                     notBefore = {utcTime, "240101000000Z"},
318 +                     notAfter = {utcTime, "340101000000Z"}},
319 +        subject = Subject,
320 +        subjectPublicKeyInfo = ec_subject_pubkey_info(PubKey),
321 +        extensions = Extensions}.
322 +
323 + build_ocsp_response(IssuerName, IssuerKey, NonceExt, SignKey) ->
324 +     EncodedName = pubkey_cert_records:transform(IssuerName, encode),
325 +     IssuerNameHash = crypto:hash(sha, public_key:der_encode('Name',
    EncodedName)),
326 +     IssuerKeyHash = crypto:hash(sha, IssuerKey#'ECPrivateKey'.publicKey),
327 +     ResponseData = #'ResponseData'{
328 +         version = v1,
329 +         responderID = {byName, EncodedName},
330 +         producedAt = "202501010000000Z",
331 +         responses =
332 +             [#'SingleResponse'{
333 +                 certID = #'CertID'{
334 +                     hashAlgorithm = #'AlgorithmIdentifier'{
```

```
335 +                          algorithm = ?'id-sha1',
336 +                          parameters = <<5,0>>},
337 +                      issuerNameHash = IssuerNameHash,
338 +                      issuerKeyHash = IssuerKeyHash,
339 +                      serialNumber = 100},
340 +                certStatus = {good, 'NULL'},
341 +                thisUpdate = "20250101000000Z",
342 +                nextUpdate = asn1_NOVALUE,
343 +                singleExtensions = asn1_NOVALUE}],
344 +         responseExtensions =
345 +             [#'Extension'{
346 +                 extnID = ?'id-pkix-ocsp-nonce',
347 +                 critical = false,
348 +                 extnValue = NonceExt}]},
349 +     ResponseDataDer = public_key:der_encode('ResponseData', ResponseData),
350 +     Signature = public_key:sign(ResponseDataDer, sha256, SignKey),
351 +     #'BasicOCSPResponse'{
352 +         tbsResponseData = ResponseData,
353 +         signatureAlgorithm =
354 +             #'AlgorithmIdentifier'{
355 +                 algorithm = ?'ecdsa-with-SHA256',
356 +                 parameters = asn1_NOVALUE},
357 +         signature = Signature,
358 +         certs = asn1_NOVALUE}.
```

Comments 0

Please sign in to comment.