Commit ac7ff52

u3s committed 3 weeks ago · 23 / 27 · 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 (#10873) · OTP-28.4.2

1 parent c8fca7e commit ac7ff52

2 files changed +148 -2 lines changed

Top

Filter files…

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

2 files changed +148 -2 lines changed

Search within code

lib/public_key/src/pubkey_ocsp.erl

```
@@ -231,7 +231,9 @@ is_authorized_responder(CombinedResponderCert =
#cert{otp = ResponderCert},
231 231            %%      issue OCSP responses for that CA (id-kp-OCSPSigning)
232 232            fun() ->
233 233                    public_key:pkix_is_issuer(ResponderCert, IssuerCert) andalso
234     -                                  designated_for_ocsp_signing(ResponderCert)
    234 +                      designated_for_ocsp_signing(ResponderCert) andalso
    235 +                      public_key:pkix_verify(CombinedResponderCert#cert.der,
    236 +                                             get_public_key_rec(IssuerCert))
235 237            end,
236 238        Case3 =
237 239            %% a Trusted Responder whose public key is trusted by the requestor
```

lib/public_key/test/pubkey_ocsp_SUITE.erl

```
@@ -107,7 +107,7 @@
107 107    %% Common Test interface functions -----------------------------------
108 108    %%--------------------------------------------------------------------
109 109    all() ->
110     -      [ocsp_test].
    110 +      [ocsp_test, designated_responder].
111 111
112 112    groups() ->
113 113        [].
@@ -213,3 +213,147 @@ ocsp_test(Config) when is_list(Config) ->
213 213                                    ?ISSUER_CERT,
214 214                                    IsTrustedReponderFun),
215 215        ok.
    216 +
    217 + %%--------------------------------------------------------------------
    218 + designated_responder() ->
    219 +     [{doc, "Test Case2 (designated responder) in is_authorized_responder/3. "
    220 +       "Verifies that a legitimate designated responder cert signed by the CA "
    221 +       "is accepted, and a forged self-signed cert with the same subject DN "
    222 +       "is rejected (GHSA-gxrm-pf64-99xm)."}].
    223 + designated_responder(Config) when is_list(Config) ->
    224 +     %% Generate self-signed root CA (issuer = subject). A single CA is
```

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

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

```
302 +              #'Extension'{extnID = ?'id-ce-extKeyUsage',
303 +                           critical = false,
304 +                           extnValue = [?'id-kp-OCSPSigning']}]).
305 +
306 + sign_responder_cert(Serial, Issuer, PubKey, SigningKey) ->
307 +     Der = public_key:pkix_sign(ocsp_responder_tbs(Serial, Issuer, PubKey),
    SigningKey),
308 +     {Der, public_key:pkix_decode_cert(Der, otp)}.
309 +
310 + make_tbs(Serial, Subject, PubKey, Extensions) ->
311 +     #'OTPTBSCertificate'{
312 +        version = v3,
313 +        serialNumber = Serial,
314 +        signature = #'SignatureAlgorithm'{
315 +                      algorithm = ?'ecdsa-with-SHA256',
316 +                      parameters = asn1_NOVALUE},
317 +        issuer = Subject,
318 +        validity = #'Validity'{
319 +                     notBefore = {utcTime, "240101000000Z"},
320 +                     notAfter = {utcTime, "340101000000Z"}},
321 +        subject = Subject,
322 +        subjectPublicKeyInfo = ec_subject_pubkey_info(PubKey),
323 +        extensions = Extensions}.
324 +
325 + build_ocsp_response(IssuerName, IssuerKey, NonceExt, SignKey) ->
326 +     IssuerNameHash = crypto:hash(sha, public_key:der_encode('Name',
    IssuerName)),
327 +     IssuerKeyHash = crypto:hash(sha, IssuerKey#'ECPrivateKey'.publicKey),
328 +     ResponseData = #'ResponseData'{
329 +         version = v1,
330 +         responderID = {byName, IssuerName},
331 +         producedAt = "202501010000000Z",
332 +         responses =
333 +             [#'SingleResponse'{
334 +                 certID = #'CertID'{
335 +                     hashAlgorithm = #'CertID_hashAlgorithm'{
336 +                         algorithm = ?'id-sha1',
337 +                         parameters = {asn1_OPENTYPE, <<5,0>>}},
338 +                     issuerNameHash = IssuerNameHash,
339 +                     issuerKeyHash = IssuerKeyHash,
```

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

Comments 0

Please sign in to comment.