The ISP Column
A monthly column on things Internet

April 2026
Geoff Huston

Revocation

This is not the first time I've looked at Domain Name Certificate revocation in these blog articles, first in 2020 and again in 2022. It's an evolving story and there have been a number of recent changes by the Certificate Authority and Browser Forum (CAB Forum) and the largest of the Certification Authorities, Let's Encrypt, which have a very fundamental impact on the way in which we use (and trust) these certificates, so I thought it would be useful to look at the topic of certificate revocation once more in the light of these changes.

A Public Key Infrastructure (PKI) is a system designed to support the use of public/private keyed digital signatures through a system of structured transitive trust. The objective of a PKI is to enable trusted communications between parties who may have never directly met and may not necessarily even know each other at all. A PKI normally uses X.509 public key certificates, which are digital objects that contain a verifiable attestation that the certificate issuer has satisfied itself, using application procedures documented in its Certificate Practice Statement, that the holder of a given public/private key pair has met certain criteria, as specified by the certificate issuer. The certificate issuer then publishes a certificate that associates a subject name (such as an individual's details for identity certificates or a DNS name for a domain name certificates) with the holder's public key, and then attaches a digital signature to this object, generated using the certificate issuer's private key. This act is both verifiable by any party that has knowledge of the issuer's public key and cannot be subsequently repudiated by the issuer.

These X.509 public key certificates are used for many purposes, as they support authenticity, verifiability, and attribution, and can play a role in setting up an encrypted session. For example, if an individual signs a digital document with their certified private key, then anyone who refers to the associated public key identity certificate can validate (*verifiability*) that it was this particular individual who signed the document (*attribution*), and the document is unaltered (*authenticity*), as long as they are prepared to trust the integrity of the certificate issuance practices of the certificate issuer. If a client uses a server's public key as part of the process of setting up a session key, the client and server can exchange encrypted messages that can only be deciphered by each other (*encryption)*, such as in the use of keys by Transport Layer Security (TLS).

In the context of the Internet, we see very widespread use of domain name public key certificates in the web, where clients can access the servers' published content and online services using Transport Layer Security (TLS), as seen in the use of HTTP**S** URLs. The use of the web PKI allows a client to *validate* the authenticity of the remote server's identity and *encrypt* the ensuing access session, ensuring that the transaction cannot be eavesdropped by third parties, that the contents of the transaction are not altered in any way, and that the server cannot repudiate the transaction. Obviously, this level of trust is vital for the Internet, and this implies that these days TLS is one of the foundational protocols for the Internet. If I had to nominate just a handful of critical common Internet protocols, I'd have to say that today TLS ranks up there with the DNS and HTTP. This means that the associated Web PKI's system of domain name certification is also critically important for the Internet. I'd got even further and claim that in the

pastiche of various security mechanisms at use on the Internet, including DNSSEC and RPKI, we rely exclusively on these domain name certificates to provide the necessary assurance that we are not being hoodwinked and misled.

X509 Public Key Certificates enable *transitive trust* (on the basis that if Alice trusts every action of Bob's and Bob trusts Carol, then Alice can trust Carol.) However, trust is never eternal, and neither should the implicit trust described in a certificate be regarded as eternal. An X.509 Public Key Certificate has two date fields, `notBefore` and `notAfter`, specifying the time span when the certificate can be used. (Although it must be noted RFC 5280 does include the specification of a 'forever' `notAfter` date value if eternal trust really is the intended outcome! But using this value would be an incredibly foolhardy action!) The usual practice of certification management is to set the `notBefore` field to the date of certificate issuance and setting the `notAfter` field to some period specified by a contract or agreement between the certificate issuer and the subject. The subject is expected to apply for a new certificate before the expiration of the current certificate if they wish to continue to be certified beyond the `notAfter` date. Prior to Let's Encrypt's entry into the Domain Name certification market, typical certification periods were one or two years. Let's Encrypt is now a major player, and their certificates have a 90-day validity period, so the average validity period for all these Domain Name certificates has come down. There is a change coming in these Let's Encrypt certificates next month, in May 2026, but we'll discuss this further on in this article.

There is always the case that the unexpected happens, and X.509 certificates are no exception. There are circumstances where the certificate should be marked as unusable immediately, which is before the `notAfter` expiration time. The private key may have been compromised, or the certificate was issued in error, or the subject is no longer undertaking the activity or service for which it was certified, or a myriad of other reasons. The subject may want a new certificate issued before it expires and retire the old certificate as soon as its replacement is brought into service. Or the certificate issuer may have been compromised. These things happen.

Revocation

How can a certificate be marked as untrustable, or be *revoked*, before its scheduled expiration time?

The certificate issuer should remove the revoked certificate from its online publication point, so that the revoked certificate is no longer available for upload and use by relying parties.

But that's not good enough. There are many reasons for certificate users to gather and locally store copies of such certificates. My web server, for example, has a local copy of the server name's certificate which it uses to support secure sessions. When a client starts a TLS session with this server how is the client meant to know that the certificate that is being used to set up this secure session has been revoked? Somehow, the client needs some additional assurance that the certificate is still trustworthy.

Before looking at revocation mechanisms, I should note one rather esoteric point about revocation, namely its irrevocability.

Revoking a certificate causes the Certification Authority (CA) to create a metadata record about the unusable status of the certificate in a special list of revoked certificates, called imaginatively a *Certificate Revocation List* (CRL), digitally signed by the CA to attest to its authenticity.

The CA does not issue an altered replacement certificate that records its revoked status within the certificate itself. The entry in the CRL is the only record that the certificate must not be used. A CA could, in theory, subsequently remove the listing of the revoked status of the certificate in the CRL, and because the certificate itself has not been altered in any way, it could restore the certificate back into its publication point. In

effect, the published certificate state after this removal of the revocation metadata would the same as it was prior to the revocation action. The certificate has been unrevoked.

In practice, this would be a truly terrible idea and CA's must never attempt this! The only permitted way to signal the re-instatement of trust is by issuing a new certificate for the same subject with a new serial number. It would be highly prudent for the subject to use a new public/private key pair in a case.

A conventional PKI response to manage revocation is for the Certification Authority (CA) to regularly publish a signed Certificate Revocation List (CRL). A CRL contains a list of the certificate serial numbers of all unexpired revoked certificates that have been issued by the same CA as the issuer of the CRL and the time of revocation. A CRL also contains the date of issuance of this CRL and the anticipated date of the next CRL to be published by this issuer. CRLs are signed documents, signed with the private key of the CA. A standard profile for CRLs for use on the Internet is published in RFC 5280. An example CRL is shown in Figure 1.

```
$ wget http://e8.c.lencr.org/111.crl

$ openssl crl -inform DER -text -noout -in 111.crl
Certificate Revocation List (CRL):
    Version 2 (0x1)
    Signature Algorithm: ecdsa-with-SHA384
    Issuer: C=US, O=Let's Encrypt, CN=E8
    Last Update: Apr 22 01:42:43 2026 GMT
    Next Update: May  1 01:42:42 2026 GMT
    CRL extensions:
        X509v3 Authority Key Identifier:
            8F:0D:13:A2:F6:2E:7E:D1:50:6C:33:18:38:5D:59:8E:23:72:91:CA
        X509v3 CRL Number:
            1776822163859760709
        X509v3 Issuing Distribution Point: critical
            Full Name:
              URI:http://e8.c.lencr.org/111.crl
            Only User Certificates

Revoked Certificates:
    Serial Number: 05F5F9651D5AA120AFFB590755A2C51979EE
        Revocation Date: Jan 21 01:06:17 2026 GMT
    Serial Number: 063914206A1E038705B852D44A5051E3396E
        Revocation Date: Jan 21 04:33:49 2026 GMT
    [repeated 1,560 times]

Signature Algorithm: ecdsa-with-SHA384
Signature Value:
    30:66:02:31:00:b7:59:6a:b2:61:fc:d5:57:92:1e:5c:08:86:
    11:77:7a:c0:91:07:3c:b3:32:62:6b:d4:d8:6b:7a:d9:33:9c:
    ea:a1:e9:5d:a5:75:e7:d0:cb:f4:a1:65:d5:47:94:21:0c:02:
    31:00:86:7b:d7:6b:14:19:7a:8d:ad:2e:ec:d8:e5:ce:83:ad:
    3e:7d:57:84:21:09:38:af:31:90:97:32:74:61:f5:b8:59:1a:
    89:68:ef:66:03:43:fd:12:71:b6:8d:e1:b8:14
```

Figure 1 – A CRL issued by Let's Encrypt

The CRL is intended to be complete, in that at any time at the date specified by the `Last Update` field of the CRL, all unexpired revoked certificates issued by this CA in a given scope are listed in the CRL. If the scope of the CRL is the entire set of certificates issued by this CA, then the corollary is that if an unexpired certificate is not listed in the CRL, then it can be trusted (used as a valid certificate) until the next CRL issuance time.

Relying parties can consider a CRL itself to be valid if the current time is between the CRL time and the CRL's `nextUpdate` time, and the CRL's signature can be validated.

Having a certificate listed in a CRL effectively curtails a certificate's validity, but the action is not necessarily immediate across the broader domain of use. An end user ("relying party") may hold a local copy of an issuer's CRL until the CRL's `Next Update` time, and therefore revocation will not necessarily be visible to relying parties until the next CRL is published and retrieved by relying parties. What this means is that while a CRL can curtail a certificate's lifetime before its scheduled expiration date, the CRL may not necessarily be up to date, and may lag by as much as the CRL's publication interval. In the case of the CRL in Figure 1 there is a 9-day window where a relying party may be unaware that an issued certificate may have been revoked. In practice, a CRL improves the timeliness of certificate currency from years to a week or so. Revocation is still not immediate, but it's an improvement.

How can a client check the revocation status of a certificate that is presented to it when starting a TLS session? The CRL procedure entails:

1. Find the CRL publication URL in the certificate provided as part of the TLS handshake.

2. Retrieve the CRL.

3. Validate that the digital signature was generated by the CA's private key, and create a validation chain to a Trust Anchor.

4. Validate the currency of the CRL with Update Date of the CRL.

5. Look for the Certificate's Serial Number in the CRL.

If the certificate was revoked prior to the CRL issue date, then its serial number will be listed in this CRL. If it cannot be found in the CRL then either the certificate has not been revoked, or, more accurately, it had not been revoked at the time of the CRL creation date.

The more the number of unexpired revoked certificates the larger the CRL will get. The longer the lifetime of issued certificates, the larger the CRL may get. For a large CA the workload associated with CRLs can be significant. Delivering a complete list of all revoked certificates seems to be a case of over-answering, particularly if all the querier wanted to know was the revocation status of a single certificate. In the case of the CRL in Figure 1, the client is given a list of 1,562 revoked certificate serial numbers to check through.

The generation of CRLs is not a mandatory requirement for CAs. A CA may elect to regularly publish CRLs, or may elect to publish CRLs and update them with delta CRLs, or may not publish CRLs at all!

The question is: Do we use CRLs in this manner?

No!

It takes too long to load the CRLs and perform the CRL checking actions during the process of certificate validation. Nobody is willing to pay this time penalty. It's also a totally inefficient design. Why retrieve the entire list of revoked certificates in a CRL when all you want to know is the status of a single certificate? For these reasons CRLs were typically not used by end clients when setting up a TLS session.

Plan B: Online Certificate Status Protocol (OCSP)

OCSP is a refinement to the omnibus style of CRLs. In OCSP a client generates an OCSP request that contains a certificate serial number and sends it to the CA that issued the certificate. The CA responds

with a signed certificate status report indicating whether the certificate is good, or whether the certificate has been revoked (Figure 2). The protocol is documented in RFC 6960.

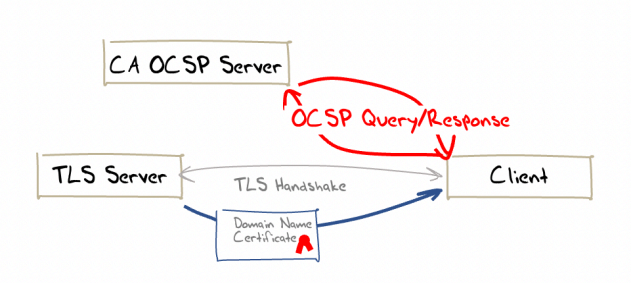


Figure 2 – The OCSP model

An OCSP request is an ASN.1 object, containing one or more certificate serial numbers of certificates that the client wants the CA to check.

An OCSP response is digitally signed by the CA who issued the certificate, or a CA-designated responder. The response includes the identity of the responder, the time of the response, responses for each certificate in the request and optional extensions. The response codes used by OCSP indicate that the certificate is either:

- ***good***, which actually means no certificate with this serial number issued by this CA is revoked. As RFC 6960 explains: "This state does not necessarily mean that the certificate was ever issued or that the time at which the response was produced is within the certificate's validity interval."

- ***revoked***, which indicates an unexpired revoked certificate, but may also indicate that this CA has not issued a certificate with this serial number.

- ***unknown***, which indicates that the CA does not recognise this certificate serial number.

An example OCSP response for a revoked certificate is shown in Figure 3.

```
$ openssl ocsp -issuer lets-encrypt-r3-cross-signed.pem.txt -serial \
  0x04F6351FB48399440794386973D8BD9C4095 -url http://r3.o.lencr.org -text
OCSP Request Data:
    Version: 1 (0x0)
    Requestor List:
        Certificate ID:
          Hash Algorithm: sha1
          Issuer Name Hash: 48DAC9A0FB2BD32D4FF0DE68D2F567B735F9B3C4
          Issuer Key Hash: 142EB317B75856CBAE500940E61FAF9D8B14C2C6
          Serial Number: 04F6351FB48399440794386973D8BD9C4095
    Request Extensions:
        OCSP Nonce:
            0410E32D127F0CAE78737814324013BF904C
OCSP Response Data:
    OCSP Response Status: successful (0x0)
    Response Type: Basic OCSP Response
    Version: 1 (0x0)
    Responder Id: C = US, O = Let's Encrypt, CN = R3
    Produced At: Mar 13 16:22:00 2022 GMT
    Responses:
    Certificate ID:
      Hash Algorithm: sha1
      Issuer Name Hash: 48DAC9A0FB2BD32D4FF0DE68D2F567B735F9B3C4
      Issuer Key Hash: 142EB317B75856CBAE500940E61FAF9D8B14C2C6
      Serial Number: 04F6351FB48399440794386973D8BD9C4095
    Cert Status: revoked
    Revocation Time: Mar  8 16:22:24 2022 GMT
    This Update: Mar 13 16:00:00 2022 GMT
    Next Update: Mar 20 15:59:58 2022 GMT
    Revocation Time: Mar  8 16:22:24 2022 GMT
    Signature Algorithm: sha256WithRSAEncryption
```

```
73:88:aa:a1:1d:ff:5f:5b:eb:30:9d:43:ca:76:b8:3e:70:9f:
d7:d2:3f:6e:dd:dd:fb:69:0f:16:e4:b3:3c:f3:75:d3:6b:37:
f4:fa:cc:10:15:8c:cf:59:e0:f4:2a:60:d9:4c:5e:b2:df:24:
d9:a1:20:b2:7e:14:d8:d0:03:92:97:9e:be:b3:e5:4e:c9:6c:
db:96:8c:ff:6c:c7:4f:cf:88:35:bb:52:90:4f:6e:b9:51:70:
f1:51:93:9d:de:b0:91:44:69:12:47:15:b2:18:c3:0d:bd:d5:
af:01:ff:c3:8d:c0:31:94:87:e0:0e:06:18:35:7a:a8:4a:dd:
2a:e4:61:2a:6d:db:6e:9f:87:d4:9f:79:25:17:f5:7a:e3:4d:
7b:44:95:56:d4:ff:b2:38:50:f6:58:7c:d3:97:0c:e6:ab:2c:
f6:2b:9a:55:a8:63:c6:f4:b9:97:2b:21:a2:bf:38:0d:91:e6:
af:64:22:b8:50:b4:e8:70:27:ee:60:0d:fd:96:6e:b2:54:f8:
38:ed:14:31:ca:1e:3f:c3:7f:ae:f5:d3:ff:9b:75:bf:4d:12:
e7:1b:9a:62:a8:d9:c0:a4:8f:48:33:b2:f5:ea:d0:e9:27:e3:
4f:ed:c3:1a:80:3a:1b:94:27:7c:90:56:c3:b3:65:7a:6e:5f:
94:a9:56:79
```

Figure 3 – An OCSP response for a revoked certificate

The extensions in the response may include a *nonce* that cryptographically links a request to the response, preventing replay attacks. It may also include a reference to a CRL. OCSP responses also may include four times:

- `thisUpdate` – the time that the responder generated this status information
- `nextUpdate` – the time by when updated information will be available
- `producedAt` – the time the responder signed this response
- `revocationTime` – the time when the certificate was revoked

These fields allow the client to cache an OCSP response, as the response can be cached until the `nextUpdate` time. OCSP responses can be generated in advance, with the time of the generation of the response indicated by the `producedAt` time.

There are some very serious privacy concerns with OCSP in having the client contact the CA. The CA is aware of the identity of clients using this certificate via the source of the OCSP queries and also aware of when the client is using the certificate. This is a significant privacy leak each time the client needs to access the OCSP data for a certificate.

There are also performance issues with the additional time taken to generate the OCSP request and waiting for the response. This is not necessarily a single request, as a prudent client would check not only the revocation status of the certificate used by the server, but also check the revocation status of all the CA certificates used by the client to assemble the validation chain from a Trust Anchor to this certificate.

It is also worth remembering that this is not necessarily a query as to the *current* revocation status of this certificate. The OCSP response is generally an extract from the CA's current CRL, and it reflects the certificate's revocation status at the time of the creation of the CRL, not the revocation status at the precise time of the OCSP query.

It's still effectively a CRL lookup, but now the lookup is being performed by the CA, not the client.

Who Uses OCSP?

Let's look at a few common browsers and ask whether the browser is willing to successfully complete a TLS connection even when the certificate is revoked. For this test we will not use OCSP Stapling, so the only way that the browser can tell if the certificate has been revoked is via an OCSP query. The results are shown in Table 1.

Browser	OCSP
Chrome	NO
Safari	YES
Firefox	YES

Table 1 – OCSP Support for common Browsers

It appears that Safari and Firefox browsers perform OCSP-based certificate revocation queries, while Chrome does not (and has not since 2012).

If we are going to rely on OCSP to perform certificate revocation, then it seems that this is largely an ineffectual measure, as the dominant browser platform, Chrome, does not perform OCSP checks. What's wrong with OCSP?

The OCSP Scorecard

So, what's good about OCSP? What's not so good? Here's a scorecard from my impressions about OCSP:

- ✓ It's faster than performing a full CRL retrieval and lookup.
- ✗ There's still an additional imposed delay to perform the OCSP query and response.
- ✗ Both on-demand CRLs and OCSP require CAs' service points to be available. In the case of CRLs it is possible to rely on local caching of CRLs and to some limited extent alleviate this availability requirement, but for on-demand OCSP there is simply no way out. The OCSP server has to be available, and available at a speed commensurate with the tight time constraints of a TLS session setup in an application. This is a major change to the CA's operational model, which was not directly involved when a client wanted to validate a certificate, as long as the client was not going to perform a revocation check.

> Let's Encrypt reported that at the start of 2025 their OSCP servers handled some 340 billion OCSP requests per month, or an average of 140,000 requests per second.
> *https://letsencrypt.org/2025/08/06/ocsp-service-has-reached-end-of-life*

- ✗ What should the client do if the OCSP request is not answered? Proceeding in any case is a *soft-fail* that exposes the client to the potential perils that OCSP was intended to avoid. A cautious denial, or *hard-fail* may simply generate unnecessary blocking. The OCSP service point becomes a single point of potential failure and in a world of various forms of constrained and unreliable access adding yet another point of potential service failure is hardly a sensible move. A *hard-fail* framework also runs the risk of making these OCSP servers yet another point of vulnerability in a hostile denial of service scenario. It appears that many client applications and operating system service libraries use *soft-fail* if an OCSP query elicits no response. This has consequences, as an attacker who is on the path between the user and the CA may see the unencrypted OCSP query and simply block it. The client's certificate validation function will then *soft-fail* and regard the revoked certificate as valid. The client is then placed into the vulnerable position of trusting a revoked certificate anyway.
- ✗ OCSP is also a significant privacy leak. With OCSP the CA is now aware of which clients use a certificate to complete a TLS connection and when. In an Internet that appears to be solidly based on surveillance capitalism as its dominant business model this places a significant body of highly valuable information about current user behaviour into the hands of a new set of actors. It's quite challenging to imagine any scenario where this information stream would not be exploited in one manner or another by the CA.

So OCSP is not looking all that good.

> "That's why I claim that online revocation checking is useless - because it doesn't stop attacks. Turning it on does nothing but slow things down. You can tell when something is security theater because you need some absurdly specific situation in order for it to be useful."
> Adam Langley,
> *https://www.imperialviolet.org/2014/04/19/revchecking.html*

Plan C: Stapled OCSP Responses

One response to these concerns related to on-demand certificate status checking is to package the OCSP response with the certificate, in a framework called *Stapled OCSP* (described in RFC 6066 and RFC 6961). As the OCSP response is already signed and dated by the CA, and the server knows which certificate it has passed to the client, it also knows what OCSP requests the client will make to the CA in order to confirm that the certificate has not been revoked by the CA. The server uses TLS stapling to attach the OCSP response to the TLS material used in the handshake. It can do this for the OCSP response of other CA certificates that will form the validation chain used by the client to validate the certificate.

To ensure that a middle-attack does not attempt to strip out the OCSP response, we can also add a flag to the signed certificate to indicate that the OCSP response must always be used when using this certificate, through the OCSP Must Staple attribute in the certificate, which makes the signal to use Stapled OCSP tamper-resistant.

> "If we want a scalable solution to the revocation problem then it's probably going to come in the form of short-lived certificates or something like OCSP Must Staple. Recall that the original problem stems from the fact that certificates are valid for years. If they were only valid for days then revocation would take care of itself."
> Adam Langley,
> https://www.imperialviolet.org/2014/04/19/revchecking.html

What stapled OCSP responses are trying to achieve is to offload the OCSP check from the client to the server. This is shown in Figure 4.

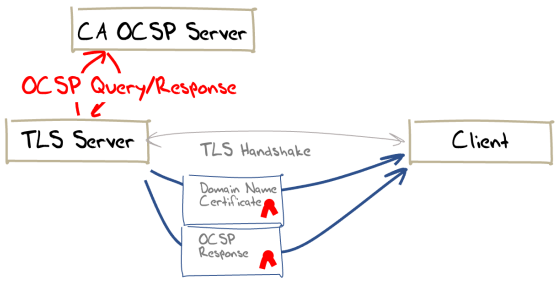


Figure 4 – The Stapled OCSP model

Let's compare the OCSP scorecard against this stapled OCSP approach.

The Stapled OCSP Scorecard

Stapled OCSP is:

- ✓ It's faster than performing a full CRL retrieval and lookup.
- ✓ It's faster than OCSP as there is no additional query and response delay for the client.
- ✓ It reduces the OCSP query load on the CA, as the OCSP response can be used for the entire OCSP validity time, rather than having each client query the CA upon every use of this certificate.
- ✓ The CA gains no knowledge of the identity of the client or the times of use of this certificate.
- ✓ Persistent unavailability of the CA's OCSP server can cause the server to *hard fail* as it has no current OCSP response to staple to the TLS material.

We can also perform the same tests on a range of commonly used platforms and browsers to see if they will correctly detect the attempt to use a revoked certificate to set up a TLS session. This is shown in Table 3.

Browser	**Stapled OCSP**
Chrome	NO
Safari	YES
Firefox	YES

Table 3 – Stapled OCSP Support for common Browsers

Chrome and Revocation

What's going on with Chrome? Almost a decade ago Chrome announced that it was going to use a different form of revocation checks (). The approach, termed *CRLsets*, involves Chrome using Googlebots to crawl across CRLs and collecting a set of current revocations. Details are a little sparse, but many reports claim that the list of revoked certificates is then trimmed, and "unimportant" revocations are stripped out. The resultant list is sent to instances of the Chrome browser.

The article that described this approach also included the comment: "For the curious, there is a tool for fetching and parsing Chrome's list of revoked certificates at https://github.com/agl/crlset-tools." After retrieving and running this tool I was surprised to see a total of 1,081 revoked certificate serial numbers in this list. This seems oddly low. The CA Lets Encrypt lists some 1062 revoked certificates, and its by no means the only CA out there. If this tool is the same as that used within Chrome, then it would seem that the trimming being carried out here is quite extensive.

> I must admit that this strikes me as inconsistent on Chrome's part. On the one hand Chrome (and Google) have been part of the pressure to transform encryption from an expensive luxury to a freely available commodity. Chrome complains stridently when it is directed to sites using port 80 without channel encryption, and when a certificate is signed by a private CA, Chrome will refuse to connect on the basis of its assumed insecurity. Yet at the same time Chrome is saying that this enthusiasm for security and encryption does not extend to support for revocation. Revocation support, timely or otherwise, is only for some chosen few in Chrome.

Now if Chrome had no significant market share in the Internet browser space, then Chrome's position would be insignificant. But this browser appears to have some 80% of the market and Chrome's position is the de facto Internet position. And in Chrome's case it seems that Chrome is all too willing to ignore revocation for all but the "most important" services.

What's the point of Stapled OCSP?

Good question!

If the certificate is not revoked at the time of the CRL generation, then, as already noted, the OCSP reported status is *good,* which adds nothing to the information already contained in the certificate. In this case stapled OCSP is adding nothing.

If the OCSP reported certificate status is *revoked* then the CA is saying that this certificate should not be used. If that is the case, then why should the server convey the certificate and the OCSP status to the client and defer to the client on the decision not to proceed with the TLS connection? Why shouldn't the server simply terminate the TLS connection immediately itself? In other words, why should the server pass a revoked certificate to the client? In this case stapled OCSP is a long way around to reach the inevitable outcome of a correctly failed TLS connection attempt.

So why bother?

Is Revocation worth the effort?

Another good question!

A compromised private key should not be accepted. An attacker might use a compromised private key to impersonate a site, and this vulnerability needs to be prevented to ensure that users can use services over the network with trust in their integrity and security. The way to stop a compromised key from being accepted is to disseminate the information that the key is no longer trustable, and this is achieved by revoking the public key certificate. Or so goes the conventional thinking on X.509 certificates and revocation.

But we are having some problems in taking this theory and creating practical implementations of revocation.

Certificate Revocation Lists only really work efficiently when nobody revokes certificates! Otherwise, we have a scaling problem with clients incurring the considerable overheads of very large CRLs being passed around unnecessarily, as the client only really wanted to query the status of a single issued certificate in any single transaction.

OCSP avoids the scaling problem of large CRLs, but this exacerbates the issue of privacy, as the CA's OCSP server is then aware then aware when clients access individual services that have been certified by this CA.

We can address this by having the server retrieve the OCSP entry and staple it into the TLS exchange, but at that point we arrive at the obvious conclusion that if the server is aware that the certificate has been revoked by the CA, then the server should not complete the TLS exchange at all. However, with the must staple attribute in the certificate the potential attacker is frustrated when the certificate is presented to the user.

But there is a little more going on here.

There is the issue of currency and timeliness. If the point of this entire certificate architecture is to inform the user that the location that they have reached is, or is not, the location that they intended to reach, then why is it useful at all if it can't inform the user that the certificate that is being used is not to be trusted **right now**?

If the best that CRLs, OCSP, Stapled OCSP and even Chrome's CRLsets can inform you is the trust status of a certificate **at some time in the past**, then why is this any different from the certificate itself? A certificate claims that the subject met the CA's criteria to issue a certificate at some point in the past. If the entire purpose of these revocation notification mechanisms is only to reduce the "trust window" of a certificate and report on the status of a certificate at a time closer to the present, then why not just use certificates with a more constrained trust window and avoid revocation completely?

This seems to be a challenging conclusion. The problem with operating with certificates that provide a highly constrained trust window of hours, or at most days, is that the current CA infrastructure was not designed to behave in this way. The certificate infrastructure that is built around assumptions that certificates with a trust window of years probably could not cope with such a dramatically reduced certificates and the associated increased intensity of certificate issuance. Instead, we find ourselves in a situation that has the incompatible attributes of long-lived certificates and non-functional revocation mechanisms. If certificates are incapable of informing a client in real time that they are about to be drawn into misplaced trust, then what exactly are these certificates good for anyway? The entire objective here was to answer the simple question: "Is the service that I am about to connect to the service that I intended to connect to?" It is not a question about the situation a week ago, but a question about the current status of the certificate.

If we want to use public/private keys pairs to underpin security on the Internet using a conventional certification framework then are we necessarily forced to use X.509 certificates and just accept the shortcomings of no robust form of timely certificate revocation?

The answer to that question is: "No, not necessarily." For example, DNSSEC, which uses public/private keys, does not rely on revocation capability, yet still manages to provide timely information. In DNSSEC the zone administrator can sign the zone with a new key and rely on the DNS cache management directives to flush out the old key values from the various DNS caches. (Yes, that's a pretty gross simplification of key rollover, but the underlying design is that the end clients of the DNS regularly refresh locally cached information against the original authoritative source of information. This implies that "stale" information can be flushed from the DNS within the time scale of the cache retention time.

The DNS typically uses cache retention timers (TTLs) of hours or days, compared to the use of months, years and even multiple years in Web PKI certificates. That means that the window of vulnerability in DNSSEC from a compromised key is far shorter than that of the Web PKI, and perhaps this is the major reason why revocation is a far bigger issue in Web PKI certificates than it is in DNSSEC.

There are other issues with long lived certificates and only partial support for revocation, in that it is more challenging to defend against substitution attacks that exploit compromised keys. An attacker can use the compromised private key to generate new credentials that would permit the attacker to re-certify the compromised service with the attacker's keys. By the time the original key compromise is detected the problem now is that the attacker is using different keys and a different certificate, and the onus is now placed on the original service owner to convince the replacement-issuer CA that the replacement certificate was incorrectly issued and get that fake certificate revoked. An agile attacker could repeat this re-certification process any number of times, further frustrating the efforts to remove these fraudulently obtained certificates from the PKI.

Maybe the problem with revocation is best solved by avoiding long-lived certificates in the first place. As Google's Adam Langley pointed out some years ago:

> A much better solution would be for certificates to only be valid for a few days and to forget about revocation altogether. This doesn't mean that the private key needs to change every few days, just the certificate. And the certificate is public data, so servers could just download their refreshed certificate over HTTP periodically and automatically (like OCSP stapling). Clients wouldn't have to perform revocation checks (which are very complex and slow), CAs wouldn't have to pay for massive, DDoS proof serving capacity and revocation would actually work. If the CA went down for six hours, nobody cares. Only if the CA is down for days is there a problem. If you want to "revoke" a certificate, just stop renewing it.
> https://www.imperialviolet.org/2011/03/18/revocation.html

Has anyone done this in the 15 years since this was written? Not to my knowledge.

Are there other approaches to certificate revocation?

Mozilla has revived examination of CRL based approaches, addressing the issue of the size of the CRL. Their approach is to use Bloom Filters to compress the effective size of the CRL, using a technique they call CRLite (https://blog.mozilla.org/security/2020/01/09/crlite-part-2-end-to-end-design/). With this approach they have shown a reduction on CRL data from a list of all enrolled and unexpired certificate serial numbers from 6.7G to a filter of just 1.3M. Their approach is to use this technique on the larger CAs and fall back to OCSP where CRL data has not been set up as a filter. The approach attempts to strip out additional delays in on-demand OCSP and not rely on the piecemeal implementation of must staple server-side OCSP support.

As always, whatever the problem might be, the answer is "just use the DNS," and OCSP is no exception. An internet draft (from 2017) proposes OCSP as a DNS resource record type (draft-pala-odin-03.txt). The OCSP query is encoded into the DNS query name, which itself is packaged in the certificate's `AuthorityInfoAccess` (AIA) extension. For example, a query for the OCSP RR type with a query name of `123456.ca1.example.com` would respond with the OCSP status of the certificate with serial number 123456 issued by this CA. A prudent implementation would also require this DNS zone to be DNSSEC-signed, providing assured presence, authenticity and currency of the DNS response.

It's an interesting idea, but it managed to get nowhere in the IETF. Some of the security folk in the IETF appears to have a pronounced antipathy to any approach that uses the DNS, and many good suggestions intended to improve the landscape of certificates on the Internet have been arbitrarily discarded as a result of this pronounced (and I would claim unfounded) anti-DNS bias in the IETF's security realms. The concept of using the DNS to reveall the current OCSP status illustrates the point that the DNS is adequately capable of delivering timely information in a scalable fashion, while the X.509 certificate infrastructure is simply incapable of performing at the scale, efficiency and speed that we require of it.

Let's remind ourselves of where we are.

CA's issue long-lived certificates and that means that when a key pair is compromised, this vulnerability may persist for years. If the underlying key has been compromised or any other event has occurred which requires that the certificate be annulled, then we'd like some mechanism that works at a faster speed than just waiting for the certificate's expiry time. Certificate Revocation Lists provide such a mechanism, but these lists can become large, and they pose issues in delivering this high volume of data on a just-in-case basis to clients. We turned to use OCSP, so we could query the revocation status of a single certificate but because of doubts as to the resilience of OCSP and the poor scaling properties, which shifted the burden to the CA's OCSP server, the client systems turned to a *fail-safe* approach which allows invalid certificates to continue to be trusted, while some client applications (such as Chrome) abandoned the use of OCSP completely.

Reducing the Validity Period of Certificates

Let's Encrypt is the highest volume CA in today's Internet. Whatever Let's Encrypt does in certificate management sets the momentum for the entire certificate framework. When Let's Encrypt announced that it was reducing the certificate lifetimes of the certificates that they issued from 90 days to 45 days it was not an isolated action. As Let's Encrypt noted: "This change is being made along with the rest of the

industry, as required by the CA/Browser Forum Baseline Requirements, which set the technical requirements that we must follow. All publicly-trusted Certificate Authorities like Let's Encrypt will be making similar changes. Reducing how long certificates are valid for helps improve the security of the internet, by limiting the scope of compromise, and making certificate revocation technologies more efficient."

I must admit I find the rationale for this claim that a reduced certificate lifetime improves the security of the Internet somewhat hard to follow. These days compromise incidents are executed in a matter of minutes, not months, so dropping a certificate's lifetime from 3 months to 1 ½ months appears to be an inadequate response. If the goal is to reduce the window of opportunity of exploitation to something that would impact hostile exploitation, then 45 days is pretty much the equivalent of 90 days, and 7 days is probably equivalent to 45 days. Is there a "right" period of certificate lifetime?

For more than a decade the certificate landscape has been highly fragmented. Safari and Firefox browsers supported OCSP and Stapled OCSP, while Chrome only supported Stapled OCSP. The CA picture was similarly fragmented, where some CAs supported only OCSP (including Let's Encrypt) while others supported only CRLs. What a mess! No wonder the claim that "revocation is broken" has been a widely accepted truism for this certificate infrastructure.

These days it seems that OCSP is heading into disuse. As Let's Encrypt has pointed out, the burden for supporting OCSP is pushed back to the CA, not the client, and for large CAs (such as Let's Encrypt) the demands of maintaining a fast and accurate OCSP responder can be forbidding even in an age of abundant computing, storage and communications. But that's not all. If you look at the date fields in the example OCSP response in Figure 5 you will see a validity period spanning 7 days. So, there is still a significant window between notifying a CA that a certificate should be revoked and other relying parties having their previously cached "good" OCSP responses expiring. So even with OCSP there is a lag in the system between the act of revocation and the time that other parties are informed that the certificate is revoked. We've already mentioned the horrendous privacy leak where every client is informing the CA's OCSP server of their identity and their certificate use history, and the OCSP servers are a point of potential vulnerability in service delivery.

The direction that the CAB forum is evidently taking is to revert back to CRLs, and use a shorted certificate lifetime. It is attempting to push the revocation checking workload back to the browser by working on CRLsets and CRLlite. The thinking is that with a better implementation of CRL storage and lookup in the browsers, and with reduced certificate lifetimes, we can make CRL lookup great again!

Will this address the inherent lag between revocation and the updating of a browser's CRLset? Not really. With all this effort certificate revocation still remains a case of same week service in a nanosecond wor

The DNS, DNSSEC and DANE

There is no panacea here, and every approach to certificate revocation represents some level of compromise.

We can live with long-lived certificates with high enrolment costs, but only if we can support a robust and fast revocation mechanisms, which to date has been an elusive goal. CRLs and OCSP are not instant responses, but they can drastically reduce the timeframe of vulnerability arising from a compromised private key, even though there are some clear issues with robustness with both CRLs and various flavours of OCSP.

We can head down the path forged by Let's Encrypt and only use short lived certificates generated by highly automated processes (where "short" is still a somewhat lengthy 45 days!). Given their short lifetime revocation is not as big an issue, and it's possible to contemplate even shorter certificate lifetimes. If revocation lists are refreshed at one-week intervals, then a one-week certificate could simply be allowed to expire as revocation would not substantially alter the situation for relying parties.

But if we head down this path of short-lived certificates, then why do we need to bother with the X.509 wrapper at all? Why don't we take the approach of packaging OCSP responses in the DNS one step further and dispense with the X.509 packaging completely and place the entire public key infrastructure into the DNS?

In the case of the DNS, all data items provided by the authoritative server have a "*Time To Live*" (TTL), which governs the elapsed time that the data item can be locally stored and (re)used from the local cache before it should be discarded and a new copy of the data retrieved from the authoritative server. There is no implication that the DNS data has necessarily changed in this TTL interval, and for a lot of the DNS it's the case that nothing has changed at all, but it governs the maximum time that an item of information can exist in a local cache without re-syncing this information with that published by the authoritative server. The analogous attribute of a certificate is the `NotAfter` field, that limits the period of validity of the key pair that is being described in this certificate. If you changed the semantic definition of this `NotAfter` field be the time when this instance of the certificate is no longer to be used and the client should request a new copy of the certificate from the CA's publication point, then you have a defined a certificate use behaviour that is functionally equivalent to that of a DNS resource record with its TTL.

At this point why not just use DANE (RFC6698) and store the public keys in the DNS and rely on DNSSEC to provide the necessary authenticity, using DNS TTL settings to control cached lifetime of the public key? With a combination of DNSSEC Chain Extensions and DNSSEC stapling it is possible to perform a security association handshake by having the server provide the client with the server's public key response, a DNSSEC signature and the associated DNSSEC validation responses without performing any query in the DNS at all. So, just like OCSP stapling, it's possible to use DANE and DNSSEC Chain stapling in TLS. The advantage of a DANE approach is to support a far shorter timeframe of local credential autonomy. Anyone who holds a local copy of these public key credentials is limited by the DNS TTL and is required to refresh the data within the TTL interval. The layered intermediary model of DNS resolution already interposes one or more resolvers between the client and the authoritative publisher of the information, so the privacy issues are not as strident, and if clients are sensitive about this topic DNS privacy measures can provide levels of protection.

Are we done here? Not at all!

I'd like to think that this story is by no means over. We are seeing faster hostile attacks that perform the entire process of infiltration, deception, and data exfiltration in just a few hours rather than days or weeks. The responses we rely on today, including certificate transparency logs and the piecemeal use of OCSP, introduce time lags in the currency of credential data of a scale of days and weeks rather than a preferred timescale of seconds or even faster. While we pay both large volumes of lip service to cyber security and equally large volumes of money in operating all this cybersecurity infrastructure, it seems completely appalling to me that the most basic of defence functions, namely pull out of circulation a potentially compromised domain name service in a time frame of seconds rather than months, is not a measure that is accessible to all of us.

Frustratingly, we have a good idea of what we can do, and do so in a way that is robust and scalable. But we are not doing it. Moving back from stapled OCSP to CRLs is a gigantic leap backwards by a few decades, and wrapping this up in bloom filters is a lot like putting lipstick on a pig! Under all the lipstick, its still a pig!

More and more it appears that the resistance to improve the infrastructure of security on the Internet is based around the entrenched position of the incumbent certification infrastructure operators, and these days has nothing to do with any desire to improve the obvious pitfalls in the current certificate-based measures. It appears that the technology discussion these days is more about self-interest and preserving current roles in the domain name certification framework and little to do with any interest in trying to take tangible steps to improve the resilience and security of the Internet.

The situation points to the uncomfortable conclusion that as far as the security of the Internet is concerned, due to the deliberate retrograde inertia of entrenched certificate operators, we being forced to place completely undue and inappropriate reliance on a security framework that lags the demands of today's Internet by an entire generation. Or two!

Afterword

In the preparation for this article, I asked Let's Encrypt to issue a certificate for the DNS name `revoked.potaroo.net`. This is that certificate:

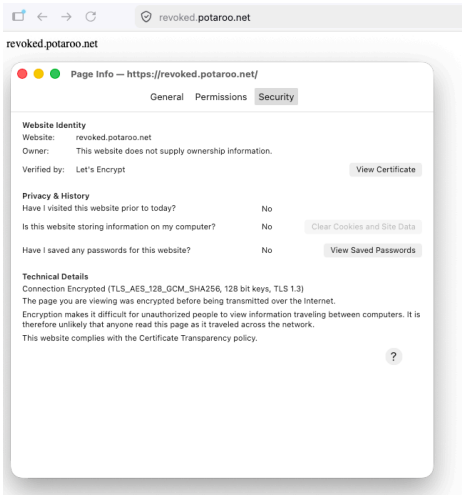
```
$ openssl x509 -in cert.pem -noout -text
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number:
            05:05:c3:14:d6:91:f6:81:9d:35:5b:9d:8f:f9:88:50:42:ee
        Signature Algorithm: ecdsa-with-SHA384
        Issuer: C = US, O = Let's Encrypt, CN = E8
        Validity
            Not Before: Apr 21 03:56:19 2026 GMT
            Not After : Jul 20 03:56:18 2026 GMT
        Subject: CN = revoked.potaroo.net
        Subject Public Key Info:
            Public Key Algorithm: id-ecPublicKey
                Public-Key: (256 bit)
                pub:
                    04:c7:4f:a7:70:1d:3a:9c:38:16:df:7c:82:8c:dd:
                    20:f6:69:7a:21:4f:8a:67:ba:f8:14:85:96:fd:7c:
                    9a:77:40:76:26:c2:42:1f:c9:cc:e7:ff:81:6e:b0:
                    b1:68:c6:af:86:f0:a1:00:95:ca:8c:fe:a3:2a:23:
                    0c:f9:51:10:80
                ASN1 OID: prime256v1
                NIST CURVE: P-256
        X509v3 extensions:
            X509v3 Key Usage: critical
                Digital Signature
            X509v3 Extended Key Usage:
                TLS Web Server Authentication
            X509v3 Basic Constraints: critical
                CA:FALSE
            X509v3 Subject Key Identifier:
                5D:A4:4D:1F:C5:C8:39:29:2E:36:A5:BD:34:AE:1B:1E:B3:E9:B8:3E
            X509v3 Authority Key Identifier:
                8F:0D:13:A2:F6:2E:7E:D1:50:6C:33:18:38:5D:59:8E:23:72:91:CA
            Authority Information Access:
                CA Issuers - URI:http://e8.i.lencr.org/
            X509v3 Subject Alternative Name:
                DNS:*.revoked.potaroo.net, DNS:revoked.potaroo.net
            X509v3 Certificate Policies:
                Policy: 2.23.140.1.2.1
            X509v3 CRL Distribution Points:
                Full Name:
                  URI:http://e8.c.lencr.org/111.crl
            CT Precertificate SCTs:
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : CB:38:F7:15:89:7C:84:A1:44:5F:5B:C1:DD:FB:C9:6E:
                                F2:9A:59:CD:47:0A:69:05:85:B0:CB:14:C3:14:58:E7
                    Timestamp : Apr 21 04:54:49.553 2026 GMT
                    Extensions: none
                    Signature : ecdsa-with-SHA256
                                30:44:02:20:1E:DF:43:40:A2:19:E8:52:69:71:14:8C:
                                2B:DA:87:01:33:43:7A:DA:B5:89:FC:BF:4F:64:DF:E9:
                                39:AF:12:AF:02:20:5C:53:93:CD:8C:BC:12:DF:56:CC:
                                C1:34:7C:A9:53:28:00:06:E5:1B:32:98:8C:87:91:EE:
                                24:B7:22:3A:EA:E0
                Signed Certificate Timestamp:
                    Version   : v1 (0x0)
                    Log ID    : A8:26:CB:E3:0A:C6:35:12:46:53:3F:E0:65:F1:4F:19:
                                D9:6E:19:08:13:C4:1D:D9:6D:79:00:B3:12:3C:55:27
                    Timestamp : Apr 21 04:54:50.497 2026 GMT
                    Extensions: 00:00:05:00:07:B2:57:08
                    Signature : ecdsa-with-SHA256
```

```
                                30:45:02:21:00:BD:58:5C:62:B8:EF:7D:E1:31:60:51:
                                58:F9:02:77:FF:75:B8:53:88:DA:3B:A6:03:FF:FE:DD:
                                B6:7F:33:F5:90:02:20:5D:38:A9:58:80:25:CD:95:8D:
                                A2:3E:2F:10:0D:06:62:FA:30:3E:3D:65:04:AA:62:A4:
                                8A:99:C6:FC:72:6D:57
    Signature Algorithm: ecdsa-with-SHA384
    Signature Value:
        30:65:02:30:35:51:53:68:2d:f8:9a:ed:46:e5:05:d7:14:23:
        14:77:be:92:be:7e:94:14:a6:cd:d9:ce:c7:4c:ee:3f:4b:37:
        4b:1e:0d:eb:ff:63:62:e0:3c:36:2b:47:cb:2e:99:ec:02:31:
        00:f8:3c:b7:cb:2b:2a:1c:c7:f1:0e:16:0d:a0:06:95:4f:36:
        68:b4:bf:71:0c:a3:8e:ac:17:00:0b:c6:39:34:73:86:c4:18:
        2f:20:c0:67:50:65:58:c1:fb:71:f5:c4:4e
```

I then revoked this certificate with the *certbot* tool. To confirm this, I retrieved the current CRL from Let's Encrypt: and confirmed the presence in the CRL:

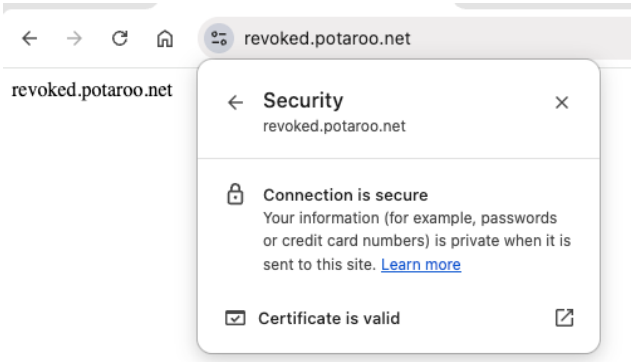
```
$ curl http://e8.c.lencr.org/111.crl --output 111.crl
$ head -16 111.crl
Certificate Revocation List (CRL):
    Version 2 (0x1)
    Signature Algorithm: ecdsa-with-SHA384
    Issuer: C=US, O=Let's Encrypt, CN=E8
    Last Update: Apr 23 05:04:36 2026 GMT
    Next Update: May  2 05:04:35 2026 GMT
    CRL extensions:
        X509v3 Authority Key Identifier:
            8F:0D:13:A2:F6:2E:7E:D1:50:6C:33:18:38:5D:59:8E:23:72:91:CA
        X509v3 CRL Number:
            1776920676311319801
        X509v3 Issuing Distribution Point: critical
            Full Name:
              URI:http://e8.c.lencr.org/111.crl
            Only User Certificates
$ openssl crl -inform DER -text -noout -in 111.crl | grep -i -A 1 \
"0505c314d691f6819d355b9d8ff9885042ee"
    Serial Number: 0505C314D691F6819D355B9D8FF9885042EE
        Revocation Date: Apr 21 07:28:20 2026 GMT
```

So, the certificate was revoked on the 21st April 2026. Don't forget this is a Let's Encrypt certificate, so the CA does not support OCSP. I waited for 48 hours and then used Firefox v148.0.2 to access the URL https://revoked.potaroo.net.

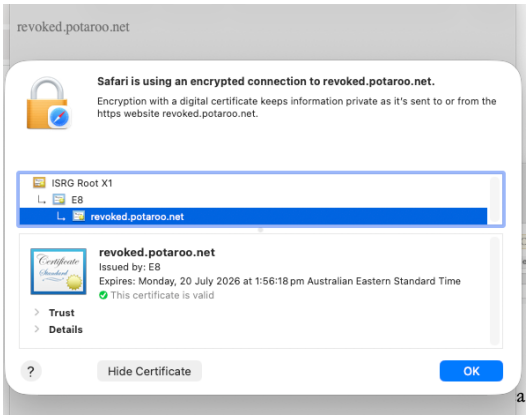


Unfortunately, Firefox has not detected the fact that the certificate used to secure the connection has been revoked! This is badly broken behaviour!

What about Chrome v147.0.7727.117 ?



And Safari v25.4 performs the same way.



It appears that these days browsers have a very selective view of what they store in their local CRLsets, and it's completely unclear to me what I need to do to have the revoked state of this particular certificate loaded into these browsers. All I can conclude from this exercise is that without OCSP things are worse and certificate revocation is indeed a failure!

Considering we all rely on certification as the one and only means of assuring ourselves that we are exchanging data with the service that we intended to communicate with and expose our passwords and secrets, then to observe that certificate compromise can pass unnoticed on today's Internet and we can be duped into communicating with the wrong servers to is a deeply troubling observation.

This is a badly broken situation!

References and Further Reading

RFC 5280, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", D. Cooper et.al, May 2008.
https://tools.ietf.org/html/rfc5280

RFC 6960, "X.509 Internet Public Key Infrastructure Online Certificate Status Protocol – OCSP", S. Santesson, et.al., June 2013.
https://tools.ietf.org/html/rfc6960

RFC6961, "The Transport Layer Security (TLS) Multiple Certificate Status Request Extension", Y. Pettersen, June 2013.
https://tools.ietf.org/html/rfc6961

"X.509v3 Extension: OCSP Stapling Required", Internet Draft, P. Hallam-Baker, April 2013
https://tools.ietf.org/id/draft-hallambaker-muststaple-00.txt

"OCSP over DNS (ODIN)", Internet Draft, M. Pala, November 2017.
https://tools.ietf.org/id/draft-pala-odin-02.txt

"Revocation Doesn't Work", Adam Langley, March 2011
https://www.imperialviolet.org/2011/03/18/revocation.html

"No, don't enable revocation checking", Adam Langley, April 2014
https://www.imperialviolet.org/2014/04/19/revchecking.html

"OCSP Stapling: How CloudFlare Just Made SSL 30% Faster", Matthew Prince, October 2012
https://blog.cloudflare.com/ocsp-stapling-how-cloudflare-just-made-ssl-30/

"High-reliability OCSP stapling and why it matters", Nick Sullivan, July 2017
https://blog.cloudflare.com/high-reliability-ocsp-stapling/

"Revocation is Broken", Scott Helme, July 2017
https://scotthelme.co.uk/revocation-is-broken/

"The Problem with OCSP Stapling and Must Staple and why Certificate Revocation is still broken", Hanno Böck, May 2017
https://blog.hboeck.de/archives/886-The-Problem-with-OCSP-Stapling-and-Must-Staple-and-why-Certificate-Revocation-is-still-broken.html

"Ending OCSP Support in 2025", December 2025
https://letsencrypt.org/2024/12/05/ending-ocsp

"Six-Day and IP Address Certificates are Generally Available", January 2026
https://letsencrypt.org/2026/01/15/6day-and-ip-general-availability

"The Slow Death of OCSP", January 2025
https://www.feistyduck.com/newsletter/issue_121_the_slow_death_of_ocsp

"Certificate Lifetimes to Shrink to just Forty Seven Days", April 2025
https://www.feistyduck.com/newsletter/issue_124_certificate_lifetimes_to_shrink_to_just_forty_seven_days

"Internet PKI to integrate DNSSEC", June 2025
https://www.feistyduck.com/newsletter/issue_126_internet_pki_to_integrate_dnssec

Disclaimer

The above views do not necessarily represent the views or positions of the Asia Pacific Network Information Centre.

Author

Geoff Huston AM, B.Sc., M.Sc., is the Chief Scientist at APNIC, the Regional Internet Registry serving the Asia Pacific region. *www.potaroo.net*