The ISP Column
A monthly column on things Internet

March 2022
Geoff Huston,
João Damas

Revocation

Two years ago, I wrote an article on X.509 certificate revocation (https://www.potaroo.net/ispcol/2020-03/revocation.html). I'd like to report that a lot has happened between then and now, but that's not the case. So why revisit this topic today?

What drew my attention was a tweet earlier this month that reported that the Certification Authority, Thawte, had revoked some certificates that have been previously issued for some Russian bank domains names, and it was speculated that this was part of some form of sanction action (Figure 1,2).

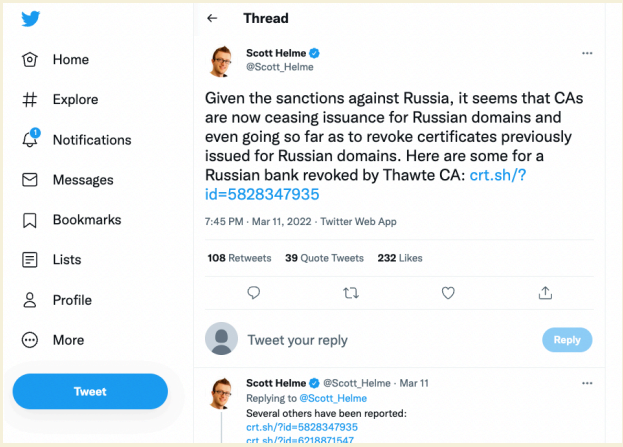

Figure 1 – Report of certificate revocation – Scott Helme, https://twitter.com/scott_helme/status/1502204103132393472 0

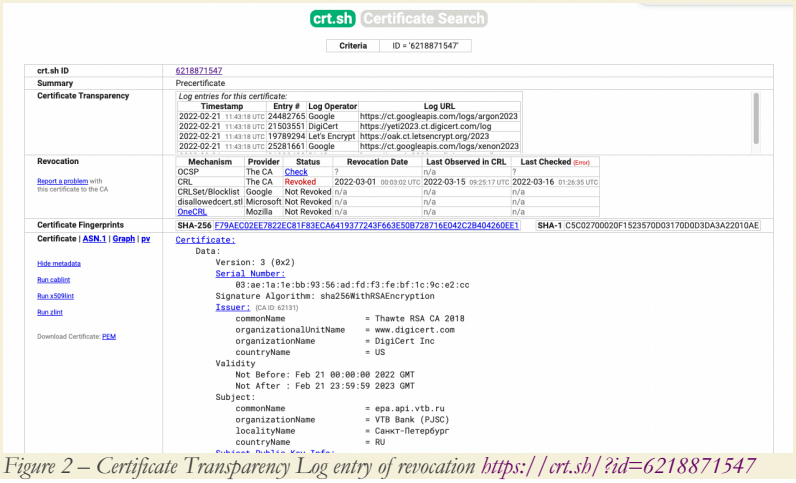

Figure 2 – Certificate Transparency Log entry of revocation https://crt.sh/?id=6218871547

The question I had was whether this action would have any impact at all. Do client applications even check for certificate revocation?

Let's revisit this topic and take a second look at certificate revocation and considered whether it works as intended, or even works at all!

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).*

In the context of the Internet, we see increasing use of PKIs in the web, where web sites publish their content using Transport Layer Security (TLS) (as seen in the use of HTTPS URLs). The use of the web PKI allows a client to validate the authenticity of the remote server's identity and ensures 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, and these X.509 certificates are these days as important as the DNS in terms of the internet's common infrastructure. 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 system of certification is critically important. X.509 certificates come in many forms and their use in a particular context is typically defined within the parameters of a standard profile. A standard profile for X.509 certificates for use in the Internet is published in RFC 5280.

As already noted, 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 SSL 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 these certificates has come down.

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 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 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 unusable, (or be *revoked*) before its scheduled expiration?

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).
>
> 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 is 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 such cases.

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 in the Internet is published in RFC 5280. An example CRL is shown in Figure 3.

```
$ wget http://crl.entrust.net/level1m.crl
$ openssl crl -inform DER -text -noout -in level1m.crl
Certificate Revocation List (CRL):
        Version 2 (0x1)
    Signature Algorithm: sha256WithRSAEncryption
        Issuer: /C=US/O=Entrust, Inc./OU=See www.entrust.net/legal-terms/OU=(c)
            2014 Entrust, Inc. - for authorized use only/CN=Entrust Certification
            Authority - L1M
        Last Update: Mar 14 05:00:49 2022 GMT
        Next Update: Mar 21 05:00:49 2022 GMT
        CRL extensions:
            X509v3 Authority Key Identifier:
                keyid:C3:F7:D0:B5:2A:30:AD:AF:0D:91:21:70:39:54:DD:BC:89:70:C7:3A
            X509v3 CRL Number:
                5765
            2.5.29.60:
                ..20220312050049Z
Revoked Certificates:
    Serial Number: 4D2931EF3C9592A49F43E286B4CEADE7
        Revocation Date: Feb 11 12:22:55 2022 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Superseded
    Serial Number: 6CD01168AC0B47C1C8B643393883DADD
        Revocation Date: Dec 26 01:02:59 2021 GMT
        CRL entry extensions:
            X509v3 CRL Reason Code:
                Key Compromise

    [repeated 5,070 times]

    Signature Algorithm: sha256WithRSAEncryption
         88:5d:54:03:5e:8b:10:00:5a:5e:e9:76:7d:8f:b9:12:13:ae:
         49:6c:cc:65:91:09:07:66:8c:0f:e0:90:be:a6:8b:31:9b:68:
         b3:1f:d8:f9:4a:60:62:49:57:50:fb:25:12:13:e6:59:6d:49:
         97:b4:73:7d:f8:99:95:e9:dd:5c:f3:ad:5f:6c:a7:49:87:9d:
         d3:db:41:1b:5b:8e:a2:f8:9b:ad:69:ed:2f:d0:4f:58:40:1c:
         76:a3:f0:ca:c2:77:47:dc:a1:5b:16:d8:d6:30:52:4a:40:f1:
         a0:bb:6d:92:85:78:7e:35:cd:bc:00:14:30:30:0e:b5:d1:58:
         94:8a:54:f5:f6:c6:b1:67:6d:c7:c0:cc:01:ce:14:29:90:85:
         a9:1d:19:6d:c0:d1:17:00:16:8f:00:8b:33:7f:25:f7:ec:c7:
         7b:35:4c:b9:3a:b0:05:c7:6c:77:41:19:d7:f1:e5:10:9c:04:
         87:ef:cb:e5:99:56:d5:45:fe:2a:e9:3d:4d:9b:f8:e0:e2:e7:
         92:67:ac:ed:74:80:bf:f6:16:80:0b:ab:10:9f:08:51:65:e7:
         a6:7f:10:e0:d5:47:ca:b6:8f:ee:f1:ca:df:81:ae:6a:78:ec:
         f7:c8:c9:f5:f9:92:c7:fa:f1:6c:ad:89:9c:a9:5f:58:9e:24:
         72:6a:94:55
```

Figure 3 – A CRL issued by Entrust

The CRL is intended to be complete, in that at any time within the date specified by 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 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. A relying party may hold a local copy of an issuer's CRL until the CRL's `nextUpdate` time, and therefore revocation will not necessarily be visible to relying parties until the next CRL is published. 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 practice, a CRL improves the timeliness of certificate currency from years to a week or so. It's 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.
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 it's 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. 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. Also, 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! For these reasons CRLs are typically not used by end clients when setting up a TLS session.

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 4). The protocol is documented in RFC 6960.

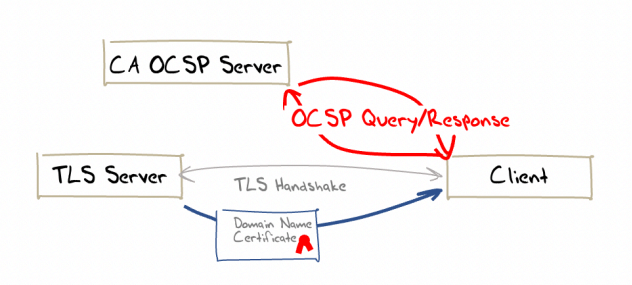

Figure 4 – The OCSP model

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 5.

```
$ 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 5 – 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 privacy concerns with OCSP in having the client contact the CA, in that the CA is then aware of the identity of clients using this certificate via the source of the OCSP request 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 browser and operation system platforms and test 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 of these tests are shown in Table 1.

Platform	Chrome	Firefox	Safari	Edge
Mac OS X 12.2.1	YES	YES	YES	
iOS 15.4	YES	YES	YES	
Android 12	NO	NO		
Windows 11	NO	YES		NO
Debian 11	NO	YES		

Table 1 – OCSP Tests for common Platforms and Browsers

It appears that Apple platforms and Firefox perform OCSP-based certificate revocation queries, while other platform and browser combinations do not. If we look at the user population of today's Internet, mobile platforms dominate, and the global market share of mobile platforms appears to be approximately 80% Android and 20% iOS. Based on these test results, this would mean that a large-scale measurement of client platforms and their capability to detect revoked certificates by using OCSP should reveal a rate of detecting revocation at around 20%.

And this is what we observed in March 2022. A large-scale measurement program directed client browsers to a HTTPS URL where the certificate associated with the service has been revoked. We are unable to directly instrument the client, so we look for connection attempts by picking up the Server Name Indication (SNI) in the TLS handshake from the server's packet logs, and then looking at the server's web logs to see if the TLS handshake completed, and the web object was delivered over the TLS session. The results of a 5-day measurement are shown in Table 2.

OCSP Test – March 2022		
Total Count:	16,480,316	
OCSP Checking Enabled:	3,512,478	(21%)
No OCSP Check:	12,967,835	(79%)

Table 2 – OCSP Measurement Tests

We have enough data to show the geolocation of these users, and the map of OCSP checking is shown in Figure 6. This appears to be roughly correlated with a map of the adoption of Apple products and the use of the Firefox browser platform.

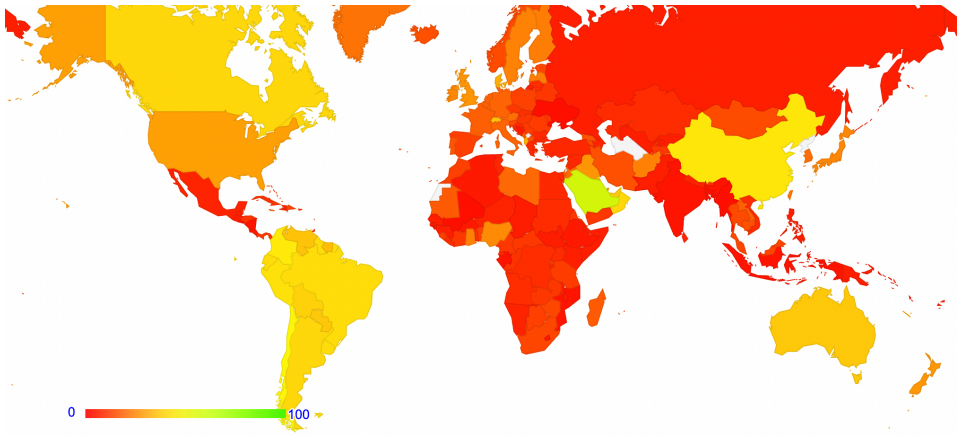

Figure 6 – Incidence of OCSP checking

If we are going to rely on OCSP to perform certificate revocation, then it seems that this is largely an effectual measure. A revoked certificate is only detected by some 21% of the client population. Why don't all platforms and some browsers perform OCSP checks? Surely users would expect that the platforms and applications that they use would implement such seemingly basic protection mechanisms to protect the user from being misled by a compromised security credential. 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.

- ✗ What should the client do if the OCSP request is not answered? Proceeding 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 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

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 certificate to indicate that the OCSP response must always be used when using this certificate, through the OCSP Must Staple attribute in the certificate.

> "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 7.

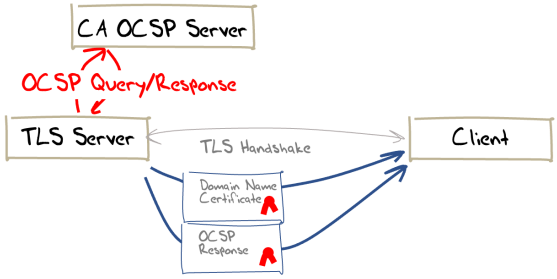

Figure 7 – The Stapled OCSP model

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

- ✓ It's faster than performing a full CRL retrieval and lookup.
- ✓ Its faster than OCSP as there is no additional query and response delay for the client

- ✓ This reduces the load 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 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.

Platform	Chrome	Firefox	Safari	Edge
Mac OS X 12.2.1	YES	YES	YES	
iOS 15.4	YES	YES	YES	
Android 12	NO	YES		
Windows 11	NO	YES		NO
Debian 11	NO	YES		

Table 3 – Stapled OCSP Tests for common Platforms and Browsers

This is an improvement of sorts. The Chrome browser on all tested platforms other than Apple devices still is not checking for revocation when the OCSP response is stapled into the TLS credentials.

We performed a similar large-scale measurement over 5 days in March 2022. The results are shown in Table 4.

Stapled OCSP Test – March 2022		
Total Count:	15,716,397	
OCSP Checking Enabled:	3,138,923	(20%)
No OCSP Check:	12,577,471	(80%)

Table 4 – Stapled OCSP Measurement Tests

The results are not that different from the OCSP case. It appears that stapling OCSP adds nothing to the outcome of revocation checking beyond OCSP queries.

Chrome and Revocation

What's going on with Chrome? Almost a decade ago Chrome signalled that it was going to use a different form of revocation checks (https://www.imperialviolet.org/2012/02/05/crlsets.html). 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 1,020 revoked certificate serial numbers in this list. This seems oddly low. As noted in Figure 3 just one CA, Entrust, lists 5,072 serial numbers in its current CRL. 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. It is also evident that our test certificate, issued by Let's Encrypt and revoked for over a week is not in Chrome's list. Evidently our test is not sufficiently important, or perhaps it's been deemed a frivolous revocation unworthy of inclusion into Chrome's list. The

consequence is that Chrome browsers will happily connect to this test site using TLS despite the certificate's revoked state.

> 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 more and more 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.

We can move to use OCSP, which avoids the scaling problem of large CRLs, but this exacerbates the issue of privacy, as the CA 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.

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 exclusive 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 in the Internet 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 isd 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 eleven 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. The concept illustrates the point that the DNS is more 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 here.

CA's issue long-lived certificates and that means that when a key pair is compromised, this vulnerability may persist for years. We'd like to annul the validity of the certificate in a faster timeframe. Certificate Revocation Lists provide such a service, but they can become large and 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 the client systems turned to a *fail-safe* approach which allows invalid certificates to continue to be trusted.

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 90 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?

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.

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 infrastructure operators, and these days has nothing to do with any desire to improve the obvious pitfalls in the current certificate-based measures.

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 rather than a preferred timescale of seconds or even faster.

The situation points to the uncomfortable conclusion that as far as the security of the Internet is concerned, we are still placing undue reliance on a security framework that at best offers same week service in a millisecond world!

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

Disclaimer

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

Authors

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*

João Damas B.Sc., is the Senior Researcher at APNIC. He is a member of ICANN's RSSAC, an RSTEP panelist and a root zone signing TCR. He participates in ISOC, RIPE, and ESNOG. He is co-chair of the DNS WG at RIPE.