Revocation

Geoff Huston, Joao Damas
APNIC Labs
Certificate Revocation as a "sanction"

Given the sanctions against Russia, it seems that CAs are now ceasing issuance for Russian domains and even going so far as to revoke certificates previously issued for Russian domains. Here are some for a Russian bank revoked by Thawte CA:

<table>
<thead>
<tr>
<th>Certificate ID</th>
<th>Summary</th>
<th>Certificate Transparency</th>
<th>Revocation</th>
</tr>
</thead>
<tbody>
<tr>
<td>crt.sh/ID:5823847935</td>
<td>Entry #</td>
<td>Log Operator</td>
<td>Log URL</td>
</tr>
</tbody>
</table>

Mechanism Provider | Status | Revocation Date | Last Observed in CRL | Last Checked Until |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>OCSP</td>
<td>The CA</td>
<td>2022-03-01</td>
<td>False</td>
<td>2022-03-15</td>
</tr>
<tr>
<td>CRL</td>
<td>The CA</td>
<td>2022-03-16</td>
<td>False</td>
<td>2022-03-15</td>
</tr>
<tr>
<td>CRL/SET/Blocklist</td>
<td>Google</td>
<td>2022-03-16</td>
<td>False</td>
<td>2022-03-15</td>
</tr>
<tr>
<td>Disallowedorsites</td>
<td>Microsoft</td>
<td>Not Reversed</td>
<td>False</td>
<td>2022-03-15</td>
</tr>
<tr>
<td>OutCRL</td>
<td>Mozilla</td>
<td>Not Reversed</td>
<td>False</td>
<td>2022-03-15</td>
</tr>
<tr>
<td>SHA-256</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>F8AEC0292E7328EC1FE943A37724966359B87816D429C2464620E01</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Certificate:

<table>
<thead>
<tr>
<th>Data</th>
</tr>
</thead>
<tbody>
<tr>
<td>Version: 3 (8x2)</td>
</tr>
<tr>
<td>Signature Algorithm: sha256WithRSAEncryption</td>
</tr>
<tr>
<td>Issuer: O=Digital Certification Services, C=US</td>
</tr>
<tr>
<td>Subject:</td>
</tr>
<tr>
<td>commonName = epa.spl.vtb.ru</td>
</tr>
<tr>
<td>organizationName = VTB Bank (PJSC)</td>
</tr>
<tr>
<td>localityName = Russian Federation</td>
</tr>
</tbody>
</table>

Download Certificate: CRL
Certificate Revocation as a "sanction"

Does Certification Revocation even “work”?

Will clients still be able to connect to your ‘secured’ content or service even when the certificate has been revoked?
Let's take a step back...
Why should I trust this?
How can I be assured that this is my bank and not a clever scam?
This sounds reassuring, but why the hell should I trust “Entrust Inc”?

With a name like that they sound pretty dodgy! I have never met these folk and I have no idea what this means.

OK - what does this certificate say?
Fair enough, but why should I trust it?

“This certificate is “valid”
This certificate is valid. Fair enough, but why should I trust it?

This certificate is 7 months old, and I am being asked to trust it for the next 5 months!

What if the private key is leaked in the next 5 months? What if the CA is breached? What if...

What if something happens in the next 5 months that says that I really should not trust this certificate any more?
The Answer is Revocation!

• Each CA maintains a “Certificate Revocation List”
• This is a list of the serial numbers of all current certificates issued by this CA that should no longer be trusted
• If a Bad Thing happens, or for any other reason, and the CA believes that the certificate cannot be trusted, then the certificate’s serial number is added to this CA’s Certificate Revocation List
• Anyone who is worried about the “currency” of a certificate should check to see if its serial number is listed in the CA’s current CRL
The Answer is Revocation!

• Each CA maintains a “Certificate Revocation List”.
• This is a list of the serial numbers of all current certificates issued by this CA that should no longer be trusted.
• If a Bad Thing happens for any other reason, and the CA believes that the certificate is no longer trusted, then the certificate’s serial number is added to the CA’s Certificate Revocation List.
• Anyone who is worried about the “currency” of a certificate should check to see if its serial number is listed in the CA’s current CRL.
The URL for the CA's CRL is listed in the certificate, if the CA publishes a CRL. Here's the one for Entrust Inc:

<table>
<thead>
<tr>
<th>DNS Name</th>
<th>netbank.commbank.com.au</th>
</tr>
</thead>
<tbody>
<tr>
<td>DNS Name</td>
<td>netbank.com.au</td>
</tr>
<tr>
<td>DNS Name</td>
<td>mobile.netbank.com.au</td>
</tr>
</tbody>
</table>

- **Extension**: Certificate Policies (2.5.29.32)
  - **Critical**: NO
- **Policy ID #1**: (2.16.840.1.114408.10.1.2)
- **Qualifier ID #1**: Certification Practice Statement (1.3.6.1.5.5.72.1)
- **CPS URI**: [https://www.entrust.net/cps](https://www.entrust.net/cps)
- **Policy ID #2**: (2.23.140.1.1)

- **Extension**: CRL Distribution Points (2.5.29.31)
  - **Critical**: NO
  - **URI**: [http://crl.entrust.net/crl/crl](http://crl.entrust.net/crl/crl)

- **Extension**: Embedded Signed Certificate Timestamp List (1.3.6.1.4.1.11129.2.4.2)
  - **Critical**: NO
- **SCT Version**: 1
- **Log Operator**: DigiCert
- **Log Key ID**: 55 14 06 9A 2F D7 C2 EC D3 F5 E1 BD 44 B2 3E C7 46 76 B9 BC 99 11 5C CO EF 94 98 55 D6 89 D0 DD
The URL for the CA’s CRL is listed in the certificate, if the CA publishes a CRL. Here’s the one for Entrust Inc... 

It’s not an HTTPS URL, but it’s a signed object requiring authentication, so tampering is challenging whether or not the retrieval transport is authenticated and encrypted.
What's in that CRL?

$ 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
  Last Update: Mar 14 05:00:49 2022 GMT
  Next Update: Mar 21 05:00:49 2022 GMT
  CRL extensions:
    X509v3 Authority Key Identifier:

  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]
How do you check a certificate using a CRL?

1. Retrieve the CRL
2. Validate the digital signature of the CRL against the CRL contents
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 you find it listed in the CRL then the certificate is a dud!*
Does anyone actually do this?
Does anyone actually do this?

No!
Does anyone actually do this?

No!

It takes too long to perform the CRL checking actions and nobody is willing to pay this time penalty.

It's also a totally inefficient design - why retrieve the entire CRL when all you want to know is the status of a single certificate?

This may have worked for circulating printed lists of revoked credit card numbers in a bygone age, but it's completely useless today (though even then nobody checked!)
Plan B - OCSP

Online Certificate Status Protocol

• Allows a client to query the CA to query the revocation status of an individual certificate

• The retrieval, validation and lookup function associated with CRLs is pushed back to the CA, and the client now only needs to validate the OCSP response

• Sounds interesting
Does it work? Is OCSP being used?

• How many users would still visit a website (using HTTPS of course) if the site certificate was revoked and the CA does not publish a CRL?
• Let’s try and answer this question
Measurement Process

• Use a scripted Ad with a URL
• DNS name component is unique (removing caching considerations)
• Generate a Let’s Encrypt wildcard certificate – and then immediately revoke it!
$ 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: 142EB317B75856CBAE500940E61FAF9D88B14C2C6
      Serial Number: 04F6351FB48399440794386973D8BD9C4095
  Request Extensions:
    OCSP Nonce:
      0410E32D127FOCAE78737814324013BF904C

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: 142EB317B75856CBAE500940E61FAF9D88B14C2C6
      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

A manual OCSP check shows that the certificate has been revoked
Measurement Process

- Use a scripted ad with a URL
- DNS name component is unique (removing caching considerations)
- Generate a Let’s Encrypt wildcard certificate – and then immediately revoke it!
- Capture TCP packets at the server(s)
  - SNI field shows the client commencing the TLS initial exchange
- Capture the web logs
  - Log entry is written on server’s object delivery if the TLS session completes successfully (which probably should not happen when the certificate is revoked)
Measurement Expectations

• Let’s Encrypt does not publish its CRL in its issued certificates – so this is an OCSP check
• If every client application performed an OCSP revocation check then we’d see no web fetches at all!
• And if nobody supports OCSP then we’d see a high correlation between the SNI capture and the web logs
• What do we expect to see?
What do applications do today?

Let’s bench test a few common platforms and browsers

<table>
<thead>
<tr>
<th></th>
<th>Chrome</th>
<th>Firefox</th>
<th>Safari</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>MAC OS 12.2.1</td>
<td>OCSP</td>
<td>OCSP</td>
<td>OCSP</td>
<td></td>
</tr>
<tr>
<td>iOS 15.4</td>
<td>OCSP</td>
<td>OCSP</td>
<td>OCSP</td>
<td></td>
</tr>
<tr>
<td>Android 12</td>
<td>NO</td>
<td>NO</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows 11</td>
<td>NO</td>
<td>OCSP</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Debian</td>
<td>NO</td>
<td>OCSP</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Apple platforms and Firefox generally perform an OCSP check, and other's don't.
What do applications do today?

Let's bench test a few common platforms and browsers:

- Chrome
- Firefox
- Safari
- Edge

MAX OS 12.2.1

Apple Platforms plus Firefox appear to have approx. 20% market share.

So we would expect a 20% OCSP check rate in a measurement.
Measurements

Global Outcomes – 5 days in March 2022

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

Seems that theory and practice correlate tolerably well
World Map of OCSP Checking

This is really a map of Apple platform market share by CC.
OCSP Scorecard

✗ It’s still a round time time penalty of additional delay
✗ It tells the CA what each client is doing = a significant privacy leak
✗ It imposes critical load on the CA’s OCSP servers
✗ What should the client do if the OCSP server is uncontactable?
  • Fail? – DDOS vector
  • Allow? – Vulnerability vector

Some platforms and browsers support OCSP checking
Some don’t

This seems like a stupidly inconsistent and haphazard basis for the web’s only security framework!
OCSP = Fail?

“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 - OCSP Stapling?

“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 about OCSP Stapling?

Rather than pushing the responsibility for revocation checking entirely onto the client with CRLs, or turning it into a query/response interrogation of the CA’s server with OCSP, can the content server furnish the ‘current’ OCSP response as a stapled attribute of the TLS credentials?
Stapled OCSP

1. TLS Handshake
   - Domain Name
   - Certificate
2. OCSP Query/Response
   - OCSP Response

Client

TLS Server

CA OCSP Server
What about OCSP Stapling?

Rather than pushing the responsibility for revocation checking entirely onto the client with CRLs, or turning it into a query/response interrogation of the CA’s server with OCSP, can the content server furnish the ‘current’ OCSP response as a stapled attribute of the TLS credentials?

Yes – OCSP Stapling (RFC6066) and TLS Must Staple (RFC7633)

✓ There is no additional network latency in TLS start as the OCSP data is in-band

✓ The CA does not attain knowledge of client-initiated sessions

✓ The CA does not have to operate a high capacity server infrastructure to respond to client OCSP queries

✓ The client can fail ‘hard’ on missing or unvalidatable OCSP data
What do applications do today?

Let’s bench test a few common platforms and browsers

<table>
<thead>
<tr>
<th>OCSP Stapled</th>
<th>Chrome</th>
<th>Firefox</th>
<th>Safari</th>
<th>Edge</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mac OS X 12.2.1</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>iOS 15.4</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td></td>
</tr>
<tr>
<td>Android 12</td>
<td>NO</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Windows 11</td>
<td>NO</td>
<td>YES</td>
<td></td>
<td>NO</td>
</tr>
<tr>
<td>Debian 11</td>
<td>NO</td>
<td>YES</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Why is Chrome not checking OCSP Stapling?
We need to talk about Chrome

• Chrome does not perform OCSP checks – it uses “CRLsets”
  (https://www.imperialviolet.org/2012/02/05/crlsets.html)

• Chrome crawls across participating CAs, trims the CRLs to strip out “unimportant” revocations and sends this to the chrome browser

• What happens if:
  • your CA is not part of Chrome’s CRLset?
  • Your certificate revocation is not “sufficiently important”?

then you lose! Chrome will happily set up the TLS session anyway.
We need to talk about Chrome

• Chrome does not perform OCSP checks, it uses "CRLsets" (https://www.imperialviolet.org/2012/02/05/crlsets.html)

• Chrome crawls across participating CAs, trims the CRLs to strip out "unimportant" revocations and sends this to the chrome browser.

• What happens if:
  • your CA is not part of Chrome’s CRLset?
  • Your certificate revocation is not "sufficiently important"?
then you lose! Chrome will happily set up the TLS session anyway.

I thought the entire approach over the last few years with free certificates and browsers forcing everything to use HTTPS was to shift object security from being a luxury good to a universal commodity.

Yet Chrome is saying that revocation is not universally accessible. It’s more like an exclusive luxury good once more!

Why has Chrome done this?
Who's Serving Stapled OCSP content?

• Cloudflare – yes (https://blog.cloudflare.com/ocsp-stapling-how-cloudflare-just-made-ssl-30/)
• Azure – under construction?
• Fastly – yes: (https://support.fastly.com/hc/en-us/community/posts/360040448792-Support-for-OCSP-stapling-)
• <insert CDN here>
What's the point of OCSP Stapling?

• The default outcome of a evaluation of a current certificate is to accept it, as long as the client can construct a validation chain to a local trust anchor

• CRLs and OCSP is meant to alert you to a changed circumstance that indicates that the certificate should not be trusted

• So the only OCSP item that is useful is one that indicates that the certificate has been revoked

• But why should a server send me the certificate and a stapled OCSP response if the signed OCSP response says that the certificate has been revoked?
  
  • *Shouldn’t the server use this OCSP information itself and fail the TLS handshake if the certificate that it was going to proffer is revoked?*
Revocation is a Failure!

• 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 **NOW**?

• If the best that CRLs, OCSP, Stapled OCSP and CRLsets can inform you is the trust status of a certificate **at some time in the past** then why is this information any different from the certificate itself?

• If the entire purpose of these revocation mechanisms is only to reduce the “trust window” of a certificate, then why not just use certificates with a more constrained trust window (of a few hours or so)?
Certificates are a Failure?

• The problem with certificates that provide a trust window of a few hours, is that the existing CA infrastructure and the use models of locally stashed certificates just can’t cope with such an increased intensity of certificate re-issuance
• So we just persist with long-lived certificates and non-functional revocation mechanisms, because it’s the path of least resistance
• If certificates are incapable of informing a client that they are about to be drawn into misplaced trust then what exactly are they 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?” And the problem is that this entire certificate structure can only answer a question that relates to the past, not the present!
Certificates are a Failure?

• The problem with certificates that provide a trust window of a few hours, is that the existing CA infrastructure and the use model of locally-stashed certificates just can’t cope with such an increased certificate re-issuance.

• So we just persist with long-lived certificates and non-functional revocation mechanisms.

• If certificates are incapable of informing a client that they are about to be drawn into misplaced trust then what exactly are they 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?” And the problem is that this entire certificate structure can only answer a question that relates to the past, not the present!
And then there's the DNS…

The problem here is not TLS, and not the use of digital signatures to assure the veracity of information, but the properties of the certificate infrastructure to allow this veracity to be established as a current piece of information.

The DNS has a similar issue of accuracy and timeliness:
- It manages to contain this time lag though the use of Cache Timers (TTLs), managed by zone admins, to set a maximal time between referral back to the authoritative information sources.
- There is a constant refresh of DNS information passed to the client from the authoritative servers.
And then there's Stapled DANE ...

If the entire purpose of this security measure is to associate a key pair with an intended service name then why not Just Use the DNS?

• Place the public key of a service into the DNS alongside its other attributes
• Use conventional TTL mechanisms to control the maximal caching lag of this information
• Sign these records with DNSSEC
• And staple the DNSSEC validation chain of the public key record into the TLS handshake as a stapled chained response as an alternative to using X.509 certificates

i.e. use DANE, Chained Responses and TLS Stapling
Where have we got to?

• X.509 certificate revocation is broken **and we can’t fix it**

• About the only fix is to pull certificate lifetimes down to a small number of hours – i.e. limit the scope of potential damage of a compromised certificate
  • But the certificate system as we know it won’t scale with such massively shortened certificate lifetimes

• Yet in the DNS, the common TTL is measured in hours

• So why don’t we just ditch all this X.509 brokenness and just turn all of this over to DANE and the DNS?
Questions?