Fixing Revocation for Web Browsers on the Internet

Tom Ritter — tritter@isecpartners.com

iSEC Partners, Inc
123 Mission Street, Suite 1020
San Francisco, CA 94105
https://www.isecpartners.com

November 28, 2012

1 Introduction

Revocation for a certificate occurs when the certificate must be recalled prior to the 'Valid To' date specified in it. While it is possible to revoke an expired certificate (and it may be useful to do so in some cases), in the context of browsers an expired certificate is invalid anyway. A certificate may be revoked if the certificate recipient breaches contract with the Certificate Authority (non-payment perhaps), if the private key of the certificate is no longer controlled by the original organization (important in the case of theft, less important in the case of deletion), or if the certificate was found to be issued to a fraudulent applicant.

In the past year, the last case of fraudulent applicants has shown a spotlight on CA practices and security, but strangely a much smaller spotlight on revocation. In all cases, the issuing CA was entirely unable to revoke the certificates with any meaningful effect and most browser vendors had to take action to protect their users. Browser patches are overkill for revocations that do not impact user's security, and would never be done. But even when the revocation does protect users they are costly, inconvenient, may not always be applied quickly and subject to blocking. As long as the Certificate Authority system exists, a working revocation system is in the best interests of all consumers of the CA-based Public Key Infrastructure.

Systems that sit adjacent to Certificate Authorities - such as TLSA Records[1] or alternative-PKI Systems like Convergence[2] - have different requirements for Revocation. Because it seems that Certificate Authorities will be around for a while, we leave discussion of the alternate approaches to PKI for another time. Additionally, there are a myriad of PKI implementations that have different requirements than web browsers. Mail Transfer Agents, Client Certificates, and less common deployments are exceptions to the general case of web browsers. We focus on revocation in browsers, and the billions of people who rely on it daily.

2 Existing Methods

Although nearly a dozen methods of checking certificate revocation have been proposed, only three have been implemented widely in clients. The three implemented protocols are documented, along with short notes about a few other methods. A good resource on revocation is *Engineering Security* by Peter Gutmann, currently a draft[3], around page 567.

2.1 Certificate Revocation Lists

A Certificate Revocation List (CRL) is a list of Serial Numbers of certificates a Certificate Authority has revoked, along with issue and next-issue dates, signed by the CA. CRLs are updated every couple days, and are cached on the client for up to a week. CRLs often must be explicitly installed on clients - while it is possible to put a CRL loca-

[1] https://datatracker.ietf.org/doc/draft-ietf-dane-protocol/
[2] http://convergence.io
[3] http://www.cs.auckland.ac.nz/~pgut001/pubs/book.pdf

tion into the certificate itself via an extension, this is not always done. Besides problems with locating them and timeliness - CRLs can grow to be very, very large - which is a concern particularly with mobile clients. Techniques to segment them by revocation reason or perform delta CRLs have been proposed, but not widely adopted.

2.2 Online Certificate Status Protocol

The Online Certificate Status Protocol (OCSP) is a challenge-response mechanism that's designed to be more timely than CRLs. When a client receives a certificate from a server they query an OCSP responder, which is almost always run by the issuing CA, to check the revocation status of the certificate. The responder will reply stating that the certificate is either 'good', 'revoked', or with a third status 'unknown'.

OCSP responders require another two roundtrips before the certificate can be verified - one for the TCP handshake, another for the OCSP request and response. This is required for each new certificate seen - several on a page with third party resources. OCSP requests should be made for each certificate in the certificate chain (except for the hardcoded root certificate). OCSP requests can add up quickly on a complex web page, potentially requiring a couple dozen round-trips for all the resources of a page.

It is important to distinguish between two modes that browsers take when working with OCSP responders: hard-fail and soft-fail. 'Soft-fail OCSP' is when a browser will silently continue a TLS connection even if it does not receive a positive (or any) response from an OCSP responder. This provides no security, because an attacker performing a man in the middle attack can simply block OCSP requests or responses. Browsers and Site operators are not fond of 'Hard-Fail OCSP' which will prevent access to the site if a OCSP response is not received - the OCSP responder becomes a new single point of failure for the site, one that is out of the site's operational control.

Other problems are present with OCSP. The 'unknown' status code is ambiguous and there is no recommendation as to how clients should behave or when it will be sent. OCSP is also driven off serial numbers instead of certificates. Additionally, OCSP leaks the client's browsing habits to the operator of the OCSP responder. It's also worthwhile to note that OCSP was originally implemented with support for nonces - a challenge/response mode that prevented an attacker from replaying an old (but still valid) OCSP response - this was removed so OCSP responders could scale[4]. Today, OCSP responses are valid for a day to two days usually, and can be replayed with impunity during that window.

2.3 OCSP Stapling

OCSP Stapling (aka the CertificateStatusRequest TLS Extension as defined in RFC 6066) attempts to address the three largest problems with OCSP. Instead of a client obtaining an OCSP response, the server will obtain the OCSP response in advance and provide it to the client. Because the response is signed by the CA, it is fine for the server to provide it to us, and this significantly reduces the single point of failure, the additional round-trip required, and also the privacy leakage.

OCSP Stapling is in the early stages of being supported by webservers and user agents. Additionally, the current TLS extension only supports stapling a single OCSP response. Most certificates have a chain of three: the root, an intermediate, and the leaf. The root is hardcoded as trusted, and thus would really need a patch to properly distrust. But an intermediate cert can be revoked, and thus should have a revocation check. Practically, intermediate certs are rarely revoked, which provides some leeway, and a proposal to add multi-stapling is being discussed[5].

2.4 Browser Updates & Short-Lived Certificates

Google has announced[6] that Chrome will move off OCSP queries and will instead push CRLs down to the browser using a system similar to its autoupdate mechanism - although a browser restart won't be required for them to take effect. This technique, dubbed CRL-sets, is very similar to a software patch, which is what has ultimately been done in nearly all major certificate compromises: Code-Signing Certificates for Microsoft, mis-issued certificates from Comodo, the hundreds of certificates issued from Diginotar, and the intermediate certificate issued from Trustwave.

[4] http://www.imc.org/ietf-pkix/old-archive-04/msg00168.html
[5] http://datatracker.ietf.org/doc/draft-pettersen-tls-ext-multiple-ocsp/
[6] http://www.imperialviolet.org/2012/02/05/crlsets.html

A proposed solution to revocation is to do away with it altogether and use short-lived certificates. If a certificate is only valid for a week, revocation will be achieved through expiration. Although a certificate can be re-issued for the same public key if no problems are present, this ultimately requires close coordination between Certificate Authorities and customers, and automatic online signing.

3 Desirable Properties

A working revocation system should provide the following properties:

- Privacy Preserving
- Performant
- No Single Point of Failure
- Uniquely Identifies Certificate
- Effective

3.1 Privacy Preserving

A Privacy Preserving revocation system must not leak a certificate status request to any third party, including a Certificate Authority. While it's true that a majority of sites leak a user's visit to third parties via CDNs, third party includes, analytics, or ads - it is the site's *prerogative* to do so. It is possible to build and deploy a site that does not do those things, and correspondingly it would be unacceptable to build a revocation system that required such a site to choose between violating their privacy policy or require their users to be at risk.

There are two techniques to achieve a privacy preserving revocation system. In the first technique, an entity will act as a proxy for the client, and pass the status request through to the verifier. In some cases, the proxying entity already knows the site being visited - this is the case in OCSP Stapling and could be the case if OCSP responses were put into DNS. In other cases, the proxying entity is a trusted (or untrusted) party and will make the request on behalf of the client, passing the response back. This is a technique used by Convergence, and similar to onion routing employed by Tor.

The other technique is borrowed from a method that defeats traffic analysis: send all the data everywhere. If every client knew every certificate that was bad, it would know the inverse set: all the certificates that are good. Because every client knows all the bad certificates (the innocuous and the embarrassing) no client is distinguished from any other client. Status requests about individual or groups of certificates are never made. This is an approach similar to CRLs, with the exception that clients don't download all CRLs, only the ones that apply to the certificates they see. This could potentially pose a problem if a Certificate Authority segmented CRLs - the CA would know they had visited one of the sites serviced by that CRL. By strict segmenting, such as certificates related to adult industries, this could potentially expose information to the CA. Browser-pushed updates come the closest to the "send all the data everywhere" technique.

A third technique, using Private Information Retrieval (PIR)[7] could probably be devised, but would require a considerable development and review process for an entirely new protocol. If a simpler solution can be used, it is preferred.

Acceptable Methods:

- OCSP Stapling
- CRLs, with caveats
- Browser Updates (CRL-Sets or Updates)
- As-Yet-Undeveloped

3.2 Performant

Verifying the status of the certificate presented by the server should not require any additional network requests. If CRLs or browser updates are fetched asynchronously by the client (browser or operating system), they will not block the handshake except in the case they are very out of date (e.g. the client has not had the opportunity to update). The other solution is to offer proof of non-revocation along with the certificate. Because the server is usually not the certificate-verifier, the proof must be signed by the issuing Certificate Authority to provide the proof of non-revocation. This is how OCSP Stapling works.

A less critical performance consideration is the size of the additional data sent by the server. A concern voiced by the more technical individuals in the discussion has been about the size of the OCSP Stapled data - a large enough staple could cause the TCP window to overflow

[7] https://en.wikipedia.org/wiki/Private_information_retrieval

during the TLS handshake and result in the server waiting for an ACK by the client on the TCP-level before being able to complete the ServerHello. While the lower layers are often abstracted over, it's important to remember that being conservative in bytes sent on the wire can result in real-world performance improvements.

Acceptable Methods:

- OCSP Stapling
- CRLs, to a lesser extent
- Browser Updates, to a lesser extent

3.3 No Single Point of Failure

A single point of failure is a part of the system that, if it fails, will stop the entire system from working. A server operator can eliminate single points of failure for their web site by having redundant internet connections, web servers, load balancers, nameservers, application servers, database servers, firewalls, switches, or even an entire hot-standby site.

If a client will not connect to the server because a certificate verifier is down, the certificate verifier becomes a single point of failure. Furthermore, the certificate verifier is a black box from the point of view of the server operator: they don't operate it, can't monitor it, upgrade it, or improve its redundancy. It may be fully redundant with a hot standby site, or it may be a commodity server running under someone's desk: there is no transparency. Additionally, the certificate verifier and the server operator may not have a Service Level Agreement. There is no way for the server operator to ameliorate the risk the certificate verifier imposes.

Hard-fail OCSP requires perfect uptime for an OCSP Responder. However, CRLs and OCSP Stapling permit short periods of downtime. OCSP Stapling is a form of this: as long as a server is able to contact the OCSP Responder sometime in a 12 to 24-hour window, they should be able to receive an updated response. (Also, because OCSP requests would be less frequent, the responder handles less traffic.) A CRL is intended to be updated every couple days - if a client is at hour 24 of the 28 hour validity window and unable to contact the CRL server - they would be still be considered 'safe' if they waited for an hour and were able to update it then.

Another non-technical approach is for browsers to impose a Service Level Agreement on the OCSP Responders - to be considered for the root program of the browser, the CA must be able to operate a responder with a specific degree of uptime and throughput, and if they find themselves unable to meet the requirements, they would be removed or restricted in the browser.

Acceptable Methods:

- OCSP Stapling
- CRLs
- Browser Updates

3.4 Uniquely Identifies a Certificate

A certificate consists of a variety of fields including a Serial Number, Issuer, Validity Period, Subject, Subject Public Key Info (SPKI), and Critical and Non-Critical Extensions. The Serial Number is intended to be a number unique within the domain of a Certificate Authority, and has been from the original standard in 1999[8]. However, serial number uniqueness is not a technical constraint, nor does it provide guidance on the method to guarantee it within a CA. CAs have acted in good faith with respect to Serial Numbers, but even still Serial Numbers overlap across CAs. This should not present a security issue, as revocation information should come from the CA that signed the certificate - but it can be confusing.

It is possible to uniquely identify a certificate however. The Subject Public Key Info (SPKI) is the first thought to uniquely identify a certificate; however, there are legitimate reasons to revoke a certificate but reuse the same key - a misspelling in the identifying attributes for example. A better technique is to use a hash of the certificate, known as the fingerprint or thumbprint.

Five years ago, attacks on the issuance of certificates by Certificate Authorities was a largely-dismissed concern. Today, the threat of duplicate serial numbers is similarly dismissed. But as an experiment - let us consider what would happen if an attacker *could* fraudulently issue a certificate with a Serial Number of their choosing. The smart attacker would issue a certificate whose Serial Number duplicated the Root or a widely-used Intermediate CA. The certificate could not be revoked through CRLs or OCSP - as it would simultaneously revoke the CA's valid signing cert!

Today, this would actually be business as usual - all browsers issue patches to distrust fraudulent certificates today, so distrusting this overlapping certificate would

[8] https://tools.ietf.org/html/rfc2459#section-4.1.2.2

be easy. But the end goal of fixing revocation is to move to a solution where software patches are not necessary. A system that never allows an attacker to control the Serial Number or a revocation system that can revoke based on a unique identifier is required - we must have one or the either. The former is a policy constraint but the latter is a technical constraint and thus more desirable. As an implementation detail, it is entirely reasonable to use the Serial Number to augment database queries, but the check should be performed against a unique identifier - the fingerprint.

Today when blacklisting fraudulent certs Chrome[9] and Firefox[10] appear to perform the blacklist by the Public Key and not Serial Number. Other vendors are unknown. If a vendor did block the certificate by Serial Number, a duplicate serial number would mean they would remain vulnerable until they could architect a solution to their certificate validation that blocked by Public Key, and deployed it.

Having a revocation mechanism that uses technical methods to uniquely identify a certificate, instead of hoping that attackers are never able to perform a specific action, is a forward-thinking solution. A revocation system that is designed to meet the requirements of the Public-Key Infrastructure for years to come should rely on technical constraints instead of policy agreements.

Acceptable Methods:

- Browser Patches

3.5 Effective

Currently, our Revocation system does not work. It's conceivable any of the revocation systems we have now could be made to work - but they may not have all the desired properties and the problems encountered with them so far would still be valid.

CRLs can provide complete coverage *if* a client (browser or operating system) is diligent about updating them. Additionally, every new certificate seen should have coverage under a CRL - because users cannot and will not add CRLs manually, the CRL distribution point must be specified in the certificate or signing certificate. If the CRL is not present on the system or up to date, it must be refreshed (or downloaded) prior to trusting the certificate. This can turn into a significant side channel, especially with a large CRL, if clients don't pro-actively pursue CRLs. It's reminiscent of the difference between OCSP soft and hard fail - for security the client would have to hard fail but as we've seen, clients are reticent to do so. This does not bode well for CRLs.

We know already what it would take to make OCSP work - clients would have to hard-fail if the OCSP responder never replies, cannot be contacted, or anything other than a positive assertion is received. Moxie Marlinspike showed in 2009 that most implementations would not fail if they received an unsigned "Try Later" response[11]. But server administrators are uncomfortable with this and clients are loathe to be the first to turn it on. Browsers are still competing for Market Share - besides slowing down their product compared to their competitor's, it could cause more failures that an uninformed user would attribute to their product being "broken". And there's practical problems with captive portals. If a hotel or airline wifi network only allows access to a gateway until you pay or accept an agreement, the client can't obtain revocation information about the certificate presented by the gateway! Well-implemented gateways could allow such the revocation traffic through, but these sorts of gateways are not usually known for their robust implementations.

OCSP Stapling's problems in comparison are relatively minor. The major problem of OCSP Stapling as it exists now is the standard allows only a single certificate status to be stapled. Without being able to staple Intermediate Certificate's status, they would need to be checked against CRLs or OCSP. As previously mentioned, this deficiency is in the process[12] of being corrected. A few webservers and browsers need to add support for Stapling as well. But if these relatively minor deficiencies were corrected, and OCSP Stapling deployed, it would work well for certificates consumed by browsers. The browser would know that as of so many hours ago the certificate was still valid, and high-profile sites (with good relationships with Certificate Authorities) could have their OCSP Staples refreshed hourly or even more granularly. Unfortunately, the major drawback of OCSP Stapling is that server operators must take significant steps to upgrade their infrastructure and test OCSP Staple fetching, as opposed to laying the burden on the Certificate Authorities or Browser Vendors.

[9] http://code.google.com/p/chromium/issues/detail?id=94673
[10] https://bugzilla.mozilla.org/show_bug.cgi?id=682927
[11] https://www.blackhat.com/presentations/bh-usa-09/MARLINSPIKE/BHUSA09-Marlinspike-DefeatOCSP-PAPER2.pdf
[12] http://datatracker.ietf.org/doc/draft-pettersen-tls-ext-multiple-ocsp/

Finally, for a revocation system to be effective, a broken system must not be an option for an attacker. OCSP Stapling, if deployed universally, would only improve security if a client also hard-failed on CRL or OCSP queries.

4 Conclusion

Taking into account the properties we desire, and desiring a path of least resistance to a working system - we should develop OCSP Stapling into the choice *de rigueur*.

OCSP Stapling support must be built into the webservers that do not support it, and Certificate Authorities must document and support stapling, and move away from CRLs. It's in their interest to do so: they also want a working revocation system, and stapling is less resource-intensive than direct OCSP queries from clients. Minor tweaks to the protocol can be developed today to handle multi-stapling, perform identification via a fingerprint, and perhaps convey additional status information from the Certificate Authority. Even if the improved standard is not completed for several years, performing an evolutionary change will be easier than the first, revolutionary change we must begin today in earnest: upgrading the internet.

In ten years time, we would like the vast majority of SSL/TLS-Enabled sites, including the overwhelming majority of the top-visited sites, to use OCSP Stapling. This should be accomplished by making it the default for software, for CAs, through browser clues, and social pressure. When the sites that account for the majority of traffic staple OCSP responses, browsers can feel confident enough to enable OCSP Hard-Fail and reject a TLS connection when they cannot connect to the OCSP Responder.

Even the most long-lived certificates issued today will have been renewed by the time OCSP Hard-Fail is turned on. Web-site operators will have no excuse not to take into account the drawbacks of Hard-Fail, but they also have the ability to punt the problem to the browser and CA - they don't actually have to upgrade their software if they don't want to. They will simply see a slower connection time, and rely on the uptime of their CA's responder.

When CRLs are gone, browsers perform hard-fail OCSP checking, and the majority of sites provide OCSP staples, we will have a working revocation system.

5 More on OCSP

If the goal is to move to an improved version of OCSP Stapling, we should document what we mean by 'improved'.

5.1 Multistapling, Cached Info & Unique Identification

Multi-stapling [13] will allow Intermediate Certificates to have OCSP staples provided in the initial TLS handshake.

TLS Cached Info[14] could be extended to add a type for 'OCSP Staple'. If a client has a valid staple that has not expired yet for a certificate belonging to this domain (or potentially for Intermediate Certificates), it can specify it in the TLS Client Hello and save the server from returning the additional data, reducing the size of the server response.

As mentioned previously, it is a technical constraint to perform revocation on a property that uniquely identifies a certificate - therefore, OCSP should be improved to require the client to include the fingerprint in the OCSP query, in addition to the serial number. The OCSP Server could use the serial number to key into a database, but before responding "good", must verify the fingerprint requested by the client matches the fingerprint in the database.

5.2 Online Status Checking & Resolved Ambiguities

There have been several recent discussions around having OCSP responders move from a model that indicates "This certificate is not revoked" to one that indicates "This certificate has been validly issued and should be trusted for the following window." The CA/B Forum has recently voted and passed an amendment[15] [16] that requires Certificate Authorities "MUST NOT respond with a 'good' status for [certificates that have not been issued]" with an implementation deadline of August 1, 2013.

This is a welcome improvement to OCSP but mapping it to the existing statuses of `good`, `revoked`, and `unknown`

[13] https://datatracker.ietf.org/doc/draft-pettersen-tls-ext-multiple-ocsp/
[14] http://tools.ietf.org/html/draft-ietf-tls-cached-info
[15] http://cabforum.org/pipermail/public/2012-July/000181.html
[16] http://cabforum.org/pipermail/public/2012-August/000302.html

is proving troublesome. An `unknown` response *sounds* like the correct choice - but clients today interpret this as meaning the OCSP responder cannot speak authoritatively about the certificate, and they should look at other revocation sources. Likewise, a `good` response *sounds* like the incorrect choice because it implies this certificate's status is 'good' - but the specification says this "does not necessarily mean that the certificate was ever issued".

The client does not care about databases or CRLs, it wants to know if it can trust the certificate. The standard needs to convey unequivocally to a client a `valid` response indicating the client can trust this certificate to the best of the responder's knowledge; an `invalid` response indicating the client cannot trust this certificate to the best of the responder's knowledge; and just-in-case in a fraction of cases - an `unknown` indicating the responder cannot speak authoritatively about this certificate and the client should seek the data elsewhere. If the client has no other source of information, it should revert to not trusting the certificate. It seems reasonable that additional information (such as "certificate on hold", "whitelist backend", and "CRL backend") could be conveyed in extensions - but they are supplemental to the `valid` or `invalid` response.

The standard also defines five exception cases, with some guidance given. However, the guidance does not indicate what the client should do in any of these cases. The most important statement to make is the client fail closed - if it cannot receive a 'valid' response from the OCSP responder, potentially after retrying, it should treat the certificate as invalid. Otherwise, some nuanced suggestions can be made:

- malformedRequest - The client MAY retry the request, potentially removing extensions, before treating the certificate as invalid
- internalError - The client SHOULD retry the request, potentially with another responder, before treating the certificate as invalid
- tryLater - The client MAY retry the request before treating the certificate as invalid
- sigRequired - The client SHOULD sign the request. If no mechanism is available for signing, the client SHALL NOT treat the certificate as valid
- unauthorized - The client SHALL NOT treat the certificate as valid

Because all error cases require a client to treat the certificate as invalid, it is less important to sign these responses. However, if an adversary can issue an unsigned `malformedRequest` exception code, they may be able to cause a downgrade attack in the protocol. This is the same issue that is being dealt with today with TLS versions. However, a signed `malformedRequest` could be retrieved from a server by an attacker, and replayed (unless the nonce extension is used, which is covered later). More thought should be given to this case and different scenarios explored.

The specifics of how OCSP must be changed is being debated still in the IETF, and is not likely to be resolved soon. However it is important the protocol convey unambiguous information about whether the client should or should not trust the certificate.

6 Replay Window

The replay window is the window in which invalid revocation information is considered valid. As an example, an attacker compromises a webserver, stealing the certificate at 8 AM on June 1st, and the certificate is revoked at 9 AM the same day. After stealing the certificate, they also retrieve an OCSP response from the CA, which is presigned at 1 AM each day. The attacker is able to replay this OCSP response from 8 AM (time of compromise) until 1 AM (time the response will be considered invalid) - for a total of 16 hours. The validity period of an OCSP response is the maximum replay window available.

6.1 Securing the Replay Window

A replay window exists in any situation where there is not a challenge/response protocol. For high-value transactions, it may be desirable to eliminate the replay window entirely and utilize the challenge/response mode of OCSP. OCSP supports this already with the `id-pkix-ocsp-nonce` extension. If an OCSP staple is not provided in the TLS handshake, the server may wish to use the nonce option - there is no way for the client to know, absent a new certificate extension. A nonce as short as 128 bit should never be duplicated, much less in a short validity period, so it would be wise to have a client include a short nonce in any explicit OCSP request. The server *may* respond with a signed request with the nonce for such a high-value transaction.

If the server does not wish to use the nonce, it would reply with a normal response ignoring the nonce - this may

be presigned. But it should not be possible to replay a nonce-less reply for a high value certificate, because the OCSP responder should never sign a request without a nonce for such a high-value certificate.

The nonce is likely to be ignored in the majority of cases, so it and the supporting structure should be as small as possible in the request. Although it may seem ludicrous that any CA would move back to supporting nonces, it is not unreasonable to imagine a future internet where a company like Cisco operates an Intermediate CA with a name-constraint to *.cisco.com and chooses to use the nonce feature for something like product updates. Without including a nonce in the request, Cisco would be prevented from choosing their own security policy.

6.2 Standardizing the Replay Window

However, although the nonce extension eliminates the vulnerability of the replay window, it has not been in wide use for some time - migration off the extension began in 2004[17]. Therefore the replay window is a fact of life, and as a critical security detail, should have some guidelines around it. We would suggest a window of 24 hours.

[17] http://www.imc.org/ietf-pkix/old-archive-04/msg00168.html