Protecting Content from the Cache

Richard Barnes and Alissa Cooper

Real-time applications like the web always face a tension between security and performance. On the one hand, content can be delivered more quickly when it is relayed by a point closer to the consumer. On the other hand, realizing these benefits across a large set of consumers requires introducing relays that are trusted by producers and consumers of content, even if only to a limited degree.

The web has long relied on such relays, typically referred to as “caches.” The word “cache” appears more than six hundred times in RFC 2616. Server operators routinely rely on caches to improve performance, and network operators have long relied on ISP-provided caches to avoid unnecessary uplink utilization. However, earlier notions of caching have failed to keep up with the security requirements of the web.

HTTPS breaks caching. Because HTTPS relies on a channel-based encryption scheme (HTTP over TLS), it can only be intermediated by breaking the channel[1]. Client-side proxies and network-provided proxies are both rendered ineffective by HTTPS, or else they are introduced by terminating and re-initiating TLS, preventing end-to-end assurances and often weakening the overall security of the connection. The server that the client authenticates is often no longer a server operated by the holder of the server’s domain name, but instead some delegated party such as a CDN.

How can we re-introduce the benefits of caching, while maintaining the security benefits of HTTPS? In particular:

- How can we allow third parties such as enterprises and local organizations in bandwidth-limited areas to volunteer caches to improve delivery of web content?
- How can we maintain the security assurances of HTTPS in the presence of such caches?

In both of these regards, the current suite of proprietary solutions present improvements over the state of the art and areas for improvement.

Making the web work for more of the world will require it to work with lower bandwidth and higher latency than the modern web often assumes, and caching is a key part of this transformation. We are already seeing signs of this in current efforts to expand connectivity:

[1] Ironically, HTTPS came after an earlier effort (SHTTP, RFC 2660) that was object-based and would have been more cacheable.

- One of the primary goals of Service Workers is to enable an “offline-first” user experience, which continue to work even when the Internet is down.
- The Google Play Store has taken steps to enable peer-to-peer distribution of Android apps, so that users can install apps without being able to access the Play Store.
- The Facebook Free Basics program relies on highly invasive proxying of web traffic to allow carriers to reduce the bandwidth consumed by web pages.
- Community networking efforts such as the Community LTE Project seek to leverage locally-provided caches to increase the efficacy of their networks.

The need for local caches is even more pronounced when it comes to less “webby” uses of HTTP, such as software updates. In many such use cases, many clients are downloading the same, large file at nearly the same time -- an ideal use case for caching.

However, as noted above, the current approaches to integrating caches in an HTTP interaction can’t address these cases without significant reduction in security. There is a need for new technologies that enable caches to be securely introduced and used.

Some requirements for these technologies follow from the underlying security and privacy goals. HTTPS assures that the only parties involved in an HTTP interaction are those admitted by the client or the server. So the process of introducing a cache will have the form of a discovery process, by which the client or server learns of the existence of a proxy and admits it to HTTP interactions. To prevent abuse of this discovery process, there will be a need for some framework to authenticate and authorize caches. And to avoid leaking information to the cache unnecessarily, there will need to be some means for the client and server to agree on whether the cache should be involved in a given request.

The caching process itself also needs to maintain the security properties of the content, including protecting it from the cache. Historically, this has not been done. HTTP interactions via a proxy not only exposed all of the information exchanged to the proxy, but also allowed the proxy to modify content. (Indeed, modifying content is the motivation for many proxies, such as the Flywheel compression proxy or the Free Basics proxy mentioned above.)

At least in their original designs, all the syndication schemes mentioned in the workshop call for papers (from Google, Apple, Facebook, and Baidu) failed in the same way. Users would receive no direct assurance that the article they see is genuinely provided by the publisher, and all of the content accessed by a user can be read by the syndication/caching provider.

The Signed Exchanges and Bundled Exchanges proposals from Google provide a partial solution to this problem. They provide integrity and authenticity protection with respect to the cache, so that the client can verify that the cache is providing what the publisher intended. They provide no confidentiality, though. The proponents of these solutions have tried to argue that HTTPS itself doesn't provide meaningful confidentiality protection in such syndication use cases, even without bundled content. But these arguments fail for a couple of reasons: First, they rely on special features of the AMP use case that may not apply more generally (e.g., the ability of the cache to inject event listeners). Second, while some traffic analysis research has shown success in identifying widely-available content (e.g., films being watched on Netflix), such mechanisms require training that would not be possible if the plaintext of the content were not already available to the attacker.

Providing confidentiality is not impossible. The Blind Caching proposal from a few years ago illustrates the general outline of a solution: Caches store and redistribute encrypted data under opaque identifiers, and the server informs the client of the appropriate identifier and encryption key for the desired content directly. If the key fetch is tunneled via the proxy, this pattern still meets the "private pre-fetch" objectives expressed SXG/BXG specifications. While the details of the scheme might need to change according to the use cases being addressed, the general point is that the cost of confidential caching is one round-trip to the origin server carrying a few hundred bytes of data. For cases where caching matters, this is not much of a loss.

As the Internet connects more of the planet, the diversity of network conditions an application can expect to encounter will grow. Applications will need to adapt to be more in harmony with the network environments in which they're deployed, and in particular to make more efficient use of constrained networks. And they'll need to do it without sacrificing the high levels of security that modern protocols provide.

The current suite of content syndication protocols are designed to provide a specific type of efficiency in a narrow set of use cases. As we consider building common technologies for better content caching, we should insist that these maintain the security standards we've established for the web, and we should not tailor these technologies so narrowly that we miss important adjacent use cases.