The Security Implications of HTTP/2.0

Isobel Redelmeier
iredelmeier@gmail.com
Mentor: Ming Chow

Abstract

While the main changes that HTTP/2.0, the upcoming upgrade to HTTP that is being built on a foundation of the HTTP/1.1 and SPDY protocols, focus on improving performance, these changes also have significant security relevance. Performance improvements such as multiplexing and header compression, as relatively new technologies to the World Wide Web, should be used with proper awareness that exposed vulnerabilities, such as the CRIME exploit, will not necessarily be fixed in HTTP/2.0. However, HTTP/2.0 also looks to patch existing security issues in HTTP/1.1, including limiting the cacheability of pushed resources by servers that host multiple tenants. The most significant security question of HTTP/2.0 – pervasive or selective encryption – may have important effects the Web as a whole, but it remains undecided.

1. Introduction

The Hypertext Transfer Protocol (HTTP) has existed in its current state, HTTP/1.1, since 1999. Fourteen years is a significant reign in today's world of rapidly changing technologies, and the next iteration, HTTP/2.0, is currently scheduled to be submitted for standardization at the end of 2014 [7]. As HTTP forms the backbone of the entire World Wide Web, the security of the protocol is vital to both individuals and corporations seeking privacy and confidentiality in their operations. What are the security implications of the upgrade from HTTP/1.1 to HTTP/2.0? I will examine this question through by comparing the proposed and discussed versions of HTTP/2.0 to HTTP/1.1 and SPDY, a new network protocol and significant inspiration of HTTP/2.0, with regard to the security considerations outlined in the current specifications as well as some that are not listed.

Answering this question in its entirety would be impossible at the present time for two main reasons. First and somewhat trivially, perfect security of something as widespread as HTTP would be nigh on impossible – especially when one factors in all of the emerging protocols and technologies as well as the recent adaptations to existing protocols that introduce new use cases of HTTP. Second and more importantly, the specifications of HTTP/2.0 are far from completed. A time machine or crystal ball would currently be necessary to anticipate what the final version of HTTP/2.0 will entail, not to mention all of the possible exploits and vulnerabilities that it may contain. Many significant decisions are from concluded. Most obviously, the question of how to integrate Transport Layer Security (TLS) or another form of encryption remains hotly debated amongst the working group and other participants in the HTTP/2.0 community. Altogether, this

means that in the ensuing paper I will attempt to not only address the questions already being asked about the security of HTTP/2.0, but also to reveal those which have yet to be asked.

1.1 Background

While the working group acknowledges the significance of security in framing HTTP/2.0, security is not the focus of the upgrade. Instead the primary purpose is to improve the performance of the HTTP protocol, especially "end-user perceived latency, network and server resource usage," and in part by "allow[ing] the use of a single connection from browsers to a Web site" [8]. The charter acknowledges that, among other aims, HTTP/2.0 is expected to include a method of negotiating between HTTP versions as well as between HTTP URLs and other protocols; header compression; server push; and multiplexing, which enables multiple streams to be open between one connection at the same time [7].

HTTP/2.0 is meant to stay similar enough to its predecessor, HTTP/1.1, that the same APIs and methods will still be applicable. However, it also builds heavily on SPDY, a protocol proposed by Google. SPDY offers many of the improvements planned for HTTP/2.0, including header compression, server push, and multiplexing [1]. In adddition, the protocol requires TLS. It has been available (albeit not standardized) for a couple of years in certain browsers, including Chrome, Firefox, and Opera [3]. Currently it is used by a limited but expanding set of Web sites, including Google, Twitter, Wordpress.com, and Facebook [5]. Of note, SPDY beat two other proposals including Microsoft's Speed+Mobility, which built heavily on SPDY but did not require pervasive TLS [12].

2. To the Community

An upgrade to HTTP may seem like an irrelevant concern to many, let alone its relative security to the present version. Few people in the industry were even particularly aware of the plans for HTTP/2.0 until the recent – and likely premature – announcement that it would require HTTP Secure (HTTPS) encryption across all Web resources accessed by way of it [13].[1,2] Insofar as they are interested in jeeping their own data secure online, however, the security of HTTP is not a trivial matter.

Corporations with any amount of confidential data available online need to be aware of any extra security measures that they need to take to ensure that confidentiality remains intact. Likewise, individuals should be aware of the risks to their privacy incurred by the use of certain Web sites and technologies. Even if a person is not responsible for an HTTP/2.0 server, they will still be likely to interact with one, and the technologically knowledgeable citizen will want to be aware of their environment online. Ultimately, anyone using the World Wide Web is only as secure as the Web itself is; and if the structure of the Web is changing, then the current security status quo may be in flux.

3. Action Items

In the ensuing section I will examine a series of potential or known areas of vulnerability for HTTP/2.0. Of these, 3.3 through 3.8 are explicitly addressed in the current specification draft. The headlines of these subsections are borrowed directly from the specifications.

3.1 CRIME

1 Based on anecdotal evidence obtained through conversations with various persons in technological positions in the industry and in academia, as well as with others in the public; as well as a non-quantitative examination of the online and media responses.

2 As of writing time, the decision to enforce pervasive encryption remains unresolved and will be treated as such throughout the duration of this paper.

Compression Ratio Info-leak Made Easy (CRIME) is a security exploit uncovered by Juliano Rizzo and Thai Duong.[3] CRIME takes advantage of header compression at the time of certification renegotiation by following the same zlib pattern that the compression does, allowing an attacker to insert a dangerous path among the cookie byes that will allow them to obtain and read the cookie data. Because it involves header compression, CRIME requires a system in which both the server and the browser allow for this compression. CRIME was therefore relevant only with respect to TLS compression and SPDY at the time of its discovery. Furthermore, browsers such as Firefox, Chrome, and possibly others immediately took aims (thanks to advance warning from Rizzo and Duong) to mitigate the vulnerability by disabling header compression [6]. The current specifications of HTTP/2.0 still involve header compression, however, potentially leaving users vulnerable to the exploit.

An attacker using CRIME "needs to be able to observe your network traffic and to be able to cause many arbitrary requests to be sent," according to Adam Langley, who was involved in patching Chrome against the crime [10]. While these may seem like relatively unlikely conditions for most attackers, Matthew Green of Johns Hopkins University, in an interview with Ars Technica, posited that, "The CRIME attack is the nation-state attack. […] It's really something that Iran is going to do to try to find dissidents or China is going to do for the same reason. And it's a big deal because of that" [6]. CRIME is a significant enough vulnerability that HTTP/2.0 ought to take countermeasures against it beyond trusting browsers to provide sufficient patches for prevention.

3 Rizzo and Duong presented their findings at the 2012 Ekoparty Security Conference in Buenos Aires. Unfortunately, their paper could not be found online by this author at the time of writing. See [16] for their demonstration of the attack as performed against Stripe.

Please see the accompanying material for a brief demonstration of how CRIME can crack compressed cookies.

3.2 Multiplexing

Multiplexing, as mentioned previously, allows there to be multiple streams simultaneously open and active across the same connection. While multiplexing has been available for the Transmission Control Protocol (TCP), which provides error-checking of data sent between computers and protocols and is typically used by HTTP in the transport layer, since 1981, it was not implemented for HTTP/1.1 [4, 17]. It is, however already a significant component of SPDY. HTTP/2.0 will most likely adapt the SPDY implementation.

Multiplexing does not currently appear to introduce any new security considerations. Nevertheless, it may be worth it to investigate whether the increase in streams over the same connection increases opportunities for attackers to exploit vulnerabilities in TLS or any other form of encryption that HTTP/2.0 users implement. Most of the Web sites that have already adopted SPDY, such as Google and Twitter, are particularly large and more likely to have implemented TLS as securely as is presently known to be possible. Not all Web sites that use TLS have implemented it thus, however. For example, the Trustworthy Internet Movement claims that almost 70% of the Web sites they surveyed are still susceptible to the BEAST attack, which takes advantage of a vulnerability in TLS 1.0 [18]. It is conceivable that multiplexing, while not creating new vulnerabilities, may make it easier for attackers to take advantage of existing vulnerabilities.

3.3 Cross-Protocol Attacks

The current specifications are somewhat ambiguous about how HTTP/2.0 might affect vulnerability to cross-protocol attacks. The present draft says, “When using TLS, we believe that HTTP/2.0 introduces no new cross-protocol attacks. TLS encrypts the contents of all transmission (except the handshake itself), making it difficult for attackers to control the data which could be used in a cross-protocol attack,” only to add a bracketed comment afterwards which states, “Issue: This is no longer true” [2]. This author could find very little discussion of cross-protocol attacks in the discussions about HTTP/2.0, none of which explained the comment.

The current specifications of SPDY corroborate the original claim in the HTTP/2.0 specifications regarding cross-protocol attacks [1]. However, SPDY currently requires pervasive TLS, which HTTP/2.0 seems unlikely to do. The claim therefore is irrelevant for Web sites that may be implementing HTTP/2.0 without the protection of TLS. Might HTTP/2.0 introduce new cross-protocol vulnerabilities for Web sites that do not use TLS?

Furthermore, while TLS may make it harder for attackers to execute cross-protocol attacks, exploits such as the Wagner and Schneier attack demonstrate that it does not make it impossible [11]. HTTP/2.0 used in conjunction with TLS is likely to maintain all of the existing cross-protocol vulnerabilities, given that there do not seem to be any new attempts to defend against them.

3.4 Server Authority and Same Origin

HTTP/1.1 allows headers to apply same-origin policies, which require requests for Web resources to stem from the same origin, across DOM, XMLHttpRequest, and cookie requests [20]. However, same-origin policies are sometimes relaxed through policies that cross-origin

sharing requests (CORS) [19]. Same-origin policies are significant for limiting scripting and other exploits of a Web site or resource by someone from a different origin, which can involve injections, cross-site scripting (XSS), and other attacks.

The specifications of HTTP/2.0 suggest that the protocol will enforce an overarching same-origin policy. They warn that clients "MUST NOT use, in any way, resources provided by a server that is not authoritative for those resources" (emphasis in the original). The specifications describe server authorization as resolving the IP address to the same domain as the Web resource being requested, for clients that do not use TLS; and as ensuring that the request "has been successfully authenticated for the domain part of the origin of the resource that it is providing" for those that do. This seems to be a more proactive stance in support of same-origin policies than what is currently implemented for HTTP/1.1, as the default policy of the latter is ambiguous [2].

3.5 Intermediary Encapsulation Attacks

The HTTP/2.0 specifications suggest that translation from HTTP/2.0 to HTTP/1.1 may be vulnerable. Specifically, they explain that the backwards translation "could permit the creation of corrupted HTTP/1.1 messages," allowing attackers to "create HTTP/1.1 messages with illegal header fields, extra header fields, or even messages that are entirely falsified" [2]. Falsified headers could allow attacks such as XSS and web cache poisoning [9]. The backwards translation therefore appears to expose a significant vulnerability in HTTP/2.0, insofar as the translation will be necessary for users accessing HTTP/2.0 Web resources with older browsers, among other use cases.

The specifications argue that a differing process when going from HTTP/1.1 to HTTP/2.0 prevents this vulnerability in the forwards translation [2]. It may be worth exploring such an option for the backwards translation, or else limiting backwards translations from occurring at all. The latter might be possible if complete headers are not sent until after observing the client's HTTP support.

3.6 Cacheability of Pushed Resources

The specifications describe an area of potential vulnerability when servers that host multiple Web sites or other users allow these users to "push representations of resources that they do not have authority over." Since pushed resources do not involve explicit requests, they require identification by the server. Because of how cached resources sometimes get cached, permitting users to push resources to parts of the server for which they are not directly authorized would enable attackers to push versions of resources that, for example, contain embedded malware [2].

It is unclear to this author why HTTP/2.0 might make this vulnerability worse than it already is for HTTP/1.1. It may be that multiplexing could make tenant authorization more ambiguous, or that the working group simply wants to mitigate the vulnerability by insisting that servers are properly diligent about pushed resources.

3.7 Denial of Service Considerations

Denial of service (DoS) attacks limit the availability of Web resources. For example, they may cause bandwidth exhaustion such that a web server is rendered inaccessible by legitimate users. The HTTP/2.0 specifications warn that multiplexing may increase DoS vulnerabilities since the connection type "can demand a greater commitment of resources to operate than a

HTTP/1.1 connection." Furthermore, they suggest that frame requests may be manipulated so that servers waste extra time (and therefore also other resources) processing them [2].

While HTTP/2.0 increases the speed with which an attacker exhausts a server, DoS attacks are far from a new threat. The recommended defense for HTTP/2.0 clients against them is therefore that they track, monitor, and limit the usage of suspicious frames. This advice is similar to HTTP/1.1 protection against DoS attacks, which also involves monitoring and limiting requests that are repeated too frequently or are otherwise suspicious.

3.8 Privacy Considerations

To improve performance, HTTP/2.0 encourages keeping connections between clients and servers open longer than they typically are for HTTP/1.1. Attackers may gain access to confidential data by sniffing or otherwise observing connections that have been left open. While this is not an unknown issue in HTTP/1.1, longer-lasting connections provide attackers with better opportunities [2].

HTTP/2.0 servers might be encouraged to limit connections more than they might need to, perhaps somewhat limiting the performance benefits gained by the upgrade in favour of improved security. This advice is especially true of financial institutions and other servers that may be at higher risk for attacks or for which attacks would be more dangerous.

3.9 Encryption

I have already mentioned briefly that there is an ongoing debate as to whether HTTP/2.0 should be entirely encrypted and, if so, what type or types of encryption it should use. As the debate is still vehemently undecided I cannot offer any conclusions regarding the decision, and

will instead summarize a few points on either side of the argument as summarized in a Google Doc shared publicly amongst participants in the HTTP/2.0 community [15].[4]

The proponents of pervasive encryption espouse it mostly as a measure against pervasive surveillance. While privacy and measures to guarantee or improve privacy are laudable, and pervasive encryption would protect against more minor threats posed by the average attacker sniffing on coffee shop WiFi. However the recent NSA-related leaks demonstrate that TLS is insufficient to protect one's privacy from the strongest of prying eyes [14]. Claiming that TLS (or SSL) will guarantee individuals' privacy is disingenuous if not outright dangerous, as a false sense of security can prevent people or corporations from taking the necessary precautions against remaining threats and vulnerabilities, as the opponents of pervasive security point out. Opponents furthermore point out that TLS relies on the security of the authorities providing the certificates. If one of these is compromised, all of the servers that use their certificates will likewise be compromised – possibly while still thinking that they are secure.

Proponents and opponents also argue about the effects of the cost of security certificates required for encryption. The latter point out that the expense will represent yet another barrier to entry for many HTTP/2.0 clients whose services do not involve confidential data (blogs with comments by anonymous users only, for example), and that these clients will therefore be forced to stay with HTTP/1.1. Proponents, on the other hand, argue that more clients would prefer to use certificates if not for the cost, and that more widespread use of certificates will drive the

4 The Google Doc provides a description of the major reasonings for and against pervasive encryption primarily held in the thread begun at [13], in addition to numerous thread available in the mailing list at http://lists.w3.org/Archives/Public/ietf-http-wg/. I have chosen to focus on the Google Doc as it represents a more concise collaboration of opinions than do a series of posts by individuals in the mailing list.

price down for everyone. Clients will therefore no longer need to limit their online services for the sake of limiting the data they have available online.

Other arguments concern the legality and user experience of pervasive versus selective encryption. However, this argue would argue that selective security is better than a false sense of security. So long as TLS and other methods of protocol encryption available today are fallible, the HTTP protocol as a whole should not be tied to their fallibility.

4. Conclusion

The most pivotal security decision regarding HTTP/2.0 – the question of whether it should include pervasive or selective encryption and, in either situation, what type or types of encryption it should support – is still undecided. Regardless of its outcome, however, HTTP/2.0 seems to promise a relatively secure upgrade from HTTP/1.1: that is, the vulnerabilities that HTTP/2.0 introduces and continues from HTTP/1.1 do not seem to be urgent new issues and, for the most part, can be mitigated by browsers and servers that take the proper measures. With strong support with most of the Web's biggest players, particularly Google, HTTP/2.0 should improve the quality of the modern Web to a greater degree than any new risks it poses – so long as participants are aware of the risks that HTTP/2.0 may pose.

Works Cited

[1] Belshe, M., and R. Peon. "SPDY Protocol." *IETF TOOLS: Network Working Group*. IETF Tools Team, 4 Aug. 2012. Web. 13 Dec. 2013. <http://tools.ietf.org/html/draft-mbelshe-httpbis-spdy-00>.

[2] Belshe, M., R. Peon, M. Thomson, and A. Melnikov. "Hypertext Transfer Protocol Version 2.0." *IETF Tools: HTTPbis Working Group*. IETF Tools Team, 9 Dec. 2013. Web. 13 Dec. 2013. <http://http2.github.io/http2-spec/index.html>.

[3] "Can I use SPDY networking protocol?" *Can I use. Can I use.* Web. 13 Dec. 2013. < http://caniuse.com/spdy>.

[4] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Mastiner, P. Leach, T. Berners-Lee. "Hypertext Transfer Protocol -- HTTP/1.1." *W3C*. The Internet Society, June 1999. Web. 13 Dec. 2013. <http://www.w3.org/Protocols/rfc2616/rfc2616.html>.

[5] Finley, Klint. "Facebook Makes Itself a Bit More SPDY." *Conde Naste Digital*. Wired.com. Web. <http://www.wired.com/wiredenterprise/2013/03/facebook-spdy>.

[6] Goodin, Dan. "Crack in Internet's foundation of trust allows HTTPS session hijacking.". *Ars Technica*. Ars Technica, 13 Sept. 2012. Web. 13 Dec. 2013. <http://arstechnica.com/security/2012/09/crime-hijacks-https-sessions/>.

[7] "Hypertext Transfer Protocol Bis (httpbis): Charter for Working Group." *IETF Datatracker*. IETF Datatracker. Web. 13 Dec. 2013. <http://datatracker.ietf.org/wg/httpbis/charter/>.

[8] "HTTP/2.0." *HTTP Working Group. HTTP Working Group*. Web. 13 Dec. 2013. <http://http2.github.io/>.

[9] "Interpreter Injection." *OWASP*. OWASP, 12 May 2013. Web. 13 Dec. 2013. <https://www.owasp.org/index.php/Interpreter_Injection>.

[10] Langley, Adam. "CRIME." Weblog post. *Imperial Violet*. 21 Sept. 2012. Weblog psot. 13 Dec. 2013. <https://www.imperialviolet.org/2012/09/21/crime.html>.

[11] Mavrogiannopoulos, N. and B. Preneel. "Preventing cross-protocol attacks on the TLS protocol." *IETF Tools: Network Working Group*. IETF Tools Team, 9 November 2012. Web. 13 Dec. 2013. <http://tools.ietf.org/search/draft-mavrogiannopoulos-tls-cross-protocol-03>.

[12] Nielsen, Henrik Frystyk. "HTTP 2.0 and a Faster, more Mobile-friendly web."

IETF-HTTP-WG. W3C, 29 July 2012. 13 Dec. 2013. <http://lists.w3.org/Archives/Public/ietf-http-wg/2012JulSep/0855.html>.

[13] Nottingham, Mark. "Moving forward on improving HTTP's security." *IETF-HTTP-WG.* W3C, 13 Nov. 2013. 13 Dec. 2013. <http://lists.w3.org/Archives/Public/ietf-http-wg/2013OctDec/0625.html>.

[14] Perliroth, Nicole, Jeff Larson, and Scott Shane. "N.S.A. Able t Foil Basic Safeguards of Privacy on Web." *New York Times*. The New York Times Company, 5 Sept. 2013. Web. 13 Dec. 2013 <http://www.nytimes.com/2013/09/06/us/nsa-foils-much-internet-encryption.html?_r=0>.

[15] "Pervasive Privacy Pros and Cons." Google Doc. *IETF-HTTP-WG*. Google. Web. 13 Dec. 2013. <https://docs.google.com/document/d/10bgFxzaXFf24q-BR6lOxYJXi-ObN0bhBfHrfTVIMPj4/edit?pli=1#>.

[16] Rizzo, Juliao and Thai Duong. "CRIME vs startups." Online video clip. *YouTube.* YouTube, 14 Sept. 2012. Web. <http://www.youtube.com/watch?v=gGPhHYyg9r4&feature=youtu.be>.

[17] "Transmission Control Protocol." *Internet Engineering Task Force*. IETF, Sept. 1981. Web. 13 Dec. 2013. <http://www.ietf.org/rfc/rfc793.txt>.

[18] "SSL Pulse." *Trustworthy Internet Movement*. Trustworthy Internet Movement, 2 Dec. 2013. Web. 13 Dec. 2013. <https://www.trustworthyinternet.org/ssl-pulse>.

[19] Van Kesteren, Anne. "Cross-Origin Resource Sharing." *W3C*. W3C, 5 Dec. 2013. Web. 13 Dec. 2013. <http://www.w3.org/TR/cors/>.

[20] Zalewski, Michael. "Browser Security Handbook, part 2." *Google*. Google, 2009. Web. 13 Dec. 2013. <http://code.google.com/p/browsersec/wiki/Part2>.