

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 5, 2013

J. Hodges
PayPal
Feb 2013

Web Security Framework: Problem Statement and Requirements
draft-ietf-websec-framework-reqs-00

Abstract

Web-based malware and attacks are proliferating rapidly on the Internet. New web security mechanisms are also rapidly growing in number, although in an incoherent fashion. This document provides a brief overview of the present situation and the various seemingly piece-wise approaches being taken to mitigate the threats. It then provides an overview of requirements as presently being expressed by the community in various online and face-to-face discussions.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 5, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Where to Discuss This Draft	4
2. Document Conventions	5
3. Overall Constraints	5
4. Overall Requirements	6
5. Vulnerabilities, Attacks, and Threats	8
5.1. Attacks	8
5.2. Threats	9
6. Use Cases	9
7. Detailed Functional Requirements	11
8. Extant Policies to Coalesce	15
9. Example Concrete Approaches	15
10. Security Considerations	15
11. References	15
12. Informative References	19
Appendix A. Acknowledgments	21
Appendix B. Discussion References	21
B.1. Source: Attacks and Threats	21
B.2. Source: Policy Expression Syntax [1]	21
B.3. Source: Policy Expression Syntax [2]	22
B.4. Source: Tooling	22
B.5. Source: Performance	23
B.6. Source: Granularity	23
B.7. Source: Notifications and Reporting	23
B.8. Source: Facilitating Separation of Duties	24
B.9. Source: Hierarchical Policy Application	24
B.10. Source: Framing Policy Hierarchy, cross-origin, granularity	24
B.11. Source: Policy Delivery [1]	26
B.12. Source: Policy Delivery [2]	26
B.13. Source: Policy Conflict Resolution	27
Author's Address	28

1. Introduction

Over the past few years, we have seen a proliferation of AJAX-based web applications (AJAX being shorthand for asynchronous JavaScript and XML), as well as Rich Internet Applications (RIAs), based on so-called Web 2.0 technologies. These applications bring both luscious eye-candy and convenient functionality--e.g. social networking--to their users, making them quite compelling. At the same time, we are seeing an increase in attacks against these applications and their underlying technologies [1]. The latter include (but aren't limited to) Cross-Site-Request Forgery (CSRF) -based attacks [2], content-sniffing cross-site-scripting (XSS) attacks [3], attacks against browsers supporting anti-XSS policies [4], clickjacking attacks [5], malvertising attacks [6], as well as man-in-the-middle (MITM) attacks against "secure" (e.g. Transport Layer Security (TLS/SSL)-based [7]) web sites along with distribution of the tools to carry out such attacks (e.g. sslstrip) [8].

During the same time period we have also witnessed the introduction of new web security indicators, techniques, and policy communication mechanisms sprinkled throughout the various layers of the Web and HTTP. We have a new cookie security flag called HTTPOnly [9]. We have the anti-clickjacking X-Frame-Options HTTP header [10], the Strict-Transport-Security HTTP header [RFC6797], anti-CSRF headers (e.g. Origin) [12], an anti-sniffing header (X-Content-Type-Options: nosniff) [13], various approaches to content restrictions [14] [15] and notably Mozilla Content Security Policy (CSP; conveyed via a HTTP header) [16], the W3C's Cross-Origin Resource Sharing (CORS; also conveyed via a HTTP header) [17], as well as RIA security controls such as the crossdomain.xml file used to express a site's Adobe Flash security policy [18]. There's also the Application Boundaries Enforcer (ABE) [19], included as a part of NoScript [20], a popular Mozilla Firefox security extension. Sites can express their ABE rule-set at a well-known web address for downloading by individual clients [21], similarly to Flash's crossdomain.xml. Amidst this haphazard collage of new security mechanisms at least one browser vendor has even devised a new HTTP header that disables one of their newly created security features: witness the X-XSS-Protection header that disables the new anti-XSS features [22] in Microsoft's Internet Explorer 8 (IE8).

Additionally, there are various proposals aimed at addressing other facets of inherent web vulnerabilities, for example: JavaScript postMessage-based mashup communications [23], hypertext isolation techniques [24], and service security policies advertised via the Domain Name System (DNS) [25]. Going even further, there are efforts to redesign web browser architectures [26], of which Google Chrome and IE8 are deployed examples. An even more radical approach is

exhibited in the Gazelle Web Browser [27], which features a browser kernel embodied in a multi-principal OS construction providing cross-principal protection and fair sharing of all system resources.

Not to be overlooked is the fact that even though there is a plethora of "standard" browser security features--e.g. the Same Origin Policy (SOP), network-related restrictions, rules for third-party cookies, content-handling mechanisms, etc. [28]--they are not implemented uniformly in today's various popular browsers and RIA frameworks [29]. This makes life even harder for web site administrators in that allowances must be made in site security posture and approaches in consideration of which browser a user may be wielding at any particular time.

Although industry and researchers collectively are aware of all the above issues, we observe that the responses to date have been issue-specific and uncoordinated. What we are ending up with looks perhaps similar to Frankenstein's monster [30]--a design with noble intents but whose final execution is an almost-random amalgamation of parts that do not work well together. It can even cause destruction on its own [31].

Thus, the goal of this document is to define the requirements for a common framework expressing security constraints on HTTP interactions. Functionally, this framework should be general enough that it can be used to unite the various individual solutions above, and specific enough that it can address vulnerabilities not addressed by current solutions, and guide the development of future mechanisms.

Overall, such a framework would provide web site administrators the tools for managing, in a least privilege [33] manner, the overall security characteristics of their web site/applications when realized in the context of user agents.

[[The author(s) understand that many of the references to web security issues are now somewhat dated and more recent work has appeared in the literature. Suggestions of references to use in superseding the ones herein are welcome; references to web security survey papers would be good.]]

1.1. Where to Discuss This Draft

Please discuss this draft on the websec@ietf.org mailing list [WebSec].

2. Document Conventions

NOTE: ..is a note to the reader. These are points that should be expressly kept in mind and/or considered.

[[TODO_n: Things to fix (where "n" in "TODO_n" is a number). --JeffH]]

We will also be making use of the WebSec WG issue tracker, so use of the TODO marks will evolve accordingly.

3. Overall Constraints

Regardless of the overall approaches chosen for conveying site security policies, we believe that to be deployed at Internet-scale, and to be as widely usable as possible for both novice and expert alike, the overall solution approach will need to address these three points of tension:

Granularity:

There has been much debate during the discussion of some policy mechanisms (e.g. CSP) as to how fine-grained such mechanisms should be. The argument against fine-grained mechanisms is that site administrators will cause themselves pain by instantiating policies that do not yield the intended results. E.g. simply copying the expressed policies of a similar site. The claim is that this would occur for various reasons stemming from the mechanisms' complexity [34].

Configurability:

Not infrequently, the complexity of underlying facilities, e.g. in server software, is not well-packaged and thus administrators are obliged to learn more about the intricacies of these systems than otherwise might be necessary. This is sometimes used as an argument for "dumbing down" the capabilities of policy expression mechanisms [34].

Usability:

Research shows that when security warnings are displayed, users are often given too much information as well as being allowed to relatively easily bypass the warnings and continue with their potentially compromising activity [35] [36] [37] [38] [39]. Thus users have become trained to "click through" security notifications "in order to get work done", though not infrequently rendering themselves insecure and perhaps

compromised [40].

In the next section we discuss various high-level requirements derived with the guidance of the latter tension points.

4. Overall Requirements

1. Policy conveyance:

in-band:

HTTP header(s) are already presently used to convey policy to user agents. However, devising generalized, extensible HTTP security header(s) such that the on-going "bloat" of the number of disjoint HTTP security headers is mitigated, is a stated requirement by various parties. Also, then there would be a documented framework that can be leveraged as new approaches and/or threats emerge.

It may be reasonable to devise distinct sets of headers to convey different classes of policies, e.g., web application content policies (such as [W3C.CR-CSP-20121115]) versus web application network connection policies (such as [RFC6797]).

out-of-band:

An out-of-band policy communication mechanism must be secure and should have two facets, one for communicating securely out-of-band of the HTTP protocol to allow for secure client policy store bootstrapping. potential approaches are factory-installed web browser configuration, site security policy download a la Flash's crossdomain.xml and Maone's ABE for Web Authors [21], and DNS-based policy advertisement leveraging the security of the Secure DNS (DNSSEC) [32].

NOTE: The distinction between in-band and out-of-band signaling is difficult to characterize because some seemingly out-of-band mechanisms rely on the same protocols (HTTP/HTTPS) and infrastructure (e.g., transparent proxy servers) as the protocols they ostensibly protect.

2. Granularity:

In terms of granularity, vast arrays of stand-alone blog, wiki, hosted web account, and other "simple" web sites could

ostensibly benefit from relatively simple, pre-determined policies. However, complex sites--e.g. payment, ecommerce, software-as-a-service, mashup sites, etc.--often differ in various ways, as well as being inherently complex implementation-wise. One-size-fits-all policies will generally not work well for them.

Thus, to be effective for a broad array of web site and application types, some derived requirements are:

the policy expression mechanism should fundamentally facilitate fine-grained control. For example, CSP [W3C.CR-CSP-20121115] offers such control.

In order to address the less complex needs of the more simple classes of web sites, the policy expression mechanism should have some facility for enabling "canned policy profiles".

In addition, the configuration facilities of various components of the web infrastructure can be enhanced to provide an appropriately simple veneer over the complexity.

3. Configurability:

With respect to configurability, development effort should be applied to creating easy-to-use administrative interfaces addressing the simple cases, like those mentioned above, while providing advanced administrators the tools to craft and manage fine-grained multi-faceted policies. Thus more casual or novice administrators can be aided in readily choosing, or be provided with, safe default policies while other classes of sites have the tools to craft the detailed policies they require. Examples of such an approach are Microsoft's "Packaging Wizard" [41] that easily auto-generates a quite complicated service deployment descriptor on behalf of less experienced administrators, and Firefox's simple Preferences dialog [42] as compared to its detailed about:config configuration editor page [43]. In both cases, simple usage by inexperienced users is anticipated and provided for on one hand, while complex tuning of the myriad underlying preferences is provided for on the other.

4. Usability:

Much has been learned over the last few years about what does and does not work with respect to security indicators in web browsers and web pages, as noted above, these lessons should

be applied to the security indicators rendered by new proposed security mechanisms. We believe that in cases of user agents venturing into insecure situations, their response should be to fail the connections by default without user recourse, rather than displaying warnings along with bypass mechanisms, as is current practice. For example, the Strict Transport Security specification [RFC6797] suggests the former so-called "hard-fail" behavior.

5. Vulnerabilities, Attacks, and Threats

This section enumerates vulnerabilities and attacks of concern, and then illustrates plausible threats that could result from exploitation of the vulnerabilities. The intent is to illustrate threats that ought to be mitigated by a web security policy framework.

The definitions of vulnerability, attack, and threat used in this document are based on the definitions from [RFC4949], and are paraphrased here as:

- Vulnerability:** A flaw or weakness in a system's design, implementation, or operation and management that could be exploited.
- Attack:** An intentional act of vulnerability exploitation, usually characterized by one or more of: the method or technique used, and/or the point of initiation, and/or the method of delivery, etc. For example: active attack, passive attack, offline attack, Cross-site Scripting (XSS) attack, SQL injection attack, etc.
- Threat:** Any circumstance or event with the potential to adversely affect a system and its user(s) through unauthorized access, destruction, disclosure, or modification of data, or denial of service.

See also Appendix B.1 Source: Attacks and Threats.

5.1. Attacks

These are some of the attacks which are desirable to mitigate via a web application security framework (see [WASC-THREAT] for a more complete taxonomy of attacks):

1. Cross-site-scripting (XSS) [2] [WASC-THREAT-XSS]
2. Cross-Site-Request Forgery (CSRF) [WASC-THREAT-CSRF]
3. Response Splitting [WASC-THREAT-RESP]
4. User Interface Redressing [UIRedress], aka Clickjacking [Clickjacking].
5. Man-in-the-middle (MITM) attacks against "secure" web applications, i.e., ones accessed using TLS/SSL [RFC5246] [WASC-THREAT-TLS] [SSLSTRIP].
6. [[TODO2: more? (e.g. from [WASC-THREAT] ?) --JeffH]]

5.2. Threats

Via attacks exploiting the vulnerabilities above, an attacker can..

1. Obtain a victim's confidential web application credentials (e.g., via cookie theft), and use the credentials to impersonate the victim and enter into transactions, often with the aim of monetizing the transaction results to the attacker's benefit.
2. Insert themselves as a Man-in-the-Middle (MITM) between victim and various services, thus is able to instigate, control, intercept, and attempt to monetize various transactions and interactions with web applications, to the benefit of the attacker.
3. Enumerate various user agent information stores, e.g. browser history, facilitating views of the otherwise confidential habits of the victim. This information could possibly be used in various offline attacks against the victim directly. E.g.: blackmail, denial of services, law enforcement actions, etc.
4. Use gathered information and credentials to construct and present a falsified persona of the victim (e.g. for character assassination).

There is a risk of exfiltration of otherwise confidential victim information with all the threats listed above.

6. Use Cases

This section outlines various example use cases.

1. I'm a web application site administrator. My web app includes static user-supplied content (e.g. submitted from user agents via HTML FORM + HTTP POST), but either my developers don't properly sanitize user-supplied content in all cases or/and content injection vulnerabilities exist or materialize (for various reasons).

This leaves my web app vulnerable to cross-site scripting. I wish I could set overall web app-wide policies that prevent user-supplied content from injecting malicious content (e.g. JavaScript) into my web app.

2. I'm a web application site administrator. My web application is intended, and configured, to be uniformly served over HTTPS, but my developers mistakenly keep including content via insecure channels (e.g. via insecure HTTP; resulting in so-called "mixed content").

I wish I could set a policy for my web app that prevents user agents from loading content insecurely even if my web app is otherwise telling them to do so.

3. I'm a web application site administrator. My site has a policy that we can only include content from certain trusted providers (e.g., our CDN, Amazon S3), but my developers keep adding dependencies on origins I don't trust. I wish I could set a policy for my site that prevents my web app from accidentally loading resources outside my whitelist.
4. I'm a web application site administrator. I want to ensure that my web app is never framed by other web apps.
5. I'm a developer of a web application which will be included (i.e. framed) by third parties within their own web apps. I would like to ensure that my web app directs user agents to only load resources from URIs I expect it to (possibly even down to specific URI paths), without affecting the containing web app or any other web apps it also includes.
6. I'm a web application site administrator. My web app frames other web apps whose behavior, properties, and policies are not 100% known or predictable.

I need to be able to apply policies that both protect my web app from potential vulnerabilities or attacks introduced by the framed web apps, and that work to ensure that the desired interactions between my web app and the framed apps are securely realized.

7. [[TODO3: additional use cases to add? --JeffH]]

7. Detailed Functional Requirements

Many of the below functional requirements are extracted from a discussion on the [public-web-security] mailing list (in early 2011). Particular messages are cited inline and appropriate quotes extracted and reproduced here. Inline citations are provided for definitions of various terms.

The overall functional requirement categories are:

1. Policy mechanism scope
2. Policy expression syntax
3. Tooling
4. Performance
5. Granularity
6. Notifications and reporting
7. Facilitating Separation of Duties
8. Hierarchical Policy Application
9. Policy Delivery
10. Policy Conflict Resolution

[[TODO4: additional functional requirement categories to add? --JeffH]]

These requirements are discussed in more detail below:

1. Policy mechanism scope and extensibility:

There is a current proliferation of orthogonal atomic mechanisms intended to solve very specific problems. Web developers have a hard enough time with security already without being expected to master a potentially large number of different security mechanisms, each with their own unique threat model, implementation and syntax. Not to mention trying to figure out how they're expected to interact with each other; e.g., how to manage the gaps and intersections

between the models.

Thus the desire to have an extensible security policy mechanism that could evolve as the web evolves, and the threats that the web faces also evolve. For example, an extensibility model similar to HTML where the security protections could grow over time.

See also Appendix B.2 Source: Policy Expression Syntax [1].

2. Policy expression syntax:

The policy expression syntax for a web security framework should be declarative [DeclLang] (and extensible, as noted above). This is for simplicity sake, as the alternative to declarative is procedural/functional, e.g., the class of language ECMAScript falls into.

See also Appendix B.2 Source: Policy Expression Syntax [1], and, Appendix B.3 Source: Policy Expression Syntax [2].

3. Tooling:

We will need tools to (ideally) analyze a web application and generate an initial security policy.

See also Appendix B.4 Source: Tooling.

4. Performance:

Minimizing performance impact is a first-order concern.

See also Appendix B.5 Source: Performance.

5. Granularity:

For example, discriminate between:

- + "inline" script in <head> versus <body>, or not.
- + "inline" script and "src=" loaded script.
- + Classes of "content", e.g. scriptable content, passive multimedia, nested documents, etc.

See also Appendix B.6 Source: Granularity.

6. Notifications and Reporting:

Convey to the user agent an identifier (e.g. a URI) denoting where to send policy violation reports. Could also specify a DOM event to be dedicated for this purpose.

An ability to specify that a origin's policies are to be enforced in a "report only" mode will be useful for debugging policies as well as site-policy interactions. E.g. for answering the question: "does my policy 'break' my site?".

See also Appendix B.7 Source: Notifications and Reporting.

7. Facilitating Separation of Duties:

Specifically, allowing for Web Site operations/deployment personnel to apply site policy, rather than having it being encoded in the site implementation code by site developers/implementors.

See also Appendix B.8 Source: Facilitating Separation of Duties.

8. Hierarchical Policy Application:

The notion that policy emitted by the application's source origin is able to constrain behavior and policies of contained origins.

See also Appendix B.9 Source: Hierarchical Policy Application.

9. Framing Policy Hierarchy, cross-origin, granularity, auditability, roles:

[[TODO5: Need more fully coalesce the source info from appendix into this item. --JeffH]]

- + "Framing" is a client-side instance notion, where webapp1's client-side instance (aka "document") loads another webapp, "webapp2", into an HTML <IFRAME> element.
- + A webapp may wish to never be framed by another webapp.
- + webapps are denoted by "origins" [RFC6454].

- + an origin can emit policy (i.e. from the server-side webapp component) to the user agent (i.e. the execution environment/container for the client-side webapp component) in at least two fashions: HTML element attributes, HTTP header fields, ecmaScript code. See also Paragraph 10.
- + Policy expressed via HTML <IFRAME> elements is "fine-grained" because the webapp can control such policies on iframe-by-iframe basis. Policies conveyed by HTTP header fields applies "document-wide" (i.e., to the webapp client-side instance) as a whole.
- + Note that either or both of the "framing" or "framed" webapp client-side instance may be an attacker (in which case the other webapp client-side instance would be considered a "victim" (whether or not its actual intentions are malicious or not)).

See also Appendix B.10 Source: Framing Policy Hierarchy, cross-origin, granularity.

10. Policy Delivery:

[[TODO6: Need more fully coalesce the source info from appendix into this item, and blend/resolve/contrast with above item. --JeffH]]

The web application policy must be communicated by the web application to the user agent. There are various approaches and they have tradeoffs between security, audience, and practicality.

See also Appendix B.11 Source: Policy Delivery [1], as well as, Appendix B.12 Source: Policy Delivery [2].

11. Policy Conflict Resolution:

[[TODO7: Need more fully coalesce the source info from appendix into this item. --JeffH]]

What is the model for resolving conflicts between policies expressed by "parent" and "child" webapps?

Should policies conveyed via HTTP header fields (i.e., that apply webapp-wide) be handled differently than those expressed by webapp client-side instances (e.g., via HTML elements and their attributes)?

See also Appendix B.13 Source: Policy Conflict Resolution.

8. Extant Policies to Coalesce

Presently, this section lists a grab-bag of individually-expressed web app security policies which a more general substrate could ostensibly encompass (in order to, for example, reduce "header bloat" and bytes-on-the-wire issues), as well as reduce functional policy duplication/overlap.

CORS

XDomainRequest

toStaticHtml

innerSafeHtml

X-Frame-Options

CSP frame-ancestors

more?

9. Example Concrete Approaches

An overall, broad approach (from [0]):

As for an overall policy mechanism, we observe that leveraging a combination of CSP [16] and ABE [19], or their employment in tandem, as a starting point for a multi-vendor approach may be reasonable. For a near-term policy delivery mechanism, we advocate use of both HTTP headers and a policy file at a well-known location. Leveraging DNSSEC is attractive in the intermediate term, i.e. as it becomes more widely deployed.

10. Security Considerations

Security considerations go here.

11. References

[[TODO1: re-code refs into xml and place in proper refs section.
--JeffH]]

- [0] J. Hodges, A. Steingruebl, "The Need for Coherent Web Security Policy Framework(s)", Web 2.0 Security & Privacy, Oakland CA, 20 May 2010. <http://w2spconf.com/2010/papers/p11.pdf>
- [1] Breach Security, "THE WEB HACKING INCIDENTS DATABASE 2009," Aug. 2009. http://www.breach.com/resources/whitepapers/downloads/WP_TheWebHackingIncidents-2009.pdf
- [2] R. Auger, The Cross-Site Request Forgery (CSRF/XSRF) FAQ, 2007. <http://www.cgisecurity.com/articles/csrf-faq.shtml>
- [3] A. Barth, J. Caballero, and D. Song, "Secure Content Sniffing for Web Browsers--or How to Stop Papers from Reviewing Themselves," Proceedings of the 30th IEEE Symposium on Security & Privacy, Oakland, CA: 2009.
- [4] D. Goodin, "Major IE8 flaw makes 'safe' sites unsafe - Microsoft's XSS buster busted," The Register, Nov. 2009. http://www.theregister.co.uk/2009/11/20/internet_explorer_security_flaw/
- [5] J. Grossman, "Clickjacking: Web pages can see and hear you," Oct. 2008. <http://jeremiahgrossman.blogspot.com/2008/10/clickjacking-web-pages-can-see-and-hear.html>
- [6] W. Salusky, Malvertising, 2007. <http://isc.sans.org/diary.html?storyid=3727>
- [7] T. Dierks and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC5246, Internet Engineering Task Force, Aug. 2008. <http://www.ietf.org/rfc/rfc5246.txt>
- [8] M. Marlinspike, SSLSTRIP, 2009. <http://www.thoughtcrime.org/software/sslstrip/>
- [9] Scope of HTTPOnly Cookies. http://docs.google.com/View?docid=dxqxgkd_0cvcqhsdw
- [10] E. Lawrence, IE8 Security Part VII: ClickJacking Defenses, 2009. <http://blogs.msdn.com/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx>
- [11] J. Hodges, C. Jackson, and A. Barth, "Strict Transport Security," Work-in-progress, Internet-Draft, Jul. 2010. <http://tools.ietf.org/html/draft-hodges-strict-transport-sec>
- [12] A. Barth, C. Jackson, and I. Hickson, "The Web Origin Concept," Internet-Draft, work in progress, Internet Engineering Task Force, 2009. <http://tools.ietf.org/html/draft-abarth-origin>

- [13] E. Lawrence, IE8 Security Part VI: Beta 2 Update, 2008. <http://blogs.msdn.com/ie/archive/2008/09/02/ie8-security-part-vi-beta-2-update.aspx>
- [14] G. Markham, Content restrictions, 2007. <http://www.gerv.net/security/content-restrictions/>
- [15] T. Jim, N. Swamy, and M. Hicks, "BEEP: Browser-Enforced Embedded Policies," Proceedings of the 16th International World Wide Web Conference, Banff, Alberta, Canada, 2007.
- [16] B. Sterne, "Content Security Policy (CSP)," 2011. <https://dvcs.w3.org/hg/content-security-policy/raw-file/bcf1c45f312f/csp-unofficial-draft-20110303.html>
- [17] A.V. Kesteren, "Cross-Origin Resource Sharing (CORS)," Mar. 2009. <http://www.w3.org/TR/2009/WD-cors-20090317/>
- [18] Adobe Systems, "Cross-domain policy file specification." http://learn.adobe.com/wiki/download/attachments/64389123/CrossDomain_PolicyFile_Specification.pdf?version=1
- [19] G. Maone, ABE - Application Boundaries Enforcer, 2009. <http://noscript.net/abe/>
- [20] G. Maone, NoScript. <http://noscript.net/>
- [21] G. Maone, ABE for Web Authors, 2009. <http://noscript.net/abe/web-authors.html>
- [22] Microsoft, "Event 1046 - Cross-Site Scripting Filter," MSDN Library, undated. <http://msdn.microsoft.com/en-us/library/dd565647%28VS.85%29.aspx>
- [23] A. Barth, C. Jackson, and W. Li, "Attacks on JavaScript Mashup Communication," Proceedings of the Web 2.0 Security and Privacy Workshop, 2009.
- [24] M. Ter Louw, P. Bisht, and V. Venkatakrisnan, "Analysis of Hypertext Isolation Techniques for XSS Prevention," Proceedings of the Web 2.0 Security and Privacy Workshop, 2008 .
- [25] A. Ozment, S.E. Schechter, and R. Dhamija, "Web Sites Should Not Need to Rely on Users to Secure Communications," W3C Workshop on Transparency and Usability of Web Authentication, 2006.
- [26] C. Reis, A. Barth, and C. Pizano, "Browser Security: Lessons from Google Chrome," ACM Queue, 2009, pp. 1-8.

- [27] H.J. Wang, C. Grier, A. Moshchuk, S.T. King, P. Choudhury, and H. Venter, "The Multi-Principal OS Construction of the Gazelle Web Browser," USENIX Security Symposium, 2009.
- [28] M. Zalewski, Browser Security Handbook.
<http://code.google.com/p/browsersec/>
- [29] A. Stamos, D. Thiel, and J. Osborne, Living in the RIA World: Blurring the Line between Web and Desktop Security, BlackHat presentation, iSecPartners, 2008.
https://www.isecpartners.com/files/RIA_World_BH_2008.pdf
- [30] Mary Shelley, "Frankenstein, or The Modern Prometheus," ca. 1831. http://en.wikipedia.org/wiki/Frankenstein%27s_monster
- [31] D. Goodin, "cPanel, Netgear and Linksys susceptible to nasty attack - Unholy Trinity," The Register, 2009.
http://www.theregister.co.uk/2009/08/02/unholy_trinity_csrf/
- [32] R. Arends, R. Austein, M. Larson, D. Massey, and S. Rose, "DNS security introduction and requirements," RFC4033, Internet Engineering Task Force, Mar. 2005.
<http://www.ietf.org/rfc/rfc4033.txt>
- [33] J.H. Saltzer and M.D. Schroeder, "The Protection of Information in Computer Systems," Communications of the ACM, vol. 17, Jul. 1974.
- [34] I. Hickson and many others, "Comments on the Content Security Policy specification," discussion on mozilla.dev.security newsgroup.
http://groups.google.com/group/mozilla.dev.security/browse_frm/thread/87ebe5cb9735d8ca?tvc=1&q=Comments+on+the+Content+Security+Policy+specification
- [35] S. Egelman, L.F. Cranor, and J. Hong, "You've Been Warned: An Empirical Study of the Effectiveness of Web Browser Phishing Warnings," CHI 2008, April 5 - 10, 2008, Florence, Italy, 2008.
- [36] S.E. Schechter, R. Dhamija, A. Ozment, and I. Fischer, "The Emperor's New Security Indicators," Proceedings of the 2007 IEEE Symposium on Security and Privacy.
- [37] R. Dhamija and J.D. Tygar, "The Battle Against Phishing: Dynamic Security Skins," Proceedings of the 2005 Symposium on Usable Privacy and Security (SOUPS).
- [38] J. Sobey, T. Whalen, R. Biddle, P.V. Oorschot, and A.S. Patrick, Browser Interfaces and Extended Validation SSL Certificates: An

Empirical Study, Ottawa, Canada: School of Computer Science, Carleton University, 2009.

[39] J. Sunshine, S. Egelman, H. Almuhiemedi, N. Atri, and L.F. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness," USENIX Security Symposium, 2009.

[40] C. Jackson and A. Barth, "ForceHTTPS: Protecting High-Security Web Sites from Network Attacks," Proceedings of the 17th International World Wide Web Conference (WWW), 2008.

[41] Microsoft, "Packaging Wizard."
[http://msdn.microsoft.com/en-us/library/aa157732\(officed.10\).aspx](http://msdn.microsoft.com/en-us/library/aa157732(officed.10).aspx)

[42] Mozilla, "Options window."
<http://support.mozilla.com/en-US/kb/Options+window>

[43] S. Yegulalp, "Hacking Firefox: The secrets of about:config," ComputerWorld, May. 2007. http://www.computerworld.com/s/article/9020880/Hacking_Firefox_The_secrets_of_about_config

12. Informative References

[Clickjacking]

"Clickjacking", Sep 2008,
<<http://www.sectheory.com/clickjacking.htm>>.

[DeclLang]

"declarative languages", A Dictionary of Computing Encyclopedia.com, 2004, <<http://www.encyclopedia.com/doc/1011-declarativelanguages.html>>.

[RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.

[RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

[RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, November 2012.

[SSLSTRIP]

Marlinspike, M., "SSLSTRIP", 2009,
<<http://www.thoughtcrime.org/software/sslstrip/>>.

- [UIRedress] "Dealing with UI redress vulnerabilities inherent to the current web", Sep 2008, <<http://lists.whatwg.org/htdig.cgi/whatwg-whatwg.org/2008-September/016284.html>>.
- [W3C.CR-CSP-20121115] Sterne, B. and A. Barth, "Content Security Policy 1.0", World Wide Web Consortium CR CR-CSP-20121115, November 2012, <<http://www.w3.org/TR/2012/CR-CSP-20121115>>.
- [WASC-THREAT] Web Application Security Consortium, "The WASC Threat Classification v2.0", January 2010, <http://projects.webappsec.org/f/WASC-TC-v2_0.pdf>.
- [WASC-THREAT-CSRF] Web Application Security Consortium, "Cross Site Request Forgery", The WASC Threat Classification v2.0 Reference ID: WASC-9, January 2010, <<http://projects.webappsec.org/w/page/13246919/Cross%20Site%20Request%20Forgery>>.
- [WASC-THREAT-RESP] Web Application Security Consortium, "HTTP Response Splitting", The WASC Threat Classification v2.0 Reference ID: WASC-25, January 2010, <<http://projects.webappsec.org/w/page/13246931/HTTP%20Response%20Splitting>>.
- [WASC-THREAT-TLS] Web Application Security Consortium, "Insufficient Transport Layer Protection", The WASC Threat Classification v2.0 Reference ID: WASC-04, January 2010, <<http://projects.webappsec.org/w/page/13246945/Insufficient%20Transport%20Layer%20Protection>>.
- [WASC-THREAT-XSS] Web Application Security Consortium, "Cross Site Scripting", The WASC Threat Classification v2.0 Reference ID: WASC-8, January 2010, <<http://projects.webappsec.org/w/page/13246920/Cross%20Site%20Scripting>>.
- [WebSec] "Web HTTP Application Security Minus Authentication and Transport", <<https://www.ietf.org/mailman/listinfo/websec>>.
- [public-web-security] "public-web-security@w3.org: Improving standards and implementations to advance the security of the Web.",

<<http://lists.w3.org/Archives/Public/public-web-security/>>.

Appendix A. Acknowledgments

Text and ideas were prototypically incorporated from various mailing list discussions, notably the ones on the [public-web-security] mailing list in early 2011, into this document. The authors wish to acknowledge and thank these individuals in particular for their tacit contributions to this document: Lucas Adamski, Adam Barth, gaz Heyes, Andrew Steingruebl, Brandon Sterne, Daniel Veditz, John Wilander.

Appendix B. Discussion References

B.1. Source: Attacks and Threats

In terms of defining threats in contrast to attacks:

<"Re: More on XSS mitigation (was Re: XSS mitigation in browsers)" (Lucas Adamski). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0089.html>>

"... There's a fundamental question about whether we should be looking at these problems from an attack vs threat standpoint. An attack is [exploitation of, ed.] XSS [or CSRF, or Response Splitting, etc.]. A threat is that an attacker could compromise a site via content injection to trick the user to disclosing confidential information (by injecting a plugin or CSS to steal data or fool the user into sending their password to the attacker's site). ..."

B.2. Source: Policy Expression Syntax [1]

In terms of policy expression syntax and extensibility, Lucas Adamski supplied this: <"Re: XSS mitigation in browsers" (Lucas Adamski). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0066.html>>

"On a conceptual level, I am not really a believer in the current proliferation of orthogonal atomic mechanisms intended to solve very specific problems. Security is a holistic discipline, and so I'm a big supporter of investing in an extensible declarative security policy mechanism that could evolve as the web and the threats that it faces do. Web developers have a hard enough time with security already without being expected to master a potentially large number of different security mechanisms, each

with their own unique threat model, implementation and syntax. Not to mention trying to figure out how they're expected to interact with each other... how to manage the gaps and intersections between the models."

B.3. Source: Policy Expression Syntax [2]

In terms of policy expression syntax and extensibility, Adam Barth supplied this: <"Re: Scope and complexity (was Re: More on XSS mitigation)" (Adam Barth). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0108.html>>

"I guess I wish we had an extensibility model more like HTML where we could grow the security protections over time. For example, we can probably agree that both <canvas> and <video> are great additions to HTML that might not have made sense when folks were designing HTML 1.0.

As long as you're not relying on the security policy as a first line of defense, the extensibility story for security policies is even better than it is with HTML tags. With an HTML tag, you need a fall-back for browsers that don't support the tag, whereas with a security policy, you'll always have your first line of defense.

Ideally, we could come up with a policy mechanism that let us nail XSS today and that fostered innovation in security for years to come. In the short term, you could view the existing CSP features (e.g., clickjacking protection) as the first wave of innovation. If those pieces are popular, then it should be easy for other folks to adopt them."

B.4. Source: Tooling

In terms of tooling needs, John Wilander supplied this: <"Re: More on XSS mitigation" (John Wilander). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0082.html>>

"*Developers Will Want a Policy Generator* A key issue for in-the-field success of CSP is how to write, generate and maintain the policies. Just look at the epic failure of Java security policies. The Java policy framework was designed for static releases shipped on CDs, not for moving code, added frameworks, new framework versions etc. The world of web apps is so dynamic I'm still amazed. If anything, for instance messy security policies, gets in the way of daily releases it's a no go. At least until there's an exploit. Where am I going with this? Well, we should implement a PoC *policy generator* and run it on some fairly large websites before we nail the standard. There

will be subtleties found which we can address and we can bring the PoC to production level while the standard is being finalized and shipped in browsers. Then we release the policy generator along with policy enforcement -- success! "

B.5. Source: Performance

In terms of performance, John Wilander supplied this: <"Re: More on XSS mitigation" (John Wilander). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0082.html>>

"*We Mustn't Spoil Performance* Web developers (and browser developers) are so hung up on performance that we really need to look at what they're up to and make sure we don't spoil things. Especially load performance now that it's part of Google's rating."

B.6. Source: Granularity

In terms of granularity, Daniel Veditz supplied this: <"Proposal to move the debate forward" (Daniel Veditz). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0122.html>>

"We oscillated several times between lumpy and granular. Fewer classes (simpler) is always more attractive, easier to explain and understand. The danger is that future features then end up being added to the existing lumps, possibly enabling things that the site isn't aware they need to now filter. It's a constant problem as we expand the capabilities of browsers -- sites that used to be perfectly secure are suddenly hackable because all the new browsers added feature-X."

B.7. Source: Notifications and Reporting

In terms of notifications and reporting, Brandon Sterne supplied this: <"[Content Security Policy] Proposal to move the debate forward" (Brandon Sterne). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0118.html>>

"...

3. Violation Reporting

- a. report-uri: URI to which a report will be sent upon policy violation
 - b. SecurityViolation event: DOM event fired upon policy violations
- ..."

B.8. Source: Facilitating Separation of Duties

In terms of facilitating separation of duties, Andrew Steingruebl supplied this: <"RE: Content Security Policy and iframe@sandbox" (Andrew Steingruebl). <http://lists.w3.org/Archives/Public/public-web-security/2011Feb/0050.html>>

"... 2. SiteC is also totally in control of all HTTP headers it emits. It could just as easily indicate policy choices for all frames via CSP. It could advertise a blanket policy (No JS, No ActiveX). Advertising a page-specific, or frame/target specific policy is substantially more difficult and probably unwieldy. But, depending on how SiteC is configured, setting a global site policy via headers offers a potential separation of duties that #1 does not, it allows website admin to specific things that each web developer might not be able to. ..."

B.9. Source: Hierarchical Policy Application

In terms of hierarchical policy application, Andrew Steingruebl supplied this: <"RE: Content Security Policy and iframe@sandbox" (Andrew Steingruebl). <http://lists.w3.org/Archives/Public/public-web-security/2011Feb/0048.html>>

"... I could imagine a tweak to CSP wherein CSP would control all contents hierarchically. I already spoke to Brandon about it, but it was just a quick brainstorm.

You could imagine revoking permissions in the frame hierarchy and not granting them back. This does start to get awfully ugly, but just as CSP controls loading policy for itself, it could also control loading policy for children, ..."

B.10. Source: Framing Policy Hierarchy, cross-origin, granularity

In terms of framing policy hierarchy, cross-origin, granularity, Andy Steingruebl and Adam Barth supplied this:

<"Re: Content Security Policy and iframe@sandbox" (Andy Steingruebl, Adam Barth) <http://lists.w3.org/Archives/Public/public-web-security/2011Feb/0051.html>>

On Sat, Feb 12, 2011 at 9:01 PM, Steingruebl, Andy
<asteingruebl@paypal-inc.com> wrote:
>> -----Original Message-----
>> From: Adam Barth [mailto:w3c@adambarth.com]

>
>> That all sounds very abstract. If you have some concrete examples,
>> that might be more productive to discuss. When enforcing policy
>> supplied by one origin on another origin, we need to be careful to
>> consider the case where the policy providing origin is the attacker
>> and the origin on which the policy is being enforced is the victim.
>
> SiteA wants to make sure it cannot ever be framed. It deploys
X-Frame-Options headers and framebusting JS, and maybe even CSP
frame-ancestors.
>
> SiteB wants to make sure it never loads data from anything other than
SiteB (no non-origin loads). It outputs CSP headers to this effect
>
> SiteC wants to make sure that any content it frames cannot run ActiveX
controls, nor do a 401 authentication. It can't really do this with
current iframe sandboxing, but pretend it could...
>
> SiteC wants to control the behavior of children that it frames. It
needs to advertise this policy to a web browser. It has two choices:
>
> 1. It can do it inline in the HTML it outputs with extra attributes of
the iframe it creates. SiteC is in complete control of the HTML that
creates the iframe. I can impose any policy via sandbox attributes.
Currently for example, it can disable JS in the frame. If it frames
SiteA, SiteA's framebusting JS will never run, but the browser will
respect its X-Frame-Options headers.
>
> 2. SiteC is also totally in control of all HTTP headers it emits. It
could just as easily indicate policy choices for all frames via CSP. It
could advertise a blanket policy (No JS, No ActiveX). Advertising a
page-specific, or frame/target specific policy is substantially more
difficult and probably unwieldy. But, depending on how SiteC is
configured, setting a global site policy via headers offers a potential
separation of duties that #1 does not, it allows website admin to
specific things that each web developer might not be able to.
>
> 3. Because all of Site A,B,C are in different origins, they don't
really have to worry about polluting other origins, but they do have to
worry about problematic behavior such as top-nav, 401-auth popups, etc.
Parents need to constrain certain behavior of things they embed,
according to certain rules of whether the child allows itself to be
framed.
>
> I totally get how existing iframe sandboxing that turns off JS is
problematic for sites [due to] older browsers that don't support
X-Frame-Options. We already have a complicated interaction between
these multiple security controls.

>

> Can you give me an example of why my #1/#2 are actually that different? Whether we control behavior with headers of inline content, each site is totally responsible for what it emits, and can already control in some interesting ways the behavior of content it frames/includes.

In this example, the trade-off for Site C seems to boil down to the granularity of the policy. Using attributes on a frame is more fine-grained because Site C can make these decisions on an iframe-by-iframe basis whereas using a document-wide policy is more coarse-grained.

Of course, there's a trade-off between different granularities. On the one hand, fine-grained gives the site more control over how different iframes behavior. On the other hand, it's much easier to audit and understand the effects of a coarse-grained policy.

Adam

B.11. Source: Policy Delivery [1]

In terms of policy delivery, Brandon Sterne supplied this: <"[Content Security Policy] Proposal to move the debate forward" (Brandon Sterne). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0118.html>>

"...

6. Policy delivery

- a. HTTP header
- b. <meta> (or <link>) tag, to be superseded by header if present
- c. policy-uri: a URI from which the policy will be fetched; can be specified in either header or tag

..."

B.12. Source: Policy Delivery [2]

In terms of defining policy delivery, gaz Heyes supplied this: <"Re: [Content Security Policy] Proposal to move the debate forward" (gaz Heyes). <http://lists.w3.org/Archives/Public/public-web-security/2011Jan/0148.html>>

"...

- a) Policy shouldn't be defined in a http header it's too messy and what happens when there's a mistake?

b) As discussed on the list there is no need to have a separate method as it can be generated by an attacker. If a policy doesn't exist then an attacker can now DOS the web site via meta.

c) We have a winner, a http header specifying a link to the policy file is the way to go IMO, my only problem with it is devs implementing it. Yes facebook would and probably twitter would but Dave's tea shop wouldn't pay enough money to hire a web dev who knew how to implement a custom http header yet they would know how to validate HTML. So the question is are we bothered about little sites that are likely to have nice tea and XSS holes? If so I suggest updating the HTML W3C validator to require a security policy to pass validation if not I suggest a policy file delivered by http header.
..."

B.13. Source: Policy Conflict Resolution

In terms of defining policy conflict resolution, Andrew Steingruebl supplied this: <"RE: Content Security Policy and iframe@sandbox" (Andrew Steingruebl). <http://lists.w3.org/Archives/Public/public-web-security/2011Feb/0048.html>>

> -----Original Message-----
> From: public-web-security-request@w3.org [mailto:public-web-security-
> request@w3.org] On Behalf Of Adam Barth
>
> @sandbox and CSP are very different. The primary difference is who
> choses the policy. In the case of @sandbox, the embedder chooses
> the policy. In CSP, the provider of the resource chooses the policy.

While this is true today, I could imagine a tweak to CSP wherein CSP would control all contents hierarchically. I already spoke to Brandon about it, but it was just a quick brainstorm.

You could imagine revoking permissions in the frame hierarchy and not granting them back. This does start to get awfully ugly, but just as CSP controls loading policy for itself, it could also control loading policy for children, right?

Fundamentally, since the existing security model doesn't really provide for strict separation of parent/child (popups, 401's, top-nav) CSP and iframe sandbox both try to control the behavior of resources we pull from other parties.

Do we think that these are both special cases of a general security policy (my intuition says yes) or that they have some quite orthogonal types of security controls that cannot be mixed into a single policy declaration?

One clear problem that comes to mind is that there are policies that come from the "child" such as X-Frame-Options that must break the ordinary parent/child relationship from a precedence standpoint.

Author's Address

Jeff Hodges
PayPal
2211 North First Street
San Jose, California 95131
US

Email: Jeff.Hodges@PayPal.com

Web Security
Internet-Draft
Intended status: Standards Track
Expires: June 10, 2013

C. Evans
C. Palmer
R. Sleevi
Google, Inc.
December 7, 2012

Public Key Pinning Extension for HTTP
draft-ietf-websec-key-pinning-04

Abstract

This memo describes an extension to the HTTP protocol allowing web host operators to instruct user agents (UAs) to remember ("pin") the hosts' cryptographic identities for a given period of time. During that time, UAs will require that the host present a certificate chain including at least one Subject Public Key Info structure whose fingerprint matches one of the pinned fingerprints for that host. By effectively reducing the number of authorities who can authenticate the domain during the lifetime of the pin, pinning may reduce the incidence of man-in-the-middle attacks due to compromised Certification Authorities.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on June 10, 2013.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of

publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Language	3
2. Server and Client Behavior	3
2.1. Response Header Field Syntax	3
2.1.1. The max-age Directive	5
2.1.2. The includeSubDomains Directive	5
2.1.3. The report-uri Directive	5
2.1.4. The strict Directive	6
2.1.5. Examples	6
2.2. Server Processing Model	6
2.2.1. HTTP-over-Secure-Transport Request Type	6
2.2.2. HTTP Request Type	7
2.3. User Agent Processing Model	7
2.3.1. Public-Key-Pins Response Header Field Processing	7
2.3.2. Noting a Pinned Host – Storage Model	8
2.3.3. HTTP-Equiv <Meta> Element Attribute	8
2.3.4. UA Processing Examples	8
2.4. Semantics of Pins	9
2.5. Noting Pins	9
2.6. Validating Pinned Connections	11
2.7. Interactions With Preloaded Pin Lists	11
2.8. Pinning Self-Signed End Entities	12
3. Reporting Pin Validation Failure	12
4. Security Considerations	13
4.1. Backup Pins	14
5. IANA Considerations	14
6. Usability Considerations	14
7. Acknowledgements	14
8. What's Changed	14
9. References	15
9.1. Normative References	15
9.2. Informative References	16
Appendix A. Fingerprint Generation	16
Appendix B. Deployment Guidance	17
Authors' Addresses	18

1. Introduction

We propose a new HTTP header to enable a web host to express to user agents (UAs) which Subject Public Key Info (SPKI) structure(s) UAs SHOULD expect to be present in the host's certificate chain in future connections using TLS (see [RFC5246]). We call this "public key pinning". At least one UA (Google Chrome) has experimented with shipping with a user-extensible embedded set of pins. Although effective, this does not scale. This proposal addresses the scale problem.

Deploying public key pinning safely will require operational and organizational maturity due to the risk that hosts may make themselves unavailable by pinning to a SPKI that becomes invalid. (See Section 4.) We believe that, with care, host operators can greatly reduce the risk of MITM attacks and other false-authentication problems for their users without incurring undue risk.

We intend for hosts to use public key pinning together with HSTS ([RFC6797]), but it is possible to pin keys without requiring HSTS.

This draft is being discussed on the WebSec Working Group mailing list, websec@ietf.org.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Server and Client Behavior

2.1. Response Header Field Syntax

The Public-Key-Pins HTTP response header field (PKP header field) indicates to a UA that it SHOULD perform Pin Validation (Section 2.6) in regards to the host emitting the response message containing this header field, and provides the necessary information for the UA to do so.

Figure 1 describes the ABNF (Augmented Backus-Naur Form) syntax of the header field. It is based on the Generic Grammar defined in Section 2 of [RFC2616] (which includes a notion of "implied linear whitespace", also known as "implied *LWS").


```
Public-Key-Pins =  
    "Public-Key-Pins" ":" [ directive ] *( ";" [ directive ] )  
Public-Key-Pins-Report-Only =  
    "Public-Key-Pins-Report-Only" ":" [ directive ] *( ";" [ directive ] )  
  
directive          = simple-directive  
                    / pin-directive  
  
simple-directive = directive-name [ "=" directive-value ]  
directive-name   = token  
directive-value  = token  
                  / quoted-string  
  
pin-directive     = "pin-" token "=" quoted-string
```

Figure 1: HPKP Header Syntax

token and quoted-string are used as defined in [RFC2616], Section 2.2.

The directives defined in this specification are described below. The overall requirements for directives are:

1. The order of appearance of directives is not significant.
2. All simple-directives MUST appear only once in a PKP header field. Directives are either optional or required, as stipulated in their definitions.
3. Directive names are case-insensitive.
4. UAs MUST ignore any PKP header fields containing directives, or other header field value data, that do not conform to the syntax defined in this specification.
5. If a PKP header field contains any directive(s) the UA does not recognize, the UA MUST ignore the those directives.
6. If the PKP header field otherwise satisfies the above requirements (1 through 5), the UA MUST process the directives it recognizes.

Additional directives extending the semantic functionality of the PKP header field can be defined in other specifications, with a registry (having an IANA policy definition of IETF Review [RFC2616]) defined for them at such time. Such future directives will be ignored by UAs implementing only this specification, as well as by generally non-conforming UAs.

In the pin-directive, the token is the name of a cryptographic hash algorithm, and MUST be either "sha1" or "sha256". The quoted-string is a sequence of base 64 digits: the base 64-encoded SPKI Fingerprint. See Section 2.4.

2.1.1. The max-age Directive

The REQUIRED "max-age" directive specifies the number of seconds, after the reception of the PKP header field, during which the UA SHOULD regard the host (from whom the message was received) as a Known Pinned Host. The delta-seconds production is specified in [RFC2616].

The syntax of the max-age directive's REQUIRED value (after quoted-string unescaping, if necessary) is defined as:

```
max-age-value = delta-seconds
delta-seconds = 1*DIGIT
```

Figure 2: max-age Value Syntax

delta-seconds is used as defined in [RFC2616], Section 3.3.2.

NOTE: A max-age value of zero (i.e., "max-age=0") signals the UA to cease regarding the host as a Known Pinned Host, including the includeSubDomains directive (if asserted for that Known Pinned Host). See Section 2.3.1.

2.1.2. The includeSubDomains Directive

The OPTIONAL "includeSubDomains" directive is a valueless directive which, if present (i.e., it is "asserted"), signals to the UA that the Pinning Policy applies to this Pinned Host as well as any subdomains of the host's domain name.

2.1.3. The report-uri Directive

The OPTIONAL "report-uri" directive indicates the URI to which the UA SHOULD report Pin Validation failures (Section 2.6). The UA POSTs the reports to the given URI as described in Section 3.

TODO: Describe the meaning of Public-Key-Pins-Report-Only and the interaction between it and report-uri. In particular, describe how it is possible to be in enforcement mode (i.e. not -Report-Only) and still POST reports to the report-uri.

2.1.4. The strict Directive

The OPTIONAL "strict" directive is a valueless directive which, if present (i.e., it is "asserted"), signals to the UA that the Pinning Policy contained should be applied to the Pinned Host exactly as specified, ignoring local client policy.

2.1.5. Examples

Figure 3 shows some example response header fields using the pins extension (folded for clarity).

```
Public-Key-Pins: max-age=500;
    pin-sha1="4n972HfV354KP560yw4uqe/baXc=";
    pin-sha1="IvGeLsbqzPxdI0b0wuj2xVTdXgc="

Public-Key-Pins: max-age=31536000;
    pin-sha1="4n972HfV354KP560yw4uqe/baXc=";
    pin-sha256="LPJNul+wow4m6DsqxnbnihsWHlwfp0JecwQzYpOLmCQ="

Public-Key-Pins: pin-sha1="4n972HfV354KP560yw4uqe/baXc=";
    pin-sha1="qvTGHdzF6KLavt4PO0gs2a6pQ00=";
    pin-sha256="LPJNul+wow4m6DsqxnbnihsWHlwfp0JecwQzYpOLmCQ=";
    max-age=2592000

Public-Key-Pins: pin-sha1="4n972HfV354KP560yw4uqe/baXc=";
    pin-sha1="qvTGHdzF6KLavt4PO0gs2a6pQ00=";
    pin-sha256="LPJNul+wow4m6DsqxnbnihsWHlwfp0JecwQzYpOLmCQ=";
    max-age=2592000; includeSubDomains
```

Figure 3: HPKP Header Examples

2.2. Server Processing Model

This section describes the processing model that Pinned Hosts implement. The model comprises two facets: the processing rules for HTTP request messages received over a secure transport (e.g. TLS [RFC5246]); and the processing rules for HTTP request messages received over non-secure transports, such as TCP.

2.2.1. HTTP-over-Secure-Transport Request Type

When replying to an HTTP request that was conveyed over a secure transport, a Pinned Host SHOULD include in its response exactly one PKP header field that MUST satisfy the grammar specified above in Section 2.1. If the Pinned Host does not include the PKP header field, and if the connection passed Pin Validation, UAs MUST treat the host as if it had set its max-age to 0 (see Section 2.3.1).

Establishing a given host as a Known Pinned Host, in the context of a given UA, MAY be accomplished over the HTTP protocol, which is in turn running over secure transport, by correctly returning (per this specification) at least one valid PKP header field to the UA. Other mechanisms, such as a client-side pre-loaded Known Pinned Host list MAY also be used.

2.2.2. HTTP Request Type

Pinned Hosts SHOULD NOT include the PKP header field in HTTP responses conveyed over non-secure transport. UAs MUST ignore any PKP header received in an HTTP response conveyed over non-secure transport.

2.3. User Agent Processing Model

This section describes the HTTP Public Key Pinning processing model for UAs.

TODO: Add a note referring to the HSTS RFC's discussion of IDNs.

2.3.1. Public-Key-Pins Response Header Field Processing

If the UA receives, over a secure transport, an HTTP response that includes a PKP header field conforming to the grammar specified in Section 2.1, and there are no underlying secure transport errors or warnings (see Section 2.5), the UA MUST either:

- o Note the host as a Known HSTS Host if it is not already so noted (see Section 2.3.2),

or,

- o Update the UA's cached information for the Known Pinned Host if any of of the max-age, includeSubDomains, strict, or report-uri header field value directives convey information different than that already maintained by the UA.
- o The max-age value is essentially a "time to live" value relative to the time of the most recent observation of the PKP header field.
- o If the max-age header field value token has a value of 0, the UA MUST remove its cached Pinning Policy information (including the includeSubDomains and strict directives, if asserted) if the Pinned Host is Known, or, MUST NOT note this Pinned Host if it is not yet Known.

- o If a UA receives more than one PKP header field in an HTTP response message over secure transport, then the UA MUST process only the first such header field.

Otherwise:

- o If the UA receives the HTTP response over insecure transport, or if the PKP header is not a Valid Pinning Header (see Section 2.5), the UA MUST ignore any present PKP header field(s).
- o The UA MUST ignore any PKP header fields not conforming to the grammar specified in Section 2.1.

2.3.2. Noting a Pinned Host - Storage Model

If the substring matching the host production from the Request-URI (of the message to which the host responded) syntactically matches the IP-literal or IPv4address productions from Section 3.2.2 of [RFC3986], then the UA MUST NOT note this host as a Known Pinned Host.

Otherwise, if the substring does not congruently match a Known Pinned Host's domain name, per the matching procedure specified in Section 8.2 of [RFC6797], then the UA MUST note this host as a Known Pinned Host, caching the Pinned Host's domain name and noting along with it the expiry time of this information, as effectively stipulated per the given max-age value, as well as whether or not the includeSubDomains or strict directives are asserted, the value of the report-uri directive (if present), and any other metadata from optional or future PKP header directives.

The UA MUST NOT modify the expiry time nor the includeSubDomains directive of any superdomain matched Known Pinned Host.

A Known Pinned Host is "expired" if its cache entry has an expiry date in the past. The UA MUST evict all expired Known Pinned Hosts from its cache, if at any time, an expired Known Pinned Host exists in the cache.

2.3.3. HTTP-Equiv <Meta> Element Attribute

UAs MUST NOT heed http-equiv="Public-Key-Pins" attribute settings on <meta> elements [W3C.REC-html401-19991224] in received content.

2.3.4. UA Processing Examples

TODO.

2.4. Semantics of Pins

An SPKI Fingerprint is defined as the output of a known cryptographic hash algorithm whose input is the DER-encoded ASN.1 representation of the SubjectPublicKeyInfo (SPKI) field of an X.509 certificate. A Pin is defined as the combination of the known algorithm identifier and the SPKI Fingerprint computed using that algorithm.

The SPKI Fingerprint is encoded in base 64 for use in an HTTP header. (See [RFC4648].)

In this version of the specification, the known cryptographic hash algorithms are SHA-1, identified as "sha1", and SHA-256, identified as "sha256". (Future versions of this specification may add new algorithms and deprecate old ones.) UAs MUST ignore Pins for which they do not recognize the algorithm identifier. UAs MUST continue to process the rest of a PKP response header field and note Pins for algorithms they do recognize; UAs MUST recognize "sha1" and "sha256".

Figure 4 reproduces the definition of the SubjectPublicKeyInfo structure in [RFC5280].

```
SubjectPublicKeyInfo ::= SEQUENCE {
    algorithm             AlgorithmIdentifier,
    subjectPublicKey      BIT STRING  }

AlgorithmIdentifier ::= SEQUENCE {
    algorithm             OBJECT IDENTIFIER,
    parameters           ANY DEFINED BY algorithm OPTIONAL  }
```

Figure 4: SPKI Definition

If the SubjectPublicKeyInfo of a certificate is incomplete when taken in isolation, such as when holding a DSA key without domain parameters, a public key pin cannot be formed.

We pin public keys, rather than entire certificates, to enable operators to generate new certificates containing old public keys (see [why-pin-key]).

See Appendix A for an example non-normative program that generates SPKI Fingerprints from SubjectPublicKeyInfo fields in certificates.

2.5. Noting Pins

Upon receipt of the Public-Key-Pins response header field, the UA notes the host as a Pinned Host, storing the Pins and their associated directives in non-volatile storage (for example, along

with the HSTS metadata). The Pins and their associated directives collectively known as Pinning Metadata.

The UA MUST observe these conditions when noting a Host:

- o The UA MUST note the Pins if and only if it received the Public-Key-Pins response header field over an error-free TLS connection. If the host is a Pinned Host, this includes the validation added in Section 2.6.
- o The UA MUST note the Pins if and only if the TLS connection was authenticated with a certificate chain containing at least one of the SPKI structures indicated by at least one of the given SPKI Fingerprints. (See Section 2.6.)
- o The UA MUST note the Pins if and only if the given set of Pins contains at least one Pin that does NOT refer to an SPKI in the certificate chain. (That is, the host must set a Backup Pin; see Section 4.1.)

If the Public-Key-Pins response header field does not meet all three of these criteria, the UA MUST NOT note the host as a Pinned Host. A Public-Key-Pins response header field that meets all these criteria is known as a Valid Pinning Header.

The UA MUST ignore Public-Key-Pins response header fields received on connections that do not meet the first criterion.

TODO: Consider whether or not this requirement makes sense: If the UA receives a Public-Key-Pins header from a Pinned Host that meets the first criterion, but not the following two, the UA MUST discard any previously set Pinning Metadata for that host in its non-volatile store. Whether or not the Known Pinned Host is in strict mode, should the UA note new pins when Pin Validation is disabled per local policy?

Whenever a UA receives a Valid Pinning Header, it MUST set its Pinning Metadata to the exact Pins, max-age, and (if any) report-uri and strict mode given in the most recently received Valid Pinning Header.

For forward compatibility, the UA MUST ignore any unrecognized Public-Key-Pins header directives, while still processing those directives it does recognize. Section 2.1 specifies the directives max-age, pins, includeSubDomains, report-uri, and strict, but future specifications and implementations might use additional directives.

2.6. Validating Pinned Connections

When a UA connects to a Pinned Host, if the TLS connection has errors, the UA MUST terminate the connection without allowing the user to proceed anyway. (This behavior is the same as that required by [RFC6797].)

If the connection has no errors, then the UA will determine whether to apply a new, additional correctness check: Pin Validation. A UA SHOULD perform Pin Validation whenever connecting to a Known Pinned Host, but MAY allow Pin Validation to be disabled for Hosts according to local policy. For example, a UA may disable Pin Validation for Pinned Hosts whose validated certificate chain terminates at a user-defined trust anchor, rather than a trust anchor built-in to the UA. However, if the Pinned Host Metadata indicates that the Pinned Host is operating in "strict mode" (see Section 2.1.4), then the UA MUST perform Pin Validation.

To perform Pin Validation, the UA will compute the SPKI Fingerprints for each certificate in the Pinned Host's validated certificate chain, using each supported hash algorithm for each certificate. (For the purposes of Pin Validation, the UA MUST ignore certificates whose SPKI cannot be taken in isolation and superfluous certificates in the chain that do not form part of the validating chain.) The UA will then check that the set of these SPKI Fingerprints intersects the set of SPKI Fingerprints in that Pinned Host's Pinning Metadata. If there is set intersection, the UA continues with the connection as normal. Otherwise, the UA MUST treat this Pin Failure as a non-recoverable error. Any procedure that matches the results of this Pin Validation procedure is considered equivalent.

Note that, although the UA has previously received Pins at the HTTP layer, it can and MUST perform Pin Validation at the TLS layer, before beginning an HTTP conversation over the TLS channel. The TLS layer thus evaluates TLS connections with pinning information the UA received previously, regardless of mechanism: statically preloaded, via HTTP header, or some other means (possibly in the TLS layer itself).

2.7. Interactions With Preloaded Pin Lists

UAs MAY choose to implement built-in public key pins, alongside any built-in HSTS opt-in list. UAs MUST allow users to override a built-in pin list, including turning it off.

UAs MUST use the newest information -- built-in or set via Valid Pinning Header -- when performing Pin Validation for the host. If the result of noting a Valid Pinning Header is to disable pinning for

the host (such as because the host set a max-age directive with a value of 0), UAs MUST allow this new information to override any built-in pins. That is, a host must be able to un-pin itself even from built-in pins.

2.8. Pinning Self-Signed End Entities

If UAs accept hosts that authenticate themselves with self-signed end entity certificates, they MAY also allow hosts to pin the public keys in such certificates. The usability and security implications of this practice are outside the scope of this specification.

3. Reporting Pin Validation Failure

When a Known Pinned Host has set the report-uri directive, the UA SHOULD report Pin Validation failures to the indicated URI. The UA does this by POSTing a JSON message to the URI; the JSON message takes this form:

```
{
  "date-time": date-time,
  "hostname": hostname,
  "port": port,
  "certificate-chain": [
    pem1, ... pemN
  ],
  "known-pins": [
    known-pin1, ... known-pinN
  ]
}
```

Figure 5: JSON Report Format

Whitespace outside of quoted strings is not significant. The key/value pairs may appear in any order, but each SHOULD appear only once.

The date-time indicates the time the UA observed the Pin Validation failure. It is provided as a string formatted according to Section 5.6, "Internet Date/Time Format", of [RFC3339].

The hostname is the hostname to which the UA made the original request that failed Pin Validation. It is provided as a string.

The port is the port to which the UA made the original request that failed Pin Validation. It is provided either as a string or as an integer.

The `certificate-chain` is the certificate chain, as constructed by the UA during certificate chain verification. (This may differ from the certificate chain as served by the Known Pinned Host, of course.) It is provided as an array of strings; each string `pem1`, ... `pemN` is the PEM representation of each X.509 certificate as described in [I-D.josefsson-pkix-textual].

The `known-pins` are the Pins that the UA has noted for the Known Pinned Host. They are provided as an array of strings with the syntax:

```
known-pin = token "=" quoted-string
```

Figure 6: Known Pin Syntax

As in Section 2.4, the token refers to the algorithm name, and the quoted-string refers to the base 64 encoding of the SPKI Fingerprint.

4. Security Considerations

Pinning public keys helps hosts strongly assert their cryptographic identity even in the face of issuer error, malfeasance or compromise. But there is some risk that a host operator could lose or lose control of their host's private key (such as by operator error or host compromise). If the operator had pinned only the key of the host's end entity certificate, the operator would not be able to serve their web site or application in a way that UAs would trust for the duration of their pin's max-age. (Recall that UAs MUST close the connection to a host upon Pin Failure.)

Therefore, there is a necessary trade-off between two competing goods: pin specificity and maximal reduction of the scope of issuers on the one hand; and flexibility and resilience of the host's cryptographic identity on the other hand. One way to resolve this trade-off is to compromise by pinning to the key(s) of the issuer(s) of the host's end entity certificate(s). Often, a valid certificate chain will have at least two certificates above the end entity certificate: the intermediate issuer, and the trust anchor. Operators can pin any one or more of the public keys in this chain, and indeed could pin to issuers not in the chain (as, for example, a Backup Pin). Pinning to an intermediate issuer, or even to a trust anchor or root, still significantly reduces the number of issuers who can issue end entity certificates for the Known Pinned Host, while still giving that host flexibility to change keys without a disruption of service.

4.1. Backup Pins

The primary way to cope with the risk of inadvertant Pin Failure is to keep a Backup Pin. A Backup Pin is a fingerprint for the public key of a secondary, not-yet-deployed key pair. The operator keeps the backup key pair offline, and sets a pin for it in the Public-Key-Pins header. Then, in case the operator loses control of their primary private key, they can deploy the backup key pair. UAs, who have had the backup key pair pinned (when it was set in previous Valid Pinning Headers), can connect to the host without error.

Because having a backup key pair is so important to recovery, UAs MUST require that hosts set a Backup Pin. (See Section 2.5.)

5. IANA Considerations

This document has no actions for IANA.

6. Usability Considerations

When pinning works to detect impostor Pinned Hosts, users will experience denial of service. UAs MUST explain the reason why, i.e. that it was impossible to verify the confirmed cryptographic identity of the host.

UAs MUST have a way for users to clear current pins for Pinned Hosts. UAs SHOULD have a way for users to query the current state of Pinned Hosts.

7. Acknowledgements

Thanks to Tobias Gondrom, Jeff Hodges, Adam Langley, Nicolas Lidzborski, SM, James Manger, and Yoav Nir for suggestions and edits that clarified the text. Thanks to Trevor Perrin for suggesting a mechanism to affirmatively break pins ([pin-break-codes]). Adam Langley provided the SPKI fingerprint generation code.

8. What's Changed

Removed the section "Pin Validity Times", which was intended to be in harmony with [I-D.perrin-tls-tack]. Now using max-age purely as specified in [RFC6797].

Added new directives: includeSubDomains, report-uri and strict.

Added, but have not yet described, a new variant of the PKP Header: Public-Key-Pins-Report-Only.

Removed the section on pin break codes and verifiers, in favor the of most-recently-received policy (Section 2.5).

Now using a new header field, Public-Key-Pins, separate from HSTS. This allows hosts to use pinning separately from Strict Transport Security.

Explicitly requiring that UAs perform Pin Validation before the HTTP conversation begins.

Backup Pins are now required.

Separated normative from non-normative material. Removed tangential and out-of-scope non-normative discussion.

9. References

9.1. Normative References

- [I-D.josefsson-pkix-textual]
Josefsson, S. and S. Leonard, "Text Encodings of PKIX and CMS Structures", draft-josefsson-pkix-textual-01 (work in progress), July 2012.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4648] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", RFC 6797, November 2012.
- [W3C.REC-html401-19991224]
Hors, A., Raggett, D., and I. Jacobs, "HTML 4.01 Specification", World Wide Web Consortium Recommendation REC-html401-19991224, December 1999, <<http://www.w3.org/TR/1999/REC-html401-19991224>>.

9.2. Informative References

- [I-D.perrin-tls-tack]
Marlinspike, M., "Trust Assertions for Certificate Keys", draft-perrin-tls-tack-01 (work in progress), September 2012.
- [pin-break-codes]
Perrin, T., "Self-Asserted Key Pinning", September 2011, <<http://trevp.net/SAKP/>>.
- [why-pin-key]
Langley, A., "Public Key Pinning", May 2011, <<http://www.imperialviolet.org/2011/05/04/pinning.html>>.

Appendix A. Fingerprint Generation

This Go program generates SPKI Fingerprints, suitable for use in pinning, from PEM-encoded certificates. It is non-normative.

```
package main

import (
    "io/ioutil"
    "os"
    "crypto/sha1"
    "crypto/x509"
    "encoding/base64"
    "encoding/pem"
    "fmt"
)

func main() {
    if len(os.Args) < 2 {
        fmt.Printf("Usage: %s PEM-filename\n", os.Args[0])
        os.Exit(1)
    }
    pemBytes, err := ioutil.ReadFile(os.Args[1])
    if err != nil {
        panic(err.String())
    }
    block, _ := pem.Decode(pemBytes)
    if block == nil {
        panic("No PEM structure found")
    }
    derBytes := block.Bytes
    certs, err := x509.ParseCertificates(derBytes)
    if err != nil {
        panic(err.String())
    }
    cert := certs[0]
    h := sha1.New()
    h.Write(cert.RawSubjectPublicKeyInfo)
    digest := h.Sum()

    fmt.Printf("Hex: %x\nBase64: %s\n", digest,
        base64.StdEncoding.EncodeToString(digest))
}
```

Figure 7: Example SPKI Fingerprint Generation Code

Appendix B. Deployment Guidance

This section is non-normative guidance which may smooth the adoption of public key pinning.

- o Operators SHOULD get the backup public key signed by a different (root and/or intermediary) CA than their primary certificate, and store the backup key pair safely offline. The semantics of an SPKI Fingerprint do not require the issuance of a certificate to construct a valid Pin. However, in many deployment scenarios, in order to make a Backup Pin operational the server operator will need to have a certificate to deploy TLS on the host. Failure to obtain a certificate through prior arrangement will leave clients that recognize the site as a Known Pinned Host unable to successfully perform Pin Validation until such a time as the operator can obtain a new certificate from their desired certificate issuer.
- o It is most economical to have the backup certificate signed by a completely different signature chain than the live certificate, to maximize recoverability in the event of either root or intermediary signer compromise.
- o Operators SHOULD periodically exercise their Backup Pin plan -- an untested backup is no backup at all.
- o Operators SHOULD start small. Operators SHOULD first deploy public key pinning by using the report-only mode together with a report-uri directive that points to a reliable report collection endpoint. When moving out of report-only mode, operators should start by setting a max-age of minutes or a few hours, and gradually increase max-age as they gain confidence in their operational capability.

Authors' Addresses

Chris Evans
Google, Inc.
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: cevans@google.com

Chris Palmer
Google, Inc.
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: palmer@google.com

Ryan Sleevi
Google, Inc.
1600 Amphitheatre Pkwy
Mountain View, CA 94043
US

Email: sleevi@google.com

WEBSEC
Internet-Draft
Intended status: Informational
Expires: August 29, 2013

D. Ross
Microsoft
T. Gondrom
Thames Stanley
February 25, 2013

HTTP Header Field X-Frame-Options
draft-ietf-websec-x-frame-options-02

Abstract

To improve the protection of web applications against Clickjacking, this specification describes the X-Frame-Options HTTP response header field that declares a policy communicated from the server to the client browser on whether the browser may display the transmitted content in frames that are part of other web pages. This informational document serves to document the existing use and specification of this X-Frame-Options HTTP response header field.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 29, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must

include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction	3
1.1. Requirements Language	3
2. X-Frame-Options Header	3
2.1. Syntax	4
2.2. Augmented Backus-Naur Form (ABNF)	5
2.2.1. Examples of X-Frame-Options	5
2.2.2. Variations of the ALLOW-FROM field	5
2.3. Design Issues	5
2.3.1. Enable HTML content from other domains	5
2.3.2. Browser Behaviour and Processing	6
2.3.2.1. Violation of X-Frame-Options	6
2.3.2.2. Variation in current browser behaviour	6
2.3.2.3. Usage design pattern and example scenario for the ALLOW-FROM parameter	7
3. Acknowledgements	7
4. IANA Considerations	8
4.1. Registration Template	8
5. Security Considerations	8
5.1. Privacy Considerations	9
6. References	9
6.1. Normative References	9
6.2. Informative References	9
Appendix A. Browsers that support X-Frame-Options	10
Appendix B. Description of a Clickjacking attack	10
B.1. Shop	10
B.2. Online Shop Confirm Purchase Page	11
B.3. Flash Configuration	11
Authors' Addresses	11

1. Introduction

In 2009 and 2010 many browser vendors ([Microsoft-X-Frame-Options], [CLICK-DEFENSE-BLOG], [Mozilla-X-Frame-Options]) introduced the use of a non-standard HTTP [RFC2616] header field "X-Frame-Options" to protect against Clickjacking [Clickjacking]. HTML-based web applications can embed or "frame" other web pages. Clickjacking is a type of attack that occurs when an attacker uses multiple transparent or opaque layers in the user interface to trick a user into clicking on a button or link on another page from server B when they were intending to click on the same place of the overlaying page from server A. Thus, the attacker is "hijacking" clicks meant for their page A and routing them to another page B, possibly belonging to another domain and thereby triggering actions on the second server B without the knowledge nor intention of the user and potentially using an existing session context and login in that step.

This specification provides informational documentation about the current use and definition of the X-Frame-Options HTTP header field. Given that the "X-" construction is deprecated [RFC6648], the X-Frame-Options header field will in the future be replaced by the Frame-Options directive in the Content Security Policy Version 1.1 [CSP-1-1].

Existing anti-ClickJacking measures, e.g. Frame-breaking Javascript, have weaknesses so that their protection can be circumvented as a study [FRAME-BUSTING] demonstrated.

Short of configuring the browser to disable frames and script entirely, which massively impairs browser utility, browser users are vulnerable to this type of attack.

"X-Frame-Options" allows a secure web page from host B to declare that its content (for example a button, links, text, etc.) must not be displayed in a frame (<frame> or <iframe>) of another page (e.g. from host A). In principle this is done by a policy declared in the HTTP header and enforced by conforming browser implementations.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. X-Frame-Options Header

The X-Frame-Options HTTP response header field indicates a policy on

whether the browser should render the transmitted resource within a <frame> or <iframe>. Servers can declare this policy in the header of their HTTP responses to prevent clickjacking attacks, and by this ensuring that their content is not embedded into other pages or frames.

2.1. Syntax

The header field name is:
X-Frame-Options

There are three different values for the header field. These values are mutually exclusive, that is exactly one of the three values MUST be set.

DENY

A browser receiving content with this header MUST NOT display this content in any frame.

SAMEORIGIN

A browser receiving content with this header field MUST NOT display this content in any frame from a page of different origin than the content itself.

If a browser or plugin can not reliably determine whether the origin of the content and the frame have the same origin, this MUST be treated as "DENY".

Please note that current implementations vary on the interpretation of this criteria: In some it only allows to be framed if the origin of the top-level browsing-context is identical to the origin of the content using the X-FRAME-OPTIONS directive, in others it may compare to the origin of the framing page.

ALLOW-FROM (followed by a URI [RFC3986] of a trusted origin)

A browser receiving content with this header MUST NOT display this content in a frame from any page with a top-level browsing context of different origin than the specified origin. While this can expose the page to risks by the trusted origin, in some cases it may be necessary to allow the framing by content from other domains.

If the ALLOW-FROM value is used, it MUST be followed by a valid URI. Any data beyond the domain address (i.e. any data after the "/" separator) is to be ignored. And the algorithm to compare origins from [RFC6454] SHOULD be used to verify that a referring page is of the same origin as the content or that the referring page's origin is identical with the ALLOW-FROM URI. Though in conflict with [RFC6454], current implementations do not consider the port as a

defining component of the origin.

Wildcards or lists to declare multiple domains in one ALLOW-FROM statement are not permitted.

2.2. Augmented Backus-Naur Form (ABNF)

The RFC 2234 [RFC2234] ABNF of the X-Frame-Options header is:

```
X-Frame-Options = "DENY"  
                  / "SAMEORIGIN"  
                  / ( "ALLOW-FROM" RWS URI )
```

With URI as defined in [RFC3986] and RWS and OWS as defined in [HTTPbis-P1]. The values are specified as ABNF strings, and therefore are case-insensitive.

2.2.1. Examples of X-Frame-Options

```
X-FRAME-OPTIONS: DENY
```

```
X-FRAME-OPTIONS: SAMEORIGIN
```

```
X-FRAME-OPTIONS: ALLOW-FROM https://example.com/
```

2.2.2. Variations of the ALLOW-FROM field

Regarding the syntax, it should be noted that existing implementations have variations in whether a colon (":") should be between "ALLOW-FROM" and the URI. E.g. in IE8+ the colon ":" is not needed, while Firefox and Chrome implementations at the time of writing of this document support both forms.

Alternative ABNF of the X-Frame-Options header:

```
X-Frame-Options = "DENY"  
                  / "SAMEORIGIN"  
                  / ( "ALLOW-FROM" OWS ":" OWS URI )
```

2.3. Design Issues

2.3.1. Enable HTML content from other domains

There are a number of main direct vectors that enable HTML content from other domains:

- o IFRAME tag

- o Frame tag
- o The Object tag (requires a redirect)
- o Applet tag
- o Embed tag

Besides these, other ways to host HTML content can be possible. For example some plugins may host HTML views directly. If these plugins appear essentially as frames (as opposed to top-level windows), the plugins MUST conform to the X-FRAME-OPTIONS policy as specified in this document as well.

2.3.2. Browser Behaviour and Processing

To allow secure implementations, browsers must behave in a consistent and reliable way.

If an X-Frame-Options HTTP header field prohibits framing, the user-agent of the browser MAY immediately abort downloading or parsing of the document.

2.3.2.1. Violation of X-Frame-Options

When a browser discovers that loaded content with the X-FRAME-OPTIONS header field would be displayed in a frame against the specified orders of the header, the browser SHOULD redirect as soon as possible to a "No-Frame" page.

"No-Frame" Page

If the display of content is denied by the X-FRAME-OPTIONS header an error page SHOULD be displayed. For example this can be a noframe.html page also stating the full URL of the protected page and the hostname of the protected page.

The NoFrame page MAY provide the user with an option to open the target URL in a new window.

Implementations of this vary, some browsers will show a message that allows the user to safely open the target page in a new window. Other implementations will simply render an empty frame.

2.3.2.2. Variation in current browser behaviour

There are currently variations in the implementation of the X-FRAME-OPTIONS header. For example not all browsers support the "ALLOW-FROM" option. "ALLOW-FROM" was initially an IE (Internet Explorer)

extension and at the time of writing has not been uniformly implemented by other user agents.

And the criteria for the SAMEORIGIN option is not evaluated unanimously either: one implementation may evaluate the SAMEORIGIN option based on the origin of the framed page and the framing page, while another may evaluate based on the framed page and the top-level browsing-context.

These variations in the evaluation of the header by different implementations impair the usage and reliability of this http header. A revised version of x-frame-options in the form of a frame-options directive in the CSP 1.1[CSP-1-1] shall unify the behaviour and replace this document in the future.

2.3.2.3. Usage design pattern and example scenario for the ALLOW-FROM parameter

As the "ALLOW-FROM" field does support only one URI, in cases when the server wishes to allow more than one resource to frame its content, the following design pattern is recommended:

1. A page that wants to render the requested content in a frame supplies its own origin information to the server providing the to-be-framed content via a querystring parameter.
2. The Server verifies the hostname meets its criteria so that the page can be allowed to be framed by the target resource. This may for example happen via a look-up of a white-list of trusted domain names that are allowed to frame the page. For example, for a Facebook "Like" button, the server can check to see that the supplied hostname matches the hostname(s) expected for that "Like" button.
3. The server return the hostname in X-FRAME-OPTIONS: ALLOW-FROM if the proper criteria was met in step #2.
4. The browser enforces the X-FRAME-OPTIONS: ALLOW-FROM header.

3. Acknowledgements

This document was derived from input from specifications published by various browser vendors like Microsoft (Eric Lawrence, David Ross), Mozilla, Google, Opera and Apple.

4. IANA Considerations

This memo is a request to IANA to include the specified HTTP header in the registry as outlined in Registration Procedures for Message Header Fields [RFC3864]

4.1. Registration Template

PERMANENT MESSAGE HEADER FIELD REGISTRATION TEMPLATE:

Header field name: X-Frame-Option

Applicable protocol: http [RFC2616]

Status: Standard

Author/Change controller: IETF

Specification document(s): draft-ietf-websec-x-frame-options

Related information:

Figure 1

5. Security Considerations

The introduction of the X-FRAME-OPTIONS http header field does improve the protection against Clickjacking. However, it is not self-sufficient on its own, but must be used in conjunction with other security measures like secure coding and the Content Security Policy [CSP].

It is important to note that current implementations do not check the origins of the entire ancestor tree of frames of the framing resources, and this may expose the resource to attack in multiply-nested scenarios. For example, if a resource on origin A embeds untrusted content from origin B, that untrusted content can embed another resource from origin A with an X-Frame-Options: SAMEORIGIN policy and that check would pass if the user agent only verifies the top-level browsing context.

Furthermore, X-Frame-Options must be sent as an HTTP header field and is explicitly ignored by user agents when declared with a meta http-equiv tag.

5.1. Privacy Considerations

The parameter ALLOW-FROM allows a page to guess who is framing it. This is inherent by design, but may lead to data leakage or data protection concerns.

6. References

6.1. Normative References

- [HTTPbis-P1] IETF, "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", 2013, <<http://tools.ietf.org/html/draft-ietf-httpbis-p1-messaging-22>>.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.
- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC6454] Barth, A., "The Web Origin Concept", RFC 6454, December 2011.

6.2. Informative References

- [CLICK-DEFENSE-BLOG] Microsoft, "Clickjacking Defense", 2009, <<http://blogs.msdn.com/b/ie/archive/2009/01/27/ie8-security-part-vii-clickjacking-defenses.aspx>>.
- [CSP] W3C, "Content Security Policy 1.0", November 2012, <<http://www.w3.org/TR/CSP/>>.
- [CSP-1-1] W3C, "Content Security Policy 1.1", December 2012, <<http://www.w3.org/TR/CSP11/>>.
- [Clickjacking] OWASP (Open Web Application Security Project), "Clickjacking", 2010, <<http://www.owasp.org/index.php/Clickjacking>>.
- [FRAME-BUSTING]

Stanford Web Security Research, "Busting frame busting: a study of clickjacking vulnerabilities at popular sites", 2010, <<http://seclab.stanford.edu/websec/framebusting/>>.

[Microsoft-X-Frame-Options]

Microsoft, "Combating ClickJacking With X-Frame-Options", 2010, <<http://blogs.msdn.com/b/ieinternals/archive/2010/03/30/combating-clickjacking-with-x-frame-options.aspx>>.

[Mozilla-X-Frame-Options]

Mozilla, "The X-Frame-Options response header", 2010, <https://developer.mozilla.org/en-US/docs/The_X-FRAME-OPTIONS_response_header>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration Procedures for Message Header Fields", BCP 90, RFC 3864, September 2004.

[RFC6648] Saint-Andre, P., Crocker, D., and M. Nottingham, "Deprecating the "X-" Prefix and Similar Constructs in Application Protocols", BCP 178, RFC 6648, June 2012.

Appendix A. Browsers that support X-Frame-Options

- o Internet Explorer 8+
- o Firefox 3.6.9+
- o Opera 10.5+
- o Safari 4+
- o Chrome 4.1+

Appendix B. Description of a Clickjacking attack

More detailed explanation of Clickjacking scenarios

B.1. Shop

An Internet Marketplace/Shop offering a feature with a link/button to "Buy this" Gadget

The marketplace wants their affiliates (who could be malicious attackers) to be able to stick the "Buy such-and-such from XYZ" IFRAMES into their pages. There is a possible Clickjacking threat here, which is why the marketplace/onlineshop needs to then immediately navigate the main browsing context (or a new window) to a confirmation page which is protected by anti-Clickjacking protections.

B.2. Online Shop Confirm Purchase Page

The "Confirm Purchase" page of an online shop must be shown to the end user without the risk of an overlay or misuse by an attacker. For that reason, the confirmation page uses a combination of anti-CSRF tokens and the X-FRAME-OPTIONS HTTP header field, mitigating ClickJacking attacks.

B.3. Flash Configuration

Macromedia Flash configuration settings are set by a Flash object which can run only from a specific configuration page on Macromedia's site. The object runs inside the page and thus can be subject to a ClickJacking attack. In order to prevent ClickJacking attacks against the security settings, the configuration page uses the X-FRAME-OPTIONS directive.

Authors' Addresses

David Ross
Microsoft
U.S.

Phone:
Email:

Tobias Gondrom
Thames Stanley
Kruegerstr. 5A
Unterschleissheim,
Germany

Phone: +44 7521003005
Email: tobias.gondrom@gondrom.org

Network Working Group
Internet-Draft
Intended status: Informational
Expires: July 5, 2013

N. Williams
Cryptonector
January 1, 2013

Hypertext Transport Protocol (HTTP) Session Continuation: Problem
Statement
draft-williams-websec-session-continue-prob-00

Abstract

One of the most often talked about problems in web security is "cookies". Web cookies are a method of associating requests with "sessions" that may have been authenticated somehow. Cookies are a form of bearer token that leave much to be desired. This document describes the session "continuation" problem for the HyperText Transport Protocol (HTTP).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at <http://datatracker.ietf.org/drafts/current/>.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 5, 2013.

Copyright Notice

Copyright (c) 2013 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of

the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1.	Introduction	3
1.1.	Motivation	3
1.2.	Conventions used in this document	4
2.	Requirements	5
2.1.	Statelessness	6
3.	IANA Considerations	8
4.	Security Considerations	9
5.	Acknowledgements	10
6.	References	11
6.1.	Normative References	11
6.2.	Informative References	11
	Author's Address	12

1. Introduction

Today most web applications use "cookies" to associate HTTP requests with "sessions". A "session" is a set of related HTTP requests (and responses), where the relation is to some request(s) that created the session. Some sessions are created by the act of authenticating a user, in which case the primary goal of "sessions" is to avoid having to re-authenticate the user on every request. Other times a session is created when a request is received that is not associated with any session, in which case the primary purpose of "sessions" may be to provide a pseudonymous identifier for an otherwise anonymous user. We call the mechanisms by which requests are strung into sessions "session continuation".

"Cookies" are server-assigned bearer tokens - nothing more, nothing less, though some cookies are used just to store things like "shopping cart" state. A bearer token is an octet blob which can be presented as-is, possibly repeatedly, to authenticate a user to some party; mere possession of the bearer token is sufficient to act on the user's behalf to at least one service. As such they are susceptible to theft via passive attacks (eavesdropping) if not protected in some other way (e.g., by using TLS), or via active attacks such as BEAST and CRIME [http://www.xors.me/?attachment_id=3727], as well as to leakage in various ways [XXX expand].

We would like a session continuation mechanism to replace or augment cookies that has better security semantics than bearer tokens. In particular we would like a system that is not susceptible to theft via active attacks like BEAST and CRIME. We believe that such a scheme should use cryptographic algorithms and cryptographic session keys, and should be amenable to being keyed by HTTP- and web-authentication mechanisms. A new session continuation mechanism should be suitable for use in web and non-web HTTP applications, and should work even for unauthenticated sessions.

1.1. Motivation

The motivation for this document and the related session continuation protocol [I-D.draft-williams-websec-session-continue-proto] document is as follows. We want:

- o A variable authentication token instead of (or in addition to) web cookies, for resistance to BEAST, CRIME, and other adaptive chosen plaintext active attacks on TLS.
- o The ability to explicitly logout and destroy all session state even if the session has been compromised, assuming there is no Man

In The Browser (MITB).

- o The ability to manage sessions.
- o The ability to negotiate replay protection.
- o Cryptographic binding ("channel binding" [RFC5056]) to the lower transport layer (TLS, where available).
- o Cryptographic binding to the user authentication mechanisms (where the authentication mechanism can export a secret value).
- o The ability to use HTTP/Negotiate [RFC4559] in such a way that a) new HTTP(S) connections need not result in re-authentication, b) does not strobgly bind requests in a single HTTP connection to the HTTP/Negotiate authentication that precedes them.

1.2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

2. Requirements

Any session continuation scheme to replace or augment cookies must provide the following functionality:

1. Support for authenticated and unauthenticated sessions alike.
2. Support for http: and https: both.
3. Session continuation must be possible to implement without keeping state on the server side (see below), and it must be possible to keep some state on the server and some on the client.
4. Resistance to active attacks on https. [NOTE: This should probably NOT be a requirement. Instead we should be happy to note where a proposed protocol provides this.]
5. Session continuation must be expressed via HTTP headers.
6. Session continuation header values must be cryptographically difficult for attackers to spoof, and servers must be able to validate these values.
7. Session continuation header values used with TLS must be cryptographically distinct from those used without TLS such that no such values taken from HTTP requests sent without TLS can be used in HTTP requests with TLS.
8. Session continuation must provide protection against man-in-the-middle (MITM) attacks when using TLS. (This is important when using anonymous Diffie-Hellman cipher suites for TLS, as well as when using server certificates from low-value Public Key Infrastructures (PKI)).
9. Must support explicit session termination ("logout"), initiated by either party, client or server. Once a session is logged out there should be no way to use it again, even if any session keys are compromised. Note that this is not a deployment requirement, just a protocol requirement; a fully stateless deployment may not be able to implement faithful logout.
10. Must work across all types of proxies. Proxies that can modify the plaintext HTTP requests and responses can (but should not) interfere with any session continuation protocol.
11. Sessions should be tied to "origins"; multi-origin sessions (sharing sessions across servers) are allowed, but there are

user interface considerations.

[[anchor1: Can you move a session from one server to another? No, probably not. Servers can share sessions, so we need to at least be able to scope sessions to sets of servers or DNS sub-domains. This appears to require that sessions have names. Once we have proper session continuation we may well end up needing a mechanism by which to authenticate to a service as a user of a given session on a foreign service that is "friends" with the first.]]

Recommendations:

1. Session continuation SHOULD use proof-of-possession of secret session key(s).
2. Session continuation header values SHOULD include a cryptographically-secure value (indistinguishable from random) that can be validated by the server and is hard for attackers to guess.
3. Session continuation header values should be salted with a nonce to defeat BEAST- and CRIME-style active attacks.

2.1. Statelessness

Session continuation protocols for HTTP MUST allow for stateless implementation on the server side, at least when TLS is used. Statelessness is not a requirement of deployments; implementations SHOULD support both, stateful and stateless servers. This generally means that any state must be encrypted and encoded into a session state cookie that is re-sent by the client to the server on every request. The server, of course, must be the one to assign such state, and it must use an encryption key known only by the server.

Server-side statelessness is NOT REQUIRED in actual deployments, but the ability to implement session continuation in a stateless fashion on the server side is REQUIRED.

Note that statelessness implies that there is no way to implement replay protection. In the case of session continuation with TLS this is not a concern because TLS itself protects against replays. But when session continuation is used without TLS then statelessness really does mean that there can be no replay protection (of course, this is also thus with web cookies). Therefore servers that require replay protection must either require the use of TLS or must use stateful sessions.

Note also that statelessness makes session logout a no-op on the

server-side, which means that a compromised session can continue to be used even after a client attempts to logout. A session continuation protocol **MUST** allow for storing some state on the server, and some on the client, allowing deployments where the only state stored is the existence of a session.

Probabilistic data structures (e.g., Bloom filters) **MAY** be used to record logouts. This may require the ability to expire and refresh sessions to render the logout system scalable. In other words, HTTP responses **MUST** be allowed to replace session server state stored on the client side.

3. IANA Considerations

This document does not specify any protocols and has no IANA considerations.

4. Security Considerations

This document does not specify any protocols and is Informational. There are, however, few security considerations to document here.

We seek to improve security on the web (as well as for non-web HTTP applications) by:

1. reducing the need for expensive HTTP authentication exchanges (e.g., HTTP/Negotiate), thereby removing an obstacle to their use;
2. reducing exposure to session credentials theft via attacks on TLS such as BEAST and CRIME;
3. reducing exposure to session credentials theft when not using TLS;
4. introducing a replacement for / augmentation of cookies that will give browsers a chance to pursue better security policies.

As discussed in Section 2.1, there is a security consideration regarding session continuation without TLS and with server-side statelessness: there can be no replay protection in this case. However, this is not a loss of security relative to web cookies. Applications must use TLS if they require integrity protection.

5. Acknowledgements

The author thanks Yaron Sheffer, Yoav Nir, and Phillip Hallam-Baker, all of whom are practically co-authors, and invited to be listed as such. The term "session continuation" is Phillip's. The motivation, requirements and recommendations text is a group effort.

6. References

6.1. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

6.2. Informative References

- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2617] Franks, J., Hallam-Baker, P., Hostetler, J., Lawrence, S., Leach, P., Luotonen, A., and L. Stewart, "HTTP Authentication: Basic and Digest Access Authentication", RFC 2617, June 1999.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5056] Williams, N., "On the Use of Channel Bindings to Secure Channels", RFC 5056, November 2007.
- [RFC5929] Altman, J., Williams, N., and L. Zhu, "Channel Bindings for TLS", RFC 5929, July 2010.
- [RFC5849] Hammer-Lahav, E., "The OAuth 1.0 Protocol", RFC 5849, April 2010.
- [RFC4559] Jaganathan, K., Zhu, L., and J. Brezak, "SPNEGO-based Kerberos and NTLM HTTP Authentication in Microsoft Windows", RFC 4559, June 2006.
- [I-D.draft-williams-websec-session-continue-proto] Williams, N., "Hypertext Transport Protocol (HTTP) Session Continuation Protocol", draft-williams-websec-session-continue-proto-00 (work in progress), January 2013.

Author's Address

Nicolas Williams
Cryptonector, LLC

Email: nico@cryptonector.com

