

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: January 2, 2013

J. Hodges
PayPal
Jul 2012

Web Security Framework: Problem Statement and Requirements
draft-hodges-websec-framework-reqs-02

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 January 2, 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. Where to Discuss This Draft	4
2. Document Conventions	4
3. Overall Constraints	5
4. Overall Requirements	6
5. Attacks and Threats to Address	8
5.1. Attacks	8
5.2. Threats	8
6. Use Cases	9
7. Detailed Functional Requirements	10
8. Extant Policies to Coalesce	18
9. Example Concrete Approaches	19
10. Security Considerations	19
11. References	19
12. Informative References	23
Author's Address	23

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 [11], 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.

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.

[[XXXn: Some of the more major known issues are marked like this (where "n" in "XXXn" is a number). --JeffH]]

[[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 above two issue & 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:

We believe that a regime based on HTTP header(s) is appropriate. However we must devise a generalized, extensible HTTP security header(s) such that the on-going "bloat" of the number of disjoint HTTP security headers is mitigated and there is a documented framework that we can leverage as new approaches and/or threats emerge.

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 (transparent proxy servers) as the protocols they ostensibly protect.

It may be reasonable to devise a small set of headers to convey different classes of policies, e.g. web application content policies versus web application network capabilities policies.

out-of-band:

This 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 DNS Security (DNSSEC) [32].

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, we believe that to be effective for a broad array of web site and application types,

the policy expression mechanism must fundamentally facilitate fine-grained control. For example, CSP offers such control. In order to address the less complex needs of the more simple classes of web sites, the policy expression mechanism could have a "macro"-like feature enabling "canned policy profiles". Or, 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 [I-D.draft-ietf-websec-strict-transport-sec-11] suggests the former hard-fail behavior.

5. Attacks and Threats to Address

This section enumerates various attacks and threats that ought to be mitigated by a web security policy framework. In terms of defining threats in contrast to attacks, Lucas supplied this:

```
<"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 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). ..."
```

5.1. Attacks

The below is an enumeration of attacks which are desirable to mitigate via a web application security framework (see [WASC-THREAT] for a definition and taxonomy of attacks):

1. cross-site-scripting (XSS) [2] [WASC-THREAT]
2. Man-in-the-middle (MITM) attacks against "secure" (e.g. Transport Layer Security (TLS/SSL)-based [7] [8] [WASC-THREAT]) web sites. For example, be able to subsume the HSTS header [11].
3. User Interface Redressing [UIRedress], aka Clickjacking [Clickjacking].
4. Cross-Site-Request Forgery (CSRF) [3] [WASC-THREAT] (?)
5. Response Splitting [WASC-THREAT]
6. more (ie eg from [WASC-THREAT] ?) ?

5.2. Threats

Via the attacks 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 concrete use cases. Where applicable, source email messages are cited.

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. Detailed Functional Requirements

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

1. Policy expression syntax:

- * Declarative.

<"declarative languages". <http://www.encyclopedia.com/doc/1011-declarativelanguages.html>>

- * Extensible.

<"Extensibility". <https://secure.wikimedia.org/wikipedia/en/wiki/Extensible>>

<"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."

<"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."

2. Tooling:

- * We will need tools to (ideally) analyze a web application and generate a starting point security policy.

<"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! "

3. Performance:

* Minimizing performance impact is a first-order concern.

<"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."

4. 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.

<"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."

5. 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?".

<"[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

..."

6. 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.

<"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. ..."

7. Hierarchical Policy Application:

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

<"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, ..."

8. Framing Policy Hierarchy, cross-origin, granularity:

<"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 or 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 behave. On the other hand, it's much easier to audit and understand the effects of a coarse-grained policy.

Adam

9. Policy Delivery:

- * 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.

<"[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

..."

<"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.
..."

10. Policy Conflict Resolution:

*

<"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.

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/pl1.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=dxxxqgkd_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/bcflc45f312f/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/aal57732\(office.10\).aspx](http://msdn.microsoft.com/en-us/library/aal57732(office.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>>.

[I-D.ietf-websec-strict-transport-sec]

Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)",
draft-ietf-websec-strict-transport-sec-11 (work in progress), July 2012.

[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>>.

[WASC-THREAT]

Web Application Security Consortium, "The WASC Threat Classification v2.0", January 2010,
<http://projects.webappsec.org/f/WASC-TC-v2_0.pdf>.

[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/>>.

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: April 19, 2013

C. Evans
C. Palmer
Google, Inc.
October 16, 2012

Public Key Pinning Extension for HTTP
draft-ietf-websec-key-pinning-03

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 or more of the pinned fingerprints for that host. By effectively reducing the scope 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 and other authentication errors and attacks.

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 April 19, 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.

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 MUST expect to be present in the host's certificate chain in future connections using TLS (see [rfc-5246]). We call this "public key pinning". At least one user agent (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 3.) 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 (as defined in [hsts-draft]), but 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 [rfc-2119].

2. Server and Client Behavior

2.1. Response Header Field Syntax

To set a pin, hosts use a new HTTP header field, `Public-Key-Pins`, in their HTTP responses. Figure 1 describes the syntax of the header field.

```
Public-Key-Pins = "Public-Key-Pins" ":" LWS directives

directives      = max-age LWS ";" LWS pins
                  / pins LWS ";" LWS max-age

max-age         = "max-age" LWS "=" LWS delta-seconds

pins            = pin
                  / pin LWS ";" LWS pins

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

Figure 1

In the pin rule, the token is the name of a cryptographic hash algorithm, and MUST be either "sha1" or "sha256". (Future versions of this specification may change the hash functions.) The quoted-string is a sequence of base64 digits: a base64-encoded hash. See Section 2.2.

Figure 2 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+wow4m6DsqxnbnnhsWHlwfp0JecwQzYpOLmCQ="

Public-Key-Pins: pin-sha1="4n972HfV354KP560yw4uqe/baXc=";
  pin-sha1="qvTGHdzF6KLavt4PO0gs2a6pQ00=";
  pin-sha256="LPJNul+wow4m6DsqxnbnnhsWHlwfp0JecwQzYpOLmCQ=";
  max-age=2592000
```

Figure 2

2.2. Semantics of Pins

The fingerprint is the SHA-1 or SHA-256 hash of the DER-encoded ASN.1 representation of the SubjectPublicKeyInfo (SPKI) field of the X.509 certificate. Figure 3 reproduces the definition of the SubjectPublicKeyInfo structure in [rfc-5280].

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

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

Figure 3

The SPKI hash is then encoded in base-64 for use in an HTTP header.
(See [rfc-4648].)

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 public key fingerprints from SubjectPublicKeyInfo fields in certificates.

2.3. 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 max-age in non-volatile storage (for example, along with the HSTS metadata). The pins and their associated max-age are 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. The UAs MUST ignore Public-Key-Pins response header fields received on HTTP (non-HTTPS) connections.

- 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 fingerprints. (See Section 2.4.)
- 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 3.1.)

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, and MUST discard any previously set Pinning Metadata for that host in its non-volatile store. Public-Key-Pins response header fields that meet all these criteria are known as Valid Pinning Headers.

Whenever a UA receives a Valid Pinning Header, it MUST set its Pinning Metadata to the exact pins and max-age given in the most recently received Valid Pinning Header.

2.3.1. max-age

max-age specifies the number of seconds, after the reception of the Public-Key-Pins HTTP Response Header, during which the UA regards the host as a Pinned Host (up to a maximum of 30 days; see Section 2.5). The delta-seconds production is specified in [rfc-2616].

Note that by setting a low or 0 value for max-age, hosts effectively instruct UAs to cease regarding them as Pinned Hosts.

2.4. 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 [hsts-draft].)

If the connection has no errors, the UA will then apply a new, additional correctness check: Pin Validation. To perform Pin Validation, the UA will compute the fingerprints of the SPKI structures in each certificate in the host's validated certificate chain. (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 fingerprints intersects the set of fingerprints in that 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.

Note that, although the UA has previously received public key 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, such as specified in [tack-draft]).

2.5. Pin Validity Times

In harmony with section 5.3.4 "Create and activate pins" of [tack-draft], clients MUST enforce a maximum age for pins that is no longer than the least of (a) 30 days (30 * 24 * 60 * 60 seconds) after the most recent time that the client noted the pin; (b) the amount of time the pin has been noted; or (c) the most recent time the pin was noted + max-age:

```
active_period_end = MIN(current + current - initial,
                        time_pin_noted + max-age,
                        current + 30 days)
```

Figure 4

2.6. 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.

2.7. 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. Security Considerations

Pinning public keys helps hosts assert their cryptographic identity, but there is some risk that a host operator could lose or lose control of their host's private key. In this case, 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.)

3.1. 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 2.3.)

4. IANA Considerations

This document has no actions for IANA.

5. 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.

6. 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.

7. What's Changed

Removed the section on pin break codes and verifiers, in favor the of most-recently-received policy (Section 2.3).

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.

8. References

8.1. Normative References

[hsts-draft]

Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", October 2011, <<http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-03>>.

[rfc-2119]

Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <<http://www.ietf.org/rfc/rfc2119.txt>>.

[rfc-2616]

Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", June 1999, <<http://www.ietf.org/rfc/rfc2616.txt>>.

[rfc-4648]

Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", October 2006, <<http://www.ietf.org/rfc/rfc4648.txt>>.

[rfc-5246]

Rescorla, E. and T. Dierks, "The Transport Layer Security (TLS) Protocol Version 1.2", August 2008, <<http://www.ietf.org/rfc/rfc5246.txt>>.

[rfc-5280]

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", May 2008, <<http://www.ietf.org/rfc/rfc5280.txt>>.

8.2. Informative References

[why-pin-key]

Langley, A., "Public Key Pinning", May 2011, <<http://www.imperialviolet.org/2011/05/04/pinning.html>>.

[pin-break-codes]

Perrin, T., "Self-Asserted Key Pinning", September 2011, <<http://trevp.net/SAKP/>>.

[tack-draft]

Perrin, T. and M. Marlinspike, "Trust Assertions for
Certificate Keys", May 2012,
<<http://tools.ietf.org/html/draft-perrin-tls-tack>>.

Appendix A. Fingerprint Generation

This Go program generates public key 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 5

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.
- 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 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

