

Web Security
Internet-Draft
Expires: March 24, 2012

C. Evans
C. Palmer
Google, Inc.
September 21, 2011

Certificate Pinning Extension for HSTS
draft-evans-palmer-hsts-pinning-00

Abstract

This memo describes an extension to the HTTP Strict Transport Security specification allowing web host operators to instruct UAs to remember ("pin") 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 public key 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, we hope pinning reduces 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 March 24, 2012.

Copyright Notice

Copyright (c) 2011 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 to extend the HSTS HTTP header to enable a web host to

express to UAs which certificate(s) UAs may expect to be present in the host's certificate chain in future connections. We call this "certificate pinning". The Google Chrome/ium browser ships with a static set of pins, and individual users can extend the set of pins. Although effective, this does not scale. This proposal addresses the scale problem.

Deploying certificate pinning safely will require operational and organizational maturity due to the risk that HSTS Hosts may "brick" themselves by pinning to a certificate that becomes invalid. We discuss potential mitigations for those risks. 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.

This document extends the version of HSTS defined in [hsts-spec] and follows that document's notational and naming conventions.

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

1.1. About Notation

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.

This document includes some pseudocode examples written in a Python-like language, to clarify UA behavior. The examples assume that a global data structure, `hsts_metadata`, exists and contains the HSTS metadata that the UA has accumulated over time. It is indexable by domain name and includes the usual HSTS parameters (`maxAge`, `includesSubDomains`) as well as the new HSTS parameter, `pins`, that this document introduces. It also assumes a hypothetical X.509 datatype, denoted with a variable named "certificate", that includes likely X.509 fields such as `public_key` (which would correspond to the `SubjectPublicKeyInfo` field in a real X.509 certificate).

There are also some working code examples using the Python and Go languages.

The examples are intended to be illustrative, not necessarily precise or using algorithms that a real, optimized UA would employ.

2. Server and Client Behavior

To set a pin, HSTS Hosts use a new STS extension directive (`STS-d-ext`) in their HSTS response header field: `pins`. To enable pin revocation (Section 2.3), hosts may also use the new `breakv` and `breakc` directives.

```
STS-d-ext-pin      = "pins" OWS "=" OWS [fingerprints]
STS-d-ext-breakv   = "breakv" OWS "=" OWS fp-type "/" base64-digits
STS-d-ext-breakc   = "breakc" OWS "=" OWS base64-digits

fingerprints       = fingerprint
                    / fingerprint "," fingerprints

fingerprint        = fp-type "/" base64-digits

fp-type            = "sha1"
```

```
/ "sha256"
```

Figure 1

Here is an example response header field using the pins extension (folded for clarity):

```
Strict-Transport-Security: max-age=500; includeSubDomains;  
    pins=sha1/4n972HfV354KP560yw4uqe/baXc=,  
    sha1/IvGeLsbqzPxdI0b0wuj2xVTdXgc=
```

Figure 2

Here is an example response header field using both the pins and the breakv extensions (folded for clarity):

```
Strict-Transport-Security: max-age=500; includeSubDomains;  
    pins=sha1/4n972HfV354KP560yw4uqe/baXc=,  
    sha1/IvGeLsbqzPxdI0b0wuj2xVTdXgc=;  
    breakv=sha1/jUQEXH7Q2Ly+Xn/yFWJxAHT3fDc=
```

Figure 3

The fingerprint is the SHA-1 (or SHA-256) hash of the raw SubjectPublicKeyInfo field of the certificate, encoded in base-64 for brevity. We pin public keys, rather than entire certificates, to enable operators to generate new certificates containing old public keys (see [why-fingerprint-key]). (Although host operators may do this, certification authorities already do. Additionally, when UAs check certificate chains, they do so by checking that each certificate is signed by its parent's public key, making the public key -- not the certificate -- the essential identifier.)

See Appendix A for an example program that generates public key fingerprints from SubjectPublicKeyInfo fields in certificates.

The breakv directive communicates to UAs a pin break verifier, and the breakc directive communicates the pin break code. Hosts SHOULD generate pin break codes and verifiers. When present, UAs MUST note pin break verifiers and honor pin break codes. See Section 2.3 for a discussion of verifiers and codes.

2.1. Noting and Validating Pins

Upon receipt of this header field, the UA will note the HSTS Host as a Known Pinned HSTS Host. When connecting to a Known Pinned HSTS Host, the UA will compare the public key fingerprint(s) in the Host's certificate chain to the pinned fingerprints, and will fail closed unless at least one public key in the chain has a fingerprint matching one of the pinned fingerprints. (Following the HSTS specification, TLS errors for HSTS hosts must be hard, with no chance for the user to click through any warnings or errors. We treat fingerprint mismatch in the same way.)

Note that to validate pins, UAs must necessarily read the headers of a response. In case of mismatch, UAs SHOULD NOT read the response body as part of failing hard.

This pseudocode illustrates how UAs validate the certificate chains they receive from Known Pinned HSTS Hosts.

```
def chain_is_pinned_valid(chain, pins):
```

```

    for certificate in chain:
        for fingerprint in pins:
            if certificate.public_key.fingerprint == fingerprint:
                return True

    return False

# ...
if not chain_is_pinned_valid(request.tls_info.certificate_chain,
                             hsts_metadata[request.hostname].pins):
    request.fail()
# ...

```

Figure 4

The pin list appearing in an HSTS header MUST have at least one pin matching one of the public key fingerprints in the chain that was validated for the HTTPS connection. This defends against HTTP header injection attacks (see Section 3.4.1).

UAs MUST cache pins and pin break verifiers for Known Pinned HSTS Hosts, and MIGHT AS WELL do so in the same manner as other HSTS metadata. If the maxAge directive is present in the HSTS response header, the HSTS metadata -- including fingerprints in the pins directive -- expire at that time.

2.2. Interactions With Built-in HSTS Lists

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

Hosts can update built-in pin lists by using this extension. Similarly, UAs can update their built-in pin lists with software updates. In either case, UAs MUST use the newest information -- built-in or set via HSTS -- when validating certificate chains for the host.

2.3. Un-pinning

Hosts can enable pin revocation for their previously-pinned key fingerprints by setting pin break verifiers using the breakv directive. Then, when hosts want to break pins, they set the pin break code in their HSTS headers using the breakc directive. (This idea is due to Perrin in [pin-break-codes].)

Pin break codes are short random strings, kept secret until the host operator wants to break the pins. Pin break verifiers are simply hashes of the codes. Generating codes and verifiers, and verifying that codes match a previously set verifier, is trivial. See Figure 5.

```

def make_pin_break():
    code = os.urandom(16)
    verifier = hashlib.shal(code).digest()
    return base64.b64encode(code), base64.b64encode(verifier)

def verify_code(code, verifier):
    c = base64.b64decode(code)
    v = hashlib.shal(c).digest()
    return verifier == base64.b64encode(v)

```

```

if __name__ == "__main__":
    import sys

    if 1 == len(sys.argv):
        print make_pin_break()
    elif 3 == len(sys.argv):
        print verify_code(sys.argv[1], sys.argv[2])

```

Figure 5

Hosts can request that UAs forget pinned fingerprints by issuing a valid HSTS header containing the pin break code. UAs **MUST** forget all pinned fingerprints associated with the matching pin break verifier, and **MUST NOT** forget any pinned fingerprints not associated with that verifier.

In the event that a host sends an HSTS header containing a breakc that does not match a breakv the UA has previously noted, the UA **MUST** ignore that breakc and **MUST** process any pins or breakv directives as normal. This is so that hosts can break old pins but still successfully set new pins and verifiers in UAs that have not previously (or recently) noted the host.

Host operators **SHOULD** keep the pin break code secret, and **SHOULD** generate codes that are computationally infeasible to guess (such as by using their system's cryptographic random number generator; note that a 128-bit security level suffices).

2.4. Pinning Self-Signed Leaf Certificates

To the extent that UAs allow or enable hosts to 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

3.1. Deployment Guidance

To recover from disasters of various types, as described below, we recommend that HSTS Hosts follow these guidelines.

- o Operators **SHOULD** have a safety net: they should generate a backup key pair, get it signed by a different (root and/or intermediary) CA than their live certificate(s), store it safely offline, and set this backup pin in their pins directive.
- * Having a backup certificate was always a good idea anyway.
- 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** have a diverse certificate portfolio. They should pin to a few different roots, owned by different companies

if possible.

- o Operators SHOULD start small. Operators SHOULD first deploy HSTS certificate pinning by setting a maxAge of minutes or a few hours, and gradually increase maxAge as they gain confidence in their operational capability.

3.2. Disasters Relating to Compromises of Certificates

3.2.1. The private key for the pinned leaf is stolen

If a leaf certificate is compromised, the host is likely to have experienced a complete compromise, in which case the problem is greater than certificates and pins. See Section 3.4.2.

3.2.2. The root or intermediary CA is compromised

This disaster will affect many hosts (HSTS Hosts and other), and will likely require a client software update (e.g. to revoke the signing CA and/or the false certificates it issued).

If the operator has a backup pin whose signature chain is still valid, they should deploy it. In this case, the host need not even degrade from Known Pinned to Known.

3.3. Disasters Relating to Certificate Mismanagement

3.3.1. The leaf certificate expires

Operators should deploy their backup pin.

Note that when evaluating a pinned certificate, the UA MUST un-pin the fingerprint if the certificate has expired. If a pin list becomes empty, the UA downgrades the host from Known Pinned HSTS Host to Known HSTS Host. The usual HTTPS validation procedure now applies.

Operators should get any CA to sign a new cert with updated expiry, based on the existing, unchanged public key.

- o And/or, operators should deploy their backup pin and/or have a CA sign an all-new key.
- o Operators should continue to set pins in their HSTS header, and UAs will upgrade from Known HSTS Host to Known Pinned HSTS Host when the fingerprint(s) refer(s) to valid certificate(s) again.

3.3.2. The leaf certificate is lost

Operators should deploy their backup pin. Alternately, if they pinned to a root or intermediary signer, they should get a new leaf certificate signed by one of those signers.

Operators SHOULD attempt to get the certificate revoked by whatever means available (extant revocation mechanisms like CRL or OCSP, blacklisting in the UA, or future revocation mechanisms).

- o We know that extant revocation mechanisms are unreliable. Operators SHOULD NOT depend on them.

3.3.3. The CA is extorting the operator approaching renewal/expiry time

If the backup pin chains to a different signer, the operator should deploy it. (They should then get a new backup pin.)

The time running up to renewal can be used to serve additional HSTS public key hashes, pinning to new root CAs.

- o Hosts can also disable pinning altogether as described above.

If the host is pinned to leaves or its own intermediary, operators can simply get a different root CA to sign the existing public key.

If the operator fails to get new certs in time, and the host is pinned only to the one root CA, the solution is simple; see Section 3.3.1.

3.4. Disasters Relating to Vulnerabilities in the Known HSTS Host

3.4.1. The host is vulnerable to HTTP header injection

Note that header injection vulnerabilities are in general more severe than merely disabling pinning for individual users.

The attacker could set additional pins for certificates he controls, or pin break verifiers for codes he controls, allowing him to undetectably MITM clients. When or if the client is outside the scope of the attacker's MITM attack, the result is DoS.

The attacker could disable HSTS and pins.

3.4.2. The host suffers full server-side compromise

After setting up a new host, operators should deploy the backup pin. Alternately, if the host is pinned to a root or intermediary signer, the operator should get a new leaf certificate signed by one of those signers.

Operators SHOULD attempt to get the certificate containing the compromised private key revoked by whatever means available (extant revocation mechanisms like CRL or OCSP, blacklisting in the UA, or future revocation mechanisms).

- o We know that extant revocation mechanisms are unreliable. Do not depend on them.

4. Usability Considerations

When pinning works to detect impostor Known Pinned HSTS Hosts, users will experience denial of service. UAs SHOULD explain the reason why. If it happens that true positives (actual attacks) outnumber false positives (hosts bricking themselves by accident), the feature will gain a positive reputation. Note that pinning has started life with a good reputation because it provoked the discovery of the DigiNotar CA compromise. (When DigiNotar signed a certificate for *.google.com in August 2011, Chrome users discovered the attack due to the pre-loaded pins for Google domains.)

We believe that, in general, DoS is a better failure mode than user account/session compromise or other result of TLS compromise.

UAs MUST have a way for users to clear current pins that were set by HSTS. UAs SHOULD have a way for users to query the current state of

Known (Pinned) HSTS Hosts.

5. Economic Considerations

If pinning becomes common, host operators might become incentivized to choose CAs that get compromised less often, or respond better to compromise. This will require information to flow into the market, and for people to interpret no news post-compromise as bad news. Pinning itself will provide some of that information, as will sources like UA vendor communications, the EFF SSL Observatory, the Qualys SSL survey, etc.

The disaster recovery plans described above all incur new costs for host operators, and increase the size of the certificate market. Arguably, well-run hosts had already absorbed these costs because (e.g.) backup certificates from different CAs were necessary disaster recovery mechanisms even before certificate pinning. Small sites -- which although small might still need to provide good security -- may not be able to afford the disaster recovery mechanisms we recommend. (The cost of the backup certificate is not the issue; it is more the operational costs in safely storing the backup and testing that it works.) Thus, low-risk pinning may be available only to large sites; small sites may have to choose no pinning or potentially bricking their host (up to the maxAge window). This is not worse than the status quo.

6. Ideas

6.1. Requiring Backup Pins

Because bricking risk mitigation requires a backup pin, UAs could require that the pins directive have at least two fingerprints, at least one of which does not match any of the public keys in any of the certificates in the chain. (This idea due to Tom Sepez.)

6.2. Prepopulating Pin Lists

HSTS-based certificate pinning, unlike built-in pinning, suffers from the bootstrap problem. To work around this, we could pre-populate a built-in pin list with public keys as observed in the wild by one or more global observers, such as Googlebot, the EFF SSL Observatory, Convergence notaries, and so on.

One problem with this approach is that it does not involve host operators. It is best to get operator consent before signing them up for a potentially risky new protocol such as this. Therefore we leave this idea for work (including third-party UA extensions).

6.3. Tools to Assist Creation of Header

It would be good to provide tools that read X.509 certificate chains and generate example HSTS headers that operators can easily add to their webs erver configurations.

6.4. Pinning Subresources

Many hosts have pages that load subresources from domains not under the control, or under only partial control, of the main host's operators. For example, popular hosts often use CDNs, and CDN customers may have only limited, if any, ability to influence the

configuration of the CDN's servers. (This long-standing problem is independent of certificate pinning.)

To a limited extent, the includeSubDomains HSTS directive can address this: if the CDN host has a name that is a subdomain of the main host (e.g. assets-from-cdn.example.com points to CDN-owned servers), and if the main host's operators can guaranteeably keep up-to-date with the CDN's server certificate fingerprints -- perhaps as part of example.com's contract with the CDN -- then the problem may be solved.

CDNs SHOULD also use certificate pinning independently of any of their customers.

Although one can imagine an extension to this specification allowing the main resource to set pins for subresources in other domains, it is complex and fragile both from technical and business perspectives. The UA would have to accept those pins for the subresource domains ONLY when loading resources from the subdomains as part of a page load of the main host. The independence of the two domains' operations teams would still pose synchronization problems, and potentially increase the bricking risk.

Therefore, except in simple cases, this document leaves the cross-domain subresource problem to future work. Operational experience with HSTS-based certificate pinning should guide the development of a plan to handle the problem.

6.5. Pinning Without Requiring HTTPS

Some host operators would like to take advantage of certificate pinning without requiring HTTPS, but having clients require pins in the event that they do connect to the host with HTTPS. As specified above, the current HSTS-based mechanism does not allow for this: clients that receive the pins directive via HSTS will also therefore require HTTPS -- that is the purpose of HSTS after all. To have an additional directive, e.g. mode=optional, would not work because older clients that support HSTS but not the mode extension would effectively require HTTPS.

Alternatives include (a) putting the pins directive in a new header instead of extending HSTS; and (b) some kind of hack like setting maxAge=0 and having an additional directive to keep the pins alive (e.g. pinMaxAge). These alternatives seem ugly to us and we welcome suggestions for a better way to support this deployment scenario.

7. Acknowledgements

Thanks to Jeff Hodges, Adam Langley, Nicolas Lidzborski, SM, and Yoav Nir for suggestions and edits that clarified the text. Trevor Perrin for providing the pin break codes mechanism. Adam Langley provided the SPKI fingerprint generation code.

8. What's Changed

This is the first draft of this proposal submitted as an official Internet Draft.

9. References

[hsts-spec]
Hodges, J., Jackson, C., and A. Barth, "HTTP Strict Transport Security (HSTS)", August 2011, <<http://tools.ietf.org/html/draft-ietf-websec-strict-transport-sec-02>>.

[why-fingerprint-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/>>.

[rfc-2119]
Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", March 1997, <<http://www.ietf.org/rfc/rfc2119.txt>>.

Appendix A. Fingerprint Generation

This Go program generates public key fingerprints, suitable for use in pinning, from PEM-encoded certificates.

```
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 6

Authors' Addresses

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

Email: cevans@google.com

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

Email: palmer@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2011

J. Hodges
A. Steingruebl
PayPal
R. Barnes
BBN Technologies
Mar 7, 2011

Web Security Framework: Problem Statement and Requirements
draft-hodges-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 September 8, 2011.

Copyright Notice

Copyright (c) 2011 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.

This document may contain material from IETF Documents or IETF Contributions published or made publicly available before November 10, 2008. The person(s) controlling the copyright in some of this material may not have granted the IETF Trust the right to allow modifications of such material outside the IETF Standards Process. Without obtaining an adequate license from the person(s) controlling the copyright in such materials, this document may not be modified outside the IETF Standards Process, and derivative works of it may not be created outside the IETF Standards Process, except to format it for publication as an RFC or to translate it into languages other than English.

Table of Contents

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

[[TODOOn: Things to fix (where "n" in "TODOOn" 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 stipulates 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 attacks should be possible to mitigate via a web application security framework (see [44] for a definition and taxonomy of attacks):

1. cross-site-scripting (XSS) [2] [44]
2. Man-in-the-middle (MITM) attacks against "secure" (e.g. Transport Layer Security (TLS/SSL)-based [7] [8] [44]) web sites. For example, be able to subsume the HSTS header [11].
3. Cross-Site-Request Forgery (CSRF) [3] [44] (?)
4. Response Splitting [44]
5. more (ie eg from [44] ?) ?

5.2. Threats

Via the attacks above, an attacker can..

1. Obtain a victim's confidential web application credentials (e.g. 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 HTTP-only; 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 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

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 chooses 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/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=dxxqgkd_0cvcqhsw

[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\(office.10\).aspx](http://msdn.microsoft.com/en-us/library/aa157732(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

[44] Web Application Security Consortium, "The WASC Threat Classification v2.0," 2009.
http://projects.webappsec.org/f/WASC-TC-v2_0.pdf

12. Informative References

[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/>>.

Authors' Addresses

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

Email: Jeff.Hodges@PayPal.com

Andrew Steingruebl
PayPal
2211 North First Street
San Jose, California 95131
US

Email: Andy.Steingruebl@PayPal.com

Richard Barnes
BBN Technologies
9861 Broken Land Parkway
Columbia, MD 21046
USA

Phone: +1 410 290 6169
Email: rbarnes@bbn.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2011

J. Hodges
PayPal
C. Jackson
Carnegie Mellon University
A. Barth
Google, Inc.
March 14, 2011

HTTP Strict Transport Security (HSTS)
draft-ietf-websec-strict-transport-sec-01

Abstract

This specification defines a mechanism enabling Web sites to declare themselves accessible only via secure connections, and/or for users to be able to direct their user agent(s) to interact with given sites only over secure connections. This overall policy is referred to as HTTP Strict Transport Security (HSTS). The policy is declared by Web sites via the Strict-Transport-Security HTTP Response Header Field, and/or by other means, e.g. user agent configuration.

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 September 15, 2011.

Copyright Notice

Copyright (c) 2011 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	4
2. Overview	5
2.1. Use Cases	5
2.2. Strict Transport Security Policy Effects	5
2.3. Threat Model	5
2.3.1. Threats Addressed	5
2.3.1.1. Passive Network Attackers	6
2.3.1.2. Active Network Attackers	6
2.3.1.3. Web Site Development and Deployment Bugs	6
2.3.2. Threats Not Addressed	7
2.3.2.1. Phishing	7
2.3.2.2. Malware and Browser Vulnerabilities	7
2.4. Requirements	7
2.4.1. Overall Requirement	7
2.4.1.1. Detailed Core Requirements	8
2.4.1.2. Detailed Ancillary Requirements	9
3. Conformance Criteria	9
3.1. Document Conventions	9
4. Terminology	9
5. Syntax	12
5.1. Strict-Transport-Security HTTP Response Header Field	12
5.2. Examples	14
6. Server Processing Model	14
6.1. HTTP-over-Secure-Transport Request Type	15
6.2. HTTP Request Type	15
7. User Agent Processing Model	16
7.1. Strict-Transport-Security Response Header Field Processing	16
7.1.1. Noting a HSTS Host	17
7.1.2. Known HSTS Host Domain Name Matching	17
7.2. URI Loading and Port Mapping	18
7.3. Errors in Secure Transport Establishment	18
7.4. HTTP-Equiv <Meta> Element Attribute	19
8. Domain Name ToASCII Conversion Operation	19
9. Server Implementation Advice	19
10. UA Implementation Advice	20
11. Constructing an Effective Request URI	21
12. Security Considerations	23
12.1. Denial of Service (DoS)	23

12.2. Bootstrap MITM Vulnerability	23
12.3. Network Time Attacks	23
12.4. Bogus Root CA Certificate Phish plus DNS Cache Poisoning Attack	23
13. IANA Considerations	24
14. Design Decision Notes	24
15. References	25
15.1. Normative References	25
15.2. Informative References	26
Appendix A. Acknowledgments	27
Appendix B. Change Log	28
B.1. For draft-ietf-websec-strict-transport-sec	28
B.2. For draft-hodges-strict-transport-sec	28
Authors' Addresses	29

1. Introduction

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

The HTTP protocol [RFC2616] may be used over various transports, typically the Transmission Control Protocol (TCP) [RFC0793]. However, TCP does not provide channel integrity protection, confidentiality, nor secure host identification. Thus the Secure Sockets Layer (SSL) protocol [I-D.ietf-tls-ssl-version3] and its successor Transport Layer Security (TLS) [RFC4346], were developed in order to provide channel-oriented security, and are typically layered between application protocols and TCP. [RFC2818] specifies how HTTP is layered onto TLS, and defines the Universal Resource Identifier (URI) scheme of "https" (in practice however, HTTP user agents (UAs) typically offer their users choices among SSL2, SSL3, and TLS for secure transport). URIs themselves are specified in [RFC3986].

UAs employ various local security policies with respect to the characteristics of their interactions with web resources depending on (in part) whether they are communicating with a given web resource using HTTP or HTTP-over-a-Secure-Transport. For example, cookies ([RFC2109] and [RFC2965]) may be flagged as Secure. UAs are to send such Secure cookies to their addressed host only over a secure transport. This is in contrast to non-Secure cookies, which are returned to the host regardless of transport (although modulo other rules).

UAs typically announce to their users any issues with secure connection establishment, such as being unable to validate a TLS server certificate trust chain, or if a TLS server certificate is expired, or if a TLS server's domain name appears incorrectly in the TLS server certificate (see section 3.1 of [RFC2818]). Often, UAs enable users to elect to continue to interact with a web resource in the face of such issues. This behavior is sometimes referred to as "click(ing) through" security [GoodDhamijaEtAl05] [SunshineEgelmanEtAl09], and thus can be described as "click-through insecurity" .

Jackson and Barth proposed an approach, in [ForceHTTPS], to enable web sites and/or users to declare that such issues are to be treated as fatal and without direct user recourse. The aim is to prevent users from unintentionally downgrading their security.

This specification embodies and refines the approach proposed in [ForceHTTPS], e.g. a HTTP response header field is used to convey site policy to the UA rather than a cookie.

2. Overview

This section discusses the use cases, summarizes the HTTP Strict Transport Security (HSTS) policy, and continues with a discussion of the threat model, non-addressed threats, and derived requirements.

2.1. Use Cases

The high-level use case is a combination of:

- o Web browser user wishes to discover, or be introduced to, and/or utilize various web sites (some arbitrary, some known) in a secure fashion.
- o Web site deployer wishes to offer their site in an explicitly secure fashion for both their own, as well as their users', benefit.

2.2. Strict Transport Security Policy Effects

The characteristics of the HTTP Strict Transport Security policy, as applied by a UA in its interactions with a web site wielding HSTS Policy, known as a HSTS Host, is summarized as follows:

1. All insecure ("http") connections to a HSTS Host are redirected by the HSTS Host to be secure connections ("https").
2. The UA terminates, without user recourse, any secure transport connection attempts upon any and all secure transport errors or warnings, including those caused by a site presenting self-signed certificates.
3. UAs transform insecure URI references to a HSTS Host into secure URI references before dereferencing them.

2.3. Threat Model

HSTS is concerned with three threat classes: passive network attackers, active network attackers, and imperfect web developers. However, it is explicitly not a remedy for two other classes of threats: phishing and malware. Addressed and not addressed threats are briefly discussed below. Readers may wish refer to [ForceHTTPS] for details as well as relevant citations.

2.3.1. Threats Addressed

2.3.1.1. Passive Network Attackers

When a user browses the web on a local wireless network (e.g. an 802.11-based wireless local area network) a nearby attacker can possibly eavesdrop on the user's unencrypted Internet Protocol-based connections, such as HTTP, regardless of whether or not the local wireless network itself is secured [BeckTews09]. Freely available wireless sniffing toolkits, e.g. [Aircrack-ng], enable such passive eavesdropping attacks. A passive network attacker using such tools can steal session identifiers and hijack the user's web session(s), by obtaining cookies containing authentication credentials for example [ForceHTTPS].

To mitigate such threats, some Web sites support, but usually do not force, access using end-to-end secure transport -- e.g. signaled through URIs constructed with the "https" scheme [RFC2818]. This can lead users to believe that accessing such services using secure transport protects them from passive network attackers. Unfortunately, this is often not the case in real-world deployments as session identifiers are often stored in non-Secure cookies to permit interoperability with versions of the service offered over insecure transport ("Secure cookies" are those cookies containing the "Secure" attribute [RFC2109]). For example, if the session identifier for a web site (an email service, say) is stored in a non-Secure cookie, it permits an attacker to hijack the user's session if the user's UA makes a single insecure HTTP request to the site.

2.3.1.2. Active Network Attackers

A determined attacker can mount an active attack, either by impersonating a user's DNS server or, in a wireless network, by spoofing network frames or offering a similarly-named evil twin access point. If the user is behind a wireless home router, an attacker can attempt to reconfigure the router using default passwords and other vulnerabilities. Some sites, such as banks, rely on end-to-end secure transport to protect themselves and their users from such active attackers. Unfortunately, browsers allow their users to easily opt-out of these protections in order to be usable for sites that incorrectly deploy secure transport, for example by generating and self-signing their own certificates (without also distributing their CA certificate to their users' browsers).

2.3.1.3. Web Site Development and Deployment Bugs

The security of an otherwise uniformly secure site (i.e. all of its content is materialized via "https" URIs), can be compromised completely by an active attacker exploiting a simple mistake, such as the loading of a cascading style sheet or a SWF movie over an

insecure connection (both cascading style sheets and SWF movies can script the embedding page, to the surprise of many web developers -- most browsers do not issue mixed content warnings when insecure SWF files are embedded). Even if the site's developers carefully scrutinize their login page for mixed content, a single insecure embedding anywhere on the site compromises the security of their login page because an attacker can script (control) the login page by injecting script into the page with mixed content.

Note: "Mixed content" here refers to the same notion referred to as "mixed security context" later elsewhere in this specification.

2.3.2. Threats Not Addressed

2.3.2.1. Phishing

Phishing attacks occur when an attacker solicits authentication credentials from the user by hosting a fake site located on a different domain than the real site, perhaps driving traffic to the fake site by sending a link in an email message. Phishing attacks can be very effective because users find it difficult to distinguish the real site from a fake site. HSTS is not a defense against phishing per se; rather, it complements many existing phishing defenses by instructing the browser to protect session integrity and long-lived authentication tokens [ForceHTTPS].

2.3.2.2. Malware and Browser Vulnerabilities

Because HSTS is implemented as a browser security mechanism, it relies on the trustworthiness of the user's system to protect the session. Malicious code executing on the user's system can compromise a browser session, regardless of whether HSTS is used.

2.4. Requirements

This section identifies and enumerates various requirements derived from the use cases and the threats discussed above, and lists the detailed core requirements HTTP Strict Transport Security addresses, as well as ancillary requirements that are not directly addressed.

2.4.1. Overall Requirement

- o Minimize the risks to web browser users and web site deployers that are derived from passive and active network attackers, web site development and deployment bugs, as well as insecure user actions.

2.4.1.1. Detailed Core Requirements

These core requirements are derived from the overall requirement, and are addressed by this specification.

1. Web sites need to be able to declare to UAs that they should be interacted with using a strict security policy.
2. Web sites need to be able to instruct UAs that contact them insecurely to do so securely.
3. UAs need to note web sites that signal strict security policy enablement, for a web site declared time span.
4. UAs need to re-write all insecure UA "http" URI loads to use the "https" secure scheme for those web sites for which secure policy is enabled.
5. Web site administrators need to be able to signal strict security policy application to subdomains of higher-level domains for which strict security policy is enabled, and UAs need to enforce such policy.
6. For example, both example.com and foo.example.com could set policy for bar.foo.example.com.
7. UAs need to disallow security policy application to peer domains, and/or higher-level domains, by domains for which strict security policy is enabled.
8. For example, neither bar.foo.example.com nor foo.example.com can set policy for example.com, nor can bar.foo.example.com set policy for foo.example.com. Also, foo.example.com cannot set policy for sibling.example.com.
9. UAs need to prevent users from clicking-through security warnings. Halting connection attempts in the face of secure transport exceptions is acceptable.

Note: A means for uniformly securely meeting the first core requirement above is not specifically addressed by this specification (see Section 12.2 "Bootstrap MITM Vulnerability"). It may be addressed by a future revision of this specification or some other specification. Note also that there are means by which UA implementations may more fully meet the first core requirement, see Section 10 "UA Implementation Advice".

2.4.1.2. Detailed Ancillary Requirements

These ancillary requirements are also derived from the overall requirement. They are not normatively addressed in this specification, but could be met by UA implementations at their implementor's discretion, although meeting these requirements may be complex.

1. Disallow "mixed security context" (also known as "mixed-content") loads (see section 5.3 "Mixed Content" in [W3C.WD-wsc-ui-20100309]).
2. Facilitate user declaration of web sites for which strict security policy is enabled, regardless of whether the sites signal HSTS Policy.

3. Conformance Criteria

This specification is written for hosts and user agents (UAs).

In this specification, the words MUST, MUST NOT, MAY, and SHOULD are to be interpreted as described in [RFC2119].

A conformant host is one that implements all the requirements listed in this specification that are applicable to hosts.

A conformant user agent is one that implements all the requirements listed in this specification that are applicable to user agents.

3.1. Document Conventions

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

Warning: This is how a warning is shown. These are things that can have suboptimal downside risks if not heeded.

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

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

4. Terminology

Terminology is defined in this section.

ASCII case-insensitive comparison

means comparing two strings exactly, codepoint for codepoint, except that the characters in the range U+0041 .. U+005A (i.e. LATIN CAPITAL LETTER A to LATIN CAPITAL LETTER Z) and the corresponding characters in the range U+0061 .. U+007A (i.e. LATIN SMALL LETTER A to LATIN SMALL LETTER Z) are considered to also match. See [Unicode5] for details.

codepoint

is a colloquial contraction of Code Point, which is any value in the Unicode codespace; that is, the range of integers from 0 to 10FFFF(hex) [Unicode5].

Domain Name

Domain Names, also referred to as DNS Names, are defined in [RFC1035] to be represented outside of the DNS protocol itself (and implementations thereof) as a series of labels separated by dots, e.g. "example.com" or "yet.another.example.org". In the context of this specification, Domain Names appear in that portion of a URI satisfying the reg-name production in "Appendix A. Collected ABNF for URI" in [RFC3986], and the host component from the Host HTTP header field production in section 14.23 of [RFC2616].

Note: The Domain Names appearing in actual URI instances and matching the aforementioned production components may or may not be FQDNs.

Domain Name Label

is that portion of a Domain Name appearing "between the dots", i.e. consider "foo.example.com": "foo", "example", and "com" are all domain name labels.

Effective Request URI

is a URI that can be constructed by an HTTP host for any given HTTP request sent to it. Some HTTP requests do not contain a contiguous representation of the URI identifying the resource being addressed by the HTTP request. Rather, different portions of a resource's URI may be mapped to both the Request-Line header field and the Host header field in an HTTP request message [I-D.ietf-httpbis-pl-messaging]. The HTTP server coalesces these URI fragments and constructs an equivalent of the Request-URI that was used by the UA to generate the received HTTP request message.

See Section 11 "Constructing an Effective Request URI", below.

FQDN is an acronym for Fully-qualified Domain Name. A FQDN is a Domain Name that includes all higher level domains relevant to the named entity (typically a HSTS Host in the context of this specification). If one thinks of the DNS as a tree-structure with each node having its own Domain Name Label, a FQDN for a specific node would be its label followed by the labels of all the other nodes between it and the root of the tree. For example, for a host, a FQDN would include the label that identifies the particular host, plus all domains of which the host is a part, up to and including the top-level domain (the root domain is always null) [RFC1594].

HTTP Strict Transport Security is the overall name for the combined UA- and server-side security policy defined by this specification.

HTTP Strict Transport Security Host is a HTTP host implementing the server aspects of the HSTS policy.

HTTP Strict Transport Security Policy is the name of the combined overall UA- and server-side facets of the behavior specified in this specification.

HSTS See HTTP Strict Transport Security.

HSTS Host See HTTP Strict Transport Security Host.

HSTS Policy See HTTP Strict Transport Security Policy.

Known HSTS Host is a HSTS Host for which the UA has a HSTS Policy in effect.

Local policy is comprised of policy rules deployers specify and which are often manifested as "configuration settings".

MITM is an acronym for man-in-the-middle. See "man-in-the-middle attack" in [RFC4949].

Request URI is the URI used to cause a UA to issue an HTTP request message.

UA is a an acronym for user agent. For the purposes of this specification, a UA is an HTTP client application typically actively manipulated by a user [RFC2616] .

5. Syntax

This section defines the syntax of the new header this specification introduces. It also provides a short description of the function the header.

The Section 6 "Server Processing Model" section details how hosts are to use this header. Likewise, the Section 7 "User Agent Processing Model" section details how user agents are to use this header.

5.1. Strict-Transport-Security HTTP Response Header Field

The Strict-Transport-Security HTTP response header field indicates to a UA that it MUST enforce the HSTS Policy in regards to the host emitting the response message containing this header field.

The ABNF syntax for the Strict-Transport-Security HTTP Response Header field is:


```

Strict-Transport-Security =

    "Strict-Transport-Security" ":" OWS STS-v OWS

; value
STS-v      = STS-d
           / STS-d *( OWS ";" OWS STS-d OWS )

; STS directive
STS-d      = STS-d-cur / STS-d-ext

; defined STS directives
STS-d-cur  = maxAge / includeSubDomains

maxAge     = "max-age" OWS "=" OWS delta-seconds v-ext

includeSubDomains = [ "includeSubDomains" ] v-ext

; extension points
STS-d-ext  = name          ; STS extension directive

v-ext     = value          ; STS extension value

name       = token

value      = OWS / %x21-3A / %x3C-7E ; i.e. optional white space, or
           ; [ ! .. : ] [ < .. ~ ] any visible chars other than ";"

; productions imported from [ID.ietf-httpbis-pl-messaging]:

token

OWS        ; Optional White Space

```

Note: [I-D.ietf-httpbis-pl-messaging] is used as the ABNF basis in order to ensure that the new header has equivalent parsing rules to the header fields defined in that same specification. Also:

1. Quoted-string literals in the above ABNF stanza are case-insensitive.
2. In order to correctly match the grammar above, the Strict-Transport-Security HTTP Response Header MUST include at least a max-age directive with at least a single-digit value for delta-seconds.

max-age specifies the number of seconds, after the reception of the Strict-Transport-Security HTTP Response Header, during which the UA regards the host the message was received from as a Known HSTS Host (see also Section 7.1.1 "Noting a HSTS Host", below). The delta-seconds production is specified in [RFC2616].

[[TODO1: The above para wrt max-age may need further refinement.
--JeffH]]

includeSubDomains is a flag which, if present, signals to the UA that the HSTS Policy applies to this HSTS Host as well as any subdomains of the host's FQDN.

5.2. Examples

The below HSTS header field stipulates that the HSTS policy is to remain in effect for one year (there are approximately 31 536 000 seconds in a year), and the policy applies only to the domain of the HSTS Host issuing it:

```
Strict-Transport-Security: max-age=31536000
```

The below HSTS header field stipulates that the HSTS policy is to remain in effect for approximately six months and the policy applies only to the domain of the issuing HSTS Host and all of its subdomains:

```
Strict-Transport-Security: max-age=15768000 ; includeSubDomains
```

6. Server Processing Model

This section describes the processing model that HSTS Hosts implement. The model is comprised of two facets: the first being the processing rules for HTTP request messages received over a secure transport (e.g. TLS [RFC4346], SSL [I-D.ietf-tls-ssl-version3], or perhaps others, the second being the processing rules for HTTP request messages received over non-secure transports, i.e. over TCP/IP [RFC0793].

6.1. HTTP-over-Secure-Transport Request Type

When replying to an HTTP request that was conveyed over a secure transport, a HSTS Host SHOULD include in its response message a Strict-Transport-Security HTTP Response Header that MUST satisfy the grammar specified above in Section 5.1 "Strict-Transport-Security HTTP Response Header Field". If a Strict-Transport-Security HTTP Response Header is included, the HSTS Host MUST include only one such header.

Note: Including the Strict-Transport-Security HTTP Response Header is stipulated as a "SHOULD" in order to accomodate various server- and network-side caches and load-balancing configurations where it may be difficult to uniformly emit Strict-Transport-Security HTTP Response Headers on behalf of a given HSTS Host.

Establishing a given host as a Known HSTS Host, in the context of a given UA, MAY be accomplished over the HTTP protocol by correctly returning, per this specification, at least one valid Strict-Transport-Security HTTP Response Header to the UA. Other mechanisms, such as a client-side pre-loaded Known HSTS Host list MAY also be used. E.g. see Section 10 "UA Implementation Advice".

6.2. HTTP Request Type

If a HSTS Host receives a HTTP request message over a non-secure transport, it SHOULD send a HTTP response message containing a Status-Code of 301 and a Location header field value containing either the HTTP request's original Effective Request URI (see Section 11 "Constructing an Effective Request URI", below) altered as necessary to have a URI scheme of "https", or a URI generated according to local policy (which SHOULD employ a URI scheme of "https").

Note: The above behavior is a "SHOULD" rather than a "MUST" because:

There are risks in server-side non-secure-to-secure redirects [owaspTLSSGuide].

Site deployment characteristics -- e.g. a site that incorporates third-party components may not behave correctly when doing server-side non-secure-to-secure redirects in the case of being accessed over non-secure transport, but does behave correctly when accessed uniformly over secure transport. The latter is the case given a HSTS-capable UA that has already noted

the site as a Known HSTS Host (by whatever means, e.g. prior interaction or UA configuration).

[[XXX1: perhaps the "SHOULD" in the above behavior should be a "MAY" given the reasons it's presently not a "MUST". --JeffH]]

A HSTS Host MUST NOT include the Strict-Transport-Security HTTP Response Header in HTTP responses conveyed over non-secure transport.

7. User Agent Processing Model

This section describes the HTTP Strict Transport Security processing model for UAs. There are several facets to the model, enumerated by the following subsections.

Also, this processing model assumes that all Domain Names manipulated in this specification's context are already in ASCII Compatible Encoding (ACE) format as specified in [RFC3490]. If this is not the case in some situation, use the operation given in Section 8 "Domain Name ToASCII Conversion Operation" to convert any encountered internationalized Domain Names to ACE format before processing them.

7.1. Strict-Transport-Security Response Header Field Processing

If an HTTP response, received over a secure transport, includes a Strict-Transport-Security HTTP Response Header field, conforming to the grammar specified in Section 5.1 "Strict-Transport-Security HTTP Response Header Field" (above), and there are no underlying secure transport errors or warnings, the UA MUST either:

- o Note the host as a Known HSTS Host if it is not already so noted (see Section 7.1.1 "Noting a HSTS Host", below),

or,

- o Update its cached information for the Known HSTS Host if the max-age and/or includeSubDomains header field value tokens are conveying information different than that already maintained by the UA.

Note: The max-age value is essentially a "time to live" value relative to the reception time of the Strict-Transport-Security HTTP Response Header.

[[TODO2: Decide UA behavior in face of encountering multiple HSTS headers in a message. Use first header? Last? --JeffH]]

Otherwise:

- o If an HTTP response is received over insecure transport, the UA MUST ignore any present Strict-Transport-Security HTTP Response Header(s).
- o The UA MUST ignore any Strict-Transport-Security HTTP Response Headers not conforming to the grammar specified in Section 5.1 "Strict-Transport-Security HTTP Response Header Field" (above).

7.1.1. Noting a HSTS Host

If the substring matching the host production from the Request-URI, that the host responded to, 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 HSTS Host.

Otherwise, if the substring does not congruently match a presently known HSTS Host, per the matching procedure specified in Section 7.1.2 "Known HSTS Host Domain Name Matching" below, then the UA MUST note this host as a Known HSTS Host, caching the HSTS 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 the includeSubDomains flag is asserted or not.

7.1.2. Known HSTS Host Domain Name Matching

A UA determines whether a Domain Name represents a Known HSTS Host by looking for a match between the query Domain Name and the UA's set of Known HSTS Hosts.

1. Compare the query Domain Name string with the Domain Names of the UA's set of Known HSTS Hosts. For each Known HSTS Host's Domain Name, the comparison is done with the query Domain Name label-by-label using an ASCII case-insensitive comparison beginning with the rightmost label, and continuing right-to-left, and ignoring separator characters (see clause 3.1(4) of [RFC3986]).
 - * If a label-for-label match between an entire Known HSTS Host's Domain Name and a right-hand portion of the query Domain Name is found, then the Known HSTS Host's Domain Name is a superdomain match for the query Domain Name.

For example:

Query Domain Name: bar.foo.example.com

Superdomain matched

Known HSTS Host DN: foo.example.com

At this point, the query Domain Name is ascertained to effectively represent a Known HSTS Host. There may also be additional matches further down the Domain Name Label tree, up to and including a congruent match.

- * If a label-for-label match between a Known HSTS Host's Domain Name and the query domain name is found, i.e. there are no further labels to compare, then the query Domain Name congruently matches this Known HSTS Host.

For example:

Query Domain Name: foo.example.com

Congruently matched

Known HSTS Host DN: foo.example.com

The query Domain Name is ascertained to represent a Known HSTS Host. However, if there are also superdomain matches, the one highest in the tree asserts the HSTS Policy for this Known HSTS Host.

- * Otherwise, if no matches are found, the query Domain Name does not represent a Known HSTS Host.

7.2. URI Loading and Port Mapping

Whenever the UA prepares to "load", also known as "dereference", any URI where the host component of the authority component of the URI [RFC3986] matches that of a Known HSTS Host -- either as a congruent match or as a superdomain match where the superdomain Known HSTS Host has includeSubDomains asserted -- and the URI's scheme is "http", then the UA MUST replace the URI scheme with "https" before proceeding with the load.

Additionally, if the URI contains a port component [RFC3986] equal to "80", the UA MUST covert the port component to be "443". Otherwise, a present port component MUST be preserved.

7.3. Errors in Secure Transport Establishment

When connecting to a Known HSTS Host, the UA MUST terminate the connection with no user recourse if there are any errors (e.g. certificate errors), whether "warning" or "fatal" or any other error

level, with the underlying secure transport.

7.4. HTTP-Equiv <Meta> Element Attribute

UAs MUST NOT heed `http-equiv="Strict-Transport-Security"` attribute settings on <meta> elements in received content.

8. Domain Name ToASCII Conversion Operation

This operation converts a string-serialized Domain Name possibly containing arbitrary Unicode characters [Unicode5] into a string-serialized Domain Name in ASCII Compatible Encoding (ACE) format as specified in [RFC3490].

The operation is:

- o Apply the IDNA conversion operation (section 4 of [RFC3490]) to the string, selecting the ToASCII operation and setting both the AllowUnassigned and UseSTD3ASCIIRules flags.

9. Server Implementation Advice

HSTS Policy expiration time considerations:

- o Server implementations and deploying web sites need to consider whether they are setting an expiry time that is a constant value into the future, e.g. by constantly sending the same max-age value to UAs. For example:

`Strict-Transport-Security: max-age=778000`

A max-age value of 778000 is 90 days. Note that each receipt of this header by a UA will require the UA to update its notion of when it must delete its knowledge of this Known HSTS Host. The specifics of how this is accomplished is out of the scope of this specification.

- o Or, whether they are setting an expiry time that is a fixed point in time, e.g. by sending max-age values that represent the remaining time until the expiry time.
- o A consideration here is whether a deployer wishes to have signaled HSTS Policy expiry time match that for the web site's domain certificate.

Considerations for using HTTP Strict Transport Security in

conjunction with self-signed public-key certificates:

- o If a web site/organization/enterprise is generating their own secure transport public-key certificates for web sites, and that organization's root certificate authority (CA) certificate is not typically embedded by default in browser CA certificate stores, and if HSTS Policy is enabled on a site identifying itself using a self-signed certificate, then secure connections to that site will fail without user recourse, per the HSTS design. This is to protect against various active attacks, as discussed above.
- o However, if said organization strongly wishes to employ self-signed certificates, and their own CA in concert with HSTS, they can do so by deploying their root CA certificate to their users' browsers. There are various ways in which this can be accomplished (details are out of scope for this specification). Once their root CA cert is installed in the browsers, they may employ HSTS Policy on their site(s).

Note: Interactively distributing root CA certs to users, e.g. via email, and having the users install them, is arguably training the users to be susceptible to a possible form of phishing attack, see Section 12.4 "Bogus Root CA Certificate Phish plus DNS Cache Poisoning Attack".

10. UA Implementation Advice

In order to provide users and web sites more effective protection, UA implementors should consider including features such as:

- o Disallowing "mixed security context" (also known as "mixed-content") loads (see section 5.3 "Mixed Content" in [W3C.WD-wsc-ui-20100309]).

Note: In order to provide behavioral uniformity across UA implementations, the notion of mixed security context aka mixed-content will require (further) standardization work, e.g. to more clearly define the term(s) and to define specific behaviors with respect to it.

In order to provide users effective controls for managing their UA's caching of HSTS Policy, UA implementors should consider including features such as:

- o Ability to delete UA's cached HSTS Policy on a per HSTS Host basis.

Note: Adding such a feature should be done very carefully in both the user interface and security senses. Deleting a cache entry for a Known HSTS Host should be a very deliberate and well-considered act -- it shouldn't be something users get used to just "clicking through" in order to get work done. Also, it shouldn't be possible for an attacker to inject script into the UA that silently and programmatically removes entries from the UA's cache of Known HSTS Hosts.

In order to provide users and web sites more complete protection, UAs could offer advanced features such as these:

- o Ability for users to explicitly declare a given Domain Name as representing a HSTS Host, thus seeding it as a Known HSTS Host before any actual interaction with it. This would help protect against the Section 12.2 "Bootstrap MITM Vulnerability".

Note: Such a feature is difficult to get right on a per-site basis -- see the discussion of "rewrite rules" in section 5.5 of [ForceHTTPS]. For example, arbitrary web sites may not materialize all their URIs using the "https" scheme, and thus could "break" if a UA were to attempt to access the site exclusively using such URIs. Also note that this feature would complement, but is independent of the following described facility.

- o Facility whereby web site administrators can have UAs pre-configured with HSTS Policy for their site(s) by the UA vendor(s) -- in a manner similar to how root CA certificates are embedded in browsers "at the factory". This would help protect against the Section 12.2 "Bootstrap MITM Vulnerability".

Note: Such a facility complements the preceding described feature.

[[XXX2: These latter items beg the question of having some means of secure web site metadata and policy discovery and acquisition. There is extant work that may be of interest, e.g. the W3C POWDER work, OASIS XRI/XRD work (as well as XRDS-Simple), and "Link-based Resource Descriptor Discovery" (draft-hammer-discovery). --JeffH]]

11. Constructing an Effective Request URI

This section specifies how an HSTS Host must construct the Effective Request URI for a received HTTP request.

The first line of an HTTP request message is specified by the

following ABNF ([I-D.ietf-httpbis-pl-messaging] section 4.1):

```
Request-Line = Method SP request-target SP HTTP-Version CRLF
```

The request-target is following ABNF ([I-D.ietf-httpbis-pl-messaging] section 4.1.2):

```
request-target = "*"
                / absolute-URI
                / ( path-absolute [ "?" query ] )
                / authority
```

Additionally, many HTTP requests contain an additional Host request header field. It is specified by the following ABNF ([I-D.ietf-httpbis-pl-messaging] section 4.1.2):

```
Host = "Host:" OWS Host-v
Host-v = uri-host [ ":" port ]
```

Thus an example HTTP message containing the above header fields is:

```
GET /hello.txt HTTP/1.1
Host: www.example.com
```

Another example is:

```
GET HTTP://www.example.com/hello.txt HTTP/1.1
```

An HSTS Host constructs the Effective Request URI using the following ABNF grammar (which imports some productions from the above ABNF for Request-Line, request-target, and Host):

```
Effective-Request-URI = absolute-URI-present / path-absolute-form
```

```
absolute-URI-present = absolute-URI
```

```
path-absolute-form = scheme "://" Host-v path-absolute [ "?" query ]
```

where:

```
scheme is &quot;http&quot;; if the request was received over
insecure transport, or scheme is &quot;https&quot;; if the
request was received over secure transport.
```

For example, if the request message contains a request-target component that matches the grammar of absolute-URI, then the

Effective-Request-URI is simply the value of the absolute-URI component. Otherwise, the Effective-Request-URI is a combination, per the path-absolute-form production, of the Host-v, path-absolute, and query components from the request-target and Host components of the request message.

[[TODO3: This is a first SWAG at this section. Fix/add prose as appropriate, fix ABNF as needed per review. --JeffH]]

12. Security Considerations

12.1. Denial of Service (DoS)

HSTS could be used to mount certain forms of DoS attacks, where attackers set fake HSTS headers on legitimate sites available only insecurely (e.g. social network service sites, wikis, etc.).

12.2. Bootstrap MITM Vulnerability

The bootstrap MITM (Man-In-The-Middle) vulnerability is a vulnerability users and HSTS Hosts encounter in the situation where the user manually enters, or follows a link, to a HSTS Host using a "http" URI rather than a "https" URI. Because the UA uses an insecure channel in the initial attempt to interact with the specified serve, such an initial interaction is vulnerable to various attacks [ForceHTTPS] .

Note: There are various features/facilities that UA implementations may employ in order to mitigate this vulnerability. Please see Section 10 UA Implementation Advice.

12.3. Network Time Attacks

Active network attacks can subvert network time protocols (like NTP) - making this header less effective against clients that trust NTP and/or lack a real time clock. Network time attacks are therefore beyond the scope of the defense. Note that modern operating systems use NTP by default.

12.4. Bogus Root CA Certificate Phish plus DNS Cache Poisoning Attack

If an attacker can convince users of, say, <https://bank.example.com> (which is protected by HSTS Policy), to install their own version of a root CA certificate purporting to be bank.example.com's CA, e.g. via a phishing email message with a link to such a certificate -- then, if they can perform an attack on the users' DNS, e.g. via cache poisoning, and turn on HSTS Policy for their fake bank.example.com

site, then they have themselves some new users.

13. IANA Considerations

Below is the Internet Assigned Numbers Authority (IANA) Provisional Message Header Field registration information per [RFC3864].

Header field name:	Strict-Transport-Security
Applicable protocol:	HTTP
Status:	provisional
Author/Change controller:	TBD
Specification document(s):	this one

14. Design Decision Notes

This appendix documents various design decisions.

1. Cookies aren't appropriate for HSTS Policy expression as they are potentially mutable (while stored in the UA), therefore an HTTP header field is employed.
2. We chose to not attempt to specify how "mixed security context loads" (aka "mixed-content loads") are handled due to UA implementation considerations as well as classification difficulties.
3. A HSTS Host may update UA notions of HSTS Policy via new HSTS header field values. We chose to have UAs honor the "freshest" information received from a server because there is the chance of a web site sending out an erroneous HSTS Policy, such as a multi-year max-age value, and/or an incorrect includeSubDomains flag. If the HSTS Host couldn't correct such errors over protocol, it would require some form of annunciation to users and manual intervention on their part, which could be a non-trivial problem.
4. HSTS Hosts are identified only via Domain Names -- explicit IP address identification of all forms is excluded. This is for simplification and also is in recognition of various issues with using direct IP address identification in concert with PKI-based security.

15. References

15.1. Normative References

- [I-D.ietf-httpbis-pl-messaging]
Fielding, R., Gettys, J., Mogul, J., Nielsen, H.,
Masinter, L., Leach, P., Berners-Lee, T., and J. Reschke,
"HTTP/1.1, part 1: URIs, Connections, and Message
Parsing", draft-ietf-httpbis-pl-messaging-09 (work in
progress), March 2010.
- [RFC1035] Mockapetris, P., "Domain names - implementation and
specification", STD 13, RFC 1035, November 1987.
- [RFC1594] Marine, A., Reynolds, J., and G. Malkin, "FYI on Questions
and Answers - Answers to Commonly asked "New Internet
User" Questions", RFC 1594, March 1994.
- [RFC1983] Malkin, G., "Internet Users' Glossary", RFC 1983,
August 1996.
- [RFC2109] Kristol, D. and L. Montulli, "HTTP State Management
Mechanism", RFC 2109, February 1997.
- [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.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.
- [RFC2965] Kristol, D. and L. Montulli, "HTTP State Management
Mechanism", RFC 2965, October 2000.
- [RFC3454] Hoffman, P. and M. Blanchet, "Preparation of
Internationalized Strings ("stringprep")", RFC 3454,
December 2002.
- [RFC3490] Faltstrom, P., Hoffman, P., and A. Costello,
"Internationalizing Domain Names in Applications (IDNA)",
RFC 3490, March 2003.
- [RFC3492] Costello, A., "Punycode: A Bootstring encoding of Unicode
for Internationalized Domain Names in Applications
(IDNA)", RFC 3492, March 2003.
- [RFC3864] Klyne, G., Nottingham, M., and J. Mogul, "Registration
Procedures for Message Header Fields", BCP 90, RFC 3864,

September 2004.

- [RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.
- [RFC4346] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.1", RFC 4346, April 2006.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
- [Unicode5] The Unicode Consortium, "The Unicode Standard, Version 5.0", Boston, MA, Addison-Wesley ISBN 0-321-48091-0, 2007.
- [W3C.WD-html5-20100304] Hyatt, D. and I. Hickson, "HTML5", World Wide Web Consortium WD WD-html5-20100304, March 2010, <<http://www.w3.org/TR/2010/WD-html5-20100304>>.

15.2. Informative References

- [Aircrack-ng] d'Otreppe, T., "Aircrack-ng", Accessed: 11-Jul-2010, <<http://www.aircrack-ng.org/>>.
- [BeckTews09] Beck, M. and E. Tews, "Practical Attacks Against WEP and WPA", Second ACM Conference on Wireless Network Security Zurich, Switzerland, 2009, <<http://wirelesscenter.dk/Crypt/wifi-security-attacks/Practical%20Attacks%20Against%20WEP%20and%20WPA.pdf>>.
- [ForceHTTPS] Jackson, C. and A. Barth, "ForceHTTPS: Protecting High-Security Web Sites from Network Attacks", In Proceedings of the 17th International World Wide Web Conference (WWW2008) , 2008, <<https://crypto.stanford.edu/forcehttps/>>.
- [GoodDhamijaEtAl05] Good, N., Dhamija, R., Grossklags, J., Thaw, D., Aronowitz, S., Mulligan, D., and J. Konstan, "Stopping Spyware at the Gate: A User Study of Privacy, Notice and Spyware", In Proceedings of Symposium On Usable Privacy and Security (SOUPS) Pittsburgh, PA, USA, July 2005, <<http://people.ischool.berkeley.edu/~rachna/papers/>>.

spyware_study.pdf>.

[I-D.ietf-tls-ssl-version3]

Freier, A., Karlton, P., and P. Kocher, "The SSL Protocol Version 3.0", draft-ietf-tls-ssl-version3 (work in progress), November 1996, <<http://tools.ietf.org/html/draft-ietf-tls-ssl-version3-00>>.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC2396] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August 1998.

[SunshineEgelmanEtAl09]

Sunshine, J., Egelman, S., Almuhiemedi, H., Atri, N., and L. Cranor, "Crying Wolf: An Empirical Study of SSL Warning Effectiveness", In Proceedings of 18th USENIX Security Symposium Montreal, Canada, Augus 2009, <http://www.usenix.org/events/sec09/tech/full_papers/sunshine.pdf>.

[W3C.WD-wsc-ui-20100309]

Saldhana, A. and T. Roessler, "Web Security Context: User Interface Guidelines", World Wide Web Consortium LastCall WD-wsc-ui-20100309, March 2010, <<http://www.w3.org/TR/2010/WD-wsc-ui-20100309>>.

[WEBSEC] "WebSec -- HTTP Application Security Minus Authentication and Transport", <<https://www.ietf.org/mailman/listinfo/websec>>.

[owaspTLSGuide]

Coates, M., Wichers, d., Boberski, M., and T. Reguly, "Transport Layer Protection Cheat Sheet", Accessed: 11-Jul-2010, <http://www.owasp.org/index.php/Transport_Layer_Protection_Cheat_Sheet>.

Appendix A. Acknowledgments

The authors thank Michael Barrett, Sid Stamm, Maciej Stachowiak, Andy Steingrubl, Brandon Sterne, Daniel Veditz for their review and contributions.

Appendix B. Change Log

Changes are grouped by spec revision and listed in reverse chronological order.

B.1. For draft-ietf-websec-strict-transport-sec

Changes from -00 to -01:

1. Changed the "URI Loading" section to be "URI Loading and Port Mapping".
2. [HASMAT] reference changed to [WEBSEC].
3. Changed "server" -> "host" where applicable, notably when discussing "HSTS Hosts". Left as "server" when discussing e.g. "http server"s.
4. Fixed minor editorial nits.

Changes from draft-hodges-strict-transport-sec-02 to draft-ietf-websec-strict-transport-sec-00:

1. Altered spec metadata (e.g. filename, date) in order to submit as a WebSec working group Internet-Draft.

B.2. For draft-hodges-strict-transport-sec

Changes from -01 to -02:

1. updated abstract such that means for expressing HSTS Policy other than via HSTS header field is noted.
2. Changed spec title to "HTTP Strict Transport Security (HSTS)" from "Strict Transport Security". Updated use of "STS" acronym throughout spec to HSTS (except for when specifically discussing syntax of Strict-Transport-Security HTTP Response Header field), updated "Terminology" appropriately.
3. Updated the discussion of "Passive Network Attackers" to be more precise and offered references.
4. Removed para on normative/non-normative from "Conformance Criteria" pending polishing said section to IETF RFC norms.
5. Added examples subsection to "Syntax" section.

6. Added OWS to maxAge production in Strict-Transport-Security ABNF.
7. Cleaned up explanation in the "Note:" in the "HTTP-over-Secure-Transport Request Type" section, folded 3d para into "Note:", added conformance clauses to the latter.
8. Added explanatory "Note:" and reference to "HTTP Request Type" section. Added "XXX1" issue.
9. Added conformance clause to "URI Loading".
10. Moved "Notes for STS Server implementors:" from "UA Implementation Advice" to "HSTS Policy expiration time considerations:" in "Server Implementation Advice", and also noted another option.
11. Added cautionary "Note:" to "Ability to delete UA's cached HSTS Policy on a per HSTS Server basis".
12. Added some informative references.
13. Various minor editorial fixes.

Changes from -00 to -01:

1. Added reference to HASMAT mailing list and request that this spec be discussed there.

Authors' Addresses

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

Email: Jeff.Hodges@PayPal.com

Collin Jackson
Carnegie Mellon University

Email: collin.jackson@sv.cmu.edu

Adam Barth
Google, Inc.

Email: ietf@adambarth.com

URI: <http://www.adambarth.com/>

