UNIVERSITY OF TIANJIN

Host of Troubles Vulnerabilities

Author:
Qianyu Guo

Student Number:
1016216001

July 18, 2017

Contents

1 Overview

Host-of-Troubles is a class of new vulnerabilities that affect a wide range of HTTP implementations. The problem is that deployed systems are generally incorrect (non-compliant with RFC 7230) and inconsistent in parsing and interpreting Host headers in HTTP requests. This problem can be exploited by carefully crafting HTTP requests with ambiguous host information, inducing inconsistent interpretations between two parties. Such inconsistency can lead to severe security consequences, such as HTTP cache poisoning and security policy bypass.

2 Multiple Host Ambiguities

In parsing and interpreting the HTTP semantics, one of the most important designations is what host is involved with the request, because `Host` is the key protocol field for resource locating, request routing, caching, etc. The problem of multiple host ambiguities arises when two parties (the downstream and upstream) in an HTTP processing-chain parse and interpret host in a crafted, adversarial request differently. Inconsistency of host between two parties often causes disastrous consequ ences because of its semantic importance.

— *Host of Troubles: Multiple Host Ambiguilities in HTTP Implementations*

2.1 Multiple Host Headers

RFC 2616[1] states that a request with multiple same name headers is allowed only if the value of this header is defined as a single comma-separated list, which implies that a request with multiple `Host` headers is invalid. RFC 7230 [**?**] explicitly specifies that requests with multiple `Host` headers must be reject with 400 Bad Request.

— *Host of Troubles: Multiple Host Ambiguilities in HTTP Implementations*

RFC 2616

4.2 Multiple message-header fields with the same field-name MAY be present in a message if and only if the entire field-value for that header field is defined as a **comma-separated list**. It must be possible to combine the multiple header fields into one "`field-name:field-value`" pair, without changing the semantics of the message, by appending each subsequent field-value to the first, each separated by a comma. The order in which header fields with the same field-name are received is therefore significant to the interpretation of the combined field value, and thus a proxy MUST NOT change the order of these field values when a message is forwarded.

5.2 The exact resource identified by an Internet request is determined by **examining both the Request-`URI` and the `Host` header field**.

An origin server that does differentiate resources based on the host requested (sometimes referred to as virtual hosts or vanity host names) MUST use the following rules for determining the requested resource on an HTTP/1.1 request:

1. If Request-`URI` is an absolute`URI`, the host is part of the Request-`URI`. Any `Host` header field value in the request MUST be ignored.
2. If the Request-`URI` is not an absolute`URI`, and the request includes a `Host` header field, the host is determined by the `Host` header field value.
3. If the host as determined by rule 1 or 2 is not a valid host on the server, the response MUST be a 400 (Bad Request) error message.

Recipients of an HTTP/1.0 request that lacks a `Host` header field MAY attempt to use heuristics (e.g., examination of the URI path for something unique to a particular host) in order to determine what exact resource is being requested.

14.23 A client MUST include a `Host` header field in all HTTP/1.1 request messages . If the requested `URI` does not include an Internet host name for the service being requested, then the `Host` header field MUST be given with an empty value. An HTTP/1.1 proxy MUST ensure that any request message it forwards does contain an appropriate `Host` header field that identifies the service being requested by the proxy. All Internet-based HTTP/1.1 servers MUST respond with a 400 (Bad Request) status code to any HTTP/1.1 request message which lacks a `Host` header field.

19.6.1.1 It is extremely important that all implementations of HTTP (including updates to existing HTTP/1.0 applications) correctly implement these requirements:

- Both clients and servers MUST support the `Host` request-header.
- A client that sends an HTTP/1.1 request MUST send a `Host` header.
- Servers MUST report a 400 (Bad Request) error if an HTTP/1.1 request does not include a `Host` request-header.
- Servers MUST accept absolute `URIs`.

RFC 7230

5.4 A server MUST respond with a 400 (Bad Request) status code to any HTTP/1.1 request message that **lacks a Host header** field and to any request message that **contains more than one Host header** field or a Host header field with an **invalid field-value**.

2.2 Space-surrounded Host Header

2.2.1 The first header with preceding space

RFC 2616 does not have explicit text for this case. The syntax definition implies that systems should reject the request with a space preceding the first header. RFC 7230 suggests to either reject the request or ignore the header.

RFC 2616

|

RFC 7230

3. A sender MUST NOT send whitespace between the start-line and the first header field. **A recipient that receives whitespace between the start-line and the first header field MUST either reject the message** as invalid or consume each whitespace-preceded line without further processing of it (i.e., **ignore the entire line**, along with any subsequent lines preceded by whitespace, until a properly formed header field is received or the header section is terminated).

The presence of such whitespace in a request might be an attempt to trick a server into ignoring that field or processing the line after it as a new request, either of which might result in a security vulnerability if other implementations within the request chain interpret the same message differently.

3.2.4 The field value does not include any leading or trailin whitespace: **OWS occurring before the first non-whitespace octet of the field value** or after the last non-whitespace octet of the field value **ought to be excluded** by parsers when extracting the field value from a header field.

2.2.2 Non-first header with preceding space

RFC 2616 states that a such header needs to be processed as folded line of its previous header: remove its preceding line break characters to concatenate with the previous header. Although RFC 7230 already obsoletes line folding, it still allows a proxy or a server to process as line folding for backward compatibility considerations.

— *Host of Troubles: Multiple Host Ambiguilities in HTTP Implementations*

RFC 2616

2.2 HTTP/1.1 header field values can be folded onto multiple lines if the continuation line begins with a space or horizontal tab. All linear white space, including folding, has the same semantics as SP. A recipient MAY replace any linear white space with a single SP before interpreting the field value or forwarding the message downstream.

RFC 7230

3.2.4 Historically, HTTP header field values could be extended over multiple lines by preceding each extra line with at least one space or horizontal tab (obs-fold). **This specification deprecates such line folding except within the message/http media type.**

2.2.3 Headers with succeeding space

RFC 2616 does not have explicit text for this case. The syntax definition implies that systems should allow this request. The same situation is explicitly forbidden in RFC 7230.

RFC 2616

RFC 7230

3.2.4 A field value might be preceded and/or followed by optional whitespace (OWS). The field value does not include any leading or trailin whitespace: **OWS occurring** before the first non-whitespace octet of the field value or **after the last non-whitespace octet of the field value ought to be excluded** by parsers when extracting the field value from a header field.

Implementation/ Specification		Space-preceded Host as first header	Other space-preceded Host header	Space-succeeded Host header	schema of absolute-URI
Server	Apache	Not recognize	Line folding	Recognize	Recognize HTTP, not others
	IIS	Recognize	Line folding	Recognize	Recognize HTTP/S, reject others
	Lighttpd	Reject	Line folding	Recognize	Recognize HTTP/S, not others
	LiteSpeed	Reject	Line folding	Recognize	Recognize any schema
	Nginx	Not recognize	Not recognize	Not recognize	Recognize any schema
	Tomcat	Not recognize	Line folding	Not recognize	Recognize HTTP/S, reject others
Transparent Cache	ATS	Not recognize	Not recognize	Not recognize	Recognize any
	Squid	If no host before: recognize, else: not recognize	If no host before: recognize, else: not recognize	If no host before: reject, else: recognize	Recognize HTTP, reject others
Forward Proxy	Apache	Not recognize	Line folding	Recognize	Recognize HTTP, reject others
	IIS	Recognize	Line folding	Recognize	Recognize HTTP/S, reject others
	Squid	If no host before: recognize, else: not recognize	If no host before: recognize, else: not recognize	If no host before: reject, else: recognize	Recognize HTTP, reject others
Reverse Proxy	Apache	Not recognize	Line folding	Recognize	Recognize HTTP, not others
	IIS	Recognize	Line folding	Recognize	Recognize HTTP/S, reject others
	Lighttpd	Reject	Line folding	Recognize	Recognize HTTP/S, not others
	LiteSpeed	Reject	Line folding	Recognize	Recognize any schema
	Nginx	Not recognize	Not recognize	Not recognize	Recognize any schema
	Squid	If no host before: recognize, else: not recognize	If no host before: recognize, else: not recognize	If no host before: reject, else: recognize	Recognize HTTP, reject others
	Varnish	Reject	Line folding	Reject	Recognize HTTP, not others
CDN	Akamai	If no host before: recognize, else: not recognize	If no host before: recognize, else: not recognize	Reject	Recognize HTTP/S, reject others
	Alibaba	Not recognize	Not recognize	Not recognize	Recognize any schema
	Azure	Reject	Line folding	Recognize	Recognize HTTP/S, reject others
	CloudFlare	Not recognize	Not recognize	Not recognize	Recognize any schema
	CloudFront	Not recognize	Not recognize	Not recognize	Recognize any schema
	Fastly	Reject	Line folding	Reject	Not recognize any schema
	Level3	Not recognize	Not recognize	Reject	Recognize HTTP/S, reject others
	Tencent	Recognize	Recognize	Recognize	Recognize HTTP, reject others
Firewall	Bitdefender	Recognize	Recognize	Recognize	Likely fail-open
	ESET	Not recognize	Not recognize	Not recognize	Recognize any schema
	Huawei	Not recognize	Not recognize	Not recognize	Recognize any schema
	Kaspersky	Not recognize	Not recognize	Not recognize	Recognize any schema
	OS X	Not recognize	Not recognize	Not recognize	Not recognize any schema
	PAN	Not recognize	Not recognize	Not recognize	Recognize HTTP/S, not others
	Windows	Recognize	Recognize	Recognize	Recognize any
Specification	RFC 2616	Reject (implicit)	Line folding	Recognize	Not specified
	RFC 7230	Reject or not recognize	Reject or line folding	Reject	Not specified

Figure 1: Host parsing behaviours. Specifications and tested implementations ("recognize" means accepting as valid host field, "not recognize" means either ignoring or accepting as an unknown header field, "reject" means responding with 400 Bad Request).

2.3 Absolute-URI as Request-Target

Both RFC 2616 and RFC 7230 require server to accept absolute-`URI` as requset-target, and to prefer host component of absolute-`URI` than `Host` header. RFC 7230 additionally requires requests with absolute-URI to have identical host component as `Host` header. Both of the two RFCs do not explicitly state which schema is allowed in the absolute-URI.

RFC 2616

5.2 See the "RFC 2616" part in section 2.1.

RFC 7230

5.5 Since the request-target often contains only part of the user agents target URI, a server reconstructs the intended target as an "effective request URI" to properly service the request. This reconstruction involves both the **servers local configuration** and information communicated in the **request-target**, `Host` **header field**, and **connection context**.

For a user agent, the effective request URI is the target URI. If the request-target is in absolute-form, the effective request URI is the same as the request-target. Otherwise, the effective request URI is constructed as follows:

1. If the servers configuration (or outbound gateway) provides a fixed URI **scheme**, that scheme is used for the effective request URI. Otherwise, if the request is received over a TLS-secured TCP connection, the effective request URIs scheme is "https"; if not, the scheme is "http".
2. If the servers configuration (or outbound gateway) provides a fixed URI **authority component**, that authority is used for the effective request URI. If not, then if the request-target is in authority-form, the effective request URIs authority component is the same as request-target. If not, then if a `Host` header field is supplied with a non-empty field-value, the authority component is the same as the `Host` field-value. Otherwise, the authority component is assigned the default name configured for the server and, if the connections incoming TCP port number differs from the default port for the effective request URIs scheme, then a colon (":") and the incoming port number are appended to the authority component.
3. If the request-target is in authority-form or asterisk-form, the effective request URIs combined path and query component is empty. Otherwise, the combined path and query component is the same as the request-target.

The components of the effective request URI, once determined as above, can be combined into absolute-URI form by concatenating the **scheme**, "://", **authority**, and **combined path and query component**.

References

[1] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext transfer protocol–http/1.1, 1999. *RFC2616*, 2006.

[2] Roy Fielding and Julian Reschke. Hypertext transfer protocol (http/1.1): Authentication. 2014.