Network Working Group
Internet Draft
Intended status: Standards Track
Expires: October 27, 2010

Gonzalo Salgueiro
Cisco Systems
Paul E. Jones
Cisco Systems

April 27, 2010

Securing HTTP State Management Information
draft-salgueiro-secure-state-management-03.txt

Status of this Memo

Copyright Notice

Abstract

Virtually every application on the web today that allows a user to log in or manipulate information stored on a server maintains some form of state management information. Usually, the session context is established through the use of a Uniform Resource Locator (URL) parameter or a Hypertext Transfer Protocol (HTTP) cookie that identifies the session. Without the use of Transport Layer Security (TLS), such an information exchange introduces a security risk. For a variety of reasons, TLS may not be desired or preferred in all situations and, in those cases, users are left vulnerable. This memo provides a simple method for enabling secure exchange of state management information through HTTP in situations where TLS is not employed.

Table of Contents

1. Introduction.....2
2. Conventions used in this document.....4
3. The “Secure-Cookie” Header.....4
4. Security Associations.....4
4.1. Use of TLS to Create a Security Association.....5
4.2. Use of Diffie-Hellman to Create an Association and Key....5
5. Use of Symmetric Key Encryption Algorithms.....5
6. Establishing an Association.....6
6.1. Establishing a Security Association over TLS.....7
6.2. Establishing a Security Association using Diffie-Hellman..8
7. Encoding Large Integers.....10
8. Transmitting Secure Information from the Server.....10
9. Transmitting Secure Information from the User Agent.....11
10. Security Considerations.....12
11. IANA Considerations.....12
12. References.....13
12.1. Normative References.....13
12.2. Informative References.....13
13. Acknowledgments.....13

1. Introduction

Though we have HTTPS (HTTP over TLS) [2] for securing communication between HTTP [3] User Agents (i.e., web browsers) and web servers, there are many web applications and web sites that rely on insecure connections to exchange state management information in the form of HTTP URL parameters or cookies [4] that could allow rogue entities to gain access to protected resources. Even in environments where secure connections are used for initially authenticating users, the sessions established and associated with the User Agent often use a simple cookie exchange over an insecure connection for subsequent

information exchanges, thus securing only the user's password, but not the session itself. This allows HTTP sessions to be hijacked by any entity that can observe the state management information. This memo provides a simple method for enabling secure exchange of state management information through HTTP in situations where TLS [5] is not employed.

One could use HTTPS everywhere on the Internet, but there are reasons why that is not always desired or preferred:

1. In practice, the use of HTTPS requires a unique IP address per URL (i.e., https://www1.example.com and https://www2.example.com would have to have two different IP addresses, even if these are on the same physical machines). While Section 3.1 of RFC 4366 [6] does address this concern, widespread adoption is slow and does not address the other concerns listed below.

2. Using HTTPS consumes more processing time and resources, an issue that is only compounded when there are several small transactions over separate connections.

3. Using HTTPS on the Internet requires the purchase of digital certificates and, depending on one's environment, this can be costly. It is understood that private networks can use self-signed certificates, but that does not address the more general Internet use cases.

4. Installing and updating digital certificates takes time, thus increasing Total Cost of Ownership (TCO).

5. Expired certificates drive visitors away in fear due to security warnings presented by web browsers.

6. Encrypting the entire session is not needed in many instances, especially when communicating with web sites that only exchange publicly available information (e.g., bulletin boards and blogs). Even though encryption is not critical for some applications, most would agree that proper state management is nonetheless important.

7. Encrypting the entire session prevents routers or other devices from efficiently compressing otherwise highly compressible plain ASCII text over low bit-rate links.

For one or more of these stated reasons, many web applications exchange state management information that should be secured over insecure connections. Therefore, application developers need a method of providing an acceptable level of security for selected state management information that does not require the use of HTTPS.

2. Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [1].

3. The “Secure-Cookie” Header

This memo introduces a “Secure-Cookie” to HTTP that is used to exchange information securely between the User Agent (client) and HTTP server. When the client transmits information to the server or when the server transmits information to the client via a Secure-Cookie, the transmitted information inside the cookie is encrypted. This provides protection for the data, regardless of whether the Secure-Cookie is transmitted over HTTP or HTTPS.

To enable this functionality, the client and server must establish a security association. The means through which the security association is established and the mechanisms used to encrypt the Secure-Cookie contents are detailed in the subsequent sections.

It should be noted that [4] defines an attribute associated with the Cookie header that indicates that the cookie is to be transmitted only over TLS connections. This “secure” attribute is in no way related to the Secure-Cookie described in this memo.

4. Security Associations

In order to provide a means of exchanging information securely in a session, the client and server must establish one or more security association(s). The association defines the encryption algorithm and encryption key to be used when transmitting encrypted information within a Secure-Cookie header.

The security association is assigned an identifier by the server and is used in subsequent requests from the client. The format of that association identifier is discussed in Section 6.

In order to allow for multiple concurrent requests and to thwart the possibility of a replay attack, a client MAY establish multiple security associations with the server. For example, each tab on a web browser MAY establish its own client/server security association. A client MUST NOT issue concurrent requests that utilize the same security association identifier, as the server will not be able to differentiate between legitimate requests and requests that are, in fact, replay attacks.

Once an association has been established, it MAY be used subsequently over either HTTP or HTTPS when the client issues requests to the server.

4.1. Use of TLS to Create a Security Association

The server SHOULD use TLS as the means of establishing the security association. By using TLS, the encryption key is transmitted as plaintext over the encrypted TLS connection from the server to the client.

Once the security association is created via TLS, the client may be directed to use HTTP for subsequent requests. Secure-Cookie headers may then be used to securely encrypt session management information over HTTP.

4.2. Use of Diffie-Hellman to Create an Association and Key

HTTP servers MAY use a Diffie-Hellman (DH) key exchange [7] to establish a security association that will be used to encrypt sensitive state management information.

It is a well-known fact that use of Diffie-Hellman is subject to a Man-in-the-Middle (MiM) attack. While this security vulnerability exists, it is nonetheless better than the situation we have today where anyone can easily grab state management information and hijack a session.

In situations where transmitted information is sensitive or the risk of a MiM attack is significant, HTTPS SHOULD be used to establish security associations.

5. Use of Symmetric Key Encryption Algorithms

Sensitive state management information is encrypted using a symmetric key encryption algorithm provided by the server when establishing a security association.

The particular encryption algorithm, strength, and mode of operation are negotiated between the client and server. Specifically, the client MUST advertise supported encryption methods and the server will select which method to use for all encrypted information for a given security association.

The supported encryption methods MUST be advertised by the client in every request. They will appear in an HTTP header of the form

```
Secure-Cookie-Crypto: alg1,alg2,alg3
```

In this example, “alg1”, “alg2”, and “alg3” would represent three different encryption methods that specify the algorithms, strengths (i.e., key sizes), and/or modes.

The syntax for the Secure-Cookie-Crypto header follows the standard syntax for all HTTP headers as defined in Section 4.2 of RFC 2616 [3] with a comma-separated list of tokens that MAY include whitespace between tokens.

This memo specifies three standard encryption methods:

3des-cbc

Triple DES using three independent 56-bit encryption keys and cipher-block chaining mode.

aes-128-cbc

Advanced Encryption Standard using a 128-bit encryption key and cipher-block chaining mode.

aes-256-cbc

Advanced Encryption Standard using a 256-bit encryption key and cipher-block chaining mode.

All User Agents and servers MUST support aes-128-cbc and MAY support any number of additional algorithms and modes. Non-standard encryption methods MUST begin with “x-”. Any additional encryption standard methods MUST be published in an RFC and registered with IANA.

6. Establishing an Association

To issue a request that allows for the possibility of establishing a new security association, the User Agent sends a message to the server with a Secure-Cookie-Crypto header, such as the following:

```
GET / HTTP/1.1
Secure-Cookie-Crypto: aes-128-cbc, aes-256-cbc
```

In some instances, a request MAY be for information that does not require receiving state management information (e.g., company logos, JavaScript code, or other public content). In those instances, the web server will reply as it normally would without any encrypted information included and without requiring authentication.

In cases where the request or response requires the use of encrypted state management information, the web server MUST establish the

security association. A server MUST create a security association when one is desired and when the client does not advertise a recognized security association in its request.

6.1. Establishing a Security Association over TLS

When using TLS and establishing a new security association, the server MUST reply to requests that do not contain a security association with a 401 Unauthorized as shown below:

```
HTTP/1.1 200 OK
WWW-Authenticate: Secure-Cookie assoc=12345, crypto=aes-256-cbc,
                  key=yyyy
```

In the above, there are several parameters that are introduced that need discussion. They are:

assoc

This is an association handle assigned by the web server. This handle is comprised of ASCII characters constrained to those defined by the Base64 data encoding method (RFC 4648 [8]). The length of this handle MUST NOT exceed 64 octets.

crypto

This is the encryption algorithm and mode of operation selected by the server. The server MUST specify exactly one algorithm and mode of operation.

key

This parameter contains the Base64-encoded encryption key that will be used when encrypting Secure-Cookie headers. The number of octets that comprise the key MUST be equal to the number of octets expected for the selected encryption algorithm and mode of operation.

The reason for replying with a 401 rather than returning a 200 response to the request along with a security key is that the client may a have a Secure-Cookie that it wishes to transmit, but does not have a valid security association that it can utilize. The 401 response allows the server to reject the request and create a security association that may then be used subsequently in requests from the client.

Once the client has received this information, it MAY re-issue the request as in the following example:

```
GET / HTTP/1.1
Secure-Cookie-Crypto: aes-128-cbc, aes-256-cbc
Secure-Cookie-Assoc: assoc=12345
```

As shown in this example, the User Agent continues to advertise the supported cryptographic algorithms and modes. This is necessary in case the association expires between requests, prompting the server to return a 401 Unauthorized to facilitate the establishment of a new association. Note that the length of time that a server wishes to allow an association to remain valid is outside the scope of this memo.

Also included in the above request is the header Secure-Cookie-Assoc. It includes one parameter:

assoc

This is an association handle assigned by the web server and MUST be provided exactly as it was received. The client MUST NOT assume this handle is encoded in any particular way.

Note that if the client had previously established communication with the server before and had secure state management information to transmit, it MUST include that information in this request.

6.2. Establishing a Security Association using Diffie-Hellman

When using HTTP (as opposed to HTTPS) and establishing a new security association, the server MUST reply to requests that do not contain a security association with a 401 Unauthorized as shown below:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: Secure-Cookie assoc=12345, g=2, p=yyyy, A=xxxx,
                  crypto=aes-256-cbc
```

In the above, there are several parameters that facilitate the DH key exchange and establishment of an association. They are:

assoc

This is an association handle assigned by the web server. This handle is comprised of ASCII characters constrained to those defined by the Base64 data encoding method (RFC 4648 [8]). The length of this handle MUST NOT exceed 64 octets.

g

The value “g” is a primitive root mod “p” as defined by the DH key exchange algorithm. This parameter is OPTIONAL and, when absent, the value 0x02 MUST be assumed.

p

This is a large prime number that MUST be used by the client and server as a part of the DH key exchange algorithm. This parameters is OPTIONAL and, if absent, the value used MUST be 0xDCF93A0B883972EC0E19989AC5A2CE310E1D37717E8D9571BB7623731866E61EF75A2E27898B057F9891C2E27A639C3F29B60814581CD3B2CA3986D2683705577D45C2E7E52DC81C7A171876E5CEA74B1448BFDFAF18828EFD2519F14E45E3826634AF1949E5B535CC829A483B8A76223E5D490A257F05BDFF16F2FB22C583AB.

A

This is the result computed by the server A=g^a mod p, where “a” is a secret large integer not transmitted over the network.

crypto

This is the encryption algorithm and mode of operation selected by the server. The server MUST specify exactly one algorithm and mode of operation.

Once the client has received this information, it MUST complete the DH key exchange and association establishment by re-issuing the request as in the following example:

```
GET / HTTP/1.1
Secure-Cookie-Crypto: aes-128-cbc, aes-256-cbc
Secure-Cookie-Assoc: assoc=12345, B=zzzz
```

As shown in this example, the User Agent continues to advertise the supported cryptographic algorithms and modes. This is necessary in case the association expires between requests, prompting the server to return a 401 Unauthorized to facilitate the establishment of a new association. Note that the length of time that a server wishes to allow an association to remain valid is outside the scope of this memo.

Included in the above request is the header Secure-Cookie-Assoc, which completes the association. It includes two parameters:

assoc

This is an association handle assigned by the web server and MUST be provided exactly as it was received. The client MUST NOT assume this handle is encoded in any particular way.

B

This is the result computed by the client B=g^b mod p, where "b" is a secret large integer not transmitted over the network.

Note that if the client had previously established communication with the server before and had secure state management information to transmit, it MUST include that information in this request.

Subsequent requests from the client to the server need not include the "B" parameter as a part of the Secure-Cookie-Assoc header, since the association would have been fully formed.

Per the Diffie-Hellman algorithm, a shared secret is derived from the values created locally and received over the network from the peer. The shared secret, K, is an integer that MUST be consumed by both the client and server in the same way. Therefore, the value K MUST be converted into a string of octets in network byte order. The encryption key shall be the n least significant bits, where n is the number of bits required for the selected encryption algorithm and mode. If the integer is too small to yield enough bits for the encryption key, then the most significant bits of the encryption key MUST be zero-filled until the length of the key is n bits long.

7. Encoding Large Integers

Integers defined in this memo that are transmitted in messages (i.e., A, B, g, and p) MUST be represented in network byte order, zero-filling the most significant bits in order to fit the integer into an integral number of octets, then Base64-encoded.

Note that all integers are positive numbers and care should be taken to ensure that the most significant bit is not misinterpreted to be a sign bit.

8. Transmitting Secure Information from the Server

If the server wishes to provide the User Agent with secure state management information, it will do so in a reply once an association is established. Such a reply would look like this:

```
200 OK
Secure-Cookie-Assoc: assoc=12345
```

```
Set-Secure-Cookie: session=someEncryptedValue; path=/;
                   domain=.example.com
```

In this example, the server returns the association identifier "12345". It also returns a cookie whose syntax precisely aligns with draft-ietf-httpstate-cookie-08.txt [4]. The difference is that the cookie header is called Secure-Cookie and the header for setting the cookie is Set-Secure-Cookie. Within the Secure-Cookie and Set-Secure-Cookie, the cookie-value is encrypted using the algorithm and mode specified for the association. Note that the use of Set-Cookie and Set-Secure-Cookie is not mutually exclusive. Likewise, use of Cookie and Secure-Cookie is also not mutually exclusive. Set-Secure-Cookie and Secure-Cookie are only used to transmit encrypted state management information according to this memo.

State management information transmitted from the server to the User Agent MUST be encrypted and coded as follows:

```
encrypted_value = base64(IV || alg(value))
```

That is, the plaintext MUST be encrypted using the selected algorithm ("alg"). Since cipher-block chaining mode is specified for use in this memo, an initialization vector (IV) MUST be included. The IV is concatenated with the encrypted data and then base64-encoded.

9. Transmitting Secure Information from the User Agent

When issuing subsequent requests to the server and having what it believes to be a valid association identifier, the User Agent MUST include encrypted state management information in a Secure-Cookie header. The following example shows such a request:

```
GET / HTTP/1.1
Secure-Cookie-Crypto: aes-128-cbc, aes-256-cbc
Secure-Cookie-Nonce: 3
Secure-Cookie-Assoc: assoc=12345
Secure-Cookie: session=someEncryptedValue
```

This request includes the encrypted state management information. However, providing a block of encrypted state management information that might be the same from one request to the next creates the possibility for a replay attack. For this reason, the client MUST also include a nonce value. The nonce is a monotonically increasing integer in the range from 0 to 2^64 - 1. Once this integer reaches 2^64, a new association MUST be created.

The encrypted information transmitted from the User Agent to the server is encoded as follows:

```
encrypted_value = base64(IV || alg(value || nonce))
```

The means of encoding the encrypted information is virtually the same as that used by the server. The only difference is that the nonce is concatenated to the end of the plaintext prior to encryption. When concatenating the nonce to the plaintext, the User Agent MUST first convert the integer into an ASCII string that is not padded in any way. The Secure-Cookie-Nonce header MUST contain precisely the same ASCII character string concatenated with the plaintext.

A server that receives a request that includes a nonce value that is less than or equal to the same nonce value already received from the client for a given association MUST reject the request with a 401 Unauthorized response code. The server need not invalidate the association, however, since the apparently invalid request MAY be coming from a rogue entity.

10. Security Considerations

Some procedures defined in this memo rely on the Diffie-Hellman key exchange algorithm, which are subject to a Man-in-the-Middle attack. Users should be aware of this fact and utilize TLS to establish a security association as per Section 6.1 whenever one needs to guard against such attacks.

Another form of attack that is possible is one where an entity in the network is able to monitor traffic transmitted from the client to the server and initiate requests in an attempt to reach the server faster than the client. If the rogue endpoint is able to reach the server before the legitimate User Agent, then the request MAY be accepted. In the process, the rogue entity MAY modify some information in the request and access resources on the server that it should not have authorization to access. The risk in this form of attack is clearly not what information the rogue entity can retrieve, since all information is transmitted as plaintext, anyway. The risk is that a rogue entity might introduce information, such as a blog posting, that will appear to have been transmitted by the unsuspecting valid user. If such concerns exist, TLS should be employed.

The procedures defined in this memo are not a replacement for TLS and merely serve to strengthen the use of HTTP over insecure connections that wish to securely exchange state management information within the security constraints outlined herein.

11. IANA Considerations

TBD.

12. References

12.1. Normative References

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

[2] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

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

[4] Barth, A., “HTTP State Management Mechanism”, draft-ietf-httpstate-cookie-08, April 2010.

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

[6] Eastlake, D., “Transport Layer Security (TLS) Extensions: Extension Definitions”, draft-ietf-tls-rfc4366-bis-06.txt, October 2009.

[7] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, June 1999.

[8] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 4648, October 2006.

12.2. Informative References

13. Acknowledgments

This document was prepared using 2-Word-v2.0.template.dot.

Authors’ Addresses

Gonzalo Salgueiro
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709
USA

Phone: +1 919 392 3266
Email: gsalguei@cisco.com

Paul E. Jones
Cisco Systems, Inc.
7025 Kit Creek Rd.
Research Triangle Park, NC 27709

Phone: +1 919 476 2048
Email: paulej@packetizer.com