Network Working Group   Gonzalo Salgueiro
Internet Draft   Cisco Systems
Intended status: Standards Track   Paul E. Jones
Expires: August 20, 2012   Cisco Systems
                                                      February 20, 2012

Securing HTTP State Management Information
draft-salgueiro-secure-state-management-06.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

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 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.

In our previous draft, we proposed the use of "Secure Cookies". This was met with mixed reactions. Some supported the idea of

introducing a cookie that could be secured, but some rightfully argued that cookies themselves could be encrypted at the server and so there was no need to secure the cookie. Rather, we need to focus only on securing the session. Our previous draft still enabled a Man-In-The-Middle attack when using HTTP, even when security credentials were exchanged over a secure connection.

In this draft, we allow the client and server to establish one or more security associations over HTTP or, preferably, HTTPS. For the purpose of this memo, a security association is defined by use of a specific Message Authentication Code (MAC) function along with a shared secret.

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. Capability Advertisement

In every request from the user agent to the server, the user agent MUST advertise its support for the Secure State Management procedures defined in this document. This is necessary in order to establish the initial security association, but is also necessary in order to force a client to re-establish a security association that is no longer valid or no longer recognized by the server.

The capability advertisement comes in the form of a header that enumerates the Message Authentication Code (MAC) functions supported by the user agent. The syntax of the new header, like other headers introduced in this memo, follows the syntax of other headers in HTTP and is:

```
SSM-Functions = "SSM-Functions" ":" MAC-Function
                *["," *SP MAC-Function]

MAC-Function = "hmac-md5" | "hmac-sha-1" | "hmac-sha-256" |
                "hmac-sha-512" | 1*token
```

Note that the comparison of MAC functions names MUST be case insensitive. In this document, the MAC functions all utilize the HMAC [11] specification, though clients and servers MAY support other MAC functions.

Clients MAY support any number of MAC functions, but MUST support either HMAC with MD5 [10] ("hmac-md5") or HMAC with SHA-1 [9] ("hmac-sha-1"). Servers MUST support both hmac-md5 and hmac-sha-1 and SHOULD support a wide variety of popular MAC functions.

Using the above syntax, the following is an example header transmitted by a user agent:

```
SSM-Functions: hmac-md5, hmac-sha-1
```

Note that the server always selects the MAC function to employ from among those offered by the client.

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 MAC function and shared secret to be used when transmitting information between the client and server.

The security association is assigned a handle by the server and is used in subsequent requests from the client. The format of that association handle is discussed in Sections 5 and 6.

In order to allow for multiple concurrent requests, 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. Additionally, a client assigns a session handle for each concurrent session that exists within the scope of the security association. A client MUST NOT issue concurrent requests that utilize the same security association handle and session handle, as the server will not be able to differentiate between legitimate requests and requests that are, in fact, replay attacks. A client MAY issue concurrent requests that utilize the same security association handle and different session handles.

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. Establishing a Security 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 SSM-Functions header, such as the following:

```
GET / HTTP/1.1
SSM-Functions: hmac-md5, hmac-sha-1
```

In the following two sections, we discuss how a security association is established using HTTPS or HTTP (with Diffie-Hellman).

4.2. Establishing a Security Association over HTTPS

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

Once the security association is created via HTTPS, the client may be directed to use HTTP for subsequent requests. SSM-Parameters header may then be used to transmit requests over HTTP and be assured that the important parts of the request or response will not be manipulated.

When using HTTPS and establishing a new security association, the server MUST reply to requests that contain the SSM-Functions header and that do not demonstrate having a valid security association with a 401 Unauthorized as shown below:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: SSM assoc=12345, func=hmac-sha-1, secret=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 upper or lowercase letters and digits (ALPHA and/or DIGIT as defined in 2.2 of [3]). The length of this handle MUST NOT exceed 64 octets.

func

This is the MAC function selected by the server. The server MUST specify exactly one MAC function.

secret

This parameter contains the Base64-encoded shared secret in network byte order that will be used when computing the MAC transmitted from or to the server. The number of octets that comprise the secret MUST be equal to or greater than the number of octets produced by the MAC function or, if applicable, the underlying hash function, whichever is greater. However, the number of octets that comprise the secret should not be more than two times the number of octets produced by the selected function. (For example, HMAC-SHA-1 produces a 20-octet MAC. Therefore, the shared secret should

be between 20 and 40 octets, inclusive.) Note that the secret must be Base64-decoded prior to consumption by the MAC function.

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 wish to transmit state management information, but does not have a valid security association that it can utilize. The 401 response allows the server to reject the request and establish 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
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; session=1; nonce=1;
          components=Request-Line;
          mac=2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
```

As shown in this example, the User Agent continues to advertise the supported MAC functions. 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.

In cases where the client and server are communicating using HTTP and the server wishes to force the client to switch transports to HTTPS to transmit a shared secret, the server rejects the HTTP request as shown below:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: SSM transport=https, port=443
```

In the above, there is a single case-insensitive parameter called "transport" and an optional "port" parameter that defaults to port 443. The only value presently defined for transport is "https". When the client receives this response, it re-issues the request using HTTPS. This will result in a subsequent 401 similar to the first example in this section wherein the server provides the shared secret to the client. Once the client has the shared secret in hand, it then re-issues the request using HTTP (not HTTPS).

4.3. Establishing a Security Association using Diffie-Hellman

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 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. Further, a Man-in-the-Middle attack requires an active attacker, whereas session stealing is a much easier passive attack.

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

When using HTTP to establish 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: SSM assoc=12345, g=2, p=yyyy, A=xxxx,
                  func=hmac-sha-1
```

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 upper or lowercase letters and digits (ALPHA and/or DIGIT as defined in 2.2 of [3]). 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
0xDCF93A0B883972EC0E19989AC5A2CE310E1D37717E8D9571BB7623731866
E61EF75A2E27898B057F9891C2E27A639C3F29B60814581CD3B2CA3986D268

3705577D45C2E7E52DC81C7A171876E5CEA74B1448BFDFAF18828EFD2519F1
4E45E3826634AF1949E5B535CC829A483B8A76223E5D490A257F05BDFF16F2
FB22C583AB.

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.

func

This is the MAC function selected by the server. The server MUST specify exactly one MAC function.

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
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; nonce=1; components=Request-Line;
          B=zzzz; mac=2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
```

As shown in this example, the User Agent continues to advertise the supported MAC functions. This is necessary in case the association expires or otherwise becomes invalid 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 SSM-Parameters, which completes the association. It includes several parameters that are included in all requests from the client when exchanging secure state management information. We will cover the majority of the parameters in Section 5, but we will discuss the B parameter here since it applies only when initially establishing a security association using Diffie-Hellman:

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.

Subsequent requests from the client to the server need not include the "B" parameter as a part of the SSM-Parameters header, since the

association would have been fully formed and SHOULD be ignore by the server when received.

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 shared secret is the n least significant bits, where n is the number of bits equal to two times the number of bits generated by the selected MAC function or, if applicable, the underlying hash function, whichever is greater. If the integer is too small to yield enough bits, then the most significant bits of the shared secret MUST be zero-filled until the length is n bits long.

Integers defined in this section 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.

5. Transmitting Information from the User Agent

When issuing requests to the server and having what it believes to be a valid association handle, the user agent MUST include the SSM-Functions and SSM-Parameters headers in the request. The following example shows such a request:

```
GET / HTTP/1.1
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; session=1; nonce=1;
                components=Request-Line;
                mac=2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
```

There are several parameters included in the SSM-Parameters header as described below:

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.

session

This is an optional session handle created by the user agent to enable it to issue concurrent requests using the same security association. This handle is comprised of ASCII characters constrained to upper or lowercase letters and digits (ALPHA and/or DIGIT as defined in 2.2 of [3]). The length of this handle MUST NOT exceed 64 octets. If this handle is absent, the server MUST assume the session handle has the value NULL (i.e., zero-length string). Note that, while a client can generate any number of session handles, the web server is not required to track more than 128 handles per security association. {Editor's note: in order to allow browser windows and JavaScript code to issue requests using the same security association, perhaps a JavaScript function should be provided by the browser to assign a unique session identifier?}

nonce

The nonce is a monotonically increasing integer in the range from 0 to 2^64 - 1. To enable concurrent requests, each session identified by the session parameter has its own nonce space. It is presented and consumed by the MAC function in ASCII text form. Once this integer reaches 2^64, a new association MUST be created. The user agent selects the initial value for the nonce, which is RECOMMENDED to be a random value in the range of 0 to 2^32 -1.

components

This optional parameter contains a comma-separated list of message components that are included in the message over which a MAC is computed. Those components MAY be any one of these components defined in [3]:

Request-Line
Status-Line
message-header
message-body

If used, the message-body MUST be consumed by the MAC function without modification. All other components MUST be consumed by the MAC function as-is (including all whitespace and the colon that separates the header from its value), except that any CR or LF characters MUST NOT be consumed. Each of the components is consumed by the MAC function in the order in which they are presented in the components parameter.

Headers used to generate the MAC MAY appear more than once in the message. In such a case, all headers with the same name must be consumed in the order transmitted on the wire. It is ill-advised to include headers that are intended to be modified my intermediaries, such as the Via header, as doing so will likely result in errors computing the MAC.

mac

The mac parameter is a case-insensitive hex representation of the Message Authentication Code generated by the MAC function in use with this security association, presented in network byte order. The mac is computed as follows:

```
mac = message_authentication_function(secret,message);
```

where

```
message = (Request-Line ||
           Status-Line  ||
           message-header ||
           message-body ||
           assoc ||
           session ||
           nonce);
```

The value of secret is the octets obtained from decoding the Base64-encoded secret parameter in the WWW-Authenticate header (when using HTTPS) or the n least significant bits of K when using Diffie-Hellman as explained in Section 4.3.

It is permissible to indicate in the components that a non-existent header or a zero-length message body is used as a part of the "message". In that case, there is nothing to concatenate and there is no impact on the "message" over which the MAC is generated, but does add to the integrity of the request or response. For example, indicating that the message-body is a part of the "message" when a message-body does not exist prevents an intermediary from altering or fabricating the message-body.

The server is able to associate the client using the association handle. It is able to validate the request by computing the MAC following the same recipe and comparing the computed MAC value with that received from the client.

If the server is unable to verify the MAC, the server MUST return a 401 prompting the client to attempt to create a new association. However, the server MUST NOT invalidate the association handle, since the reason the MAC may have failed to compare properly is

because a rogue user agent is attempting to use a handle not assigned to it.

If the server receives a request from a client using a nonce value that is less than a nonce value already presented by a trusted user agent, then the server MUST return a 401 error. The server MUST NOT invalidate the association, since a rogue user agent may attempt to re-use a previously used nonce value.

6. Transmitting Information from the Server

When a client send a request message to the server as described in Section 5, the server MUST include in the response an SSM-Parameters header as shown in this example:

```
HTTP/1.1 200 OK
SSM-Parameters: assoc=12345; session=1; nonce=1;
                components=Status-Line,Set-Cookie,message-body;
                mac=3931ff3e9a70d77c6b677b95d9ab7c6aed80d610
```

The parameters are identical to those defined in Section 5. One important point to note is that the nonce value in the response MUST match the nonce value used in the request.

If the client receives a response from the server containing a MAC that it cannot validate, then it must treat the response as invalid. There are only three possible reasons why the MAC does not validate, which include a software logic error, modification of the message as it passed through the network, or data corruption (either on the wire or at the remote server). Assuming the latter, the client MAY re-issue the request, but repeated failure to validate the MAC would suggest messages are being altered.

7. Example Usage to Log into a Social Network Service

In this section, we will discuss a typical exchange where a user visits a social network service and logs in.

The initial request from the client is a typical request to get the main page of the site. At the outset, there are no security associations nor a need for one. A user agent might transmit the following request:

```
GET / HTTP/1.1
Host: social.example.com
SSM-Functions: hmac-md5, hmac-sha-1
```

In response, the server will return a web page that introduces the social site:

```
HTTP/1.1 200 OK
Content-Type: text/html; charset=UTF-8
```

Included in the response would be the message body containing HTML with various link, including a link to a "login" page. Note that, up to this point, no security association has been established with the server.

The user then clicks on the button to log into the service. This link directs the user agent to a login page served over an HTTPS connection. The initial user agent request might look like this:

```
GET /login/ HTTP/1.1
Host: social.example.com
SSM-Functions: hmac-md5, hmac-sha-1
```

At this point, the server returns a response to form the security association:

```
HTTP/1.1 401 Unauthorized
WWW-Authenticate: SSM assoc=12345, func=hmac-sha-1,
                  secret=Y3VwaWQ=
```

The user agent then re-issues the request to the server, but this time including the information to demonstrate that the security association has been formed:

```
GET /login/ HTTP/1.1
Host: social.example.com
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; nonce=1; components=Request-Line;
                mac=f1784693e4bdefa9b5a1a0348fdc0791c307ed9a
```

Note that since a "session" parameter was not provided, the server assumes the value of "session" is NULL.

The server can then validate the MAC to ensure that the client has formed the association. The server will then respond to the request with a new HTML page that prompts the user for a login and password, like this:

```
HTTP/1.1 200 OK
SSM-Parameters: assoc=12345; nonce=1; components=Status-Line;
                mac=2b5cb730dac7e93e3c991918c503c8e87bd7cc82
Content-Type: text/html; charset=UTF-8
```

The user enters his username and password and click a button on the browser that results in a POST to the web server, like this:

```
POST /login/process/ HTTP/1.1
Host: social.example.com
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; nonce=2;
                components=Request-Line,message-body;
                mac=d632e1b7bc895fc2ce7752bade188b85f5d1c93a
Content-Type: application/x-www-form-urlencoded
Content-Length: 32

user=someuserid+password=abcd123
```

Upon receiving this request and successfully validating the MAC and authenticating the user, the web server might then redirect the user agent to an HTTP-accessible page (versus HTTPS) where the user can then interact with the social network service. This redirection might look like this:

```
HTTP/1.1 302 Found
Location: http://social.example.com/home/
SSM-Parameters: assoc=12345; nonce=2; components=Status-Line;
                mac=27283a874b10b9d86b50d3fa7426dd275afaeb02
Content-Length: 0
```

Note that the 302, while not a final response to the original HTTP request, is considered as such for the purposes of this memo. The next request to the same host, security association, and session MUST use a different nonce in order to avoid a replay attack.

Since the host did not change, the user agent may assume that the security association is still valid. It then issues the following request:

```
GET /home/ HTTP/1.1
Host: social.example.com
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; nonce=3; components=Request-Line;
                mac=4e51022cb7c25cc1706056d85f34a095e4a6e4e5
```

Knowing that user "someuserid" logged in and was associated on the server with the association handle "12345" and validating the MAC, the server may then serve the content that it should provide to that user. It does so with a normal 200 response that includes the HTML or other content.

While the user is interacting with the server, additional tabs or background threads may be launched that perform parallel requests to the server. Each of these separate windows or threads must use a different and unique "session" attribute. The following request,

for example, might be issued by a background thread that polls a user's message inbox:

```
GET /inbox/ HTTP/1.1
Host: social.example.com
SSM-Functions: hmac-md5, hmac-sha-1
SSM-Parameters: assoc=12345; session=ajax-thread-6; nonce=346353;
                components=Request-Line;
                mac=fbeb80b87dd8f03c418d44e4129006dca6a42dd7
```

8. 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 HTTPS to establish a security association as per Section 4.2 whenever one needs to guard against such attacks.

Note that traditionally, HTTP cookies are used to associate a user with a user agent. The procedures defined in this memo allow the server to identify the user via an association handle. If HTTP cookies are used in conjunction with the Secure State Management procedure defined herein, then the server should verify that the cookie(s) used to identify a user map to the same user identified by the association handle.

The procedures defined in this memo are not a replacement for HTTPS and merely serve to strengthen the use of HTTP over insecure connections that wish to provide for exchange of secure state management information.

9. IANA Considerations

TBD.

10. References

10.1. 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-22, February 2011.

[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", RFC 6066, January 2011.

[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.

[9] Eastlake, D., Jones, P., "US Secure Hash Algorithm 1 (SHA1)", RFC 3174, September 2001.

[10] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[11] Krawczyk, H., Bellare, M., Canetti, R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997.

10.2. Informative References

None.

11. 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