ip.buffer App Note
AN004 : SSL/TLS Security

Date	*Author*	*Release*
2008-03-19	MP	Initial draft

Scannex Electronics Ltd, UK
t: +44(0)8707 48 65 65
f: +44(0)8707 48 67 67

http://www.scannex.co.uk
info@scannex.co.uk

Scannex LLC, USA
t: 1-866-4BUFFER
(1-866-428-3337)

http://www.scannex.com
info@scannex.com

Table of Contents

1. Introduction

Many of the TCP/IP protocols used in the Internet, and across LANs, are open to eavesdroppers – the data is transferred in "plain text" meaning that it can be sniffed and examined by a third party.

Since the Internet often requires the transfer of sensitive information there are standard protocols that provide authentication and encryption services. When using these additional protocols the data is hidden from eavesdroppers and cannot easily be examined by a third party.

The ip.buffer has a version of firmware that provides Internet standard encryption technology to provide a secure method of transferring data and administering the device[1].

[1] Many countries have specific laws in place regarding the import, use, and export of encryption technologies. Some ban their use completely. For this reason, the ip.buffers are shipped with **non-**encryption firmware. The encryption-enabled firmware can be downloaded from the Scannex website and the ip.buffer can be easily upgraded.

2. SSL/TLS

The most common Internet encryption technology available is SSL – Secure Sockets Layer. In actual fact, this was the original name for the protocol when introduced by Netscape. However, the protocol underwent some changes and became an Internet standard. At this point it became TLS – Transport Layer Security.

The ip.buffer supports SSL version 3, and TLS version 1[2].

The SSL/TLS protocols are used everyday in millions of Internet transactions, including web-based shopping carts and online banking. Usually a secure web address will begin with the prefix "https://" rather than "http://" The "https://" prefix will make the web-browser use SSL/TLS to the web server, and therefore encrypt everything that is sent – including any sensitive information.

The same SSL/TLS encryption can also be used with email and FTP communications.

2.1. Authentication in SSL/TLS

One of the purposes of SSL/TLS is to provide a concrete way of identifying the server you are communicating with.

In the past, a simple password was sufficient to show that the client was a genuine client. However, the client had no way of knowing if it was *really* talking to the server. There may have been someone intercepting the communication and lifting the password from the conversation – so that the password could be used at another time. This is traditionally known as "the man in the middle attack" - where an untrusted third party sits between the client and server and examines and relays the messages.

To overcome this blatant vulnerability, the SSL/TLS protocol includes some very clever mechanisms that allow a client to uniquely identify the server. The man-in-the-middle would have no way to create the same information as the original server so the client would be immediately aware of this kind of attack.

When a client talks to a server with SSL/TLS there will be a key-exchange process where the two devices check each other out by exchanging certificates. It is also possible for the client to double check the identity of the server by using the PKI – Public Key Infrastructure – which allows a trusted third party to verify the authenticity of a server certificate.

For example, a company such as Verisign or Thawte can provide a trusted way of checking that the computer you are talking to for online banking is really owned by the bank.

2.2. Encryption in SSL/TLS

After exchanging certificates the two computers will negotiate a cipher – that is a particular mathematical algorithm that encrypts the information – and then start communicating in a secure manner.

[2] There is a special case where an SSL version 2 "hello" message is sent and SSL version 3 is used. This is sometimes referred to as SSL version 2.5. This special case is supported by the ip.buffer

There are many different ciphers supported under the full SSL/TLS specification. Some are very weak – in fact some do not encrypt the data at all. Some are extremely strong (meaning that it is exceptionally difficult to guess how the information was hidden and recover the original text).

2.3. SSL/TLS on the computer

There are many SSL/TLS libraries available for the computer. The most common is an open source product called "Open SSL" - http://www.openssl.org/

Many applications come with SSL/TLS already bundled in them. For example, all web-browsers come with SSL/TLS included – because that is a fundamental part of the Internet these days. Other applications (like email servers, FTP servers, FTP clients) may, or may not, include SSL/TLS support[3].

[3] Most applications can be made to use SSL/TLS by using freely available redirection products like www.stunnel.org

3. SSL in the ip.buffer

The ip.buffer provides SSL/TLS by using an open-source library from http://xyssl.org

3.1. Authentication in the ip.buffer

Currently, the ip.buffer will use a single certificate. This certificate can be generated directly in the unit (by using one of the administration web-pages), or can be uploaded into the ip.buffer (if the installation requires a certificate generated by someone else).

Any access to the web-server using the https protocol will pass the certificate back to the web-browser. The client can then determine if they are actually communicating with the ip.buffer in question.

In addition, any other server access (e.g. FTP Server, TCP Server) that uses SSL/TLS will also pass the certificate back to the client for verification. It is entirely up to the client (or the software that is being used to connect to the ip.buffer) to decide how to verify the certificate, and what to do if the certificate is not what is expected.

At the moment, when the ip.buffer makes an SSL/TLS connection to another server (i.e. a push operation), the certificate is ignored – the ip.buffer assumes the remote device is the real one[4].

3.1.1. Advanced Certificate Use

It is possible to use the certificate in the ip.buffer to maintain a chain-of-supply. For example, a dealer could create their own “vendor certificate” and load this certificate and key into the ip.buffer before shipping to their customers.

When user generates a device key, using the ip.buffer web-page, it will base it on the “vendor certificate”. As a result it becomes possible to prove that the ip.buffer was shipped by the dealer – because the certificate's chain-of-trust will prove it. Any buffer purchased from another source would not have the same vendor certificate.

Of course, this requires that the vendor certificate and key is guarded from misuse! (The same way you would guard your front door key!)

3.2. Encryption in the ip.buffer

Scannex have forced the XySSL library to use one of two ciphers:

1. RC4, 128bit
2. AES, 128bit

Both of these ciphers are considered “strong” - certainly strong enough to protect sensitive information. The RC4 algorithm is provided as a fall-back for compatibility with older implementations, like Internet Explorer 6 and below, while AES is the modern standard supported by all current web-browsers and most other software. AES will be chosen in preference to RC4 if the other computer supports it.

[4] This method could change in a future firmware upgrade.

The "bit length" gives an indication as to the strength of the encryption. Ten years ago, 40-bit RC4 was used for encrypting Internet data. However, information encrypted with 40-bit RC4 can be recovered using modern computers. Hence, the minimum acceptable cipher strength is 128-bit. Although the ip.buffer could support AES, 256bit, it has been disabled for export/import reasons.

3.3. Modes Using SSL/TLS

3.3.1. Web Server

The web-server provides secure access via the https protocol (using the standard port 443). Any modern web-browser should be able to view the ip.buffer status page with either the http or https protocol.

As a precaution, any access to an encryption-enabled ip.buffer on the normal (unencrypted) http channel will show "**Warning! Click for Secure Connection**" in red at the top of each web-page. *Before* proceeding onto the administration pages, the user should click on the red banner to link to the secure https connection.

When the words "**Secure Connection**" appear at the top of the web-page the user can comfortably use the administration pages without worrying that someone else can sniff the configuration or diagnostic information from the conversation – all communication will be securely encrypted.

3.3.2. FTP Server

There are three connection types to the FTP server:

1. Normal – unencrypted (FTP)
2. FTPS[5] Explicit – starts unencrypted then negotiates to encrypted
3. FTPS Implicit – starts encrypted

Option 1 is supported by all FTP clients.

Option 2 and 3 are supported by most modern FTP clients. The advantage of option 2 is that you can start using an unencrypted channel on the standard port 21 and then negotiate to use a secure channel[6].

There are many FTP clients that support SSL/TLS. For example, the excellent FileZilla application (www.filezilla-project.org) supports SSL/TLS. There is even a Windows command-line replacement for the embedded "FTP" command line that supports SSL/TLS and the passive FTP transfer mode - "MOVEit Freely" from Standard Networks - http://www.standardnetworks.com/products/?category_number=6&subcategory_number=1[7]

[5] FTPS = FTP with SSL

[6] Of course, you can use a different port number to hide the server!

[7] A command line for an explicit transfer is: "`ftps -a -natpasv -z -e:on` *`ipbufferaddress`*". -a will default to the PASV mode, -z will ignore certificate information, -e:on will force SSL/TLS after initial connection. Scannex can provide additional information if needed.

3.3.3. TCP Server

It is only possible to use an implicit SSL connection. In this mode the ip.buffer will never allow a non-SSL connection to connect. Once connected and encryption has been negotiated the password and data between the client and the ip.buffer is encrypted.

Nothing is sent "in the clear".

3.3.4. FTP Push

Like the corresponding FTP Server, the ip.buffer can communicate to a central FTP Server and push the data in the same three ways:

1. Normal – unencrypted
2. Explicit – starts unencrypted and then negotiates to encrypted
3. Implicit – starts encrypted

Some FTP servers can support all three, others just the first two. For example, the Microsoft FTP Server shipped in Windows XP Pro can support both unencrypted and explicit links. The FileZilla Server package can support all three – it can even specify that a user must use SSL/TLS when logging in.

If the ip.buffer is programmed to perform FTP Push with either Explicit or Implicit SSL/TLS then if the server fails to negotiate encryption the transfer will fail. No data will ever be "sent in the clear".

At the moment the ip.buffer will ignore the certificate issued by the FTP server.

3.3.5. SMTP Push (Email)

The rules for Email Push are the same as for FTP Push. The email server should be configured to allow either explicit and/or implicit connections. Again, no data is ever sent in the clear if the email server cannot support the encryption level specified by the ip.buffer.

At the moment the ip.buffer will ignore the certificate issued by the email server.

3.3.6. TCP Push

Like TCP Server, the only encrypted method is "implicit". If the TCP Push cannot negotiate SSL/TLS with the remote computer then nothing is transferred.

4. SSL versus SSH

SSH was designed as an extension to the “SH”, or shell, service. It is intended to provide remote-control administration of a computer system (quiet often Unix/Linux).

The original Telnet interface on port 23 was subject to sniffing attacks. This is extremely dangerous as any hacker will discover the administrative password for the server system! Port 22 was used as an encrypted version of port 23 – again to control the server.

Various extensions were added to the SSH protocol to allow operations such as the transfer of files. This is obviously a useful feature when administering a computer. This transfer became known as SFTP. Although it shares the same “FTP” letters as the Internet standard “FTP” there is little resemblance in the actual protocol. They may perform the same activity to an end-user but the underlying protocol is different.

Although it is possible to use the SSH service to perform pushes of data using SFTP, this is not necessarily a good thing. The following points are worth noting:

1. There are more FTPS servers available (FTP servers with SSL) than SSH – especially on the Windows operating system.
2. Allowing access to port 22 on a central computer, especially across the Internet, is a brave move – it is likely that others will notice and attempt to hack in to gain control of the central computer.
3. A particular SSH implementation may, or may not, use certificates (and the PKI). A simpler form of SSH allowed for a simple password. In fact, RSA key exchange (which is standard on SSL/TLS) was left out of many SSH implementations because of patent issues (which have now been removed). Although some say PKI and X.509 certificates are difficult to manage, it is a simple matter to create self-signed certificates for use on a small network – a process that is free and does not require purchasing a commercial certificate[8].
4. SSH does use a single out-bound socket for communication and can tunnel multiple conversations down the one socket. However, this can be difficult to administer[9].

Scannex chose SSL/TLS as the main encryption protocol as it forms the basis for encrypted connections with the existing Internet standards for web, email, and FTP. Setting up SSH tunnelling for another service can be complicated[10].

Any new Internet standard methods of communication will most likely use SSL/TLS rather than SSH.

[8] Scannex are able to provide links and documentation on how to do this.

[9] Although the ip.buffer will use two sockets for FTP Push, it can perform an Email Push using a single outbound encrypted socket – which is typically used in managed service environments. Also note that it does not have to use port 25 for the SMTP protocol.

[10] As an example, http://kb.adobe.com/selfservice/viewContent.do?externalId=tn_16126&sliceId=2 illustrates using PuTTY to provide tunnelling on Windows.

5. References

SSL/TLS definition: http://en.wikipedia.org/wiki/Secure_Sockets_Layer

SSH definition: http://en.wikipedia.org/wiki/Ssh

SSH file transfer protocol:
http://en.wikipedia.org/wiki/SSH_file_transfer_protocol

What's the difference between SSH and SSL/TLS?
http://www.snailbook.com/faq/ssl.auto.html

XySSL source code library: http://xyssl.org/

OpenSSL source code library: http://www.openssl.org/

stunnel SSL/TLS redirection tool: http://www.stunnel.org/

FTP client: http://filezilla-project.org/

FTP server: http://filezilla-project.org/download.php?type=server

MOVEit Freely: http://www.standardnetworks.com/products/?category_number=6&subcategory_number=1

Enabling SSL in IIS on Windows XP Professional:
http://www.somacon.com/p41.php