GIAC
CERTIFICATIONS

Global Information Assurance Certification Paper

Copyright SANS Institute
Author Retains Full Rights

This paper is taken from the GIAC directory of certified professionals. Reposting is not permited without express written permission.

Interested in learning more?

Check out the list of upcoming events offering
"Security Essentials: Network, Endpoint, and Cloud (Security 401)"
at http://www.giac.org/registration/gsec

Open Source Implementations of SSL: Why and How

Jeff Dickens

May 3, 2001

Introduction

SSL, (Secure Sockets Layer), is a protocol intended to provide privacy and reliability between two communicating applications. Privacy means that the messages are not subject to interception by eavesdroppers. Reliability means any alteration of messages will be detected.

SSL was introduced by Netscape in 1995 and has since become a defacto standard on the World Wide Web. It is commonly used to provide secure access to Internet Banking, Brokerages, and E-Commerce. The subject of SSL on the World Wide Web was explored in Alex Bravo's September 2000 paper: Secure Servers with SSL in the World Wide Web, which can be found at http://www.sans.org/infosecFAQ/covertchannels/SSL.htm.

Open Source operating systems (Linux and BSD variants) and Open Source Web Servers (primarily Apache) are popular for web sites for several reasons. One is the cost. The software can be had for the cost of the media and manuals. Another reason is that Open Source software is thought to be less likely to contain hidden security vulnerabilities because the source code is open to public scrutiny. Linux and Apache are also now so popular that expert support, both paid and free, is readily available

Linux/Apache sites that want to use SSL may purchase a commercial version of Apache with SSL Support such as Covalent's Raven (http://www.covalent.net/products/ssl/) or C2net's Stronghold, which is now owned by Red Hat (http://www.c2.net/products).

Alternatively, there are Open Source implementations of Apache with SSL support available for Linux. This is the option this paper will further explore.

Why

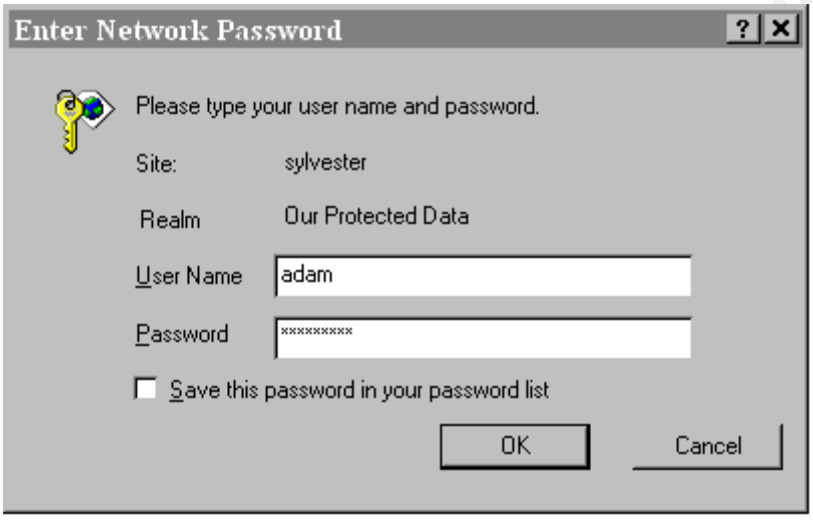
The security-savvy user is aware of when sensitive data is traveling over the network in the clear, and takes precautions to avoid this. Internet Explorer's "padlock" symbol or Netscape's "unbroken key" at the bottom of the browser indicates that SSL (Secure Sockets Layer) is in use. SSL protects the transmission of form data and passwords from the browser to the server, and the transmission of the password protected data from the server back to the browser.

Lacking this, passwords can be "sniffed" by anyone with access to any network data may travel over in between the endpoints, and the password protected data can be compromised as well as it travels back. The following example will demonstrate how this is done.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Example 1

Sylvester is a Web Server running Red Hat Linux 6.1 and Apache 1.3.9. It serves via http a directory that is protected by Basic Authentication: http://sylvester/protected. A password prompt appears when I try to access that page:



And, after supplying a valid User Name/Password pair, I am granted access:

I used the Network Instruments Observer 6.0 sniffer software to extract, with ease, the following data from the conversation between my browser and the server. The tcp connection setup and teardown, as well as the rest of the information from the ip and tcp layers have been edited out for simplicity.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

First, an HTTP GET from the browser:

```
IP,    browser     -> sylvester
HTTP Section: 254 bytes
    Request Line: GET /protected/ HTTP/1.1
    Header: Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, */*
    Header: Accept-Language: en-us
    Header: Accept-Encoding: gzip, deflate
    Header: User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
4.0)
    Header: Host: sylvester
    Header: Connection: Keep-Alive
```

Now a "401 Authorization Required" from Sylvester, specifying the type of authentication and the name of the authentication Realm.

```
IP,    sylvester     -> browser
HTTP Section: 693 bytes
    Status Line: HTTP/1.1 401 Authorization Required
    Header: Date: Wed, 04 Apr 2001 11:05:11 GMT
    Header: Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
    Header: WWW-Authenticate: Basic realm="Our Protected Data"
    Header: Keep-Alive: timeout=15, max=100
    Header: Connection: Keep-Alive
    Header: Transfer-Encoding: chunked
    Header: Content-Type: text/html
```

Now another GET from the browser , this time specifying an "Authorization" header containing some encrypted-looking stuff.

```
IP,    browser     -> sylvester
HTTP Section: 297 bytes
    Request Line: GET /protected/ HTTP/1.1
    Header: Accept: image/gif, image/x-xbitmap, image/jpeg,
image/pjpeg, */*
    Header: Accept-Language: en-us
    Header: Accept-Encoding: gzip, deflate
    Header: User-Agent: Mozilla/4.0 (compatible; MSIE 5.5; Windows NT
4.0)
    Header: Host: sylvester
    Header: Connection: Keep-Alive
    Header: Authorization: Basic YWRhbTpldkU0c25ha2U=
```

Now Sylvester sends the file the browser requested:

```
IP,    sylvester     -> browser
HTTP Section: 499 bytes
    Status Line: HTTP/1.1 200 OK
    Header: Date: Wed, 04 Apr 2001 11:05:19 GMT
    Header: Server: Apache/1.3.9 (Unix)  (Red Hat/Linux)
    Header: Last-Modified: Wed, 04 Apr 2001 10:10:09 GMT
    Header: ETag: "17dc3-c8-3acaf301"
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
    Header: Accept-Ranges: bytes
    Header: Content-Length: 200
    Header: Keep-Alive: timeout=15, max=98
    Header: Connection: Keep-Alive
    Header: Content-Type: text/html

HTTP Data
<head>.<title>Th
is is the protec
ted data</title>
.</head>..<body>
.<h1>Protected</
h1>.<h2>This dat
a is protected b
y Basic Authenti
cation..<p>.You
can only read it
if you have the
password.</h2>.
</body>.
```

Granted, I was able to contrive the circumstances under which I sniffed the data going by on the wire. But this should serve as an example of what could be done.

The "Protected" data was revealed to the sniffer. I guess it wasn't very well protected.

But beyond the loss of the data's privacy, what about that password I sent? The Authorization header looks encrypted:

```
Header: Authorization: Basic YWRhbTpldkU0c25ha2U=
```

But it's not. It's only Base64 encoded, and is protected from only the most casual observer. The HTTP Basic Authentication Scheme is documented at http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA for HTTP 1.0.

There is an HTTP Basic Authentication Decoder and Encoder at http://www.securitystats.com/tools/base64.asp. With it you can decode (or encode) any basic authentication string. It reveals that the Basic Authentication string above contains "`adam:evE4snake`".

There are other weaknesses in the Basic Authentication Scheme in general. One is that there is no password lockout mechanism, making servers that implement Basic Authentication vulnerable to iterative password cracking.

Many sites simply use HTML's forms capability to prompt for a username and password, knowing that their transmission will be encrypted by SSL.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Open Source Implementations of SSL and Apache

There are two Open Source implementations of SSL for Linux, SSLeay and OpenSSL. There are also two Open Source versions of Apache with SSL Support, Apache-SSL and Mod-SSL.

SSLeay was developed by Eric A. Young (thus the name) and Tim J. Hudson. It is available at http://www2.psy.uq.edu.au/~ftp/Crypto/.

OpenSSL is based on the work done in SSLeay. OpenSSL is available at http://www.openssl.org. OpenSSL also implements the Transport Layer Security (TLS) protocol V1.0, which is a proposed standard intended to replace SSL. The RFC for TLS is at http://www.ietf.org/rfc/rfc2246.txt and the IETF working group charter for TLS is at http://www.ietf.org/html.charters/tls-charter.html. I won't be covering TLS in this paper, but it's worth reading about.

Apache-SSL is a set of patches for Apache that add SSL Support. Apache-SSL is available at http://www.apache-ssl.org.

Mod-SSL is an Apache module that was based originally on a version of Apache-SSL. Mod-SSL is available at http://www.modssl.org.

OpenSSL and Mod-SSL are the more current work, although Apache-SSL is still being actively developed.

Note: If you decide to use any of this software for your work, you should download the source, verify the checksums where available, and compile it yourself. If you use binary RPMs like I did for this example, you are trusting the person who compiled the code not to have inserted any malicious code.

Installation

I will use SSLeay and Apache-SSL for my demonstration for the sake of expediency. There were ready-to install RPMs for these available that did not require any additional prerequisites on my test system.

Installing them is as easy as:

```
[root@sylvester rpms]# rpm -i SSLeay-0.9.0b-4.i386.rpm
[root@sylvester rpms]# rpm -i apache-ssl-1_3_4-1_31-1_i386.rpm
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
@ Don't forget to do a gendummycerts after installation @
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Certificates and Configuration

The subject of Digital Certificates is covered in David E. Savage's January 2001 paper: A Digital Certificate Introduction, which is available at http://www.sans.org/infosecFAQ/encryption/certicate.htm, and also in Stephen N. Williams's February 2001 paper: Digital Certificates: A Secure Method for Digital Transfers, which is at http://www.sans.org/infosecFAQ/encryption/digicert.htm.

A Certificate contains the server's Public Key, and is digitally signed by a trusted Certificate Authority. The signed Certificate authenticates the server's identity. The Public Key is used to securely generate shared secrets and the Session Key. The Session key is then used for the symmetric encryption protecting the transfer of data for the rest of the session. This process is called the SSL Handshake. See Iplanet's Introduction to SSL at http://www.iplanet.com/developer/docs/articles/security/ssl.html for a good concise description of the SSL Handshake.

In our example, the server's Certificate will not be signed by a trusted Certificate Authority. Rather, we will "self-sign" it, effectively implying only that we trust ourselves. This will suffice for the demonstration.

Apache-SSL provides a "gendummycerts" script to generate a "dummy" key, self-signed certificate and certificate request. The script contains the following commands:

```
ssleay req -new -out cert.csr -keyout certprivkey.pem
ssleay rsa -in certprivkey.pem -out cert.key
ssleay x509 -in cert.csr -out cert.cert -req -signkey cert.key -days 365
```

The first command generates a new RSA Private Key and an X.509 Certificate Signing Request.

The second command removes the passphrase from the private key. This is needed for the signing command below, but also comes into play when the Apache-SSL server needs to be started unattended. If the private key is protected by a passphrase, there must be a human present to provide it whenever the server is restarted

The third command signs the Certificate Signing Request (CSR) with the unprotected Private Key, creating the Certificate in the file cert.cert. If this were not a demo, this step would be omitted. Rather, the CSR would be transmitted to the Certificate Authority for signing, which would then reply with the signed certificate.

```
[root@sylvester rpms]# gendummycerts
Generating dummy certificates
Using configuration from /etc/ssleay.cnf
unable to load 'random state'
What this means is that the random number generator has not been seeded
with much random data.
Consider setting the RANDFILE environment variable to point at a file that
'random' data can be kept in.
Generating a 1024 bit RSA private key
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

```
.............+++++
.................+++++
writing new private key to 'certprivkey.pem'
Enter PEM pass phrase:
Verifying password - Enter PEM pass phrase:
-----
You are about to be asked to enter information that will be incorporated into
your certificate request.
What you are about to enter is what is called a Distinguished Name or a DN.
There are quite a few fields but you can leave some blank
For some fields there will be a default value,
If you enter '.', the field will be left blank.
-----
Country Name (2 letter code) [AU]:US
State or Province Name (full name) [Some-State]:.
Locality Name (eg, city) []:.
Organization Name (eg, company) [Internet Widgits Pty Ltd]:ERSI-COM
Organizational Unit Name (eg, section) []:BOXBORO
Common Name (eg, YOUR name) []:sylvester
Email Address []:jdickens@devnull.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:ERS International
read RSA private key
Enter PEM pass phrase:
writing RSA private key
Signature ok
subject=/C=US/O=ERSI-COM/OU=BOXBORO/CN=sylvester/Email=jdickens@devnull.com
Getting Private key
Done...
[root@sylvester rpms]#
```

This generates the following files:

```
[root@sylvester rpms]# ls -l /etc/httpsd/conf/certs
total 4
-rw-r--r--   1 root     root          839 Apr  5 10:52 cert.cert
-rw-r--r--   1 root     root          700 Apr  5 10:52 cert.csr
-rw-r--r--   1 root     root          887 Apr  5 10:52 cert.key
-rw-r--r--   1 root     root          963 Apr  5 10:52 certprivkey.pem
```

The following lines in Apache-SSL's configuration file point to the Certificate and Key files we have created:

```
SSLCACertificatePath /etc/httpsd/conf/certs
SSLCertificateFile   /etc/httpsd/conf/certs/cert.cert
SSLCertificateKeyFile /etc/httpsd/conf/certs/cert.key
```

Now to start the server:

```
[root@sylvester conf]# /etc/rc.d/init.d/httpsd start
Starting httpsd:                                           [  OK  ]
```

© SANS Institute 2000 - 2002, Author retains full rights.

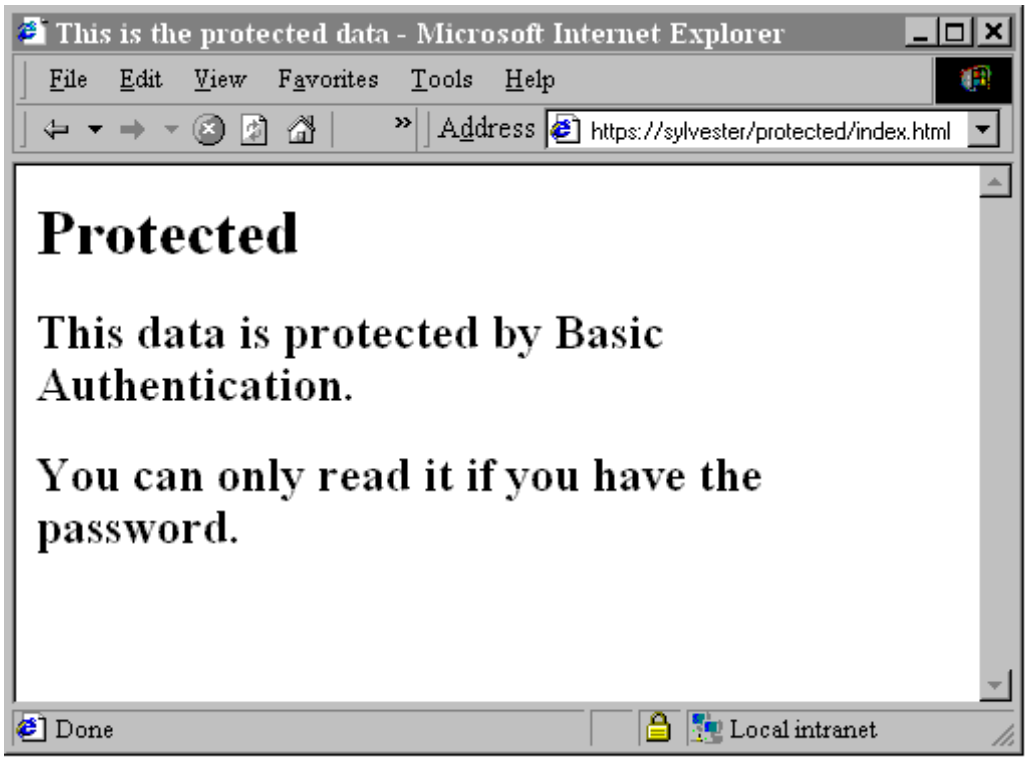
© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

Example 2

Now I point the browser at https://sylvester/protected/index.html
First I see a "Security Alert" popup:

This is warning me that there are some problems with the server's certificate. The first is because a trusted certifying authority has not signed the certificate. The second is because the time was set wrong on the server when I generated the certificate, so the latter case "is not yet valid" is true. The third check passes because I entered "sylvester" for the Common Name above, and this matches the name by which I am accessing the server. For the purpose of this demonstration, I click "Yes" to proceed and see the protected data again:

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.



Note that this time the Address bar says “https” instead of “http”, and that the “Lock” symbol appears in the status bar at the bottom of the window.

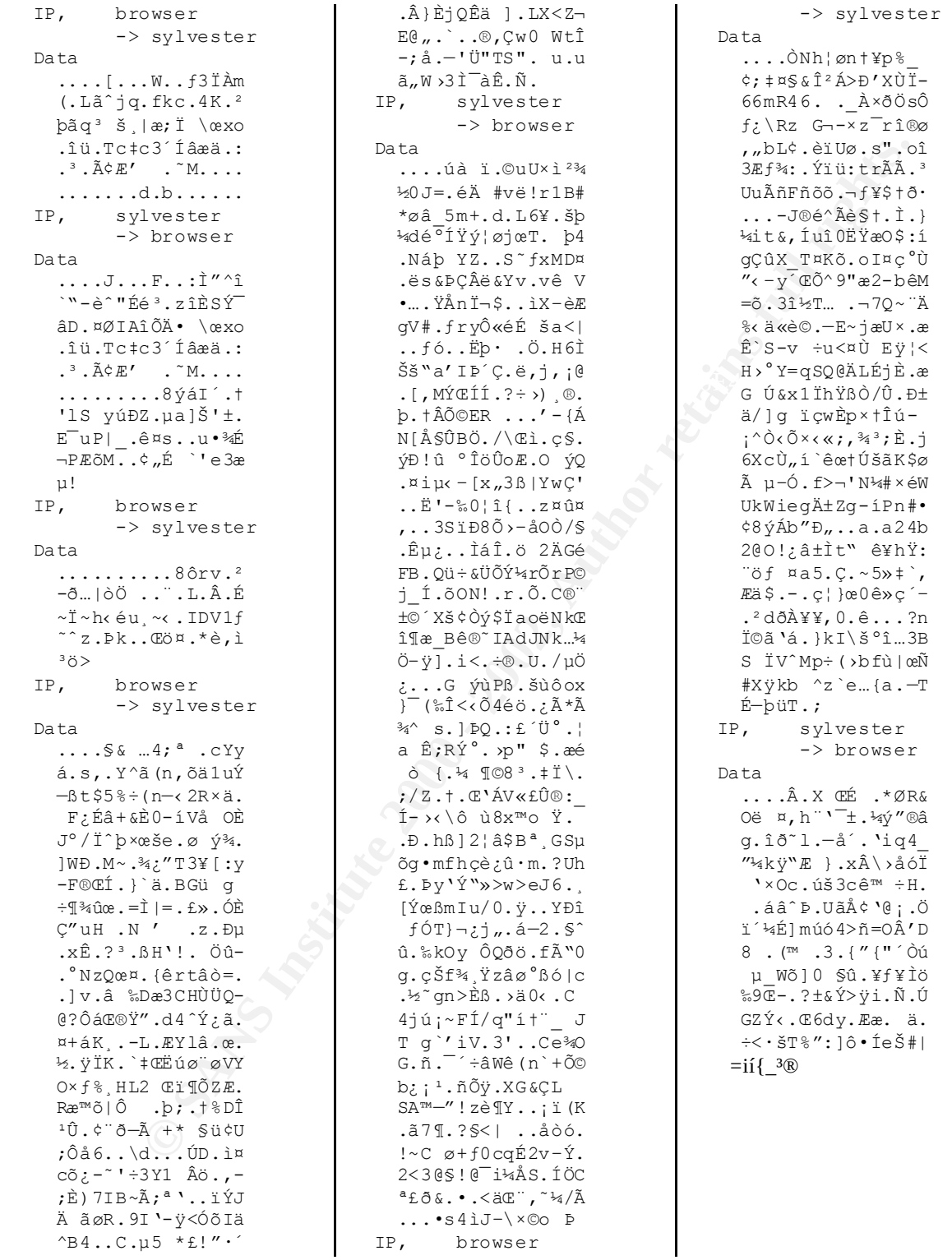
Now what does this look like from the sniffer ? Well, here’s the output, again edited to remove the tcp connection housekeeping:

```
IP,    browser
       -> sylvester
Data
   C....*.........
  ...... ..À.. ...
  ..@..d..b.......
  . .. ADÉK.CwÈ#¢g
  ãXÁ$W
IP,    sylvester
       -> browser
Data
  ....J...F..:Ì”\ß
  ½Fü.Ãk.8äæ7¿J.Ç'
  ^Í7BagxÞ¼Ô¯ \œxo
  .îü.Tc‡c3´Íâæä.:
  .³.Ã¢Æ’  .˜M....
  ...M...I..F..C0,
  .?0,.¨...0...*†H
  †÷......0h1.0...
  U....US1.0...U..
  ..ERSI-COM1.0...
  U....BOXBORO1.0.
  ..U....sylvester
  1 0...*†H†÷.....
  .jdickens@ersi.c
  om0...0104051452
```

```
42Z..02040514524
2Z0h1.0...U....U
S1.0...U....ERSI
-COM1.0...U....B
OXBORO1.0...U...
.sylvester1 0...
*†H†÷......jdick
ens@devnull.com0 Ÿ0
...*†H†÷.......
 .0 ‰.  .Ñw.¶eš"
ÐøH..2Ú½.~‡..àq×
*m t.$'¾^¶0™...c
 Ð_mz¯zìeæ#-¨mª„
_"–..j&5tâUEü „š
¦Š.,Yæ.‡ 4þEF.Œ
.iB×.…)Íÿb jÖ¿Š2
'^s.ë9"G ÜàÛxÚ†Û
¢æQKÉÍ ½......0.
..*†H†÷.......
.{d.%±æ^. ¶}Ýâè Ù
kAW Ö½~]Qåîßß@Ù!
±.â-ÿCÙ"M¥`fõŠ>¦
Š-Aïn{Èêö.?>V..)
E..Lg.‡œ EÆ¼œ[àõ
utÊG®fÌÏ¨Š¥UþuàN
/zËOªœgÆC%x÷‹S—.
```

```
  »šj..i.ÇK. '<hÂ±
   .........
IP,    browser
       -> sylvester
Data
  ....„... Ì.q .):
  {úÔ¬.1Õúð.... Jô
  .ìØÍd0ó.®ÌÕ(ýœ;ã
  Ö.K.O°Q2.˜Ûiöó..
  D.ÊÅï ì ¶ƒ˜± -Sü
  .". YÌ¢#˜Ï¾ið©.
  ˜ð+*y….Þã‡QÔ8 âŠ
  Q.Y'lÂ¥U›.^Abd*:
  ØŸÅY}.ØU........
  ...8ñª.à.-ÿ.šÊË_
  &4#.n|Ü<.,k.SHWî
  ¹Ë”^ .Z.ÄGùË$³F3
  6 7'ò2 swùhÆ
IP,    sylvester
       -> browser
Data
  ..........85ZÒ—ý
  zó!a.Êd.fì)–úÖ°~
  àU…w.Pâ ¼w.^m`^|
  )_V.äi¶.‡R  .ï/è
  œ×C
```


© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.



You can see the certificate information going by, but that's about all that's not encrypted. The password is not revealed, and the data is not revealed.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.

References

SSL:
Iplanet's Introduction to SSL:
http://www.iplanet.com/developer/docs/articles/security/ssl.html

Secure Servers with SSL in the World Wide Web, by Alex Bravo.
http://www.sans.org/infosecFAQ/covertchannels/SSL.htm.

Commercial Versions of Apache that support SSL:
Covalent (Raven): http://www.covalent.net/products/ssl/
C2net (Stronghold): http://www.c2.net/products

HTTP Basic Authentication:
W3 Consortium HTTP V1.0 Specification:
http://www.w3.org/Protocols/HTTP/1.0/spec.html#BasicAA
HTTP Basic Authentication Decoder and Encoder:
http://www.securitystats.com/tools/base64.asp

Open Source Implementations of SSL:
SSLeay: http://www2.psy.uq.edu.au/~ftp/Crypto/.
OpenSSL: http://www.openssl.org.

Open Source Implementations of Apache supporting SSL:
Apache-SSL: http://www.apache-ssl.org.
Mod-SSL: http://www.modssl.org.

Digital Certificates:
A Digital Certificate Introduction, by David E. Savage:
http://www.sans.org/infosecFAQ/encryption/certicate.htm
Digital Certificates: A Secure Method for Digital Transfers by Stephen N. Williams:
http://www.sans.org/infosecFAQ/encryption/digicert.htm.

Transport Layer Security (TLS)
IETF's TLS RFC: http://www.ietf.org/rfc/rfc2246.txt
IETF working group charter for TLS:http://www.ietf.org/html.charters/tls-charter.html.

© SANS Institute 2000 - 2002, Author retains full rights.

© SANS Institute 2000 - 2002 As part of GIAC practical repository. Author retains full rights.