

Configuring a Red Hat Linux System For Secure Websites (HTTPS)

Graham Leach

Managing Partner

TIG – The Imperators Group, Inc.

© 2002 TIG

Table Of Contents

Why Do This At All?	3
SSL Is A Good Thing.....	3
SSL Enables Good Security	3
A General Description Of How WWW Services Works	3
The Problem With Default Web Browser Behavior	4
The Redirection Workaround	5
How This Module Is Structured.....	6
SECTION 0: Preparing For an HTTPS Website.....	7
How To Secure An Additional IP Address	7
How To Change The DNS Data Files	8
How To Tell DNS To Reload Its Data Files.....	9
How To Test The New IP Address.....	10
How To Assign The New IP Address	11
Testing the Newly Assigned IP Address	12
SECTION 1: Creating A Regular Website	13
How To Configure The Web Server	13
The httpd.conf Entry, Annotated	14
Creating The Sample index.html File	15
Restarting The Web Server	16
Testing the Web Server With telnet	18
Testing the Web Server With Lynx.....	19
How to Detect An Insecure Web Session	20
How to Detect A Secure Web Session.....	21
SECTION 2: Creating an SSL Website	22
Creating the secure.dd-industries.com DNS Entry	22
Activating and Testing secure.dd-industries.com TCP/IP Connectivity.....	23
Testing secure.dd-industries.com HTTP Connectivity	24
Testing secure.dd-industries.com HTTPS Connectivity	25
Create the Private Key and Certificate Request	28
How to Generate A Certificate Signing Request File.....	29
How To Remove the Passphrase From The Private Key	30
How To Create The X.509 Certificate	31
Where To Put The X.509 Certificates And Server Key.....	32
What The Inside Of A Decoded Certificate Looks Like.....	34
Enabling The HTTPS Web Server	35
Testing the HTTPS Server	37
SECTION 3: Redirecting HTTP Traffic To The Secure Website.....	41
Rename index.html placeholder.html.....	42
Acknowledgements.....	51

Why Do This At All?

Some people would say that getting a secure web server going is not a trivial task.

It involves the installation and integration of multiple technologies (Apache and OpenSSL), the acquisition (or generation) of a **X.509** server certificate, and intimate knowledge of the underlying protocols (and tools) to get the solution going.

SSL Is A Good Thing

So why bother? Here are two compelling facts that support the idea that secure websites are a good idea.

- 1) SSL raises the security of your website in general.
- 2) SSL is at the heart of eCommerce transactions.

In other words, providing SSL may enhance your reputation and certainly helps to make your customers feel more secure when browsing. They may even become comfortable enough to overcome the psychological barrier many people have when it comes to conducting online financial transactions.

SSL Enables Good Security

First of all, understand that the vast majority of Internet traffic is **not** secure by any means. Traffic (meaning TCP/IP packets) flow on quasi-public digital highways and can be sniffed, diverted and otherwise manipulated in transit by malevolent forces.

This applies equally to the **BIG 3** Internet applications, remote access, email and web.

Technology	Port	Tool
Remote Access	21	TELNET
email – send	25	SMTP
email - read	110	POP
World Wide Web	80	HTTP

All of these technologies pass their information (including supplied credentials) **in the clear**. This means the only thing preventing someone from stealing your credentials is either indifference or ignorance – neither of which can be considered a worthwhile security policy.

A General Description Of How WWW Services Works

This particular module focuses on how to secure the third member of the BIG 3, the World Wide Web.

Here's what happens in a typical WWW session:

- 1) A web browser forms a connection with an Internet entity on **port 80**.
- 2) A web server waiting on **port 80** responds by indicating it is ready for instructions.
- 3) The web browser issues a request to the web server, usually **GET //index.html**.
- 4) The web server either returns the **index.html** file or an error message.
- 5) The web server breaks the connection and goes back to waiting for connections.

Bear in mind that this is all done using the HTTP protocol.

The Problem With Default Web Browser Behavior

Everyone in the world who is used to surfing the web is accustomed to using the HTTP protocol implicitly. This functionality has progressively been programmed into our web browsers in an effort to make using the WWW simple, easy and fun.

Consider the following:

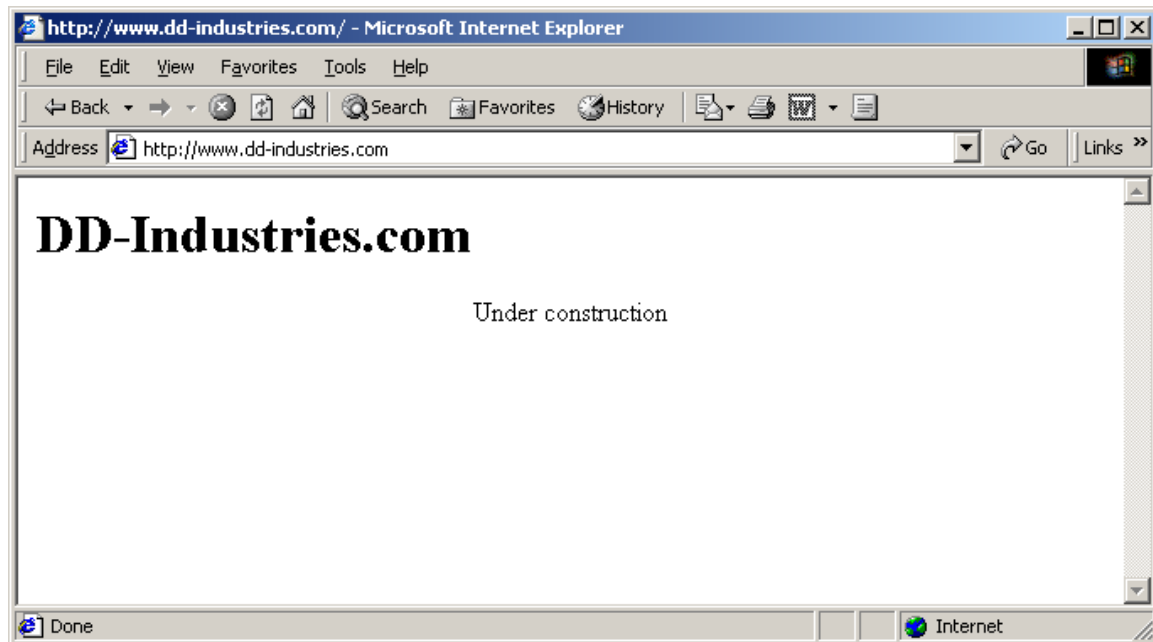


In the above example,

In the URL supplied by yours truly, there's no mention of protocol, only a Fully Qualified Domain Name (FQDN), which will hopefully be resolved to an IP address by my DNS server.

While this URL is technically incomplete, the connection is still going to be formed without any complaint by the web browser. I click on "OK" and here is what the browser returns to me:

Consider the following:



In the above example,

So, if I didn't supply enough information, how did it still work?

The web browser supplied additional information on a "best guess" basis - the "best guess" being that I was interested in connecting to **www.dd-industries.com** with the Hyper Text Transfer Protocol, HTTP, on **port 80** of that computer.

Browsers Do Not Default To HTTPS

This output in the previous example is perfectly acceptable to those surfers only ever accessing information with HTTP and casually browsing. Unfortunately, surfers get really anxious about exchanging private information (such as credit card numbers) using HTTP because they are aware of many circumstances of Internet-oriented credit card fraud.

These days, surfers look for "that little lock" when conducting financially sensitive transactions. What many of them don't know (and don't care about) is that they are no longer using HTTP when "that little lock" is visible on the bottom status bar of the web browser. At that point, they're using **HTTPS**, a completely different protocol.

As we've already seen, most web browsers default to the HTTP protocol when the protocol is not explicitly stated. The **HTTP** protocol uses **port 80**. Websites providing secure web services via **HTTPS** use **port 443**.

So, if you're interested in running an eCommerce site, HTTP is only interesting if it somehow leads the customer to HTTPS. But, and this is a **big** but, customers unaccustomed to specifying the protocol as part of the URL demand **HTTPS** when their web browsers are defaulting to **HTTP**!

The Redirection Workaround

The answer to getting around this problem is to create at least **two** websites.

The first website is available via HTTP on port 80 and it has some very basic functionality

- 1) It checks the browser capabilities.
- 2) It may issue a warning if the browser security is weak (key length is too short).
- 3) It may issue a warning if the browser is incapable of supporting HTTPS.
- 4) If everything looks OK, it transparently redirect the web browser to the secure website.

How This Module Is Structured

This module will show how to:

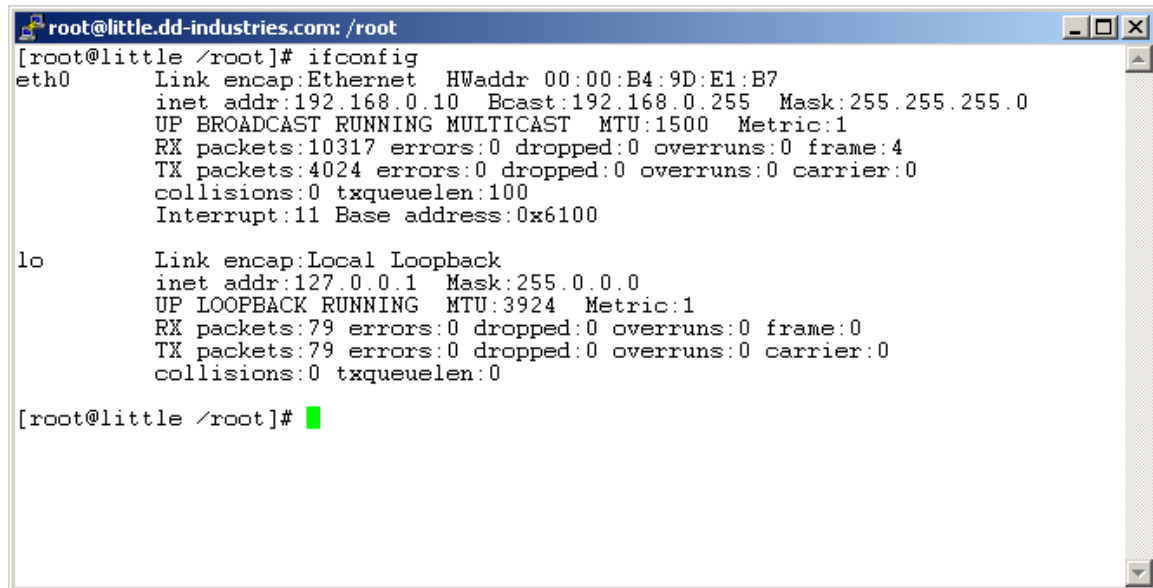
- Prepare for an HTTPS website
- Create a HTTP website & making sure it works
- Create an HTTPS website & making sure it works
- Redirecting traffic from the HTTP website to the HTTPS website

SECTION 0: Preparing For an HTTPS Website

How To Secure An Additional IP Address

This particular system starts out with the IP address **192.168.0.10** and has a functioning web server. These are a direct result of a “stock” Red Hat install.

Consider the following:



```
root@little.dd-industries.com: /root
[root@little /root]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:00:B4:9D:E1:B7
          inet addr:192.168.0.10  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:10317 errors:0 dropped:0 overruns:0 frame:4
          TX packets:4024 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0x6100

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:79 errors:0 dropped:0 overruns:0 frame:0
          TX packets:79 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

[root@little /root]#
```

In the above example,

Because this process will eventually result in an SSL enabled server, some additional steps at the outset must be performed that wouldn't necessarily be required for a regular website.

One of those steps is the activation of an additional IP address. Because we are on a private network under our own administration, securing an IP address is trivial and the secure website will be installed on the IP address **192.168.0.11**.

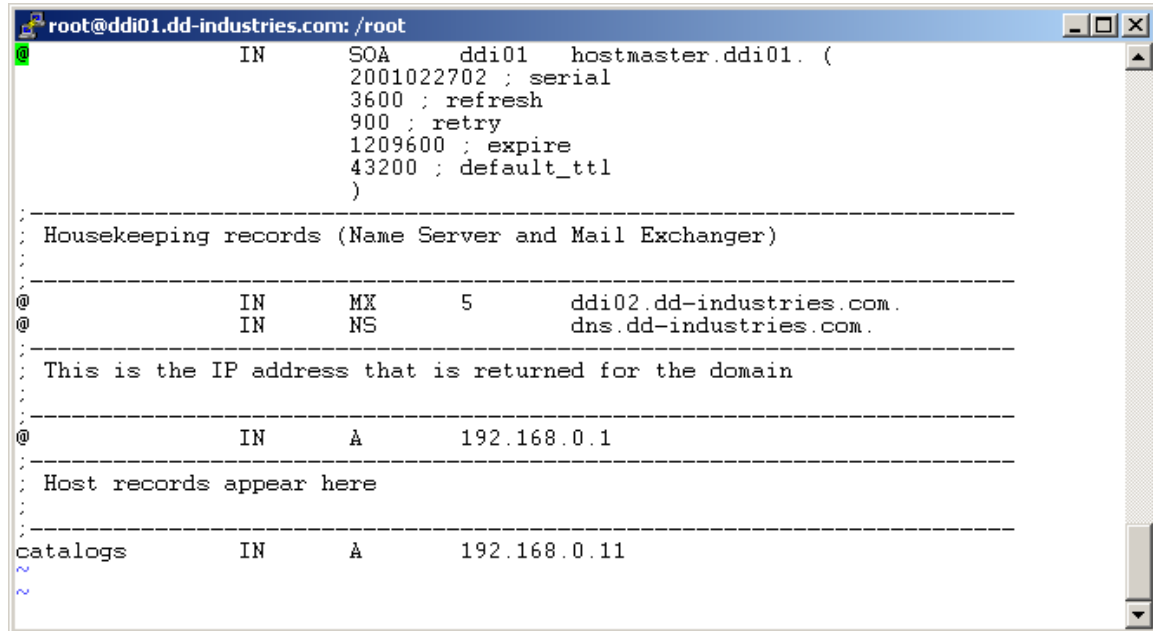
In the case of an Internet entity, your ISP will have to be contacted for a valid IP address.

They may also be required to perform some DNS work for you.

How To Change The DNS Data Files

Part of the challenge of making HTTPS work properly involves securing an additional IP address for that service. This is because of how SSL works. It operates at the transport level, below name services. This means that every SSL server requires a **unique** IP address.

Consider the following:

A terminal window titled 'root@ddi01.dd-industries.com: /root' displays a DNS configuration file. The file contains a SOA record for ddi01, followed by housekeeping records for Name Server and Mail Exchanger. It then lists MX and NS records for ddi02 and dns. Finally, it shows an A record for the domain pointing to 192.168.0.1, and a new A record for 'catalogs' pointing to 192.168.0.11. The terminal output is as follows:

```
root@ddi01.dd-industries.com: /root
@          IN      SOA      ddi01  hostmaster.ddi01. (
                        2001022702 ; serial
                        3600 ; refresh
                        900 ; retry
                        1209600 ; expire
                        43200 ; default_ttl
                        )
;-----
; Housekeeping records (Name Server and Mail Exchanger)
;-----
@          IN      MX      5      ddi02.dd-industries.com.
@          IN      NS      dns.dd-industries.com.
;-----
; This is the IP address that is returned for the domain
;-----
@          IN      A       192.168.0.1
;-----
; Host records appear here
;-----
catalogs   IN      A       192.168.0.11
~
~
```

In the above example a line has been added to the file that holds the DNS record for this domain, **dd-industries.com**. The line details the name of the new server, **catalogs**, plus the IP address that the name is associated with, **192.168.0.11**.

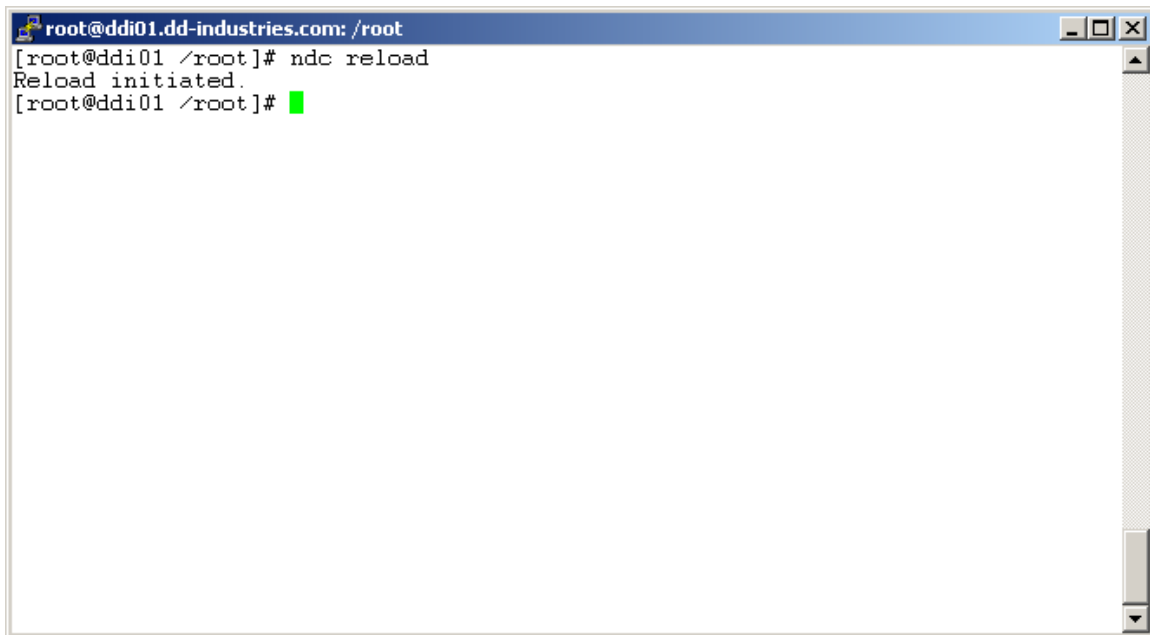
The next step is to reload the DNS database:

How To Tell DNS To Reload Its Data Files

There are a couple of ways to reload the DNS database. One involves using the startup scripts, another killing and restarting the server directly.

There is also a command suite available to tell the **named** program what to do. One of the commands tells it to reload its databases. The command is **ndc reload**.

Consider the following:

A terminal window with a blue title bar containing the text 'root@ddi01.dd-industries.com: /root'. The terminal shows the command '[root@ddi01 /root]# ndc reload' being entered, followed by the output 'Reload initiated.' and a new prompt '[root@ddi01 /root]#'. A green cursor is visible at the end of the second prompt. The window has standard Linux window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

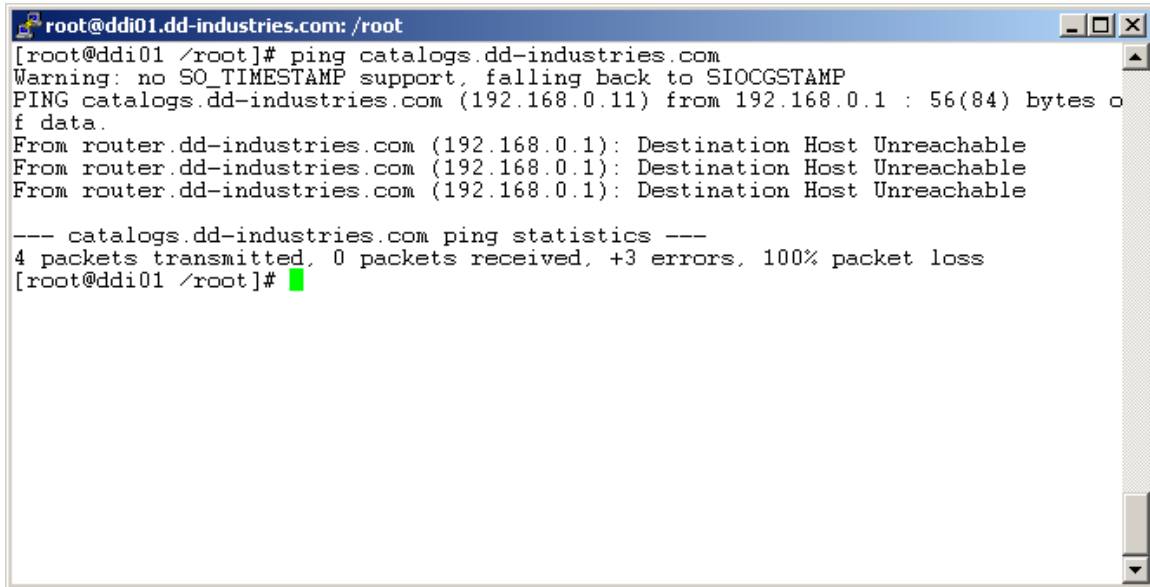
```
root@ddi01.dd-industries.com: /root
[root@ddi01 /root]# ndc reload
Reload initiated.
[root@ddi01 /root]#
```

In the above example the **ndc reload** command was used to tell the **named** DNS server to reload its data files.

How To Test The New IP Address

Just because an IP address has been secured and activated in DNS does not a webserver make!

Consider the following:

A terminal window titled 'root@ddi01.dd-industries.com: /root' showing a failed ping command. The user enters 'ping catalogs.dd-industries.com'. The terminal displays a warning about SO_TIMESTAMP support, followed by the ping command output showing three 'Destination Host Unreachable' messages from the router. Ping statistics show 4 packets transmitted, 0 received, and 100% packet loss. The prompt returns to the root user.

```
root@ddi01.dd-industries.com: /root
[root@ddi01 /root]# ping catalogs.dd-industries.com
Warning: no SO_TIMESTAMP support, falling back to SIOCGSTAMP
PING catalogs.dd-industries.com (192.168.0.11) from 192.168.0.1 : 56(84) bytes of data.
From router.dd-industries.com (192.168.0.1): Destination Host Unreachable
From router.dd-industries.com (192.168.0.1): Destination Host Unreachable
From router.dd-industries.com (192.168.0.1): Destination Host Unreachable

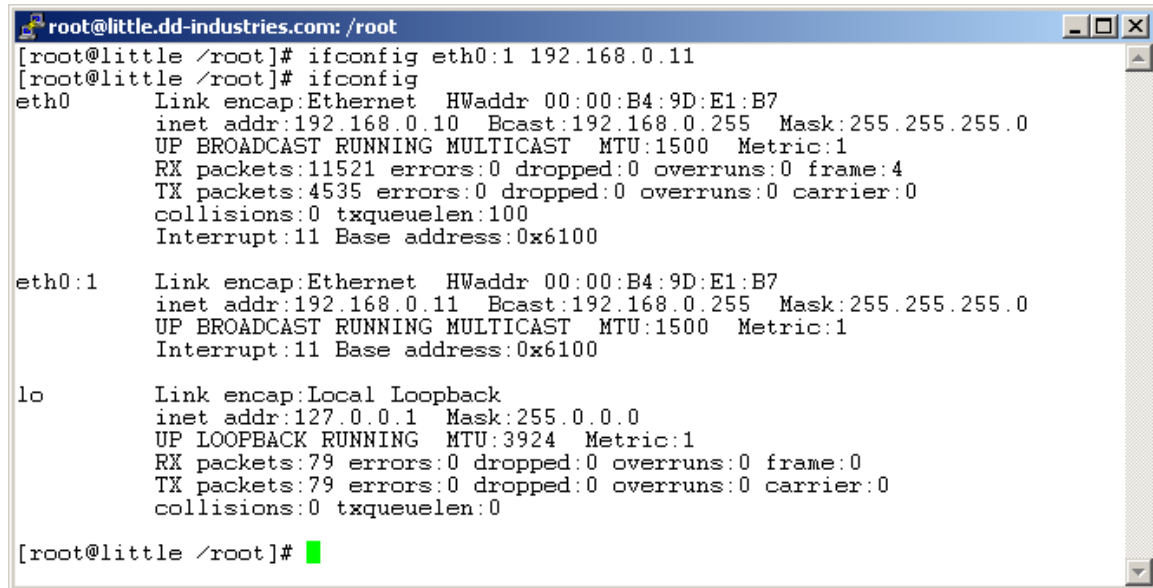
--- catalogs.dd-industries.com ping statistics ---
4 packets transmitted, 0 packets received, +3 errors, 100% packet loss
[root@ddi01 /root]#
```

In the above example the **ping** command was used to test both DNS and IP. While the Fully Qualified Domain Name (FQDN) **catalogs.dd-industries.com** resolved properly to the IP address **192.168.0.11**, there was no answer to the **ping** because no machine has yet had the IP address

How To Assign The New IP Address

Now that an IP address has been secured and activated in DNS, it is now necessary to link that IP address to the physical computer.

Consider the following:



```
root@little.dd-industries.com: /root
[root@little /root]# ifconfig eth0:1 192.168.0.11
[root@little /root]# ifconfig
eth0      Link encap:Ethernet  HWaddr 00:00:B4:9D:E1:B7
          inet addr:192.168.0.10  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:11521 errors:0 dropped:0 overruns:0 frame:4
          TX packets:4535 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          Interrupt:11 Base address:0x6100

eth0:1    Link encap:Ethernet  HWaddr 00:00:B4:9D:E1:B7
          inet addr:192.168.0.11  Bcast:192.168.0.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          Interrupt:11 Base address:0x6100

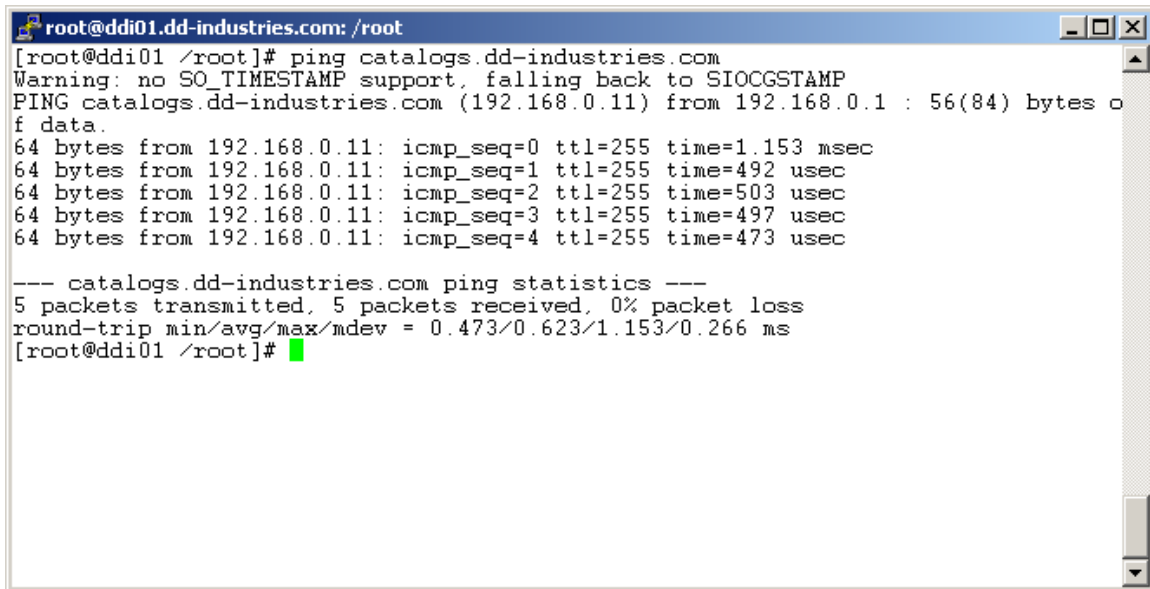
lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:3924  Metric:1
          RX packets:79 errors:0 dropped:0 overruns:0 frame:0
          TX packets:79 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0

[root@little /root]#
```

In the above example,

Testing the Newly Assigned IP Address

Consider the following:

A terminal window titled 'root@ddi01.dd-industries.com: /root' showing the execution of a ping command. The command is 'ping catalogs.dd-industries.com'. The output shows a warning about SO_TIMESTAMP support, followed by five successful ping responses from 192.168.0.11 with varying times. Finally, it shows ping statistics: 5 packets transmitted, 5 received, 0% loss, and round-trip times. The prompt returns to the root user.

```
root@ddi01.dd-industries.com: /root
[root@ddi01 /root]# ping catalogs.dd-industries.com
Warning: no SO_TIMESTAMP support, falling back to SIOCGSTAMP
PING catalogs.dd-industries.com (192.168.0.11) from 192.168.0.1 : 56(84) bytes of data.
64 bytes from 192.168.0.11: icmp_seq=0 ttl=255 time=1.153 msec
64 bytes from 192.168.0.11: icmp_seq=1 ttl=255 time=492 usec
64 bytes from 192.168.0.11: icmp_seq=2 ttl=255 time=503 usec
64 bytes from 192.168.0.11: icmp_seq=3 ttl=255 time=497 usec
64 bytes from 192.168.0.11: icmp_seq=4 ttl=255 time=473 usec

--- catalogs.dd-industries.com ping statistics ---
5 packets transmitted, 5 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.473/0.623/1.153/0.266 ms
[root@ddi01 /root]#
```

In the above example,

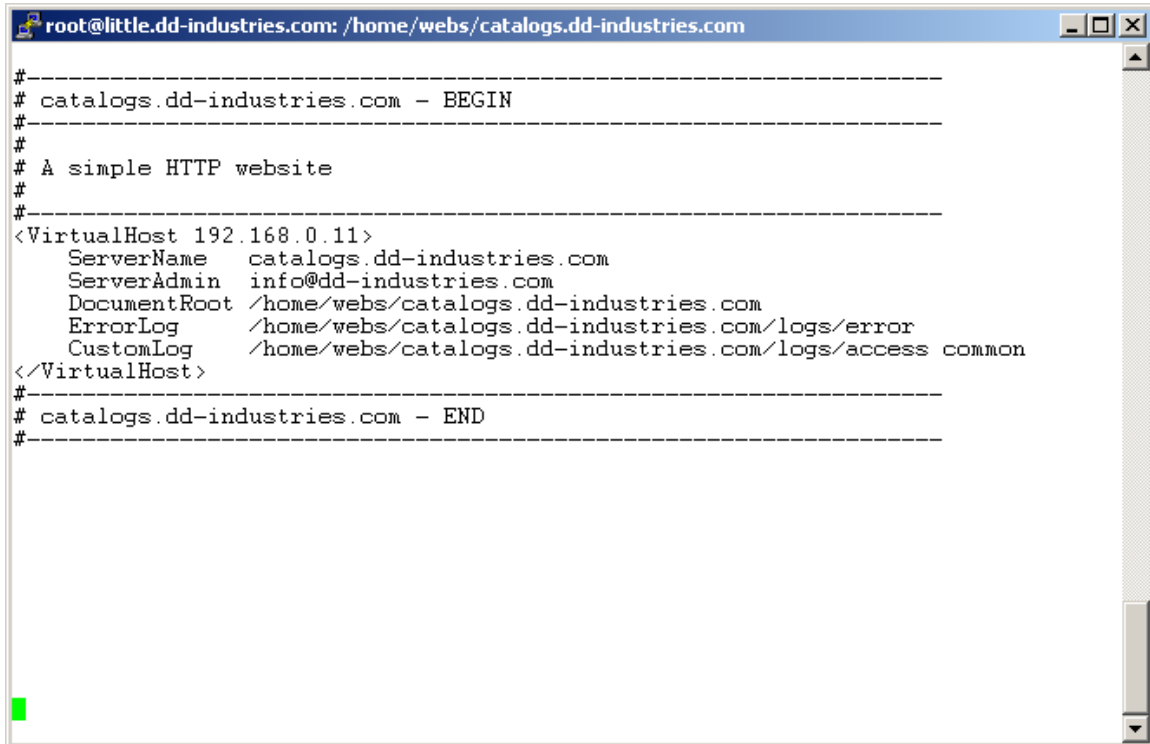
The ping resolves and gets an answer!

SECTION 1: Creating A Regular Website

How To Configure The Web Server

To create a regular website the **httpd.conf** file for your Linux box must be edited and an entry added.

Consider the following:



```
root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com
#-----
# catalogs.dd-industries.com - BEGIN
#-----
#
# A simple HTTP website
#
#-----
<VirtualHost 192.168.0.11>
    ServerName    catalogs.dd-industries.com
    ServerAdmin   info@dd-industries.com
    DocumentRoot  /home/webs/catalogs.dd-industries.com
    ErrorLog      /home/webs/catalogs.dd-industries.com/logs/error
    CustomLog     /home/webs/catalogs.dd-industries.com/logs/access common
</VirtualHost>
#-----
# catalogs.dd-industries.com - END
#-----
```

In the above example the **httpd.conf** file has been edited and a website entry has been added.

The httpd.conf Entry, Annotated

```
#-----  
# catalogs.dd-industries.com - BEGIN  
#-----  
#  
# A simple HTTP website  
#-----
```

This is a comment. All characters after the hash mark, #, are ignored by the web server.

<VirtualHost 192.168.0.11>

This ties the web server to a particular IP address. Because no port is specified, the default port (80) is assumed to be the port that the server is being instructed to listen to.

ServerName **catalogs.dd-industries.com**

This tells the web server the FQDN that corresponds to the IP address supplied above.

ServerAdmin **info@dd-industries.com**

This tells the web server who to contact if there are problems

DocumentRoot **/home/webs/catalogs.dd-industries.com**

This tells the web server where to locate its files, where the website for this entry physically resides.

ErrorLog **/home/webs/catalogs.dd-industries.com/logs/error**

Tells the web server which file to store error messages.

CustomLog **/home/webs/catalogs.dd-industries.com/logs/access**

Tells the web server which file to store access information and other information not related to errors.

</VirtualHost>

This terminates the entry, telling the web server that no further information is available on this website.

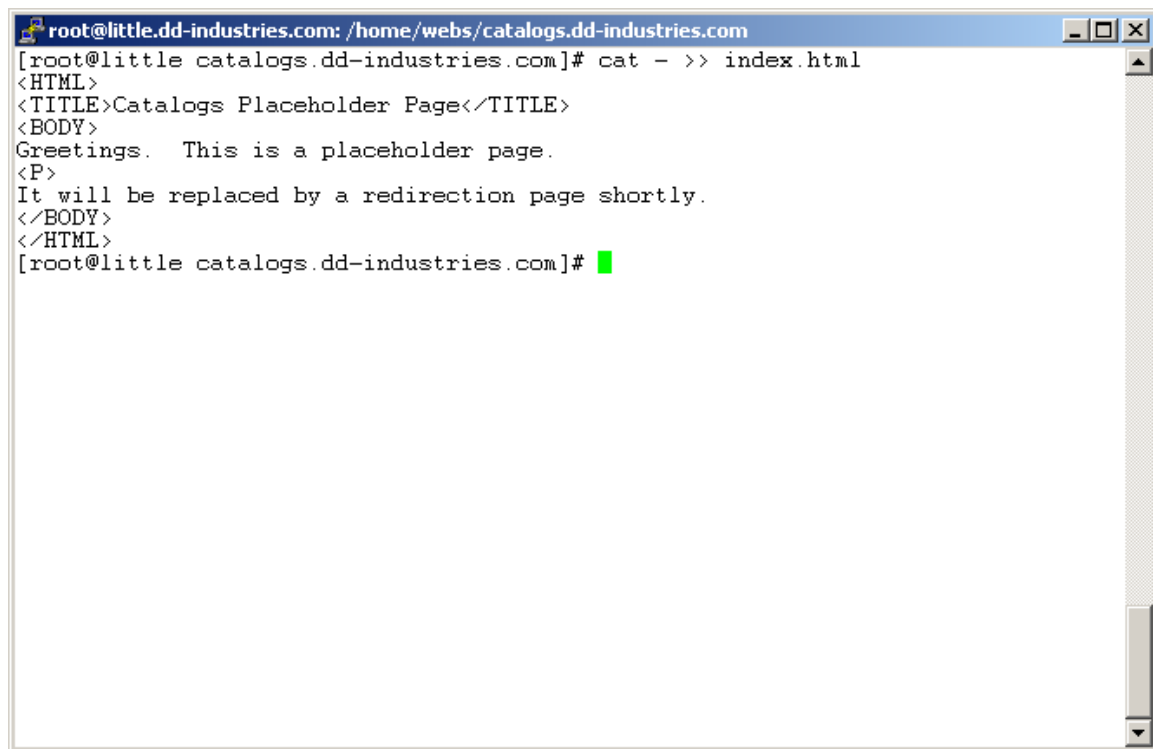
Testing the Web Server

Testing this installation is simple.

- Create a sample **index.html** on the target server
- Restart the web server so it reads in the new entry
- Test the web server with telnet
- Test the web server with lynx

Creating The Sample index.html File

Consider the following:

A terminal window titled 'root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com'. The prompt is '[root@little catalogs.dd-industries.com]#'. The user enters 'cat - >> index.html'. The terminal shows the following HTML content being typed: '<HTML>', '<TITLE>Catalogs Placeholder Page</TITLE>', '<BODY>', 'Greetings. This is a placeholder page.', '<P>', 'It will be replaced by a redirection page shortly.', '</BODY>', '</HTML>'. The prompt returns to '[root@little catalogs.dd-industries.com]#'.

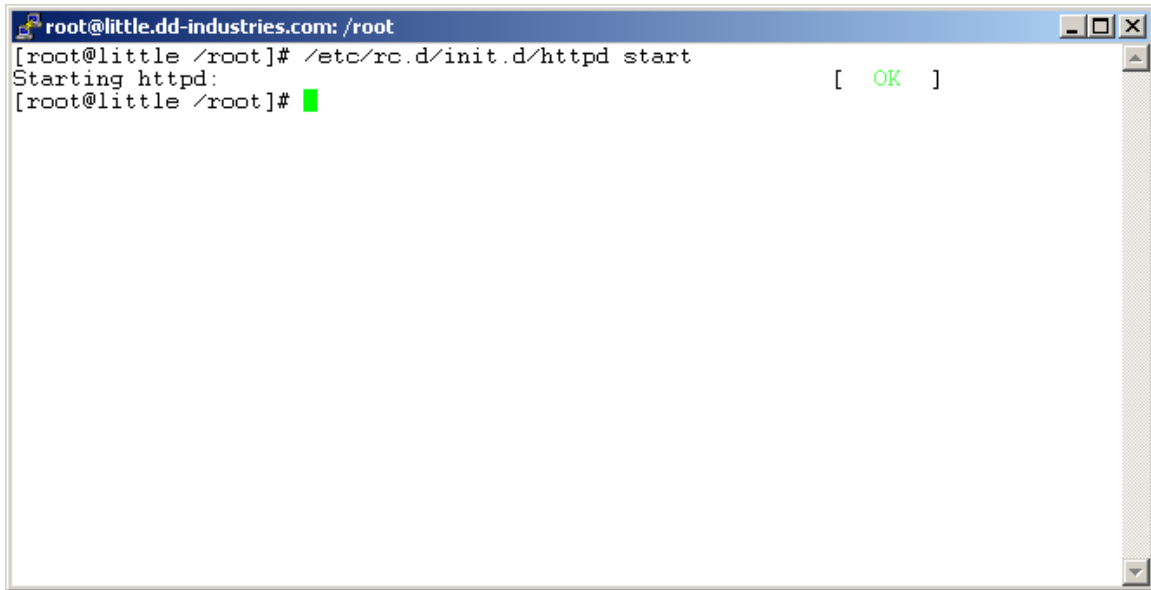
```
root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com
[root@little catalogs.dd-industries.com]# cat - >> index.html
<HTML>
<TITLE>Catalogs Placeholder Page</TITLE>
<BODY>
Greetings. This is a placeholder page.
<P>
It will be replaced by a redirection page shortly.
</BODY>
</HTML>
[root@little catalogs.dd-industries.com]#
```

In the above example the `-` character was used to redirect standard input to the file **index.html**. The lines were typed in directly and then **CTRL-D** was used to tell Linux that the file was complete. **CTRL-D** is the file termination character in Unix.

Restarting The Web Server

Use the script to restart the server:

Consider the following:

A terminal window with a blue title bar containing the text 'root@little.dd-industries.com: /root'. The terminal shows the command '/etc/rc.d/init.d/httpd start' being executed. The output is 'Starting httpd:' followed by a green space and '[OK]' in green. The prompt returns to '[root@little /root]#'.

```
root@little.dd-industries.com: /root
[root@little /root]# /etc/rc.d/init.d/httpd start
Starting httpd: [ OK ]
[root@little /root]#
```

In the above example,

You can also accomplish this with the **apachectl** command:

Consider the following:

A terminal window with a blue title bar containing the text "root@little.dd-industries.com: /etc/httpd/conf". The terminal shows the command "apachectl restart" being executed, followed by the output "/usr/sbin/apachectl restart: httpd restarted". The prompt returns to "[root@little conf]#" with a green cursor.

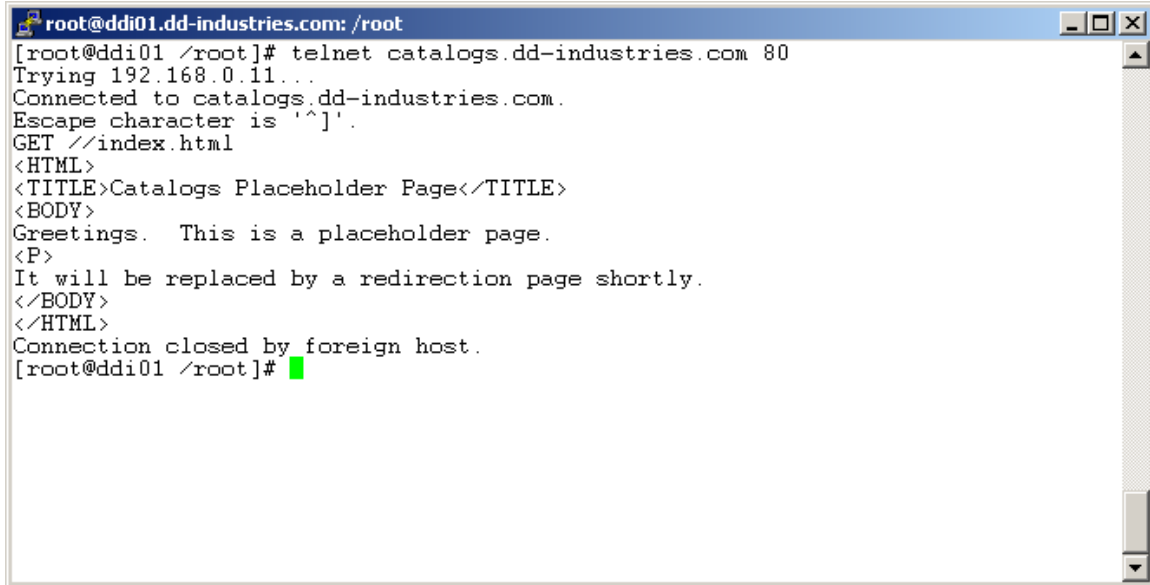
```
root@little.dd-industries.com: /etc/httpd/conf
[root@little conf]# apachectl restart
/usr/sbin/apachectl restart: httpd restarted
[root@little conf]#
```

In the above example,

Testing the Web Server With telnet

There is a way to test the web server from the command line using the **telnet** tool.

Consider the following:



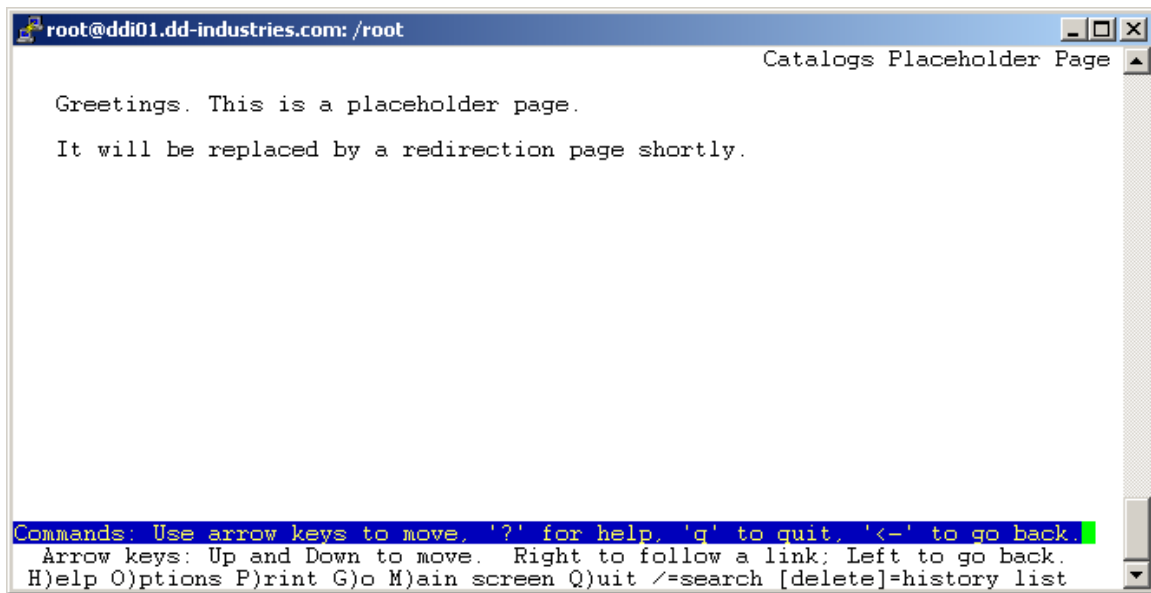
```
root@ddi01.dd-industries.com: /root
[root@ddi01 /root]# telnet catalogs.dd-industries.com 80
Trying 192.168.0.11...
Connected to catalogs.dd-industries.com.
Escape character is '^]'.
GET //index.html
<HTML>
<TITLE>Catalogs Placeholder Page</TITLE>
<BODY>
Greetings. This is a placeholder page.
<P>
It will be replaced by a redirection page shortly.
</BODY>
</HTML>
Connection closed by foreign host.
[root@ddi01 /root]#
```

In the above example, a connection to a web server was established on port **80** using the **telnet** command. Once the connection was established, the **GET** command was used to ask for the file **//index.html**, which was immediately returned by the web server.

Note that once **index.html** was delivered, the web server terminated the connection. In other words, the connection was initiated by the client, but ended by the server.

Testing the Web Server With Lynx

Consider the following:

A screenshot of a Lynx web browser window. The title bar shows the URL 'root@ddi01.dd-industries.com: /root' and window control buttons. The page title is 'Catalogs Placeholder Page'. The main content area displays the text: 'Greetings. This is a placeholder page.' followed by 'It will be replaced by a redirection page shortly.' The bottom status bar contains navigation instructions: 'Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.' and 'Arrow keys: Up and Down to move. Right to follow a link; Left to go back.' followed by a list of shortcuts: 'H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list'.

```
root@ddi01.dd-industries.com: /root Catalogs Placeholder Page

Greetings. This is a placeholder page.

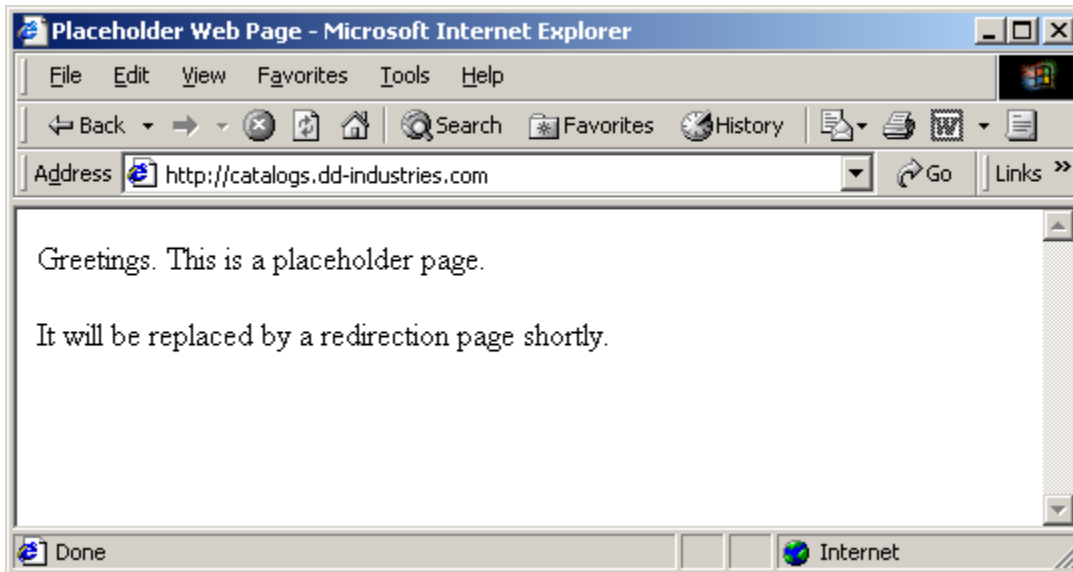
It will be replaced by a redirection page shortly.

Commands: Use arrow keys to move, '?' for help, 'q' to quit, '<-' to go back.
Arrow keys: Up and Down to move. Right to follow a link; Left to go back.
H)elp O)ptions P)rint G)o M)ain screen Q)uit /=search [delete]=history list
```

In the above example the **lynx** web browser was used to verify that the website was working.

How to Detect An Insecure Web Session

Consider the following:

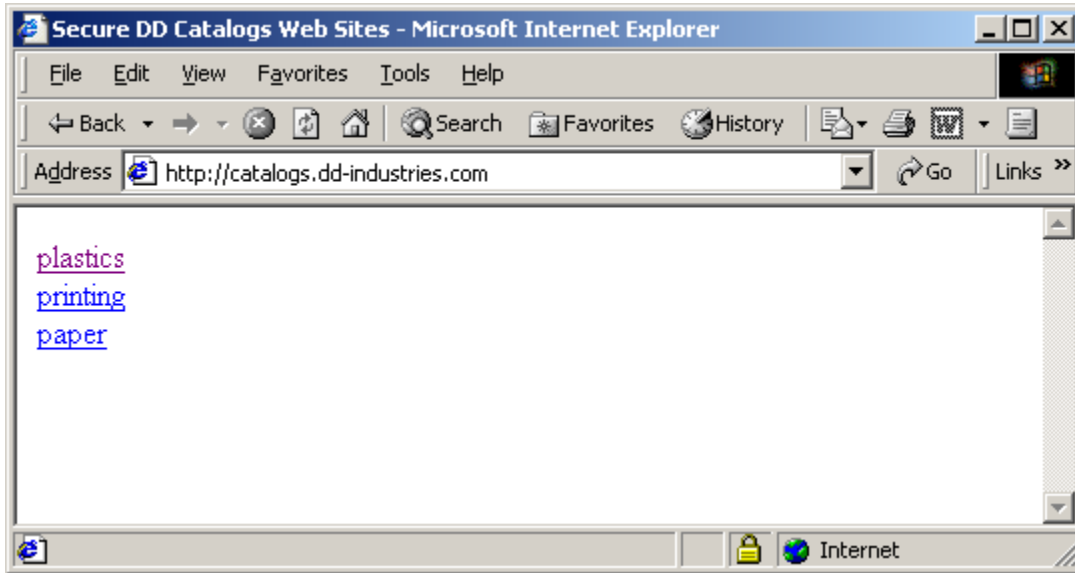


In the above example, Microsoft Internet Explorer was used to verify that the web server configured for **catalogs.dd-industries.com** responded to client requests. As you can see from the lack of a “little lock” on the bottom status bar, this is an insecure connection based on HTTP.

How to Detect A Secure Web Session

Here is a preview of what this module is trying to accomplish:

Consider the following:



In the above example, Microsoft Internet Explorer was used to verify that the web server configured for **catalogs.dd-industries.com** responded to client requests. As you can see from the "little lock" on the bottom status bar, this is a secure connection based on HTTPS (even though the URL in the **Address box** may lead you to conclude otherwise).

SECTION 2: Creating an SSL Website

Now that a basic web site has been enabled and established the SSL website must be configured.

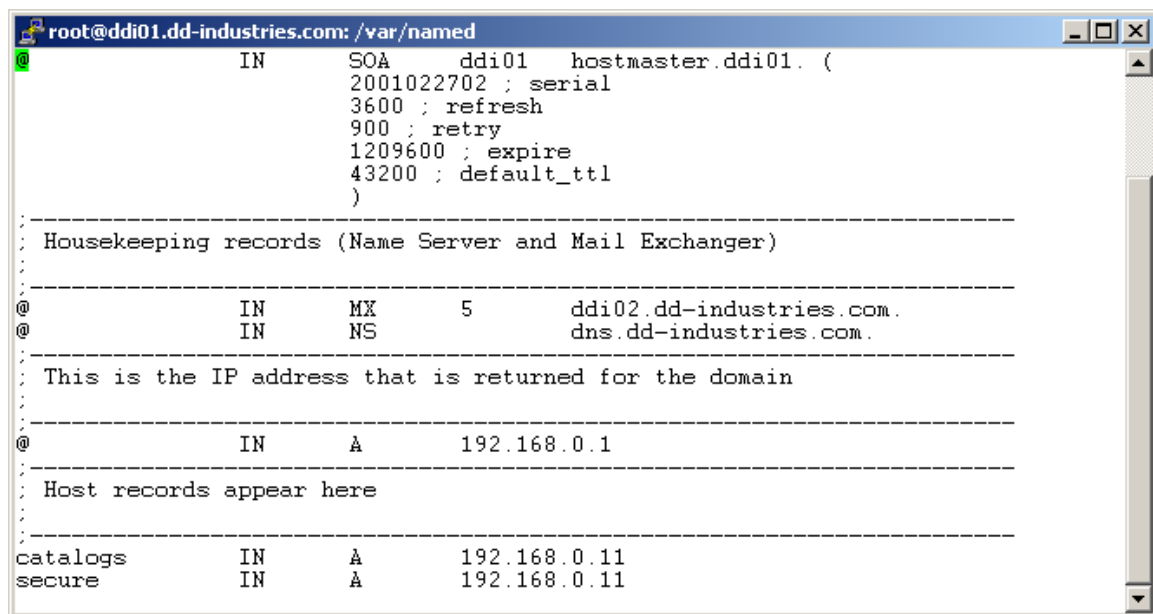
SSL is a complicated topic that revolves around **certificates**, particularly **X.509** certificates. For this example to work, a certificate must be obtained or generated for this server.

First, we want a destination for all of our secure traffic – called **secure.dd-industries.com**. To make that destination a reality we have to enter it in a DNS server.

Creating the secure.dd-industries.com DNS Entry

Add the entry to the DNS database

Consider the following:



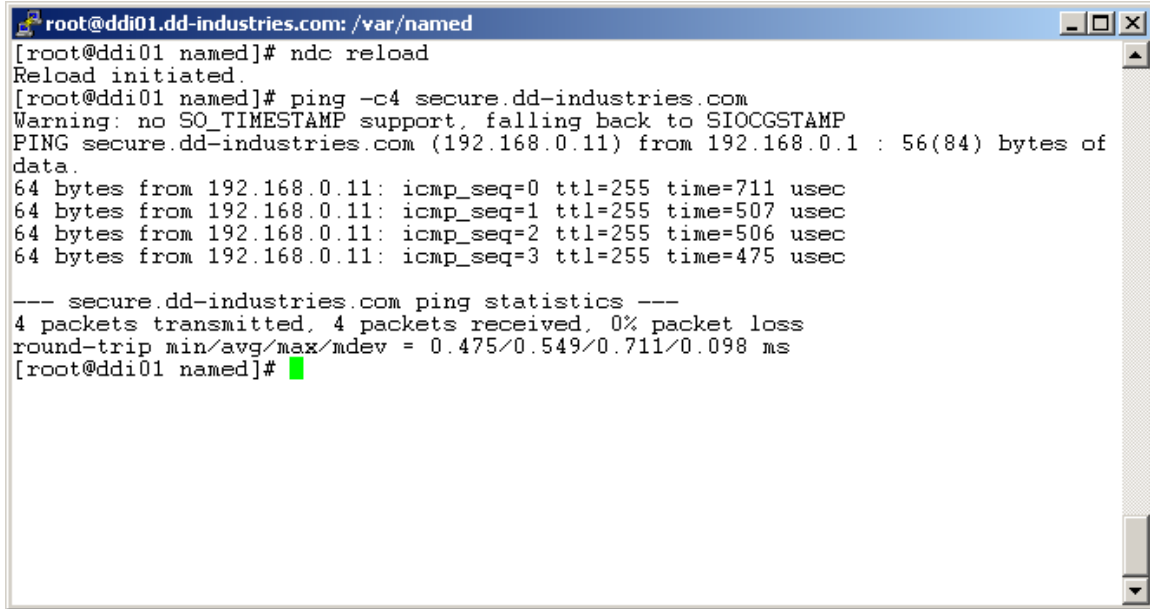
```
root@ddi01.dd-industries.com: /var/named
IN      SOA      ddi01  hostmaster.ddi01. (
        2001022702 ; serial
        3600      ; refresh
        900       ; retry
        1209600   ; expire
        43200     ; default_ttl
        )
;-----
; Housekeeping records (Name Server and Mail Exchanger)
;-----
@       IN      MX      5      ddi02.dd-industries.com.
@       IN      NS      dns.dd-industries.com.
;-----
; This is the IP address that is returned for the domain
;-----
@       IN      A       192.168.0.1
;-----
; Host records appear here
;-----
catalogs      IN      A       192.168.0.11
secure        IN      A       192.168.0.11
```

In the above example,

Activating and Testing secure.dd-industries.com TCP/IP Connectivity

Restart the name server and test name resolution with **ping**.

Consider the following:



```
root@ddi01.dd-industries.com: /var/named
[root@ddi01 named]# ndc reload
Reload initiated.
[root@ddi01 named]# ping -c4 secure.dd-industries.com
Warning: no SO_TIMESTAMP support, falling back to SIOCGSTAMP
PING secure.dd-industries.com (192.168.0.11) from 192.168.0.1 : 56(84) bytes of
data.
64 bytes from 192.168.0.11: icmp_seq=0 ttl=255 time=711 usec
64 bytes from 192.168.0.11: icmp_seq=1 ttl=255 time=507 usec
64 bytes from 192.168.0.11: icmp_seq=2 ttl=255 time=506 usec
64 bytes from 192.168.0.11: icmp_seq=3 ttl=255 time=475 usec

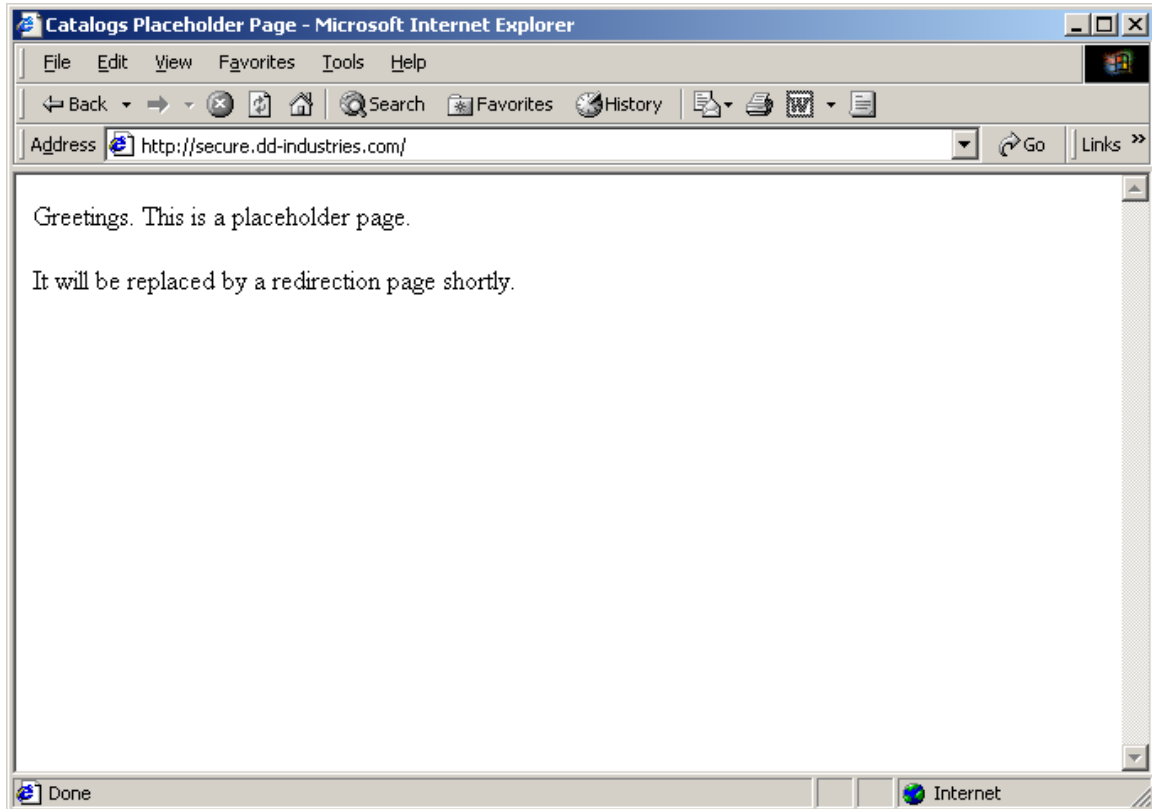
--- secure.dd-industries.com ping statistics ---
4 packets transmitted, 4 packets received, 0% packet loss
round-trip min/avg/max/mdev = 0.475/0.549/0.711/0.098 ms
[root@ddi01 named]#
```

In the above example,

Testing secure.dd-industries.com HTTP Connectivity

Confirm that **secure.dd-industries.com** responds to normal HTTP requests:

Consider the following:

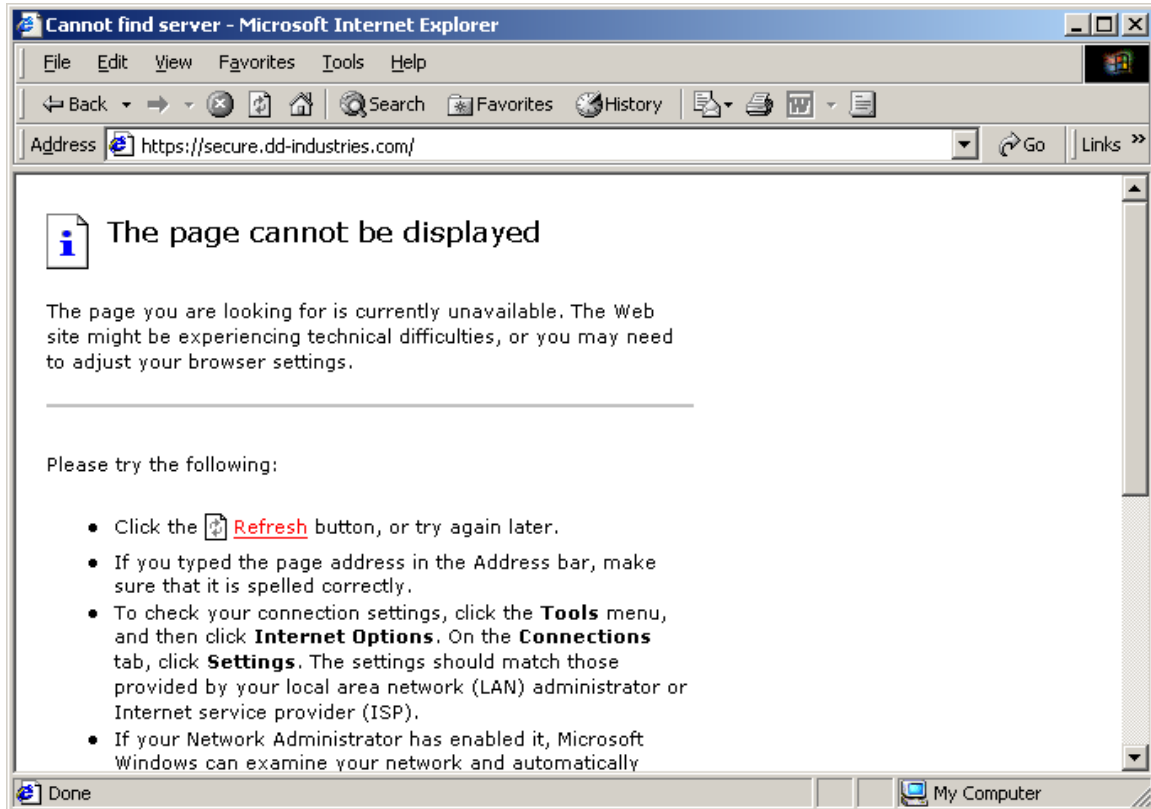


In the above example,

Testing secure.dd-industries.com HTTPS Connectivity

Without any further configuration this should fail:

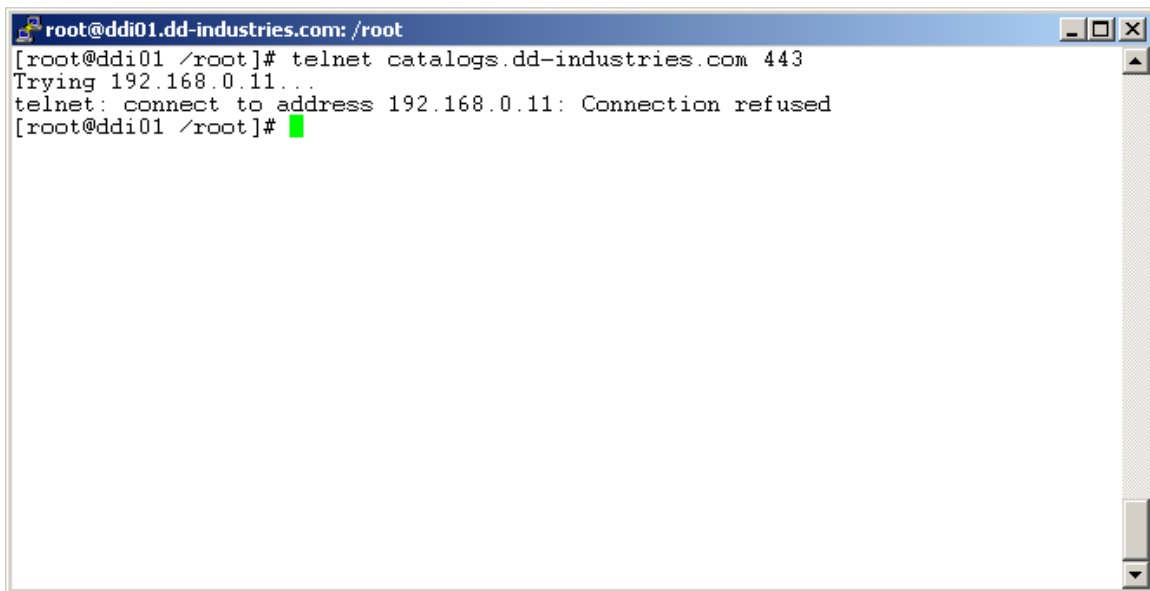
Consider the following:



In the above example,

Time to test the connection directly with **telnet**.

Consider the following:



```
root@ddi01.dd-industries.com: /root
[root@ddi01 /root]# telnet catalogs.dd-industries.com 443
Trying 192.168.0.11...
telnet: connect to address 192.168.0.11: Connection refused
[root@ddi01 /root]#
```

In the above example,

The server is not yet available

Enabling SSL

OpenSSL

Two things are needed – a certificate and a key.

X.509 Certificate Generation

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

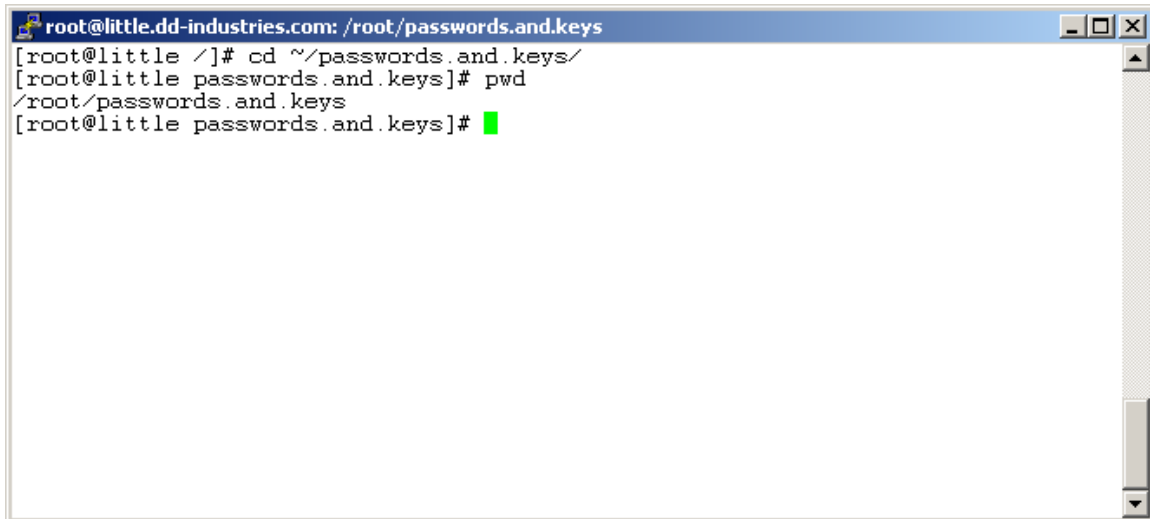
This entire section needs to be written!!!!!!!!!!!!!!

This entire section needs to be written!!!!!!!!!!!!!!

Create the Private Key and Certificate Request

First of all, have a care and generate your keys in a safe place! I recommend that you use the **/root** directory

Consider the following:

A terminal window titled 'root@little.dd-industries.com: /root/passwords.and.keys'. The window shows a sequence of commands and their outputs: first, 'cd ~/passwords.and.keys/' is entered, followed by 'pwd', which outputs '/root/passwords.and.keys'. The prompt then returns to '[root@little passwords.and.keys]#'.

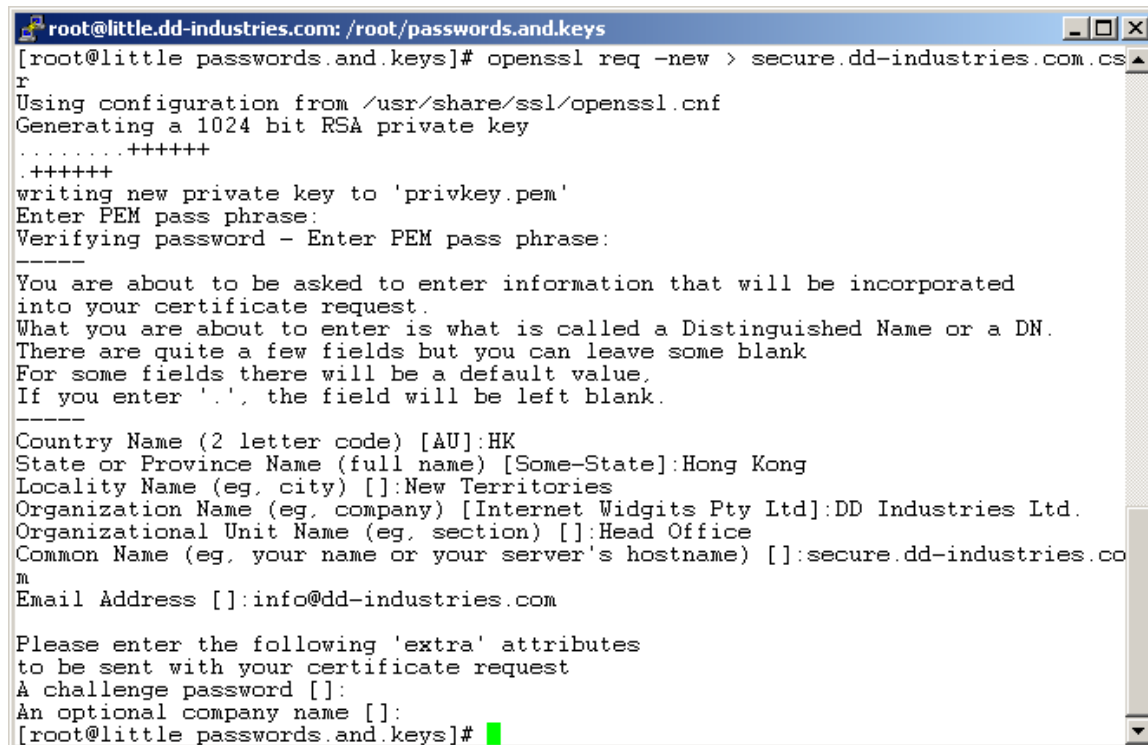
```
root@little.dd-industries.com: /root/passwords.and.keys
[root@little /]# cd ~/passwords.and.keys/
[root@little passwords.and.keys]# pwd
/root/passwords.and.keys
[root@little passwords.and.keys]#
```

In the above example the **cd** and **pwd** commands were used to navigate to the **/root/passwords.and.keys** directory.

How to Generate A Certificate Signing Request File

The first step in obtaining an **X.509** certificate is to create a file that contains the information necessary to generate the certificate. The **csr** file contains information about the organization requesting the certificate.

Consider the following:



```
root@little.dd-industries.com: /root/passwords.and.keys
[root@little passwords.and.keys]# openssl req -new > secure.dd-industries.com.csr
Using configuration from /usr/share/ssl/openssl.cnf
Generating a 1024 bit RSA private key
.....++++++
++++++
writing new private key to 'privkey.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]:HK
State or Province Name (full name) [Some-State]:Hong Kong
Locality Name (eg, city) []:New Territories
Organization Name (eg, company) [Internet Widgits Pty Ltd]:DD Industries Ltd.
Organizational Unit Name (eg, section) []:Head Office
Common Name (eg, your name or your server's hostname) []:secure.dd-industries.com
Email Address []:info@dd-industries.com

Please enter the following 'extra' attributes
to be sent with your certificate request
A challenge password []:
An optional company name []:
[root@little passwords.and.keys]#
```

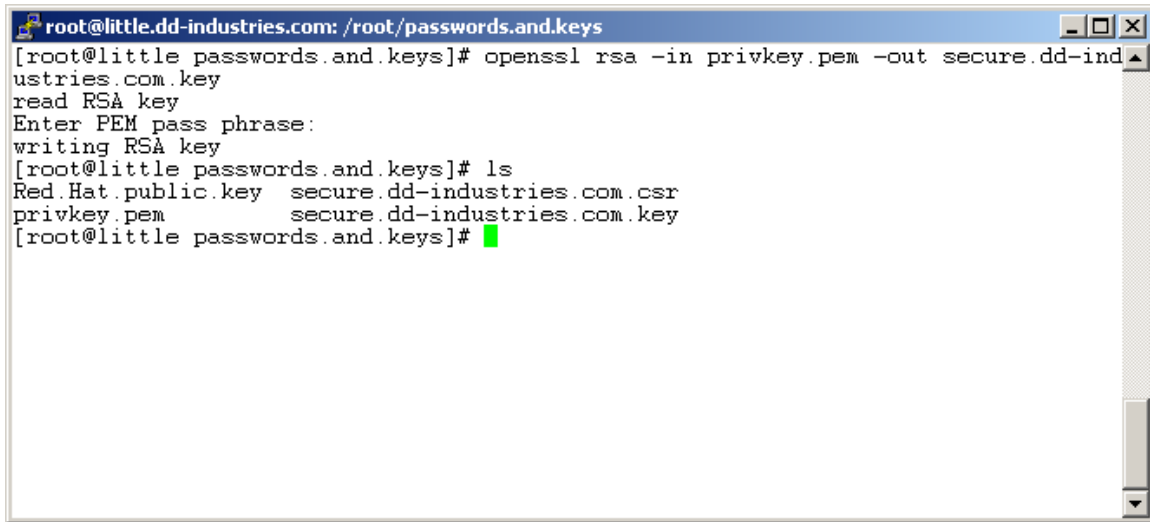
In the above example the **secure.dd-industries.com.csr** file was generated with the **openssl** command. This file contains the information required to create the **X.509** server certificate.

How To Remove the Passphrase From The Private Key

In some instances it may not be desirable to have a passphrase on the private key. In this case, the passphrase would be requested each time the web server process is re-started. This can cause problems when your web server is physically remote.

Many people remove the passphrase from the private key – remember, this can open a **big** security issue that must then be managed. Choose your strategy appropriately.

Consider the following:

A terminal window titled 'root@little.dd-industries.com: /root/passwords.and.keys'. The user runs the command 'openssl rsa -in privkey.pem -out secure.dd-industries.com.key'. The terminal shows the process of reading the RSA key, entering the PEM passphrase, and writing the new key. After the command, the user runs 'ls' to show the files in the directory: 'Red.Hat.public.key', 'secure.dd-industries.com.csr', 'privkey.pem', and 'secure.dd-industries.com.key'. The prompt returns to the user.

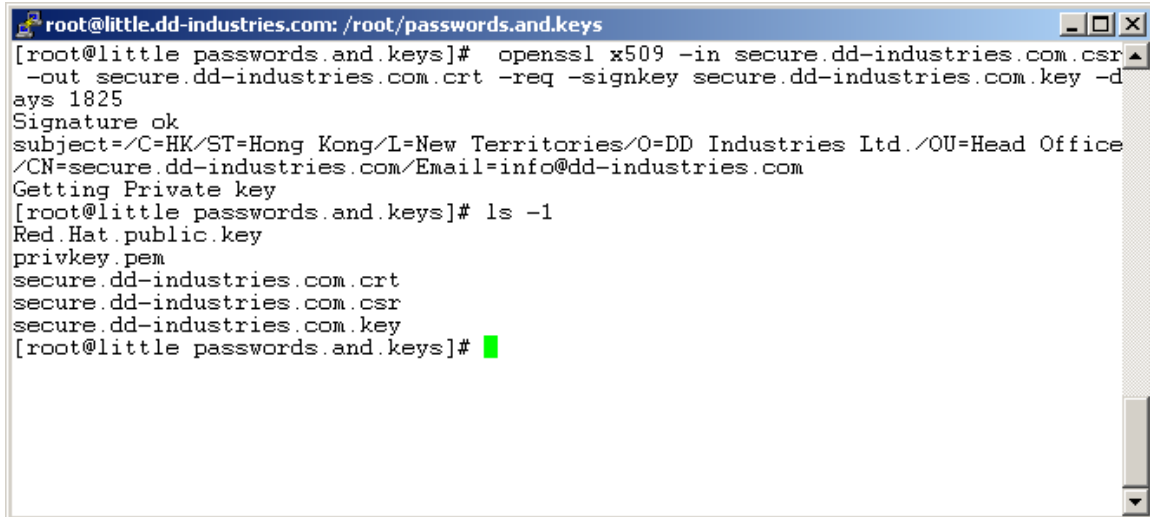
```
root@little.dd-industries.com: /root/passwords.and.keys
[root@little passwords.and.keys]# openssl rsa -in privkey.pem -out secure.dd-industries.com.key
read RSA key
Enter PEM pass phrase:
writing RSA key
[root@little passwords.and.keys]# ls
Red.Hat.public.key  secure.dd-industries.com.csr
privkey.pem         secure.dd-industries.com.key
[root@little passwords.and.keys]#
```

In the above example the passphrase was stripped from the file **privkey.pem** to derive a passphrase-free key file, **secure.dd-industries.com.key**.

How To Create The X.509 Certificate

Now that the **csr** has been prepared and a passphrase-free private key is available, it is time to perform the final step in terms of creating the files necessary to support **X.509** certificates.

Consider the following:

A terminal window titled 'root@little.dd-industries.com: /root/passwords.and.keys'. The prompt is '[root@little passwords.and.keys]#'. The user enters the command 'openssl x509 -in secure.dd-industries.com.csr -out secure.dd-industries.com.crt -req -signkey secure.dd-industries.com.key -days 1825'. The output shows 'Signature ok' and the certificate details: 'subject=/C=HK/ST=Hong Kong/L=New Territories/O=DD Industries Ltd./OU=Head Office/CN=secure.dd-industries.com/Email=info@dd-industries.com'. The prompt changes to '[root@little passwords.and.keys]#'. The user enters 'ls -l'. The output lists: 'Red.Hat.public.key', 'privkey.pem', 'secure.dd-industries.com.crt', 'secure.dd-industries.com.csr', and 'secure.dd-industries.com.key'. The prompt returns to '[root@little passwords.and.keys]#' with a green cursor.

```
root@little.dd-industries.com: /root/passwords.and.keys
[root@little passwords.and.keys]# openssl x509 -in secure.dd-industries.com.csr
-out secure.dd-industries.com.crt -req -signkey secure.dd-industries.com.key -d
ays 1825
Signature ok
subject=/C=HK/ST=Hong Kong/L=New Territories/O=DD Industries Ltd./OU=Head Office
/CN=secure.dd-industries.com/Email=info@dd-industries.com
Getting Private key
[root@little passwords.and.keys]# ls -l
Red.Hat.public.key
privkey.pem
secure.dd-industries.com.crt
secure.dd-industries.com.csr
secure.dd-industries.com.key
[root@little passwords.and.keys]#
```

In the above example the **openssl** command was used to create an **X.509** certificate contained in the file **secure.dd-industries.com.crt**.

Where To Put The X.509 Certificates And Server Key

Move the certificate and key to the appropriate places

Consider the following:

A terminal window titled 'root@little.dd-industries.com: /root/passwords.and.keys'. The window shows a series of commands and their outputs. First, 'ls -l' lists files in the current directory: 'Red.Hat.public.key', 'privkey.pem', 'secure.dd-industries.com.crt', 'secure.dd-industries.com.csr', and 'secure.dd-industries.com.key'. Then, 'ls /etc/httpd/conf/' lists files in the httpd configuration directory, including 'Makefile', 'httpd.conf', 'magic', 'ssl.crl', 'ssl.csr', 'ssl.prm', 'access.conf', 'httpd.conf.rpmnew', 'srn.conf', 'ssl.crt', and 'ssl.key'. Next, 'mv secure.dd-industries.com.key /etc/httpd/conf/ssl.key' moves the private key. Then, 'mv secure.dd-industries.com.crt /etc/httpd/conf/ssl.crt' moves the certificate. Finally, 'ls -l' is run again, showing the remaining files in the current directory: 'Red.Hat.public.key', 'privkey.pem', and 'secure.dd-industries.com.csr'. The prompt is now '[root@little passwords.and.keys]# ' with a green cursor.

```
[root@little.dd-industries.com: /root/passwords.and.keys]# ls -l
Red.Hat.public.key
privkey.pem
secure.dd-industries.com.crt
secure.dd-industries.com.csr
secure.dd-industries.com.key
[root@little passwords.and.keys]# ls /etc/httpd/conf/
Makefile      httpd.conf      magic          ssl.crl  ssl.csr  ssl.prm
access.conf  httpd.conf.rpmnew  srn.conf      ssl.crt  ssl.key
[root@little passwords.and.keys]# mv secure.dd-industries.com.key /etc/httpd/conf/ssl.key
[root@little passwords.and.keys]# mv secure.dd-industries.com.crt /etc/httpd/conf/ssl.crt
[root@little passwords.and.keys]# ls -l
Red.Hat.public.key
privkey.pem
secure.dd-industries.com.csr
[root@little passwords.and.keys]#
```

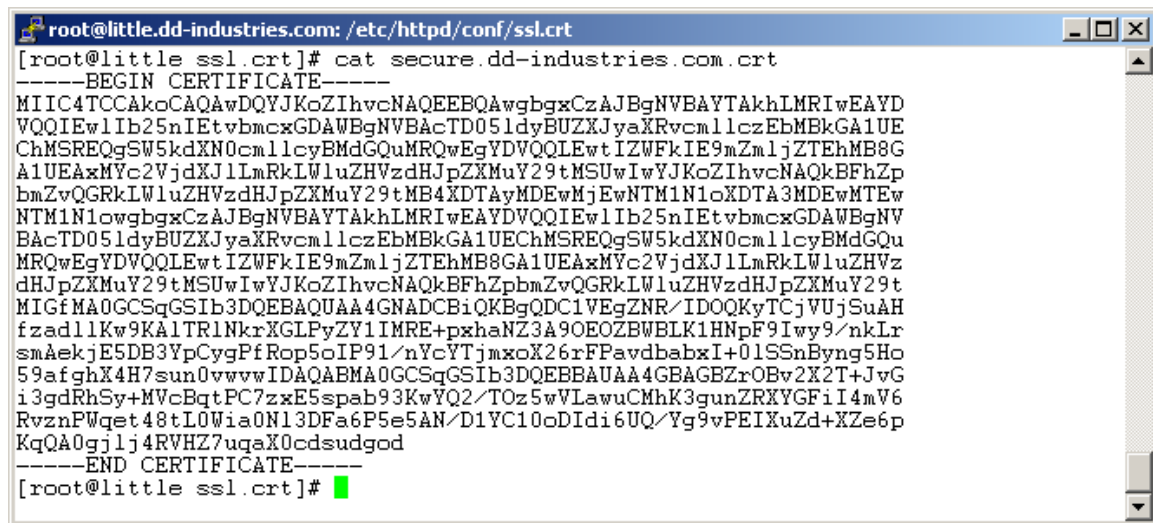
In the above example,

OK, now the certificate and the key are in the appropriate places.

Time to do some investigating!

What The Inside Of An X.509 Certificate Looks Like

Consider the following:



```
root@little.dd-industries.com: /etc/httpd/conf/ssl.crt
[root@little ssl.crt]# cat secure.dd-industries.com.crt
-----BEGIN CERTIFICATE-----
MIIC4TCCAkoCAQAwDQYJKoZIhvcNAQEEBQAwgBgxCzAJBgNVBAYTAkhLMRIwEAYD
VQOIEwlib25nIETvbmcxGDAWBgNVBACTD05ldyBUZXJyaXRvcml1czEbmBkGA1UE
ChMSREQgSW5kdXN0cmllcyBmdGQuMRQwEgYDVQOLEwtIZWFKIE9mZmljZTEhMB8G
A1UEAxMYc2VjdXJlLmRkLWluZHVzdHJpZXMuy29tMSUwIwYJKoZIhvcNAQkBFhZp
bmZvQGRkLWluZHVzdHJpZXMuy29tMB4XDTAyMDEwMjEwNTM1N1oXDTA3MDEwMTEw
NTM1N1owgBgxCzAJBgNVBAYTAkhLMRIwEAYDVQOIEwlib25nIETvbmcxGDAWBgNV
BACTD05ldyBUZXJyaXRvcml1czEbmBkGA1UEChMSREQgSW5kdXN0cmllcyBmdGQu
MRQwEgYDVQOLEwtIZWFKIE9mZmljZTEhMB8GA1UEAxMYc2VjdXJlLmRkLWluZHVz
dHJpZXMuy29tMSUwIwYJKoZIhvcNAQkBFhZpbmZvQGRkLWluZHVzdHJpZXMuy29t
MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQDC1VEgZNR/IDOQKyTCjVUjSuAH
fzadllKw9KA1TR1NkrXGLPyZY1IMRE+pxhaNZ3A9OEOZBWBLK1HNpF9Iwy9/nkLr
smAekjE5DB3YpCygPfRop5oIP91/nYcYTjmxcX26rFPavdbabxI+0lSSnByng5Ho
59afghX4H7sun0vwwwIDAQABMA0GCSqGSIb3DQEBAUAA4GBAGBZrOBv2X2T+JvG
i3gdRhSy+MVCBqtPC7zxESpab93KwYQ2/TOz5wVLawuCMhK3gunZRXYGFiI4mV6
RvznPWqet48tL0Wia0N13Dfa6P5e5AN/D1VC10oDIdi6UQ/Yg9vPEIXuZd+XZe6p
KqQA0gjlj4RVHZ7uqaX0cdsudgod
-----END CERTIFICATE-----
[root@little ssl.crt]#
```

In the above example,

What The Inside Of A Decoded Certificate Looks Like

Consider the following:

```
root@little.dd-industries.com: /etc/httpd/conf/ssl.crt
[root@little ssl.crt]# openssl x509 -noout -text -in secure.dd-industries.com.crt
Certificate:
  Data:
    Version: 1 (0x0)
    Serial Number: 0 (0x0)
    Signature Algorithm: md5WithRSAEncryption
    Issuer: C=HK, ST=Hong Kong, L=New Territories, O=DD Industries Ltd., OU=
Head Office, CN=secure.dd-industries.com/Email=info@dd-industries.com
    Validity
      Not Before: Jan  2 10:53:57 2002 GMT
      Not After : Jan  1 10:53:57 2007 GMT
    Subject: C=HK, ST=Hong Kong, L=New Territories, O=DD Industries Ltd., OU=
=Head Office, CN=secure.dd-industries.com/Email=info@dd-industries.com
    Subject Public Key Info:
      Public Key Algorithm: rsaEncryption
      RSA Public Key: (1024 bit)
        Modulus (1024 bit):
          00:c2:d5:51:20:64:d4:7f:20:33:90:2b:24:c2:8d:
          55:23:4a:e0:07:7f:36:9d:96:52:b0:f4:a0:25:4d:
          19:4d:92:b5:c6:2c:fc:99:63:52:0c:44:4f:a9:c6:
          16:8d:67:70:3d:38:43:99:05:60:4b:2b:51:cd:a4:
          5f:48:c3:2f:7f:9e:42:eb:b2:60:1e:92:31:39:0c:
          1d:d8:a4:2c:a0:3d:f4:68:a7:9a:08:3f:dd:7f:9d:
          87:18:4e:39:b1:a1:7d:ba:ac:53:da:bd:d6:da:6f:
          12:3e:d2:54:92:9c:1c:a7:83:91:e8:e7:d6:9f:82:
          15:f8:1f:bb:2e:9f:4b:f0:bf
        Exponent: 65537 (0x10001)
      Signature Algorithm: md5WithRSAEncryption
        60:59:ac:e0:6f:d9:7d:93:f8:9b:c6:8b:78:1d:46:14:b2:f8:
        c5:5c:06:ab:4f:0b:bc:f1:13:9b:29:69:bf:77:2b:06:10:db:
        f4:ce:cf:9c:15:2d:ac:2e:08:c8:4a:de:0b:a7:65:15:d8:18:
        58:88:e2:65:7a:46:fc:e7:3d:6a:9e:b7:8f:2d:2f:45:a2:6b:
        43:65:dc:31:5a:e8:fe:5e:e4:03:7f:0f:56:02:d7:4a:03:21:
        d8:ba:51:0f:d8:83:db:cf:10:85:ee:65:df:97:65:ee:a9:2a:
        a4:00:d2:08:e5:8f:84:55:1d:9e:ee:a9:a5:f4:71:db:2e:76:
        0a:1d
[root@little ssl.crt]#
```

In the above example the **openssl x509** command was used to display the decrypted contents of the **secure.dd-industries.com.crt** file.

Enabling The HTTPS Web Server

The Edit the httpd.conf file to include the IP address and name **secure.dd-industries.com**.

Consider the following:



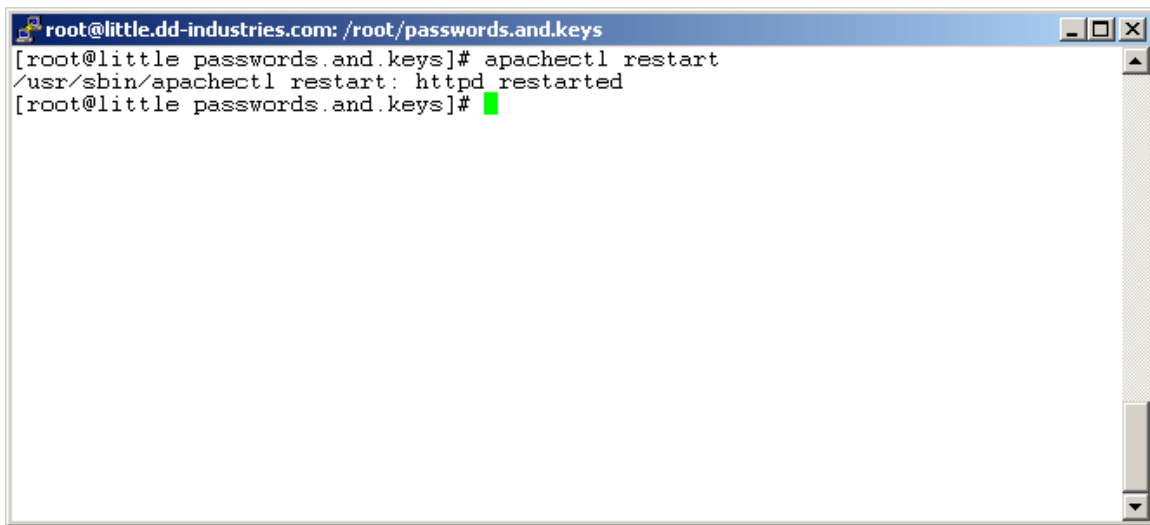
```
root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com

#-----
#
# Customization for SSL functionality
#
# GL      2002-01-02      17h34
#
#-----
Listen 443
<VirtualHost 192.168.0.11:443>
DocumentRoot /home/webs/secure
ServerName secure.dd-industries.com
ServerAdmin root@secure.dd-industries.com
ErrorLog /etc/httpd/logs/error_log
TransferLog /etc/httpd/logs/access_log
SSLEngine on
SSLCertificateFile /etc/httpd/conf/ssl.crt/secure.dd-industries.com.crt
SSLCertificateKeyFile /etc/httpd/conf/ssl.key/secure.dd-industries.com.key
#SSLCACertificateFile /etc/httpd/conf/ssl.crt/ca-bundle.crt
<Files ~ "\.(cgi|shtml)$">
    SSLOptions +StdEnvVars
</Files>
<Directory "/etc/httpd/cgi-bin">
    SSLOptions +StdEnvVars
</Directory>
SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown
CustomLog /etc/httpd/logs/ssl_request_log \
    "%t %h %{SSL_PROTOCOL}x %{SSL_CIPHER}x \"%r\" %b"
</VirtualHost>
"/etc/httpd/conf/httpd.conf" 1230L, 42251C written
```

In the above example,

Restart **httpd** to enable the secure functionality

Consider the following:

A terminal window with a blue title bar containing the text "root@little.dd-industries.com: /root/passwords.and.keys". The terminal shows the command "[root@little passwords.and.keys]# apachectl restart" and its output "/usr/sbin/apachectl restart: httpd restarted". The prompt "[root@little passwords.and.keys]#" is followed by a green cursor.

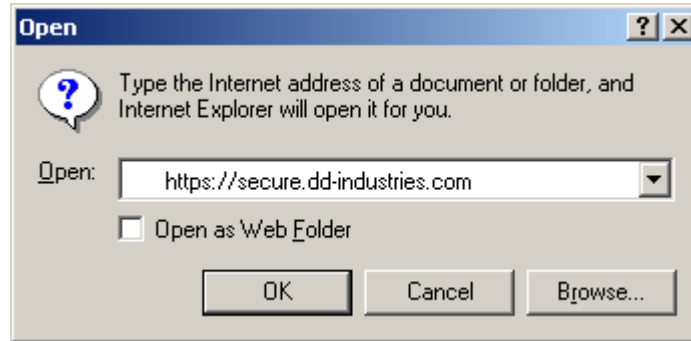
```
root@little.dd-industries.com: /root/passwords.and.keys
[root@little passwords.and.keys]# apachectl restart
/usr/sbin/apachectl restart: httpd restarted
[root@little passwords.and.keys]#
```

In the above example,

Testing the HTTPS Server

Test if **secure.dd-industries.com** responds to HTTPS requests

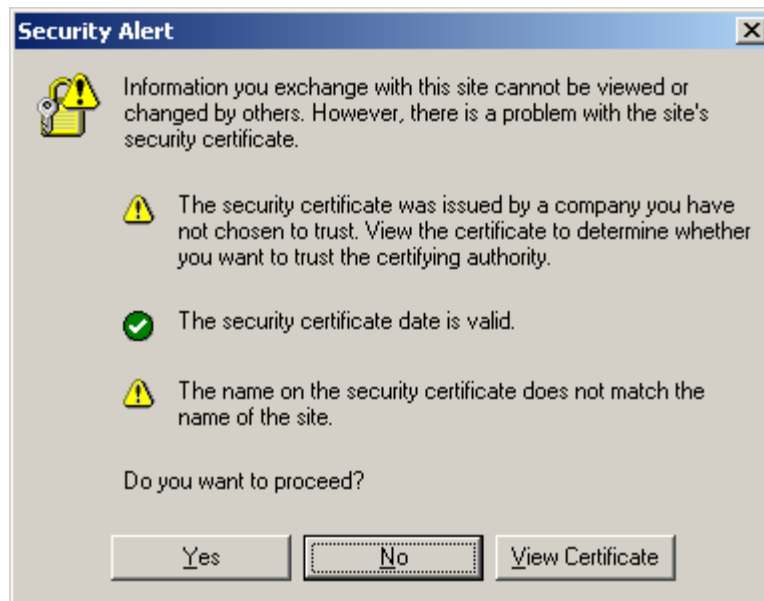
Consider the following:



In the above example,

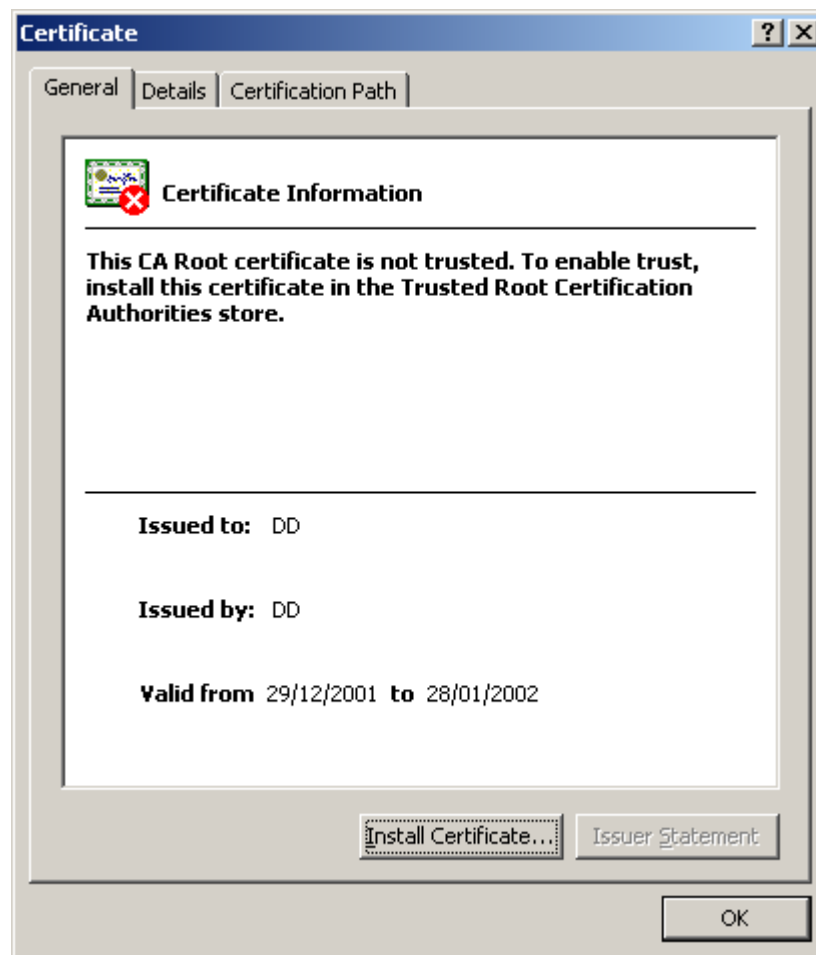
Now the site is available and the web browser and server begin to communicate

Consider the following:



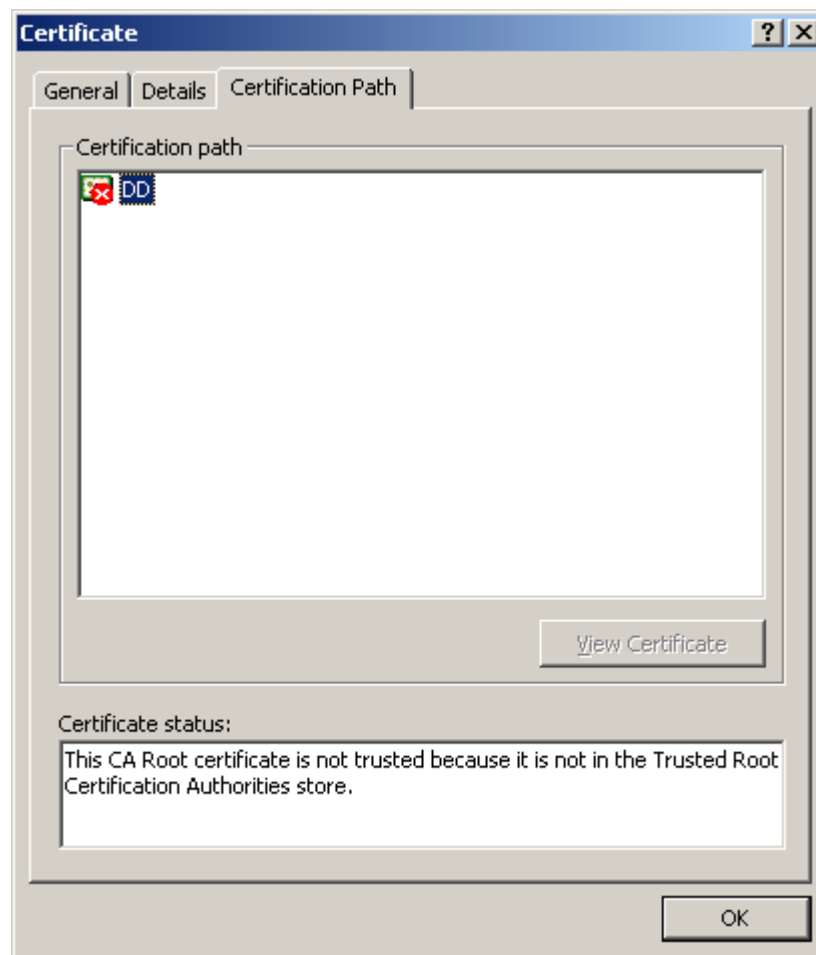
In the above example,

Consider the following:



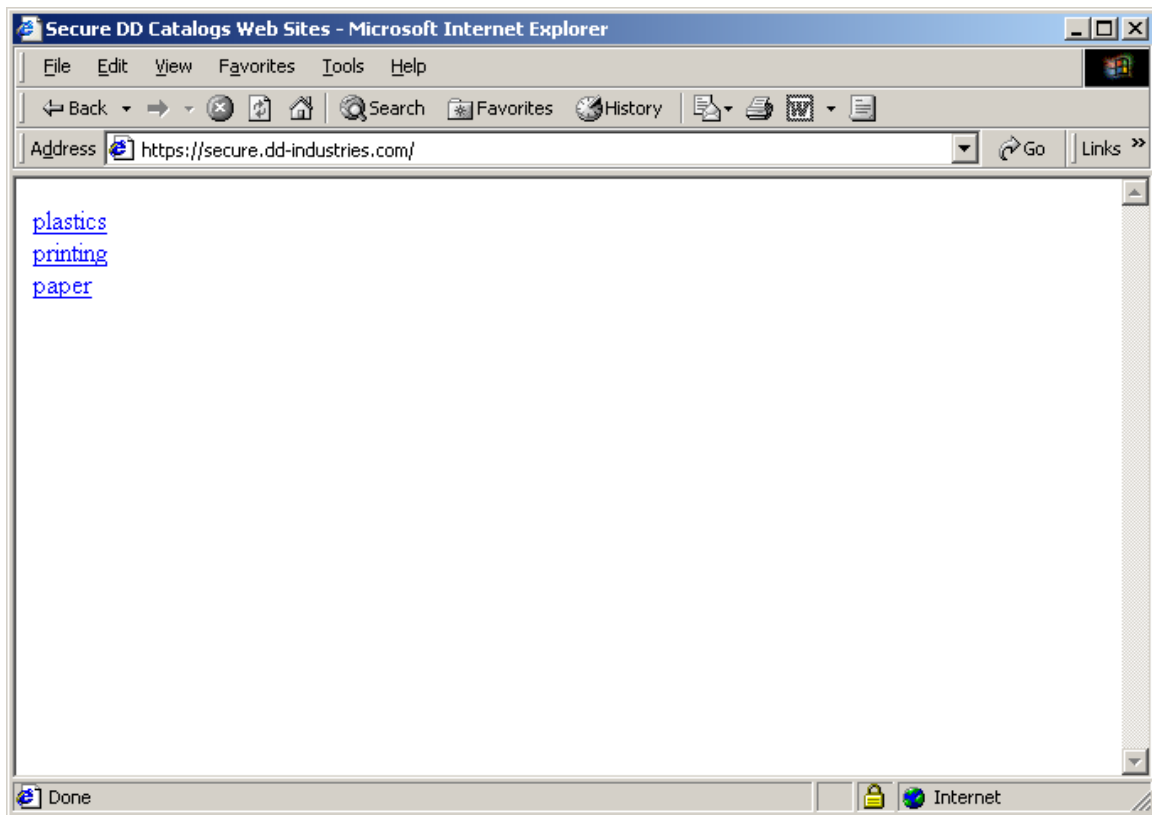
In the above example,

Consider the following:



In the above example,

Consider the following:



In the above example,

SECTION 3: Redirecting HTTP Traffic To The Secure Website

The easiest way to do this is via PHP. Just replace **index.html** with a PHP page called **index.php**. When the web server fails to find **index.html** it will instead load **index.php**, which contains the redirection instructions necessary to enter the SSL website.

Consider the following:

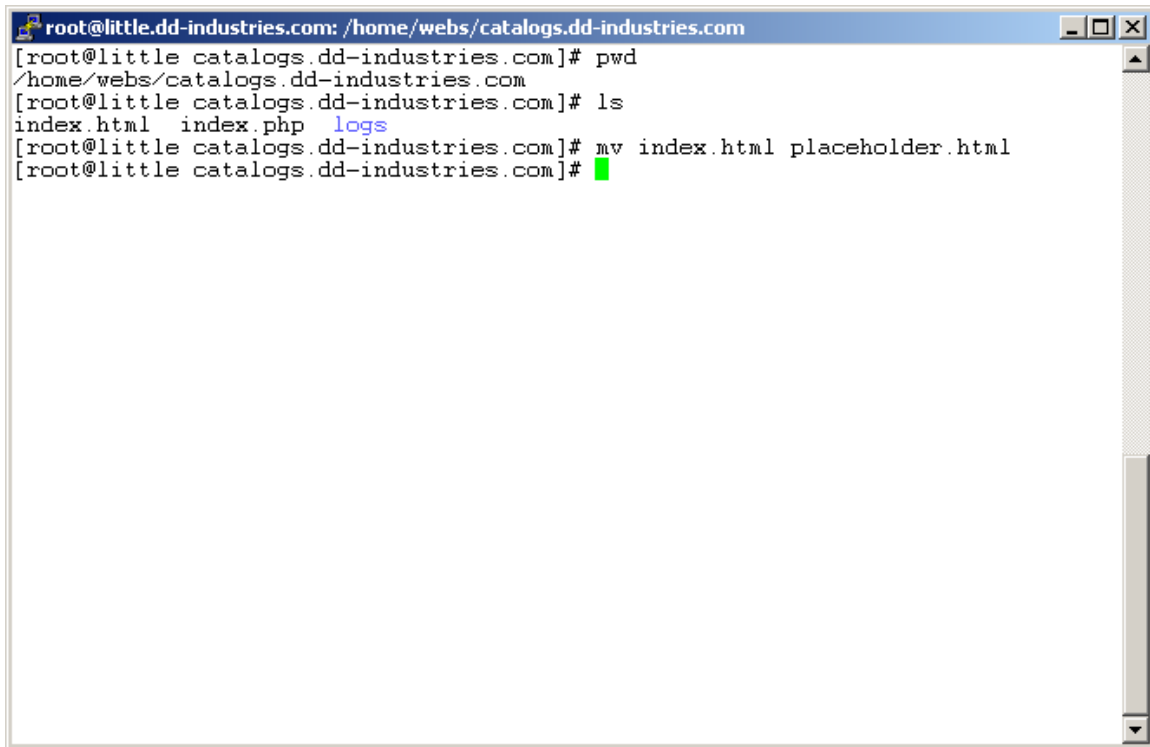
A terminal window with a blue title bar containing the text 'root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com'. The terminal shows a command prompt '[root@little catalogs.dd-industries.com]# cat -> index.php' followed by the command '<?php header("Location: https://secure.dd-industries.com"); ?>' and another prompt '[root@little catalogs.dd-industries.com]#'. A green cursor is visible at the end of the second line.

```
root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com
[root@little catalogs.dd-industries.com]# cat -> index.php
<?php header("Location: https://secure.dd-industries.com"); ?>
[root@little catalogs.dd-industries.com]#
```

In the above example the **index.php** file was created using standard input redirection.

How To Enable HTTPS Redirection

Consider the following:

A terminal window with a blue title bar containing the text 'root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com'. The terminal shows a series of commands and their outputs. The first command is 'pwd', which outputs '/home/webs/catalogs.dd-industries.com'. The second command is 'ls', which outputs 'index.html index.php logs'. The third command is 'mv index.html placeholder.html', which is executed successfully. The prompt returns to the root user at the same directory.

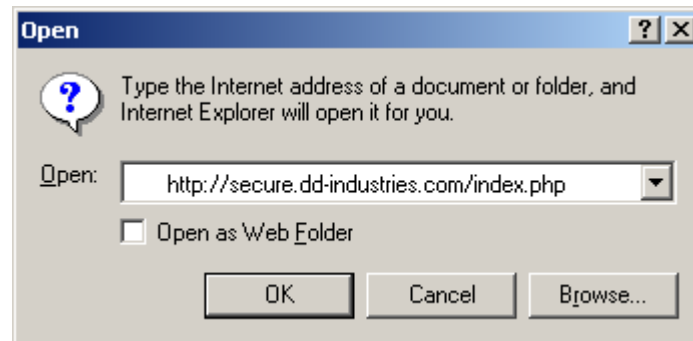
```
root@little.dd-industries.com: /home/webs/catalogs.dd-industries.com
[root@little catalogs.dd-industries.com]# pwd
/home/webs/catalogs.dd-industries.com
[root@little catalogs.dd-industries.com]# ls
index.html index.php logs
[root@little catalogs.dd-industries.com]# mv index.html placeholder.html
[root@little catalogs.dd-industries.com]#
```

In the above example,

This makes the web server load **index.php** instead – effecting the redirection.

How To Test For Correct Redirection

Consider the following:



In the above example, the browser is being directed to load **secure.dd-industries.com/index.php**.

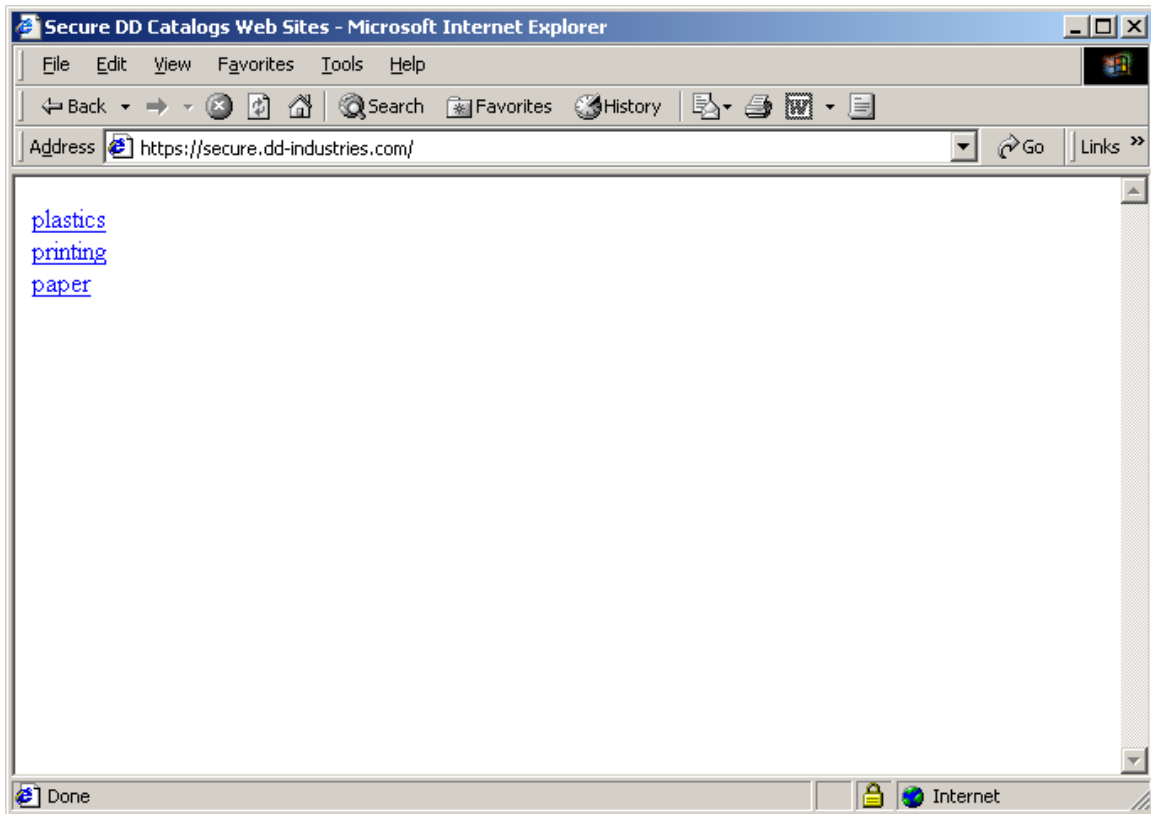
Consider the following:



In the above example,

What This Document Enables

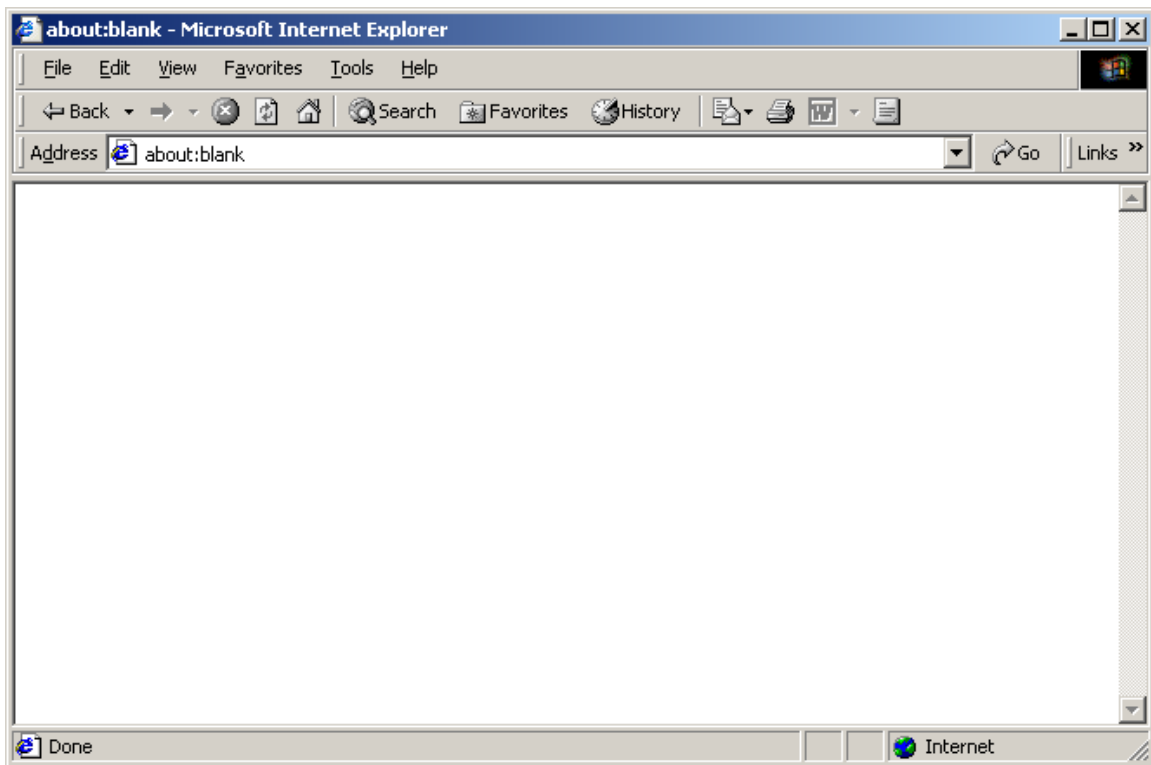
Consider the following:

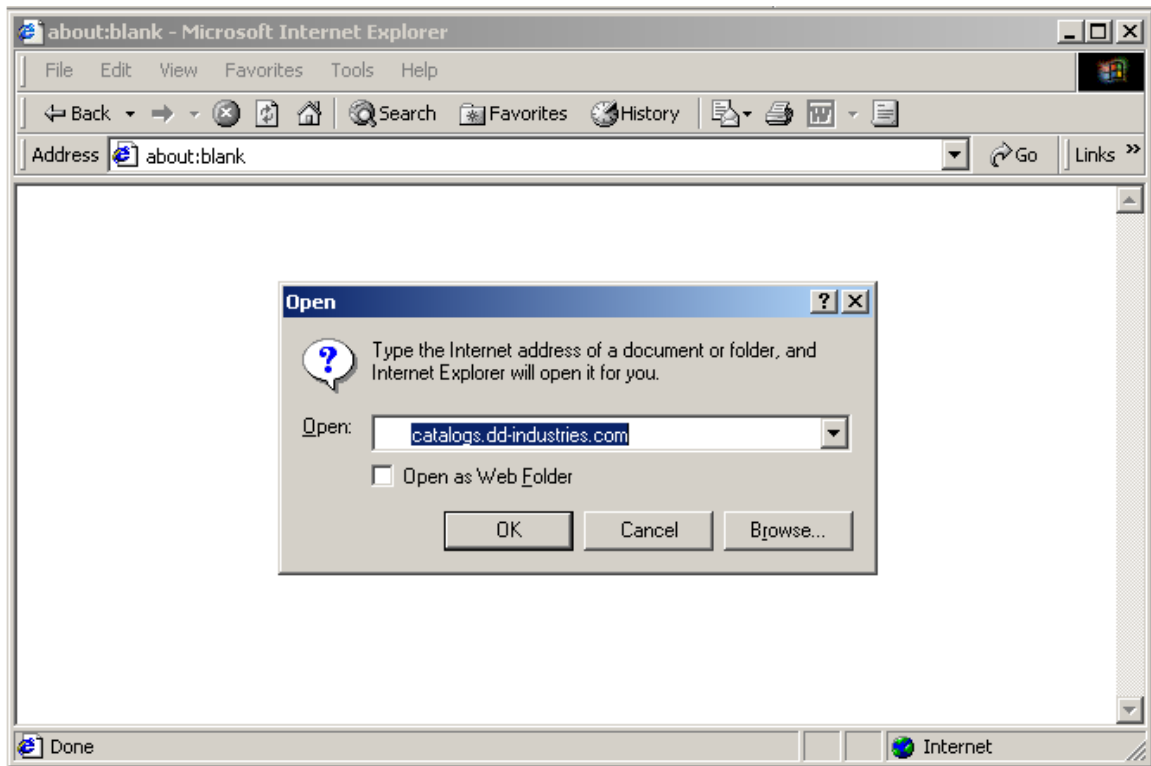


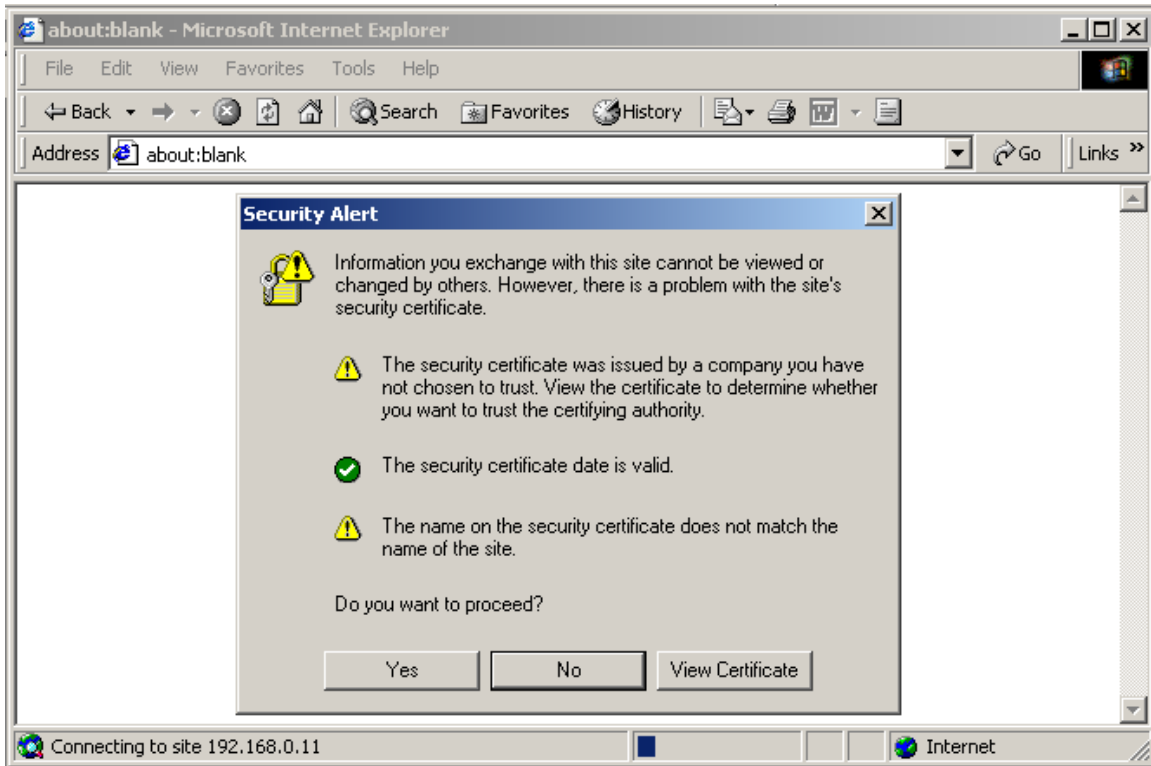
In the above example the web session is now being conducted via HTTPS – indicated in two ways:

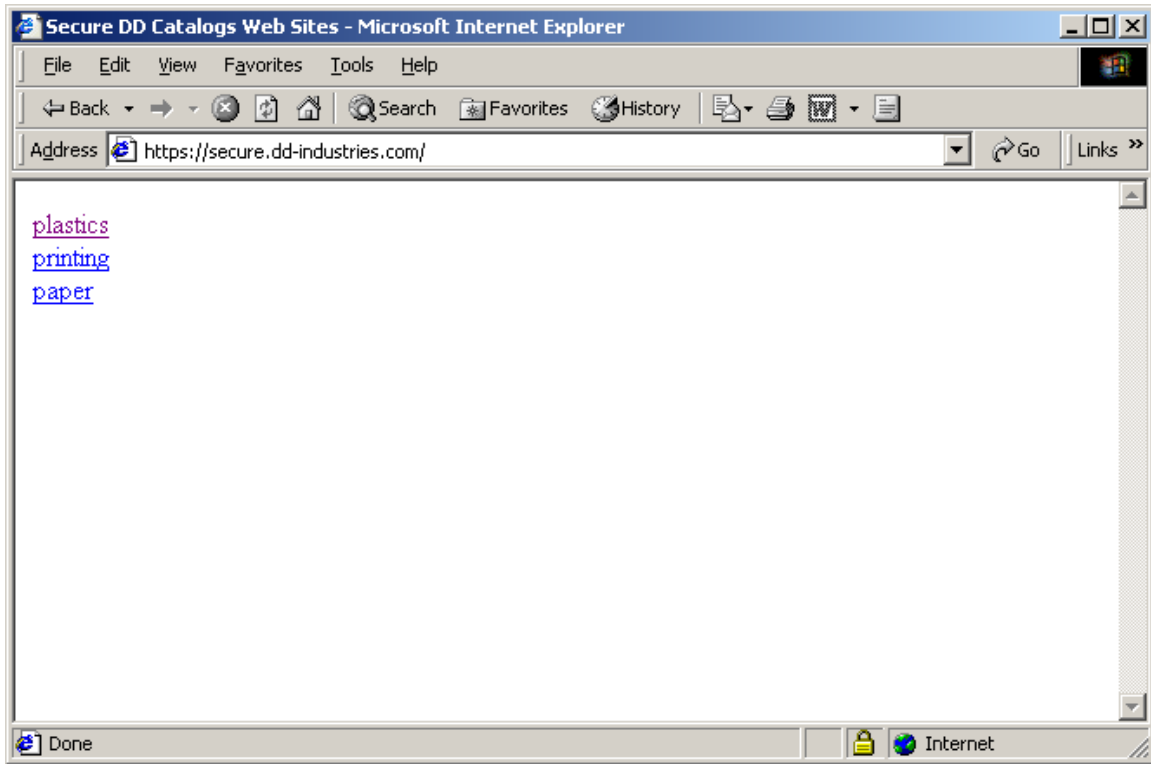
- The URL includes the keyword **https**.
- The lock on the bottom status bar is closed.

Section 4: The Acid Test - Doing It For Real



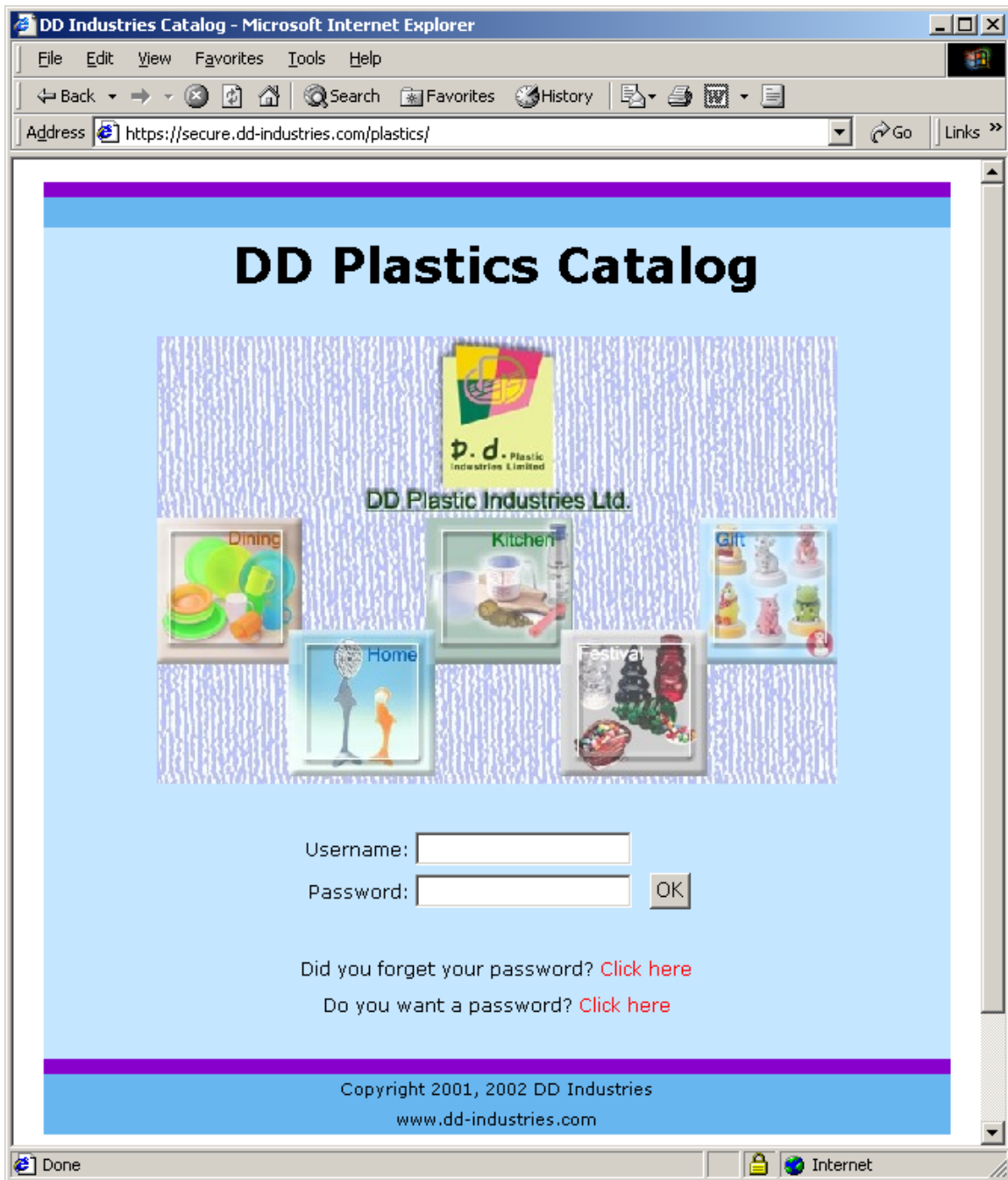






What The Purpose Of This Document Is

Consider the following:



In the above example,

Looks like a secure site to me ☺

14/04/2002

Acknowledgements

<http://www.linuxdoc.org/HOWTO/SSL-RedHat-HOWTO-4.html>

http://www.apache-ssl.org/#Digital_Certificates

<http://www.phpbuilder.com/forum/read.php3?num=2&id=122994&thread=122991>

http://www-zeuthen.desy.de/computing/projects/security/SSL/ssl_commands.html