Configuring a Red Hat Linux System

For Secure Websites

(HTTPS)

Graham Leach

Managing Partner

TIG – The Imperators Group, Inc.

Table Of Contents

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:

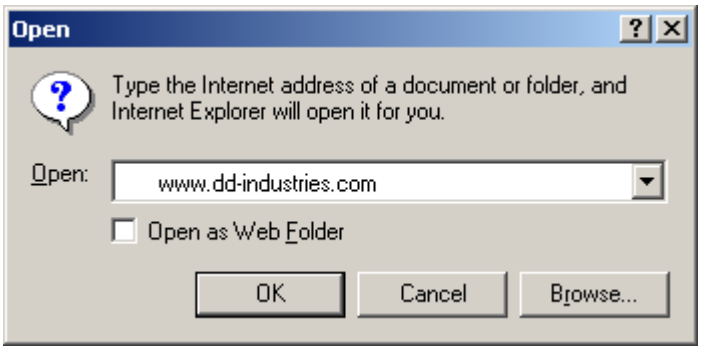
1) A web browser forms a connection with an Internet entity on **port 80**.
2) A web server waiting on **port 80** responds by indicating it is ready for instructions.
3) The web browser issues a request to the web server, usually **GET //index.html**.
4) The web server either returns the **index.html** 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:

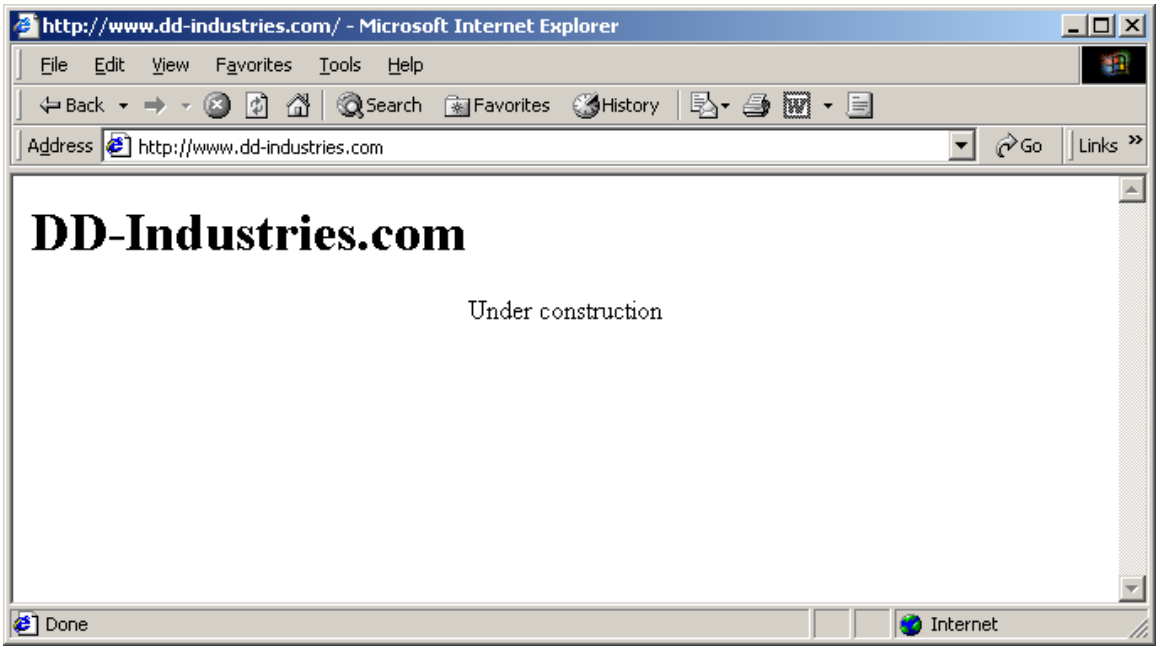


In the above example,

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:

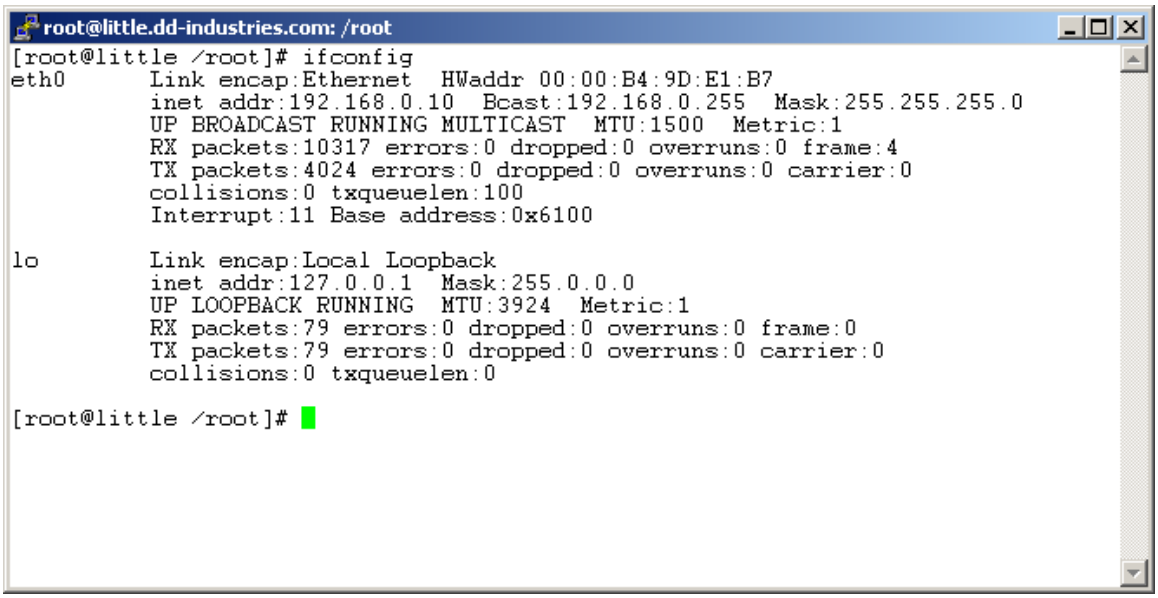
- Prepare for an HTTPS website
- Create a HTTP website & making sure it works
- Create an HTTPS website & making sure it works
- Redirecting traffic from the HTTP website to the HTTPS website

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:



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**.

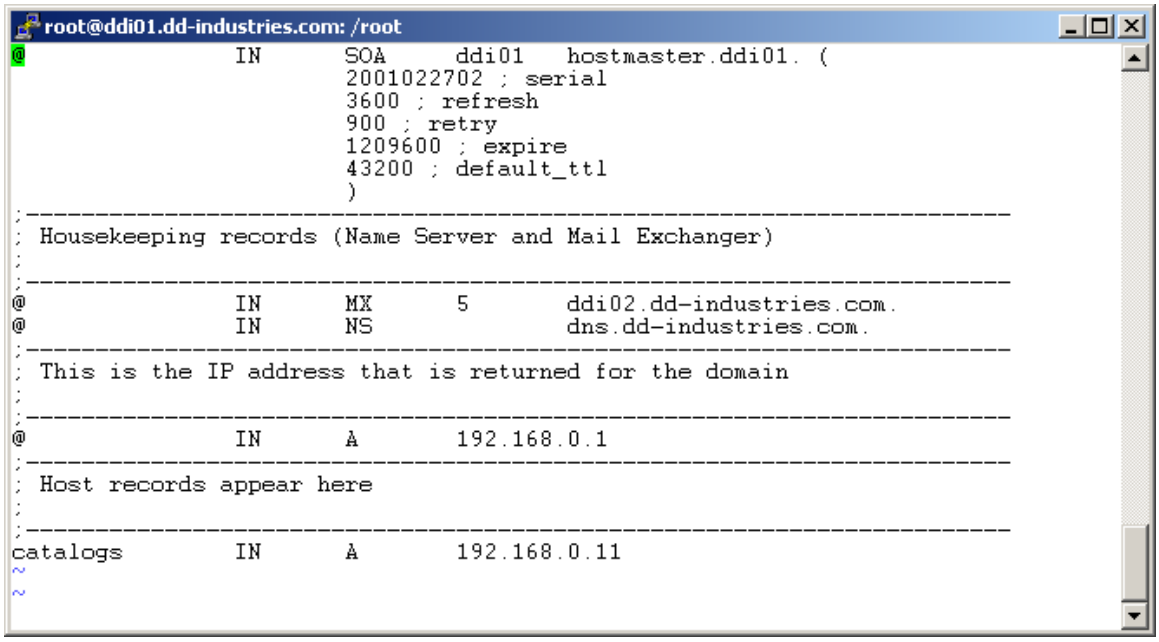
In the case of an Internet entity, your ISP will have to be contacted for a valid IP address.

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:



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**.

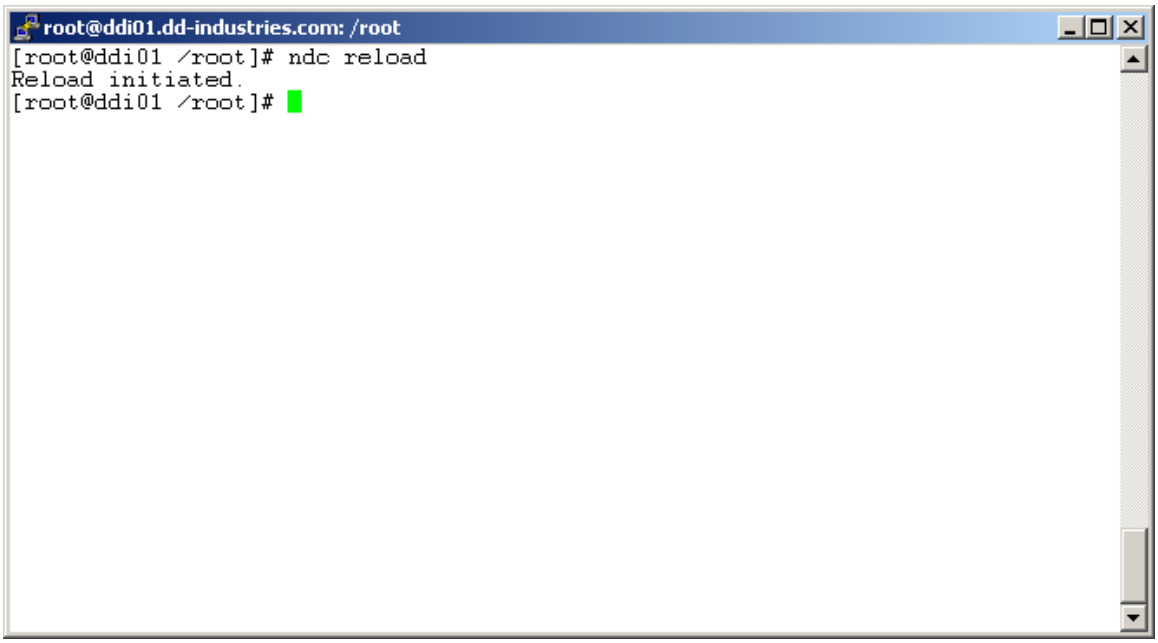
The next step is to reload the DNS database:

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:

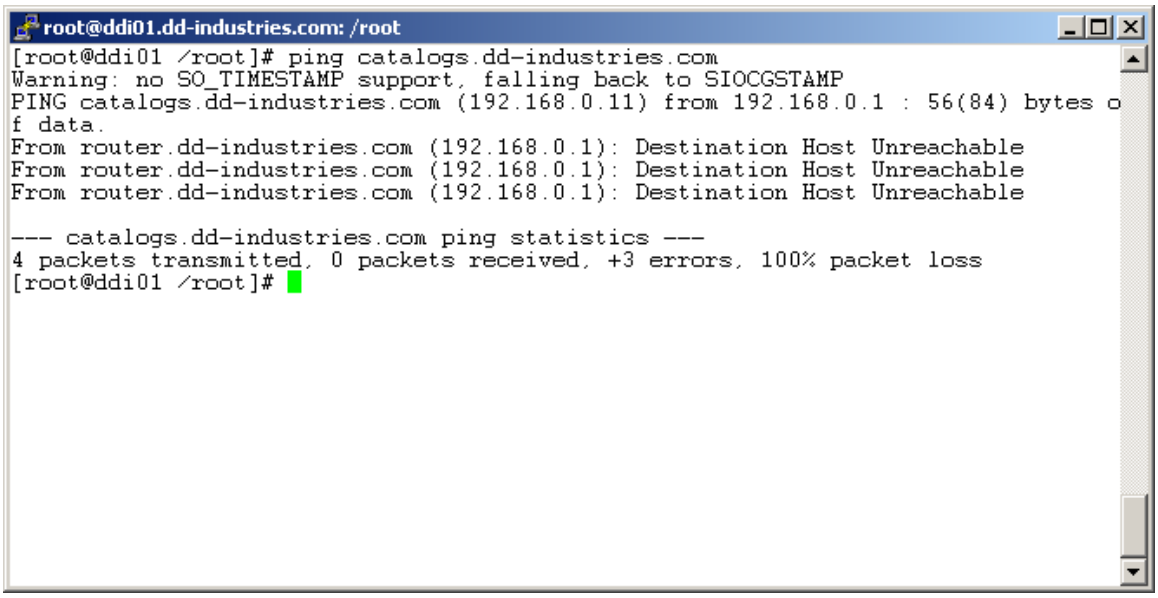


In the above example the **ndc reload** command was used to tell the **named** DNS server to reload its data files.

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:

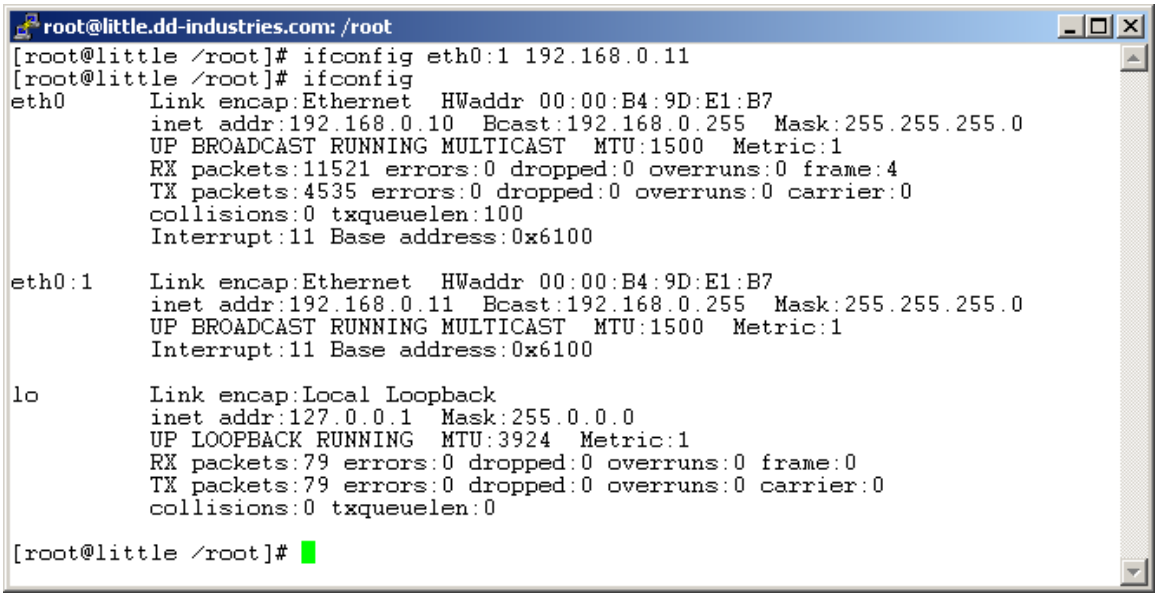


In the above example the **ping** command was used to test both DNS and IP. While the Fully Qualified Domain Name (FQDN) **catalogs.dd-industries.com** resolved properly to the IP address **192.168.0.11**, there was no answer to the **ping** because no machine has yet had the IP address

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.

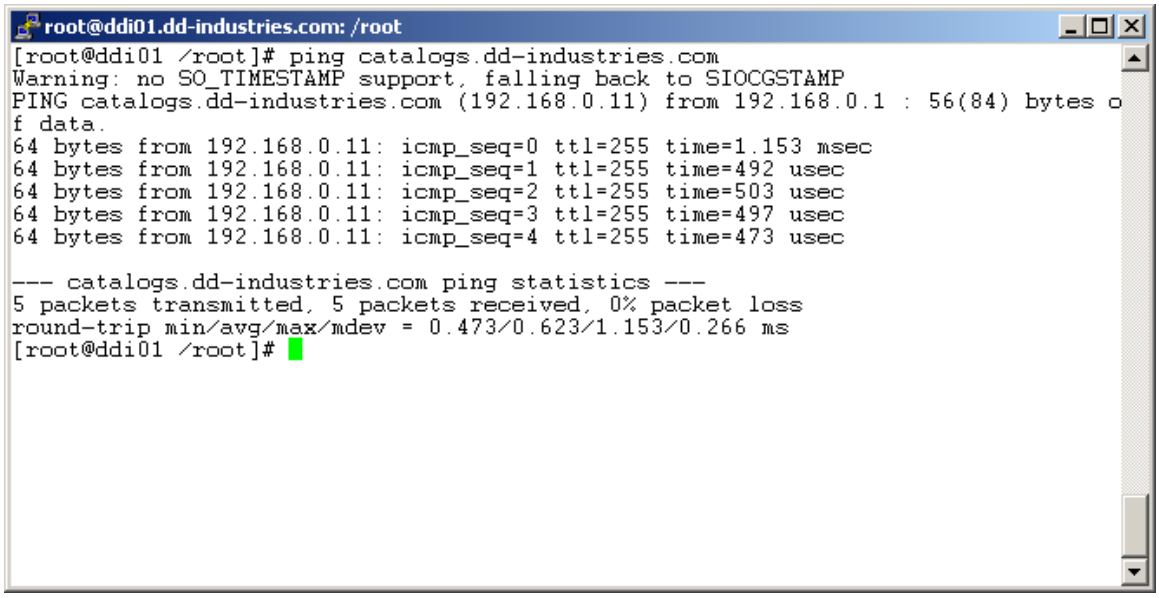
Consider the following:



In the above example,

Testing the Newly Assigned IP Address

Consider the following:



In the above example,

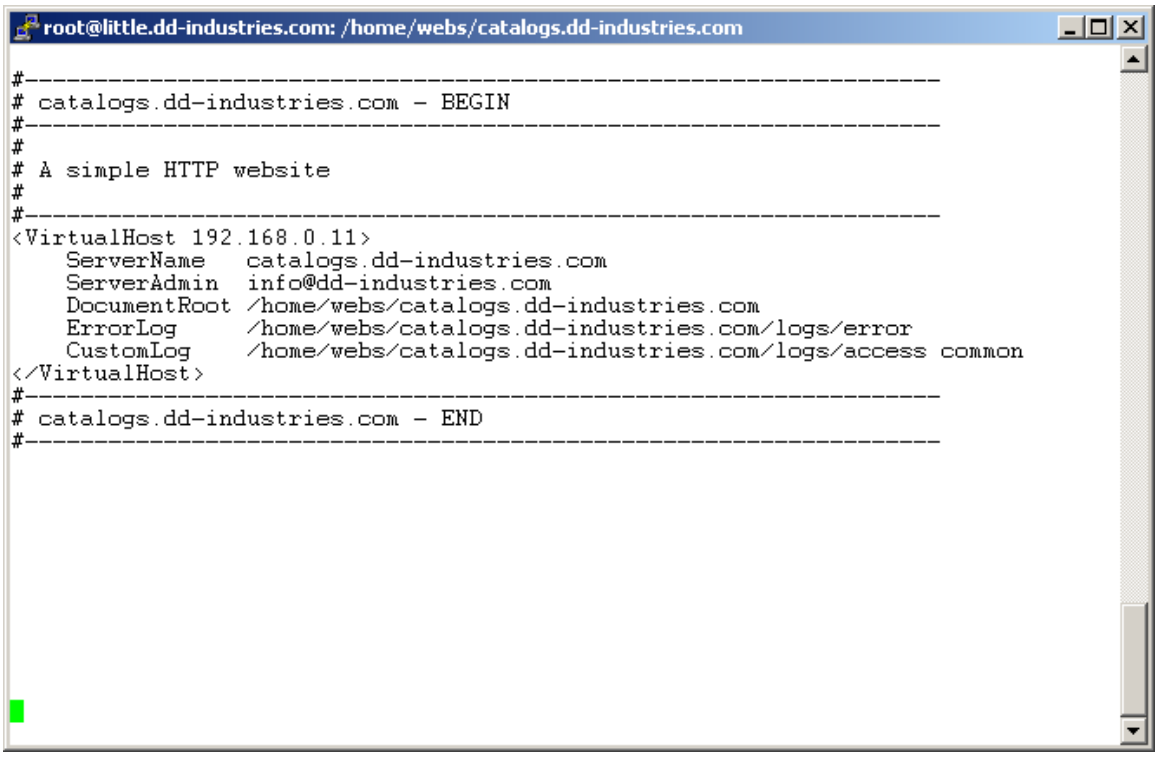
The ping resolves and gets an answer!

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:



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.

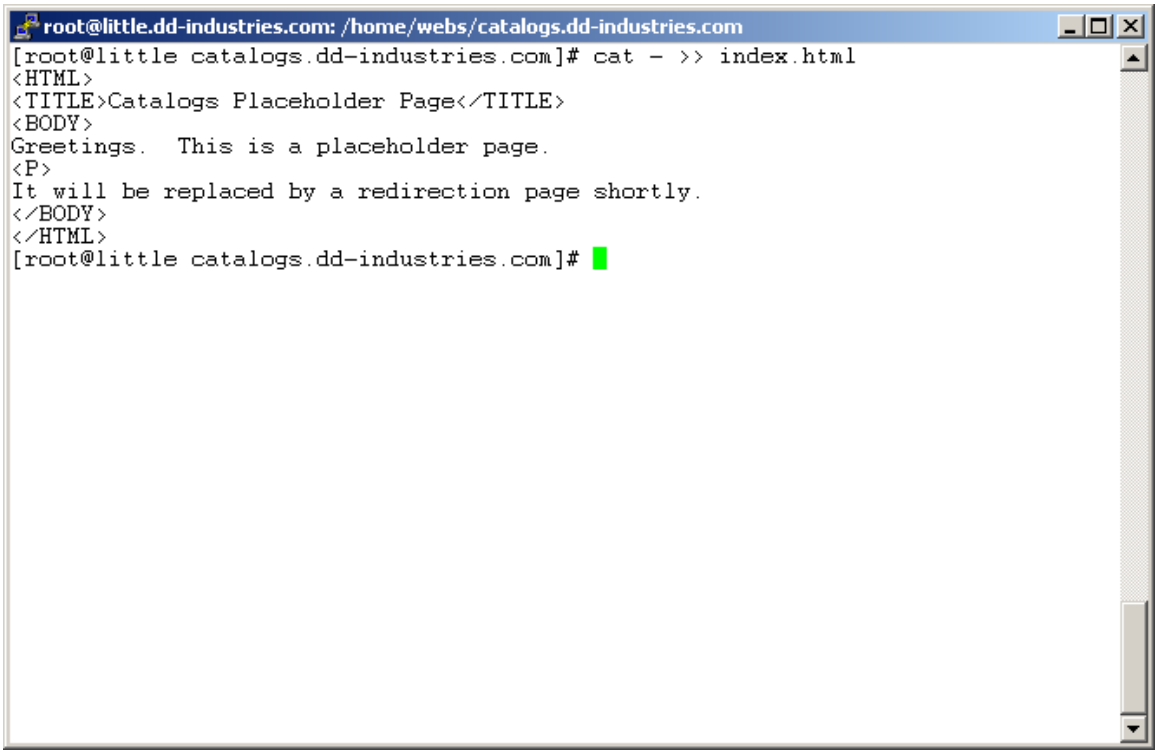
Testing the Web Server

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:

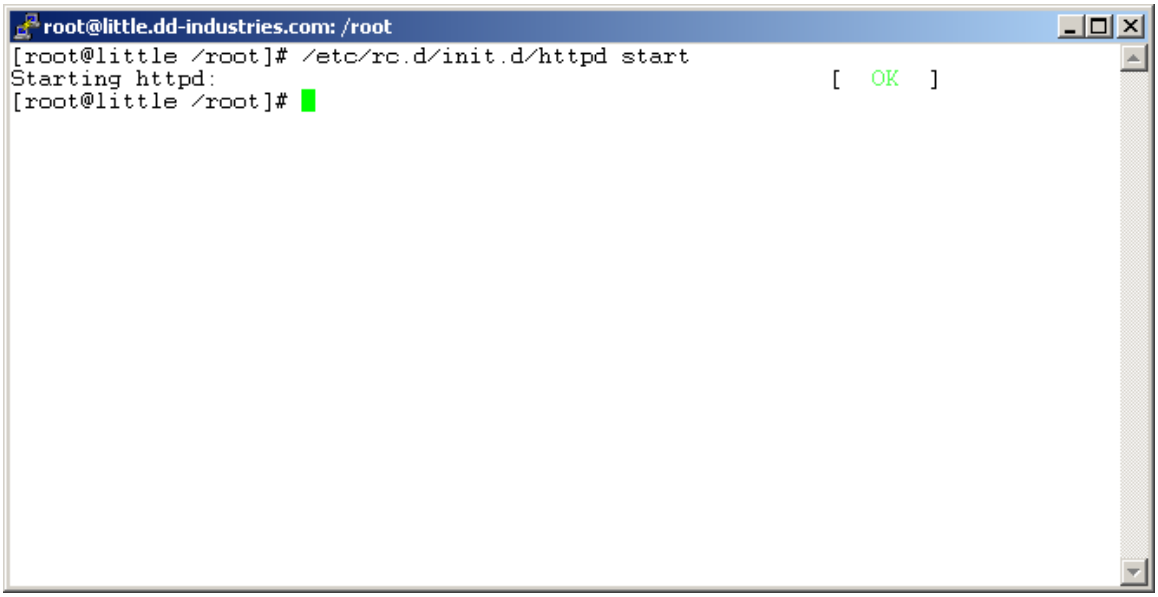


In the above example the **–** character was used to redirect standard input to the file **index.html**. The lines were typed in directly and then **CTRL-D** was used to tell Linux that the file was complete. **CTRL-D** is the file termination character in Unix.

Restarting The Web Server

Use the script to restart the server:

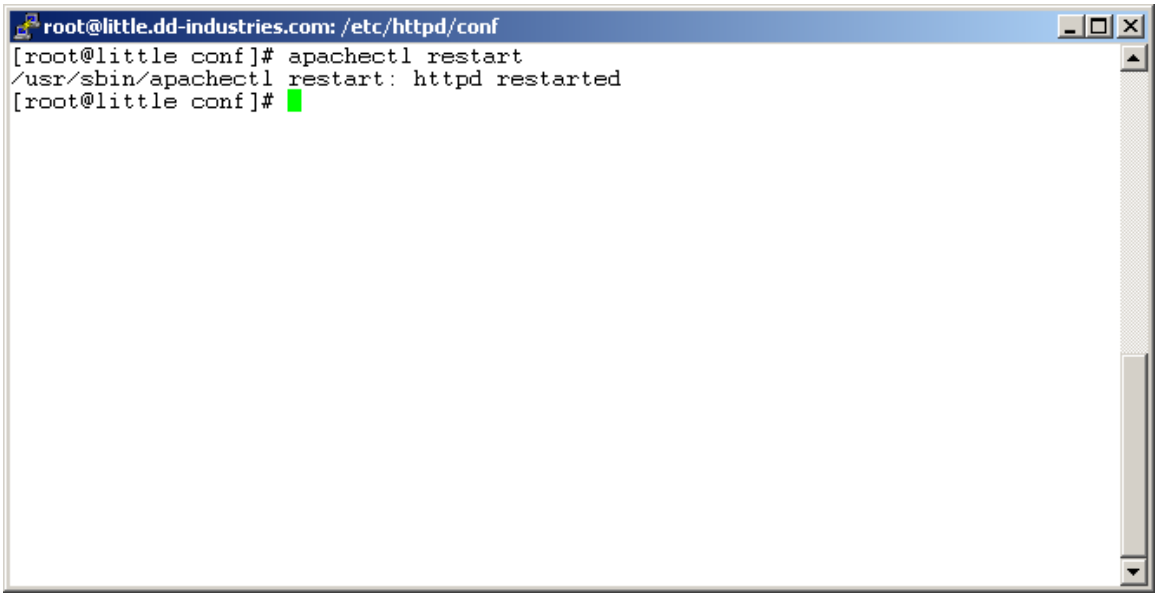
Consider the following:



In the above example,

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

Consider the following:

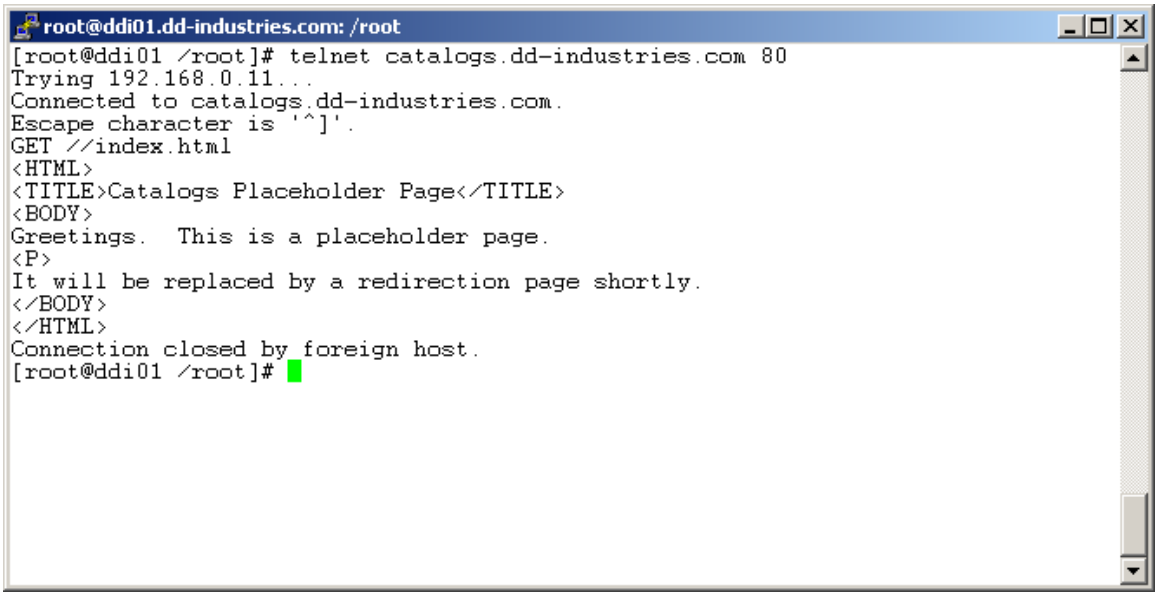


In the above example,

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:

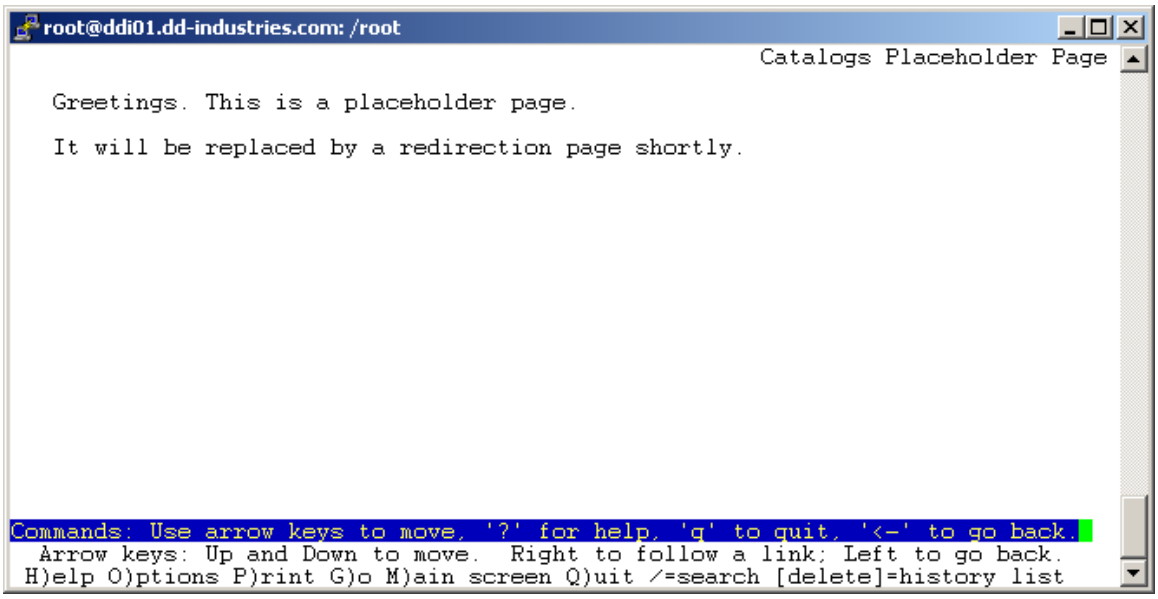


In the above example, a connection to a web server was established on port **80** using the **telnet** command. Once the connection was established, the **GET** command was used to ask for the file **//index.html**, which was immediately returned by the web server.

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

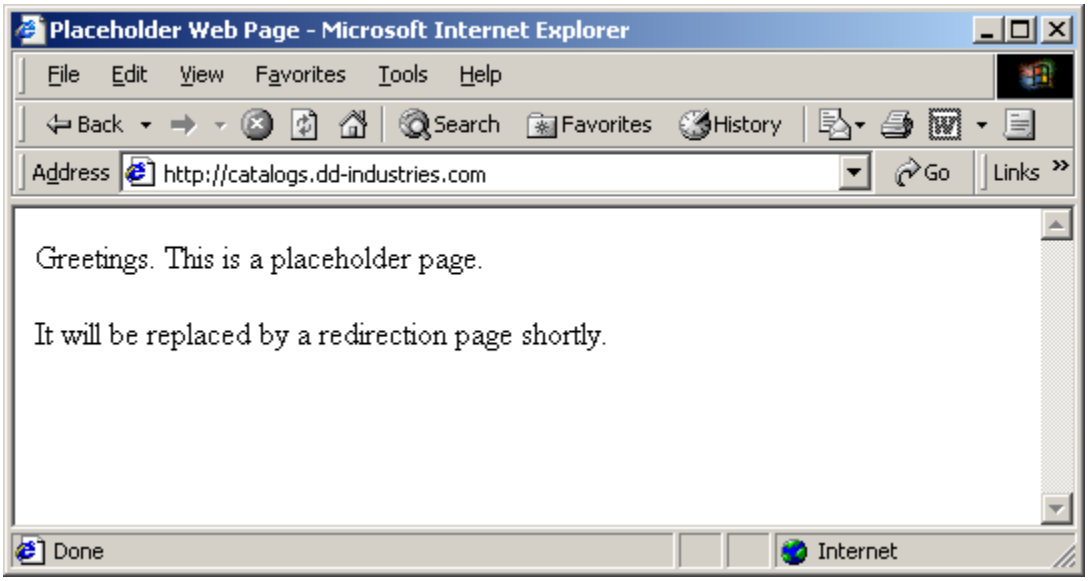
Consider the following:



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:

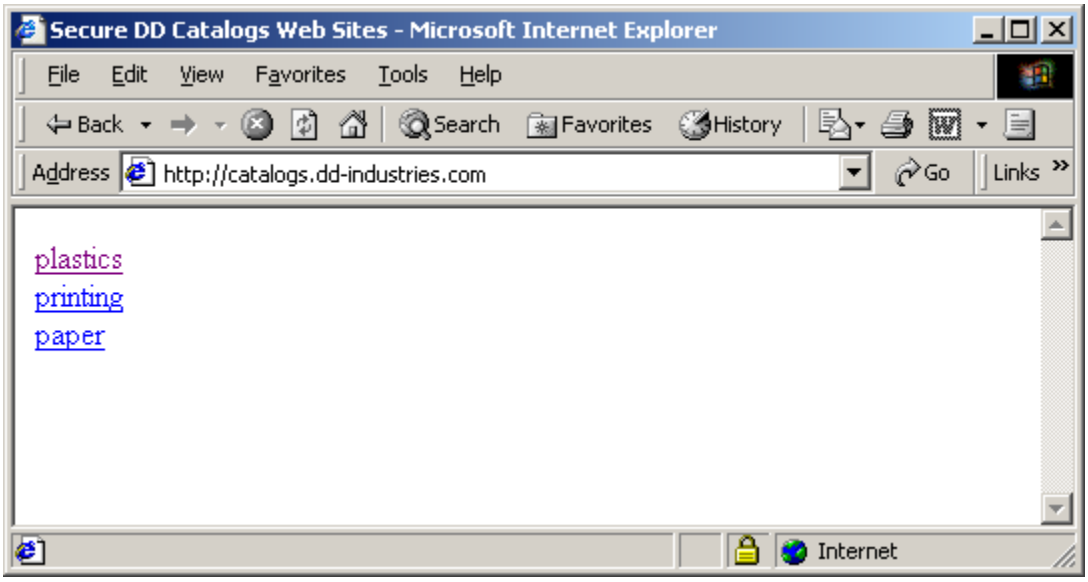


In the above example, Microsoft Internet Explorer was used to verify that the web server configured for **catalogs.dd-industries.com** responded to client requests. As you can see from the lack of a “little lock” on the bottom status bar, this is an insecure connection based on HTTP.

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.

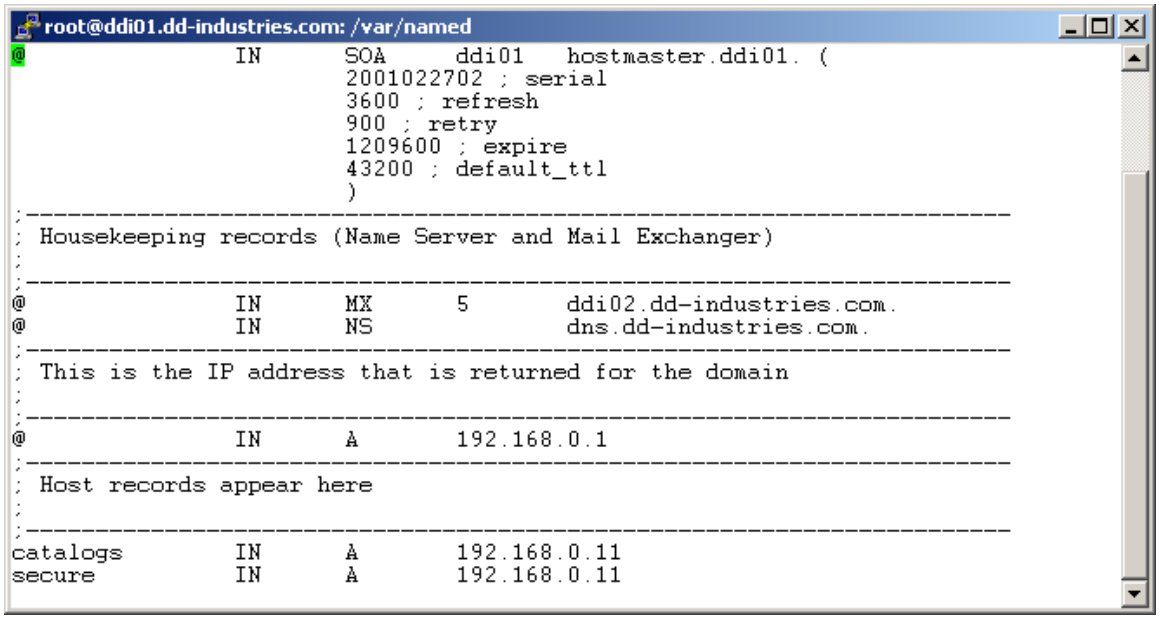
SSL is a complicated topic that revolves around **certificates**, particularly **X.509** certificates. For this example to work, a certificate must be obtained or generated for this server.

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:

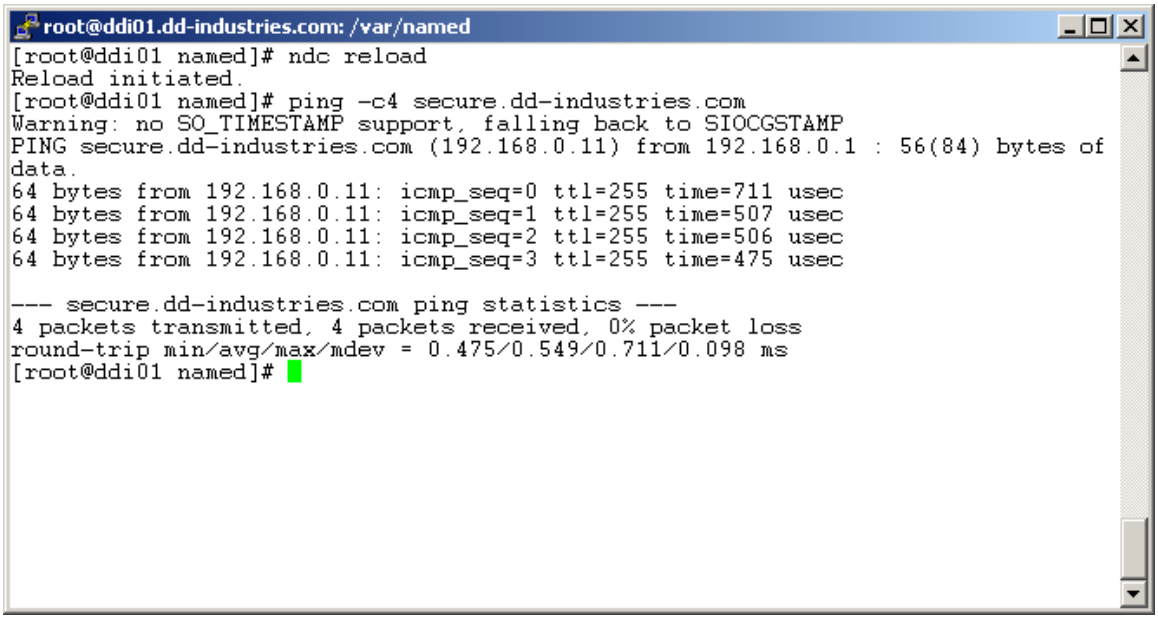


In the above example,

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

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

Consider the following:

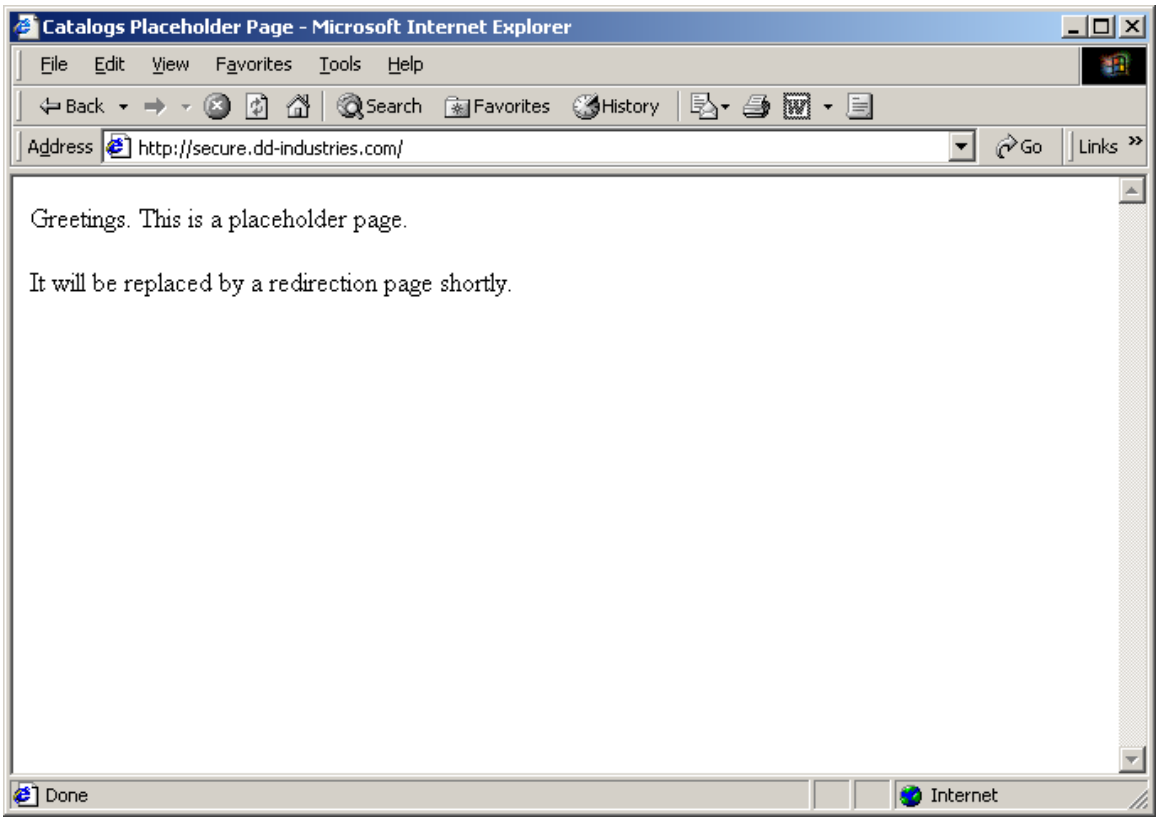


In the above example,

Testing secure.dd-industries.com HTTP Connectivity

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

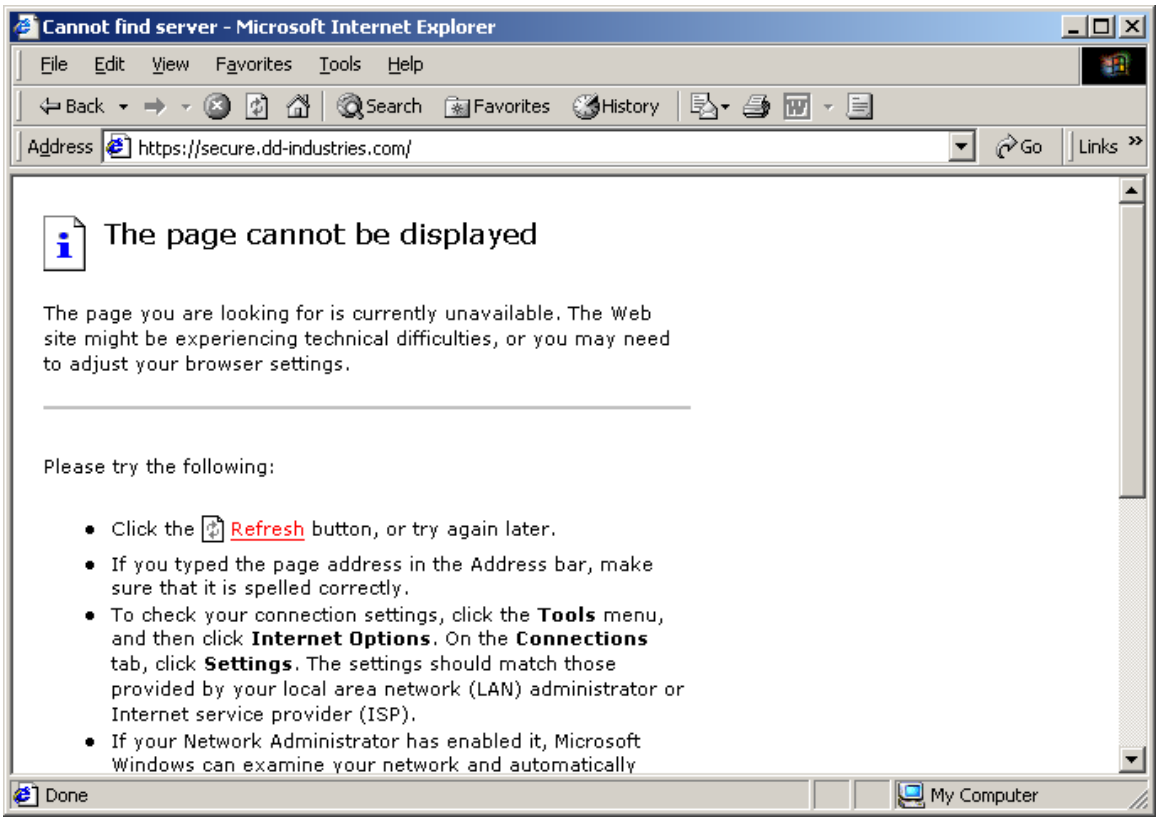
Consider the following:



In the above example,

Testing secure.dd-industries.com HTTPS Connectivity
Without any further configuration this should fail:

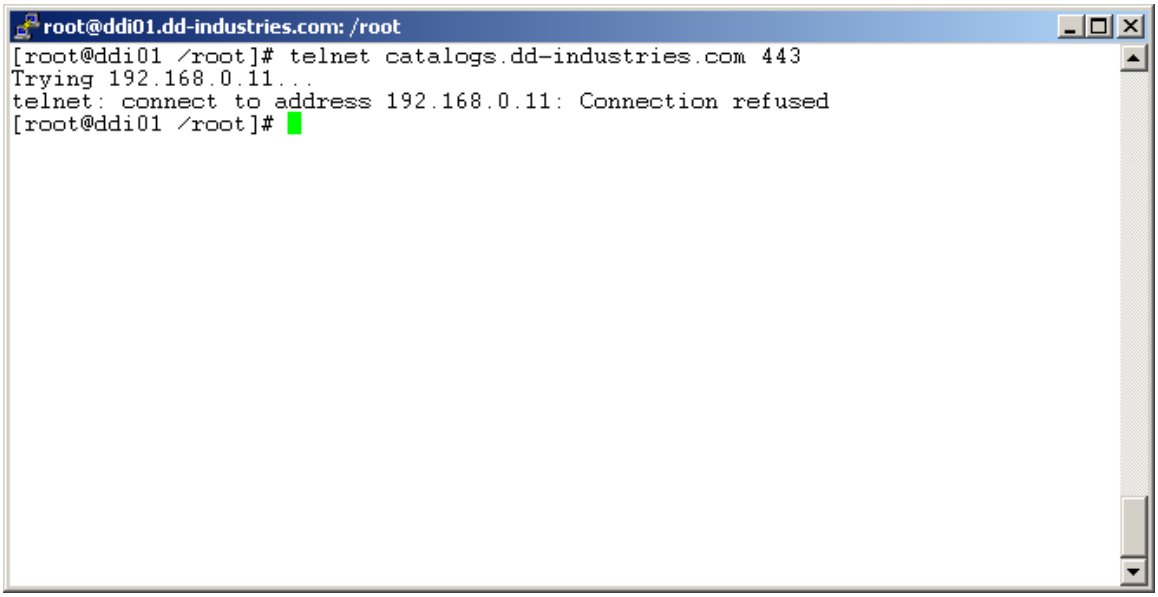
Consider the following:



In the above example,

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

Consider the following:



In the above example,

The server is not yet available

Enabling SSL

OpenSSL

Two things are needed – a certificate and a key.

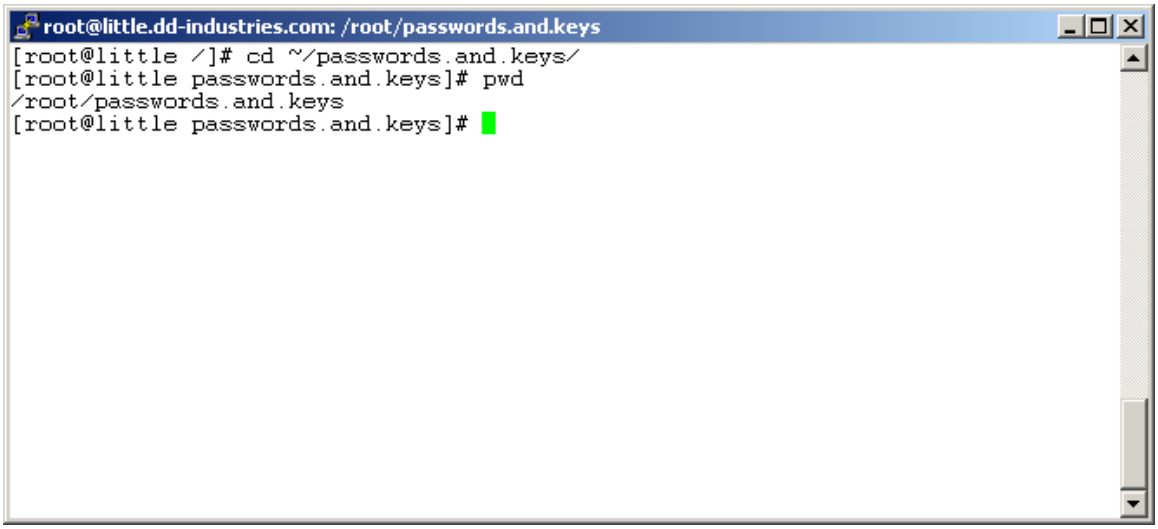
X.509 Certificate Generation

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:

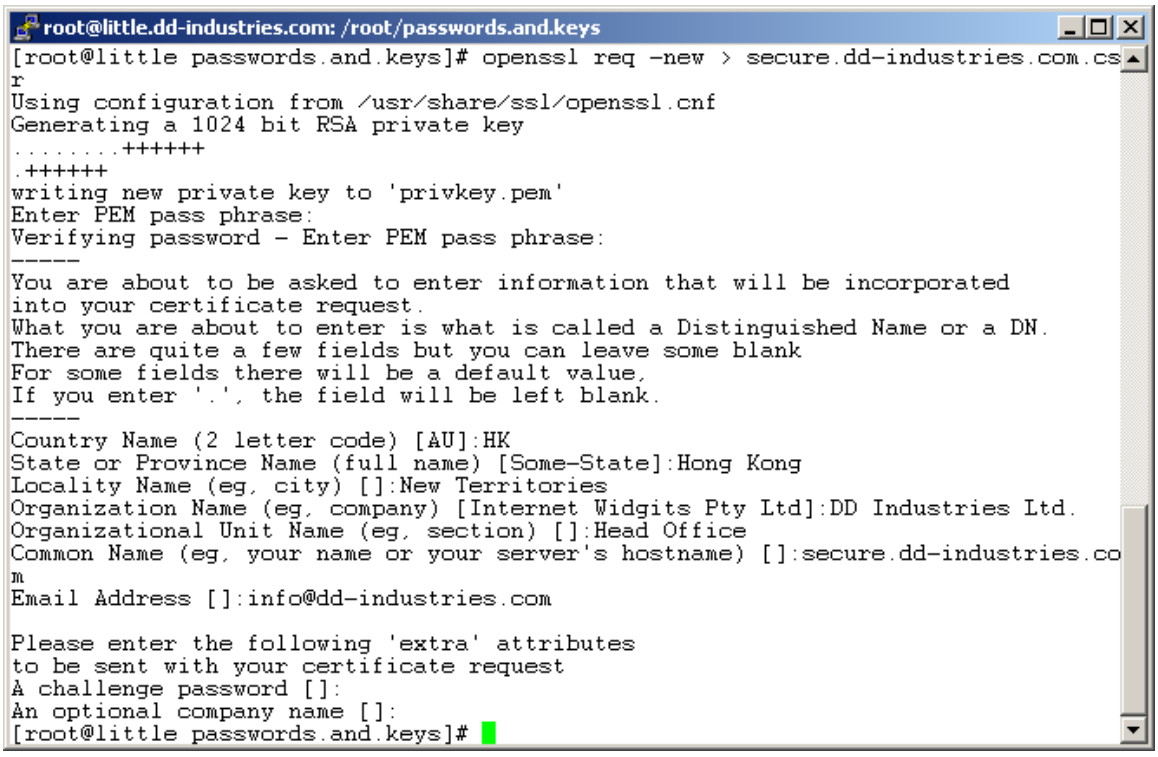


In the above example the **cd** and **pwd** commands were used to navigate to the **/root/passwords.and.keys** directory.

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:



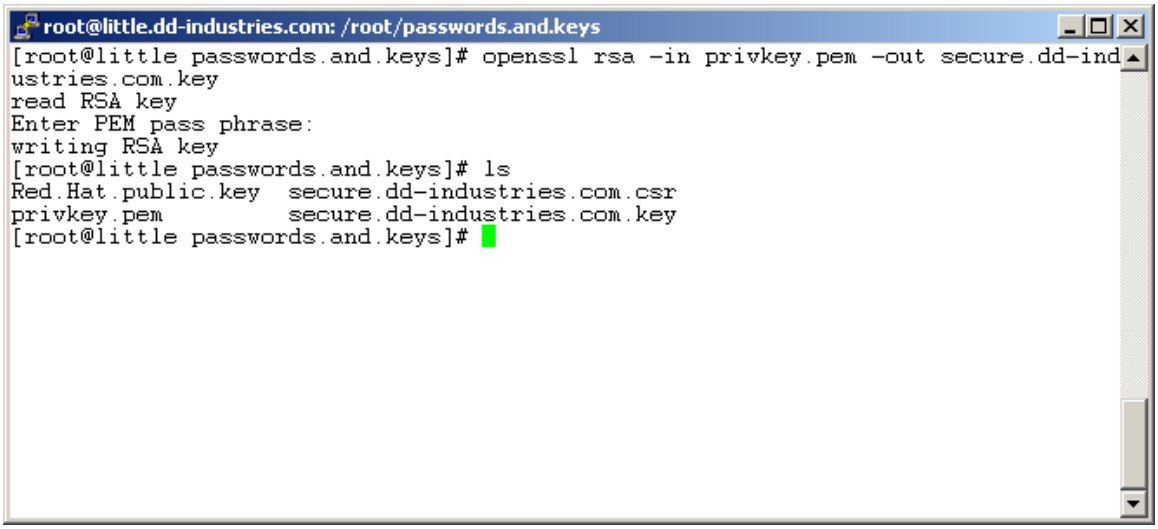
In the above example the **secure.dd-industries.com.csr** file was generated with the **openssl** command. This file contains the information required to create the **X.509** server certificate.

How To Remove the Passphrase From The Private Key

In some instances it may not be desireable 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:

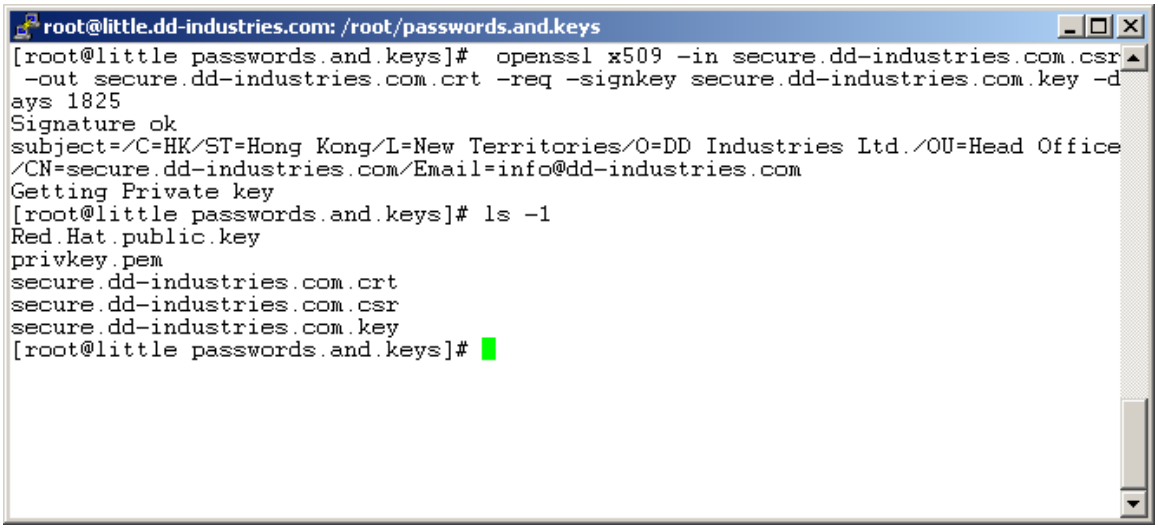


In the above example the passphrase was stripped from the file **privkey.pem** to derive a passphrase-free key file, **secure.dd-industries.com.key**.

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:

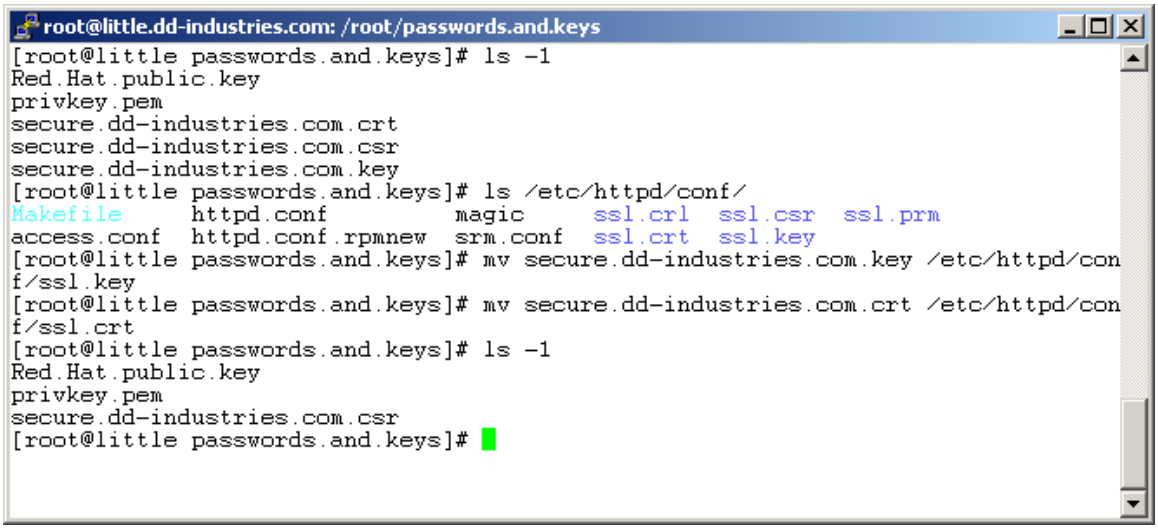


In the above example the **openssl** command was used to create an **X.509** certificate contained in the file **secure.dd-industries.com.crt**.

Where To Put The X.509 Certificates And Server Key

Move the certificate and key to the appropriate places

Consider the following:



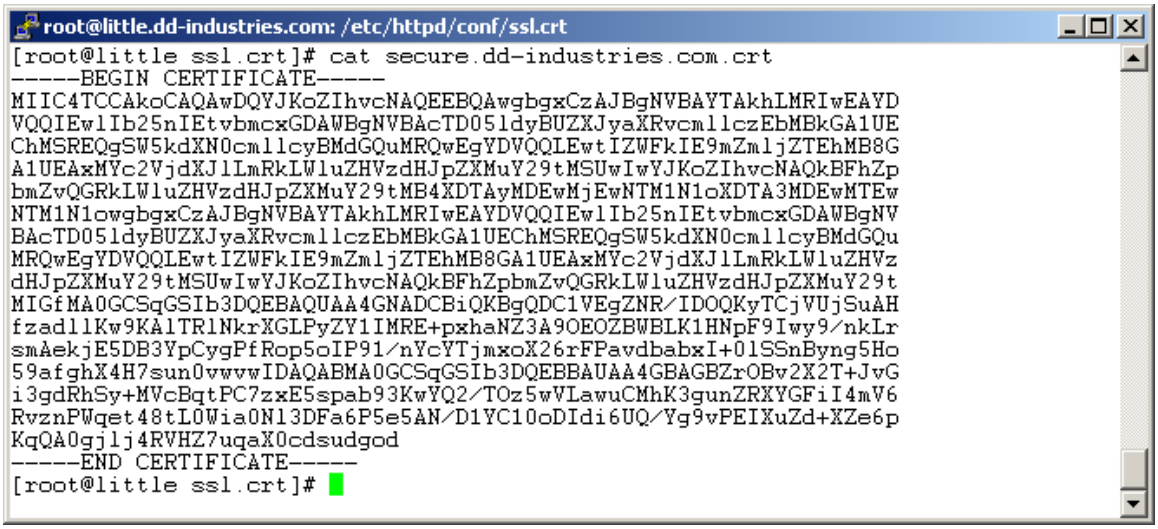
In the above example,

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

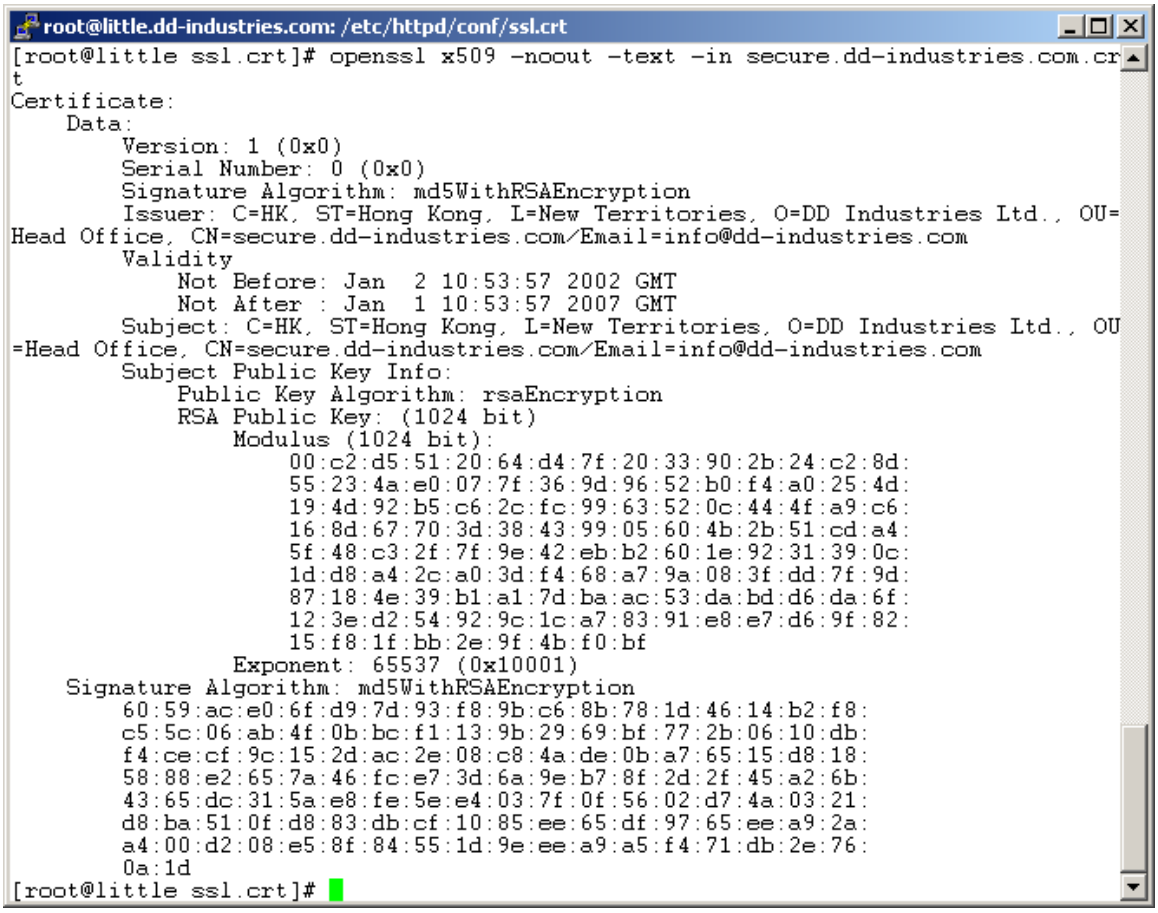
Consider the following:



In the above example,

What The Inside Of A Decoded Certificate Looks Like

Consider the following:



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

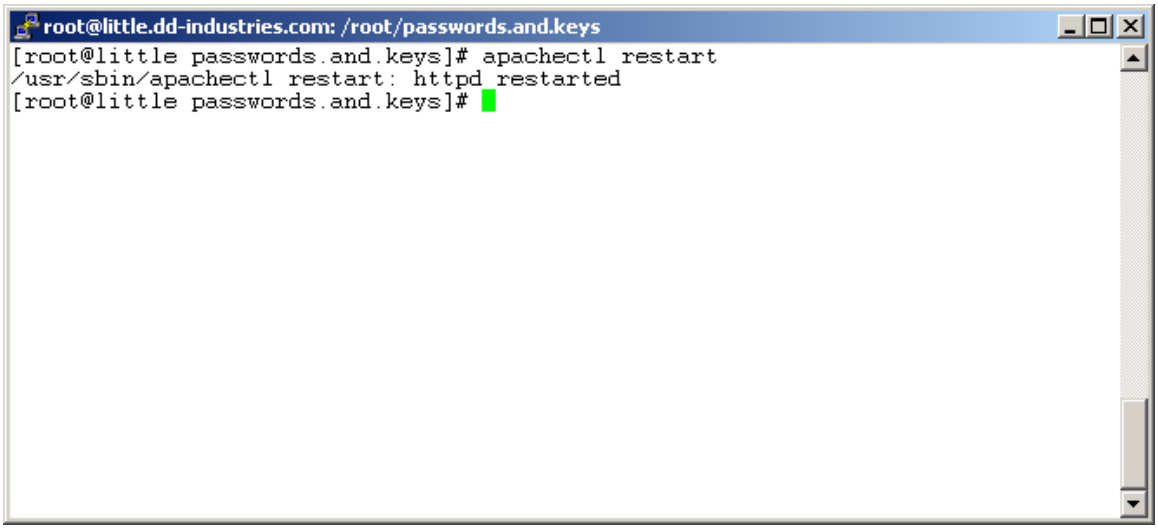
The Edit the httpd.conf file to include the IP address and name **secure.dd-industries.com**.

Consider the following:

In the above example,

Restart **httpd** to enable the secure functionality

Consider the following:

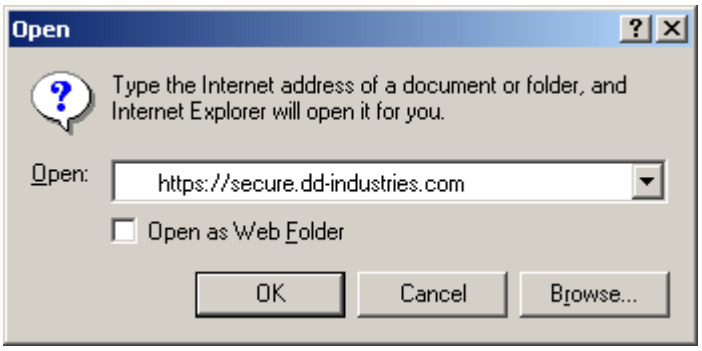


In the above example,

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,

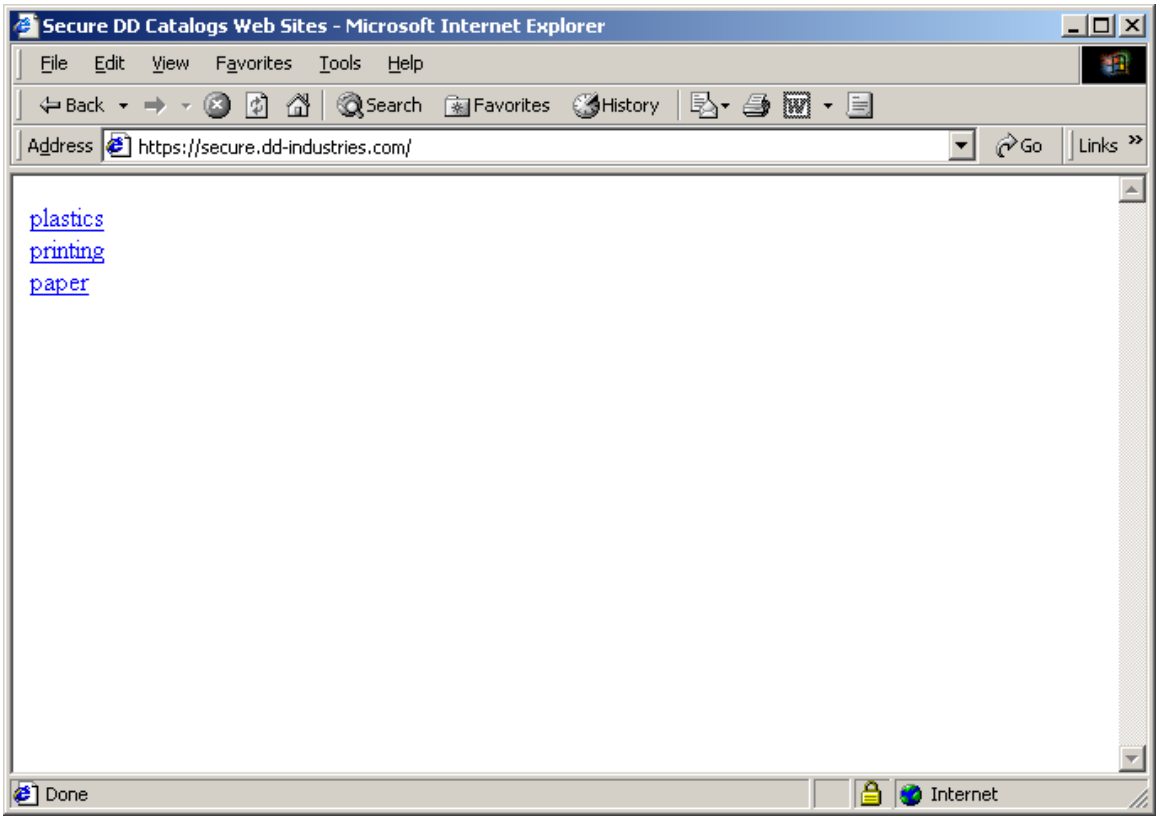
Consider the following:

In the above example,

Consider the following:

In the above example,

Consider the following:

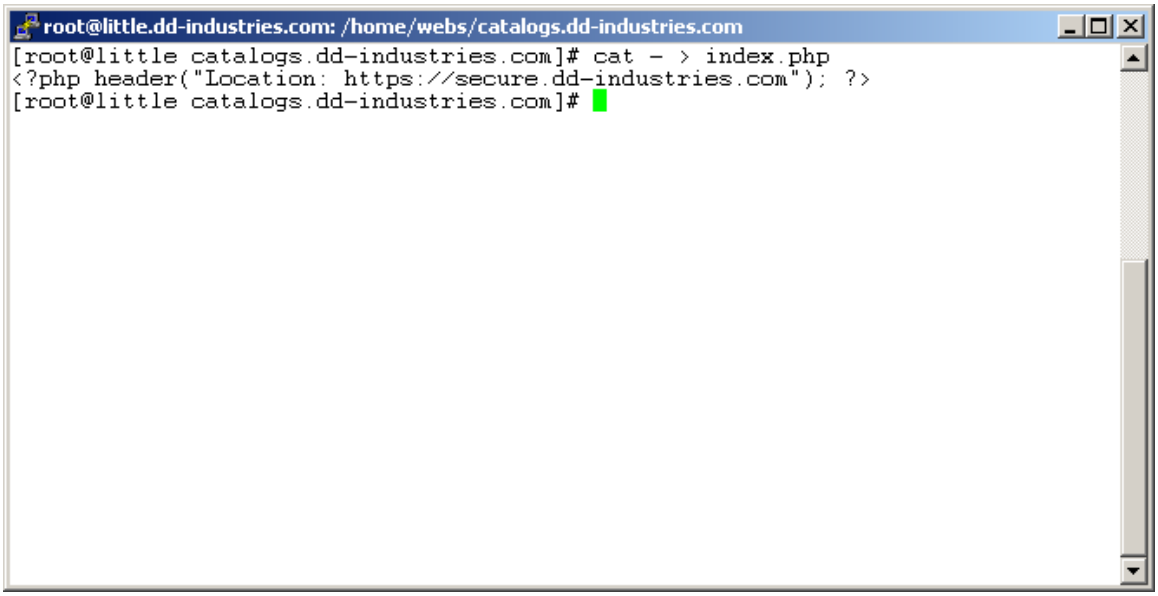


In the above example,

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.

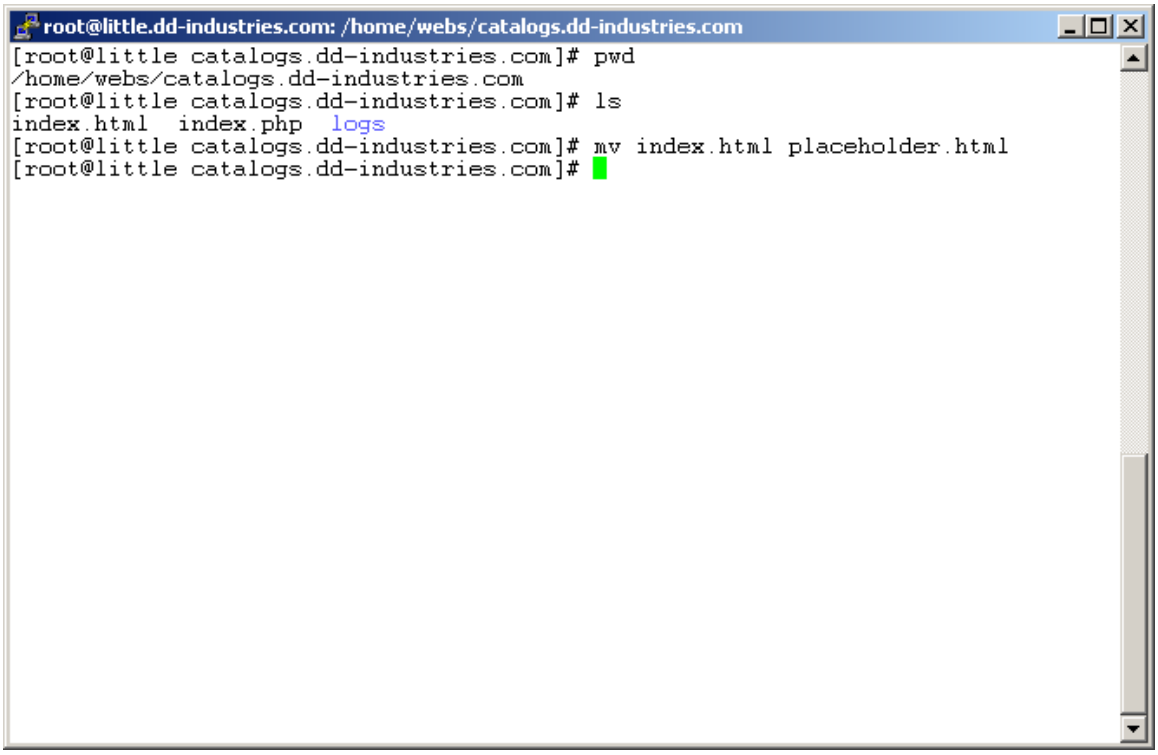
Consider the following:



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

How To Enable HTTPS Redirection

Consider the following:

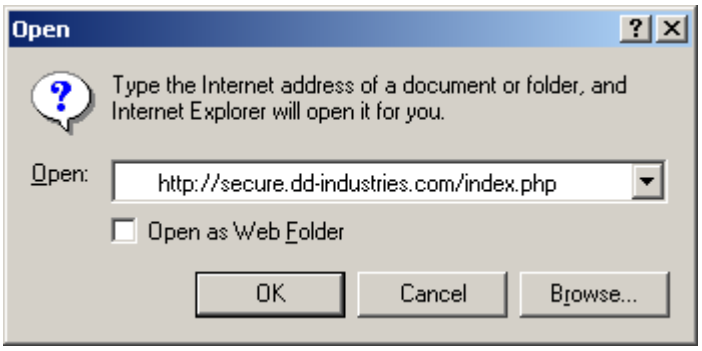


In the above example,

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

How To Test For Correct Redirection

Consider the following:



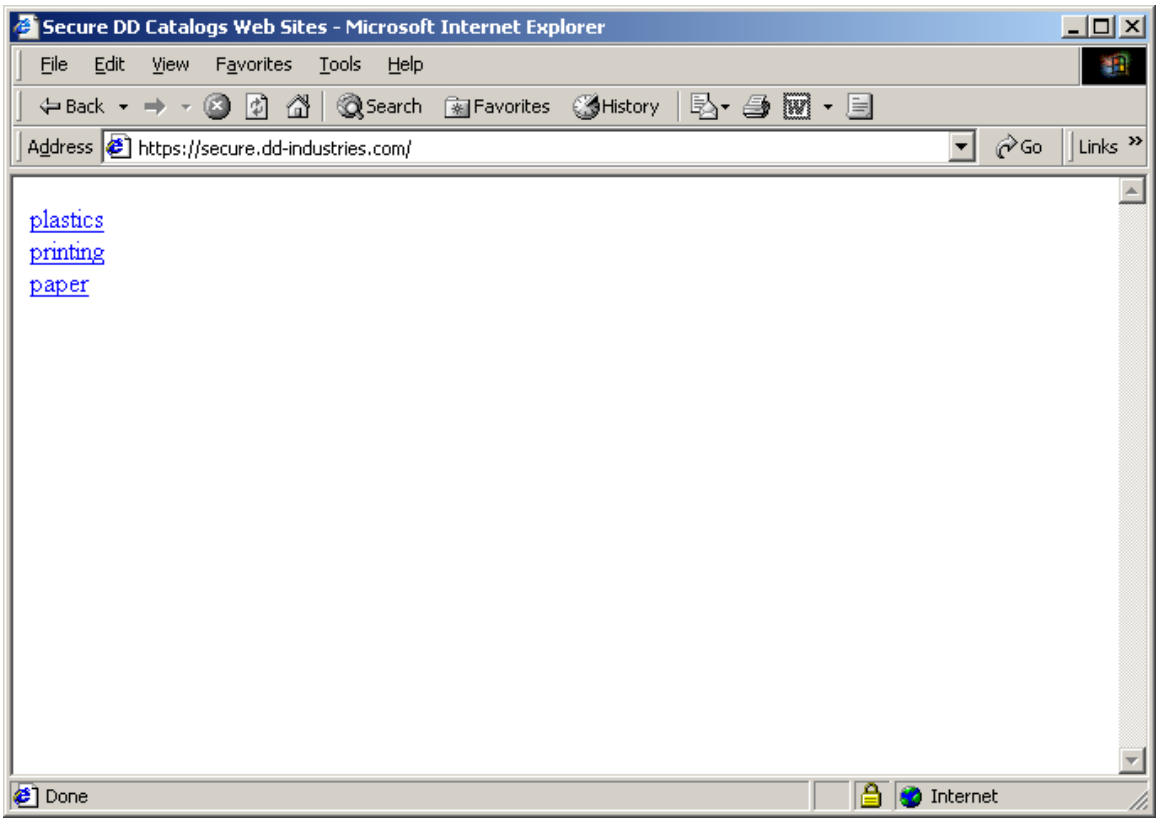
In the above example, the browser is being directed to load **secure.dd-industries.com/index.php**.

Consider the following:

In the above example,

What This Document Enables

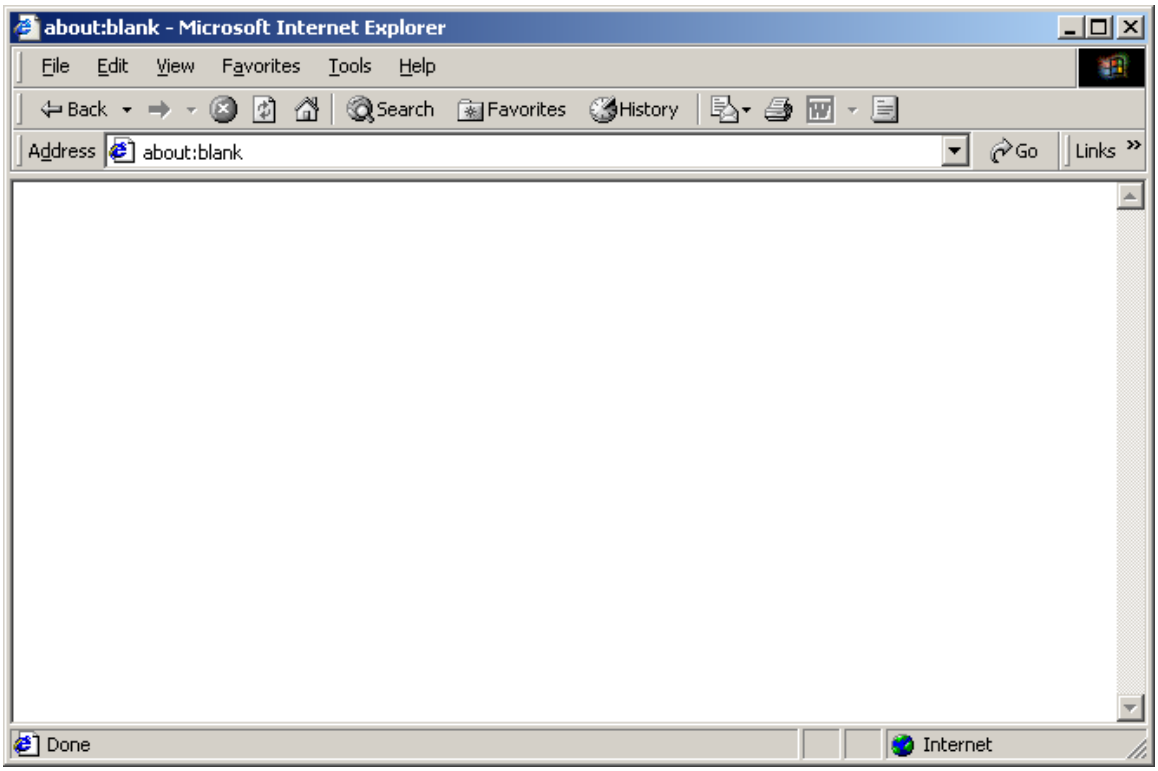
Consider the following:

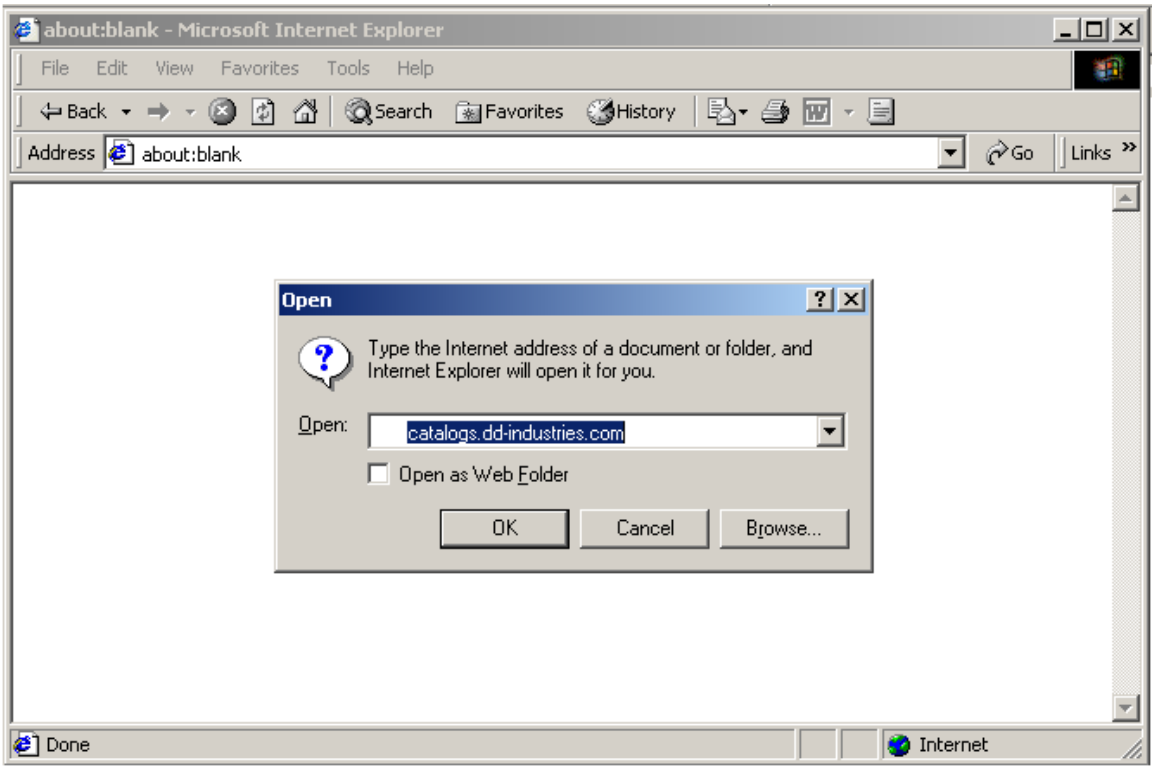


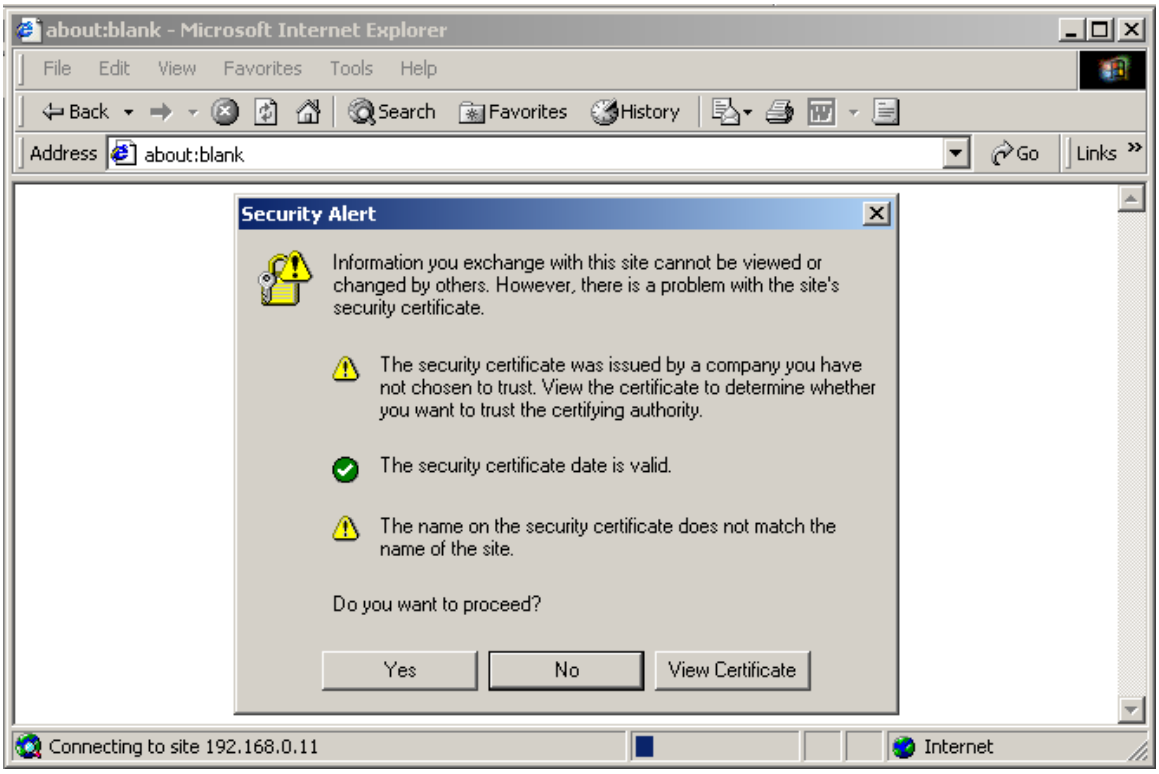
In the above example the web session is now being conducted via HTTPS – indicated in two ways:

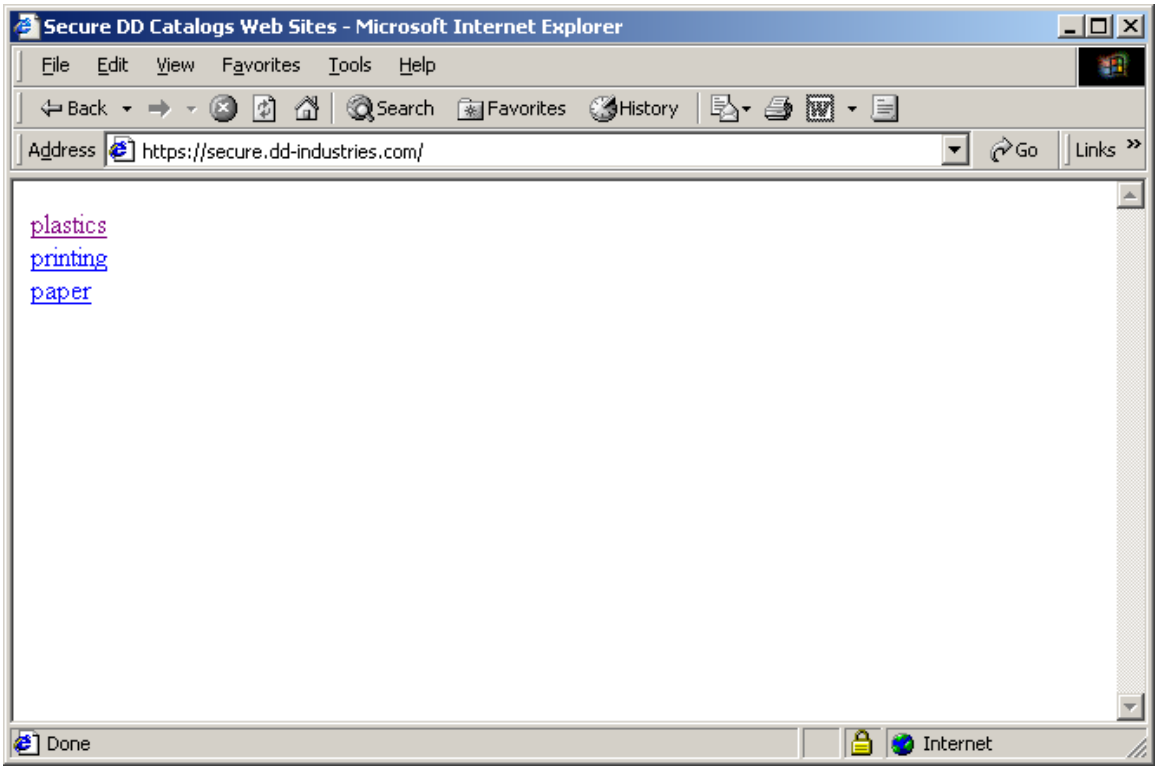
- The URL includes the keyword **https**.
- The lock on the bottom status bar is closed.

Section 4: The Acid Test - Doing It For Real



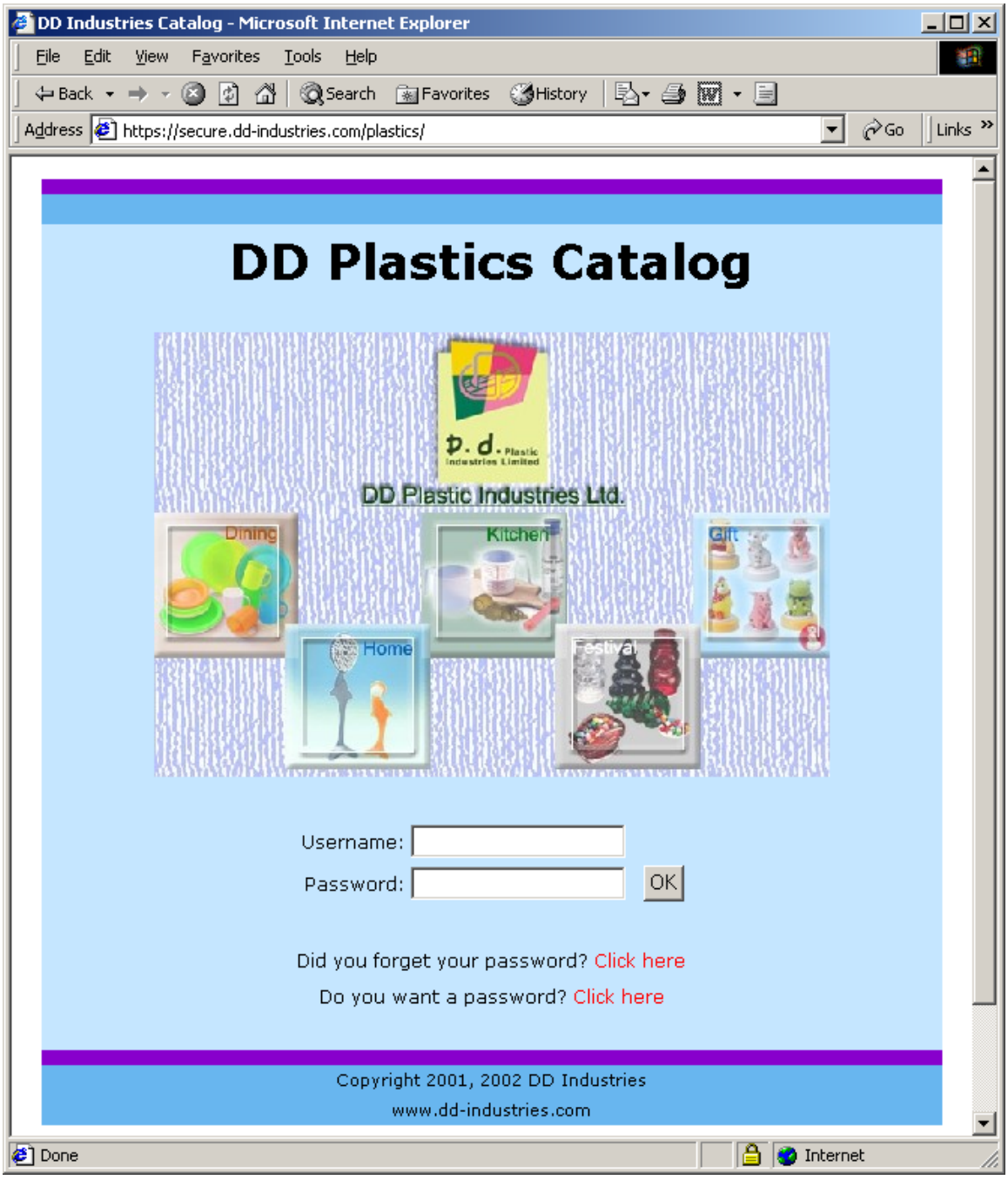

about:blank - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address about:blank
Go Links
Open
Type the Internet address of a document or folder, and Internet Explorer will open it for you.
Open:
catalogs.dd-industries.com
Open as Web Folder
OK
Cancel
Browse...
Done
Internet


about:blank - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Back Search Favorites History
Address about:blank
Go Links
Security Alert
Information you exchange with this site cannot be viewed or changed by others. However, there is a problem with the site's security certificate.
The security certificate was issued by a company you have not chosen to trust. View the certificate to determine whether you want to trust the certifying authority.
The security certificate date is valid.
The name on the security certificate does not match the name of the site.
Do you want to proceed?
Yes
No
View Certificate
Connecting to site 192.168.0.11
Internet



What The Purpose Of This Document Is

Consider the following:



In the above example,

Looks like a secure site to me ☺

13/08/2002

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