KIT304 Server Administration and Security Assurance

Tutorial 13

Goal

In this tutorial you will configure both CentOS and Windows Server 2016 to serve web content in an encrypted form.

Introduction

So far in this module you have set up multiple websites on CentOS and Windows Server. All of these sites delivered their content in plain text. This is how HTTP was first designed, and even HTTP version 2, which was formally published in 2015, does not make encryption of web traffic mandatory. Email and other protocols are not that different either, sending their content across the Internet without encryption by default. However, as of July 1st of 2018, Google's Chrome browser began marking all pages that are not encrypted as insecure to alert users of the risk. Most other web browsers have followed Chrome's lead.

Sending content as plain text is risky. Considering our security goals, confidentiality is obviously not met. But the goals of integrity and authentication cannot be met either. Classic attacks like the Man-In-The-Middle (MITM), where a malicious entity inserts themselves between two communicating parties, subvert those goals. The third party can not only read all communications as it passes through – they can also modify it in transit.

Carrying out such attacks in the context of web traffic is not a new concept, and there have been a number of high-profile examples over the last decade of services being vulnerable to this. In 2010 for example, the **Firesheep** extension to the Firefox browser exposed a significant vulnerability that exists when using unencrypted Wi-Fi networks. The extension included a packet sniffer which enabled it to intercept unencrypted session cookies found on the local network. These could then be re-used to masquerade as the user they were stolen from – consequently hijacking the web presence of anyone else on the same wireless access point. This demonstrated that it was possible to take over the social media presence of another person, and as a result many platforms (such as Twitter and Facebook) began adopting encryption for all of their web pages to prevent further similar exploits.

In this tutorial you will first examine how easy it is to see the content of web traffic, before subsequently looking at how to enable Transport Layer Security (TLS), also referred to as Secure Sockets Layer (SSL), on your webservers. TLS enables the server to authenticate itself to the client and provide a confidential channel of communication between the two parties.

Activity

1. Launch the **CentOS** virtual machine, become the root user and assign the machine an IP address in the **`192.168.1.x`** network.

2. Create a new website with a domain name of your choice. Make sure that the site's **`index.html`** file has some content in it. Test that it loads in the browser on CentOS.

3. Launch the **Kali** virtual machine, and give it an IP address on the **`192.168.1.x`** network (**`ip a add 192.168.1.x/24 dev eth0`**). Next, launch **Wireshark** (located in the **Applications** menu under **Sniffing and Spoofing**) so that you can view traffic sent over the virtual network. That's all you need to do on the Kali VM for the moment – you'll return to Wireshark shortly.

4. Launch the **Windows Server 2016** VM and give it an IP address on the same network as your other virtual machines. Edit the **`hosts`** file (at **`C:\Windows\System32\drivers\etc`**) and add an entry for the domain that you created on the CentOS server. Check that you can ping the CentOS server by domain name from the Windows command line – if you get no response, double check your settings.

   Finally, open **Internet Explorer**, but don't try to visit any sites yet.

5. In real networks, entities that have access to the network infrastructure through which our packets are travelling can take steps to view those packets. Thus, ISPs, other corporations and governments all have ready access to the vast majority of traffic on the internet. You'll now play the role of such an entity. Go back to the Kali VM and start capturing traffic on the virtual network in **Wireshark** by double clicking on **eth0**.

6. In **Internet Explorer** on your 2016 server, browse to the website that you created on CentOS. Once the page loads, go back to Wireshark, and stop the capture by clicking on the red square, representing a "Stop" action, in the toolbar.

   If you examine the Wireshark capture, you should be able to find and explore the packets that were captured while transiting the network between the CentOS and Windows 2016 Server IP addresses, and you should be able to locate the content of the **`index.html`** file sent via the HTTP protocol. If you can't find this file, and see its content, try the capture again, or seek assistance from the tutor.

Aside: Transport Layer Security (also known as Secure Sockets Layer)

In the prerequisite networking unit to this one (either KIT111, or KIT201) you were introduced to several cryptographic protocols – one of which was TLS. You examined the steps it goes through to establish a secure tunnel between the server and the client. You won't be dwelling on that in this

tutorial, but you will be configuring your servers to use TLS, and then observing the transfers in Wireshark, comparing them to the same transfers sent without using any form of encryption.

SSL was first proposed in 1996 by Netscape as a secure way of communicating across the internet. It has since been standardised and renamed to TLS. TLS 1.0 is equivalent to SSL 3.0, although TLS 1.0 has effectively been deprecated since March 2020 when the main browser developers started removing support for it. The current version is TLS 1.3, having been published as RFC 8446 in August 2018, although most TLS connections are still using the 1.2 standard.

TLS creates a secure tunnel to carry network traffic between client and server. To do this it completes a multi-step handshake procedure involving certificate(s) to prove identity (the server's certificate is mandatory, while the client's is optional depending on the security goals) and randomly generated data to produce session keys. Symmetric sessions keys are established and used to encrypt the traffic for the conversation and are then discarded afterward.

A significant component of TLS in the context of the web, and in combatting man-in-the-middle attacks, is that it certifies who the server is. The server's public/private key pair enables the user to be certain that they are communicating with the server and not someone pretending to be the server. Secondarily, all traffic is encrypted, and cannot be eavesdropped by someone intercepting that traffic as it makes its way between server and client. (It is not impossible to do a man-in-the-middle attack against TLS, and such attacks do exist, but it is significantly harder to do than it is against unsecured HTTP, especially against a server that has also implemented HSTS – also known as HTTP Strict Transport Security – which allows web servers to indicate to browsers that they require all connections to use TLS).

7. OpenSSL, an open source implementation of TLS, is already installed on **CentOS**, but it is not yet configured. If you are using your own CentOS virtual machine, the **`yum`** command that you need to match the lab configuration is **`yum install mod_ssl openssl`**.

   To configure OpenSSL, you first need to create a directory to store the related SSL files, as follows:

   ```
   mkdir /etc/httpd/ssl
   cd !$
   ```

8. Next, you will use OpenSSL to generate a private key and a certificate so that your server can accept and respond to SSL requests. Enter the following command to do this (note that this is one long command, entered as a single line in the Terminal window).

   > ➢ In the following command, the name of the key file (**`web.key`**) and the certificate file (**`web.crt`**) can be any values you want – for example, you may choose to use your domain name instead of **`web`**, to reflect which site they are relevant to.

```
openssl req -x509 -nodes -days 365 -newkey rsa:2048 -keyout
/etc/httpd/ssl/web.key -out /etc/httpd/ssl/web.crt
```

Once you enter this command, **openssl** will ask you a series of questions, the answers to which will be incorporated into your certificate. For today's tutorial, it doesn't matter what you enter in response to these questions, as you're only generating a certificate for use within your own virtual network (this is known as a **self-signed certificate**).

> ➢ Note: self-signed certificates provide encryption (so they give us confidentiality) but they don't come with any guarantees about the authenticity of the site that serves them – they simply prove that the site has a certificate and can use encrypted traffic. For this reason, most web browsers will treat such sites as insecure, and will require the user to explicitly indicate that they want to proceed to the site's content.

The generated **.crt** and **.key** files are in plain text – be sure to **cat** them to the terminal so you can see their contents.

9. Next you need to edit the Apache web server's SSL configuration to use your new certificate and key when a client requests a secure connection. To do that, edit **/etc/httpd/conf.d/ssl.conf**.

   Scroll down until you see the Virtual Host heading **<VirtualHost _default_:443>** (typically around line 40).

   Just inside this section, remove the comment tags (**#**) in front of the **DocumentRoot** and **ServerName** settings, and update their values to reflect your own domain and its root directory. If you're using the same naming approach as we've been using for recent tutorials, that's likely to be something like this:

   ```
   DocumentRoot "/var/www/yourDomain/html"
   ServerName yourDomain:443
   ```

   Now move further down and locate the **SSLCertificateFile** and **SSLCertifiateKeyFile** lines, and edit them to match the actual location of the files you created earlier:

   ```
   SSLCertificateFile /etc/httpd/ssl/web.crt
   SSLCertificateKeyFile /etc/httpd/ssl/web.key
   ```

   Save your changes and exit the editor.

10. Reload the **`httpd`** web server so the system picks up the configuration changes you made, and go back to your Internet Explorer session in Windows 2016 Server. Replace the **`http:`** in front your domain name with **`https:`**, and load the page again.

    The browser (if it's Internet Explorer) should issue a security alert that "*You are about to view pages over a secure connection*", and that "*Any information you exchange with this site cannot be viewed by anyone else on the web*". That's an odd alert, since this is a desirable state, and is your goal today! Click the OK button to proceed.

    Next, after a rather lengthy one-time delay, you'll see another warning stating that "***There is a problem with this website's security certificate***". That is because the site is using a self-signed certificate. As stated earlier, such certificates provide confidentiality but they do not guarantee identity, and are therefore not trusted by default.

    In general, you should not trust self-signed certificates in public contexts. The public key infrastructure that is used in many cryptographic protocols operates on the principle that certificates are signed by trusted third parties which validate the identity of the owner, and issue a public key (and its certificate) to that entity for use with secure connections. Self-signed certificates aren't signed by a trusted third party, and therefore you don't know whether the site you're connecting to is in fact owned by the entity that it claims to be.

    Multinational Certificate Authorities (such as IdenTrust and Sectigo, which had 49.5% and 26.5% respectively, of the global market share for SSL website certificates as of April 2019 according to **w3techs.com**) serve as trusted third parties to fill this role. They issue millions of certificates globally to businesses and individuals who pay a fee and go through a verification process to validate that they are who they say they are.

    A free alternative, if you have a publicly accessible domain name, is to use the **letsencrypt** certificate service, but that's not an option for servers hosted on private networks.

    Today you can accept the risk – you created the certificate yourself after all – so click the **Continue to this website** link. This is something you should be very careful about doing on a real website – it may be a sign of a poorly executed phishing scam that is attempting to steal information from you.

    Once the web page loads, you should see the address bar now has a pink background – a warning that you are at an untrusted site. If you click on the **Certificate…** warning at the right-side of the address bar, you can view the certificate. You should be able to see, on the **Details** tab, the **Issuer** and **Subject** fields contain the responses you entered earlier when you created the certificate and key. Once you've explored the certificate, close its window.

11. Back on the **Kali** VM, click the blue shark fin icon in the toolbar to start a new capture session, then go back to Internet Explorer on Windows 2016 Server, and **re-enter the URL for the secure**

page on the CentOS server (don't just reload the page – this won't generate the traffic you need to see). When the page finishes loading, stop Wireshark and study the traffic that was captured.

If you trace the packet captures, you should be able to see the TLSv1.2 **Client Hello**, and after that, you'll see a packet **Server Hello, Certificate, Server Key Exchange, Server Hello Done** – this is where the server sends its certificate to the browser. If you isolate this packet, you'll even be able to see the certificate content inside, including the values you entered when you created the certificate.

You'll also see other TLS handshake packets (which you should recall from KIT111/KIT201) and you'll see packets that the **Info** column describes as **Application Data**. These are the encrypted responses from the server that you are unable to read, and this is why https is used – to prevent snooping on traffic.

12. Next, you'll repeat the tasks that you carried out above, but this time you'll be setting up a certificate and encrypted web service on Windows 2016 Server, and checking that you can access it from CentOS.

    - Start by creating a website on the Windows 2016 Server with its root folder inside the **`inetpub`** directory of the **`C:`** drive.
    - Create an **`index.html`** file for this site so that you can distinguish it from the website you set up on CentOS.
    - Back on the CentOS VM, edit the **`/etc/hosts`** file in order to add the domain name for the new Windows site.
    - Finally confirm that the Firefox browser on CentOS can load the new (unencrypted) site.

13. The next step is to create a certificate and private key on Windows that IIS can use to securely serve your web site's content:

    - Open the **Internet Information Services (IIS) Manager** and select the root node of the **Connections** hierarchy named **CORPDC1**.
    - Double-click on **Server Certificates** in the main window to open the Server Certificates panel.
    - At this point, you could import a certificate that was previously created by a third party (for example, after they verified your identity) but since you can't go through that process for the tutorial, you'll again need to create a self-signed certificate. To do this, click on **Create Self-Signed Certificate** from the **Actions** pane on the right-hand side of the window.
    - Enter a name for the certificate (for example, this could be your domain name), and then select **Web Hosting** from the drop-down menu and click **OK**.

14. Now you need to modify your web site, first enabling SSL connections, and then associating your new certificate with the site.

- Return to the **Connections** hierarchy on the left-hand side of the window, and expand the **Sites** branch of the hierarchy if necessary, and click on your site's name.
- Click on **Bindings** from the **Actions** pane on the right-hand side of the window and add a new binding using the **https** type.
- Enter your domain name into the **Host name** field, and select your new self-signed certificate from the **SSL certificate** dropdown menu.
- Click **OK**, and then click **Close**.

15. Now, on CentOS, try to load the secure version of the site by replacing the **http** prefix with **https** in Firefox's address bar. As before, you'll get a security warning telling you "*Warning: Potential Security Risk Ahead*", and you'll need to click the **Advanced** button to add an exception to the browser's security policy to force it to accept the certificate. With this done, you should now see the web site's content in the browser.

16. Quit Firefox (to ensure that it doesn't retain a persistent TCP connection to the web server), and then re-run it.

17. Return to **Kali**, start a new **Wireshark** capture, and then return to Firefox on CentOS. **Enter the URL for the secure page on the Windows server** in the Firefox URL bar, and then head back to Kali and stop the Wireshark capture. As before you should be able to trace the initial TCP handshake, the TLS connection set up, and then the Application Data packets that hold the encrypted version of the web content.

Conclusion

Today you've learned about encryption of web content, underpinned by TLS and the use of self-signed certificates.

In real-world scenarios, self-signed certificates prevent eavesdropping and tampering with content, but they don't guarantee that the source of the content is who they claim to be, since anyone can generate them without going through a validation process.

10 years ago, the use of self-signed certificates was considered acceptable, but because they don't authenticate the sender they've fallen out of favour, and modern web browsers will display prominent warnings if you visit a site using them.

The solution is to use a free service like **`letsencrypt.org`**, or pay an organisation (such as Verisign) to verify your identity and issue certificates for you that they've signed with their own private key. These organisations have trust relationships with browser vendors so that your signed certificate is trusted.

Skills to Master

To successfully complete this tutorial, you must have mastered the following skills:

- creating a self-signed certificate on CentOS and applying it to the Apache configuration files
- creating a self-signed certificate on Windows 2016 server and applying it to a web site

You could be required to demonstrate any of these skills in this module's practical exam.