B2: Web Server

Motivation

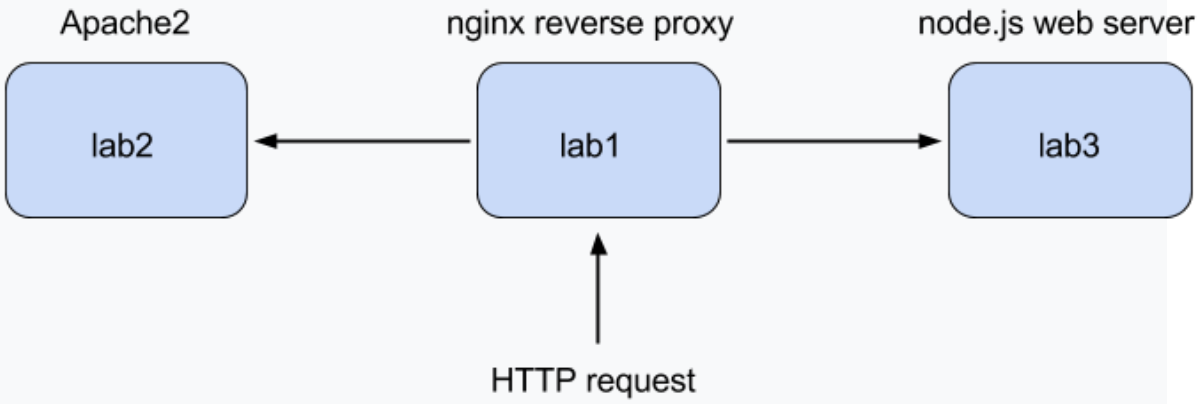
Serving a web page requires a running server, listening for incoming HTTP requests, and answering them by returning the requested page. Taking last exercise's bogus server further, this time you will serve an actual web page to a browser requesting it. Websites can consist of multiple servers around the world, to load balance, to keep latency low, or to keep important information on more secure servers. To simulate this, you will set up a reverse proxy that will forward different requests to different servers.

Suggested reads

- Apache module index - You can check the usage of the needed modules
- RFC 5280 - Public Key Infrastructure Certificate Profile

Description of the exercise

In this exercise, you will introduce yourself to some basic features of Apache web server and its plugins. In addition to that you will set up a Node.js server for serving a webpage, and configure an nginx server as a reverse proxy for the servers. Take into account that from now on you'll have to do extensive self-research to be able to successfully complete the assignments. You will need three virtual machines to complete this assignment. The final web server network configuration will look like the image below.



1. Apache2

Apache2 allows for quick and easy configuration of a web server on Linux machines, while being extensible with many easy to use modules. It is a quick first step, if you need to host a website on your own computer for example, but is also usable in a production environment.

Install Apache2 on lab2. The modules used later for serving user directory contents, rewriting URLs and setting up SSL should come with Apache2 by default. Set up SSH port forwarding for HTTP and HTTPS so that you can test the server on your local machine (localhost) with your favourite web browser. Note that VirtualBox port forwarding is not the

way to do this! Instead, look into the ssh(1) manual page and use an ssh command to link the ports on the virtual machines to ports on the host.

1.1	Serve a default web page using Apache2	1p
1.2	Show that the web page can be loaded on local browser (your machine or Niksula) using SSH port forwarding.	1p

2. Serve a web page using Node.js

Node is a cross platform browser-free JavaScript runtime environment, which can also be used as a HTTP server, when configured with the appropriate modules. The purpose of this assignment is to familiarize yourself with the increasingly popular and simple method of serving web applications using Node.js. Although javascript skills are not really needed in this assignment, we strongly recommend that you take a deeper look at javascript and also node.js

Install *nodejs* on lab3 from package manager and create an HTTP application *helloworld.js* listening on port 8080 that serves a web page with the text "Hello world!".

The web pages served by Node.js are written in javascript, but you do not actually need to know how to write it, because there's plenty of hello world examples on the internet. You do need to understand the contents.

2.1	Provide a working web page with the text "Hello World!"	2p
2.2	Explain the contents of the helloworld.js javascript file.	1p
2.3	What does it mean that Node.js is event driven? What are the advantages in such an approach?	2p

3. Configuring SSL for Apache2

This next part introduces you to SSL certificates by configuring your Apache server with one that you create.

On lab2, use the Apache ssl module to enable https connections to the server. Create a 2048-bit RSA-key for the Apache2 server. Then create a certificate that matches to the key. The suggested method for achieving these two steps is using openssl. Configure Apache2 to use this certificate for HTTPS traffic. Set up again SSH port forwarding for the HTTPS port to test the secure connection using your local browser (if it is not active already).

Note: Taking a shortcut with CA.pl is not accepted, you need to understand the process! Only a few commands are needed, though. Both the key and certificate can be created simultaneously using a single shell command.

3.1	Provide and explain your solution.	1p
3.2	What information can a certificate include? What is necessary for it to work in the context of a web server?	1p
3.3	What do PKI and requesting a certificate mean?	1p

4. Enforcing HTTPS

Next, you will enforce the use of an SSL encrypted connection for one page on your server, while still allowing http on another. You will also learn to serve a directory from the user's home directory.

On lab2, create a “public_html” directory and subdirectory called "secure_secrets" in your user’s home directory. Use the Apache userdir module to serve public_html from users' home directories.

Enforce access to the secure_secrets subdirectory with HTTPS by using rewrite module and .htaccess file, so that Apache2 forwards "http://localhost/~user/secure_secrets" to "https://localhost/~user/secure_secrets". Please note that this is a bit more complicated to test with the ssh forwarding, so you can test it locally with lynx or netcat at the virtual machine. If your demo requires, you may hard-code your port numbers to the forwarding rules.

4.1	Provide and explain your solution.	2p

4.2	What is HSTS?	1p
4.3	When to use .htaccess? In contrast, when not to use it?	1p

5. Install nginx as a reverse proxy

Nginx is a third commonly used way of serving webpages, and also allows functioning as a proxy. Next, you are going to serve both Apache2 and Node.js hello world from lab1 using *nginx* as a reverse proxy.

Install *nginx* on lab1 and configure it to act as a gateway to both Apache2 at lab2 and Node.js at lab3 the following way:

HTTP requests to *http://lab1/apache* are directed to Apache2 server listening on lab2:80 and requests to *http://lab1/node* to Node.js server on lab3:8080.

5.1	Provide a working solution serving both web applications from nginx.	2p
5.2	Explain the contents of the nginx configuration file.	1p
5.3	What is commonly the primary purpose of an nginx server and why?	1p

6. Test Damn Vulnerable Web Application

For security purposes, security professionals and penetration testers set up a Damn Vulnerable Web Application to practice some of the most common vulnerabilities. To achieve this goal, you can download the file in https://github.com/digininja/DVWA/archive/master.zip and install it on lab2. You can look at

for guidance. You can delete the default Apache index.html and install the web application to be served as the default webpage. Finally, install Nikto tool which is an open-source web server scanner on lab 1 and scan your vulnerable web application.

6.1	Using Nmap, enumerate the lab2, and detect the os version, php version, apache version and open ports	2p
6.2	Using Nikto, to detect vulnerabilities on lab2	2p

7. Finishing your work

When finishing your work, please remember to backup files related to the assignment and after your demo possibly reset the configuration changes that you did to the lab environment (Lab1, Lab2, Lab3) to start the next assignment with a clean slate.