HyperionDev

Setting Up for Cyber Security

Task

Visit our website

Introduction

In this task, you will explore how computers communicate over networks such as the Internet using the Hypertext Transfer Protocol (HTTP) and the client–server architecture, and how these concepts form the foundation of many cyber attacks and defenses. Almost every web vulnerability, from injection attacks to session hijacking, is rooted in how requests and responses move between clients and servers. Understanding HTTP and the client–server model is therefore not only a web development topic but a core part of how cybersecurity professionals analyze, secure, and test modern applications.

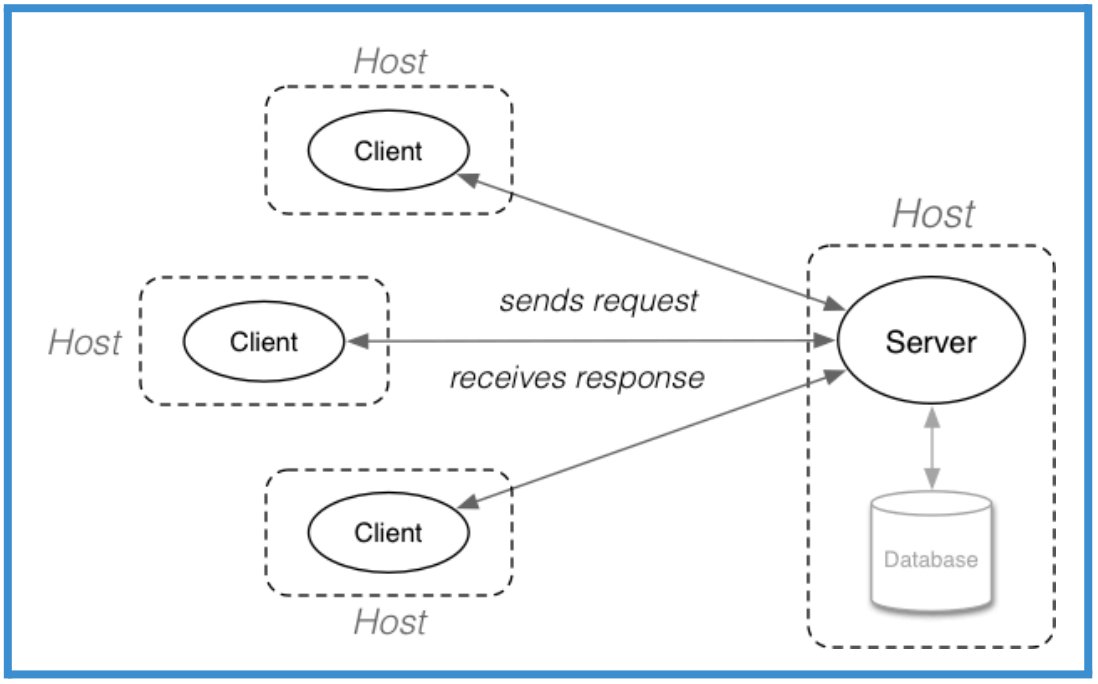
You will also learn how to set up Kali Linux using the additional reading materials provided with this lesson. Kali Linux will be your primary security lab environment in this bootcamp, allowing you to practice network analysis, web application testing, and other hands-on cybersecurity activities in a controlled virtual machine rather than on your main operating system. Be sure to review the additional readings carefully so that you can successfully install and configure your Kali Linux virtual machine before moving on to later, more advanced tasks.

Client–server architecture

Client–server architecture, also known as a client–server model, is a network architecture that breaks down tasks and workloads between clients and servers that reside on the same system, or are linked by a computer network such as an intranet or the Internet.

Client–server architecture typically features multiple workstations, PCs, or other devices belonging to users, connected to a central server via an Internet connection, or another network connection. The client sends a request for data, and the server accepts and processes the request, sending the requested data back to the user who needs it on the client side.

For the purposes of this bootcamp, we will be focusing on how this model relates to cyber security in particular.



Client–server model

The above illustration shows how clients interact with a server and how that server interacts with the database so that data is processed by a server and sent back to the clients.

It is not entirely accurate to say that a server is a computer that clients make requests to. A computer must have appropriate server software running on it to make it a server. Examples of server software are **Apache**, **Tomcat**, and **NGINX**.

A client also does not just refer to a device that makes the request to a server, but as with a server, a client needs the appropriate software to make requests. The most common client that you will have come across in your everyday life is your web browser. However, there are other clients as well, such as your Netflix application when you are streaming videos. Here the application is the client and there is a server that is serving the video data to the client.

Another kind of client is an SSH client, which connects to the shell of a remote computer using the SSH protocol. The connection is encrypted, so only the client and server computers know what data is being transmitted. Here, the client is the computer you connect from, and the server is the computer you connect to.

A client can therefore be any device with the software necessary to communicate with the server.

HTTP

HTTP is an underlying protocol used by the World Wide Web. A protocol is basically a set of rules that are decided on and adopted so that there is consistency in how to perform particular tasks. HTTP defines how messages are formed and transmitted between clients and servers, and what actions web servers and clients should take in response to various commands.

A simple example of how HTTP is implemented is when a URL is entered into the browser, and the browser sends an HTTP command to the web server directing it to search for and transmit the requested web page. The response would in this case be an HTML file that the browser can interpret and display to the user.

The HTTP protocol is a **stateless** one. This means that every HTTP request the server receives is independent and does not relate to requests that came prior to it. For example, imagine the following scenario: A request is made for the first ten user records from a database, and then another request is made for the next ten records. These requests would be unrelated to each other.

With a **stateful** protocol, the server remembers each client position inside the result set, and therefore the requests will be similar to the following:

- Give me the first ten user records.
- Give me the next ten records.

With a stateless protocol, however, the server does not hold the state of its client, and therefore the client's position in the result set needs to be sent as part of the requests, like this:

- Give me user records from user 1 to user 10.
- Give me user records from user 11 to user 20.

HTTPS, a secure version of HTTP, includes authentication and security protocols or encryption. HTTPS allows a secure channel on the otherwise insecure Internet, using something called the Secure Socket Layer (SSL).

The request/response cycle

As we start to build out web applications, it is essential to be able to visualise the way information flows through the system, typically called the request/response cycle.

First, a user gives a client a URL, and the client builds a request for information (or resources) to be generated by a server. The request is sent from the client to the server.

When the server receives that request, it uses the information included in the request to build a response that contains the requested information. Once created, that response is sent back to the client in the requested format to be rendered to the user.

It is a developer's job to build out and maintain servers that can successfully build responses based on standardised requests that will be received from clients. But, what does a standard request look like? We need to know that before we can build servers that will respond successfully.

The HTTP message

HTTP messages are used for the requests and responses. HTTP messages are composed of textual information encoded in ASCII, and span multiple lines.

Web developers, or webmasters, rarely craft these textual HTTP messages themselves (although this can be done): software, a web browser, proxy, or web server perform this action.

Compulsory reading

The Mozilla Developer Network documentation breaks down the anatomy of an **HTTP message** very well. **This is compulsory reading for this task.**

Please read the resource above before moving on.

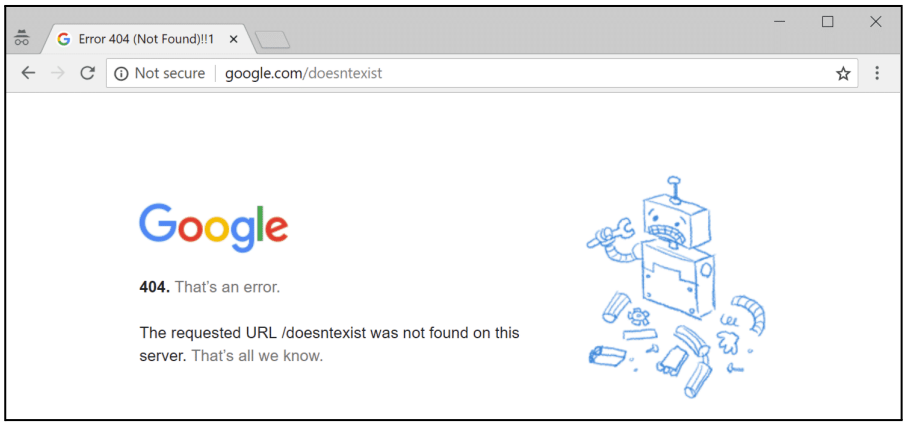
Status codes

HTTP status codes are like short notes from a server that get tacked onto a web page. They're not actually part of the site's content. Instead, they're messages from the server letting you know how things went when it received the request to view a certain page.

These kinds of messages are returned every time your browser interacts with a server, even if you don't see them. If you're a website owner or developer, understanding HTTP status codes is critical. When they do show up, HTTP status codes are an invaluable tool for diagnosing and fixing website configuration errors.

HTTP status codes are delivered to your browser in the HTTP header. They are returned with an HTTP response message every time a request is made.

It's usually only when something goes wrong that you might see one displayed in your browser. A common status code that you may have discovered in your daily dealings in a browser is the **404 page not found error**:



HTTP status codes fall into five classes:

1. **100s**: Informational codes indicating that the request initiated by the browser is continuing.
2. **200s**: Success codes returned when the browser request was received, understood, and processed by the server.
3. **300s**: Redirection codes returned when a new resource has been substituted for the requested resource.
4. **400s**: Client error codes indicating that there was a problem with the request.
5. **500s**: Server error codes indicating that the request was accepted but an error on the server prevented the fulfilment of the request.

Extra resource

Familiarise yourself with the types of **HTTP response status codes**.

MIME types

A Multipurpose Internet Mail Extension, or MIME type, is an Internet standard that describes the contents of Internet files based on their natures and formats. Nowadays, they are more frequently called media types because they are no longer only used in emails, but also in web applications.

This cataloguing helps the browser open the file with the appropriate extension or plugin and also helps a server identify how to process a specific file if it is sent from client to server. Although the term includes the word “mail” for electronic mail, it's used for web pages too.

MIME types contain two parts: a type and a subtype. The type describes the categorisation of MIME types linked to each other. In contrast, a subtype is unique to a specific file type that is part of the larger type category.

The syntax of the structure is: `type/subtype.`

A MIME type is also insensitive to text case, but they are traditionally written in lowercase.

Discrete types indicate the category of the document, and this can be one of the following:

Type	Description	Examples of typical subtypes
text	Represents any document that contains text and is theoretically human-readable.	• text/plain • text/html • text/css • text/javascript
image	Represents any kind of image. Videos are not included, though animated images (like an animated GIF) are described with an image type.	• image/gif • image/png • image/jpeg • image/bmp • image/webp
audio	Represents any kind of audio file.	• audio/midi • audio/mpeg • audio/webm • audio/ogg • audio/wav
video	Represents any kind of video file.	• video/webm • video/ogg
application	Represents any kind of binary data.	• application/octet-stream • application/pkcs12 • application/vnd.mspowerpoint • application/xhtml+xml • application/xml • application/pdf

MIME types (Mozilla Developer Network, 2016)

Here is a **list of all MIME types**.

Virtual machines

Virtual machines (VMs) can be installed on your local computer or in the cloud, providing flexibility and isolation for various tasks. For some of the upcoming tasks, you will need to set up a local virtual machine to use Linux command line tools, particularly on Kali Linux, which is widely used for penetration testing and security research. Running Kali Linux in a VM allows you to work within a dedicated environment without affecting your main operating system, ensuring that your primary operating system remains untouched.

To help you with this setup, detailed instructions for **local virtual machine installation** are available in the additional reading materials. These instructions will guide you through setting up your virtual machine and configuring Kali Linux, enabling you to create a controlled space for safe experimentation and practice.

Take note

The task(s) below is/are **auto-graded**. An auto-graded task still counts towards your progression and graduation. Give it your best attempt and submit it when you are ready.

When you select “Request Review”, the task is automatically complete, you do not need to wait for it to be reviewed by a mentor.

You will then receive an email with a link to a model answer, as well as an overview of the approach taken to reach this answer.

Take some time to review and compare your work against the model answer. This exercise will help solidify your understanding and provide an opportunity for reflection on how to apply these concepts in future projects.

In the same email, you will also receive a link to a survey, which you can use to self-assess your submission.

Once you’ve done that, feel free to progress to the next task.

Auto-graded task

Create a file named **answers.txt** and answer the following questions. Please note that some research may need to be done to answer these questions, and you will be required to read through the resources linked in this document as well. This is a vital part of the learning process, and if done properly will significantly improve your understanding of the topic.

1. The "P" in HTTP means “Protocol”. Explain in your own words what a protocol is.
2. Which protocol is HTTP built on top of?
3. If a request is successful, what will the status code of the response be?
4. What is a stateless protocol?
5. SSH is encrypted. What is the encrypted equivalent of HTTP?
6. What happens if there is a key mismatch between the client and server while establishing an SSH connection?
7. Which of the following are valid MIME types?
    a. text/time
    b. image/jpeg
    c. text/javascript
    d. text/jsx
    e. image/psd
    f. text/calendar
8. Which status message can you expect if a server does not allow you permission to request a specific resource?

Important: Be sure to upload all files required for the task submission inside your task folder and then click "Request review" on your dashboard.

Share your thoughts

Please take some time to complete this short feedback **form** to help us ensure we provide you with the best possible learning experience.

Reference list

Mozilla Developer Network. (2016). *MIME types.*
https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types