# 第5章 Web系统安全

Web系统是目前互联网上最普遍的应用系统，其使用的核心协议HTTP(HyperText Transfer Protocol)和HTTPS(Hypertext Transfer Protocol Secure)是TCP/IP协议簇的应用层协议。随着Web系统的普及和应用范围的扩大，Web系统的安全问题日益凸显。Web系统中存在的各种脆弱点和漏洞，可能被黑客利用进行恶意活动，威胁互联网用户的信息和财产安全。了解和掌握Web系统的常见威胁与防御手段，是网络安全领域的重要课题之一。本章主要介绍Web系统的组成与工作原理、Web系统面临的常见威胁、主要安全机制、常见攻击原理及防御手段，并通过具体的Web攻击实验使读者掌握Web系统安全的实践技能。

## 5.1 Web系统及HTTP

Web系统遵循B/S架构(Browser/Server架构，即浏览器/服务器架构)，主要由Web浏览器(如Edge、Chrome、Safari、Firefox、Opera等)、Web服务器(如Apache、Nginx、IIS、Caddy等)和核心交互协议HTTP/HTTPS组成。HTTP(HyperText Transfer Protocol，超文本传输协议)定义了Web浏览器和Web服务器之间请求和响应的消息格式和规则，HTTPS(Hypertext Transfer Protocol Secure，超文本传输安全协议)通过使用传输层安全性协议(TLS或其前身SSL)确保数据传输的安全性。整个Web系统的工作过程可以描述为：用户首先通过浏览器向服务器发送请求，随后服务器处理该请求并返回页面内容，最终由浏览器渲染并呈现页面内容给用户。

Web服务器通过HTTP或HTTPS向浏览器传递的网页或文件最初都是HTML(HyperText Markup Language)格式的。HTML是一种用于创建网页的标记语言，它由蒂姆·伯纳斯-李(Tim Berners-Lee)于1989年发明，同时，他还设计了最初的Web浏览器和Web服务器，使用户能通过浏览器访问服务器上的HTML文档，实现信息的浏览和交流。可以说，HTML的发明奠定了万维网(World Wide Web，WWW)的基础，推动了WWW的快速普及和发展。在现代互联网体系中，WWW和Web这两个术语在某种程度上是互相代指的。随着技术的进步，现代Web系统传输的媒体格式更加丰富多彩，可以直接传输图片、视频、压缩文件、JSON等多种不同文件格式的文件，也可以使用HTML或XML(Extensible Markup Language，可扩展标记语言)将不同类型的媒体内容(如图片、视频、音

频、文本等)整合在同一个页面,令 Web 系统提供更丰富的用户体验。

考虑到 Web 系统涵盖的知识点较多,本节主要介绍 Web 客户端和服务器相关技术、在浏览器地址栏中输入的 URL(即访问 Web 系统的第一步)、HTTP、以及 HTTP 的状态管理和 Cookie 等内容,这些内容涉及的技术在 Web 系统中具有重要意义。本节最后提供了一个实际访问 Web 系统的例子,旨在帮助读者更好地理解 Web 系统的运作方式。

### 5.1.1　Web 客户端相关技术

Web 客户端通常指 Web 浏览器,它是一种用于访问互联网上的网页和信息的软件应用程序,也是用户浏览网页、获取信息和进行在线交互的主要工具。各大浏览器厂商为了争夺更多用户和市场份额,一直不断努力改进其浏览器的性能、功能和用户体验,这种竞争推动浏览器相关技术不断发展。本节介绍几种目前广泛使用的 Web 客户端相关技术,探讨其可能带来的网络安全风险。

**1. 层叠样式表**

层叠样式表(Cascading Style Sheets,CSS)使得 Web 页面的样式与内容分离,解决了早期 Web 页面在描述内容的同时还需描述页面结构和样式等带来的修改和维护难题,让网页开发者可以更容易地控制网页的外观和布局。比如,可以根据不同设备的屏幕尺寸和特性调整网页布局和样式,以及仅修改 CSS 文件即可实现对整个网站外观的改变等。通过统一的样式规则,CSS 也解决了同一页面在不同浏览器中显示不一致的问题。随着 CSS3(CSS 的第三个主要版本)的发展,其新增的包括动画、渐变和阴影等特性进一步扩展了网页的设计空间,使得开发者能更加灵活地实现各种视觉效果和交互效果,丰富了网页的设计表现力。

尽管 CSS 提供了强大的样式控制功能,但也带来一些安全风险。这些风险包括恶意注入、CSS 解析器解析漏洞,以及利用 CSS 解析器版本差异等问题。攻击者可以通过在 CSS 中注入恶意代码或规则改变网页外观,从而诱导用户执行恶意操作。浏览器的 CSS 解析器在解析过程中遇到语法错误、非标准 CSS 语法或处理转义字符出现错误时,会继续解析,并在下一个语法匹配的括号处重新开始解析,这种重新同步逻辑可能被攻击者利用来解析恶意 CSS 规则集。攻击者还可以利用 CSS 解析器版本的差异,导致低版本解析器在解析高版本 CSS 规则时出现错误,从而利用透明样式等技巧混淆用户或诱导用户点击,危害用户安全和隐私。

**2. JavaScript**

JavaScript 是一种开放的、解释性的跨平台脚本语言,能在各种操作系统和设备上运行。JavaScript 主要应用于网页开发,旨在赋予网页交互式功能,使页面不再仅限于静态展示,而是呈现复杂功能和交互性。JavaScript 可用于操作 DOM、与服务器进行数据交互、弹出新窗口、移动或关闭窗口、重定向页面,以及读取和写入 Cookie 等。如今,JavaScript 已成为构建复杂 Web 应用程序的核心技术之一。

由于 JavaScript 具有强大的功能和灵活性,其也成为网络攻击者利用的工具之一。常见的利用 JavaScript 实施的攻击主要包括用户隐私数据窃取和拒绝服务攻击。

在用户隐私数据窃取方面,常见的攻击手段包括跨站脚本(该攻击原理具体参见 5.4.2 节)、点击劫持和 JSON(JavaScript Object Notation)数据劫持。在跨站脚本攻击中,攻击者通过

在网页中插入恶意 JavaScript 代码，利用用户对受信任的网站的信任，使恶意脚本在用户浏览器中执行，进而达到修改页面内容、劫持会话、窃取用户敏感信息等恶意目的。在点击劫持攻击中，攻击者通过在网页上层创建一个透明或隐藏的框架，使用 JavaScript 将恶意链接添加到页面上，当用户以为自己点击的是正常页面时，实际上点击的却是恶意链接来实施网络攻击。在 JSON 数据劫持中，攻击者通过构建包含 JSON 数据的 URL，诱使用户访问该链接，这导致恶意 JavaScript 代码在用户浏览器中执行，最终可能使攻击者利用用户的真实身份获取在受信任网站上的敏感数据。此外，编写 JavaScript 代码时，如果没有对所有控制字符都进行转义处理，或者网页设计存在逻辑错误，如未正确验证输入、缺乏必要的安全标头等，或者没有妥善处理敏感数据，例如将敏感信息存储在客户端的 JavaScript 变量中，都可能导致用户的敏感数据泄露。

在拒绝服务攻击方面，常见的攻击手段包括制造 JavaScript 死循环、间歇式释放 CPU 资源和不断弹出新窗口等。这些攻击手段之所以能够实施，主要是因为攻击者利用了浏览器对 JavaScript 在同一执行环境中按顺序执行且外部事件无法中断代码执行的特性。在制造 JavaScript 死循环攻击中，攻击者通过在 Web 页面中编写 JavaScript 死循环，导致一些对 JavaScript 所用资源没有限制的浏览器甚至底层操作系统瘫痪，无法再响应用户的任何操作。在间歇式释放 CPU 资源攻击中，攻击者通过编写可以间歇性释放 CPU 资源的、不会触发浏览器死循环保护措施的恶意 JavaScript 实现拒绝服务攻击。在不断弹出新窗口的拒绝服务攻击中，攻击者通过 JavaScript 代码不断打开新的浏览器对话框、在旧的对话框被用户关掉后又立马弹出一个新的对话框的方式实现拒绝服务攻击，除非用户执行某个恶意下载操作，否则完全没有办法访问正常的用户界面，甚至没法关掉浏览器窗口或离开恶意页面。

**3. 插件**

浏览器插件通常是由第三方开发者开发的、可以被用户选择性安装和启用的浏览器额外功能或特性，其通过浏览器的插件系统安装，并在浏览器中运行，为用户提供定制化的功能和体验。常见的浏览器插件如用于不同格式文件呈现的插件，当用户在浏览网站时打开一个 PDF 文件，浏览器会以类似于显示常规 HTML 文件的方式展示该 PDF 文件。其他常见的浏览器插件功能包括视频播放、翻译、广告拦截等。

激活插件的常见方法是使用带有 type 参数的标签（比如 type＝"application/x-shockwave-flash"）指明所访问的文件类型，这样，当浏览器进行显示时，就会与在浏览器中注册的插件的 MIME（Multipurpose Internet Mail Extensions）类型进行匹配。若匹配成功，浏览器会将文件转给相应的插件进行处理；若没有匹配成功，理论上会出现一个提示窗口，让用户下载插件，或者浏览器会检查 HTTP 响应报文中的 Content-Type 头或 URL 中的文件后缀来确定所要显示的文件的类型。

浏览器的很多插件实际上都有自己完整的代码运行环境，这些可执行的插件程序在与 Web 服务器交互时拥有一定的特权，这带来很多安全风险：一些插件可能会收集用户的浏览历史、搜索习惯、个人信息，用于不正当的广告推送，甚至身份盗窃等；一些插件会被设计为恶意软件，安装后会导致弹出恶意广告、重定向到恶意网站，或下载更多的恶意软件等。

### 5.1.2　Web 服务器相关技术

Web 服务器是一种软件或计算机系统，其接收来自 Web 客户端（如浏览器）的 HTTP/HTTPS 请求，这些请求通常是针对特定网页或资源的 URL，然后根据请求的内容和配置处理请求，可能涉及读取文件、执行脚本、查询数据库等操作，最后将请求处理的结果生成 HTTP/HTTPS 响应并发送回客户端，以便客户端显示请求的内容。为了提高服务器性能、降低网络传输延迟、提高用户访问速度、实现个性化服务等，已有很多 Web 服务器相关技术被广泛应用。尽管这些技术在 HTTP 系列标准规范中都包含，但它们的具体实施细节并没有在标准规范中详细规定，这也带来许多安全风险。本节介绍几种目前广泛采用的 Web 服务器相关技术及其可能带来的安全风险。

**1. 持续对话**

Web 服务器的持续对话（Keep-Alive）是一种机制，允许客户端与服务器之间建立一种持久的连接，即在单个连接上可以发送多个 HTTP 请求和响应，而不是为每个请求均建立一个新的连接。持续对话机制减少了频繁建立和关闭连接所消耗的资源，降低了 HTTP 请求和响应之间的延迟，提高了网络传输效率，可以说，有效地优化了 Web 系统的性能。持续对话机制一般是通过 HTTP/1.1 中的“Connection：keep-alive”头实现的，这告诉服务器在发送响应后保持连接打开，以便后续请求可以在同一连接上处理。HTTP/2 协议进一步提供了更高效的多路复用机制，允许在同一连接上并行发送多个请求和响应，进一步提高了 Web 系统的性能。

尽管持续对话机制提升了 Web 系统的性能，但这种机制也在 HTTP 请求和响应拆分时增加了脆弱点。比如，若 Web 服务器给出的 HTTP 响应头中使用多个 Content-Length 字段，Web 客户端可能会将该响应与不同的请求而不是同一个请求对应起来而导致客户端显示混乱；或者，在使用 Web 代理服务器的情况下，Web 代理服务器在收到来自 Web 服务器的某个持续对话的响应后，可能会错误地在全局进行缓存，导致将缓存内容发送给其他用户。

**2. 分块传输编码**

分块传输编码是指 Web 服务器将响应的内容分成一系列数据块（chunk），逐块发送给客户端，而不需要等整个响应内容准备就绪后再发送。这项技术便于服务器和客户端更好地控制和处理数据，同时也减少了客户端等待的时间，非常适用于需要动态生成、大量数据响应、在线视频等实时流式传输数据等场景。服务器使用分块数据传输时，会在 HTTP 响应头中包含 Transfer-Encoding：chunked 字段，以表示响应将以分块形式进行传输，在这种情况下，服务器事先并不知道整个要返回的数据文件长度，对于每个要返回的数据块，服务器会计算该数据块的长度，并将长度在响应头中进行标识，当整个数据文件传输完成后，会用一个长度为 0 的字节标识。

虽然分块传输机制有助于提高 Web 系统对动态内容的处理和传输效率，但也带来一些安全风险。例如，过多的数据分块可能导致浏览器代码中的整数溢出问题，或者利用设置在 HTTP 响应头中的 Content-Length 和分块长度不匹配的方式造成浏览器展示的混乱，导致请求走私（HTTP Request Smuggling）等攻击。

**3. Web 缓存**

Web 缓存器一般也称为代理服务器，即它代表初始的 Web 服务器来缓存和传递经常

请求的 Web 页面或资源。代理服务器可以缓存已经获取的内容，并在后续请求中直接提供缓存的副本，从而减少对原始服务器的访问次数，降低网络延迟，减轻服务器负担，并提高整体性能和用户体验。Web 缓存器可以是距离用户位置较近的网关服务器，也可以是大型的内容分发网络(CDN)的缓存服务器。

尽管 Web 缓存有利于提高 Web 系统性能、节省网络带宽，但其也带来一些安全风险。比如，被缓存的内容只要请求的方法和目标 URL 一致，即使如 Cookie 值等其他参数不一致，仍可以重用缓存，这导致被缓存数据的非授权访问风险。通过 Data/If-Modifaied-Since 和 ETag/If-None-Match 响应/请求头的搭配，再结合使用 Catch-Control：Private 响应头，可以获得用户在某一时间段内的访问行为和使用习惯。在 HTTP 响应头中利用 Expires、Date、Cache-Control 等不同字段设置相互冲突的缓存指示，也会导致浏览器处理的异常。使用缓存的另一个风险是，缓存本身也存在被攻击者注入的风险，导致用户通过浏览器访问的根本不是其目标网站，而是攻击者注入的伪造的恶意网站。

**4. 认证**

Web 服务器的认证是对用户或客户端的身份进行验证，以确保其有权限访问服务器上存储的受保护的资源或服务。一些常见的认证方法包括要求用户提供用户名和密码的基本认证、使用摘要算法对密码进行哈希的摘要认证、SSL/TLS 数字证书的认证、基于 JWT 标准的 Token 认证(RFC 7519)、基于时间和密码的双因子认证等。通过认证技术，Web 服务器可以确保只有经过授权的用户能访问受保护的资源，从而提高系统的安全性。

尽管目前使用 HTTP 协议本身认证机制的 Web 服务器不多，但攻击者仍可利用 Web 认证机制获取用户的隐私信息。比如，攻击者可通过修改服务器返回的 HTTP 响应头中的摘要认证方式为基本认证方式，以便在用户输入认证信息后获得用户的明文密码；或者通过在论坛上放置一个引用外部链接的图片，该图片来自由攻击者控制的 Web 服务器且需要认证，当同一论坛的用户浏览到该图片时，将被要求输入验证密码，基于对论坛的信任，用户很大可能会填写密码，该密码将被发送至攻击者控制的服务器上。

### 5.1.3 统一资源定位符

统一资源定位符(Uniform Resource Locator，URL)是一种特定格式的字符串，用于标识互联网资源的地址。浏览网站时，浏览器地址栏中的网址其实就是一种 URL，呈现在浏览器中的网页则是该 URL 对应的互联网资源。URL 的格式通常如下。

```
协议://主机名[:端口号]/路径/[?查询字符串]
```

其中各字段的含义如下。

- 协议(Protocol)指定了访问该资源需要使用的协议，可以是 HTTP、HTTPS、FTP 等。
- 主机名(Host)标识了资源所在的主机，可以是域名或 IP 地址，如 seclab.online。
- 端口号(Port)是可选的，标识了用于与服务器通信的端口。如果未提供，则使用协议的默认端口号，如 HTTP 的默认端口号是 80，HTTPS 的默认端口号是 443。
- 路径(Path)指定资源在主机上的位置，可以是文件路径、目录路径或其他标志符。
- 查询字符串(Query)是可选的，用于指定向服务器传递的额外参数。查询的内容通常是以 & 分隔的多个键值对，其中每个键值对的格式为 key=value。

下面是一个典型的 URL 的例子：

```
http://music.a.com/index.php?id=1
```

这个例子中，协议是 http，主机名为 music.a.com，端口号为 80，路径为/index.php，查询字符串为 id=1。实际上，对于 music.a.com 而言，/index.php 对应一个搜索结果页面，而查询参数 id=1 指定了具体的搜索关键词。在浏览器地址栏中输入该 URL，将获得 music.a.com 网站针对“id=1”的搜索结果。

### 5.1.4　HTTP

HTTP(Hyper Text Transfer Protocol，超文本传输协议)是 Web 系统的应用层协议，也是当前互联网上应用较广泛的网络协议之一。该协议的主要目标是提供一种标准的通信方式，使 Web 客户端(如浏览器)与 Web 服务器之间能交换数据，它是 Web 系统的基石。

**1. HTTP 简介**

HTTP 的工作方式是请求-响应式的。典型的 HTTP 请求过程如图 5-1 所示，包括如下 3 个步骤。

(1) Web 客户端(以下简称客户端)发送 HTTP 请求到 Web 服务器(以下简称服务器)，请求中包含了要获取的资源的信息，如 URL、请求方法(如 GET()、POST()等)、请求头部等。

(2) 服务器接收到请求后，根据请求的内容进行处理，然后生成 HTTP 响应。响应中包含了响应头部信息，以及请求的结果或资源。

(3) 服务器将响应发送回客户端，客户端接收到响应后进行解析，然后根据响应中的数据进行相应的处理，例如显示网页内容或执行其他操作。

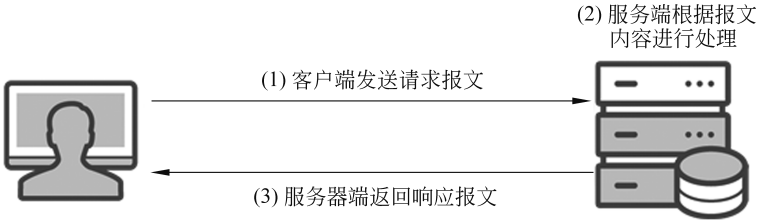


图 5-1　典型的 HTTP 请求过程

HTTP 在网页的加载和交互过程中起着重要作用。一个网页通常包含多种静态资源，包括包含页面内容的 HTML 文档、指定页面样式的层叠样式表(CSS)、实现页面动态功能的 JavaScript 脚本，以及各种多媒体资源等。此外，网页中的 JavaScript 脚本也会利用 HTTP 与服务器进行通信以获取必要数据。完整渲染一个页面通常需要几条甚至几十条 HTTP 请求。

值得注意的是，HTTP 中的“客户端”与“服务器”是相对的概念，仅用于区分 Web 服务的发起方与应答方。在某些情况下，一个应用程序可能既是客户端，又是服务器。例如，作为 Web 缓存器的 HTTP 代理服务器，在收到的用户请求文件未在缓存中存在时，其会将请求转发到目标服务器，然后将响应转发给原始客户端。这种情况下，HTTP 代理服务器既扮演了客户端的角色，又扮演了服务器的角色。图 5-2 中位于中间的 HTTP 代理服务器，

对于右侧的网站服务器而言，它是客户端；对于左侧的用户端的浏览器而言，它是服务器。

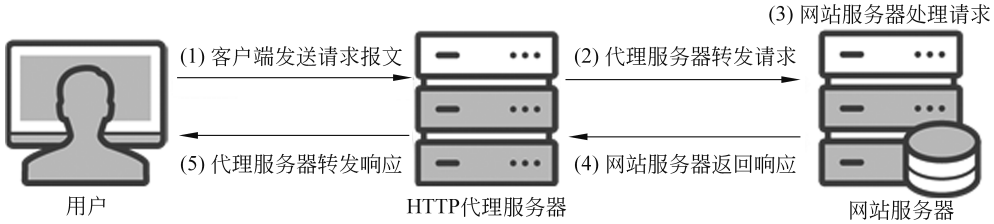


图 5-2　HTTP 代理服务器的工作过程

**2. HTTP 请求报文**

HTTP 请求报文是客户端向服务器发送请求时使用的数据格式，它由 3 部分组成：请求行、请求头和请求体(可选)。代码清单 5-1 以一个简单的例子展示了 HTTP 请求报文的一般格式。其中，第 1 行为请求行，其包含了请求的方法(如 GET)、所请求的资源(如/index.html)和 HTTP 的版本(如 HTTP/1.1)；第 2～6 行为请求头，其包含一系列键值对，用于传递请求的附加信息；最后为请求体，用于传递请求中需要发送的数据，如表单数据或 JSON 数据等，在该例子中请求体为空。

**代码清单 5-1　HTTP 请求报文格式示例**

```
1    GET /index.html HTTP/1.1
2    Host: music.a.com
3    User-Agent: Mozilla/5.0
4    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
5    Accept-Language: en-US,en;q=0.5
6    Connection: keep-alive
7
8    (请求体,对于 GET 请求通常为空)
```

1) HTTP 方法

HTTP 提供了方法字段用来区分不同目的请求。在 HTTP 请求中，客户端的目的多种多样，比如，获取资源(如图片、HTML 文本或 JavaScript 脚本等)、向服务器提交数据(如更新用户信息、上传文章等)，或者删除服务器上的某些资源等。与之对应，HTTP 提供的方法包括：GET()、POST()、OPTIONS()、HEAD()、PUT()、DELETE()等。其中，GET()、POST()是最常用的两个 HTTP 方法，此处简单介绍一下它们。

• GET()方法

GET()方法用于从服务器请求特定资源，它是最常用的 HTTP 方法之一，用于检索各种类型的资源，例如 HTML 页面、图像、视频和数据等。如果请求的 URL 标识了某个文件，那么服务器将文件的内容作为响应结果；如果请求的 URL 对应服务器端程序中的某个数据处理函数，那么服务器将返回该函数产生的数据。

• POST()方法

POST()方法用于向服务器提交数据，提交的数据包含在请求体中，通常用于创建或更新资源。它是最常用的 HTTP 方法之一，通常用于发布资源，例如在论坛、新闻组、评论区中发表评论，在视频网站上传视频、网站登录、填写问卷等。请求行中的请求对象通常对应服务器的某段数据处理程序或代码，程序对客户端提交的数据进行处理，服务器将处理结果

返回给客户端。例如，在一个网站登录请求中，客户端通过 POST 请求上传用户名和密码，服务器则利用登录功能的处理函数验证登录信息，创建相应的会话和 Cookie，最终通过响应报文将 Cookie 返回给客户端。

2）请求头

HTTP 中的请求头用于传递关于请求的相关信息。请求头中的每一项都是键值对，帮助服务器了解客户端的要求和处理请求的方式。表 5-1 列出了常见的请求头字段及含义。

**表 5-1　常见的请求头字段及含义**

| 请　求　头 | 含　　义 |
|---|---|
| User-Agent | 客户端使用的浏览器和操作系统信息 |
| Accept | 客户端能接收并处理的内容类型 |
| Content-Type | 请求体中的数据格式类型，取值与数据内容有关 |
| Accept-Encoding | 客户端支持的内容编码方式 |
| Authorization | 进行身份验证的凭证信息 |
| Cookie | 来自客户端的 Cookie 信息 |
| Referer | 当前请求是从哪个 URL 页面发起的 |
| Host | 服务器的域名或 IP 地址 |
| Content-Length | 请求体的长度 |
| Cache-Control | 控制缓存行为的指令，用于指定客户端和代理服务器如何缓存响应 |

3）请求体

请求体主要用于传递请求中需要发送的数据，其内容与请求的目的有关。请求体可以包含各种类型的数据，如表单数据、JSON 数据、XML 数据，甚至二进制数据等。请求体的格式通常由请求头的 Content-Type 字段指定，其内容长度由请求头中的 Content-Length 指定。表 5-2 列举了常见的请求体数据类型，以及它们对应的 Content-Type。

**表 5-2　常见的请求体数据类型**

| 数 据 类 型 | 用　　途 | Content-Type |
|---|---|---|
| 表单数据 | 传递 POST()方法的参数及上传的文件 | application/x-www-form-urlencoded 或 multipart/form-data |
| JSON 数据 | 传递 JSON 类型的数据 | application/json |
| XML 数据 | 传递 XML 类型的数据 | application/xml |
| 二进制数据 | 传递图像、音频、视频或其他类型的文件 | application/octet-stream |

除上述格式外，Web 应用还可以根据自身需求传输其他自定义格式的数据。

4）一个较为复杂的 HTTP 请求示例

代码清单 5-2 给出了使用 12345678910 作为用户名和密码登录真实网站 https://seclab.online 时浏览器发送的 HTTP 请求报文。出于隐私考虑，隐去了请求报文中的大部分 Cookie 字段。

**代码清单 5-2　一个较为复杂的 HTTP 请求示例**

```
1  POST /api/user/account/login HTTP/1.1
2  Accept: */*
3  Accept-Encoding: gzip, deflate, br
4  Accept-Language: en-US,en;q=0.9,zh-CN;q=0.8,zh-TW;q=0.7,zh;q=0.6,pt;q=0.5
5  Content-Length: 61
6  Content-Type: application/x-www-form-urlencoded
7  Cookie: team_token=; think_language=en-US; PHPSESSID=
   02u549lb2u1695fkvmlfhuf17n; token=
8  Origin: https:// seclab.online
9  Referer: https:// seclab.online /
10 Sec-Ch-Ua: "Chromium";v="122", "Not(A:Brand";v="24", "Microsoft Edge";v="122"
11 Sec-Ch-Ua-Mobile: ?0
12 Sec-Ch-Ua-Platform: "Windows"
13 Sec-Fetch-Dest: empty
14 Sec-Fetch-Mode: cors
15 Sec-Fetch-Site: same-origin
16 User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
   (KHTML, like Gecko) Chrome/122.0.0.0 Safari/537.36 Edg/122.0.0.0
17
18 username=12345678910&password=12345678910&login_type=password
```

仔细观察上面的请求报文，可以发现：HTTP 报文的第 1 行是请求行，它指明此次请求使用了 POST 方法，访问的路径为 /api/user/account/login，HTTP 版本号为 1.1。在请求头中，Accept-Encoding 字段（第 3 行）指定了客户端支持 gzip、deflate 以及 br 这 3 种压缩算法；Content-Length 字段（第 5 行）指定了请求体的长度为 61 字节；Content-Type 字段（第 6 行）指定了请求体的数据的类型为 application/x-www-form-urlencoded，即经过 URL 编码的表单数据；User-Agent（第 16 行）指定了客户端的浏览器与操作系统版本。请求体（第 18 行）包含多个由 & 分隔的表单参数。从参数的名字和取值可以看出：username 参数与 password 参数分别是用户名与密码，而 login_type 参数标志了登录类型。

**3. HTTP 响应报文**

HTTP 响应报文是服务器向客户端发送响应时使用的数据格式。与请求报文类似，它也由 3 部分组成：状态行、响应头和响应体（可选）。代码清单 5-3 以一个简单的例子展示了 HTTP 响应报文的一般格式。其中，第 1 行为状态行，其包含 HTTP 的版本（如 HTTP/1.1）、状态码（如 200）和状态码的描述短语（如 OK）；第 2～5 行为响应头，其包含一系列键值对，用于传递响应的附加信息；最后为响应体，为返回给客户端的实际数据，如 HTML、JSON 数据、XML、图像、音频、视频等文件，在该例子中响应体为长度为 900 字节的 HTML 文件。

**代码清单 5-3　HTTP 响应报文格式**

```
1  HTTP/1.1 200 OK
2  Date: Sun, 22 Aug 2024 12:00:00 GMT
3  Server: Apache
4  Content-Type: text/html
```

```
5   Content-Length: 900
6
7   (响应体,包含返回的 HTML 内容等)
```

1）HTTP 状态码

HTTP 状态码是表示请求的处理结果的 3 位数字代码，用于标识 HTTP 请求的处理状态，并为客户端提供有关请求处理结果的信息。根据表示的含义，状态码分为 5 类，分别以数字 1～5 开头。

• 1XX 信息响应

1XX 状态码表示请求已被接收并正在处理，但还需要进一步操作或等待客户端继续操作。101 是比较常见的状态码，用于应答客户端升级协议的请求，例如从 HTTP/1.1 升级为 WebSocket。

• 2XX 成功响应

此类状态码表示客户端的请求被成功接受和处理，如表 5-3 所示。

**表 5-3　用于 HTTP 响应的常见 2XX 状态码**

| 状 态 码 | 描 述 短 语 | 具 体 含 义 |
|---|---|---|
| 200 | OK | 表示请求已被成功处理，并返回了请求的资源。成功的含义取决于请求的方法，例如，对于 GET 请求，200 表示资源存在且在响应正文中传输 |
| 201 | Created | 表示请求成功，并因此创建了一个新的资源 |
| 202 | Accepted | 表示请求已收到，但尚未完成。服务器已开始处理请求，但需要时间才能完成。客户端应该继续轮询服务器，以获取最终结果，或者服务器在最终结果可用时发送另一个响应 |

• 3XX 重定向响应

3XX 状态码表示重定向，即客户端需要根据重定向的 URL 再次访问才能得到所需资源。

• 4XX 客户端错误响应

4XX 状态码表示客户端错误。当客户端发送的请求有问题或无法被服务器理解时，服务器会返回 4XX 状态码。客户端可能需要检查请求的参数、身份验证凭据和资源路径等，以确保自身发送了正确的请求。表 5-4 列出一些用于 HTTP 响应的常见 4XX 状态码。

**表 5-4　用于 HTTP 响应的常见 4XX 状态码**

| 状 态 码 | 描 述 短 语 | 具 体 含 义 |
|---|---|---|
| 400 | Bad Request | 客户端的请求存在错误，如报文格式错误、请求无效等，服务器无法处理该请求 |
| 401 | Unauthorized | 客户端必须进行身份验证，才能获得请求的响应 |
| 403 | Forbidden | 服务器知道客户端的身份，但客户端没有访问内容的权限 |
| 404 | Not Found | 服务器上找不到所请求的资源 |
| 405 | Method Not Allowed | 服务器知道请求方法，但目标资源不支持该方法。例如，服务器的 API 可能不支持 DELETE 来删除资源 |