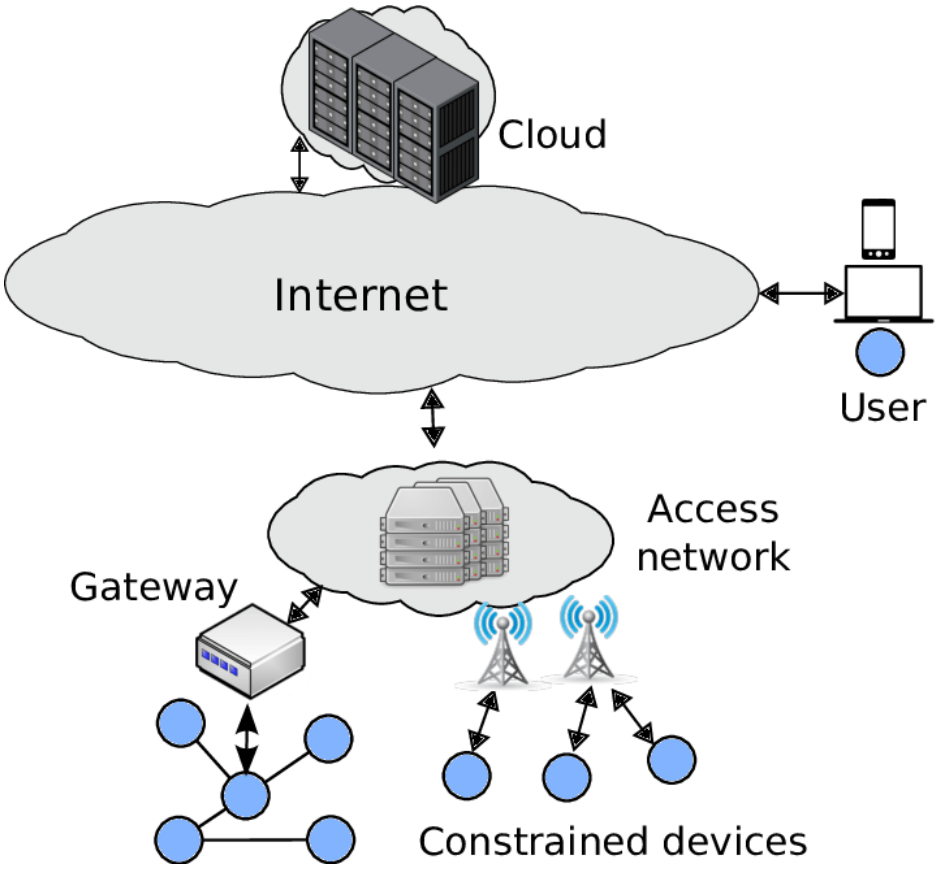
Introdução à Web Hacking

A exploração de aplicações web, também conhecido como *web hacking*, envolve identificar e explorar falhas de segurança em sites ou sistemas online, permitindo acesso não autorizado a dados, manipulação de funcionalidades ou interrupção de serviços. O conteúdo deste módulo é o pontapé inicial para quem deseja entender como funcionam as aplicações web e como podem ser exploradas.

Como a internet funciona

Para entender o web hacking, é essencial compreender como a internet e as redes funcionam. Uma rede (*network*) é um grupo de computadores conectados que trocam dados (informações), semelhante a um círculo social onde pessoas compartilham informações entre si.

A internet é uma rede global de redes, onde computadores e dispositivos estão interconectados, permitindo a troca de informações em escala mundial. Tanto é que a palavra "*internet*" vem do inglês "*interconnected network*", que significa rede interconectada.



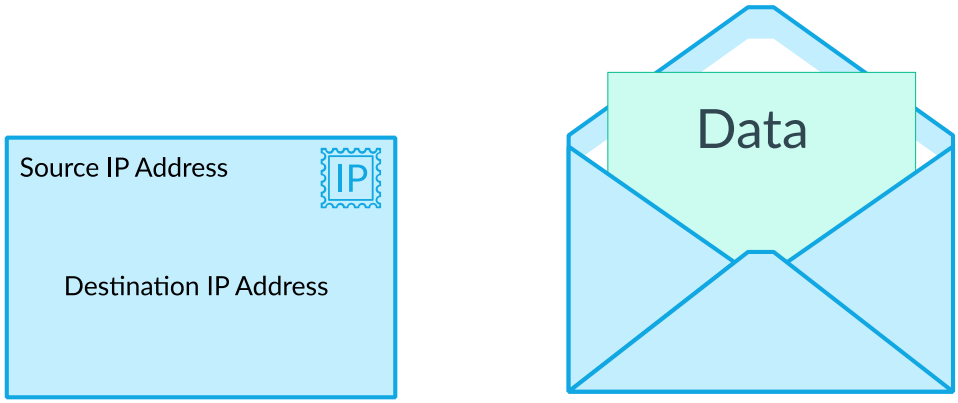
Para entender como os computadores se comunicam entre si, vamos pegar como exemplo o envio de uma carta. Quando você envia uma carta, ela passa por várias etapas: escrita, colocada em um envelope, endereçada, enviada pelo correio e finalmente entregue ao destinatário. Na internet, esse processo é semelhante, mas envolve protocolos e endereços IP.

Conceitos básicos de redes

Há alguns conceitos básicos que são fundamentais para entender como a internet funciona:

Pacotes

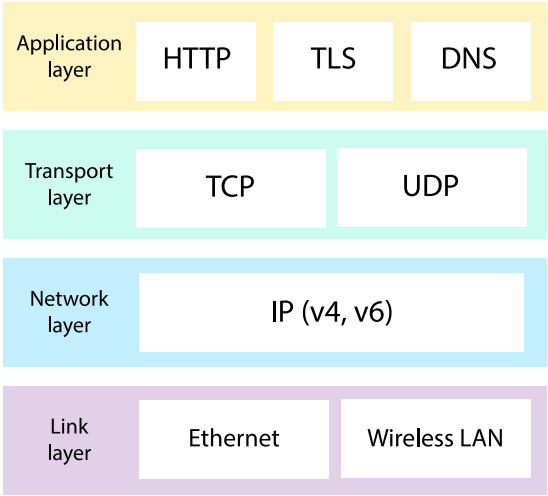
Um pacote é um **fragmento de dados** de uma mensagem que é enviado pela rede. Cada pacote contém informações sobre o remetente, destinatário e o conteúdo da mensagem, em uma região chamada "*header*", e isso permite que os dados sejam enviados de forma que o destinatário consiga entender o que deve ser feito com eles.



Quando você envia uma mensagem pela internet, ela é dividida em pacotes menores para facilitar o envio e a entrega. Esses pacotes são enviados por diferentes caminhos e, ao chegarem ao destino, são reagrupados para formar a mensagem original.

Protocolos

Um protocolo é um **conjunto de regras** que define como os dados são transmitidos e recebidos em uma rede para que dois dispositivos possam se entenderem. É como um **idioma comum** que todos os dispositivos usam para se comunicar. Existem protocolos para diferentes tipos de comunicação, como para dispositivos na mesma rede (*Ethernet*), para comunicação entre redes (*IP*), para pacotes que devem ser entregues em ordem (*TCP*) e para a troca de informações na web (*HTTP*).



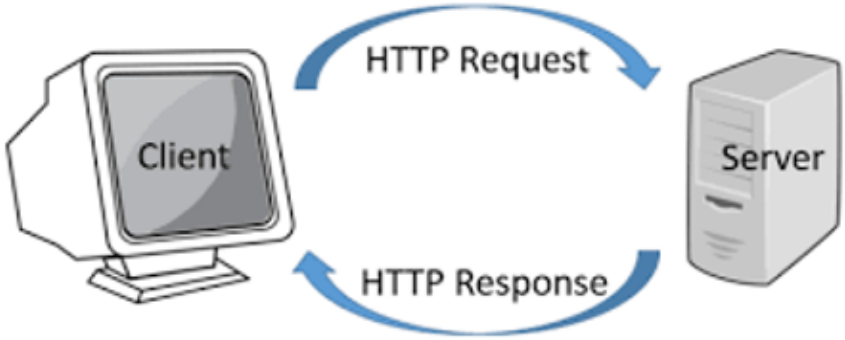
Protocolo HTTP

O **Hypertext Transfer Protocol** (RFC2616) é o protocolo que permite a troca de dados na web, sendo a base para aplicações web e APIs. Ele é um protocolo de camada de aplicação que define como as mensagens são

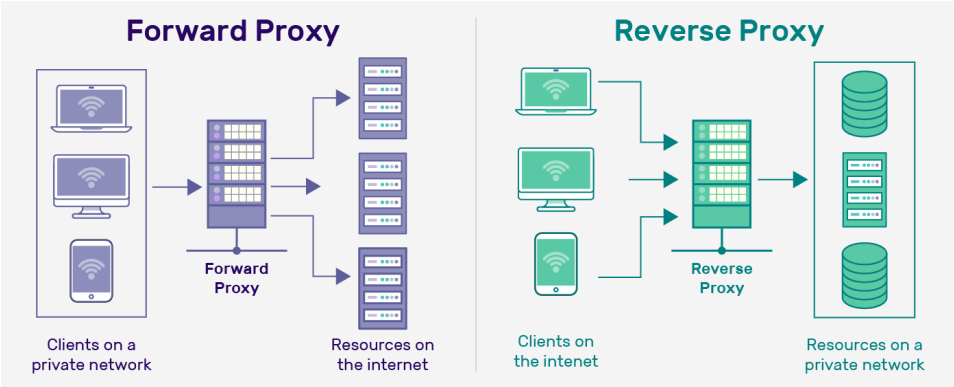
formatadas e transmitidas, e quais ações os servidores da web e os navegadores devem tomar em resposta a vários comandos.

Cliente e Servidor

A comunicação HTTP funciona em um modelo **cliente-servidor**. O cliente (geralmente seu navegador) envia uma requisição HTTP para um servidor (onde o site está hospedado). O servidor então processa essa requisição e envia de volta uma resposta HTTP.



- **Cliente (User-agent)**: É a entidade que inicia a comunicação, geralmente um navegador web, mas pode ser qualquer ferramenta que atue em nome do usuário (como curl, Postman). O cliente envia uma requisição HTTP a um servidor para obter um recurso ou executar uma ação.
- **Servidor Web**: É a entidade que recebe a requisição do cliente, processa-a e envia uma resposta HTTP de volta. O servidor hospeda os recursos (documentos, imagens, etc.) ou é capaz de gerá-los dinamicamente.
- **Proxies**: Entre o cliente e o servidor, podem existir intermediários chamados proxies. Eles podem desempenhar várias funções, como cache, filtragem, balanceamento de carga, autenticação e registro.



Evolução do HTTP

O HTTP passou por uma evolução significativa desde sua criação para atender às crescentes demandas da web. As principais versões incluem:

- **HTTP/0.9**: A primeira versão, muito simples, com requisições de uma única linha (apenas o método `GET` e o caminho do recurso) e sem cabeçalhos ou códigos de status.
- **HTTP/1.0**: Introduziu conceitos como versionamento, códigos de status, cabeçalhos HTTP e a capacidade de transmitir outros tipos de documentos além de HTML (graças ao cabeçalho `Content-Type`). Cada requisição geralmente abria uma nova conexão TCP.

- **HTTP/1.1**: Trouxe melhorias significativas como conexões persistentes (reutilização de conexão), pipelining (permitindo enviar múltiplas requisições antes de receber as respostas), respostas em blocos (*chunked responses*), melhor controle de cache, negociação de conteúdo e o cabeçalho `Host` (permitindo a hospedagem de múltiplos domínios no mesmo IP).
- **HTTP/2**: Focado em melhorias de desempenho, introduziu um protocolo binário (em vez de textual), multiplexação de requisições e respostas sobre uma única conexão TCP, compressão de cabeçalhos (*HPACK*) e server push. Essas mudanças visaram reduzir a latência e melhorar a eficiência no uso dos recursos de rede, sem alterar a semântica fundamental do `HTTP/1.1`.
- **HTTP/3**: A mais recente evolução, utiliza o QUIC (um novo protocolo de transporte sobre UDP) em vez de TCP. Isso visa resolver problemas como o head-of-line blocking do TCP e reduzir a latência no estabelecimento de conexões, mantendo as mesmas semânticas do `HTTP/2`.

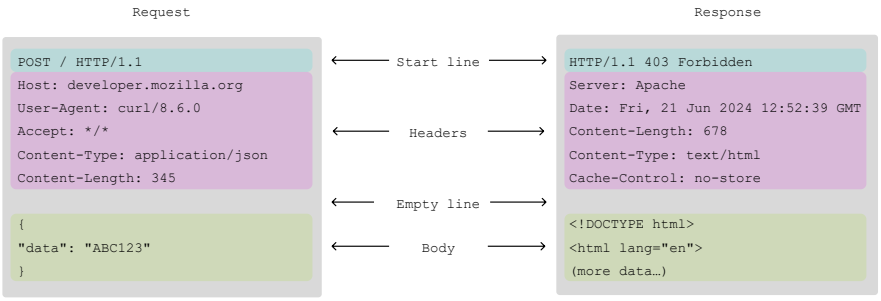
Mensagens HTTP

Existem dois tipos de mensagens HTTP: **requisições** (enviadas pelo cliente para acionar uma ação no servidor) e **respostas** (a resposta do servidor a uma requisição). Nas versões `HTTP/1.x`, essas mensagens são textuais e legíveis por humanos, enquanto no `HTTP/2` e `HTTP/3`, elas são encapsuladas em um formato binário para maior eficiência, embora a semântica subjacente permaneça a mesma.

Estrutura Geral da Mensagem HTTP

Tanto as requisições quanto as respostas `HTTP/1.1` compartilham uma estrutura básica semelhante :

- **Linha Inicial (Start-line)**: A primeira linha da mensagem.
  - Para requisições, é chamada de **request-line**.
  - Para respostas, é chamada de **status-line**.
- **Cabeçalhos HTTP (Headers)**: Zero ou mais linhas de cabeçalho que fornecem metadados sobre a mensagem ou o recurso. Cada cabeçalho consiste em um nome (*case-insensitive*) seguido por dois pontos (`:`) e um valor.
- **Linha em Branco**: Uma única linha em branco (*CRLF*) que indica o fim da seção de cabeçalhos.
- **Corpo da Mensagem (Message Body/Content)**: Uma parte opcional que contém os dados da mensagem (por exemplo, dados de um formulário em uma requisição `POST` ou o conteúdo de um recurso em uma resposta `GET`). A presença e o tipo do corpo são determinados pela linha inicial e pelos cabeçalhos.

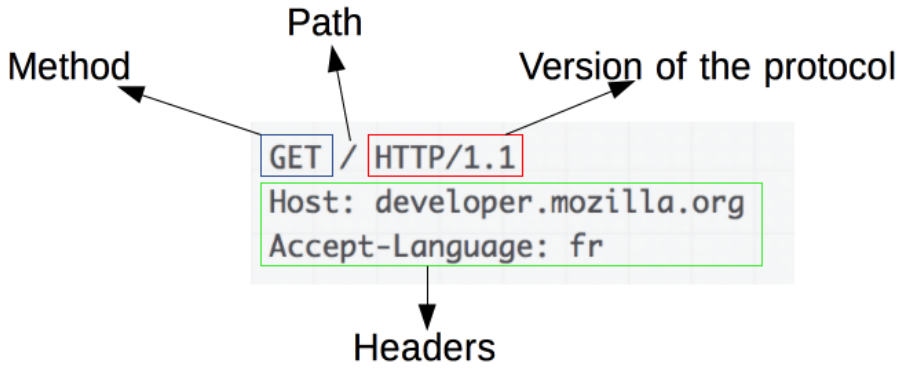


Linha de Requisição (Request-Line)

Contém três partes separadas por espaços:

- **Método HTTP (Method)**: Um verbo (`GET`, `POST`) que indica a ação desejada no recurso.

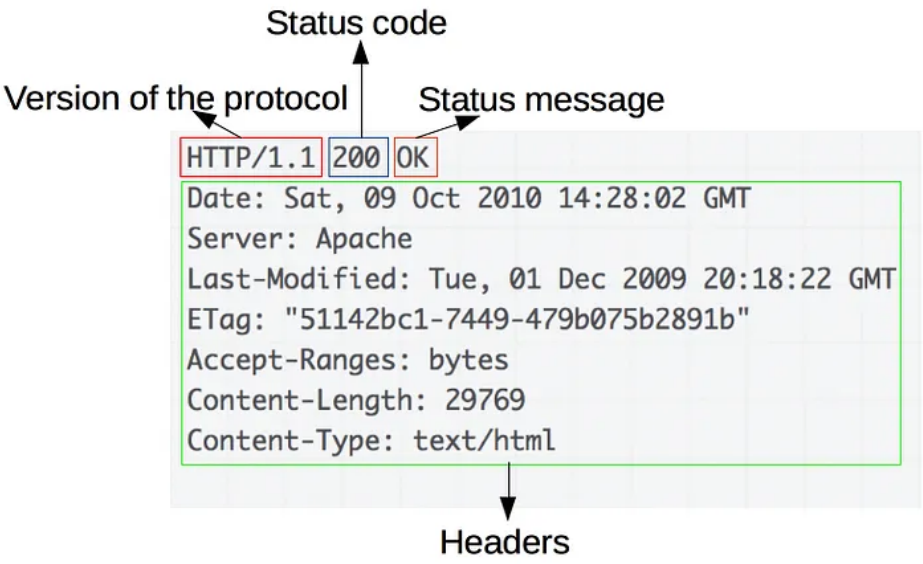
- **Alvo da Requisição (Request-Target)**: Geralmente uma URL (ou parte dela) que identifica o recurso ao qual a requisição se aplica. O formato do alvo depende do método e do contexto (*origin-form*, *absolute-form*).
- **Versão HTTP (HTTP Version)**: A versão do protocolo HTTP que o cliente está usando (ex: HTTP/1.1).



Linha de Status (Status-Line)

Contém três partes separadas por espaços:

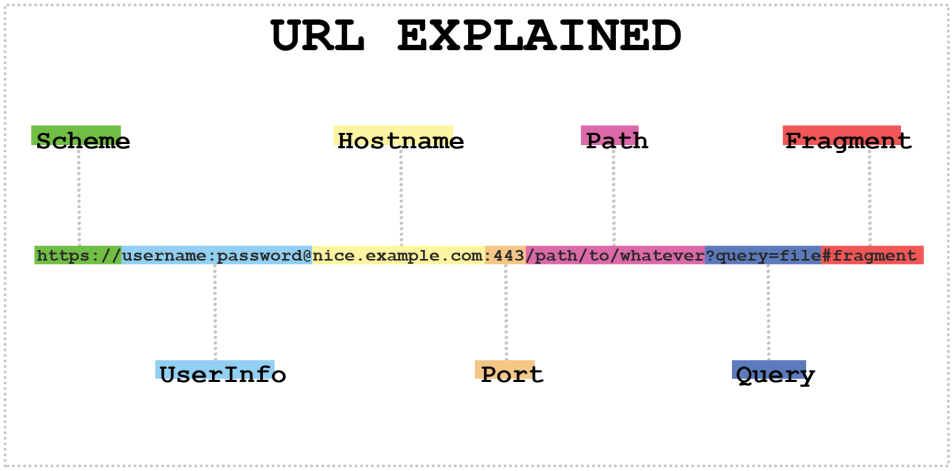
- **Versão HTTP (HTTP Version)**: A versão do protocolo HTTP que o servidor está usando (`HTTP/1.1`).
- **Código de Status (Status Code)**: Um código numérico de três dígitos que indica o resultado da requisição (200, 404).
- **Frase de Motivo (Reason Phrase)**: Uma breve descrição textual do código de status (OK, Not Found). Embora presente, os clientes devem ignorar a reason phrase, pois ela não é confiável para determinar o resultado.



URL

Uma URL é um tipo específico de `URI` **(Uniform Resource Identifier)** que, além de identificar um recurso, especifica como localizá-lo e o mecanismo para acessá-lo. No contexto do protocolo HTTP, a URL é fundamental, pois indica ao cliente qual recurso ele deseja acessar no servidor e como chegar até ele.

A estrutura de uma URL, para o uso com HTTP/HTTPS, geralmente se decompõe nos seguintes componentes:



Esquema (Scheme)

Define o protocolo a ser usado para acessar o recurso. Para o HTTP, os esquemas mais comuns são:

- `http`: Protocolo HTTP padrão, geralmente operando na porta 80.
- `https`: Protocolo HTTP seguro (HTTP sobre TLS/SSL), geralmente operando na porta 443.

O esquema é seguido por `://`. Exemplo: `http://`, `https://`

Autoridade (Authority)

Esta parte é precedida por `//` e contém informações para localizar o servidor. Ela pode incluir:

- **Anfitrião (Host)**: O nome de domínio (`www.exemplo.com`) ou o endereço IP (`192.168.1.1`) do servidor que hospeda o recurso. Este é o componente obrigatório dentro da autoridade.
- **Porta (Port)**: Um número de porta opcional que indica o "portão" específico no servidor ao qual a conexão deve ser feita. É separado do host por dois pontos (`:`). Se omitido, a porta padrão para o esquema é usada (80 para http e 443 para https).
- **Informações do Usuário (User Information - raro e desencorajado para senhas)**: Opcionalmente, pode conter um nome de usuário e senha, separados por dois pontos (`:`) e seguidos por um `@` antes do host (`usuario:senha@exemplo.com`). O uso de senhas nesta parte da URL é altamente desaconselhado por razões de segurança.

Caminho (Path)

Fornece a localização específica do recurso no servidor, seguindo uma estrutura hierárquica semelhante a um caminho de arquivo em um sistema operacional. Começa com uma barra (`/`) após a autoridade (ou após o esquema se não houver autoridade, o que é raro para HTTP).

String de Consulta (Query String)

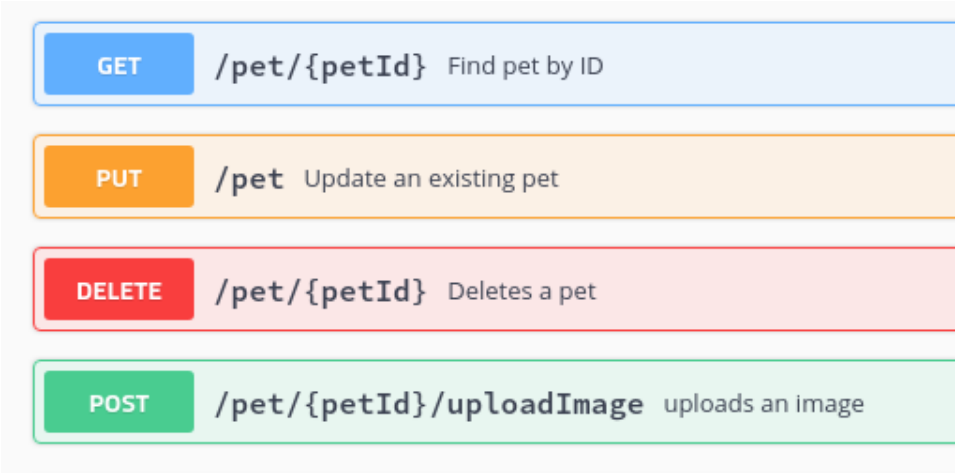
É uma parte opcional, precedida por um ponto de interrogação (`?`), que contém um conjunto de parâmetros (geralmente pares chave-valor) que são passados para o recurso no servidor. Múltiplos parâmetros são geralmente separados por um "e" comercial (`&`).

Fragmento (Fragment)

É uma parte opcional, precedida por uma cerquilha (#), que se refere a uma seção ou recurso secundário dentro do recurso principal. O fragmento é processado exclusivamente pelo cliente (geralmente o navegador web) após o recurso principal ter sido carregado. Ele não é enviado ao servidor na requisição HTTP.

Métodos HTTP

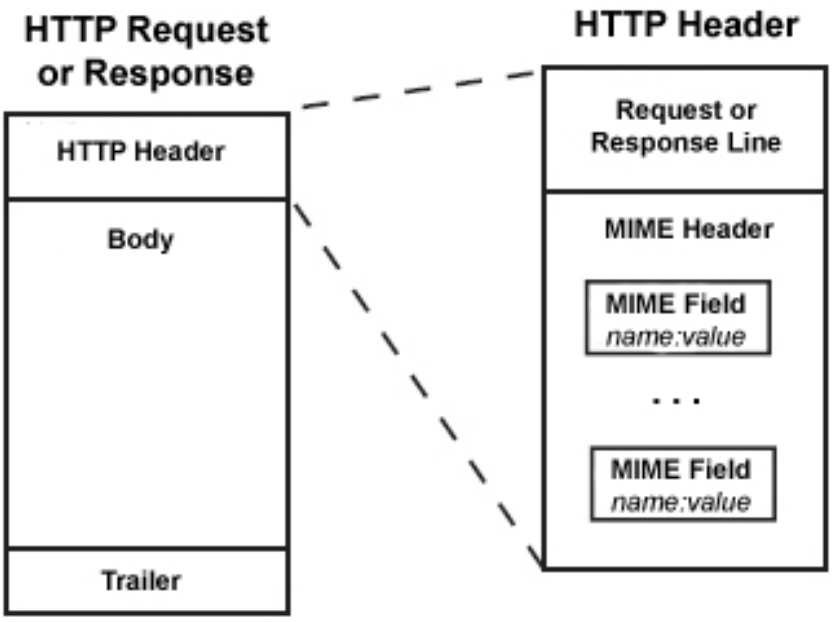
Os métodos HTTP, também conhecidos como verbos HTTP, indicam a ação desejada que o cliente solicita ao servidor para ser realizada no recurso identificado pelo request-target.



- **GET**: Solicita uma representação do recurso especificado. É o método mais comum, usado para buscar dados de um servidor, como carregar uma página da web ou uma imagem.
- **POST**: Envia dados para serem processados pelo recurso especificado. Frequentemente resulta na criação de um novo recurso ou na alteração do estado do servidor.
- **DELETE**: Remove o recurso especificado.
- **HEAD**: Solicita os cabeçalhos que seriam retornados se o recurso especificado fosse requisitado com um método `GET`. É idêntico ao `GET`, mas o servidor não envia o corpo da resposta.
- **PUT**: Cria um novo recurso ou substitui uma representação do recurso de destino com os dados do corpo da requisição. A URL de destino especifica o local exato do recurso.
- **OPTIONS**: Descreve as opções de comunicação (métodos HTTP permitidos, cabeçalhos suportados, etc.) para o recurso de destino. O navegador pode enviar uma requisição `OPTIONS` antes de uma requisição em cenários de **CORS (Cross-Origin Resource Sharing)** para verificar se a requisição real será permitida pelo servidor.
- **PATCH**: Aplica modificações parciais a um recurso. Diferente do PUT (que substitui o recurso inteiro), o PATCH aplica um conjunto de instruções de como modificar o recurso.

Cabeçalhos HTTP

Os cabeçalhos HTTP são pares `nome-valor` que transmitem metadados sobre a requisição, a resposta ou o próprio corpo da mensagem. Eles são uma parte crucial da extensibilidade do HTTP. Os nomes dos cabeçalhos não são sensíveis a maiúsculas e minúsculas. Existem centenas de cabeçalhos HTTP, padronizados e não padronizados.



https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Reference/Headers

Corpo da Mensagem

O corpo de uma mensagem HTTP é uma parte opcional da mensagem, tanto em requisições quanto em respostas. Ele é usado para transportar os dados associados à mensagem. Quando presente, geralmente é separado dos cabeçalhos por uma linha em branco (*CRLF*).

Tamanho do Corpo

É crucial que o receptor saiba como determinar o fim do corpo da mensagem para poder processá-la corretamente. Isso é geralmente feito através de dois cabeçalhos:

- `Content-Length`: Este cabeçalho especifica o tamanho do corpo da mensagem em bytes. Quando presente, o receptor lê exatamente esse número de bytes após a linha em branco que separa os cabeçalhos do corpo.
  - Exemplo: `Content-Length: 1024` indica que o corpo da mensagem tem 1024 bytes.
- `Transfer-Encoding: chunked`: Se o tamanho do corpo da mensagem não for conhecido antecipadamente (por exemplo, em streaming de dados gerados dinamicamente), o servidor pode usar a codificação de transferência em pedaços ("*chunked*").
  - Nesse caso, o corpo é enviado em uma série de "pedaços". Cada pedaço é precedido por seu tamanho em hexadecimal, seguido por *CRLF*, e então o próprio pedaço, seguido por outro *CRLF*.
  - A mensagem termina com um pedaço de tamanho zero, seguido por um *CRLF* final.

Tipos de Conteúdo (Media Types)

O cabeçalho `Content-Type` é fundamental quando uma mensagem tem um corpo. Ele informa ao receptor qual é o tipo de mídia dos dados no corpo, para que possa ser processado adequadamente. O valor deste cabeçalho é um "*Media Type*" (anteriormente conhecido como tipo **MIME**), consistindo em um tipo e um subtipo, e parâmetros opcionais (como charset).

Exemplos comuns:

- `text/html; charset=utf-8` (para documentos HTML)
- `application/json` (para dados em formato JSON)
- `application/xml` (para dados em formato XML)
- `image/jpeg` (para imagens JPEG)
- `image/png` (para imagens PNG)
- `application/octet-stream` (para dados binários arbitrários)
- `multipart/form-data` (usado para enviar formulários que contêm arquivos ou dados binários)
- `application/x-www-form-urlencoded` (para dados de formulário codificados como string de consulta)

Codificação de Conteúdo (Content Encoding)

O cabeçalho **`Content-Encoding`** especifica quais codificações foram aplicadas ao corpo da mensagem e, portanto, como os dados devem ser decodificados antes de serem usados. Isso é comumente usado para compressão.

Exemplos comuns:

- `gzip`
- `compress`
- `deflate`
- `br`

Se o corpo foi comprimido (exemplo com gzip), o cabeçalho `Content-Encoding: gzip` indicará isso, e o `Content-Length` se referirá ao tamanho do corpo comprimido. O receptor deve descomprimir o corpo antes de usar seu conteúdo.

Código de Status HTTP

Os códigos de status HTTP são respostas numéricas de três dígitos enviadas pelo servidor para indicar o resultado do processamento de uma requisição HTTP feita pelo cliente. Cada código de status é acompanhado por uma "*Reason Phrase*" (Frase de Razão) textual, que é uma breve descrição do status (por exemplo, "OK", "Not Found", "Internal Server Error"). A frase de razão é somente para leitura e não afeta o processamento do protocolo.

Os códigos de status são agrupados em cinco classes, identificadas pelo primeiro dígito:

1xx (Informacional)

Esta classe de códigos de status indica uma resposta provisória. A requisição foi recebida e o processo continua. Essas respostas consistem apenas na linha de status e cabeçalhos opcionais, e são terminadas por uma linha vazia. O cliente deve estar preparado para receber uma ou mais respostas 1xx antes de receber a resposta final.

- **100** `Continue`: Indica que a parte inicial da requisição foi recebida e o cliente deve continuar com a requisição ou ignorar esta resposta se a requisição já estiver completa. É usado para permitir que um cliente verifique se o servidor aceitará uma requisição (por exemplo, uma com um corpo grande) antes de enviá-la completamente.
- **101** `Switching Protocols`: O servidor entende e está disposto a cumprir a solicitação do cliente, através do cabeçalho `Upgrade`, para mudar o protocolo de aplicação sendo usado nesta conexão. O servidor incluirá um cabeçalho `Upgrade` na resposta para indicar para qual protocolo ele mudou.

2xx (Sucesso)

Esta classe de códigos de status indica que a requisição do cliente foi recebida, entendida, aceita e processada com sucesso.

- **200** `OK`: É a resposta padrão para requisições HTTP bem-sucedidas. O significado da resposta depende do método da requisição.
- **201** `Created`: A requisição foi bem-sucedida e um novo recurso foi criado como resultado. Este é tipicamente o código enviado após requisições `POST` ou algumas requisições `PUT`. A resposta geralmente inclui a *URI* do novo recurso no cabeçalho `Location` e/ou no corpo da resposta.
- **202** `Accepted`: A requisição foi aceita para processamento, mas o processamento ainda não foi concluído. A requisição pode ou não ser eventualmente atendida, pois pode ser desaprovada quando o processamento realmente ocorrer. Não há como o HTTP reenviar uma indicação assíncrona do resultado do processamento.
- **204** `No Content`: O servidor processou a requisição com sucesso, mas não há conteúdo para retornar no corpo da resposta. Isso é útil para casos onde a ação foi bem-sucedida, mas não há necessidade de retornar dados, como após uma requisição `DELETE` ou uma requisição `PUT` que atualiza um recurso sem alterar seu conteúdo visível. A resposta não deve incluir um corpo.

3xx (Redirecionamento)

Esta classe de códigos de status indica que o cliente precisa tomar ações adicionais para completar a requisição. Muitas dessas respostas são usadas em redirecionamento de URL. Um user agent pode realizar a ação adicional sem interação do usuário apenas se o método usado na segunda requisição for `GET` ou `HEAD`.

- **301** `Moved Permanently`: O recurso solicitado foi movido permanentemente para uma nova URI, e futuras referências a este recurso devem usar uma das URIs retornadas. A nova URI é fornecida no cabeçalho `Location`. Navegadores geralmente redirecionam automaticamente, e motores de busca atualizam seus links para o recurso.
- **302** `Found`: O recurso solicitado reside temporariamente sob uma URI diferente. Como o redirecionamento pode ser alterado ocasionalmente, o cliente deve continuar a usar a URI de requisição efetiva para futuras requisições. A nova URI é fornecida no cabeçalho `Location`.
- **304** `Not Modified`: Usado para fins de cache. Indica que o recurso não foi modificado desde a versão especificada pelos cabeçalhos condicionais da requisição (como `If-Modified-Since` ou `If-None-`

`Match`). O servidor informa ao cliente que ele pode usar sua cópia em cache do recurso. Esta resposta não deve conter um corpo.

- **307** `Temporary Redirect`: O recurso solicitado reside temporariamente sob uma URI diferente. O user agent não deve alterar o método HTTP usado: se um `POST` foi usado na primeira requisição, um `POST` deve ser usado na segunda requisição para a URI no cabeçalho `Location`.
- **308** `Permanent Redirect`: O recurso solicitado foi permanentemente movido para uma URI diferente. O user agent não deve alterar o método HTTP usado. Similar ao 301, mas mais estrito quanto à não alteração do método.

4xx (Erro do Cliente)

Esta classe de códigos de status indica que a requisição parece conter um erro do lado do cliente (por exemplo, sintaxe malformada da requisição, URI inválida, autenticação necessária, etc.) ou não pode ser processada pelo servidor como está. O servidor acredita que o cliente errou. Exceto quando respondendo a uma requisição HEAD, o servidor deve incluir uma entidade contendo uma explicação da situação de erro, e se é uma condição temporária ou permanente.

- **400** `Bad Request`: O servidor não pôde entender a requisição devido à sintaxe inválida. O cliente não deve repetir a requisição sem modificações.
- **401** `Unauthorized`: A requisição requer autenticação do usuário.
- **403** `Forbidden`: O servidor entendeu a requisição, mas se recusa a autorizá-la. Diferente do **401** `Unauthorized`, a identidade do cliente é conhecida pelo servidor, mas ele não tem permissão para acessar o recurso. A autenticação não fará diferença e a requisição não deve ser repetida.
- **404** `Not Found`: O servidor não encontrou uma representação atual para o recurso de destino ou não está disposto a divulgar que tal representação existe. Este é um dos códigos de erro mais comuns na web.
- **405** `Method Not Allowed`: O método especificado na linha de requisição (`GET`, `POST`, etc.) não é permitido para o recurso identificado pela URI. A resposta deve incluir um cabeçalho `Allow` contendo uma lista de métodos válidos para o recurso solicitado.
- **409** `Conflict`: Indica que a requisição não pôde ser processada devido a um conflito com o estado atual do recurso de destino. Este código é usado em situações onde o usuário pode ser capaz de resolver o conflito e reenviar a requisição (por exemplo, um `PUT` para criar um recurso que já existe de forma conflitante, ou edições simultâneas).
- **429** `Too Many Requests`: O usuário enviou muitas requisições em um determinado período ("*rate limiting*"). As respostas podem incluir um cabeçalho `Retry-After` indicando quanto tempo esperar antes de fazer uma nova requisição.

5xx (Erro do Servidor)

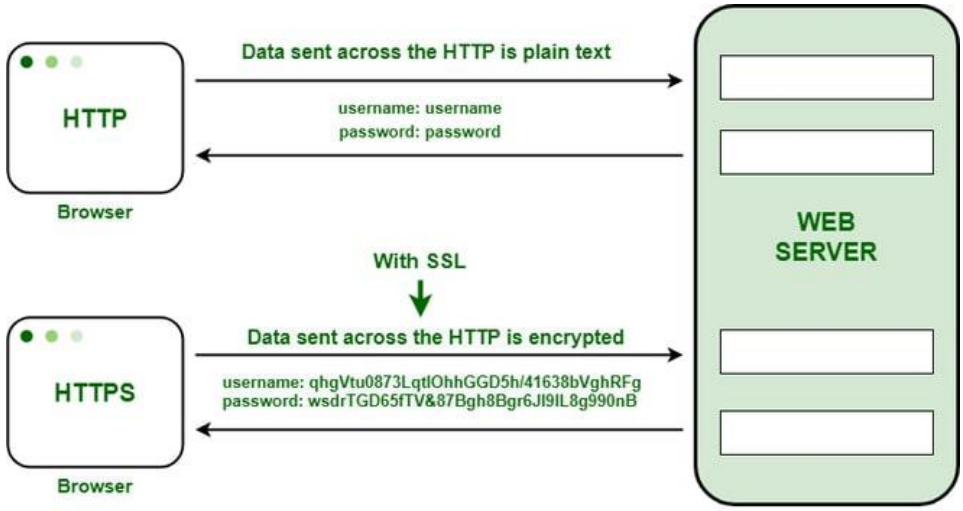
Esta classe de códigos de status indica que o servidor encontrou um erro ou está incapacitado de executar uma requisição aparentemente válida. O servidor está ciente de que errou ou é incapaz de realizar a requisição.

- **500** `Internal Server Error`: Uma mensagem genérica de erro, dada quando uma condição inesperada foi encontrada pelo servidor e nenhuma mensagem mais específica é adequada. Este é um erro "*catch-all*" do lado do servidor.
- **501** `Not Implemented`: O servidor não suporta a funcionalidade necessária para cumprir a requisição. Este é o status apropriado quando o servidor não reconhece o método da requisição e não é capaz de

suportá-lo para nenhum recurso.

- **502** `Bad Gateway`: O servidor, enquanto atuava como um gateway ou proxy, recebeu uma resposta inválida do servidor upstream ao qual acessou na tentativa de atender à requisição.
- **503** `Service Unavailable`: O servidor não está atualmente pronto para lidar com a requisição. Causas comuns são um servidor que está em manutenção ou que está sobrecarregado. Esta condição é geralmente temporária. A resposta pode incluir um cabeçalho `Retry-After` indicando ao cliente quanto tempo esperar antes de tentar novamente.
- **504** `Gateway Timeout`: O servidor, enquanto atuava como um gateway ou proxy, não recebeu uma resposta a tempo do servidor upstream ou de algum outro servidor auxiliar que precisava acessar para completar a requisição.

Segurança com HTTPS



O **HTTPS (Hypertext Transfer Protocol Secure)** não é um protocolo separado, mas sim o uso do HTTP sobre uma camada de segurança criptográfica, que é o **TLS (Transport Layer Security)** ou seu predecessor, **SSL (Secure Sockets Layer)**. Essa camada adicional é fundamental para proteger a comunicação na web, garantindo três pilares essenciais da segurança:

Confidencialidade (Confidentiality)

O TLS criptografa os dados trocados entre o cliente (navegador) e o servidor. Isso significa que, mesmo que um invasor consiga interceptar a comunicação, os dados estarão em um formato ilegível, protegendo informações sensíveis como credenciais de login, números de cartão de crédito, mensagens pessoais e outros dados privados contra espionagem. A criptografia é estabelecida durante um processo inicial chamado "*TLS handshake*", onde o cliente e o servidor negociam os algoritmos criptográficos e geram chaves de sessão secretas que são usadas para criptografar todas as mensagens HTTP subsequentes.

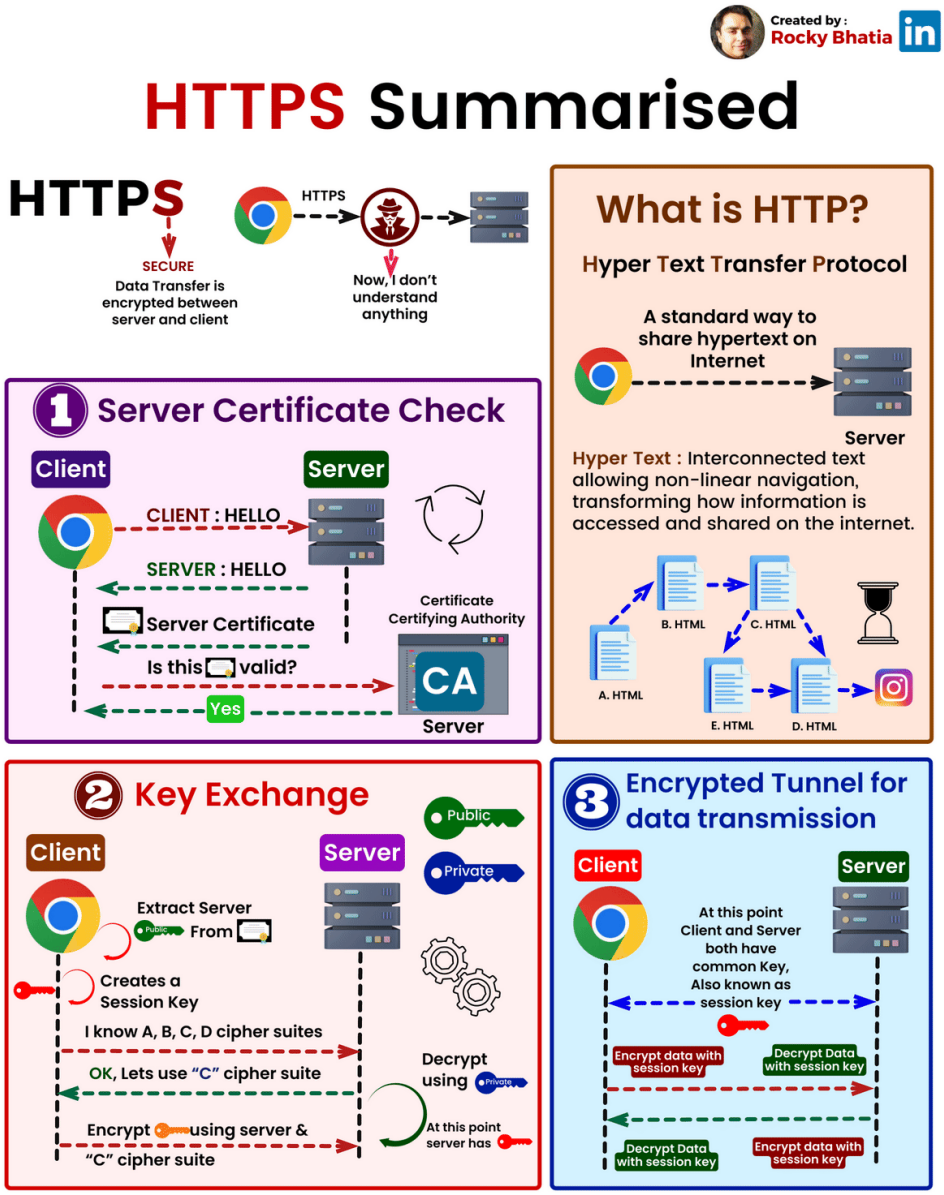
Integridade (Integrity)

O TLS garante que os dados enviados entre o cliente e o servidor não sejam alterados ou corrompidos durante a transmissão sem que a alteração seja detectada. Isso é alcançado através do uso de códigos de autenticação de mensagem (MACs) ou construções similares. Se um invasor tentar modificar uma mensagem HTTP em trânsito, o receptor será capaz de detectar essa adulteração, prevenindo que o usuário receba ou envie informações falsificadas.

Autenticação (Authentication)

O HTTPS, através do TLS, permite que o cliente verifique a identidade do servidor ao qual está se conectando. Isso é crucial para prevenir ataques *"man-in-the-middle"*, onde um invasor se passa pelo servidor legítimo. A autenticação do servidor é tipicamente realizada usando certificados digitais X.509. Esses certificados são emitidos por **Autoridades Certificadoras (CAs)** confiáveis e contêm a chave pública do servidor, informações sobre sua identidade e a assinatura digital da CA. O navegador do cliente verifica a validade do certificado (incluindo se ele não expirou, se foi emitido por uma CA confiável e se corresponde ao domínio acessado). Embora menos comum, o TLS também pode ser configurado para autenticar o cliente perante o servidor.

Como Funciona (De Forma Simplificada)



Quando você acessa um site usando `https://`, seu navegador e o servidor web executam as seguintes etapas antes que qualquer dado HTTP seja realmente trocado:

1. **Handshake TLS**: O cliente e o servidor negociam a versão do protocolo TLS a ser usada, os algoritmos de criptografia e trocam informações para gerar chaves de criptografia simétricas que serão usadas para a sessão. Durante este processo, o servidor apresenta seu certificado digital ao cliente.
2. **Verificação do Certificado**: O navegador verifica a autenticidade do certificado do servidor.

3. **Troca de Chaves Segura**: As chaves de sessão são estabelecidas de forma segura.
4. **Comunicação Criptografada**: Uma vez que o túnel TLS seguro é estabelecido, as mensagens HTTP (requisições e respostas, incluindo cabeçalhos e corpo) são criptografadas e enviadas através deste túnel. Para o protocolo HTTP em si, nada muda; ele opera da mesma forma, mas sua "conversa" é agora privada e protegida.

Introdução às vulnerabilidades web

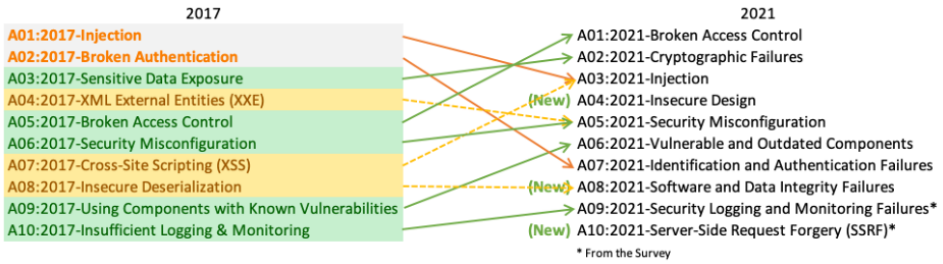
Apesar da robustez e flexibilidade do protocolo HTTP e das tecnologias web associadas, as aplicações web são alvos frequentes de ataques. As vulnerabilidades em sistemas web não surgem do protocolo HTTP em si, mas sim de falhas na forma como as aplicações são projetadas, desenvolvidas, configuradas e mantidas. Compreender a origem dessas brechas é o primeiro passo para construir defesas mais eficazes.

As vulnerabilidades podem se manifestar de diversas formas e ter causas variadas, incluindo:

- **Falhas de Implementação e Lógica de Código**: Muitos problemas de segurança originam-se diretamente no código-fonte da aplicação. Isso pode incluir lógica de programação defeituosa que não lida corretamente com todos os cenários possíveis, validação inadequada ou ausente de dados de entrada do usuário (permitindo ataques como *SQL Injection* ou *Cross-Site Scripting - XSS*), gerenciamento incorreto de sessões e autenticação, ou tratamento inadequado de erros que pode expor informações sensíveis.
- **Más Configurações (Misconfigurations)**: Uma aplicação pode ser construída com código seguro, mas ainda assim estar vulnerável devido a configurações inseguras do ambiente onde é executada. Isso abrange servidores web com configurações padrão permissivas, bancos de dados com senhas fracas ou expostos publicamente, frameworks com funcionalidades de depuração habilitadas em produção, cabeçalhos HTTP de segurança ausentes ou mal configurados, e políticas de controle de acesso falhas.
- **Gerenciamento Incorreto de Dependências**: Aplicações modernas raramente são construídas do zero. Elas dependem de uma gama de bibliotecas, frameworks e componentes de terceiros. Se essas dependências contiverem vulnerabilidades conhecidas e não forem atualizadas ou corrigidas, a aplicação como um todo herda esses riscos. A falta de um inventário claro de componentes e de um processo para monitorar e aplicar patches em dependências é uma causa comum de brechas.
- **Design Inseguro da Aplicação**: Às vezes, a própria arquitetura ou o design da aplicação podem introduzir fraquezas, mesmo que a implementação de componentes individuais seja correta. Isso pode incluir fluxos de autenticação frágeis, falta de segregação de privilégios adequada, ou a incapacidade de antecipar como diferentes funcionalidades podem ser abusadas quando combinadas.

Nesse contexto de conscientização e combate às vulnerabilidades web, destaca-se a **OWASP (Open Web Application Security Project)**. A OWASP é uma comunidade online global dedicada a encontrar e combater as causas que tornam o software inseguro. Ela produz materiais de referência, ferramentas, guias e documentos de forma aberta e colaborativa.

Um dos recursos mais conhecidos da OWASP é o **OWASP Top 10**, um documento de conscientização que lista os dez riscos de segurança mais críticos para aplicações web. Este ranking é atualizado periodicamente e serve como um guia fundamental para desenvolvedores, arquitetos de software e profissionais de segurança sobre onde concentrar seus esforços para mitigar as ameaças mais prevalentes. Além do Top 10, a OWASP oferece uma vasta gama de projetos, incluindo o **Application Security Verification Standard (ASVS)**, guias de teste, *CheatSheets* para desenvolvedores e ferramentas de análise.



Exercícios práticos

https://overthewire.org/wargames/natas/
https://tryhackme.com/path/outline/web
https://tryhackme.com/room/owasptop102021
https://owasp.org/www-project-juice-shop/
https://portswigger.net/web-security
https://play.picoctf.org/practice?category=1&page=1

Links úteis

How Does the Internet Work? - https://web.stanford.edu/class/msande91si/www-spr04/readings/week1/InternetWhitepaper.htm
What is Internet? - https://www.cloudflare.com/learning/network-layer/how-does-the-internet-work/
RFC2616 Definição do protocolo HTTP - https://datatracker.ietf.org/doc/html/rfc2616
Protocolo QUIC - https://blog.cloudflare.com/the-road-to-quic/
Vídeo HTTP/1.1 vs HTTP/2 vs HTTP/3 - https://youtu.be/ocGtt0IX0Js
Cabeçalhos HTTP - https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Reference/Headers
MIME types - https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Guides/MIME_types
Status HTTP - https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Reference/Status
Status HTTP mas com gatos - https://http.cat/
TLS/SSL - https://www.cloudflare.com/pt-br/learning/ssl/how-does-ssl-work/
Vídeo sobre TLS/SSL (SegInfoBrasil) - https://youtu.be/k4R8VOJ5mJg