Autenticação e Autorização com JavaScript

Bootcamp Desenvolvedor NODE.JS

Angelo Assis
Lucas Goulart

2021

Autenticação e Autorização com JavaScript

Bootcamp Desenvolvedor NODE.JS

Angelo Ferreira Assis

Lucas Goulart

Sumário

Capítulo 1. O Protocolo HTTP

O Capítulo 1 apresenta o HTTP, protocolo de transferência de hipertexto que está no coração da Web e da Internet. São apresentados: o modelo de arquitetura de comunicação; a pilha de protocolos utilizados; o serviço DNS; o conceito de cookies e sessões; a evolução para o protocolo HTTP 2.0; e os modelos de autenticação definidos pelo protocolo. Dependendo do nível de experiência e maturidade, alguns dos conceitos podem parecer básicos. Porém, todo o conteúdo apresentado é de fundamental importância para o prosseguimento da disciplina, uma vez que os conceitos mais avançados de autenticação e autorização fazem uso de recursos avançados do HTTP.

Protocolos de Comunicação

Nos últimos anos, tem sido observado o aumento do número de dispositivos de diferentes finalidades conectados à Internet. Sistemas como câmeras de segurança, laptops, TVs, webcams, automóveis, entre outros, estão sendo conectados de tal forma que o termo rede de computadores está começando a soar um tanto desatualizado, dados os muitos equipamentos não tradicionais que estão sendo ligados à Internet. Há alguns anos, eram basicamente computadores pessoais e servidores armazenando e transmitindo informações, como páginas e mensagens de e-mail. Hoje, a Internet é uma rede que conecta centenas de milhões de dispositivos de computação ao redor do mundo, e fornece meios para entretenimento, negócios e educação. Gerenciar essa teia de informações e serviços é função dos protocolos que controlam o envio e o recebimento de informações.

Quando quer perguntar as horas a alguém, as boas maneiras ditam que primeiro cumprimentamos a outra pessoa. Implicitamente, tomamos uma resposta cordial como uma indicação de que podemos prosseguir e perguntar as horas. Uma reação diferente poderia indicar falta de vontade ou incapacidade de comunicação. Nesse caso, o mais indicado seria não perguntar as horas. Ou seja: há mensagens específicas enviadas e ações específicas realizadas em reação às respostas recebidas ou a outros eventos (como nenhuma resposta após certo tempo).

Um protocolo de rede é semelhante a essa ideia. Todas as atividades na Internet que envolvem duas ou mais entidades remotas comunicando são governadas por um protocolo. Por exemplo, protocolos de controle de congestionamento controlam a taxa com que os pacotes são transmitidos entre a origem e o destino, protocolos em roteadores determinam o caminho de um pacote da origem ao destino.

Quando fazemos uma requisição a um servidor Web, por exemplo, digitamos a URL de uma página Web no browser. Primeiro, o computador enviará uma mensagem de requisição de conexão ao servidor Web e aguardará uma resposta. O servidor receberá essa mensagem de requisição de conexão e retornará uma mensagem de resposta de conexão. Sabendo que está tudo certo para requisitar o documento da Web, o computador envia então o nome da página Web desejada por meio de uma mensagem GET. Por fim, o servidor retorna um arquivo contendo a página para o computador.

Redes são organizadas em camadas. Cada protocolo pertence a uma das camadas. Uma camada oferece serviços à camada acima dela executando certas ações dentro de sua responsabilidade e utilizando os serviços da camada diretamente abaixo. A camada de aplicação é onde residem aplicações de rede e seus protocolos. A camada de aplicação da Internet inclui muitos protocolos, tais como o HTTP (que provê requisição e transferência de documentos pela Web), o SMTP (que provê transferência de mensagens de correio eletrônico) e o FTP (que provê a transferência de arquivos entre dois sistemas finais). A camada de transporte carrega mensagens fim a fim, ligando o cliente e o servidor.

O TCP provê serviços orientados a conexão para suas aplicações. Alguns desses serviços são a entrega garantida de mensagens da camada de aplicação ao destino e controle de fluxo que compatibiliza as velocidades do remetente e do receptor. A camada de rede é responsável pela movimentação de pacotes, conhecidos como datagramas, de um nó a outro. Essa camada inclui o protocolo IP, que define os campos no datagrama e o modo como os sistemas finais e os roteadores agem nesses campos. Existe apenas um único protocolo IP, e todos os componentes da Internet que têm uma camada de rede devem executá-lo. Para que um host seja capaz de suportar mais de um serviço em execução simultaneamente,

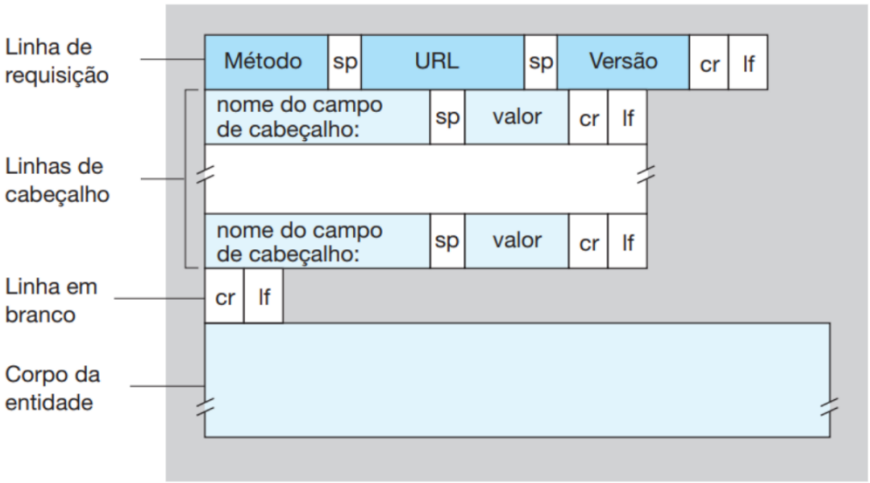
o protocolo prevê que esses serviços sejam identificados com um número de 16 bits, denominado porta.

O HTTP — Protocolo de Transferência de Hipertexto (Hyper Text Transfer Protocol) – está no coração da Web. O HTTP é executado em um cliente e um servidor. Os dois, executados em sistemas finais diferentes, conversam entre si por meio da troca de mensagens HTTP. O protocolo define a estrutura dessas mensagens e o modo como o cliente e o servidor as trocam. Quando um usuário requisita uma página Web, o navegador envia ao servidor mensagens de requisição HTTP para os recursos desejados. O servidor recebe as requisições e responde com mensagens de resposta HTTP que contêm os recursos. O HTTP usa o TCP como seu protocolo de transporte e o IP como protocolo de rede. Opcionalmente, uma camada de segurança TLS pode ser adicionada.

Requisição HTTP

Uma requisição HTTP possui a seguinte estrutura:

Figura 1 – Requisição HTTP



Fonte: Redes de computadores e a Internet: uma abordagem top-down.

As especificações do HTTP definem os formatos das mensagens HTTP de requisição e de resposta. Uma mensagem de requisição HTTP é escrita em texto ASCII comum, de modo que pode ser lida por qualquer um. É constituída por linhas

de cabeçalho separadas por quebras "carriage return" e "line feeds". A primeira linha é denominada linha de requisição, as subsequentes são denominadas linhas de cabeçalho. A linha de requisição tem três campos: o método, a URL e a versão do HTTP. O campo do método pode assumir vários valores diferentes, entre eles GET, POST, HEAD, PUT e DELETE. Por fim, a requisição possui ainda o corpo da entidade.

URL

A URL (Uniform Resource Locator - Localizador Uniforme de Recursos) é o localizador que um usuário digita para encontrar recursos na Internet. A estrutura de uma URL é dividida em diversas partes, como por exemplo:

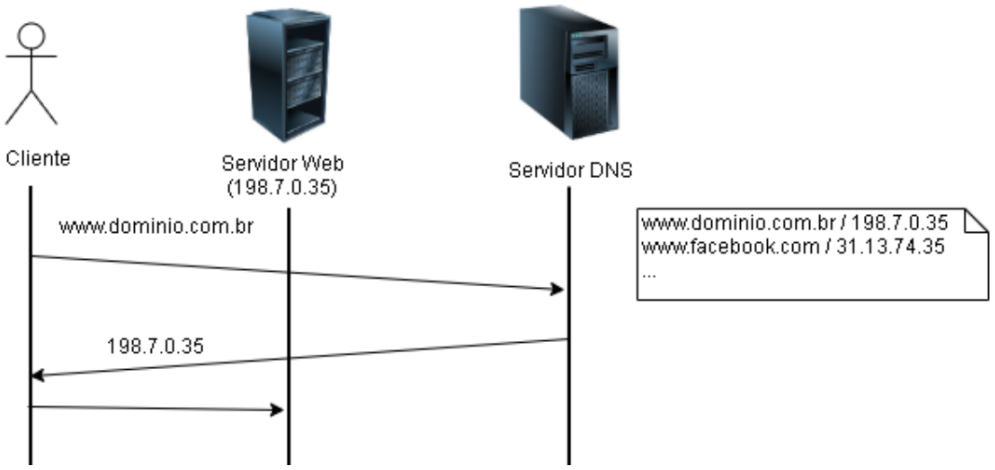
- Protocolo: é o protocolo que o servidor usará para associar ao recurso. O mais comum é o uso de "http://" e "https://". O primeiro significa "Hypertext Transfer Protocol", e o segundo tem a adição da letra S, que representa “Secure” no final.
- Domínio: é a parte principal da URL. É basicamente o nome do site. Se o domínio estiver livre, pode ser registrado.
- Top Level Domain (TLD): É a extensão que acompanha o domínio. Uma extensão ".com" serve para endereços comerciais, ".org" para organizações sem fins lucrativos, ".edu" é para instituições de ensino, entre outros.
- Country Code: É dividido em níveis que indicam a região e o segmento de funcionamento da instituição em questão. Domínios .br, por exemplo, são relacionados ao Brasil. ".de" pertencem à Alemanha.
- Recurso ou Diretório: representa uma estrutura de subdiretórios e arquivos criados a partir do domínio principal

O Serviço DNS

Apesar de poderem ser localizados por meio de uma URL, os domínios na web são de fato encontrados por um endereço IP. Para que os domínios possam ser

traduzidos como endereços IP e, consequentemente, acessados por clientes, foi desenvolvido o DNS (Domain Name System - Sistema de Nomes de Domínio). O DNS é um sistema hierárquico e distribuído para gestão de nomes para máquinas conectadas à Internet. Assim como o HTTP, o DNS apresenta uma arquitetura cliente/servidor. O servidor DNS resolve nomes para os endereços IP e de endereços IP para os nomes respectivos, permitindo a localização de hosts num determinado domínio.

Figura 2 – Esquema simplificado do serviço DNS.



Para tratar da questão de escalabilidade, o DNS utiliza diversos servidores organizados de maneira hierárquica e distribuídos por todo o mundo. Nenhum servidor DNS isolado tem todos os mapeamentos para todos os servidores da Internet. Há três classes de servidores DNS: raiz, de domínio de alto nível (Top Level Domain - TLD) e servidores DNS autoritativos. Ao acessar o domínio "www.meudominio.com", primeiro, o cliente contatará um dos servidores raiz, que retornará endereços IP dos servidores TLD para o domínio de alto nível com. Então, o cliente contatará um desses servidores TLD, que retornará o endereço IP de um servidor autoritativo para meudominio.com. Por fim, o cliente contatará um dos servidores autoritativos, que retornará o endereço IP para o nome fornecido.

Cabeçalhos HTTP

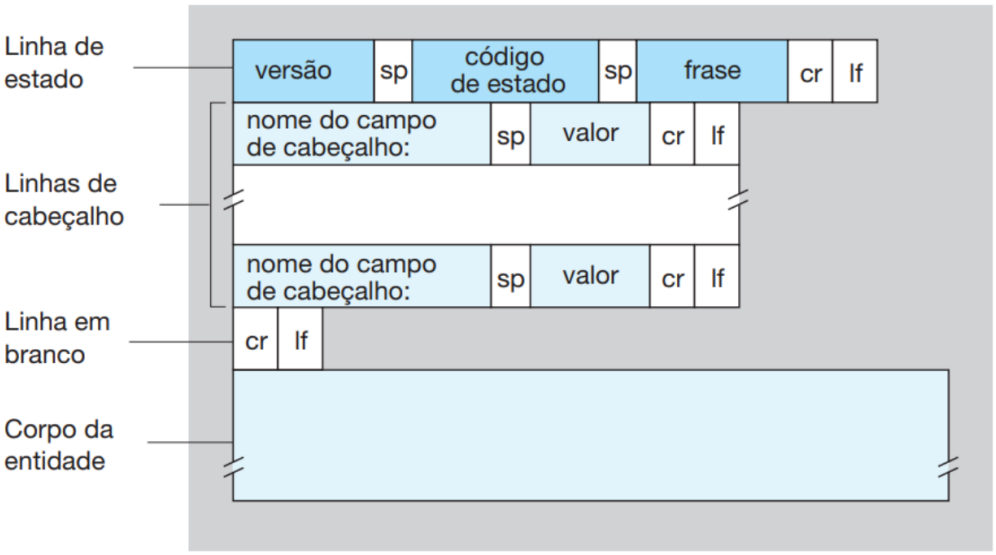
Uma requisição HTTP, como já mostrado, é constituída de várias partes. Uma parte importante para alguns processos de autenticação é o cabeçalho. As linhas de cabeçalho são um conjunto de chaves e valores que podem referir-se às informações como a hora e a data em que a requisição HTTP foi criada e enviada ao servidor, à aplicação cliente que gerou a requisição, ao tamanho e o formato do corpo da entidade, entre outras. Os cabeçalhos podem ser classificados como cabeçalhos genéricos, de solicitação, de resposta e de entidade. Veja alguns exemplos de cabeçalhos bem comuns utilizados em aplicações web:

- Autenticação:
  - Authorization;
  - WWW-Authenticate.
- Cache:
  - Cache-Control;
  - Expires.
- Gerenciamento de Conexão:
  - Connection;
  - Keep-Alive.
- Informações do corpo da mensagem:
  - Content-Length;
  - Content-Type.

Respostas HTTP

Já vimos que a requisição HTTP tem um formato bem definido e já conhecemos todas as partes que a compõem. As mensagens de resposta possuem um formato até certo ponto semelhante. Uma linha inicial ou linha de estado, um conjunto de cabeçalho e, em seguida, o corpo da entidade, que é a parte da mensagem que contém o objeto solicitado. A linha de estado tem três campos: o de versão do protocolo, um código de estado e uma mensagem de estado correspondente. Possíveis estados de resposta são 100 (respostas informativas), 200 (respostas de sucesso), 300 (redirecionamento), 400 (erro do cliente) ou 500 (erro do servidor). As linhas de cabeçalho possuem o mesmo formato daquele observado nas mensagens de requisição, porém, as informações são referentes ao servidor, como a versão do servidor web, data que a resposta foi gerada, entre outros.

Figura 3 – Requisição HTTP.



Fonte: Redes de computadores e a Internet: uma abordagem top-down.

Para mais detalhes é possível obter uma listagem com todos os métodos e códigos de estado do protocolo HTTP, respectivamente, nas fontes:

- https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Methods
- https://developer.mozilla.org/pt-BR/docs/Web/HTTP/Status

Conteúdo das Mensagens HTTP

Em uma requisição HTTP ou em uma resposta recebida, podemos encontrar o corpo da mensagem HTTP contendo dados associados à requisição (como o conteúdo de um formulário HTML), ou o documento associado à resposta. A presença do corpo e seu tamanho são especificados pela linha inicial e os cabeçalhos HTTP.

Nem todas as requisições possuem um corpo de mensagem, como por exemplo, em requisições que utilizam os métodos GET, HEAD, DELETE ou OPTIONS. Por outro lado, o corpo da mensagem é um item de extrema importância quando se quer enviar dados ao servidor, por exemplo, por meio de uma requisição utilizando o método POST.

Os principais formatos utilizados para o corpo da mensagem são HTML e JSON. A resposta HTTP deve especificar o tipo de conteúdo do corpo. Isso é feito no cabeçalho, no campo content-type, por exemplo, “Content/Type: application/json”.

Sessões HTTP

Um servidor HTTP não tem estado o que simplifica o protocolo e vem permitindo que sejam desenvolvidos servidores Web de alto desempenho, capazes de manipular milhares de conexões TCP simultâneas. No entanto, é sempre bom que um site identifique usuários, seja porque o servidor deseja restringir acesso ou porque quer apresentar conteúdo em função da identidade do usuário. Para poder criar sessões e monitorar os usuários, o HTTP usa cookies.

Cookies são arquivos criados pelos websites para diversas finalidades, uma das principais é identificar o usuário. Quando a requisição chega ao servidor, ele cria um número de identificação exclusivo e uma entrada no seu banco de dados. Então, o servidor responde ao navegador, incluindo na resposta HTTP um cabeçalho “Set-cookie”: que contém o número de identificação. Quando recebe a mensagem de resposta HTTP, o navegador vê o cabeçalho e, então, cria um arquivo na base de cookies que ele gerencia, incluindo o nome do servidor e seu número de identificação

no cabeçalho. A partir de então, toda vez que enviar uma requisição para o mesmo servidor, o navegador consulta seu arquivo de cookies, extrai seu número de identificação para esse site e insere na requisição HTTP uma linha de cabeçalho de cookie que inclui o número de identificação. Dessa maneira, o servidor em questão pode monitorar a atividade do cliente em seu site e, embora não saiba necessariamente quem é o usuário, sabe com exatidão quais páginas foram visitadas, em que ordem e em que horários.

Os cookies têm quatro componentes: uma linha de cabeçalho na resposta HTTP; uma linha de cabeçalho na requisição HTTP; um arquivo de cookie gerenciado pelo navegador do usuário; um banco de dados no site. Cookies podem ser usados para criar uma camada de sessão de usuário sobre HTTP sem estado. Por exemplo, quando um usuário acessa uma aplicação de e-mail baseada na Web, o navegador envia informações de cookie ao servidor, permitindo que o servidor identifique o usuário por meio da sessão deste com a aplicação.

Embora cookies quase sempre simplifiquem a experiência de uso da Internet, continuam provocando muita controvérsia, porque também podem ser considerados invasão da privacidade do usuário.

Capítulo 2. Autenticação HTTP

Autenticação no protocolo HTTP é uma forma segura de permitir somente que usuários identificados acessem recursos de um determinado servidor.

Tipos de Autenticação

Existem algumas formas de criar uma autenticação HTTP. Algumas serão explicadas a seguir, e outras serão detalhadas nos próximos

A Autenticação Básica (será explicada em detalhes mais adiante) é uma maneira de autenticação onde em cada requisição deve ser enviado um cabeçalho com o nome de usuário e senha. É a técnica mais simples de controle de acesso e não garante muita segurança ao aplicativo.

Uma das técnicas mais utilizadas é a Bearer Authentication, que também é conhecida como Token Authentication. Nesse tipo de autenticação existe, em cada requisição, um cabeçalho com a chave “Authorization” e o valor “Bearer <token>”, onde o token é um texto (uma string) gerado pelo servidor.

Outra forma de autenticação prevista pelo protocolo é o HTTP Digest, onde o cliente envia, em cada requisição, um cabeçalho com as credenciais de forma criptografada, aplicando uma função de hash paranome de usuário e senha e mais algumas informações, de forma que o servidor consiga validar o cabeçalho.

Codificação Base64

Antes de entender melhor os detalhes da Autenticação Básica, é necessário compreender melhor a codificação Base64. Esse tipo de codificação é bastante utilizado para transmissão de dados via web, seja para autenticação básica ou para simplificar a transferência de dados. É constituído por 64 caracteres ([A-Z], [a-z],[0-9], "/" e "+"). Dessa forma, o texto “Olá, mundo!” seria convertido em “T2zDoSwgbXVuZG8h”.

Um ponto importante é compreender também que codificação é diferente de criptografia. A codificação é a aplicação de um processo de conversão da mensagem com o intuito de que não seja compreendida. As regras de transformação serão definidas a fim de seja possível compreender o conteúdo da mensagem e deve ser facilmente revertida (decoded).

A criptografia é a aplicação de um algoritmo que baseado em uma chave para transformar a estrutura da mensagem, de modo que, caso seja interceptada por um terceiro, não será compreendida. Ou seja, a codificação não utiliza uma chave, por isso dizemos que o objetivo não é tornar a informação secreta, e sim garantir que os dados possam ser consumidos por diferentes tipos de sistema.

Autenticação Básica

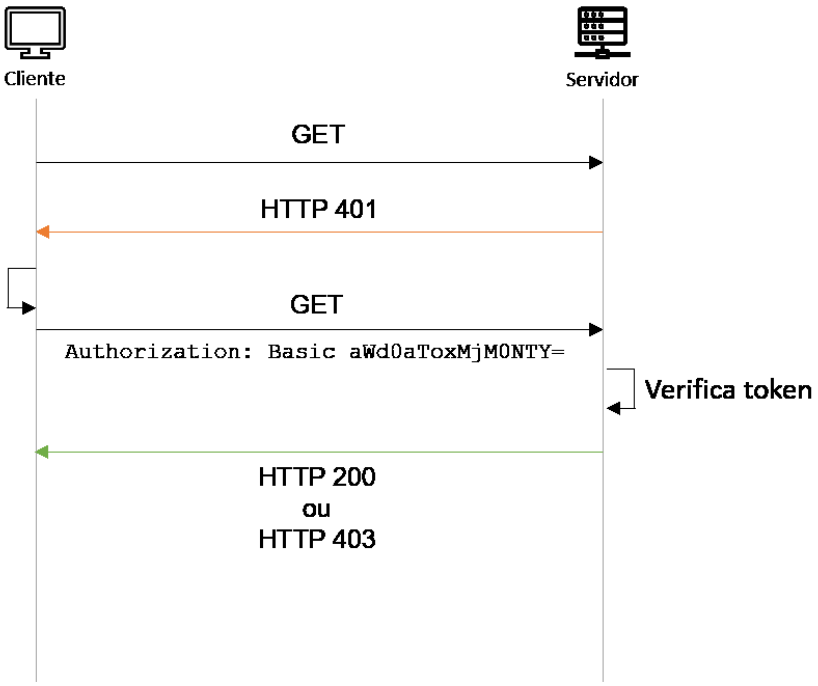
O HTTP fornece uma estrutura geral para controle de acesso e autenticação. A autenticação HTTP mais comum é chamada "Basic". Essa autenticação define a estrutura de autenticação HTTP que pode ser usada por um servidor para que um cliente possa fornecer informações de autenticação.

O fluxo se inicia quando o servidor responde ao cliente com uma mensagem do tipo 401 (Não autorizado) e fornece informações de como autorizar com um cabeçalho de resposta *WWW-Authenticate*. Um cliente que deseja autenticar-se com um servidor pode incluir na requisição um campo de cabeçalho de solicitação Authorization com as credenciais. Se um servidor recebe credenciais válidas deverá responder com um código OK 200. Caso as credenciais sejam válidas, mas inadequadas para ter acesso ao recurso informado, o servidor poderá responder com o código Forbidden 403.

As credenciais são formatadas pelo cliente antes de enviar ao servidor. O primeiro passo é concatenar usuário e senha separados por dois pontos. O texto obtido da concatenação é codificado em base64. A função da codificação não é proteger as credenciais do usuário, mas sim garantir que caracteres incompatíveis com o HTTP não sejam enviados no cabeçalho. A maioria das linguagens de

programação possui uma função de decodificação nativa. As credenciais do usuário, portanto, não são consideradas seguras. Por isso, a troca deve acontecer por meio de uma conexão HTTPS (HTTP + TLS) para ser segura. Caso o ID e senha do usuário sejam transmitidos através da rede como texto claro (é codificado em base64, mas base64 é uma codificação reversível), o esquema básico de autenticação não é seguro.

Figura 4 – Fluxo de Autenticação Básica.



JSON Web Tokens

Para autenticar uma requisição em uma aplicação sem estado, é necessário que seja enviado pelo cliente um identificador autônomo. Ou seja: um identificador capaz de provar por si só a autenticidade da requisição, sem que seja necessário confirmar junto a outro serviço. Atualmente, tem sido cada vez mais utilizado o conceito de JWT para essa função. JWT (JSON Web Token) é um padrão para formatação de tokens que permite que informações sejam enviadas entre as partes de forma segura.

Os tokens JWT são amplamente utilizados em sistemas web para autenticação e autorização, é padrão da indústria para autenticação entre duas partes por meio de um token assinado. Esse token é um código em Base64 que armazena objetos JSON com diversos dados que permitem a identificação do usuário e autenticação da requisição.

O JWT possui três partes, são elas:

- Header;
- Payload;
- Signature.

Ao definir cada uma das partes, é gerado um token no formato “xxxxx.yyyyy.zzzzz”.

O cabeçalho ou header do token contém o tipo de mídia utilizado (typ) e o algoritmo da assinatura (alg). Por exemplo:

```
{
"alg": "HS256",
"typ": "JWT"
}
```

O payload contém os todos dados que precisam ser enviados. No caso de uma autenticação, dados referentes ao usuário, suas permissões e ao próprio servidor que está emitindo o token. As informações podem ser de 3 tipos:

- Reserved claims: atributos não obrigatórios (mas recomendados) usados na validação do token.
- Public claims: atributos que usamos em nossas aplicações. Geralmente são as informações do usuário autenticado.
- Private claims: atributos definidos especialmente para compartilhar informações entre aplicações.

Por exemplo:

```
{
    "sub": "1234567890",
    "name": "John Doe",
    "admin": true
}
```

De posse dessas duas informações, o servidor deverá codificá-las em base64 e concatenar os resultados obtidos, separando o conteúdo por ponto. Ao final, uma terceira parte composta pela assinatura desse mesmo conteúdo será concatenada ao token, também separando por ponto. Ou seja, a signature ou assinatura é a concatenação dos hashes gerados a partir do Header e Payload usando base64UrlEncode, com uma chave secreta:

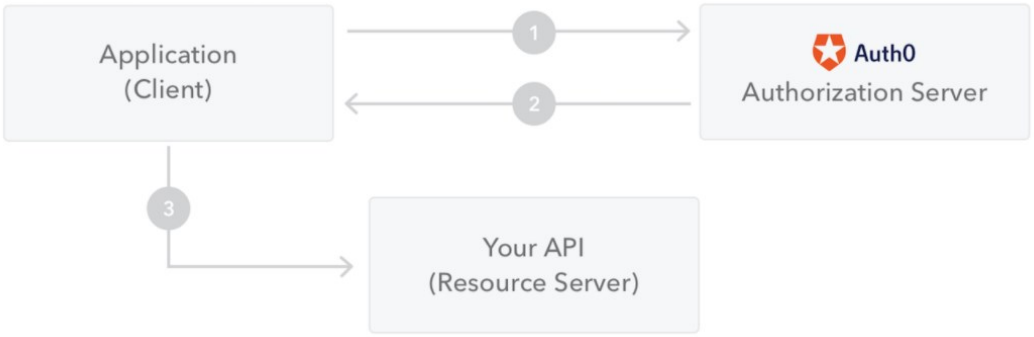
```
HMACSHA256(
    base64UrlEncode(header)
    + "."
    + base64UrlEncode(payload),
    "secret"
)
```

A figura a seguir mostra um exemplo completo, considerando a chave secreta como “IGTI”, e com cores que facilitam o entendimento de cada uma das partes:

Figura 5 – JSON Web Token.

Quando há um sistema com autenticação baseada em JWT, é importante saber que o cliente não é responsável por gerar esse token, ele apenas requisita, recebe e se responsabiliza pelo armazenamento para futuras requisições, como mostrado na figura a seguir:

Figura 6 – Fluxo JWT.



Supondo que um serviço receba um token, ele deverá verificar se a assinatura é válida. Caso o usuário tenha alterado alguma informação, como, por exemplo, seu nível de permissão para admin, a assinatura não será verificada com sucesso.

Capítulo 3. HTTPS

O Capítulo 3 trata de questões ligadas ao TLS/SSH e ao protocolo HTTPS, versão mais segura do HTTP. O protocolo HTTPS é, atualmente, de fundamental importância para que as soluções web possam prover os serviços aos quais se destinam. Uma vez que informações confidenciais circulam o tempo todo em redes não seguras, manter as propriedades dessa informação é de muito importante.

Criptografia

A criptografia, considerada como a ciência de escrever mensagens em forma cifrada ou em código, é um dos principais mecanismos de segurança utilizados para proteger dos riscos associados ao uso da Internet. Atualmente, a criptografia já está integrada ou pode ser facilmente adicionada à grande maioria dos sistemas operacionais e aplicativos. Para usá-la, muitas vezes basta a realização de algumas configurações ou cliques de mouse. Por meio do uso da criptografia, é possível:

- Proteger os dados sigilosos armazenados em um computador, como arquivo de senhas e documentos.
- Criar uma partição específica em um computador, na qual todas as informações que forem gravadas serão automaticamente criptografadas.
- Proteger backups contra acesso indevido, principalmente aqueles enviados para áreas de armazenamento externo de mídias.
- Proteger as comunicações realizadas pela Internet, como os e-mails enviados/recebidos e as transações bancárias e comerciais realizadas.

Enviar mensagens secretas é uma tarefa muito antiga. O homem sentiu, desde muito cedo, a necessidade de guardar informações em segredo. Ela nasceu com a diplomacia e com as transações militares. Generais, reis e rainhas, durante milênios, buscavam formas eficientes de comunicação para comandar seus exércitos e governar seus países. A importância de não revelar segredos e estratégias às forças

inimigas motivou o desenvolvimento de códigos, cifras e técnicas para mascarar uma mensagem, possibilitando apenas ao destinatário ler o conteúdo. As nações passaram a criar departamentos para elaborar sistemas criptográficos e, por outro lado, surgiram os decifradores de códigos, criando uma corrida armamentista intelectual. Ao longo da história, os códigos decidiram o resultado de batalhas. À medida que a informação se torna cada vez mais valiosa, o processo de codificação de mensagens tem um papel cada vez maior na sociedade. Com o advento da comunicação eletrônica, esse deixou de ser um assunto tratado unicamente como segredo de estado, pois muitas atividades essenciais dependem do sigilo na troca de mensagens, com, por exemplo, transações financeiras e uso seguro da Internet.

Para codificarmos ou decodificarmos uma mensagem, necessitamos de informações confidenciais denominadas chave. A cripto análise estuda formas de decodificar uma mensagem sem se conhecer, de antemão, a chave. Ela reconstrói, a partir da mensagem codificada, a mensagem no seu formato original, com a ajuda de métodos matemáticos. A cripto análise é responsável por quebrar o código da mensagem codificada, o que permite transformar dados ou mensagens em alguma forma legível. A criptografia, por outro lado, é utilizada para proteger informações e manter o sigilo de dados confidenciais. A criptografia utiliza métodos para a produção e distribuição segura de chaves, e estuda algoritmos que permitem transformar mensagens claras em formas de comunicação só inteligíveis pelos emissores e pelos receptores envolvidos no processo.

As cifragens de mensagem são uma forma de ocultar informações por meio da substituição de letras ou símbolos, isolados ou agrupados. Os métodos de cifragem são divididos segundo sua natureza.

Cifragem por transposição

Neste método, os conteúdos das mensagens originais e criptografadas são os mesmos, porém, as letras são postas em ordem diferente. Por exemplo, pode-se cifrar a palavra TAPETE e escrevê-la como ATEETP. No método da permutação de colunas, dada uma mensagem original, seu conteúdo pode ser dividido em blocos de tamanho fixo. Uma chave de índices é estabelecida e pode ser aplicada a cada bloco. A chave deve conter o mesmo tamanho do bloco. Exemplo:

- Mensagem original: Minha filha chora muito.
- Tamanho do bloco: 8.
- Chave: 5 2 8 4 1 6 3 7.
- Minha_fi.
- lha_chor.
- a_muito.

  Aplicando a chave à primeira linha temos:

- aiihM_nf.

Cifragem por substituição

Neste procedimento, trocam-se cada letra ou grupo de letras da mensagem de acordo com uma tabela de substituição. No método da substituição simples (monoalfabética) substituem-se cada caractere do texto por outro, de acordo com uma tabela pré-estabelecida.

Cifragem de César

Um dos primeiros sistemas de criptografia conhecido foi elaborado pelo general Júlio César, no Império Romano. Júlio César usou sua famosa cifra de substituição para criptografar comunicações governamentais. Para exemplificar, suponha um desvio de três posições. Desta forma, A se tornava D, B se tornava E, e assim sucessivamente até o fim da mensagem. Algumas considerações importantes precisam ser feitas sobre a cifra de César:

- Como é utilizada a mesma chave tanto para cifrar quanto para decifrar a mensagem, essa é uma metodologia simétrica de cifragem.
- Supondo-se um alfabeto de 26 caracteres (ignorando-se espaços, acentos, e outros caracteres) bastariam 26 tentativas para decifrar a mensagem. Esse método baseado em tentativas e erros é chamado de quebra por força bruta.
- Suponha agora que as mensagens sejam escritas em português. Se tomarmos por base a frequência de ocorrência de letras da língua, podem-se fazer algumas inferências e diminuir o número de tentativas necessárias para descobrir o conteúdo cifrado.

Informações involuntariamente disponíveis numa mensagem cifrada e que podem ser utilizadas para decifrá-las são chamadas de impressões digitais. Os pontos mais fracos da cifragem de césar são, justamente, a forte impressão digital das mensagens geradas e o baixo número de tentativas necessárias para se quebrar a segurança por meio de força bruta.

Chave Polialfabética

Uma forma de aumentar a força dos algoritmos de substituição é melhorando a força das chaves, aumentado de um para N dígitos. Essa técnica é capaz de diminuir as impressões digitais deixadas na mensagem cifrada e aumentar a resistência a quebras por força bruta, simultaneamente. A resistência a ataques de força bruta aumenta com o aumento do tamanho da chave até o limite do tamanho da própria mensagem. A esse modelo é dado o nome de cifra polialfabética.

Para exemplificar uma chave desse tipo, considere a seguinte mensagem: “Segurança é fundamental”. Utilizaremos a chave “IGTI”. Para facilitar o entendimento, considere também o quadro a seguir:

Figura 7 – Índices das letras para criptografia.

a	b	c	d	e	f	g	h	i	j	k	l	m
0	1	2	3	4	5	6	7	8	9	10	11	12
n	o	p	q	r	s	t	u	v	w	x	y	z
13	14	15	16	17	18	19	20	21	22	23	24	25

A chave nesse caso (“IGTI”) tem 4 caracteres, então nossa mensagem será dividida em partes de tamanho igual a 4 caracteres: “segu.ranc.aefu.ndam.enta.l”. Para cada uma das partes, aplicaremos o método de substituição similar ao da Cifra de César, o resultado será então: “akzc.zggk.ikyc.vjtu.mtmi.t”.

A utilização correta de parâmetros de entrada para algoritmos de criptografia é fundamental para o seu melhor funcionamento. Em alguns momentos, valores obtidos de forma não aleatória podem tornar um sistema mais vulnerável e devem ser, portanto, evitados. Mesmo que a princípio uma escolha humana possa parecer tão imprevisível quanto um valor obtido de forma realmente aleatória, sequências de dados obtidos dessa forma podem ser detectadas em análises minuciosas.

Para entender melhor esse conceito, suponha o seguinte cenário: sabendo que uma moeda possui dois lados (cara e coroa), duas pessoas devem informar uma sequência aleatória de valores possíveis. Uma dessas pessoas vai realmente atirar a moeda para cima. Já a segunda pessoa irá escolher os valores que desejar, simulando ter atirado a moeda. Um observador externo, que não vê o que as duas pessoas estão fazendo, seria capaz de determinar quem está realmente atirando a moeda? A resposta é SIM!

Para isso, basta observar uma propriedade das duas sequências, chamada de propriedade da estabilidade da frequência. O número de caras e coroas obtidos pelos dois atores será, muito provavelmente, parecido. A diferença está presente se analisarmos os valores em sequências. No primeiro cenário, as sequências tendem a

ser mais homogêneas, pois a probabilidade de obter-se uma sequência cara x cara x cara é a mesma de obter-se qualquer outra sequência. Já no segundo cenário, as sequências tendem a ser mais heterogêneas. Isso se dá devido ao fato de seres humanos terem preferências e, inconscientemente, acharem que algumas sequências são mais ou menos prováveis que outras.

Algoritmo Diffie-Hellman

Para que um algoritmo de chaves simétricas possa funcionar corretamente em um ambiente distribuído, como a Internet, é necessária que exista uma maneira segura de se transmitir a chave entre as partes sem que seja necessário que tenham que se encontrar fisicamente. A troca de chaves de Diffie-Hellman é um método de criptografia específico para troca de chaves, desenvolvido em 1976. Foi um dos primeiros exemplos práticos de métodos de troca de chaves desenvolvidos dentro do campo da criptografia. O método da troca de chaves permite que duas partes que não possuem conhecimento a priori de cada uma, compartilhem uma chave secreta sob um canal de comunicação inseguro. Tal chave pode ser usada para criptografar mensagens posteriores, usando um esquema de cifra de chave simétrica.

Para obter um algoritmo capaz de executar essa tarefa, é necessário que se utilize uma função fácil de ser calculada, mas que, uma vez dado o resultado, é difícil obter o valor original. Um bom exemplo com essa característica é o problema do logaritmo discreto.

$$3^3 \bmod 17 = x$$

$$3 \text{ ^ } x \bmod 17 = 10$$

Na primeira equação é simples calcular que x vale 10, mas na segunda, ainda que possível, o cálculo é mais complexo. Se abstrairmos questões complexas, podemos, nesse ponto, simplesmente assumir que:

- Números primos são difíceis de serem fatorados.

- Existem números cuja operação de módulo possui distribuição uniforme e, portanto, não fornecem impressões digitais.

Dadas essas afirmativas, o exemplo abaixo mostra a transmissão, entre A e B, de chaves Diffie-Hellman de maneira simplificada.

- A e B acordam com os números 3 e 17;
- A escolhe um número secreto 15:
  - 3 ^ 15 mod 17 = 6.
- B escolhe um número secreto 13:
  - 3 ^ 13 mod 17 = 12.
- A e B trocam os resultados (6 e 12);
- A calcula 12 ^ 15 mod 17 = 10;
- B calcula 6 ^ 13 mod 17 = 10.

Com isso, a chave 10 pode ser utilizada para criptografar o restante da conversa.

Criptografia de chave assimétrica

De acordo com o tipo de chave usada, os métodos criptográficos podem ser subdivididos em duas grandes categorias: criptografia de chave simétrica e criptografia de chaves assimétricas.

Criptografia de chave simétrica: também chamada de criptografia de chave secreta ou única, utiliza uma mesma chave tanto para codificar como para decodificar informações, sendo usada principalmente para garantir a confidencialidade dos dados. Casos nos quais a informação é codificada e decodificada por uma mesma pessoa não há necessidade de compartilhamento da chave secreta. Entretanto,

quando estas operações envolvem pessoas ou equipamentos diferentes, é necessário que a chave secreta seja previamente combinada por meio de um canal de comunicação seguro (para não comprometer a confidencialidade da chave).

$$E(c, m) = m'$$

$$D(c, m') = m$$

Criptografia de chaves assimétricas: também conhecida como criptografia de chave pública, utiliza duas chaves distintas: uma pública, que pode ser livremente divulgada, e uma privada, que deve ser mantida em segredo por seu dono. Quando uma informação é codificada com uma das chaves, somente a outra chave do par pode decodificá-la.

$$E(c, m) = m'$$

$$D(c', m') = m$$

Chave c é pública

Chave c' é privada

Dependendo do objetivo, pode-se escolher a chave pública ou a privada para codificar a mensagem:

- *Confidencialidade:* garantia de que uma informação só será disponibilizada para partes que tenham autorização de acesso. Mensagem deve ser codificada coma chave pública do destinatário;
- *Assinatura digital:* Preservação do conteúdo original da mensagem e garantia de sua autoria. Mensagem deve ser codificada com chave privada do remetente.

A criptografia de chaves assimétricas, apesar de possuir um processamento mais lento que a de chave simétrica, facilita o gerenciamento (pois não requer que se mantenha uma chave secreta com cada um que desejar se comunicar) e dispensa a necessidade de um canal de comunicação seguro para o compartilhamento de

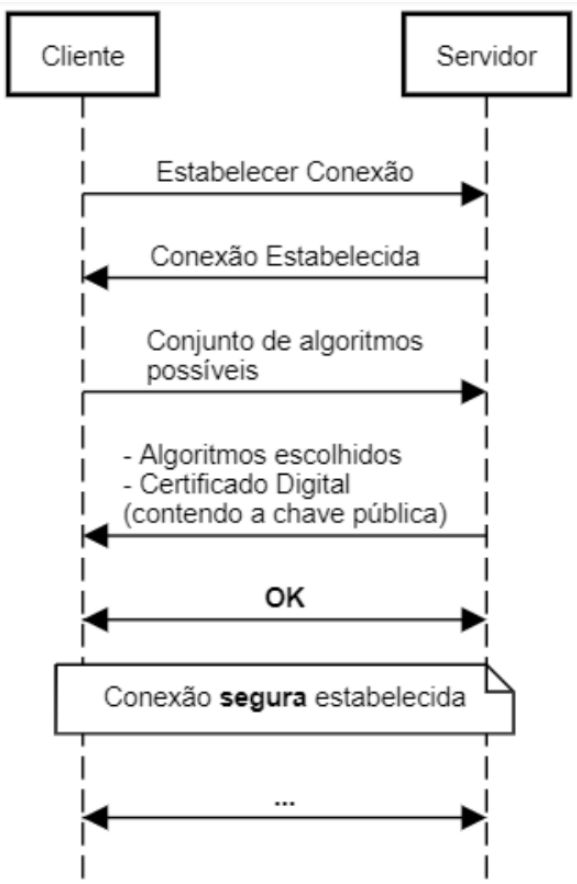
chaves. Para aproveitar as vantagens de cada um destes métodos, o ideal é o uso combinado de ambos, onde a criptografia de chaves simétricas é usada para a codificação da informação e a criptografia de chaves assimétricas é utilizada para o compartilhamento da chave secreta (neste caso, também chamada de chave de sessão). Este uso combinado é o que é utilizado pelos navegadores Web e programas leitores de e-mails. O melhor exemplo de uso deste método combinado é o TLS.

TLS

O TLS é um protocolo de segurança que certifica a proteção das informações em uma comunicação na Internet. Quando utilizado em conjunto com o HTTP, produzo assim chamado HTTPS, versão segura do protocolo HTTP. Seu funcionamento se dá baseado em um conjunto de algoritmos de criptografia (simétricos e assimétricos), bem como funções hash e assinaturas digitais utilizados por ambas as partes. Para isso, o primeiro passo é acordar sobre quais algoritmos utilizar. O TLS utiliza de uma etapa de handshake para realizar esse acordo.

O fluxo pode ser observado na figura a seguir:

Figura 8 – Fluxo TLS.



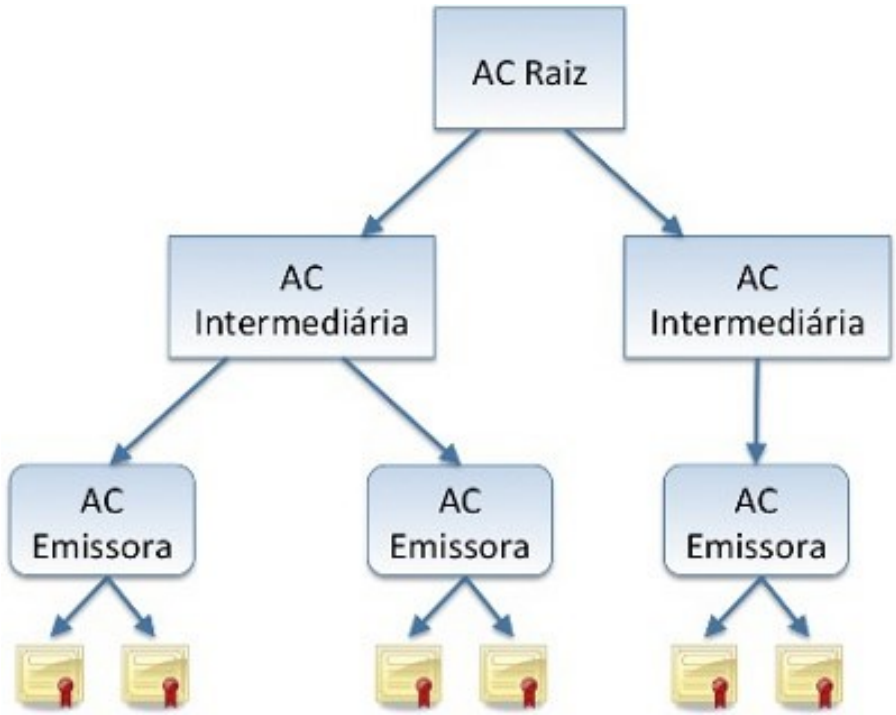
Certificado Digital

Como dito anteriormente, a chave púbica pode ser livremente divulgada. Entretanto, se não houver como comprovar a quem ela pertence, pode ocorrer de uma das partes se comunicar, de forma cifrada, diretamente com um impostor. Um impostor pode criar uma chave pública falsa e enviá-la para uma vítima ou, ainda, disponibilizá-la em um repositório. Ao usar essa chave para codificar uma informação, a vítima estará, na verdade, codificando a mensagem para o impostor, que possui a chave privada correspondente e conseguirá decodificar. Uma das formas de impedir que isto ocorra é pelo uso de certificados digitais.

O certificado digital é um registro eletrônico composto por um conjunto de dados que distingue uma entidade e associa a ela uma chave pública. Ele pode ser emitido para pessoas, empresas, equipamentos ou serviços na rede (por exemplo, um site Web) e pode ser homologado para diferentes usos, como confidencialidade e

assinatura digital. Um certificado digital pode ser comparado a um documento de identidade, por exemplo, o seu passaporte, no qual constam os seus dados pessoais e a identificação de quem o emitiu. No caso do passaporte, a entidade responsável pela emissão e pela veracidade dos dados é a Polícia Federal. No caso do certificado digital esta entidade é uma Autoridade Certificadora (AC).

Figura 9 – Cadeia de Certificados.



Fonte: Cartilha de Segurança para Internet.

Uma AC emissora é também responsável por publicar informações sobre certificados que não são mais confiáveis. Sempre que a AC descobre ou é informada que um certificado não é mais confiável, ela o inclui em uma lista negra chamada de "Lista de Certificados Revogados" (LCR) para que os usuários possam tomar conhecimento. A LCR é um arquivo eletrônico publicado periodicamente pela AC, contendo o número de série dos certificados que não são mais válidos e a data de revogação.

De forma geral, os dados básicos que compõem um certificado digital são:

- Versão e número de série do certificado;

- Dados que identificam a AC que emitiu o certificado;
- Dados que identificam o dono do certificado (para quem ele foi emitido);
- Chave pública do dono do certificado;
- Validade do certificado (quando foi emitido e até quando é válido);
- Assinatura digital da AC emissora e dados para verificação da assinatura.

O certificado digital de uma AC é emitido, geralmente, por outra AC, estabelecendo uma hierarquia conhecida como "cadeia de certificados" ou "caminho de certificação".

Existem diversas autoridades certificadoras no Brasil, e é possível conferir a lista completa em: https://www.gov.br/iti/pt-br/assuntos/icp-brasil/autoridades-certificadoras.

Capítulo 4. OAuth 2.0

O Capítulo 4 fecha a disciplina apresentando os conceitos de Autenticação Federada e Autorização Delegada, bem como a implementação destes por meio do protocolo OAuth 2.0. Estes são alguns dos conceitos mais importantes relacionados à autenticação/autorização de aplicações modernas na web.

Introdução ao OAuth 2.0

OAuth 2.0 pode ser definido com um protocolo de autorização que protege um serviço, permitindo que aplicações clientes possam dispor de um método padrão para acessá-lo. Os recursos gerenciados pelo serviço podem pertencer a um usuário desse serviço ou a ele próprio. Os tipos de recursos permitidos são, normalmente, dados armazenados pelos usuários. Para obter acesso aos recursos, as aplicações cliente devem fazer requisições HTTP para endpoints, utilizando um token que identifique o usuário que forneceu o acesso. O uso mais comum do protocolo OAuth2.0 para fornecer segurança a APIs RESTful acontece quando serviços comunicam-se caracterizando um modelo B2B. A especificação do OAuth 2.0 define-o como um framework, deixando em aberto alguns de seus aspectos, principalmente ligados à codificação.

Em uma Internet cada vez mais conectada por plataformas de colaboração e redes sociais, é comum que desenvolvedores tenham pela frente o desafio de acessar os dados de usuários em outros sistemas para atender aos requisitos de uma ferramenta em desenvolvimento. Para esse tipo de situação, o protocolo OAuth fornece a habilidade de acesso de maneira segura, sem solicitar as credenciais do usuário. Essa facilidade melhora consideravelmente o trabalho do desenvolvedor, reduzindo a curva de aprendizado e aumentando a confiança na segurança da API, uma vez que o padrão foi definido por uma comunidade vasta e experiente. Além de ser melhor do ponto de vista do desenvolvedor, o OAuth fornece outras vantagens em relação ao ato de fornecer credenciais entre partes:

- Poucos usuários confiariam seu usuário e sua senha a alguém;

- É um padrão já disseminado no mercado, várias redes sociais já aceitam esse tipo de autenticação e autorização;
- Facilita a gestão de nome de usuário e senhas para os usuários dos sistemas;
- O cliente tem controle sobre quais recursos deseja conceder ou revogar o acesso.

O OAuth define quatro papéis que interagem entre si nos seus diferentes cenários:

- **Cliente:** É a aplicação que deseja realizar requisições a uma API e manipular recursos no servidor de recursos em nome de alguém com o seu consentimento;
- **Dono do Recurso/Usuário:** É o usuário da aplicação que tem o poder de garantir acesso aos seus dados hospedados no servidor de recursos;
- **Servidor de recursos:** É um servidor que hospeda recursos pertencentes a um usuário e os protege com OAuth. É tipicamente um provedor contendo dados pessoais, contatos, fotos, vídeos, etc.;
- **Servidor de autorização:** Para que um cliente possa acessar recursos protegidos, ele precisa estar autorizado pelo usuário dono do recurso. Isso é papel do servidor de autorização. Ele solicita ao usuário, dono do recurso, que confirme que a aplicação cliente está autorizada a acessar seus dados. Paracada autorização concedida pelo usuário, o cliente recebe um token de acesso. Esse token deve ser utilizado pelo cliente para especificar em nome de qual usuário está fazendo a requisição. Como o token é concedido pelo usuário para uma única aplicação cliente, pode-se afirmar que esse token representa a relação entre o servidor de autorização, a aplicação cliente e o usuário dono do recurso.

Autenticação e Autorização

Para entender corretamente o OAuth, é importante primeiramente entender alguns termos relevantes:

- **Autenticação:** é o processo responsável por verificar a identidade de um usuário ou, em outras palavras, verificar se um usuário é de fato quem diz ser. No mundo real, quando uma autoridade solicita a identidade de um cidadão, ele está verificando se a pessoa na foto é a mesma que se apresenta pessoalmente. No mundo digital, a autenticação busca saber se o usuário de um sistema é quem diz ser, solicitando seu login e sua senha (chamados de credenciais).

- **Autenticação Federada:** apesar de muitas aplicações terem seus sistemas de contas com cadastro de usuários e senhas, algumas aplicações preferem confiar no serviço de terceiros para verificar a identidade de usuários. Esse método é chamado de autenticação federada.

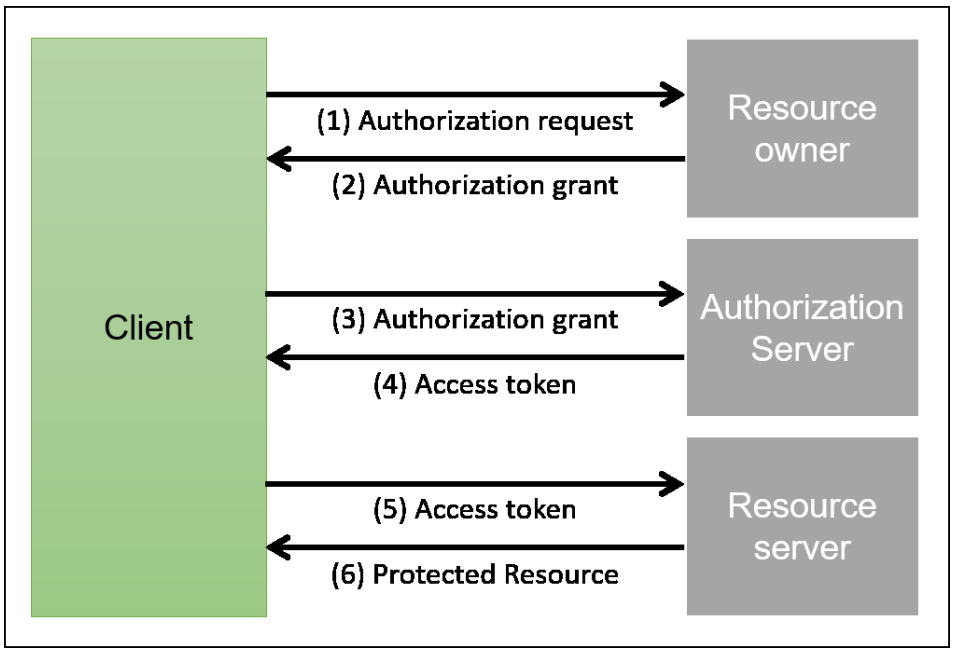
- **Autorização:** é o processo de verificar que um usuário tem o direito de executar uma ação, como, por exemplo, ler um documento ou acessar uma conta de e-mail. Isso requer que o usuário esteja previamente autenticado para que, posteriormente, seja verificado se está autorizado. No mundo real, por exemplo, para poder dirigir, é necessário uma autorização obtida com a carteira de identidade. O mesmo processo vale para exercer algumas profissões, como votar, além de outras atividades. De forma semelhante, ao realizar login em uma aplicação, será verificado o assim chamado perfil do usuário, para que se possa estabelecer a quais recursos ele tem ou não acesso.

- **Autorização delegada:** é a garantia de acesso dada por um usuário a outra pessoa ou aplicação para que possa executar ações em seu nome. O OAuth é um framework (ou protocolo) de autorização delegada que garante a uma aplicação executar ações em nome de um usuário, no qual a aplicação pode ter permissões para executar apenas algumas ações pré-definidas pelo próprio usuário.

Funcionamento do OAuth 2.0

O diagrama apresentado na figura a seguir mostra um exemplo abstrato sobre como um cliente é autorizado, obtém o token e faz requisições ao recurso protegido em nome do usuário. O primeiro e o segundo passo especificam a interação entre a aplicação cliente e o usuário dono do recurso. Primeiramente, o cliente faz a requisição ao usuário para que autorize o acesso a seus recursos protegidos e, em seguida, o usuário aprova a requisição e o cliente recebe uma autorização. Geralmente, o cliente não solicita o recurso diretamente ao usuário, deixando essa ação para o servidor de autorização.

Figura 10 – Fluxo Abstrato do OAuth 2.0.



O terceiro e o quarto passos especificam a interação entre o cliente e o servidor de autorização. Para que esses passos possam acontecer, a aplicação cliente precisa necessariamente estar autenticada no servidor. A forma como essa autenticação acontece está fora do escopo do flow. Primeiramente, o cliente fornece a garantia de autorização que obteve previamente para o servidor de autorização para obter um token de acesso e, em seguida, o servidor de autorização verifica se os dados são válidos. Em caso positivo, fornece à aplicação cliente um token de acesso.

Nos dois últimos passos, o cliente e o servidor de recursos interagem. O cliente faz a requisição especificando qual o recurso protegido ele quer acessar,

enviando o token de acesso para que possa ser autorizado. Então, o servidor verifica o token e, em caso positivo, fornece a resposta com o recurso solicitado.

Existem dois tipos de token no OAuth 2.0. O token de acesso e o token de renovação. O token de acesso é aquele utilizado pela aplicação cliente quando faz as requisições por recursos protegidos. Já o token de renovação é aquele utilizado para renovar o token de acesso quando aquele perde sua validade.

OAuth 2.0 permite que clientes sejam classificados de acordo com seu tipo:

- Cliente confidencial: É o tipo de cliente capaz de manter a confidencialidade sobre as credenciais mantidas em segurança. São aplicações executadas em ambientes mais restritos, por exemplo, aplicações em execução em servidores;
- Cliente público: São clientes que não possuem a capacidade de manter a confidencialidade das credenciais mantidas em segurança. São, por exemplo, aplicações escritas em JavaScript e executadas em um navegador, ou ainda aplicações executadas em um aplicativo móvel.

Por fim, no OAuth 2.0 existem três importantes endpoints. Dois deles são URLs no servidor e um no cliente.

- Autorização (Servidor): A aplicação cliente utiliza esse endpoint para ser autorizada pelo cliente dono do recurso. Se obtiver sucesso, o cliente recebe o código de garantia de acesso.
- Token (Servidor): O cliente utiliza esse endpoint para enviar o código de acesso e receber o token.
- Callback (Cliente): O servidor de autorização utiliza esse endereço para retornar as credenciais de autorização para o cliente. Normalmente é especificado quando a aplicação cliente está sendo registrada no servidor de autorização.

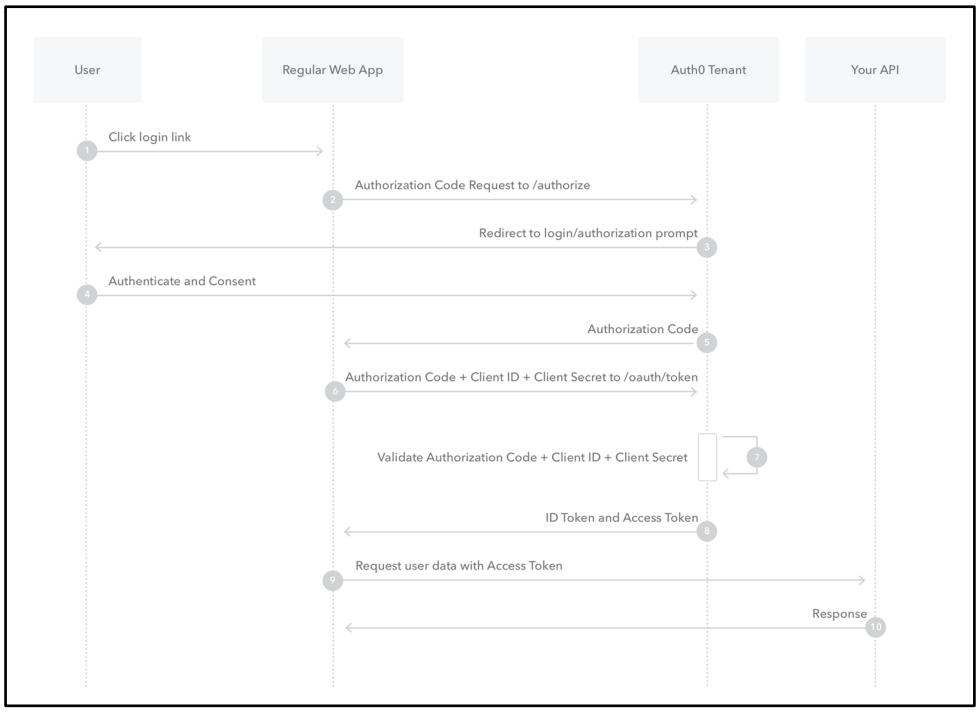
Fluxos do OAuth 2.0

São previstos diferentes possíveis fluxos para o protocolo OAuth 2.0. Neste capítulo, abordaremos quatro deles, que são:

- Authorization Code Flow;
- Implicit Flow;
- Resource Owner Password Flow;
- Client Credentials Flow.

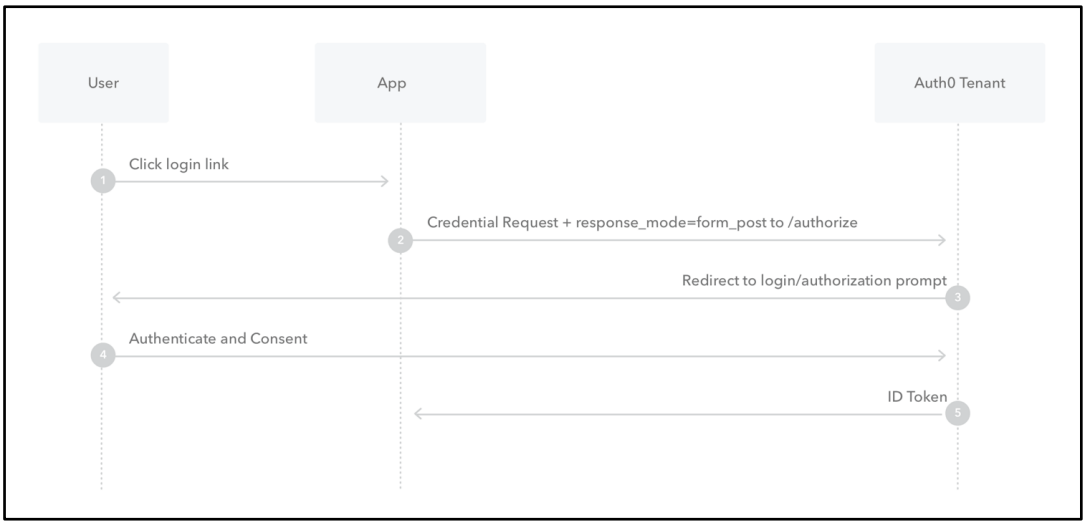
O Authorization Code Flow também é chamado de Web Application Flow ou simplesmente fluxo para servidores. O fluxo se inicia com a aplicação cliente redirecionando o navegador do usuário para o servidor de autorização. O servidor de autorização verifica a autenticação o usuário e esse escolhe se autoriza ou não o acesso. Caso o usuário autorize a requisição, esse é redirecionado utilizando a URL do cliente previamente cadastrada. Nesse momento, o código de autorização é passado como parâmetro à aplicação cliente. Utilizando o código de autorização, o cliente faz a requisição pelos tokens de acesso e renovação para o servidor de autenticação. O cliente recebe o token e a partir de então se comunica com o servidor de recursos. De tempos em tempos, uma requisição deve ser enviada para o servidor de autorização utilizando o token de renovação.

Figura 11 – Authorization Code Flow.



O Implicit Flow possui uma grande diferença em relação ao flow para servidores: ao invés de duas requisições, apenas uma requisição é realizada, e o cliente já recebe o token. Esse fluxo se inicia quando o cliente redireciona o usuário para o servidor de autorização. O servidor de autorização vai, então, autenticar o usuário dono do recurso e solicitar sua permissão para conceder acesso ao cliente. Caso a permissão seja concedida, o usuário é redirecionado para o endereço de redirecionamento do cliente. Após ser redirecionado, as informações são finalmente enviadas pelo usuário à aplicação cliente. Não é um fluxo recomendado atualmente por questões de segurança.

Figura 12 – Implicit Flow.



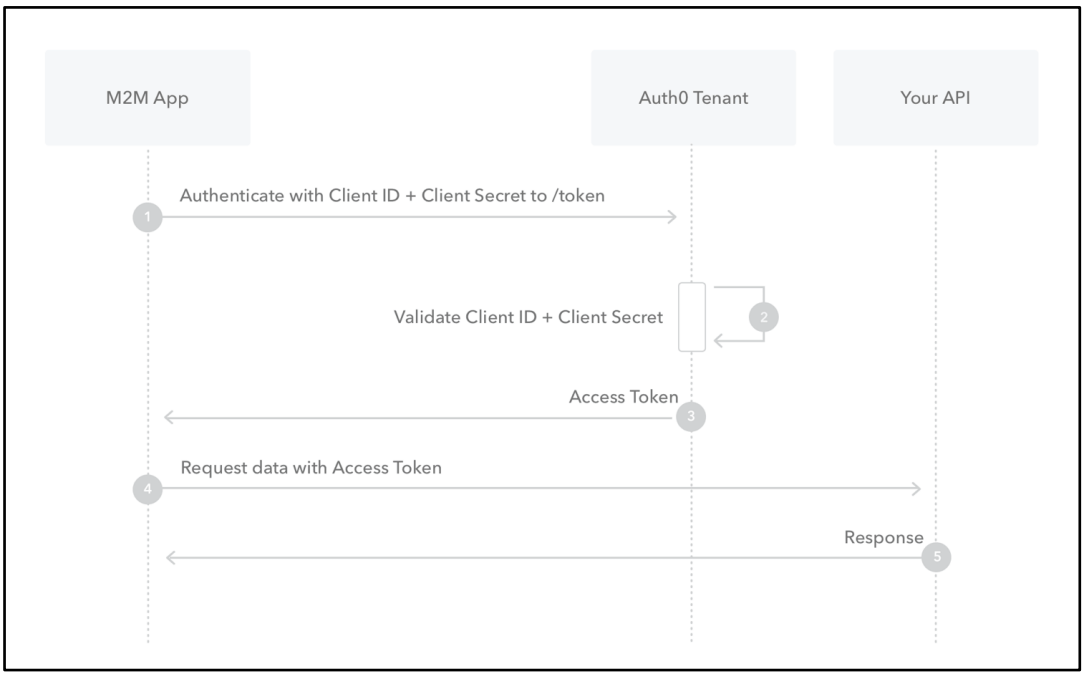
O Resource Owner Password Flow permite que o usuário envie seu login e sua senha para receber um token de acesso e, opcionalmente, um de renovação. Esse fluxo é bastante diferente dos demais modelos do OAuth. A primeira, e mais chamativa das diferenças, é a necessidade de acesso à senha do usuário por parte do cliente. Isso requer uma confiança muito grande do usuário dono do recurso na aplicação cliente. Por essa razão, esse é um modelo que deve ser evitado, e quando utilizado, que seja com muita cautela. Um exemplo de situação possível de uso é quando tanto o servidor de autenticação quanto o aplicativo cliente são produzidos pela mesma empresa. Por exemplo, você pode fornecer usuário e senha ao aplicativo mobile do Facebook sem nenhum problema, uma vez que a verificação de autorização será realizada no servidor da mesma empresa.

Os demais fluxos do OAuth são utilizados para delegar autorização de um recurso para acesso por parte de uma aplicação cliente. Há, porém, casos em que o cliente é o dono dos dados e não necessita de autorização de um dono de recurso. Ou ainda, casos em que o acesso ao recurso já foi fornecido em um contexto externo ao fluxo OAuth.

Se imaginarmos situações nas quais arquivos utilizados por uma aplicação estão armazenados externamente, fica fácil imaginar que será necessário obter um token de acesso por parte da aplicação cliente de forma direta. Nesses casos, ao invés de agir em nome de um usuário, a aplicação é a própria dona do recurso e,

portanto, realiza requisições em seu próprio nome. Além disso, os recursos em questão não dependem do usuário em questão, mas sim da aplicação cliente.

Figura 13 – Client Credentials Flow.



Por fim, cabe destacar que, nesse modelo, não há uso de token de renovação. Também não é requisitado ao usuário fornecer suas credenciais em nenhum momento.

Referências

BEZERRA, Débora de Jesus; MALAGUTTI, Pedro Luiz; RODRIGUES, Vânia Cristinada Silva. *Aprendendo Criptologia de Forma Divertida*. Paraíba: UFPB, 2010. Disponível em: <http://www.mat.ufpb.br/bienalsbm/arquivos/Oficinas/PedroMalagutti-TemasInterdisciplinares/Aprendendo_Criptologia_de_Forma_Divertida_Final.pdf>. Acesso em: 28 mai. 2021.

BOYD, Ryan. *Getting Started with OAuth 2.0* - Programming Clients for Secure WebAPI Authorization and Authentication. 1. ed. O'Reilly Media, 2012.

CERT.br. *Cartilha de Segurança para Internet*. Disponível em: <https://cartilha.cert.br/livro/cartilha-seguranca-internet.pdf>. Acesso em: 28 mai. 2021.

F5 DevCentral. YouTube. Disponível em: <https://www.youtube.com/channel/UCtVHX3fmQVjVgj_cGRIxRSg>. Acesso em: 28 mai. 2021.

GRIGORIK, Ilya. SURMA. *Introdução a HTTP/2*. Google Web Fundamentals. Disponível em: <https://developers.google.com/web/fundamentals/performance/http2/?hl=pt-br>. Acesso em: 28 mai. 2021.

KHAN ACADEMY. *Uma jornada pela criptografia*. Disponível em: <https://pt.khanacademy.org/computing/computer-science/cryptography>. Acesso em: 28 mai. 2021.

MDN Web docs. *HTTP*. Página inicial. Disponível em: <https://developer.mozilla.org/pt-BR/docs/Web/HTTP/>. Acesso em: 28 mai. 2021.

SPASOVSKI, Martin. *OAuth 2.0, Identity and Access Management Patterns* – Apractical hands-on guide to implementing secure API authorization flow scenarios with OAuth 2.0. Packt Publishing, 2013