Node.js e JWT: autenticação com tokens: Aula 1 - Atividade 7 Para saber mais: o funcionamento do bcrypt | Alura

2-2 minutes

-
- Sugerir alteração

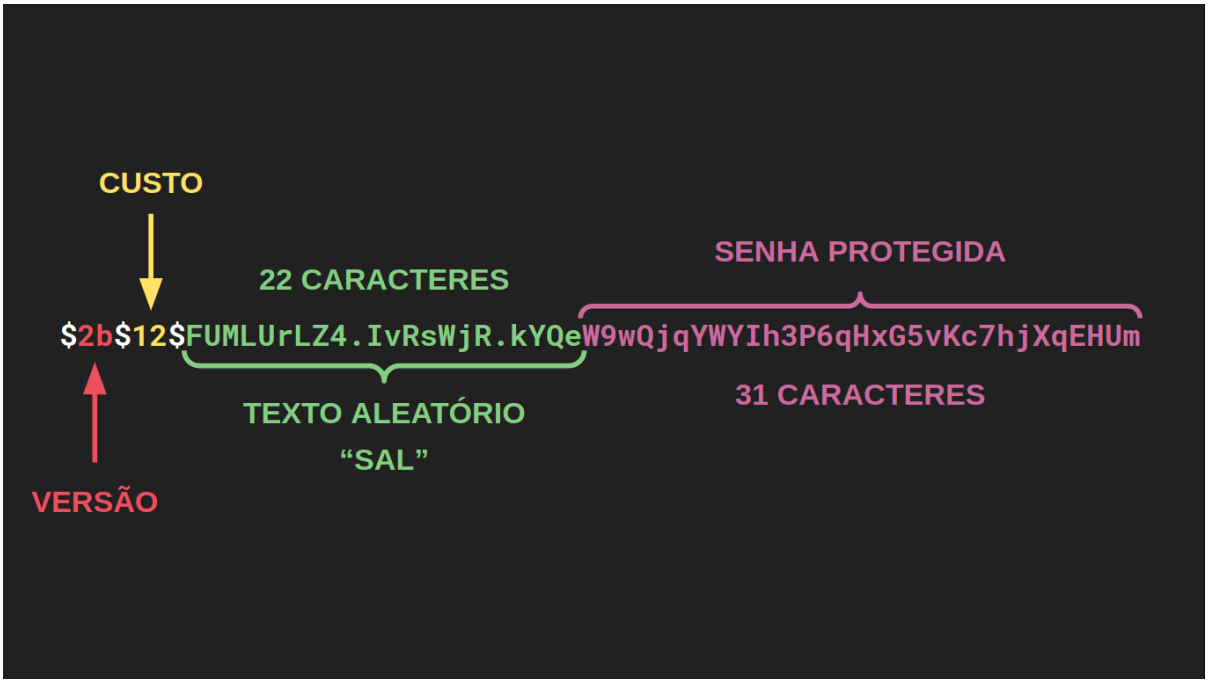
Olhando mais atentamente para os valores de `senhaHash` dos usuários, vemos que o resultado da função `bcrypt.hash` tem sempre o mesmo formato padrão. Isso é resultado de algumas propriedades específicas do algoritmo para tornar senhas seguras.

O primeiro prefixo, entre cifrões, é a **versão do algoritmo**. Pode incluir valores como `2b`, `2a` ou `2y`.

Em seguida, temos o **custo do algoritmo** que nós escolhemos. Quanto maior for esse custo mais demorado será para o algoritmo ser executado e, dessa forma, é mais difícil de ser vítima de **ataques de força bruta**, onde o atacante experimenta várias senhas na tentativa de encontrar a correta. É importante notar que a cada incremento que nós damos no custo, o algoritmo demora duas vezes mais. E para aplicações modernas, um custo de 12 é considerado suficiente.

Já os 22 caracteres depois do último cifrão formam um **texto aleatório**, frequentemente apelidado de ***salt*** (do inglês **"sal"**). Esse texto é misturado à senha no algoritmo que gera a senha protegida. Isso dificulta que atacantes consigam pré-computar tabelas de senhas na tentativa de reverter essas funções. O salt é gerado toda vez que chamamos 'bcrypt.hash`, por isso o resultado sempre muda.

Por fim, os últimos 31 caracteres são a senha protegida de fato.



Assim, por causa desse formato, o `bcrypt` consegue recuperar as informações necessárias para comparar a senha recebida do cliente com a `senhaHash` do seu usuário, guardada no banco de dados.