Utilização de *JSON Web Token* na Autenticação de Usuários em APIs REST

Lucas Souza Montanheiro[1], Ana Maria Martins Carvalho[1], Jackson Alves Rodrigues[2]

[1]Instituto Federal Goiano – Campus Morrinhos (IF Goiano)
Caixa Postal 92 – 75.650-000 – Morrinhos – GO – Brazil

[2]Departamento de TI – Grupo Privé
Rua do Balneário, Quadra 10, Lote 19 – Caldas Novas – GO – Brazil

lucasmontanheiro10@gmail.com, ana.carvalho@ifgoiano.edu.br, alves.j@live.com

Abstract. *The implementation of user authentication methods by developers is a complex task, sometimes done imperfectly and thus pose serious risks to current applications. Coding failures of these methods are considered critical and are generalized prevalence currently. This article proposes to use the open standard JSON Web Token for a correct and secure implementation of user authentication in web applications that use REST APIs.*

Resumo. *A implementação de métodos de autenticação de usuários por parte de desenvolvedores é uma tarefa complexa, algumas vezes é feita de forma imperfeita e, portanto, trazem sérios riscos para as aplicações atuais. Falhas na codificação desses métodos são consideradas críticas e tem uma prevalência generalizada atualmente. Esse artigo propõe a utilização do padrão aberto JSON Web Token para uma implementação correta e segura de autenticação de usuários em aplicações web que utilizam APIs REST.*

1. Introdução

Com o crescimento de equipamentos conectados, que se comunicam a sistemas, faz-se necessário que as informações que trafegam na rede sejam transmitidas de maneira segura e rápida. Pensando na velocidade da transmissão é necessário que se façam otimizações em serviços para que consumam poucos dados, graças a isso, o desenvolvimento de APIs REST está em ênfase na atualidade e seu uso vem crescendo a cada ano. Os navegadores modernos incluem o suporte nativo ao JSON (*JavaScript Object Notation*) o que vem colaborando para a sua adoção em aplicações [Pompeu *et al.* 2015] [Jones 2011] [Saudate 2014].

O REST (*Representational State Transfer*) é uma arquitetura de *web services* proposta em uma tese de doutorado de Roy Fielding. Tal arquitetura utiliza o protocolo de comunicação HTTP/HTTPS e formatos de dados mais simples, quando comparado ao seu antecessor, o SOAP (*Simple Object Access Protocol*), que utiliza exclusivamente o formato de dados XML (*eXtensible Markup Language*), que é mais complexo do que o formato popularmente utilizado em API com a arquitetura REST, o JSON. Uma vez que o formato de dados é mais simplificado, os esforços para a comunicação são menores ao utilizar a arquitetura REST, resultando em melhor performance quando

comparados a utilização de SOAP [Fielding 2000] [Ribeiro e Francisco 2016].

De acordo com a *Open Web Application Security Project* (OWASP), uma organização que tem por objetivo encontrar e combater as causas e riscos de falhas em aplicações *web*, a "Quebra de Autenticação e Gerenciamento de Sessão" é uma falha de prevalência generalizada e de severo impacto, sendo classificada como segunda falha mais perigosa em aplicações *web*. Esse portanto, é um dos principais motivos de se pesquisar e investir métodos seguros de autenticação de usuários em aplicações [OWASP 2013] [Souza 2012].

Desenvolvedores frequentemente implementam a autenticação em suas aplicações de maneira personalizada, mas a implementação correta é difícil, uma vez que muitos não protegem as credenciais de autenticação utilizando *hash* ou criptografia, e em determinadas situações utilizam uma criptografia fraca, que pode ser quebrada facilmente por um atacante. Algumas vezes, essas falhas são difíceis de serem encontradas, já que cada implementação é feita de maneira diferente da outra, dai a importância de se utilizar de bibliotecas e ferramentas específicas para a autenticação e controle de autorização de usuários [OWASP 2013]. Este artigo propõe a utilização do padrão aberto *JSON Web Token* para a implementação segura de um método de autenticação de usuários em aplicações *web*, como será apresentado seguir.

2. JSON Web Token

O *JSON Web Token* (JWT) é um padrão aberto (RFC 7519), que tem por objetivo definir um modo compacto e independente, que pode ser enviado dentro de um cabeçalho HTTP e conter as informações do usuário, para a transmissão segura de informações entre cliente e servidor, através de um objeto JSON. O *token* gerado pelo JWT é salvo no dispositivo do usuário e suas informações podem ser verificadas a cada solicitação, pois são criptografadas utilizando um segredo, através do algoritmo HMAC (*Hash-based Message Authentication Code*) ou de um par de chaves públicas e privadas, garantindo assim a sua confiabilidade [Jones *et al.* 2015] [Schneider 2016].

Várias bibliotecas que implementam o padrão JWT estão disponíveis para as mais variadas linguagens de programação, dentre elas: .NET, Python, NodeJs, Java, JavaScript, Perl, Ruby e PHP. As bibliotecas podem ser usadas tanto para autenticação de usuários, que é o cenário mais comum, sendo que a cada solicitação de uma rota, serviço ou recurso o usuário envia junto o *token*, permitindo seu acesso ou não com base nesse *token*, quanto para transmitir informações de forma segura, uma vez que com base em sua assinatura criptografada é possível verificar se o conteúdo não foi adulterado [JWT 2017] [Jones 2011].

É possível implementar o *JSON Web Token* em uma aplicação através do serviço chamado Auth0, que implementa o JWT em seu código e libera à seus clientes uma API na qual o cliente se cadastra por meio do serviço, podendo inclusive usar contas de terceiros como Google, Facebook e Twitter, e é enviado para a aplicação apenas as informações do usuário, uma vez que a senha e outros dados são guardados com segurança pelo Auth0. O serviço é utilizado por grandes empresas, como Nvidia, AMD, Mozilla, Dow Jones, entre outras [Auth0 2017].

3. Abordagem, experimentos e resultados

Para os experimentos mostrados a seguir, foi desenvolvido uma API REST de um

sistema genérico de consulta de produtos, utilizando a linguagem de programação Java, com a biblioteca que implementa o *JSON Web Token* para essa linguagem. Também utilizou-se da biblioteca Gson, criada pelo Google para converter um objeto JSON em um objeto Java e vice-versa. Para servidor de aplicação utilizou-se o Tomcat em sua versão 7.

Foi implementada uma classe, com o nome de *Token*, a qual possui os dois principais métodos para a utilização da biblioteca do JWT. O primeiro método, como pode-se ver no Quadro 1, é responsável pela geração do *token*, ele recebe como parâmetros o *issuer*, que pode ser utilizado tanto para identificar a aplicação que emitiu o *token*, quanto para identificar o nível de acesso de usuário na aplicação, que é nesse caso que foi utilizado. Logo em seguida é possível notar os parâmetros *idSubject* e *hours*, que são responsáveis respectivamente por guardar o identificador do usuário na aplicação e validade do *token* que foi gerado e organizado em horas.

Quadro 1. Método que gera o *Token*

```
1.  public String Generate(String issuer, int idSubject, int hours) {
2.     SignatureAlgorithm signatureAlgorithm = SignatureAlgorithm.HS256;
3.     long ttlMillis = hours * 3600000;
4.     String subject = String.valueOf(idSubject);
5.     long nowMillis = System.currentTimeMillis();
6.     Date now = new Date(nowMillis);
7.     byte[] apiKeySecretBytes = DatatypeConverter
8.           .parseBase64Binary(Parameters.TOKENKEY);
9.     Key signingKey = new SecretKeySpec(apiKeySecretBytes,
10.          signatureAlgorithm.getJcaName());
11.    JwtBuilder builder = Jwts.builder().setIssuedAt(now)
12.                        .setSubject(subject)
13.                        .setIssuer(issuer)
14.                        .signWith(signatureAlgorithm, signingKey);
15.    if (ttlMillis >= 0) {
16.    long expMillis = nowMillis + ttlMillis;
17.       Date exp = new Date(expMillis);
18.       builder.setExpiration(exp);
19.    } return builder.compact();
20. }
```

3.1. Estrutura do *Token* gerado

Um *token* gerado pela aplicação tem sua estrutura divida em três partes, cada parte é separada por um ponto final (“.”) e contém informações de diferentes tipos. A primeira parte é identificada como *header*, que é o cabeçalho do *token* e contém informações a respeito da criptografia usada para assinar o *token*. Pode-se observar no Quadro 2, a primeira parte do *token* em JSON, e no Quadro 3 a primeira parte do *token* codificada em *base64*.

Quadro 2. Cabeçalho do *Token* em JSON

```
1. {
2.  "alg": "HS256"
```

```
3. }
```

Quadro 3. Cabeçalho do *Token* em *base64*

```
eyJhbGciOiJIUzI1NiJ9
```

A segunda parte é conhecida como *payload*, que é a parte onde ficam as informações no *token*, geralmente informações do usuário que aquele *token* pertence e a sua validade. No *token* gerado as informações encontradas foram a "*iat*", que é o momento em que foi gerado, contado em segundos desde às 00:00 de 01/01/1970. A informação de "*sub*" que define do *subject* do *token*, isso é, o número de identificação do sujeito para quem o *token* pertence. Já a informação de "*iss*" é o *issuer* que foi definido para identificar o nível de acesso daquele usuário à aplicação. A última informação encontrada nessa parte é a "*exp*" que define o tempo de expiração do *token*, usando a contagem de tempo da mesma maneira que o anterior. Nos Quadros 4 e 5 pode-se observar a segunda parte do *token* em JSON e codificada em base64, respectivamente.

Quadro 4. *Payload* do *Token* em JSON

```
1. {
2.   "iat": 1491609751,
3.   "sub": "2",
4.   "iss": "client",
5.   "exp": 1491717751
6. }
```

Quadro 5. *Payload* do *Token* em *base64*

```
eyJpYXQiOjE0OTE2MDk3NTEsInN1YiI6IjIiLCJpc3MiOiJjbGllbnQiLCJleHA
iOjE0OTE3MTc3NTF9
```

A terceira e última parte do *token* é a *signature* que guarda uma assinatura criptografada com base no algoritmo especificado no cabeçalho. Para criar a assinatura, o algoritmo usa como base os dados das duas primeiras partes do *token*, que são os dados do tipo da criptografia, os dados de *payload*, e uma chave definida ao gerar o *token*. Essa chave em especial é definida em texto puro e o algoritmo a converte para *base64*, o resultado da assinatura em binário também é convertido para *base64*.

No caso da aplicação desenvolvida, o *token* foi criptografado usando o padrão HMAC SHA256, que foi criado pela Agência de Segurança Nacional dos Estados Unidos (NSA) e que até o momento dessa pesquisa não havia sido quebrado e é considerado uma criptografia segura. No Quadro 6 pode-se observar a última parte do *token*.

Quadro 6. Assinatura do *Token* criptografado em SHA256 e transformado em *base64*

```
VnEwyZdBzekCIycPoezPp_Hzi0sD4BmM_KOFnMyMhBo
```

No Quadro 7 pode-se observar a estrutura completa do *token* gerado como

resultado do método *Generate*, com suas três partes juntas. O código completo do Quadro 7 é enviado ao servidor, através do cabeçalho de uma requisição do protocolo HTTP, para conferir a autenticidade do usuário que faz a solicitação.

Quadro 7. *Token* completo

```
eyJhbGciOiJIUzI1NiJ9.eyJpYXQiOjE0OTE2MDk3NTEsInN1YiI6IjIiLCJpc3M
iOiJjbGllbnQiLCJleHAiOjE0OTE3MTc3NTF9.VnEwyZdBzekCIycPoezPp_Hzi
0sD4BmM_KOFnMyMhBo
```

3.2. Verificação do *Token* gerado

O segundo método da classe *Token* é o *getSubject*, que tem por objetivo validar o *token* e o tipo de permissão do usuário recebidos por parâmetro, e retornar à aplicação o código identificador do usuário na aplicação, caso aja algum erro durante a validação do *token* será retornado o valor zero e se tipo do usuário não for o mesmo especificado no parâmetro, será retornado uma exceção. No Quadro 8 pode-se observar o método *getSubject* da classe *Token*.

Quadro 8. Método *getSubject* da classe *Token*

```
1. public int getSubject(String jwt, String type) throws Exception{
2.       try{
3.          Claims claims = Jwts.parser()
4.             .setSigningKey(DatatypeConverter
5.             .parseBase64Binary(Parameters.TOKENKEY))
6.             .parseClaimsJws(jwt).getBody();
7.          if(!claims.getIssuer().equals(type))
8.             throw new Exception("Token invalido.");
9.          return Integer.parseInt(claims.getSubject());
10.      } catch (ExpiredJwtException | MalformedJwtException |
11.            SignatureException | UnsupportedJwtException |
12.            IllegalArgumentException e) {
13.         System.out.println(e.getMessage());
14.         return 0;
15.      }
16. }
```

3.3. Autenticação usando *JSON WEB Token*

A autenticação do usuário se dará por meio de um envio de uma requisição HTTP, do tipo POST, ao servidor de aplicação, o qual no corpo da requisição enviará o usuário e senha para *login*. Os dados serão validados pelo banco de dados e se coincidirem será gerado um *token* para aquele usuário. Esse *token* será enviado de volta para o usuário, o qual pode armazená-lo em *cookies* ou no armazenamento de sessão do dispositivo, podendo ser usado tanto em navegadores, quanto em aplicações móveis. O diagrama representado na Figura 1 demonstra esse processo.

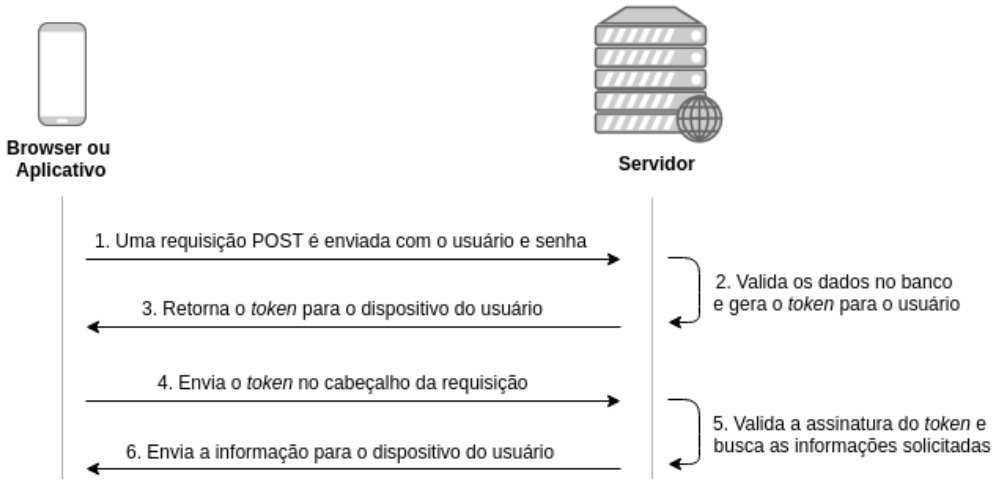

Figura 1. Diagrama de autenticação e validação de *token*

Ainda de acordo com o diagrama da Figura 1 é possível observar que os passos de 1 a 3 no diagrama representam o momento de *login* e os passos de 4 a 6 representam as solicitações seguintes, de rota, serviço ou recurso da aplicação. Com o *token* armazenado no dispositivo do usuário, a cada requisição ele será enviado dentro do cabeçalho da requisição HTTP, ao chegar a requisição no servidor será validada a assinatura do *token*, se a assinatura for autentica, a aplicação irá retornar os dados solicitados, caso o *token* tenha se expirado ou a sua assinatura não seja válida o servidor não irá retornar os dados solicitados.

Na aplicação desenvolvida, o método do recurso responsável pelo *login*, recebe do corpo da requisição os dados de e-mail e senha do cliente, que é o usuário, esses dados são validados no banco de dados e caso sejam válidos será gerado um *token* para o cliente com validade de 24 horas. O método de *login* de clientes pode ser observado no Quadro 9.

Quadro 9. Método de *login* de usuários

```
1.  @POST
2.  @Consumes(MediaType.APPLICATION_JSON)
3.  @Path("/login")
4.   public String login(String body) throws SQLException, Exception {
5.      Gson gson = new Gson();
6.      Client c = gson.fromJson(body, Client.class);
7.      c = ClientDAO.retreave(c.getEmail(), c.getPassword());
8.      String token = new Token().Generate("client", c.getId(), 24);
9.      return token;
10.  }
```

No Quadro 10 pode-se observar o recurso do *web service* para consulta de produto através do código de barras, o dispositivo do cliente faz a requisição GET na rota */eancode*, enviando o *token* no cabeçalho e o código do produto como parâmetro. O *token* é validado, e se autêntico é feita a consulta no banco de dados e retorna o produto ao dispositivo do usuário, convertido de objeto Java para objeto JSON.

Quadro 10. Método de consulta de produtos por código de barras

```
1.  @GET
2.  @Produces(MediaType.APPLICATION_JSON)
3.  @Path("/eancode")
4.  public String eancode(@HeaderParam("token") String token,
5.          @QueryParam("search") String eancode)
6.          throws SQLException, Exception {
7.      int id = new Token().getSubject(token, "client");
8.      if(id == 0) throw new Exception("Token invalido.");
9.      Gson gson = new Gson();
10.     Product p = ProductDAO.retreaveByBarcode(eancode);
11.     return gson.toJson(p);
12.  }
```

4. Conclusão

Cada vez mais, informações sensíveis são transmitidas através de aplicações *web* e a utilização de *JSON Web Token* nesse cenário é de suma importância, pois possui a garantia de que o usuário que solicita a informação é autentico e tem a autorização necessária para acessá-la, sendo que a verificação é feita a toda requisição.

Com a utilização do *JSON Web Token* no controle de usuários em APIs REST é possível garantir que o usuário que faz a solicitação de dados possui os níveis de acesso necessário para a leitura daquela informação, e sem a necessidade de consultar as informações e credenciais de acessos do usuário no banco de dados em toda requisição, uma vez que o próprio *token* é capaz de identificar o usuário e seu nível de acesso, reduzindo assim o processamento do servidor.

A implementação de um padrão aberto e de uma biblioteca *open source* para o controle de autenticação tem como vantagem uma grande comunidade utilizando e melhorando-a, sendo mais fácil de erros serem encontrados e corrigidos, e obtendo a garantia de que os métodos de criptografia utilizados sejam seguros, uma vez que qualquer interessado pode auditar o código e sugerir correções, sem a necessidade do desenvolvedor que a utilize entenda de todas as características que a envolva. Como trabalhos futuros, pretende-se realizar comparações entre o *JSON Web Token* e outros padrões disponíveis na literatura e comumente utilizado por desenvolvedores, como forma de comprovar a eficácia e a segurança na prática da utilização do método de autenticação apresentado neste artigo.

Referências

Auth0 (2017) “*Single Sign On & Token Based Authentication*”. Disponível em: <https://auth0.com/>. Acesso em: 15 Março 2017.

Fielding, R. T. (2000) “*Architectural Styles and the Design of Network-based Software Architectures*”. Tese de Doutorado, Universidade da Califórnia. Califórnia, Estados Unidos da América.

Jones, M. B. (2011) “*The emerging JSON-based identity protocol suite*”. *W3C workshop on identity in the browser*. p. 1-3.

Jones, M., Bradley, J., Sakimura, N. (2015) "*JSON Web Token (JWT) – RFC 7519*", *Internet Engineering Task Force*, IEFT. Disponível em: <https://tools.ietf.org/pdf/rfc7519.pdf>. Acesso em: 15 Março 2017.

JWT (2017) "*JSON Web Token Introduction*". Disponível em: <https://jwt.io/introduction/>. Acesso em: 15 Março 2017.

OWASP (2013) "*OWASP Top 10 2013 – The Ten Most Critical Web Application Security Risk*". Disponível em: <https://www.owasp.org/images/f/f8/OWASP_Top_10_-_2013.pdf>. Acesso em: 10 Março 2017.

Pompeu, I. F., Matos, F. B., Melo, M. A. (2015) "Testando a performance de tecnologias especialistas em *Rest API JSON* uma abordagem em JAVA, PHP, C++, NODEJS, RUBY, PYTHON e GO", Encontro Anual de Computação, EnAComp 2015, p. 147-154.

Ribeiro, M. F. e Francisco, R. E. (2016) "Web Services Rest – Conceitos, Análises e Implementação", Revista Educação, Tecnologia e Cultura, ETC, v. 14, n. 14.

Saudate, A. (2014) REST: Construa API's inteligentes de maneira simples. Editora Casa do Código.

Schneider, A. H. (2016) "Desenvolvimento web com Client Side Rendering: combinando Single Page Application e serviços de backend". Monografia, Universidade Federal do Rio Grande do Sul. Rio Grande do Sul, Brasil.

Souza, L. L. (2012) "Desenvolvimento seguro de aplicações web seguindo a metodologia OWASP". Monografia, Universidade Federal de Lavras. Minas Gerais, Brasil.