

Wireless Monitor - Aplicativo web livre para receber e mostrar dados vindos de equipamentos IoT

Abstract. *This article presents the Wireless Monitor app that has the goal to allow embeded systems developers to send data to the cloud collected by an IoT device and preview in a web browser. In devices that use microcontrollers the figure of a monitor (screen) doesn't exist, which one can check the commands output e follow its execution, there's just serial outputs and embed wi-fi boards. From there rises the need to create systems that could gather data sended by this devices e show them in a appropriate way, in a web browser for example.*

Resumo. *Este artigo apresenta o aplicativo Wireless Monitor que tem o objetivo de permitir que desenvolvedores de sistemas embarcados possam enviar para a nuvem os dados obtidos por equipamento IoT e visualizá-los no navegador de internet. Em equipamentos que usam microcontroladores não existe a figura de um monitor (tela), em que se possa verificar a saída dos comandos e acompanhar sua execução, existem apenas saídas seriais ou placas wi-fi embutidas. Daí surge a necessidade de criar sistemas que possam recolher os dados enviados por esses equipamentos e mostrá-los de forma apropriada, como por exemplo em um navegador de internet.*

1. Introdução

Com a crescente adoção de equipamentos IoT (Internet of Things) para monitoramento de sensores e acionamento de cargas, cresce também a necessidade de ambientes de acompanhamentos de tais medições. Para isso uma das melhores formas é usar a nuvem - recurso computacional sob demanda através da internet [Cloud 2015] - para fazer o armazenamento, já que uma das características dos equipamentos IoT é o acesso à internet. Para atender essa necessidade surge a ideia de criar um aplicativo web e livre que possa captar informações destes dispositivos e que o acesso possa acontecer em qualquer lugar.

Equipamentos IoT são dispositivos com internet que podem se interligar e se comunicar uns com os outros [Revell 2013]. Como exemplo temos o Raspberry Pi que em quase 5 anos já vendeu 10 milhões de unidades pelo mundo [Upton 2016].

2. Objetivos

O objetivo principal do Wireless Monitor é fornecer uma *API (Application Program Interface)* leve, simples e segura, visto que equipamentos IoT são limitados, para enviar e receber informações da nuvem.

Para que haja melhor intercâmbio das informações tanto partindo do equipamento IoT quanto chegando, o protocolo de comunicação escolhido foi o JSON (*JavaScript Object Notation*), que segundo Douglas Crockford é um formato leve e de linguagem independente para troca de informações [Crockford 2015].

3. Justificativa

Sendo um aplicativo de código-fonte licenciado pela GPLv3 (GNU Public License) poderá ser usado tanto para professores e alunos de cursos superiores e técnicos para estudo

de microcontroladores, sistemas embarcados e afins, como para empresas ou pessoas que queiram interagir com seus equipamentos pessoais.

A linguagem de programação escolhida foi o PHP, a qual é fácil de aprender, normalmente lecionada em cursos superiores e técnicos e de hospedagem barata.

Outra característica a ser levada em conta é a forma de autenticação. Uma autenticação convencional envolve a troca de *cookies* entre servidor e cliente, além de espaço em disco para guardar tais informações. Em sistemas IoT que se supõem que possam crescer de forma rápida, ou seja, o número de equipamentos pode aumentar, é necessário um sistema de autenticação capaz de ser escalável mesmo em condições limitadas. Para isso foi utilizado o padrão JWT (ou *JSON Web Tokens*), que é um padrão aberto (RFC 7519 [Michael B. Jones and Sakimura 2015]) que define uma maneira compacta e auto-contida de transmitir de forma segura informações entre pares através de um objeto JSON [JWT 2016]. Esta informação pode ser verificada e confirmada pois é assinada digitalmente. Informações JWT podem ser assinadas usando um segredo (com o algoritmo HMAC [Hugo Krawczyk and Canetti 1997]) ou um par de chave pública e privada usando RSA [Jonsson and Kaliski 2003].

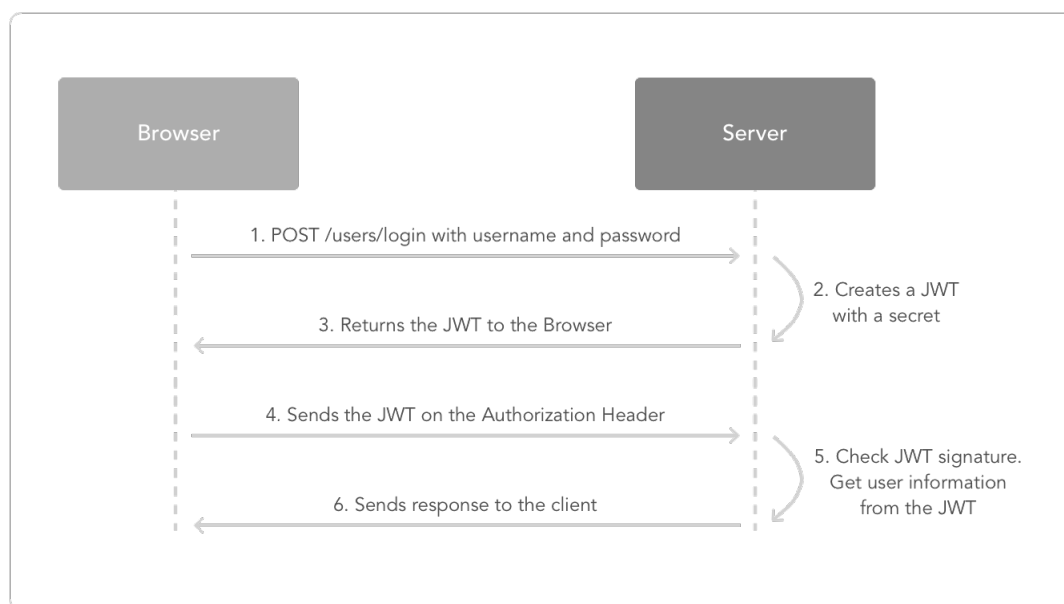


Figura 1. Diagrama do processo de autenticação - Fonte:
<https://cdn.auth0.com/content/jwt/jwt-diagram.png>

4. Revisão Teórica

Muitas são as soluções de monitoramento de equipamentos IoT. Podemos citar as plataformas Oracle IoT, AWS IoT, Google Cloud IoT e Microsoft Azure IoT Suite que são desenvolvidas por grandes empresas como Oracle, Amazon, Google e Microsoft, respectivamente; até soluções livres como Kaa, ThingSpeak, macchina.io, SiteWhere [Postscapes 2016].

O grande desafio é permitir a extensão da ferramenta para necessidades específicas. Ferramentas como o Kaa permitem criar módulos próprios, sistemas de

análises e modelo de dados, fazendo com que a ferramenta se adapte ao que você precisa [Kaa 2014].

De forma semelhante outras ferramentas como *macchina.io* oferecem opções de criar *bundles* [Macchina.io 2016], o ThingSpeak oferece opção de criar *apps*, que podem envolver visualização em gráficos e tomada de decisões [ThingSpeak 2016].

Nesse sentido a ferramenta proposta possui um sistema de plugins, que são desenvolvidos como *Laravel Packages* [Laravel 2016]. Cada nova funcionalidade é criada através da ferramenta *Laravel* e pode ser desenvolvida e habilitada localmente.

A proposta é ter uma tela de acompanhamento dos dados captados do equipamento que possam ser visualizados de forma específica. A documentação em português do Brasil para criar um novo plugin pode ser encontrada em <https://sanusb-grupo.github.io/wireless-monitor/pt-br/plugin-development.html>.

4.1. Comparativo com outras ferramentas livres

Aplicativo	Ambiente do Servidor	Suporte a plugins	SDK
Kaa	Java	Sim	Sim
macchina.io	C++/NodeJS	Sim	Sim
SiteWhere	Java	Sim	Sim
ThingSpeak	Ruby	Sim	Sim
Wireless Monitor	PHP	Sim	Não

Tabela 1: Comparativo com outras ferramentas

Como pode ser visto na Tabela 1, aplicativos como Kaa e SiteWhere rodam em ambiente Java, o que encarece a implantação devido a hospedagem ser mais cara comparada a hospedagem de aplicativos PHP, bem como o poder de processamento deve ser elevado visto que tal ambiente exige um servidor mais robusto.

Mesmo não possuindo um SDK (*Software Development Kit*), para auxiliar os desenvolvedores a utilizar o Wireless Monitor, mas por ter um número baixo de *endpoint's* e uma linguagem base bastante difundida e suportada por um grande número de linguagens como no caso do JSON, não é tão difícil criar funções de acesso a API.

Os *endpoint's* e suas descrições estão documentadas em <https://sanusb-grupo.github.io/wireless-monitor/pt-br/api-endpoints/index.html>.

5. Arquitetura

É necessário um servidor web, um equipamento IoT seja ESP8266 ou Raspberry Pi com Arduino e um navegador de internet no cliente.

A arquitetura segue o modelo da Figura 2, onde temos o equipamento IoT que

se comunica com o servidor através da internet enviando um *payload* no formato JSON e o cliente que busca os dados recolhidos do servidor e exibe no navegador, *desktop* ou *mobile*.

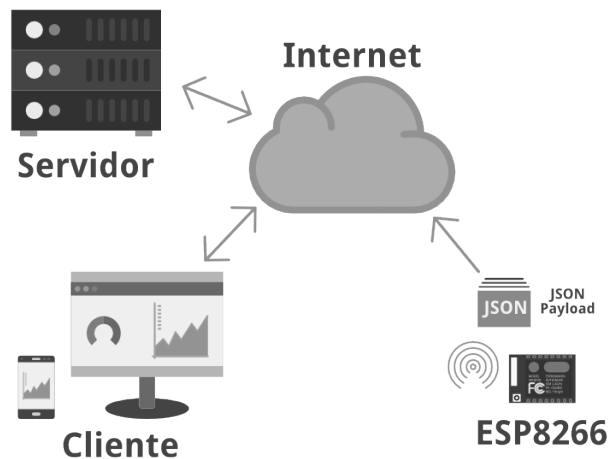


Figura 2. Arquitetura

5.1. Servidor

No servidor deve existir uma instância do Wireless Monitor, que trabalha em conjunto com um banco de dados PostgreSQL, onde são armazenadas todas as informações.

PostgreSQL é um ORDBMS (Object-relational database management system - Sistema de Gerenciamento de Banco de Dados Objeto-relacional) baseado no POSTGRES versão 4.2, desenvolvido na Universidade da Califórnia no Departamento de Ciência da Computação de Berkley. POSTGRES foi pioneiro em muitos conceitos que só se tornaram disponíveis em alguns sistemas de bancos de dados comerciais muito tempo depois [PostgreSQL 2016].

6. Aplicação

Nesta seção é mostrado um passo a passo de como o aplicativo funciona.

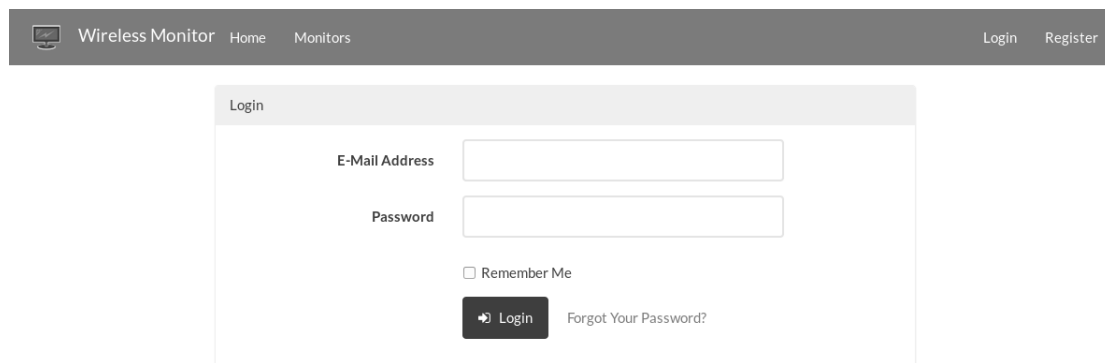
6.1. Cadastro do desenvolvedor

O desenvolvedor inicialmente deve fazer um cadastro simples no Wireless Monitor como pode ser visto na Figura 3. Esse cadastro irá criar para ele uma *api_key*, ou seja, uma chave única no formato UUID 4 [Paul J. Leach and Salz 2005].

6.2. Criar um *Monitor*

Um *Monitor* é um componente interno do sistema criado pelo desenvolvedor de acordo com sua necessidade, é o instrumento que caracteriza os dados coletados e os apresenta na interface web.

Imagine que o desenvolvedor queira medir a temperatura de um ambiente e acompanhar suas variações. Para isso ele deve criar um *Monitor* de Temperatura, que apenas



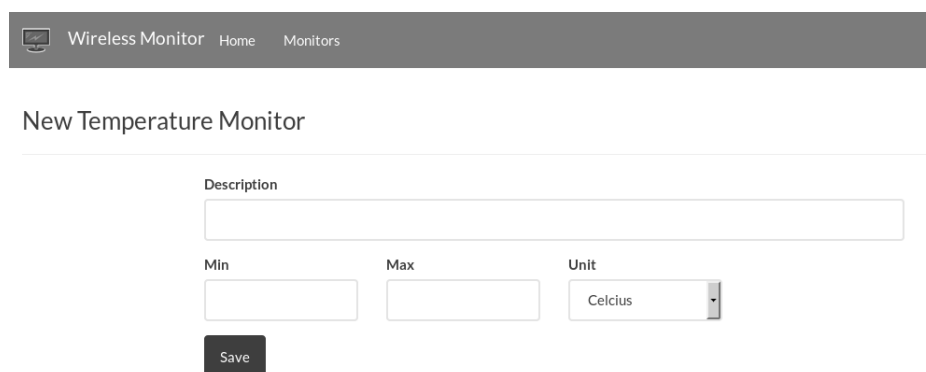
The image shows a web application header with a logo and navigation links: 'Wireless Monitor', 'Home', and 'Monitors'. On the right side of the header are links for 'Login' and 'Register'. Below the header is a 'Login' form. The form has a title 'Login' and contains two input fields: 'E-Mail Address' and 'Password'. Below these fields is a checkbox labeled 'Remember Me'. At the bottom of the form is a 'Login' button with a right-pointing arrow icon, and a link labeled 'Forgot Your Password?'.

Figura 3. Tela de Login

recebe um valor em intervalos de tempo. Dessa forma o desenvolvedor pode acompanhar as variações ou ainda ver em forma de gráfico um conjunto de variações de um período de tempo anterior.

De maneira análoga a criação de uma chave UUID para o desenvolvedor, uma chave UUID é criada para o Monitor - *monitor_key*.

As informações necessárias para criar um *Monitor* de temperatura podem ser visualizadas na Figura 4.



The image shows a web application header with a logo and navigation links: 'Wireless Monitor', 'Home', and 'Monitors'. Below the header is a form titled 'New Temperature Monitor'. The form has a 'Description' input field. Below the description field are three input fields: 'Min', 'Max', and 'Unit'. The 'Unit' field is a dropdown menu with 'Celcius' selected. Below these fields is a 'Save' button.

Figura 4. Novo Monitor de Temperatura

6.3. Autenticação do equipamento

Para autenticar e identificar o desenvolvedor e seu *monitor* é preciso enviar a *api_key* e a *monitor_key* via método *POST* para o *endpoint* */api/authenticate*. Em caso positivo o sistema irá retornar um *token*. Esse *token* servirá para qualquer troca de informações futuras entre o equipamento IoT e o Wireless Monitor.

Após ter o *token* o desenvolvedor deve passá-lo através da *Header HTTP* denominada *Authorization* usando *schema Bearer*. Algo do tipo:

Authorization: Bearer <token>

Um *token* é formado pelas seguintes informações:

- *Header*
- *Payload*
- *Signature*

Essa é uma forma segura e com pouco custo de memória. Além de ser uma forma de autenticação *stateless*, em que não são usadas sessões e nem mesmo *cookies*.

Para testar a autenticação fora do equipamento IoT é possível utilizando a ferramenta `cURL`, que é uma ferramenta de linha de comando e biblioteca para transferência de dados com sintaxe URL [Stenberg 1996].

```
curl -i -X POST \
  -F 'api_key=fa3076b3-ddb3-421f-a0ed-303a8dd04fb8' \
  -F 'monitor_key=e98fb37c-e79c-4a80-ac7f-b8fbdb82d48b' \
  http://localhost:8000/api/authenticate
```

O servidor então responde informando o *token*:

```
HTTP/1.1 200 OK
Date: Tue, 01 Oct 2016 23:34:29 GMT
Server: Apache
Cache-Control: no-cache
X-RateLimit-Limit: 60
X-RateLimit-Remaining: 59
Connection: close
Transfer-Encoding: chunked
Content-Type: application/json
```

```
{ "token": "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJtY25pdG9yX2tleSI6ImU5OGZiMzdjLWU3OWMtNGE4MCIhYzdmLWI4ZmJkYjgyZDQ4YiIsInN1YiI6MSwiaXNzIjoiaHR0cDpcL1wvd2lyZWxlc3MtbW9uaXRvci5wcm92aXNvcmlvLndzXC9hcGlcl2F1dGhlbnRpyY2F0ZSIsImhhdCI6MTQ3NjIyODg2OSwiZXhwIjoxNDc2MjMyNDY5LCJuYmYiOiE0NzYyMjg4NjksImp0aSI6IjhhN2IwM2UxNGMzOTIhYmEwZTJjZDU0NDcwOTI5NDE2In0.hxuEY_F9lEg1UL0JA1FIh0L1qw29WoMFLXAVLfeVo4" }
```

6.4. Envio dos dados

Além do cabeçalho contendo o *token* o usuário deve passar os valores coletados pelo equipamento e enviar para o sistema. Para isso ele deve enviar uma requisição *POST* para o *endpoint* `/api/send`, com o atributo *data* contendo um JSON com os dados.

No exemplo do *Monitor* de temperatura é necessário enviar apenas o valor, algo do tipo:

```
{
  "value": 23.89
}
```

Usando o `cURL` a linha de comando deve ser algo como:

```
curl -i -X POST -H 'Content-Type: application/json' \
  -H 'Authorization: Bearer <TOKEN>' \
  -d '{"data":{"value":23.89}}' \
  http://localhost:8000/api/send
```

Onde <TOKEN> deve ser substituído pelo *token* recebido da autenticação.

6.5. Descrição dos componentes

Neste tópico são descritos os componentes e as ferramentas utilizadas para o desenvolvimento e uso do plugin de Temperatura.

6.5.1. Sistema embarcado linux

Foi utilizado a plataforma Raspberry Pi como sistema embarcado, que irá servir para comunicação com o Wireless Monitor e a plataforma Arduino. É um equipamento IoT capaz de interagir com outros dispositivos e com acesso à internet.

6.5.2. Microcontrolador

A plataforma Arduino foi escolhida para servir de ponte entre o componente de medição de temperatura e o sistema embarcado.

Arduino é uma plataforma livre de eletrônica baseado em *hardware* e *software* fáceis de usar. Placas Arduino são capazes de ler entradas - luz em um sensor, controle usando botões, etc. - e converter em uma saída - acionamento de um motor, acionamento de um LED, publicação de dados online [Arduino 2016].

6.5.3. Sensor de temperatura

Como sensor de temperatura foi usado o LM35 da Texas Instruments. A série LM35 é composta de dispositivos de circuito integrado para medição de temperatura com a tensão de saída linearmente proporcional a temperatura em graus celsius. O sensor LM35 tem a vantagem sobre sensores de temperatura linear calibrados em Kelvin, devido a não ser necessário subtrair uma alta tensão constante da saída para obter uma escala conveniente [Instruments 2016].

6.6. Ambiente de execução

Para esse exemplo o ambiente de execução escolhido foi o NodeJS, que é um envólucro (*wrapper*) do ambiente de execução JavaScript de alta performance chamado V8 usado no navegador Google Chrome. O NodeJS permite que o V8 funcione em contextos diferentes do browser, principalmente fornecendo APIs adicionais que são otimizadas para casos específicos [Hughes-Croucher and Wilson 2012]. Por exemplo no caso de equipamentos IoT é perfeito, pois se trata de um dispositivo orientado a eventos, assim como o NodeJS.

Para auxiliar na conversação entre o NodeJS e o Arduino foi usado a ferramenta Johnny-Five, uma plataforma livre Javascript para Robôs e IoT [Waldron 2012].

6.7. Princípios de execução

O NodeJS deve ser instalado no Raspberry Pi já que possui suporte a arquitetura ARM. Um projeto NodeJS deve ser criado tendo como dependências o Johnny-Five e uma biblioteca de requisições HTTP, como por exemplo `request` [Request 2016]. Dessa forma o Johnny-Five se encarregará de se comunicar com o Arduino requisitando a temperatura do componente LM35. Com a resposta em mãos o NodeJS irá enviar as medições ao Servidor através da biblioteca `request`.

O código fonte deste exemplo pode ser encontrado num repositório do GitHub [Alves 2016] e a montagem do projeto pode ser vista na Figura 5, onde temos o Raspberry PI (1), o Arduino (2) e o sensor de temperatura LM35 (3).

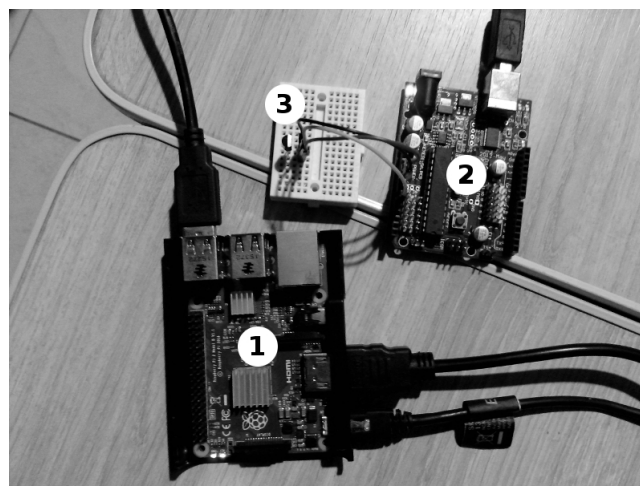


Figura 5. Montagem do projeto

6.8. Visualização dos dados

Após captar e enviar dados do IoT para a nuvem é possível acompanhar os resultados pelo sistema. A forma de visualização será como mostra a Figura 6.

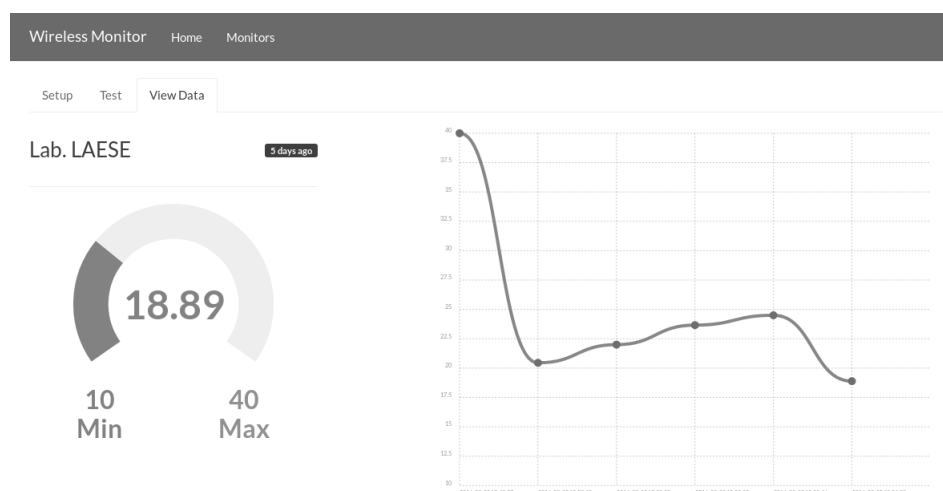


Figura 6. Visualização dos dados na web

7. Conclusão

A partir de ferramentas livres é possível criar ambientes de alta qualidade para monitoramento de dispositivos IoT. Tanto porque grande parte das ferramentas livres são estáveis e bem testadas, quanto pela liberdade de poder customizar para que a ferramenta atenda as necessidades dos envolvidos, diferentemente de ferramentas proprietárias.

Um passo importante e que intimida um pouco é a forma de autenticação. JWT é uma tecnologia muito recente e utiliza técnicas pouco convencionais para o público iniciante, mas temos que levar em conta que junto com o aumento do uso de equipamentos ligados à internet vem a necessidade de segurança na comunicação. Uma falha de segurança que se tornou comum nessas situações é chamada de *man-in-the-middle*, que pode ser definida como “Uma falha de segurança em um computador em que um usuário malicioso intercepta - e possivelmente altera - dados trafegando em uma rede” [Wordspy 2002]. Esse erro pode ocorrer simplesmente porque os vínculos entre aparelhos e criptografias de proteção cedidas por um padrão normalmente não foram implementadas corretamente, como por exemplo em travas eletrônicas que usam Bluetooth Low Energy (BLE) [Spring 2016].

Uma prática comum para autenticação de IoT's é a criação de tokens randômicos para identificar o usuário e o dispositivo, entretanto essa técnica facilita o ataque *man-in-the-middle*. Nessa linha o uso do JWT possui vantagens quando comparado com um token randômico:

- Chaves API randômicas não dizem nada a respeito do usuário, enquanto JWTs contém informações e metadados que descrevem a identidade do usuário; contém também uma validade por um período de tempo ou domínio.
- JWT não obriga a necessidade um emissor de token centralizado ou autoridade de revogação de token.
- É compatível com OAuth2 [Atwood et al. 2012].
- Dados do JWT podem ser inspecionados.
- JWTs possuem controles de expiração [Romero 2015].

Por fim o passo seguinte seria permitir o envio de comandos do navegador para o dispositivo, podendo assim controlar algumas funcionalidades remotamente como o acionamento de cargas, disparo de relés, entre outras funções.

Referências

- Alves, Á. C. (2016). Sensor de temperatura usando plataforma IoT Wireless Monitor. <https://github.com/atilacamura/wm-sensor-temperature>. [Online; accessed 23-September-2016].
- Arduino (2016). Getting Started - Introduction. <https://www.arduino.cc/en/Guide/Introduction>. [Online; accessed 27-September-2016].
- Atwood, M., Balfanz, D., Bounds, D., Conlan, R. M., Cook, B., Culver, L., de Medeiros, B., Eaton, B., Elliott-McCrea, K., Halff, L., Hammer, E., Laurie, B., Messina, C., Panzer, J., Quigley, S., Recordon, D., Sandler, E., Sergeant, J., Sieling, T., Slesinsky, B., and Smith, A. (2012). The OAuth 2.0 Authorization Framework. <https://tools.ietf.org/html/rfc6749>. [Online; accessed 23-September-2016].

- Cloud, I. (2015). What is cloud computing? <https://www.ibm.com/cloud-computing/what-is-cloud-computing>. [Online; accessed 27-September-2016].
- Crockford, D. (2015). JSON. <https://github.com/douglascrockford/JSON-js/blob/master/README>. [Online; accessed 13-September-2016].
- Hughes-Croucher, T. and Wilson, M. (2012). Node: Up and Running. <http://chimera.labs.oreilly.com/books/1234000001808/index.html>. [Online; accessed 21-September-2016].
- Hugo Krawczyk, M. B. and Canetti, R. (1997). HMAC: Keyed-Hashing for Message Authentication. <https://tools.ietf.org/html/rfc2104>. [Online; accessed 13-September-2016].
- Instruments, T. (2016). LM35 Precision Centigrade Temperature Sensors. <http://www.ti.com/lit/ds/symlink/lm35.pdf>. [Online; accessed 20-September-2016].
- Jonsson, J. and Kaliski, B. (2003). Public-Key Cryptography Standards (PKCS) 1: RSA Cryptography Specifications Version 2.1. <https://tools.ietf.org/html/rfc3447>. [Online; accessed 13-September-2016].
- JWT (2016). Introduction to JSON Web Tokens. <https://jwt.io/introduction/>. [Online; accessed 13-September-2016].
- Kaa (2014). Dev center - Complete application. <http://www.kaaproject.org/platform/#complete-application>. [Online; accessed 13-September-2016].
- Laravel (2016). Package Development. <https://laravel.com/docs/5.2/packages>. [Online; accessed 13-September-2016].
- Macchina.io (2016). Bundles Overview. <http://macchina.io/docs/00200-OSPBundles.html>. [Online; accessed 13-September-2016].
- Michael B. Jones, J. B. and Sakimura, N. (2015). JSON Web Token (JWT). <https://tools.ietf.org/html/rfc7519>. [Online; accessed 13-September-2016].
- Paul J. Leach, M. M. and Salz, R. (2005). A Universally Unique Identifier (UUID) URN Namespace. <https://tools.ietf.org/html/rfc4122>. [Online; accessed 17-September-2016].
- PostgreSQL (2016). PostgreSQL 9.6.0 Documentation. <https://www.postgresql.org/files/documentation/pdf/9.6/postgresql-9.6-A4.pdf>. [Online; accessed 27-September-2016].
- Postscapes (2016). IoT Cloud Platform Landscape. <http://www.postscapes.com/internet-of-things-platforms/>. [Online; accessed 13-September-2016].
- Request (2016). Simplified HTTP client. <https://github.com/request/request>. [Online; accessed 21-September-2016].
- Revell, S. (2013). Internet of Things (IoT) and Machine to Machine Communications (M2M) - Challenges and opportunities. <https://connect.innovateuk.org/documents/3077922/3726367/IoT+Challenges,%20final+paper>,

- %20April+2013.pdf/38cc8448-6f8f-4f54-b8fd-3babed877d1a.
[Online; accessed 27-September-2016].
- Romero, M. I. (2015). PHP Authorization with JWT (JSON Web Tokens). <https://www.sitepoint.com/php-authorization-jwt-json-web-tokens/>.
[Online; accessed 23-September-2016].
- Spring, T. (2016). Bluetooth Hack Leaves Many Smart Locks, IoT Devices Vulnerable. <https://threatpost.com/bluetooth-hack-leaves-many-smart-locks-iot-devices-vulnerable/119825/>. [Online; accessed 23-September-2016].
- Stenberg, D. (1996). cURL. <https://curl.haxx.se/>. [Online; accessed 27-September-2016].
- ThingSpeak (2016). Apps. <https://thingspeak.com/apps>. [Online; accessed 13-September-2016].
- Upton, E. (2016). Ten millionth Raspberry Pi, and a new kit. <https://www.raspberrypi.org/blog/ten-millionth-raspberry-pi-new-kit/>.
[Online; accessed 17-September-2016].
- Waldron, R. (2012). Johnny-Five: the JavaScript Robotics IoT Platform. <http://johnny-five.io/>. [Online; accessed 21-September-2016].
- Wordspy (2002). man in the middle attack. <http://wspy.ws/874>. [Online; accessed 23-September-2016].