International Journal of Electrical Engineering and Technology (IJEET)
Volume 11, Issue 5, July 2020, pp. 127-136, Article ID: IJEET_11_05_015
Available online at http://www.iaeme.com/ijeet/issues.asp?JType=IJEET&VType=11&IType=5
ISSN Print: 0976-6545 and ISSN Online: 0976-6553
DOI: 10.34218/IJEET.11.5.2020.015

INTEGRATION OF SPECIALIZED HARDWARE WITH CLOUD BASED SYSTEM

Todor Todorov
Faculty of Mathematics and Informatics,
St. Cyril and St. Methodius University of Veliko Tarnovo, Bulgaria, and
Institute of Mathematics and Informatics, BAS, Sofia, Bulgaria

Juri Stoinov
Faculty of Mathematics and Informatics,
St. Cyril and St. Methodius University of Veliko Tarnovo, Bulgaria

ABSTRACT

The paper describes the development of a cloud system and the mechanisms for connecting of specialized hardware to it. The connection is made with the help of a wireless module based on the ESP 8266 chip. The protocols for connection, identification and authorization of devices are described. Issues related to ensuring data integrity and validity and deflecting potential attacks are discussed.

Key words: Cloud based systems, Network protocols, Communication security.

Cite this Article: Todor Todorov and Juri Stoinov, Integration of Specialized Hardware with Cloud Based System. *International Journal of Electrical Engineering and Technology*, 11(5), 2020, pp. 127-136.
http://www.iaeme.com/IJEET/issues.asp?JType=IJEET&VType=11&IType=5

1. INTRODUCTION

Internet of Things or IoT is a system of connected devices with sensors and output transducers provided with unique identifiers and ability to operate in a network by sending and receiving data and also interacting with the surrounding environment autonomously, without human interface and interaction. The field has evolved because of the possible combinations between multiple other technologies including embedded computing, wireless networks, commodity sensors, control systems, automation, real-time data analytics and machine learning [3], [6]. Expert systems are applications developed to solve complex problems in a particular domain, at the level of extra-ordinary human intelligence and expertise [1]. Expert systems are well integrated in the animal health management [2], [4], [5].

The goal of this research is to explore the architecture for such device networks and its potential to reconstruct economic and social processes in such a way as to reduce or

completely exclude the human factor from the processes. We present integration of such specialized hardware to an expert system.

2. EXPERT SYSTEM FOR MILK AND ANIMAL MONITORING

We consider an expert system for milking process monitoring. It is also used for observations on animal health care [8]. Specialized tools of EkoMilk Horizon (hybrid mid-infrared and ultrasound analyser) are used for data input into the system [10].

Data is transferred from analyser to the system over internet via ESP8266 Wi-Fi module from Olimex [12]. We use HTTP protocol for communication with the system. It is a standard for all network equipment and firewalls have definitions to allow this type of communication [11].

As a part of the system security cryptographic hashes are used. All transferred data is signed and so the integrity is guaranteed. This process also authenticates the sender of the message.

In order to use it each instrument from the equipment should be introduced using its AMP identification code. After the four digit identification pin is entered all signatures are checked and data is stored in the system database.

For additional security Blockchain technology with Etherium API [9] is used.

For all the animals under investigation metadata is entered into the system. Milk quality parameters include Fat, Protein and the main milk quality indicator-somatic cells count [13].

The system has a built-in inference engine that uses parameters such as breed, herd, the barn that the animal lives in, full history of previous treatments and previous measurements when it is deducting treatments or calibrating the results. performance. Certainly other important variables that are not included in the study will contribute to an individual's work performance.

3. CONFIGURATION OF SPECIALIZED MODULE FOR WIRELESS CONNECTION

One of the most frequently used embedded computing system with WiFi communication module as a heart of IoT devices is the WiFi module ESP8266 [7]. This is due to its affordable price and its pre-loaded firmware that supports communication and configuration via the serial interface using AT commands. This makes it very easy and affordable to integrate the module with any existing hardware. It is built with 32 bit WiFi micro-controller with serial interface and two GPIO ports, additional flash memory and built in antenna. Its low power consumption makes it also suitable for integration with battery powered devices.

On Figure 1 and Figure 2 are presented module characteristics and its wiring diagram.

Figure 1 Calibrated and Uncalibrated Measurement

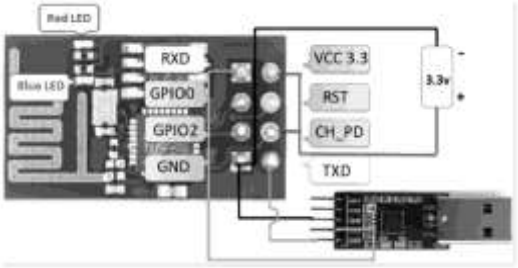

Figure 2 Calibrated and Uncalibrated Measurement

Unfortunately the default loaded firmware does not support HTTPS protocol out of the box, but just the not encrypted HTTP protocol for communication with WEB services, so extra measures has to be taken for securing the communication. This is done in the developed higher level protocol described in this paper.

Before we can form an actual HTTP requests to the cloud WEB API we will need first to configure the module for preconfigured available wireless network in range, with possible Internet access.

After the module is properly connected to power and the serial communication is set up either via USB-to-Serial convertor for computer for development and debugging purposes, or directly connected to the UART of the main micro controller in production environment, it will be ready to accept AT commands. At this point we will be able to send the commands for initialisation and for joining the mould in the wireless network.

In order to join the module to wireless network, it should be configured in client mode using the first option (1) from the available modes in the default firmware. We need then to execute commands for scanning for available wireless access points and then joining the selected wireless network. These settings are stored on the module flash memory, so we just need to do them once, and they will persist between module reboots and power cycles.

Terminal 1.

```
AT+CWMODE?
+CWMODE:1
```

Terminal 2.

```
AT+CWMODE=1
OK
```

On Terminal 1. is shown querying the current operational mode and on Terninal 2. we can see how to set the module in wireless client mode (1).

Terminal 3.

```
AT+RST
OK

ets Jan 8 2013,rst cause:4, boot mode:(3,7)

wdt reset
load 0x40100000, len 24444, room 16
tail 12
chksum 0xe0
ho 0 tail 12 room 4
load 0x3ffe8000, len 3168, room 12
tail 4
chksum 0x93
load 0x3ffe8c60, len 4956, room 4
tail 8
chksum 0xbd
csum 0xbd

ready
```

Next reset command is issued in order to activate the selected mode (Terminal 3.).

After this procedure we should send commands for scanning for available WiFi networks (Terminal 4.) and joining to fifi-wifi3 network (Terminal 5.). For this we need to pass the network passphrase as a second parameter.

Terminal 4.

```
AT+CWLAP
+CWLAP:(0,"",0)
+CWLAP:(3,"alarkov",-79)
+CWLAP:(3,"alarkov-guest",-79)
+CWLAP:(3,"fifi-wifi3",-62)
+CWLAP:(3,"M-Tel_1F5D",−84)
+CWLAP:(3,"fifi−wifi−2",−78)
+CWLAP:(3,"universal",−92)
```

Terminal 5.

```
AT+CWJAP="fifi-wifi3","fifi-net"
OK
```

In order to check the currently joined network we can execute query on Terminal 6.

The network settings are persisted even after power cycle. So the only time we need to execute this procedure is initially, or if we want to connect to different wireless network.

Once we are already connected to a network, and hopefully to Internet, we can check the IP address assigned to the module via the DHCP protocol from the wireless access point (Terminal 7.).

Terminal 6.

```
AT+CWJAP?
+CWJAP:"fifi-wifi3"
OK
```

Terminal 7.

```
AT+CIFSR
10.15.7.149
```

4. HTTP REQUEST SENDING

At this point we should have IP connectivity to the WiFi network and to the Internet so we can start sending HTTP requests. For the purposes of this research we will be sending POST HTTP requests to a test server which URL is http://dev.celtis.net/demo/dump.php. The server will just take the post parameters and print them out in the response so we can easily debug the configuration so far.

We will activate the multiple connection option. By default the module is in single connection option, but the multiple connections are recommended in the documentation, so we activate it by issuing the following command (Terminal 8.).

Terminal 8.

```
AT+CIPMUX=1
OK
```

In order to initiate a new TCP connection to dev.celtis.net on port 80 we should issue the following command. We will use 4 as the identifier for the connection (Terminal 9.).

Terminal 9.

```
AT+CIPSTART=4,"TCP","dev.celtis.net",80
OK
Linked
```

At this point we can issue the send command by telling it how many bytes we want to transfer. We use again 4 as the connection identifier (Terminal 10.).

Terminal 10.

```
AT+CIPSEND=4,190
>
```

We proceed to send the 190 bytes which represent request to the web server listening on the address, using the HTTP protocol. Every row of the request ends with \r\ n with one extra \r\ n row at the end that tells the web server that this is the end of the request (Terminal 11.).

Terminal 11.

```
POST /gas/dump.php HTTP/1.0 \r \n
Host: dev.celtis.net \r \n
User-Agent: EkoMilkHorizon/1.0 \r \n
Content-Type: application/x-www-form-urlencoded \r \n
Content-Length: 32 \r \n
\r \n
home=Cosby&favorite+flavor=flies \r \n
\r \n
SEND OK
```

On successful request, we will get a response from the server. If the request is correctly formatted, and server is operational we should get 200 OK status code. Right after that the server will return the headers and the body of the response according to the HTTP protocol. In this example the server is programmed to take 2 parameters in the request and just give them back in the response body (Terminal 12.).

Terminal 12.

```
+IPD,4,283:HTTP/1.1 200 OK
Date: Wed, 03 May 2017 15:14:48 GMT
Server: Apache/2.2.15 (CentOS)
X-Powered-By: PHP/5.4.31
Content-Length: 90
Connection: close
Content-Type: text/html; charset=UTF-8

array(2){
["home"] => string(5)"Cosby"
["favorite_flavor"] =>
string(5)"flies"
}

OK

OK
Unlink
```

Once the test run of the module is complete and we have successful communication with the WEB server using the HTTP protocol, we should specify a higher level protocol that guarantees the data integrity and uniquely identify the sender. These functions are not available in the HTTP protocol itself, so we are going to implement them using libraries and functionalities that are available in the microcontroller.

5. DESCRIPTION OF AMP 1.0 COMMUNICATION PROTOCOL

Since cost efficient micro controllers that are commonly used such as the one that we are using - ESP8266 does not have much compute power and RAM means that the industry standard RSA encryption schemes are not easily available. Even if we manage to fit the algorithm to use the system resources, the message signing will take too much time and energy for every single message. And the energy in cases of battery powered devices is very valuable resource. The option suggested below, for solving such problems will be to create a higher level protocol, over the HTTP protocol that the server and the terminal device both understand.

The main purpose of the suggested protocol is to confirm the identity of the sender as well as the integrity of the data, since the HTTP protocol is vulnerable to man in the middle attacks. To achieve this we use two hashing functions - the faster MD5 for signing the message because of its size, and the more reliable SHA1 for creating the identity token, which the server uses to verify the identity of the terminal device and determine the access rules for that device. The terminal device calculates the token for every message sent to the server and attaches its value as HTTP header. This way the token is different for every single HTTP request.

The messages that are transferred between the terminal devices and the cloud system in our example scenario are shown on Terminal 13.

Terminal 13.

```
No: 0
ID:123
2017-02-21 12:16:02
MM:N
FU:A
TN:E
DO:17.000
MI:0.67
TE:20.0
TM:20.4
TS:20.3
tc:N
LT: 4.02
SC:0
```

To begin the calculations, we need to calculate the MD5 hash of the message above. In the example the message is 127 bytes.

md5(message) = afd24349217e0bc183b220e87030024a

Once we have the hash, the next thing is to calculate the identity token using the following algorithm.

sha1(md5(message) + salt + serial + pin

Salt: is chosen as a random secret key. It is known only to the developers and it is hardcoded in every terminal device firmware and the server software. It is used to randomise generated hashes in a way that brute force attacks that aim finding the PIN code when the input message format is known much harder. For this example the sting "salt-salt" is used.

Serial: this is the serial number of the terminal device. It is unique number assigned to each device when it is manufactured, and this number is registered in the cloud system database. For the purposes of the example 162534 is used.

PIN: This is a code that the cloud system generates in the process of device registration, and the terminal device user will need to input in the device itself to complete the registration. It can be changed if needed from the cloud system administrative panel. In this example 1234 is used.

sha1(afd24349217e0bc183b220e87030024asalt-salt1625341234)=
153d1c279ec7e3772203f8a8b5dc5f02c77b525e}

Once we have the hash calculated we need to send its value for the X-AMP-Token header with every request to the server.

Since the token depends on the serial number, the pin and the message itself, if attacker tries to alter the message or part of the message, hoping to trick the system to input invalid data in its database, the token will get invalid. The system will easily detect that and reject all data attached to that token. The protocol on the client device should be implemented in such a way that if the server response is different than 200 OK, or the response does not come at all after a given timeout, because of communication problems, it should know that the data is rejected/not received form the server. If the token is not valid, the server will generate 403 Forbidden response code.

Since we need the serial number in clear text, because it is not reversible from the message digest algorithms, we need to send its value as X-AMP-SN header. On Terminal 14. is shown a sample request from the micro controller to the server that is compiled, using the data from above examples.

Terminal 14.

```
AT+CIPMUX=1
AT+CIPSTART=4,"TCP","cowmonitoring.com",80
AT+CIPSEND=4,50
POST /api/v1/saveInstrumentMeasurements HTTP/1.0
AT+CIPSEND=4,32
User-Agent: EkoMilkHorizon/1.0 // Device model for web client
AT+CIPSEND=4,41
Content-Type: text/plain; charset=utf-8
AT+CIPSEND=4,55
// identification token
X-AMP-Token: 153d1c279ec7e3772203f8a8b5dc5f02c77b525e
AT+CIPSEND=4,18
X-AMP-SN: 162534 // device serial number
AT+CIPSEND=4,21
Content-Length: 127
AT+CIPSEND=4,131
*** Message content ***
```

5.1. Server Side

The server API has to accept the HTTP request, read the two special headers that we specified, and the request body, and validates the request. This happens by applying the same algorithm to the data received to recalculate the signature.

Then the newly calculated signature is compared with the signature that the client calculated. If the signature matches, it will mean that the request is not modified in any way. The terminal device is the one that it claims to be by the sent Serial number, the PIN code for that Serial matches the one that we have registered in the database and the message payload is not modified in any way.

On Figure 3. is shown token generation.

```
public function authenticate($user, $request, $response)
{
    $serial = $request->getHeaders()->get('X-AMP-SN');
    $token = $request->getHeaders()->get('X-AMP-Token');
    $md5Body = md5($request->getRawBody());
    if ($serial)
    {
        if ($token)
        {
            $compositeToken = implode(",", [$serial, $token, $md5Body]);
            $identity = $user->loginByAccessToken($compositeToken);
            if ($identity)
            {
                return $identity;
            }
            else
            {
                $response->headers->add("X-AMP-ERROR", "Wrong X-AMP-Token");
            }
        }
        else
        {
            $response->headers->add("X-AMP-ERROR", "X-AMP-Token header is missing")
        }
    }
    else
    {
        $response->headers->add("X-AMP-ERROR", "X-AMP-SN header is missing");
    }
            return null;
}
```

Figure 3 Token generation

We read the serial and the token headers in $serial and $token variables, calculate the MD5 sum of the body in $md5Body variable and create a composite token that will be used in the loginByAccessToken function to identify the sender terminal device in the cloud database.

On Figure 4. is shown the code that calculates and verifies the signature. It tries to find the device in the database by the serial number sent. If it finds one it uses the calculated request body hash, the pre-shared key, the serial and the pin from the database to re-calculate the checksum, which it then compares with the one calculated by the terminal device, and accept the request if it matches.

```
public static function findIdentityByAccessToken($token, $type = null)
{
    list($serial, $checksum, $md5Body) = explode(",", $token);
    $identity = static::findOne(['serial_number' => $serial]);

    if($identity)
    {
        $calcChecksum = sha1($md5Body . self::PRE_SHARED_INSTRUMENT_KEY . $serial . $identity->pin);
        if($calcChecksum == $checksum)
        {
            return $identity;
        }
    }

    return null;
}
```

Figure 4 Server authentication

6. CONCLUSION

We present functionalities of a cloud based system for collection and expert analysis of specialized data. Also an integration of dedicated hardware solution to the system is described. Using the AMP 1.0 protocol over the HTTP protocol allows us on one hand to use cost effective hardware communication module with its stock firmware, which is well documented and frequently used while guaranteeing secure communication between the terminal devices and the cloud server. This approach can also work with other cost effective hardware in energy efficient mode.

REFERENCES

[1] Aggarwal, C. Neural Networks and Deep Learning, Springer, 2018.

[2] Dagnino, A., Allen, J., Moore, M. Development of an expert system for the integration of biomarker responses in mussels into an animal health index. *Biomarkers*, vol. 12(2), 2007, pp. 155-72.

[3] Greengard, S. The Internet of Things. MIT Press, 2015.

[4] Hameed, K., Sender, G., Kossakowska, A. Public health hazard due to mastitis in dairy cows. *Animal Science Papers and Reports*, vol. 25(2), pp. 73-85.

[5] Jampour, M. A Fuzzy Expert System to Diagnose Diseases with Neurological Signs in Domestic Animal. *In proceedings of Eighth International Conference on Information Technology: New Generations (ITNG)*, 2011.

[6] Kranz, M. Building the Internet of Things. Wiley, 2016.

[7] Schwartz, M. Internet of Things with ESP8266. Packt Publishing, 2016.

[8] Todorov, T., Stoinov, J. Expert system for milk and animal monitoring. *International Journal of Advanced Computer Science and Applications*, vol. 10(6), pp. 25-30, 2019.

[9] Contract APIs: https://etherscan.io/apis#contracts

[10] Ekomilk Horizon: https://www.bulteh.com/ekomilk-horizon-unlimited-hybrid-analyzer.html

[11] HTTP specification: https://tools.ietf.org/html/rfc1945

[12] MOD-WIDI-ESP8266: https://www.olimex.com/Products/IoT/ESP8266/MOD-WIFI-ESP8266/open-source-hardware

[13] Somatic Cell Count: https://dairy.ahdb.org.uk/technical-information/animal-health-welfare/mastitis/symptoms-of-mastitis/somatic-cell-count-milk-quality-indicator/#.XHJ-zS2B1QI