IJIREEICE

International Journal of Innovative Research in
Electrical, Electronics, Instrumentation and Control Engineering

ISO 3297:2007 Certified

Vol. 4, Issue 8, August 2016

ISSN (Online) 2321 – 2004
ISSN (Print) 2321 – 5526

Secure Data Transmission for IOT Applications

Radhika Munoli[1], Prof. Sankar Dasiga[2]

M.Tech Scholar, Department of ECE, NMIT, Bangalore, India[1]

Professor, Department of ECE, NMIT, Bangalore, India[2]

Abstract: Providing security for the IOT environment is the major assessment carried out where the entire world is dependent on e-communication and assures the guarantee of communication without any error causing .In this project work, Raspberry Pi, an open source and a popular choice as the hardware platform for IoT - both devices as well as gateways, has been used. It is nowadays a trend and also a more appropriate path to choose open source software for implementation for the prototyping and study purposes in academia. As such Open SSL has been employed for configuring secure access of data at the device level as well as the library for the secure communication using the MQTT and CoAP protocols. Further, the project work also involves a study of different web access vulnerabilities and suggested remedies. Even when the latest version 2 of Raspberry Pi was employed the performance of the application with Open SSL vs. a standard desktop computer system is not comparable. Further optimization of the application or use of a 128-bit key based encryption could be the possible approaches for security implementations for embedded applications. The primary objective of this project aims at implementing security procedures for IoT based devices such as nodes (for ex raspberry pi) and gateways (for ex PC) using MQTT and CoAP protocol in an embedded platform. Project approaches at different layers of the ISO/OSI model for the security of end to end nodes and gateways through cloud.

Keywords: Raspberry Pi, IOT, Open SSL, Secure, End to End Communication, MQTT, CoAP, Vulnerability.

I. INTRODUCTION

Internet of Things (IoT) has become an area of immense interest for the academia as well as the industry in the recent times. Anticipation is that by Y2020 there would be 50 billion Portable / Wearable, Consumer and Industrial etc. devices on the net. This presents significant opportunity as well as challenge to the researchers and engineers. While the amount of hardware and software that would be needed to interface & connect the things and collect & process the data from them offers many opportunities for innovation and development, the security requirements of innumerable devices and the Big Data poses multiple challenges that necessitate employment of robust measures and implementations. You do not want the doors of your car to be unlocked via the net by somebody when it is in the parking lot while you are busy shopping in the mall!! This project aims to look at some of the security considerations and the approach for implementation in the context of IoT.

While providing data security through MQTT and CoAP protocol, widely used with general purpose computer systems. Its use with embedded systems is not prevalent. IOT Application Security is a combination of Network Security, Data or information Security and Software/firmware Protection. Here, providing Network security and Data security are the point of my concern. Network security is the use of software, hardware, and procedural methods to protect IOT applications from attackers and Data security is the use of codes, algorithms and encryption techniques for the protection of IOT applications.

II. PROTOCOLS

Protocols used in this project for securing the data transmitted in either of the communication ways are:

- Node to gateway
- Gateway to node
- Node to node
- End to end

A. MQTT

MQTT (Message Queue Telemetry Transport)is an application protocol viewed as a publish subscribe model, designed for the communication of M2M .This protocol sits on top of TCP/IP layer . both client and broker need to have a TCP/IP protocol stack

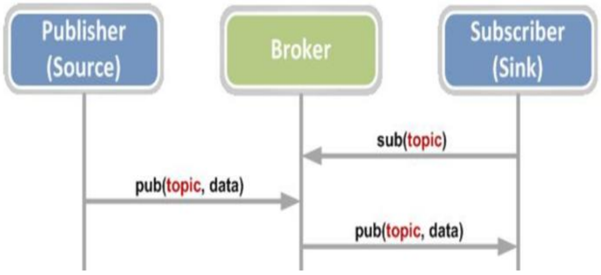


Fig 1: Publisher Subscriber model

The architecture of MQTT publisher subscriber model showed above features one central server (broker) that manages the subscriptions (sink) and publications (source) from each of its various clients. Clients can publish the

DOI 10.17148/IJIREEICE.2016.4804

data without knowing the subscriber in this model. MQTT is also designed to reduce the overhead for each packet coming through it in order to preserve bandwidth and performance for resources which are constrained in embedded devices. It's simple framework for the managed mesh networks of TCP oriented devices.

SSL/TLS - To implement security for the data transmission between the nodes (pi) and gateways (pc) in an IOT context. SSL/TLS (Secure Socket Layer/Transport Layer Security) is to be used for MQTT protocol. Since, MQTT relies only on TCP (connection oriented) as transport protocol, by default this connection does not use an encrypted communication. To encrypt the entire MQTT communication, it allows using TLS instead of plain TCP. This is carried out by TCP handshake.

B. CoAP

DTLS is used to protect the CoAP protocols. As CoAP (Constrained Application Protocol) is a web protocol which relies over UDP (User Datagram Protocol; which is connectionless) protocol used mainly for the constrained M2M devices in the IOT, TLS is not used here; instead encryption is done using DTLS (Datagram Transport Layer Security). Most of the constrained device (CoAP) implementations are carried by lib coap packages; this can also be used on the server side.

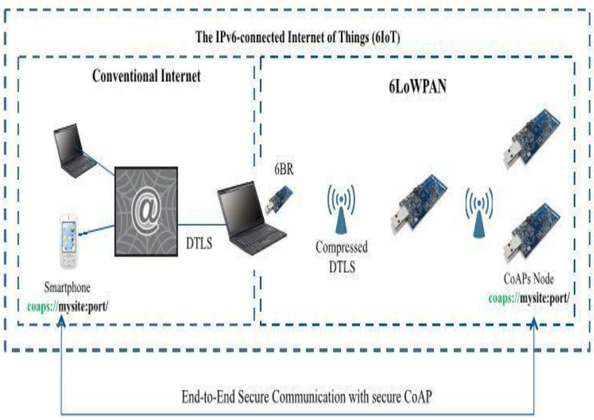


Fig 2: end to end secure communication with the CoAPs

The figure gives the detailed description about the end to end security in an CoAPs environment of IOT. It is divided into 2 parts are seen at the right hand side is the light weight protocol access network which are connected wirelessly and at the left conventional internet is been connected wired and thus CoAP provides security between both the ends making the use of secure CoAP or CoAPs.

DTLS – DTLS (Datagram TLS) is the only protocol providing channel security. Since it performs authentication, authorization key exchange, and provides protection against application data. Using this DTLS as the security suite for IoT applications; the security protection can be done using DTLS handshake

III.DESIGN APPROACH

The security implement for MQTT protocol is done using open ssl library function of SSL/TLS encryption method and security implementation for CoAP protocol is done using asyncio function of DTLS encryption method at the node transmission of data coming through the gateway (cloud) as shown in the following figures:

A. Open SSL

Creating the structure of node to node communication such as raspberry pi's and PC's acting as gateways for the two nodes. In this scenario data to be transmitted from the gateway are secured at the node point using SSL/TLS cryptographic methods which includes handshake mechanisms to establish the connections i.e., raspberry pi to PC over open ssl.

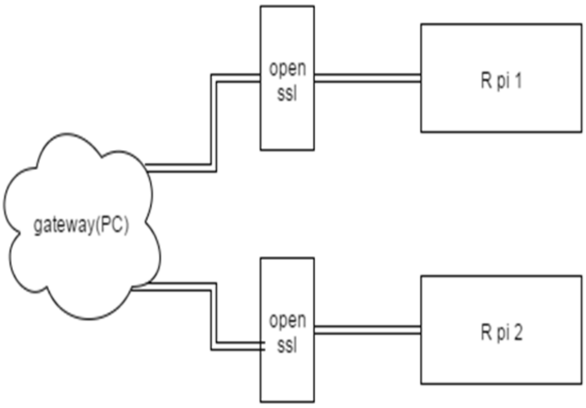


Fig 3: connection between node to node and gateway through open ssl

B. DTLS

In the below scenario the data transmission is been carried through the gateway securing it with datagram encryption method of CoAP and passing to the node point using the client and server DTLS encryption mechanism, data coming from the cloud sent by client are secured at the servers.

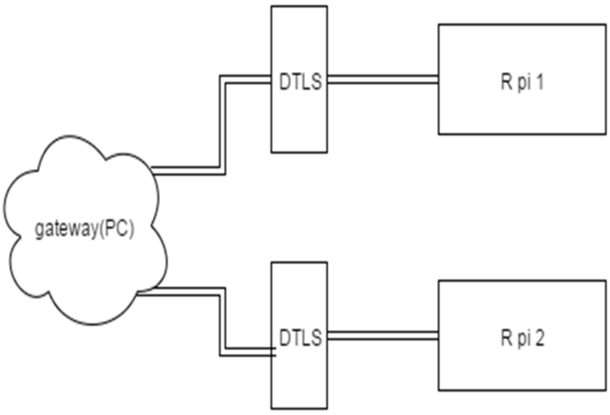


Fig 4: connection between node to node and gateway through DTLS

DOI 10.17148/IJIREEICE.2016.4804

C. End to end communication

Proposed Secure End to end connection communication system between nodes and gateway is shown in the figure: At right of the figure shown below the security of the data is maintained between the gateway and the node i.e.,the data coming from the cloud (gateway) is been secured to read it at the node point terminal

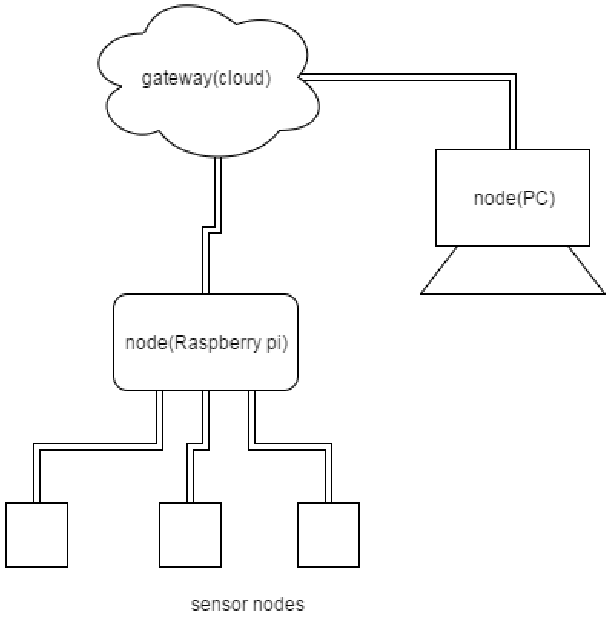


Fig 5: secure data transmission for end to end communication

Since the data /information to be transmitted is secured at the node through raspberry pi using the protocol MQTT with TLS encryption algorithm of openssl library and can also be done with the CoAP protocol using DTLS. This is possible by establishing the SSL/TLS handshake connection between the different websites. Here I have demonstrated with hp and google websites.

Now coming to the cloud i.e., gateway is secured at the MQTT over mbed TLS from the various vulnerabilities by patching it, by another node (ex: PC or laptop) using HTML.

IV. VULNERABILITY

A. Network Level Vulnerability

Some network level vulnerabilities are listed below:

Network level vulnerabilities	Description
SSL/TLS not enforced	The traffic sent is SSL/TLS encrypted over a network but can be accessed over unencrypted HTTP connection.
SSL/TLS Insecure Renegotiation	Want to access the new TLS handshake during an ongoing SSL/TLS handshake and its known as session renegotiation.
Weak SSL ciphers	If weak SSL cipher suites are configured it can decrypt and modify the traffic.
Open ssl implementation to Heart bleed	It attacks directly at the server's memory when the remote server is running.
SSL configuration vulnerable to POODLE	Padding oracle on downgraded legacy encryption is an attack which exploits a combination of downgraded cipher suites.
SSL/TLS BEAST	Allows the attacker to inject the JavaScript code into the browser to decrypt the HTTPS traffic.
SSL/TLS crime information	Compression ratio info leak made easy is an attack if plain text data is encrypted before compression.

Table 1: list of network level vulnerability

B. Application Level Vulnerability

OWASP top 10 vulnerabilities are listed below:

Injection
Broken Authentication and Session Management
Cross Site Scripting
Insecure Direct Object References
Security Misconfiguration
Sensitive Data Exposer
Missing Function Level Access Control
Cross Site Request Forgery
Using Components with Known Vulnerabilities
Invalidated Redirects and Forwards

Table 2: list of application level vulnerability

C. Tool used to detect Network level Vulnerability

Nmap ("Network Mapper") is a free tool available to download and it's also an open source (license) for network discovery and security checking. It also gives the host address using in that ip address mapping its time browsed using the session cookie and session timeout method. Nmap basically introduces vulnerability detection mostly network level vulnerability and service detection features are available in it.

- Nmap make utilization of crude IP bundles to figure out what hosts are accessible on the system, what benefits those hosts are putting forth,
- What working frameworks (and OS forms) they are running, what kind of parcel channels/firewalls are being used, and many different attributes. The screenshots of this are shown in the results.

D. Tool used to detect application level vulnerability

Burp suit is the actual tool used to detect the application level vulnerability to verify this i have created a HTML page highlighting our college annual function as an

DOI 10.17148/IJIREEICE.2016.4804

example for showing the page which is vulnerable and the pages are vulnerable to 2 different commands of XSS (third vulnerability in top 10 OWASP) and they are:

- Bad attribute XSS command
- Bad script XSS command

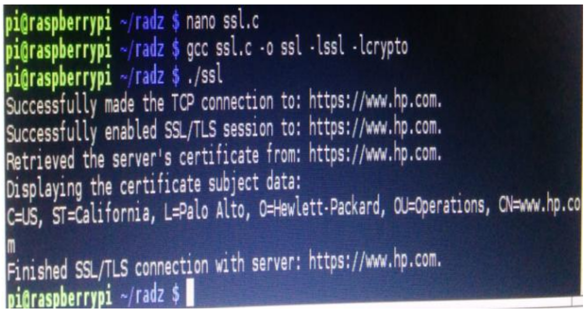
The screenshots of it are shown in the results.

V. RESULTS

The results are shown in the form of screenshots for the following parameters:

A. Open SSL

The screenshot shown below gives the execution outputs of the c code for performing open ssl security at the hp website and also for the google website. Allowing the SSL/TLS handshake for the connection establishment at the gateway and the node ensures the end to end security. The data to be transmitted now between the node to node or node to gateway or vice version is possible securely



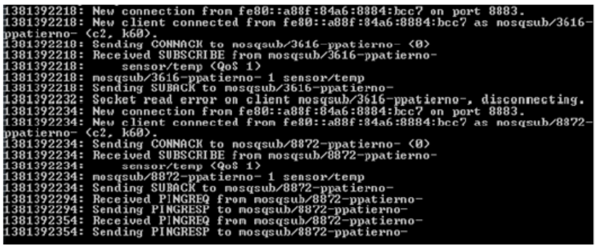
```
pi@raspberrypi ~/radz $ nano ssl.c
pi@raspberrypi ~/radz $ gcc ssl.c -o ssl -lssl -lcrypto
pi@raspberrypi ~/radz $ ./ssl
Successfully made the TCP connection to: https://www.hp.com.
Successfully enabled SSL/TLS session to: https://www.hp.com.
Retrieved the server's certificate from: https://www.hp.com.
Displaying the certificate subject data:
C=US, ST=California, L=Palo Alto, O=Hewlett-Packard, OU=Operations, CN=www.hp.co
m
Finished SSL/TLS connection with server: https://www.hp.com.
pi@raspberrypi ~/radz $
```

Screenshot 1: Output of connection establishment between client and server

B. MQTT and SSL

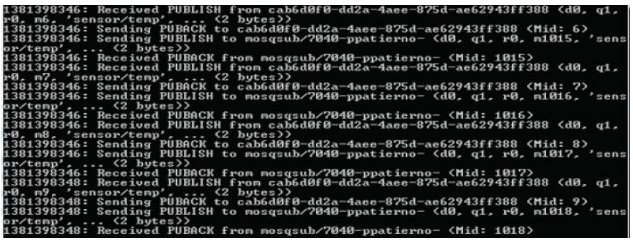
The following screenshots are the outputs of the MQTT publisher, subscriber and broker model. Where, the output of the console of the broker is shown first. Further, in the console window of the broker, screenshot of the messages received from the MQTT QoS is shown next.
A screenshot of the publisher messages received on console window of the subscriber is shown at the end.
The following screenshots shows the output of:

- Console of the broker
- Message received from MQTT QoS
- Subscriber



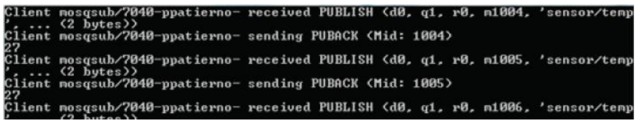
```
1381392218: New connection from fe80::a88f:84a6:8884:bcc7 on port 8883.
1381392218: New client connected from fe80::a88f:84a6:8884:bcc7 as mosqsub/3616-
ppatierno- (c2, k60).
1381392218: Sending CONNACK to mosqsub/3616-ppatierno- (0)
1381392218: Received SUBSCRIBE from mosqsub/3616-ppatierno-
1381392218:     sensor/temp (QoS 1)
1381392218: mosqsub/3616-ppatierno- 1 sensor/temp
1381392218: Sending SUBACK to mosqsub/3616-ppatierno-
1381392232: Socket read error on client mosqsub/3616-ppatierno-, disconnecting.
1381392234: New connection from fe80::a88f:84a6:8884:bcc7 on port 8883.
1381392234: New client connected from fe80::a88f:84a6:8884:bcc7 as mosqsub/8872-
ppatierno- (c2, k60).
1381392234: Sending CONNACK to mosqsub/8872-ppatierno- (0)
1381392234: Received SUBSCRIBE from mosqsub/8872-ppatierno-
1381392234:     sensor/temp (QoS 1)
1381392234: mosqsub/8872-ppatierno- 1 sensor/temp
1381392234: Sending SUBACK to mosqsub/8872-ppatierno-
1381392294: Received PINGREQ from mosqsub/8872-ppatierno-
1381392294: Sending PINGRESP to mosqsub/8872-ppatierno-
1381392354: Received PINGREQ from mosqsub/8872-ppatierno-
1381392354: Sending PINGRESP to mosqsub/8872-ppatierno-
```

Screenshot 2:output of console of the broker



```
1381398346: Received PUBLISH from cab6d0f0-dd2a-4aee-875d-ae62943ff388 (d0, q1,
r0, m6, 'sensor/temp', ... (2 bytes))
1381398346: Sending PUBACK to cab6d0f0-dd2a-4aee-875d-ae62943ff388 (Mid: 6)
1381398346: Sending PUBLISH to mosqsub/7040-ppatierno- (d0, q1, r0, m1015, 'sens
or/temp', ... (2 bytes))
1381398346: Received PUBACK from mosqsub/7040-ppatierno- (Mid: 1015)
1381398346: Received PUBLISH from cab6d0f0-dd2a-4aee-875d-ae62943ff388 (d0, q1,
r0, m7, 'sensor/temp', ... (2 bytes))
1381398346: Sending PUBACK to cab6d0f0-dd2a-4aee-875d-ae62943ff388 (Mid: 7)
1381398346: Sending PUBLISH to mosqsub/7040-ppatierno- (d0, q1, r0, m1016, 'sens
or/temp', ... (2 bytes))
1381398346: Received PUBACK from mosqsub/7040-ppatierno- (Mid: 1016)
1381398346: Received PUBLISH from cab6d0f0-dd2a-4aee-875d-ae62943ff388 (d0, q1,
r0, m8, 'sensor/temp', ... (2 bytes))
1381398346: Sending PUBACK to cab6d0f0-dd2a-4aee-875d-ae62943ff388 (Mid: 8)
1381398346: Sending PUBLISH to mosqsub/7040-ppatierno- (d0, q1, r0, m1017, 'sens
or/temp', ... (2 bytes))
1381398346: Received PUBACK from mosqsub/7040-ppatierno- (Mid: 1017)
1381398348: Received PUBLISH from cab6d0f0-dd2a-4aee-875d-ae62943ff388 (d0, q1,
r0, m9, 'sensor/temp', ... (2 bytes))
1381398348: Sending PUBACK to cab6d0f0-dd2a-4aee-875d-ae62943ff388 (Mid: 9)
1381398348: Sending PUBLISH to mosqsub/7040-ppatierno- (d0, q1, r0, m1018, 'sens
or/temp', ... (2 bytes))
1381398348: Received PUBACK from mosqsub/7040-ppatierno- (Mid: 1018)
```

Screenshot 3: output of message received from MQTT QoS



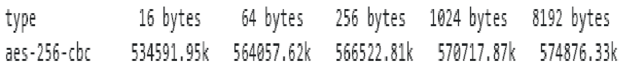
```
Client mosqsub/7040-ppatierno- received PUBLISH (d0, q1, r0, m1004, 'sensor/temp
', ... (2 bytes))
Client mosqsub/7040-ppatierno- sending PUBACK (Mid: 1004)
27
Client mosqsub/7040-ppatierno- received PUBLISH (d0, q1, r0, m1005, 'sensor/temp
', ... (2 bytes))
Client mosqsub/7040-ppatierno- sending PUBACK (Mid: 1005)
27
Client mosqsub/7040-ppatierno- received PUBLISH (d0, q1, r0, m1006, 'sensor/temp
', ... (2 bytes))
```

Screenshot 4: output of subscriber

C. Performance of SSL

Comparison of desktop and pi - Observation is that there is a factor of 33 differences in terms of performance between the computer system and the embedded platform. Part of the significantly high performance on the Intel processor based desktop system could be attributed due to the availability of hardware acceleration as well as floating point and math co-processors.

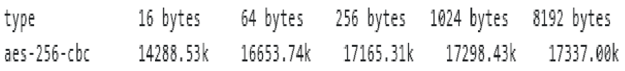
On the Desktop Computer System:



type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes-256-cbc	534591.95k	564057.62k	566522.81k	570717.87k	574876.33k

Screenshot 5: On the Desktop Computer System

On the raspberry pi platform:



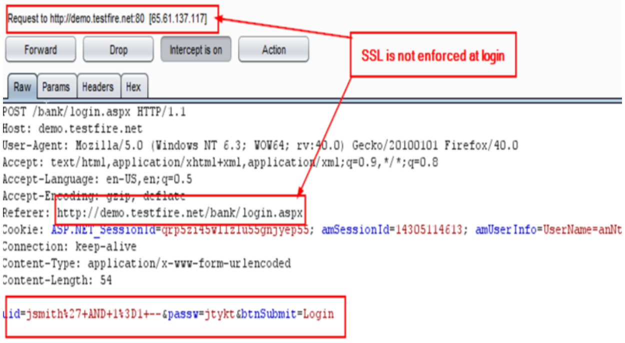
type	16 bytes	64 bytes	256 bytes	1024 bytes	8192 bytes
aes-256-cbc	14288.53k	16653.74k	17165.31k	17298.43k	17337.00k

Screenshot 6: On the raspberry pi platform

D. NMAP

In network level vulnerabilities, using NMAP the following vulnerabilities are detected:

- SSL is not enforced of login
- Poodle
- Beast

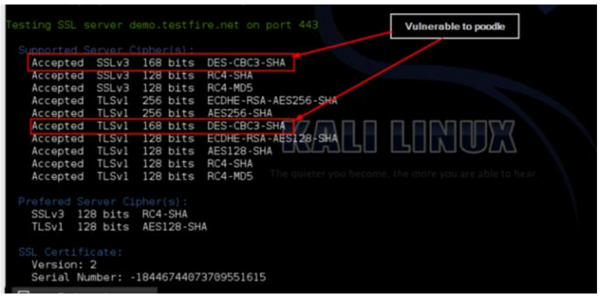


Request to http://demo.testfire.net:80 [65.61.137.117]
Forward
Drop
Intercept is on
Action
SSL is not enforced at login
Raw
Params
Headers
Hex

```
POST /bank/login.aspx HTTP/1.1
Host: demo.testfire.net
User-Agent: Mozilla/5.0 (Windows NT 6.3; WOW64; rv:40.0) Gecko/20100101 Firefox/40.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Referer: http://demo.testfire.net/bank/login.aspx
Cookie: ASP.NET_SessionId=qrp5z145w1lz1u55gnjyep55; amSessionId=14305114613; amUserInfo=UserName=anNt
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Content-Length: 54

uid=jsmith%27+AND+1%3D1+--&passw=jtykt&btnSubmit=Login
```

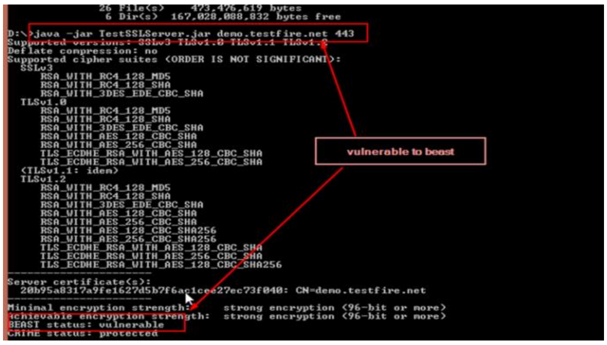
Screenshot 7: SSL not enforced at login

DOI 10.17148/IJIREEICE.2016.4804



```
Testing SSL server demo.testfire.net on port 443

  Supported Server Cipher(s):
    Accepted  SSLv3  168 bits  DES-CBC3-SHA
    Accepted  SSLv3  128 bits  RC4-SHA
    Accepted  SSLv3  128 bits  RC4-MD5
    Accepted  TLSv1  256 bits  ECDHE-RSA-AES256-SHA
    Accepted  TLSv1  256 bits  AES256-SHA
    Accepted  TLSv1  168 bits  DES-CBC3-SHA
    Accepted  TLSv1  128 bits  ECDHE-RSA-AES128-SHA
    Accepted  TLSv1  128 bits  AES128-SHA
    Accepted  TLSv1  128 bits  RC4-SHA
    Accepted  TLSv1  128 bits  RC4-MD5

  Prefered Server Cipher(s):
    SSLv3  128 bits  RC4-SHA
    TLSv1  128 bits  AES128-SHA

  SSL Certificate:
    Version: 2
    Serial Number: -18446744073709551615
```

Screenshot 8: vulnerable to poodle



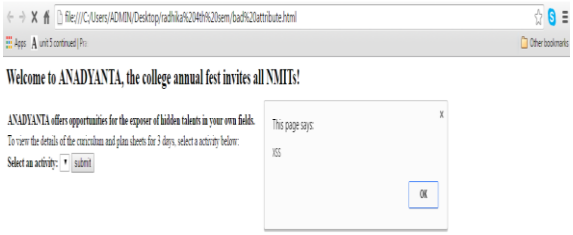
```
             26 File(s)    473,476,619 bytes
              6 Dir(s)  167,028,088,832 bytes free

D:\>java -jar TestSSLServer.jar demo.testfire.net 443
Supported versions: SSLv3 TLSv1.0 TLSv1.1 TLSv1.2
Deflate compression: no
Supported cipher suites (ORDER IS NOT SIGNIFICANT):
  SSLv3
     RSA_WITH_RC4_128_MD5
     RSA_WITH_RC4_128_SHA
     RSA_WITH_3DES_EDE_CBC_SHA
  TLSv1.0
     RSA_WITH_RC4_128_MD5
     RSA_WITH_RC4_128_SHA
     RSA_WITH_3DES_EDE_CBC_SHA
     RSA_WITH_AES_128_CBC_SHA
     RSA_WITH_AES_256_CBC_SHA
     TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
     TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
  (TLSv1.1: idem)
  TLSv1.2
     RSA_WITH_RC4_128_MD5
     RSA_WITH_RC4_128_SHA
     RSA_WITH_3DES_EDE_CBC_SHA
     RSA_WITH_AES_128_CBC_SHA
     RSA_WITH_AES_256_CBC_SHA
     RSA_WITH_AES_128_CBC_SHA256
     RSA_WITH_AES_256_CBC_SHA256
     TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA
     TLS_ECDHE_RSA_WITH_AES_256_CBC_SHA
     TLS_ECDHE_RSA_WITH_AES_128_CBC_SHA256
----------------------
Server certificate(s):
  20b95a8317a9fe1627d5b7f6ac1cee27ec73f040: CN=demo.testfire.net
----------------------
Minimal encryption strength:     strong encryption (96-bit or more)
Achievable encryption strength:  strong encryption (96-bit or more)
BEAST status: vulnerable
CRIME status: protected
```

Screenshot 9: vulnerable to beast

E. HTML

The following screenshots shown below are:
HTML page highlighting our college annual function as an example for showing the page which is vulnerable and the pages are vulnerable to 2 different commands of XSS (third vulnerability in top 10 OWASP) and they are:

- Bad attribute XSS command
- Bad script XSS command



file:///C:/Users/ADMIN/Desktop/radhika%204th%20sem/bad%20attribute.html

Welcome to ANADYANTA, the college annual fest invites all NMITs!

ANADYANTA offers opportunities for the exposer of hidden talents in your own fields.

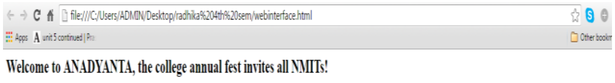
To view the details of the curriculum and plan sheets for 3 days, select a activity below:

Select an activity:

This page says:

XSS

Screenshot 10: vulnerable site shown with XSS bad attribute



file:///C:/Users/ADMIN/Desktop/radhika%204th%20sem/webinterface.html

Welcome to ANADYANTA, the college annual fest invites all NMITs!

Screenshot 11: vulnerable site shown with XSS bad script

VI.CONCLUSION

Security in the realm of IoT has to be at all levels – Device Access, M2M, Gateway to the Cloud, and Access of Data on the Cloud etc. Data access from the device via the local ports such as USB has to be secured through measures such as encryption, firewalls etc., security of the data on the net can be addresses at Network or the M2M Level through mechanisms like proprietary protocols, encryption etc. whereas at the Transport Level through securing protocols such as MQTT, CoAP or the Web Sockets – all of which are overheads for the limited-capable IoT devices and LANs. Security with regards access of data on the cloud in a way is a problem that falls in the space of highly-capable computer systems and internet resources.

In this project I have established the connection through handshake between the client and the server assuming nodes and the gateway as client and server , hence data arriving at the destination from the source either it may be gateway to node or node to gateway , in either cases it provides security between end to end scenario on embedded platform . This transmission of data security is completed with different protocols like MQTT and CoAP using its corresponding library functions libssl and libcoap respectively. Different vulnerabilities are mentioned in the project like network layer and application layer vulnerable. Performance of SSL is carried out by Comparing the desktop computer system and raspberry pi platform observing the factor of 33 differences in terms of performance between the computer system and the embedded platform. To conclude, I have created a webpage of my college (as an example) with 1 application vulnerable HTML site and preventing its consequences for the same. On remediating these vulnerabilities one can prevent and protect from attackers to attack.

REFERENCES

[1] "Secure Communication for Smart IoT Objects: Protocol Stacks, Use Cases and Practical Examples"by-Riccardo Bonetto, Nicola Bui, VishwasLakkundi, Alexis Olivereau, AlexandruSerbanati, Michele Rossi

[2] "Standards based end-to-end IP security for the internet of things" by-rennehummen, klauswehrle

[3] "A Study of Information Security for M2M of IOT" By-DU Jiang and CHAO ShiWei."

[4] "Foundations of security"by-Neil Daswani, Christoph kern, Anita kesavan

[5] "Cryptography and Network Security" by-William Stallings

[6] "Secure CoAP Using Enhanced DTLS forInternet of Things" By-AjitA.Chavan, Mininath K. Nighot

[7] http://www.computer.org/csdl/proceedings/wowmom/2012/1238/00/WSIoTSoSSecureCommunicationforSmartIoTObjectsProto.pdf

[8] http://www.computer.org/csdl/proceedings/cisim/2008/3184/00/3184a157.pdf

[9] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=1589113&queryText=security%20to%20wireless%20data%20transmission%20under%20IOT%20env&pageNumber=4&http://ieeexplore.ieee.org/xpl/articleDetails.jsp?arnumber=7270408&queryText=security%20to%20wireless%20data%20transmission%20under%20IOT%20env&pageNumber=10&newsearch=true&searchField=Search_All newsearch=true&searchField=Search_All

[10] http://ieeexplore.ieee.org/xpl/articleDetails.jsp?reload=true&arnumber=7103240

[11] http://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl

[12] http://www.raspberrypi.org

[13] http://www.elinux.org

[14] http://www.hivemq.com/blog/mqtt-security-fundamentals-tls-ssl

[15] http://www.cisco.com/c/en/us/about/security-center/secure-iot-proposed-framework.html