Flashy Lights and Security

Design the Network with Security in Mind

- Separate Networks, keeping the bad guys out, and the IoT network honest
- Set up Firewall rules (on the router) to enforce the following:
  - The **DMZ network** can be accessed from the internet
  - The **IoT Network** can **ONLY** connect to the server in the **DMZ network**!
  - The **Trusted People Network** can go anywhere
  - The **Guest Network**, you have friends, right?
- Don't forget IPv6 and firewall rules

**HTTP vs HTTPS**

- HTTP all data is transferred in plain text
- HTTPS all data is encrypted using public private key (PPK) encryption

- For our application - changing flashing lights there is no need to encrypt the data.
  - http://leds/program?s=6 is **astonishingly** uninteresting but what is interesting is
  - the user id and password. These we want to keep secret. Both in this case and - wouldn't want the NSA messing with our lights :(
  - More importantly we wouldn't want the black hat hackers messing with our
So we want to ensure that only authorized folks can get in. Ways to do this:

1. ID, Password
2. PPK and passphrase where only authorized folks hold the private key and passphrase.
3. One time pad
4. ...
Node-Red on the Pi

- Installation of node red

Connecting to the Node-RED server on your Pi:

- https://<ip address or hostname>:1880
- Description of flows
https://10.10.45.???
or
https://[fd11::45]

Choose Colour

var lightProgram = global.get('lightp') || 0;

if (lightProgram == 0 ){
    msg.background = global.get('onColour');
} else{
    msg.background = global.get('offColour');
}
return msg;

http request
Creating a Secure Certificate with Let'sEncrypt and certbot

- Follow tutorial to install 'certbot' for LetsEncrypt on the Pi
  - Do not need to compile binaris, as Pi 3 is an ARM7 system
    Added keys for debian jessie-backports
    ```bash
    > sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 8B48AD6246925553
    > sudo apt-key adv --keyserver keyserver.ubuntu.com --recv-keys 7638D0442B90D010
    ```
  - Then do
    ```bash
    apt-get install certbot
    ```
- Run Certbot on the Pi
  - Note: Pi must be running a webserver, and it must be accessible to the outside world (adjust our firewall as needed)
    ```bash
    sudo certbot certonly --webroot -w /var/www/html --email admin@domain.net --rsa-key-size 4096 -d myhost.domain.net
    ```
    Saving debug log to
    /var/log/letsencrypt/letsencrypt.log
    Obtaining a new certificate
    Performing the following challenges:
    http-01 challenge for
    6rasport.cvmiller.net
    Using the webroot path /var/www/html for all unmatched domains.
    Waiting for verification...
    Cleaning up challenges
    Generating key (4096 bits):
    /etc/letsencrypt/keys/0000_key-certbot.pem
    Creating CSR:
    /etc/letsencrypt/csr/0000_csr-
certbot.pem
Cert is stored in
/etc/letsencrypt/live/myhost.domain.net

- then copy the private and fullchain certs (pem files) to /home/pi/.node-red directory
  - `sudo cp /etc/letsencrypt/live/6rasport.cvmiller.net/privkey.pem /home/pi/.node-red/privatekey.pem`
  - `sudo cp /etc/letsencrypt/live/6rasport.cvmiller.net/fullchain.pem /home/pi/.node-red/certificate.pem`

- Edit node-red settings.js file
  - Uncomment and add the names of the certificate file you created a while ago.
    - `// Add Certs
      131  https: {
      132    key: fs.readFileSync('/home/pi/.node-red/privatekey.pem'),
      133    cert: fs.readFileSync('/home/pi/.node-red/certificate.pem')
      134 },`

- Restart node-red

- Creating and installing a home brewed secure certificate
  - cd to your home directory then cd to .node-red and execute the following commands to create a self signed certificate
    - `openssl genrsa -out privatekey.pem 1024`
    - `openssl req -new -key privatekey.pem -out private-csr.pem`
    - `openssl x509 -req -days 3650 -in private-csr.pem -signkey privatekey.pem -out certificate.pem`

- Modify the node-red config file settings.js
- Uncomment the line: `var fs = require("fs");`
- Uncomment the lines and add credentials (see below):
  - `// Deid uncommented
    107    adminAuth: {
    108      type: "credentials",
    109      users: [{

Uncomment and add credentials:
• // Deid modified uncommented and insert creds.
20190909
110  username: "deid",
111  password:
"$2a$08$etA.ES98hxaVl9zYQN8xX.uAlQT/rRuARzdokGr7SmuGCog77jYPm",
112  permissions: "*"
113  }
114 },

Uncomment and add the names of the certificate file you created a while ago.
• // Deid uncommented.
131  https: {
132   key: fs.readFileSync('/home/pi/.node-red/privatekey.pem'),
133   cert: fs.readFileSync('/home/pi/.node-red/certificate.pem')
134 },

• Creating Credentials
  • The credentials are a bcrypt hash. To generate a suitable password hash, you can use the node-red-admin command-line tool:
    • node-red-admin hash-pw
    • It will query for the password and return the bcrypt hash.

Wireshark

Looking at the data transmission
• https
From /etc/services:

- vsat-control 1880/tcp  # Gilat VSAT Control
- vsat-control 1880/udp  # Gilat VSAT Control

We are NOT doing using port 1880 for vsat control. It turns out that 1880 is also the default port for Node-red.

What's interesting here, is that nothing is interesting.

IPV6

- Although IPv6 may seem scary, it isn't. One may assume that it is Just Like IPv4, but it isn't. IPv6 is a different protocol that still uses TCP and UDP for transport. IPv6 has its own firewall rules, and routing protocols. That said, one can learn the basics of IPv6 very quickly if they understand how networks work (with IPv4).

Why use IPv6?
- **Because it is easier.** There is no NAT (Net Address Translations) in the way, and correspondingly, no port forwards to set up. Your application is simpler. IPv6 eliminates the evils of NAT
  
  o **Broken connectivity** - The many to one nature of NAT breaks any to any connectivity originally conceived by the founders of the internet
  
  o **Simplifies Communications** - Other traversal protocols like STUN (Session Traversal Utilities for NAT), TURN (Traversal Using Relays around NAT), IGDP (UPnP Internet Gateway Device Protocol), and ALGs (Application-level gateway) are no longer needed
  
  o **Easier Troubleshooting** - Since the client address is the real address, and one doesn't have to look through one or more NAT devices to see the mapping of addresses just to troubleshoot a path

- **Because you will be future-ready.** The world has run out of IPv4 addresses (a couple of years ago), but the internet continues to grow. It can only grow using a larger address space.

- **It is already deployed on every PC, Mac, and smart phone.** In fact, there are wireless providers such as T-Mobile who are running IPv6-only networks with over 20 million users on their network. According to Google Stats, over 20% of internet traffic today is over IPv6.

**Using host names rather than IP addresses**

- Utilizing IPv6 should be transparent to the user. Rather than using a literal IPv4 address, such as 10.10.45.10, or an IPv6 address such as FD11::45, a better practice is to use a host name. Host names utilize the abstraction layer of DNS. If you don't have control of DNS, then put the host name into /etc/hosts
  o mypi fd11::45

- Once a host name has been setup, then the name can be used in the browser URL like any other website:
  o https://mypi:1880/

- Using a host name, also gets around the difficult issue of colons. In the IPv4 world, colons usually represent a specific TCP (or UDP) port, e.g.
  o 10.10.45.10:1880

- But IPv6 uses colons to demark the Quibbles of the IPv6 address. So a rather cludgy way was invented to indicate the port number with an ipv6 address:
  o [fd11::45]:1880

- As you can see it is much easier to use host names:
  o mypi:1880
When sniffing an IPv6 transaction using wireshark, it is often easy to use the following capture filter which will filter out all that IPv4 cruft:

- `ip6`

As you can see, the capture looks very similar to the IPv4 wireshark capture earlier, but with IPv6 addresses.

What's interesting here, is that nothing is interesting. IPv6 should be transparent to the user.

**Node-RED and IPV6**

- Modification to `../node-red/settings.js` to listen on IPv6.

```javascript
// By default, the Node-RED UI accepts connections on all IPv4 interfaces.
26 // Deid added IPv6 stuff.
27 // To listen on all IPv6 addresses, set uiHost to "::",
28 // The following property can be used to listen on a specific interface. For
29 // example, the following would only allow connections from the local machine.
31 //uiHost: "127.0.0.1",
```
Notes:

- Certificates are by domain-name only
  - By hijacking DNS, we use local addresses, rather than the real (internet) addresses
- Certbot **Requires** Internet connectivity
  - It creates **Challenges**, and places them in the Document-Root of the webserver in order to prove to LetsEncrypt that the Cert is being generated for the correct domain-name