Secure Message Broker for IOT applications combined with secure data injection from device side.

Vishrut Desai
Department of Electrical Engineering
vdesai6@ncsu.edu

Sweta Subhra Dutta
Department of Electrical Engineering
ssdatta@ncsu.edu

1 ABSTRACT

Message brokers are an important aspect of IoT and data security. Data transfer in IoT applications is majorly governed by the MQTT protocol today. Security issues in the IoT space, specially for industrial applications and in process industry have been vulnerable. Attacks on them have caused severe damages to the organisation as well people.There have been studies on how adversaries can sniff data patterns and hamper the privacy of the operation. So we propose to have data injection to confuse the attacker as well as data validation and sensor node validations. The injection is proposed to be done in random intervals. The targeted systems are soft real time systems since data injection it itself yields latency in transmission. We also talk about the ways the data injection can be done and how can an adversary sniff the data to get the injected and the real data. Data validation is done using the python pydantic library where in we can have custom validations as well as it provides us a few standard validations.

2 PROBLEM DESCRIPTION

Major IoT sensors and data collecting/monitoring sensor in today's world operate on Real Time Operating Systems. There are some inbuilt venerability's in the RTOS as mentioned in [**?**]. The current industry standard of MQTT is definitely not a benchmark towards providing the highest level of security for messages. Further there are lapses in the security of the RTOS and weak points of this has been identified as Malicious Code Injection that has inefficient validation checks leaving weakness points in hard coded validations.

Over this issues, we plan to address them by building a message passing protocol for a micro controller unit to clone a real world case where a IoT device is running over a micro controller. This would be implemented as: • Adding a broker host /security host in between. For e.g Bare metal MCU like pico,STM connected to RPi which in turn is connected to a PC. This dual layer of host helps to add a dual security and we use it as a message broker.

3 SYSTEM MODEL

The system consists of an Rpi 4, model B. It is a Quad core 64-bit ARM-Cortex A72 running at 1.5GHz single board computer and serves as a second layer of host system in our implementation. The versatility of RPi and the extensive support documentation available makes as one of the best candidate for such prototype projects.

For a sensor implementation, we choose an pico board, also by raspberry Pi foundation. It is a low cost, tiny micro controller, based on the RP 2040 chip running a Dual-core Arm Cortex-M0+ processor with flexible clock. The m0+ and m0 belongs to the Cortex-M family of processors and are dedicated mcu chips. It is a 32 bit MCUs with two stage pipeline (m0+) and three stage pipeline(m0). It replaces ARM legacy Armv7 and Armv9, 8 bit SoCs. The M0+ also adds an mpu unit(optional) to prevent memory corruption, Jtag support which can be interfaced with OpenOCD for software level debugging. It has NVIC, wake-up interrupt controller and Single master - AHB-lite (AMBA-3). Considering its Very low power consumption when compared to other M-series cortex mcu SOCs its one of the popular choices for such IoT sensing applications.

It has 26 multi function GPIO pins, including 3 analogue inputs, 2 × UART, 2 × SPI controllers, 2 × I2C controllers, 16 × PWM channels, 1 × USB 1.1 controller and PHY, with host and device support and 8 × Programmable I/O (PIO) state machines for custom peripheral support. The pico has very good GPIO support so that we an interface analog as well as digital sensors that do the actual physical sensing and the ease of communication provided by the pico makes it one of the best candidates for our choice of as an overall sensor. The rp2040 is also used in Adafruit feather board-16kb flash. Arduino Nano rp2040 - Arduino IDE support. Sparkfun Micro -Sparkfun Qwiic connect system and Seed Studio - Small form factor with less GPIO pins.

The raspberry pi acts as a host system where it has contends and receives data from multiple field sensor i.e the pico boards. The pico and raspberry pi 4 communicate over serial protocol. The sensors are integrated to pico over analog and digital pins respectively. The data validation is done on the host system.

Further we avoid bare metal: Using a vendor sdk or open-source sdk can help in preventing rewriting the the whole hardware stack and focus more on security.

We use open-source Micropython and Circuitpython and it enables us to use python libraries and also port the code and built systems to other MCU's.

Further, the security challenges can be solved as by confusing the sensor readings: adding garbage data between certain indexes to confuse the attacker and these readings can be parsed only by the host system.

The working of the host system can be given as:

1. Open the serial interface to talk with the data port of pico flashed with CircuitPython.

2. Validate the json sent using rules set by pydantic validation layer on host.

3. If validation passes than sent to pico over data port. The baud rate for this serial communication is set at 9600.

The sensor system, i.e the pico is flashed with Circuipython and is always in listening state on data port. It receives a json ,locks the serial port and performs json task in a time sliced or preemptive manner. It then injects dummy data using RTOS Secure scheduler task. Once complete, the pico sends back the result to host and unlocks the serial port.

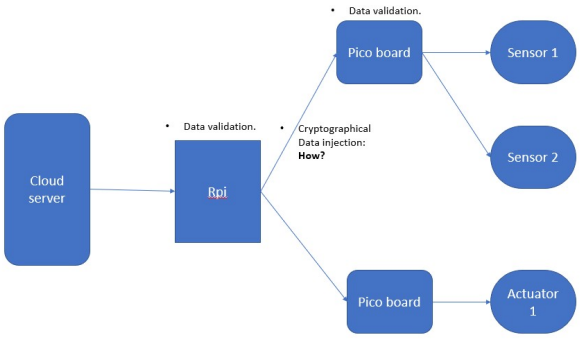

Figure 1: Overview of the system model

Figure 2: Data injection for a analog sensor

Once the host recives the data from the sensor, it converts byte array to json. Examines the Device State and Sensor state. If Device State is OK and Sensor is State OK we parse the Sensor value. The host is programmed to have the knowledge of where the dummy data is present and the actual values will be parsed negating the position of dummy data.

3.1 Data injection

Data injection needs to be done by the sensor system i.e the pico micro controller in our case. The injection needs to be done such that any adversary that tries to sniff the data over the serial , given that it is successful in reading the data, should not be able to identify the fake data from the real data. This means that we should have a true random number generation on the pico, the pico should place the true data in this position and send the data as a packet of n values where only 1 value is the real data. Further we should have a mechanism that the position of the true data is known or understood by the host system i.e in our case the raspberry pi 4.

3.2 Data validation

We follow the pydantic approach of message validation. Pydantic is a python library that helps in json validation over the network.

Pydantic is a data validation library in pure python and is commonly used for faster validation of request forms on web-based servers through RestAPI like FastApi. Pydantic has:

1. Smaller code signature.

2. Pydantic types helps to cover many corner cases which wouldn't have been possible with if-else.

3. Can be scaled as the json increases with adding few lines of validation fields in the class.

4. Pedantic validation from host side has very less/negligible overhead.

4 IMPLEMENTATION: DATA INJECTION

Theoretically, as described the data injection needs to be done in such a way that the attacker, even if is able to read the data should not be able to identify the fake data from the real data in a series of data values. Hence we send a packet consisting of n data values in which only ith data value is the true data. This value of I is a random number generated by the sensor. The value of this I has to be somehow known to the host. As we know, there is no real random number generator that exists that can generate a true random number only by software. By definition [2]: There are two essential security requirements:

1. The random numbers must have good statistical properties: Good statistical properties means more than just equal amount of '0' and '1' bits..

2. The knowledge of a sub sequence of random numbers does not allow one to guess the sequence with a better probability than without this knowledge.

As already know, there is no true random number generation possible by software, in most systems true random number generation is done by physical sources of randomness. These sources of randomness come from the nature and can include: Quantum random properties: nuclear decay, shot noise, photon emission or Classical random properties: thermal noise, atmospheric noise etc.

The system implementation that we describe such that we need to generate a true random number on the sensor device and in such a way that it is deterministic for the host system , but not the attacker is itself an conflicting requirement and this problem cannot be solved. The only way we could think over it was to have a an deterministic, software based random number generation on the host as well as the sensor system. While this helps us in data injection, but fails to satisfy the other condition where in it cannot be tracked by the attacker. Given that the attacker identifies the software based random number generation algorithm, we do not have defense mechanism against it.

The other way to deal with this is problem is to use a natural source of randomness to generate a random number. This could be as simple as the noise from the , but this noise should be at same levels for the host as well as the sensor system and such a system may is an imperfect system. It is not a reliable assumption that we can get the same level of noise captured on both the ends.

Here in we propose a pseudo-natural source that can be used as a seed for the random number generation, the clock jitter in the serial communication between the Rpi4 and the pico board. This clock jitter when seen from the Rpi and pico board remains known to both the parties at all times but is difficult for the adversary to track. The jitter in serial communcation is well known and is always accounted for[3] in all serial communications like RS232, RS485 or UART. Given that the adversary is also tracking the system, each time the adversary adds up its own interference to this and

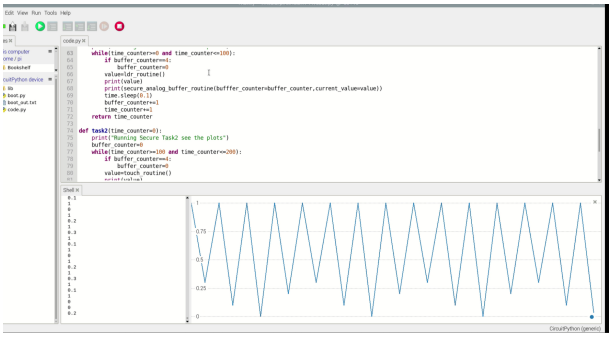

Figure 3: Data injection for a digital sensor

the seed starts to vary even more. We believe this can be a good natural source for getting a random number. This has not been implemented or tested.

5 IMPLEMENTATION: DATA VALIDATION

Data validation is implemented using the python library pydantic. Data validation and conversion is one of the tricky part of getting external data into any system. This validation may include checking for required fields, correct data types, converting from compatible types (for example, strings to numbers) etc. Pydantic has lots of inbuilt data validation as well as it supports creating our own validation checks. These validation checks can be in the form of alphanumeric, a particular sequence with valid and invalid entries, while also providing us the ability to set custom ranges to the numeric entries.

We use pydantic to validate sensor send as well as receive data. This data is made into a json format. The validation checks make sure the sensor is connected to a valid pin, i.e a analog sensor to a analog and digital sensor to digital pin.

The implemenation is done on JSON and the validation format is as seen in the figure 4.

6 IMPLEMENTATION: SYSTEM AND THE MODEL

We communicate with the sensors using Pico board and sent this data to the Raspberry Pi. Two sensors, a digital touch sensor that outputs its value as 0 or 1 and an analog light dependent resistor that outputs its value as a analog voltage have been integrated with the Pico board. We have created dummy real time tasks that read this sensor data in regular intervals.

The Pico board is programmed using Circuit Python while the Raspberry Pi 4 is programmed in python 3. Circuit Python is a derivative of micro-python that is a lightweight implementation of python for micro controllers. This sensor data is injected with fake readings and send to the Raspberry Pi by the Pico board. We have created two buffers namely a digital buffer and a analog buffer that has a real value as well as the fake data values for digital as well as analog sensors. This have been matched with the real data and its is considered that the injected data is close to the real data such that any adversary cannot guess the injected data as an anomaly.

```python
from pydantic import StrictStr,StrictBool
class DeviceSetting(BaseModel):
    DeviceName: StrictStr
    DeviceID:StrictStr
    DeviceVerified: StrictBool
    DeviceSensor:Optional[DeviceSensors]
    Tasks:Optional[Tasks]

def json_verify(json_file):
    """
    This function helps to verify the json file
    """
    try:
        json_file = DeviceSetting(**json_file)
        return True
    except pydantic.ValidationError as e:
        print(e)
        return False
```

Figure 4: Data injection for a digital sensor

The Pico and the Raspberry Pi communicate over serial communication. The Raspberry Pi distinguishes the fake data and the real data as a predefined pattern it receives from the Pico. This pattern can be implemented as a more complex function such that any adversary cannot use a pattern recognition algorithms over the data and has been described above as using the communication line itself for generating true random numbers.

We have been successful is getting the graphs for the injected data and is seen that the data injection is done as expected such that we have a fake data as every nth reading. This position of n as of now is a pattern that is hard-coded and known to Rpi as well as the Pico board. The graph of the current implementation is shown to show the data injection for digital as well as the analog sensor.

7 RESULTS: DATA INJECTION

For this project we are injecting garbage data after every t+1 seconds for both digital and analog sensors. The data injected is such that it is normalised with the previous and next data or the range of data at time T ,to prevent the attacker from identifying the type of device.

The overheads will depend on the no of readings is expected from the sensor. Given that if 20 readings takes 20 systicks than we are generating 40 readings with 20 actual and 20 garbage. This actually takes 40 systicks. So the overhead doubles over every reading. This can be improved if garbage data is inserted after certain time intervals only or at random positions but this may hamper the security of the application which is our main goal.

8 RESULTS: DATA VALIDATION

The data validation was tested and we tried to send the pin as an Int value, when it was expected as a list in python. The json we are validating is:

Figure 5: Validation failure

Figure 6: Validation failure

"DeviceName": "pico", "DeviceID": "001", "DeviceVerified": "true", "Tasks": "No":1, "pins":[0]

Here in, we configure that the DeviceName, DeviceID should be of type strings, DeviceVerified is type bool, and No is an int value, while pins is a list. When any of this is sent wrongly, the validation fails and the data is not sent from the host. We tried to send the pins as a int value and the validation fails. This is shown in the fig 5.

When we tried to send the device name and ID as an int, while sending the DeviceVerified as an string value. The errors are seen in fig 6.

8.1 Overheads

Our observation shows pydantic does a good job in validating the entire JSON file with a very less overhead and within a small code signature. Pydantic strict types like Strictstr ,Strictbool helps in covering edge cases and preventing automatic typecasting in Python. However with respect to overheads as Pydantic is class based on BaseModel memory allocation is done once everytime the host program runs. The overhead for validation from Host side will linearly increase for pydantic as the JSON file increases in size as it validates each line individually in the JSON. Optional fields in pydantic also adds to overheads even if the fields are not included in the oriuginal json file. However this issues can be resolved by:

1.Choosing a medium performance host processor like modern computers or hybrid micro-architecture or low cost single board computer. Since the host is not expected to be running in a energy critical environment, unlike the sensor system, we do not see any drawbacks in having it as a medium performance processor with a few watts of power consumption.

2. Breaking the JSON into small pieces to be sent from host like verify device JSON ,task1 json, task2 json etc. Here in each task JSON refer to individual sensor reading tasks and there could be a design where in we have a few common tasks for all nodes that do the verification and other checks.

9 FUTURE WORK

Since the current implementation does not implement a true random data injection patterns, this can be improved. We need to find a true source of random number generator which can generate the same random number for the host as well as the sensor system, while making sire the attacker is not able to deduce this random number. Further, the system can be designed that only one packet sent in a fixed time interval which is encrypted, acts as a packet with the data of random number. This data is used a seed for random number generation on bot the devices while at the same time we are sure that the attacker has no information of this. Further, to make the system more robust, we can have data padding such the the data traffic between the raspberry pi and and the Pico board as well as the cloud and the Raspberry Pi is not useful for any attacker to get any information if it tries to sniff the traffic. Also, we need to add more sensor nodes and have stricter validations with newer and better rules. The rule definition itself can be a bigger topic of study and what serves as useful and tracking the attackers.

REFERENCES

[1]]weiderreal D Yu Weider. [n. d.]. Real-Time Operating System Security. ([n. d.]).
[2]NC state universty: lecture notes-ECE 592 / CSC 591: Cryptographic Engineering and Hardware Security
[3]https://www.analog.com/en/technical-articles/determining-clock-accuracy-requirements-for-uart-communications.html