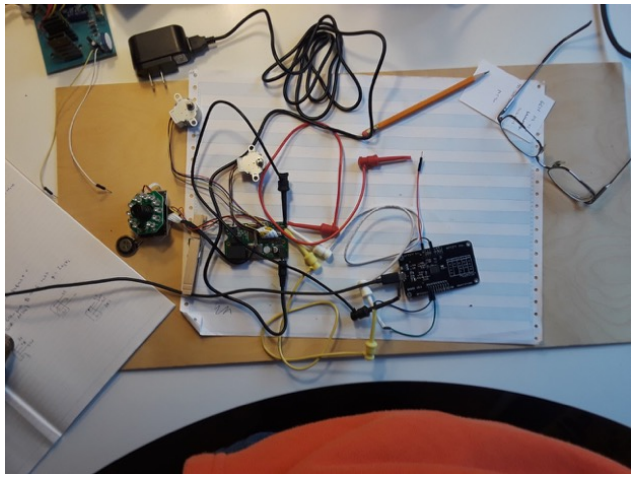
Internet of Things: Outline

John L. Manferdelli
johnmanferdelli@hotmail.com

August 4, 2020 11:00

Welcome to IoT



- IoT Devices
  - Contain digital processors and memory, just like desktop computers and mobile phones.
  - Are small and ubiquitous.
  - Are components of important things (cars, medical devices, drones, security systems).
  - Have (usually wireless) network connectivity.
  - Are opaque (You don’t usually get to see how they work inside).
  - Are physically accessible. [So physical characteristics (electronics, …) are as important as software in thinking about security.]
  - Generate more data than you can possibly imagine (I know you can imagine quite a bit).
  - Sense and control most of the systems we use and more so in the future.

Example IoT devices

- Routers, Switches
- Home alarms, cameras
- Industrial process monitoring and automation
- Washing machines, refrigerators, dryers, smart thermostats, TVs, DVRs, receivers, stereos, MP3 players
- Mobile phones, toys, game consoles, hard drives, printers, 3D printers.
- Cars, robots, drones, trains, planes, ships, phones, infrastructure components (navigation, highways, power distribution, water distribution, airports, ports, train depots, radars, communications devices)
- Medical monitoring and delivery devices
- People (pacemakers, etc.)

IoT by the numbers

- Number or ARM processors deployed per year: 12 billion
- The new Mercedes S-class has 63 microprocessors
- Number of laptops: 160 million/year
- Worldwide technology spending on the Internet of Things to reach $1.2T in 2022, attaining a CAGR of 13.6% over the 2017-2022 forecast period according to IDC.
- US dominates services and software but not hardware production

There’s a lot to learn

- Hardware (electronics)
- Software and networking
- Radio modulation, propagation and Software Defined Radios
- IoT based sensors
- IoT specific busses hardware and protocols
- Interaction with the real world: navigation, sensor characteristics, control and actuation
- IoT Reverse engineering: design information, simulation, hardware tinkering, software tinkering, exploits and attacks.
- Data collection and Machine Learning for IoT collected data
- Drones and navigation
- Cloud management of IoT devices: Amazon, Microsoft and Google.
- Ordering parts from Amazon (maybe you already know how to do this)

Introduction to Electronics

- Lumped circuits, voltage, current, impedance, circuit diagrams
- Kirchhoff, Thevenin, Norton
- Timers, circuits, feedback, oscillators, amplifiers, filters, digital gates
- Capacitance, resistance, inductance, wave forms, kirchhoff and part/circuit modelling
- Transistors, diodes, IC’s
- Oscilloscopes, meters, spectrum analyzers, perf boards, discrete circuits
- Radio: Maxwell and all that. Travelling waves, antennas, aperture, SNR.
- Computer architecture, clocked circuits, memory
- Power and computing
- IoT architecture and sensors
- Tools and resources

IoT Hardware

- Microcontrollers
- Busses (i2c, spi, canbus), EEProms, uart, jtag
- Computer architecture: CPU, MMU, IO
- Arm architecture, MIPS, Xtensa
- How do we identify chips? Get design information?
- Finger-printing and timing glitches
- Firmware

IoT Sensors and Arduino

- Cameras and CCD’s
- IMUs
- Temperature, humidity, infrared
- GPS
- Radar, lidar
- RF receivers and transmitters
- Sound, vibrations
- Barometer (altimeter)
- Labs, fabs

IoT Software and Networking

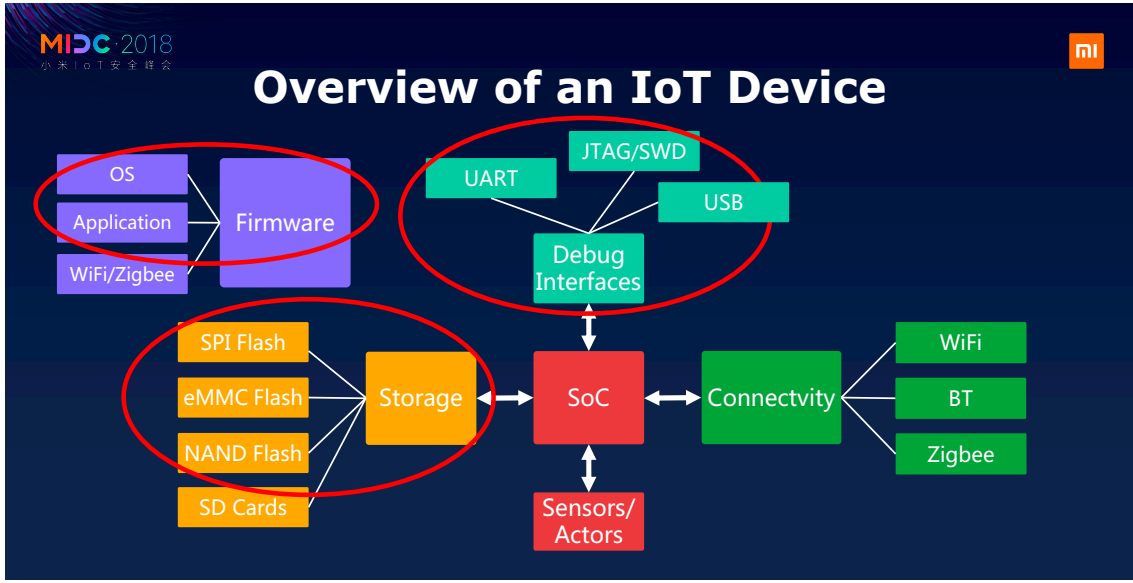
- IoT OSs, configuration, protocols.
- Booting: U-boot, GRUB, BIOS-like files
- Communications: Networking (IP, TCP, protocols)
- Distributed Computing: Authentication and authorization, crypto, ssh, ftp, tftp, telnet, netstat, nmap
- Software development.
- Software vulnerabilities and safety
- Updates: When do they come, how do you know?
- Packing and unpacking images.
- Cloud collection, provisioning and management.
- Persistent rooting.

Radio and SDR

- Modulation (OFDM, BPSK ...) and propagation
- Digital signal processing
- GNU radio
- Jamming, spread spectrum
- SDR’s: HackRF, Ettus USRP

IoT reverse engineering, hacking

- Arduino, raspberry pi, perf boards
- Disassembly and reverse engineering
- Identifying flash chip types and extracting their contents
- Analysis of flash image contents
- Vulnerability analysis with unpacked firmware image
- Patching firmware, application, OS
- Badges and interfaces
- binwalk



Data analytics

- Collection, provenance and data formatting
- Probability, randomness, information, distributions and significance
- What are we trying to learn?
  - Features, supervised and unsupervised learning
- Regression, similarity, curve fitting and loss functions
- Clustering
- Bayes theorem, maximum likelihood re-estimation, EM and latent variables
- Classification
  - Naïve Bayes
  - SVM
  - Neural networks
- Principal Component analysis and the curse of dimensionality
- Deep learning

Putting it all together

- Drones
- Navigation
- Stability and control
- Identification and characterization
- Counter-measures

Systems engineering and resilient design

- Cloud management
- Attacks, threat analysis, common mistakes (like the unencrypted SPI communication between the MCU and the Zigbee part in the door-locks in ISEC)
  - Secure development practices
- Ecosystems, standards, legacy
  1. How does a "consumer grade" IoT device differ from industrial IoT devices (e.g., factory floor applications)
  2. Map the "application" SW update mechanism on some IoT devices.
  3. Contrast the architecture of several IoT ecosystems: ARM? Risc-V?
- References

  Mitre: System Engineering

  NIST IoT Standards

Prerequisites

- I said there was a lot to learn and you'll need some background. Most people will not have a "perfect" background but if you know too little, in advance, you'll have to play a lot of catch-up. On the other hand, if what's missing in you're background is circumscribed, you'll probably have a good time. Talk to me if you have questions. Here's a guide to prerequisites.
- Electronics
  - Ideally, you'd know the basic physics of electronics and magnetism and basic circuit concepts.
  - Some "signpost's" are:
    - What's the definition of a volt and an amp?
    - What is a capacitor? resistor? inductor?
    - What's an oscilloscope?
    - What is impedance?
- If you have a degree in physics or EE, it's very likely you have this covered. If you're a ham, you're covered.

Prerequisites

- Programming
  - Do you know C, C++ or C# well?
  - Do you know how to use an IDE?
  - Signpost: Have you written and debugged a C++ program over 1000 lines of code in the past four years?
- Software
  - Do you know computer architecture?
  - Do you know how to program a network application?
  - Do you know Linux and (parts of) the system call interface?
  - Signpost's
    - What's the difference between "system mode" and "user mode"?
    - What do the system calls fork and execv do in Linux?
    - What is a race condition? What's a process on Linux? Thread?
    - Have you ever written a short assembly level program?
    - Do you know what any of the following are: IP, UDP, DNS, BGP, TLS?
    - What's the difference between a symmetric cipher and an asymmetric cipher?
- If you have a degree in computer science you likely have adequate background for these two topics.

Contributors

- Reverse engineering: Dennis Giese, Troy Shurtleff
- Reviewers: Chris Day, Bob Wisnieff

We gratefully acknowledge financial support from Stanley, Black and Decker which facilitated the development of course materials and labs.

Example exercises

1. Use an Arduino to measure light intensity (roughly) using a photo resistor.
2. Use an Arduino to measure temperature (roughly) using a thermistor.
3. Modify configuration information and update for, say a camera.
4. Modify firmware on an Arduino. How about an ARM based processor.
5. Map the boot sequence on an Arduino and an ARM based processor.
6. Use an Arduino to get position on earth using GPS.
8. Use a pair of Arduinos to transmit/receive wireless information using private channel (HC-12).
9. Reverse engineer a piece of software.
10. Log into an IP camera, change parameters (see the reverse engineering exercise).
11. Modify a router's OS/firmware.
12. Use updated SW to implement a side channel using an LED, speaker.
13. Design an Arduino based system to collect information from devices.
14. Develop an optical communication link using an LED or laser and an optical transistor.

Example exercises

14. Extract encrypted passwords from /etc/shadow and use a password cracker to find them.
15. Interrupt boot on u-boot to become root and change files.
16. Develop an entropy measurement and use it to find keys in an image.
17. Discover the "roots of trust" (public/private keys) embedded in an image and modify them.
18. Develop a mechanism to spoof a GPS signal. DO NOT TEST THIS without talking to a lawyer to ensure legal compliance. You will, at a minimum need to do tests in a Faraday cage. Seriously, don't do it.
19. MITM an IoT device via a router (a camera should work). What does it talk to? What does it transmit?
20. Map the update mechanism of an IoT device.
21. How to update firmware and OS when embedded root CA is expired.
22. Map the IoT devices in [your house, the IoT lab, a hotel, a manufacturing floor]
23. Desolder a ROM and read the image.
24. Find a "fingerprint" for an IoT device? Can you measure it remotely?

Example exercises

23. What are the available sources of entropy in an IoT device? At what rate could you generate AES 256-bit keys with such entropy?
24. Map all the important configuration files on an IoT system.
25. Do some of the exercises we did with the Arduino with a Raspberry Pi.
30. Stimulate a glitch on a pin using a few SDRs in close proximity to a known board/chip.
31. Add capacitance to a interface that makes measurements inaccurate. How would you detect HW trojans in an IoT device?
32. Estimate the RF emanations from an IoT device? How would you shield them?
33. Find a moderate cost tamper evidence system and estimate the cost to defeat it. How would you use scale to help limit the risk?
34. Intercept and read an 802.11 message using an SDR.
35. Jam an 802.11 message.
36. Locate an 802.11 emitter using a directional antenna (coffee can or pringles can depending on band).

Example exercises

37. Measure g with optical transistors and small lasers (about $5).
38. GPS/dead reckoning, now do it with RTL-SDR and HackRFOne (you'll need an amp and filter).
39. Design and build a weather station.
40. Make the weather station communicate with a computer in the house (say, using NFC).
41. Jam the protocol using an HackRFOne or other SDR.
42. Build a morse code detector and decoder (using light and sound).
43. Detect when your dog is barking.
44. Simulate a buffer overflow attack.
45. Reverse engineer a child's toy (say a remote-controlled tank).
46. Scare your kid by taking control of it.
47. Figure out where you are with a GPS sensor and verify it with other means.
48. Design and implement a GPS based route planner with waypoints.

Example exercises

49. Measure the takeoff speed of plane with an accelerometer.
50. Detect people in classroom. Can you count them?
51. Determine when to water the plants.
52. Design a tool guider with a ping sensor.
53. Record sunrise/sunset with a sensors.
54. Pick some code, fuzz, find a buffer overflow, do a code redirect.
55. Use TFTP to download an exploit kit (TO YOUR MACHINE ONLY).
56. Reverse engineer a script (like the reverse shell).
57. Find a key in firmware, change it and repackage the FW.
59. Do the exercises in the addendum to the reverse engineering section
60. Develop a side channel (RF with SDR)
61. Detect when a is plane landing.
62. Detect when a car stops.

Example exercises

63. Determine if you accelerating/decelerating too quickly.
64. Decode a telephone number from the sounds.
65. Build a coded entry system.
66. Design and test an IR communication channel.
68. Read a Flash. Reverse engineer the file system code.
69. Hit the beach? Determine what the sun exposure is today.
70. Count cars. (time of entry exit? distinguish between entry/exit).
71. Have an existing camera send pictures elsewhere.
72. Track garage doors in your neighborhood.
73. Determine patterns of life based on temperature, sound, light (sleep, watch TV, home, gone).
74. Track a route via altitude (works in SF, not Iowa).
75. Detect RF emissions from a board.

Example exercises

76. Build a blimp with radio control and auto pilot to waypoints. (See Ressler, Do it yourself engineering, The Great Courses.)
77. Listen to an FM station with a SDR.
78. Listen and display weather from a NOAA broadcast with an SDR.
79. Slurp wireless traffic with an SDR.
80. Listen to ADSB (air traffic control) with RTL-SDR.
81. We use a lot of libraries that hide the details of the GPIO interfaces like WiringPi, SoftwareSerial, and others. These are all open source. Pick one and figure out the interface details.

Instructor’s sample parts list

- Here is a list of the major parts. I haven’t shopped for the best prices.
  1. Toolkit with soldering iron: Hi-Spec 60 Piece Electronics Electrical Engineer Tool Kit with 30W Soldering Iron, Desoldering Pump, Wire Crimper, Stripper, Cutter, Magnetic Ratcheting Screwdriver and Bits, IC Extractor Tool in Case --- $35
  2. Two Arduino Unos: Elegoo EL-CB-001 UNO R3 Board ATmega328P (2 x 12)
  3. Two Raspberry Pi's: CanaKit Raspberry Pi 3 B+ (B Plus) with 2.5A Power Supply (UL Listed) --- 2 x $48
  4. Two multimeters: AstroAI Digital Multimeter, TRMS 6000 Counts Volt Meter Manual --- 2 x $35
  5. Ten prototype boards and connectors: MCIGICM 10pcs Breadboard 830 Point Solderless Prototype PCB Board Kit Protoboard MB-102 for Arduino DIY Electronics kit --- $20
  6. eBoot 240 Pieces Breadboard Jumper Wires Ribbon Cables Kit Multicolored 80 Pin M/ M, 80 Pin M/ F, 80 Pin F/ F (10 cm and 20 cm), for Arduino --- $10
  7. Sensor pack: Elegoo EL-KIT-000 37-in-1 Sensor Module Kit for Arduino UNO R3, MEGA, Nano – 25
  8. GPS module: 2 of Semoic GPS + BDS BeiDou Dual Module, Flight Control Satellite Positioning Navigator, ATGM336H Replacement, Suitable for NEO-M8N --- $10
  9. EEPROM: MICROCHIP 93C66B-I/P IC, EEPROM, 4KBIT, SERIAL, 3MHZ, DIP-8 (5 pieces) --- $4

Instructor's parts list (continued)

10. Two Attify Badges - Exploit IoT Devices Using UART, JTAG and SPI or other interfaces --- 2 x $20

OR Qunqi 3.3V 5.5V FT232RL FTDI USB to TTL Serial Adapter Module for Arduino Mini Port --- $8

11. Clips: BrightTea Mini SMD Ic Single Hook Clip Electronics MiniGrabbers to MiniGrabbers Wire Lead Test 8 pairs packing --- $20
12. HiLetgo GY-521 MPU-6050 MPU6050 3 Axis Accelerometer Gyroscope Module 6 DOF 6-axis
13. Accelerometer Gyroscope Sensor Module 16 Bit AD Converter Data Output IIC I2C for Arduino --- $5
14. Camera: TENVIS Security Camera- Wireless Camera, IP Camera with Night Vision/ Two-way Audio, 2.4Ghz Wifi Indoor Home Dome ------ $37
15. Wire, solder: Electronix Express- Hook up Wire Kit (Solid Wire Kit) 22 Gage (25 Feet) --- $15
16. Resistor pack: Ltvystore 1500pcs 75 Values 1 ohm - 10M ohm 1/4W Carbon Film Resistors Assortment Kit Assorted Set --- $14
17. Ceramic capacitor pack: 10pF to 100nF 15Values Ceramic Capacitor Set, Hilitchi 525Pcs DIP Monolithic Multilayer Ceramic Chip Capacitors --- $15
18. Electrolytic capacitors: MCIGICM 120pcs 12 Values 0.22uF-470uF Aluminum Electrolytic Capacitor Assortment Kit --- $5

Instructor's parts list (continued)

19. Diodes: (Pack of 100 Pieces) MCIGICM 1N4001 Rectifier Diode, 50V 1A DO-41 Electronic Silicon Doorbell Diodes --- $5
20. Transistors: Laqiya 100Pcs 2N3904 TO-92 NPN General Purpose Transistor --- $6
21. Various chips (555's, 741's, digital logic) for example: Make: Electronics Components Pack 1 --- $60
22. HackRF One --- $300
23. Oscilloscope: Siglent Technologies SDS1202X-E 200 mhz Digital Oscilloscope 2 Channels, Grey --- $400. This is optional and there are cheap oscilloscopes (with less bandwidth) for about $40.
24. Router: TP-Link AC1750 Smart WiFi Router - Dual Band Gigabit Wireless Internet Router ---- $60
25. SD card reader: UGREEN SD Card Reader USB 3.0 Dual Slot Flash Memory Card Reader TF --- $10
26. Linux based computer
27. Power supplies (0-15V), 5V regulated. You can often use cheap portable supplies.

Instructor's parts list (continued)

Extras

1. Desolder station: CO-Z 909D+ 5 in 1 SMD Soldering Rework Station with Hot Air Heat Gun, Solder Iron Station with Stable DC, 5V USB Outlet for Charging, LED Digital Monitor to adjust the Voltage and Temperature --- ~$100
2. Extra HackRF-1
3. Signal generator
4. Reflow oven
5. Magnifier with USB connector for circuit board imaging
6. ESP32 boards
7. Extra SD cards
8. Laser diodes
9. Shift registers, phase lock loops, extra op amps, 555's, digital logic, inductors, photo-transistors, temperature sensitive resistors…

- I'm always buying little parts like barometers, gyros, other sensors, and weird little components. I'm sure there's stuff missing. I often buy on Amazon but there's also Digikey, Jameco, and others.

Computer Security Principles

The security of IoT is, at some level, indistinguishable from that of any computer system (like a PC) but even so, the same security mistakes are made. Here are the themes from “classic” computer security we will hear repeated.

1. Think like an adversary: Most engineers design and test systems to work properly in the face of varying environmental conditions. Nature is subtle but not malicious. Adversaries are malicious and, by deliberately modifying inputs, break design and assumptions to make the system do something it isn’t supposed to do.
2. Know your system: This includes all the edge cases. Good adversaries know your system better than you do (or at least parts of it) and will use feature in ways you haven’t anticipated.
3. Develop carefully: The code should perform the functions it is supposed to and should not perform functions it is not supposed to.
4. Configure carefully: Configured guards should allow what is supposed to happen and prevent what is not supposed to happen. They should not be changed unless the full impact is assessed and should be changed when circumstances warrant very quickly.

Computer Security Principles

5. Isolate and protect computations performing different functions: Unless you fully trust all the code that can modify your code or data, it should be isolated. You should not trust data you cannot verify.
6. Update: You will make errors under the best of circumstances. Deployed code *will* have vulnerabilities and unless you can fix the code in the field, you will be fielding unsafe systems rather soon after they are deployed. That means update and rollback should be cheap, fast and foolproof.
7. Test carefully and relentlessly: At my first job, some people developed code and others did "integration testing" to verify everything works together. Integration testing is nice, but you can never exhaustively test a complex system. At Google, you could not submit code without comprehensive unit test (you often started with the test code). Tests should test that failures are properly handled. Tests should be run before change is made to "official software."
8. Measure as you use and learn: Large systems should log operational data to help ferret out performance and security problems. Analysis should be as automated as possible and done all the time. Network monitoring and probing for open ports fall into this category.

Computer Security Principles

9. Make realistic models of the confidentiality, integrity and availability needs of a system and make sure they are enforced.
10. Guard against adversarial inputs, code and infrastructure: If some program hands your API arguments, you need to ensure that acting on them cannot violate security conditions even if they were fashioned by your worst enemy. Buffer overflows are an example of failing to do this. Operating system calls that write results in user space (or kernel space) must ensure that those destinations can and should be modified by the results.
11. Authenticate with precision and rigor: The source of every piece of code, data or policy must be strongly authenticated. Passwords are not strong authenticators.
12. Authorize what you need and no more: Principle of least privilege.
13. No system is perfect, employ defense in depth: Even with all this, additional safeguards should be employed to detect and block unsafe actions. Use a proxy even if you're convinced your code is perfect.

Computer Security Principles

14. Blacklists are ineffective, allow only what you know is safe: Require safety by construction when possible. For example, I have given students assignments to block unsafe actions in browser html and script; I have never seen them succeed. Correct solutions transform the input into some representation and (automatically) generate code which *cannot* have vulnerabilities. Complex stuff generally can't be made safe nor can "all the vulnerabilities" be rooted out while preserving all functionality.
15. Systems last far longer than you expect: Plan for that.
16. The hardware and software you use to develop and deploy are just as important for safety as the hardware and software you deliver: See Thompson, Reflections on Trust.
17. If you deploy enough stuff, you will see failures you never expected: Look for them and find the causes.
18. Be paranoid but not so paranoid you can't benefit from what you build: Ken Thomson used to say security is easy: "just lock the computer in a room and unplug it." Effective, efficient security with capability are the watchwords.
19. There's some good advice at: https://www.microsoft.com/en-us/research/wp-content/uploads/2017/03/SevenPropertiesofHighlySecureDevices.pdf

License