

# IoT: The Internet of Things



# What is the “Internet of Things”?

**OED** A proposed development of the internet in which many everyday objects are embedded with microchips giving them network connectivity, allowing them to send and receive data.

**Merriam-Webster** The networking capability that allows information to be sent to and received from objects and devices (such as fixtures and kitchen appliances) using the Internet.

**Wikipedia** A system of interrelated computing devices, mechanical and digital machines, objects, animals or people that are provided with unique identifiers (UIDs) and the ability to transfer data over a network without requiring human-to-human or human-to-computer interaction.

**Me** Non-computer objects that both contain a CPU and can communicate over a wide-area network.

# Excluded

- Devices with only a CPU, e.g., my toaster and coffeemaker
- Devices with only local networking, e.g., RF remote controls for ceiling fans
- Special-purpose CPUs embedded in larger, networked objects or computers, e.g., USB flash drives, Apple's Lightning connectors, laptop cameras, keyboards, etc



Must distinguish IoT from *embedded systems*

# Attributes of IoT Devices

- Not a “computer”, and hence often lacks conventional I/O devices: a screen, a keyboard, a mouse or touchscreen, etc.
- Users generally cannot reprogram them
- Generally cannot run outside software (though that may change).
- By definition, connected to a physical device

# Questions

## Standard questions

- What are we trying to protect?
- Against whom?

## Plus...

- Why is IoT security different?
- How do we protect IoT

# What Are We Trying to Protect?

- The computer
- The physical device(s) it monitors and/or controls
- Perhaps its network bandwidth

# Against Whom?

- Who the enemies are depends on the attached physical device
- Russia has supposedly been poking at the power grid
- Iran has been accused of going after dams

# IoT Security Concerns

- Why is IoT different?
- No antivirus
- Limited support lifetime
- Poor user interface
- The programmers are not battle-hardened
- The computer is connected to a physical device



- Antivirus software requires patterns of known malware, and there aren't many patterns
- Antivirus software requires a subscription and payment, and there may not be an economic model
- Besides—is it helpful or harmful?

# Support Lifetime

- Who pays for support? What is the economic model?
- How long will the chips last? How long will they be available?
- When vendors switch to newer chips (because they have to), will older code run on new chips? Will new code run on old chips?

# Desktop and Laptop Computers

- Typical product life: about five years, with a long tail
- Businesses replace gear more often, but keep old operating systems longer
- Vendor support lifetime is often a crucial issue

# OS Support Lifetimes

- Windows: about 11 years
- MacOS: about 3 years
- Ubuntu: about 5 years

N.B. OS compatibility with new hardware can be much longer—it's not necessary to buy a new Mac every three years

- Apple supports its phone hardware for about five years
- Android: a bit over two years at best
- N.B. The Android marketplace is more fragmented

- Phones generally cost \$500–1000
- Computers cost at least that much, often more
- Many IoT gadgets cost a lot less
- For expensive devices, the lifetime support cost is bundled into the OS (and maybe hardware) price
- That doesn't work for cheap IoT devices

- How long does a light switch last? A smart light switch?
- How long does a thermostat last? A smart thermostat?
- Home routers? Security cameras? A multicooker?
- What about toys? Home appliances? Cars?

# Update Desires

- Assume that updates are available? How often will they be installed?
- For phones and computers, the impetus is often new features
- What are the new features I would want that would induce me to upgrade a smart lightbulb?
- How will I know if my washing machine needs an update?



- In general, IoT devices offer a poor *user experience*
- Things that are simply on a computer, e.g., typing a password, are very unpleasant on, say, a lightbulb
- Ergo, authentication is often poor
- There may be no display to indicate system state or to make requests of users

- Programming is not the same as secure programming; the latter has to be learned separately
- Secure programs—and secure architectures—need all manner of defensive techniques, including paranoid input checking and extensive fault recovery, plus liberal helpings of crypto
- Organizations that are used to fighting hackers have learned many such lessons the hard way. Newcomers often need to learn.

- For ordinary computers, the target is usually the data, though sometimes it's network connectivity
- IoT devices rarely have much interesting data, though they do have network connections
- But—they're attached to physical devices
- This can attract hackers...

# If That's the Structure...

- From these structural issues, it's clear that IoT security is hard
- But—security people don't get to decide if IoT is a good idea
- Our challenge: securing things anyway

**Thing** The networked computer that talks to a physical device

**Hub** An intermediary between (some) Things and the Internet. Hubs often use a local, non-TCP/IP link to talk to Things

**Manager** A local controller for Things. Managers may be built into Hubs or may be separate boxes

**Vendor** A vendor server with which Things or Hubs must associate

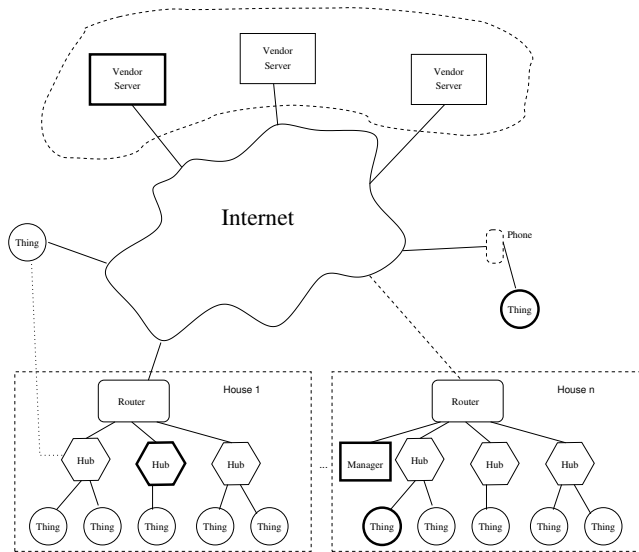
# Questions About Operational Environment

- Is the *Thing* directly connected to the Internet?
- If not, it behind a firewall or NAT, or is it behind a *Hub*?
- Must the Thing use a Hub to talk?
- Must Vendors be involved

# Why Vendors?

- In the consumer world, remote access to Things is often needed: think of thermostats, alarm systems, and more
- But—it's hard to talk directly to consumer devices; they're almost always behind NATs and usually don't have stable IPv6 addresses
- Battery-operated devices can't be online 100% of the time, but they can poll the vendor at reasonable intervals
- Usability is a problem: most consumers would have trouble learning an IP address and setting up a DNS entry for each of their devices
- The usual solution: consumers talk to their devices indirectly, via the vendor
- This means that vendor security is part of our concern

# A Possible Architecture





# Analytic Approach

- Metanote: assume that the attacker's ultimate goal is the physical device
- Identify vulnerable entities
- Identify what is at risk from their compromise, i.e., what the desired execution environment is for attackers
- Identify the weak points and devise defense

# Who Can Attack Whom?

- Hubs can attack Things—but a Hub can be a firewall for its Things (and we'll lump Managers with Hubs)
- Of course, Things can attack Hubs...
- Vendors can attack Hubs and Things, either directly or by pushing out nasty firmware updates
- Hubs and Vendors (and some Things) are subject to direct Internet attacks or attacks from compromised home computers
- In other words: this is an environment with many threats, and with threats to all components

# Mandatory Minimum Security

- Encrypt all links
- Partly for confidentiality, but often more for integrity
- Addresses vulnerabilities of all parties
- But—encrypted to *whom*? Who is the endpoint for each communication?

# Encryption: End-to-End or Hop-by-Hop?

- If a Thing needs to talk to a Vendor, where does its encryption terminate?
- End-to-end is the ideal, but we need intrusion detection
- IDS is hard for Things; they're low-powered and can't easily yell for help
- So: encrypt hop-by-hop, but use end-to-end integrity checks
- (A privacy risk? The user owns the Hubs and the Things. There may be privacy issues with respect to the vendors, but end-to-end encryption won't solve that problem.)

# Protecting Things: Hubs

- Things can be attacked by Hubs—how?
- Two obvious ways: something impersonating a Hub, and by a compromised Hub sending malicious commands
- The first can be dealt with by strong authentication
- For the second, use *good* input parsing and sanity checks (stay tuned)

# Introducing Endpoints

- How does a Thing know the proper Hub?
- How does it know the Hub's authentication credentials?
- What happens if you sell the Thing? How does the new owner reset it?
- What happens if you replace the Hub?

# The Resurrecting Duckling

- No great solutions! No keyboards, no screens, setting lots of passwords is a pain, etc.
- Best solution: let Things *imprint* on the first Hub they talk to
- Take advantage of physical proximity for initial pairing
- Have a physical reset button to restart
- Or: use a phone app as an intermediary
  - Use NFC (or maybe Bluetooth), QR codes with cameras, and more
  - Can exchange strong secrets—no passwords!

# Protecting Things: Vendors

- A compromised Vendor can send evil commands through the Hubs
- It can also push malicious firmware
- Do intrusion detection on the Hubs *and* the Things
- Why on the Hubs? They're more capable
- Why on the Things? They have to protect themselves from compromised Hubs, too
- All firmware updates *must* be digitally signed



# Digitally Signed Firmware

- Vendors have developers and a server
- Strongly separate the two halves, and protect the signing key
- Vendor developer compromise is *extremely* serious—but consumers can't tell if vendors are doing things properly
- No ducklings needed here—the vendor can manufacture in its certificate
- (But what if the vendor signing key is compromised? What if that certificate has to be revoked?)

# Sanity Checks

- For some devices, improper commands can have serious consequences
- Solution: separate sanity checks, preferably in hardware
- Example: a low temperature circuit for an IoT thermostat
- Danger: watch out for sanity checks only in regular firmware (but cheap devices probably can't do better)

# User Accounts

- What protects user accounts on the Vendor?
- We *know* how vulnerable most folks' passwords are
- MFA is best—but how to force adoption? And what about recovery from lost credentials?
- And: *must* have hardware reset on Things, in case they're sold by someone who forgets to execute some handover mechanism
- Example: IoT light switches, conveyed with a house in an estate sale


# We've Had Problems!

- Some devices, e.g., security cameras, have used default passwords and been directly exposed on the Internet
- Attackers have used these documented default passwords, which many users never change
- Sanity checks help here, too

# Structural Problems

- We still haven't solved our structural problems
- Still have limited support lifetime, poor user interface, inexperienced programmers
- Plus: if we assume reliance on Vendors, what if the Vendor turns off the servers or goes bankrupt?

# Economic, Not Technical

- These issues are fundamentally economic, not technical
  - In other words, a purely technical approach cannot solve them
  - Even “purely” technical problems have an economic dimension—what will force adoption of expensive solutions?
-  This means that the solutions will have to be economic and/or regulatory

- Perhaps: mandatory second sourcing of abandoned services and devices
- That is: a law requiring that code for all IoT devices and their services be escrowed
- When some device or service is EOled or discontinued, the relevant code *must* be offered for sale, and released to the public after 90 days if not picked up
- Exercise: what about code-signing keys?
- Exercise: how should Things or Hubs be rehomed to a new server, or to a personal server?

# Suicide Switch

- Should unpatched and unpatchable devices be allowed to exist?
- Liability laws might help with companies that still exist, but what about devices left behind by abandoned companies?
- Sue the owners? Is that fair?
- Maybe Things should have a finite lifetime...
- (Idea due to Dan Geer)



# Lifetimes and Costs

- What is the normal lifetime of a Thing, independent of these security issues?
- Cars last 10–15 years, and sometimes longer—having them die after five years is unacceptable
- Major home appliances may last ten years—but they're still too expensive to discard
- Light switches are cheap, but installation isn't—most folks would need to hire an electrician
- And some bugs aren't economically fixable, e.g., boot ROM issues
- But—the cost of an unpatched bug is imposed on society, and that's an economic issue, too

# No Perfect Answers!

- Support costs money
- Vendor servers cost money
- Lack of support and lack of vendor servers cost money
- The ultimate questions: who should pay, and how?
- Front-loading—demanding payment on first purchase—can be economically feasible, but it drives up prices; without regulation, that doesn't work in the market
- And if you think it's bad for consumer gear, what about industrial control systems?

- What are the fundamental issues?
- What are approaches to solving them?
- Are they feasible?

- Some of the basic problems are economic
- If we can't find a technical solution, we need either an economic solution or a regulatory solution
- But some of the economic issues, e.g., paying for support are pretty fundamental, so we need either a technical solution or a regulatory one
- Some of the technical issues appear extremely difficult, e.g., auto-generating patches or auto-backporting them to an old release

- Who operates servers if the vendor goes bankrupt?
- Who gains access to the firmware signing key?
- It's one thing to make the code base open source; it's another to ensure trust

# There Are Some Technical Approaches

- Auto-updating is probably a good idea
- So is auto-rollback if the patch seems to be causing problems
- There are ultra-reliable software systems: avionics, phone switch software, satellites and Mars probes, etc.
- It is currently *expensive* to produce such code—but could there be tools to make it easier and cheaper?

# “Safe Mode” on Mars Probes

- Shut down all science activities
- Find Earth
- Keep the batteries charged
- Send a “help!” signal
- Listen for software updates

# What is Safe Mode on a Car?

- Shut down entertainment, tire pressure monitoring, cruise control and driver assist, etc.
- Keep the engine running, make sure that the throttle, brakes, and steering still work
- All that is harder on drive-by-wire cars!
- Electric and hybrid vehicles need a lot more software just to function



# Conclusions

- The purely technical issues appear to be solvable, though it will take research and tool development
- There are economic issues that are probably beyond the reach of purely technical efforts
- Given the wide range of IoT devices, it is likely that no single solution, technical, economic, or regulatory, will work for everything

# Questions?



(Osprey, Central Park, October 7, 2021)