Securing an IoT Product From the Ground Up

Marti Bolivar <marti.bolivar@linaro.org>
11 September 2017
Hi.

This talk is:

- A recap of antipatterns / my own past sins in security engineering
- About a better way
- Intended for working engineers developing products on a schedule

This talk is not:

- An overview of cryptography
- About what technical solutions to use (though ping me after!)
- An exhaustive treatment of security engineering
By **IoT product**, I mean: a mass-produced consumer device which communicates via a network, whose primary purpose is not computing.

- I’m additionally going to focus a bit on “sub-Linux” microcontrollers running an RTOS.

By **securing**, I mean: “building systems to remain dependable in the face of malice, error, or mischance” (Ross Anderson).

- Parts of dependability are inherently human value judgements about the system
- Security is an ongoing process, not a yes/no attribute about any given system
“Securing an IoT Product”: Buzzword Decoder Ring

● Example IoT Products:
  ○ Home thermostats and light bulbs with 802.15.4
  ○ A/C plugs with WiFi
  ○ Heart rate monitors with radios

● Non-examples:
  ○ Laptops and smartphones (they’re computers)
  ○ ATMs (sold to banks and other businesses, not consumers)
  ○ Networked voting machines (sold to governments, not consumers)
  ○ Credit reporting agency web sites (eep)
  ○ Nuclear command and control centers (hopefully, not for sale)
About me

● I’m an embedded software engineer.
● I work at the Linaro Technologies Division (LTD) doing infrastructure work and reference implementations for IoT.
● Before that, I founded and ran an embedded shop.
  ○ Open source hardware, firmware, and software
  ○ Consulting work with industry and academia
Linaro and IoT

- Linaro is a not for profit, member-funded, open source engineering organization
  - Focus is on the Arm ecosystem, but works with various upstream projects and technologies.
  - Members include Arm licensees, product companies using Arm, Linux distros, and others.
- Security Working Group (SWG) works on Linux hardening, open source Trusted Execution Environment (OP-TEE), other projects.
- Two groups work on IoT, including security:
  - Linaro IoT and Embedded group (LITE): Develop foundational IoT infrastructure as open source. Examples include a secure MCU bootloader, and supporting RTOS development with cryptography, device memory protection APIs, application / kernel system call boundaries, etc.
  - Linaro Technologies Division (LTD): Develop and document end-to-end and open source reference IoT system implementations, based in part on LITE’s work. Contribute everything from bug fixes to entire subsystems back upstream. Maintain vigilance against bitrot.
My Projects Have Followed a Common Pattern

However, “projects with solid security stories” hasn’t kept up with this graph.

We all know things are broken.

I’m here to share what goes wrong, what I’ve learned, and how we can do better.
Good news: security engineering is a robust field with many experts and established techniques IoT can use.

Bad news: we’re Doing it Wrong right now, and there aren’t enough of these experts to work on every IoT product.

How to win big: the rest of us can (and should!) learn their ways. This will mean workflow changes, which we can do iteratively.
Why secure IoT products?

I bet you know some good reasons.

Just for fun, let’s discuss anyway.
The problems are real

- Major DDoS attacks fueled by embedded devices (e.g. Dyn attack)
- Medical devices have potentially fatal flaws (e.g. heart monitor vulnerabilities)
- Potentially serious issues in new major infrastructure (e.g. dozens of zero-days identified in Tizen)

“Security issues are externalities we have no incentive to handle”

Are you sure? Consumers and governments (IANAL!) are responding.

- Public concern about device security decreases willingness to buy; e.g. Accenture survey
- Bruce Schneier is calling for US government regulation of IoT security
- Senators are listening: Internet of Things Cybersecurity Act proposed, backed by Schneier, Symantec, VMWare, Mozilla, others
- EU advisory body: manufacturer “remains fully responsible for the security of the data processing” of IoT device
There are other reasons

- Economic concerns
  - A desire to avoid device “cloning”
  - Better security as a product differentiator
- Additional concerns in enterprise and B2B contexts
  - Support, availability, reliability contracts
  - In-house products handling important or sensitive information
- You feel it’s the right thing to do
- (Your, or your company’s, reasons go here)

So let’s start securing, hopefully with open source software! 😊
Security Engineering Antipatterns

Ways we’re Doing it Wrong.

- Do Nothing
- Do It Yourself
- Simon Says Security
- Just Add Crypto
- Security Grab Bag
- Aim for Perfection
- Release and Forget
- Kill the Messenger
Antipattern: Do Nothing

This approach just accepts every risk, so it’s not very good at mitigating them.
Antipattern: Do It Yourself

When the system is secure because of custom / secret\(^\dagger\) / awesome protection.

Examples:

- Use of non-peer-reviewed crypto
- Security through obscurity

\(^\dagger\) Let’s not say “proprietary…”
Antipattern: Do It Yourself

Why it doesn’t work:

- Let’s be honest, it might for a while, and it is easy, but security is hard and best done by experts. Hand-rolled systems haven’t fared well historically.
- If your product is commercially successful (which is what you want, right?), issues will get found.
- Compare with Kerckhoffs’ principle / Shannon’s Maxim.
Antipattern: Simon Says Security

When the system is secure because someone important says so.

Examples:

- Vague requirements documents (“End to end secure communication is a mandatory requirement”)
- Security theater (no lighters on airplanes!)

It can happen in times of panic, or when people want security, but aren’t quite sure about how to make it happen.
Antipattern: Simon Says Security

Why it doesn’t work:
The emperor has no clothes.

Going back to the buzzword decoder ring for “securing”: we need to be keeping something dependable, somehow, in the face of some specific “malice, error, or mischance”.

If we’re not being specific about that, how are we securing?
Antipattern: Just Add Crypto

When the system is secure because it uses cryptography.

Examples (it’s secure because we use...)

- SSL!
- TLS!
- DTLS!
- AES!
Antipattern: Just Add Crypto

Why it doesn’t work:

- SSL! (CVE-2014-3566, aka POODLE)
- TLS! (CVE-2014-1255, aka goto fail)
- DTLS! (CVE-2014-0160, aka Heartbleed)
- AES! (IoT light bulb attack by Matthew Garrett, Philips Hue attack by Ronen, Flynn, Shamir, Weingarten)

This doesn’t mean crypto is broken! It’s just not the entire story.
Antipattern: Just Add Crypto

- Crypto may be duct tape, but it isn’t magic
  - Algorithmic attacks happen sometimes; it’s an evolving field relying in part on conjectures in computational complexity, and the only “perfect” crypto, the one-time pad, isn’t practical for us
  - Implementations have bugs

- Crypto is easy to misuse
  - Insecure key management or default passwords
  - One master key for ALL the things (HDCP, anyone?)
  - Buggy key parsing (especially custom formats due to resource constraints!)
  - Not enough entropy
  - ...

Crypto is great! But “my product uses crypto” breaks down in the real world.
Antipattern: Security Grab Bag

When the system is secure because it uses many security technologies.

“We'll need DTLS with certificate authentication, a tamper pin that erases the keys when tripped, and memory protections for keys.”

Security architecture (maybe a good one, but without reasons, how do you know?)

Often an advanced case of Just Add Crypto, this can also be a sign of perverse incentives in an organization.
Antipattern: Security Grab Bag

Why it doesn’t work:

- Well, it actually can work, depending on what you put in the bag.
- **But**: it’s likely to sometimes be protecting the wrong thing, over-protecting something, or not protecting anything

This can waste development time, forcing you to drop features or take a schedule hit, and may cause loss of morale.

Security failures are inevitable and schedules are tight; to secure wisely, know how tech you choose keeps your system dependable against the **most important threats** to you.
Antipattern: Aim for Perfection

When you try to build the perfect system.

Examples:

- Building a bomb-proof door before adding a window lock
- Trying to stop a determined nation state (depending on who you are)

This can happen if engineers get carried away brainstorming. It can also happen if those with security sign-offs ignore trivial matters like deadlines and salaries.
Antipattern: Aim for Perfection

Why it doesn’t work:

- Well, the perfect system never ships, so it’s tautologically secure.
- Let’s say you shipped, though. You did so with issues known and unknown. Security is no different.
- IoT, where physical access is guaranteed, means your Things aren’t your computer. (And security doesn’t begin at shipment, either.)
- Zealotry leaves a bad taste, and decreases buy-in.
Antipattern: Release and Forget

When the system is secure and we’re done, right?

Examples:

● You don’t have all the sources (and your vendors won’t support you)
● You can’t reproduce the build
● You don’t have a support window
● You don’t have a public security vulnerability reporting process
● You can’t update deployed devices

This can happen because your company is in a commodity market and faces tight margins, because it’s a new startup or otherwise doesn’t know any better, etc.
Antipattern: Release and Forget

Why it doesn’t work:

- Makes vulnerabilities unfixable
- Antagonizes your customers
- Antagonizes the security community

Judging by their actions, many companies don’t care. But that’s just saying “security problems are an externality we don’t have to handle” — and we’ve talked about that already.
Antipattern: Kill the Messenger

When the system is secure because you’ll sue anyone who says otherwise.

Examples:

● Making legal threats in response to reports of security vulnerabilities
● Lobbying for legislation to prevent security research about your technology
Antipattern: Kill the Messenger

Why it doesn’t work:

- Maybe it does, if you’re explicitly including legal powers as defenses against malice†, but:
- It can cause **bad press** and **damage your brand**
- You might be concerned about legal expectations getting higher
- It antagonizes people who can **sell your vulnerabilities** instead
- You can’t sue **anonymous actors**
- It doesn’t handle **error or mischance**

† Not a crazy idea, actually.
What to do instead

● Don’t connect or collect unless you need to
● Iteratively build and use threat models
● Use your existing workflows to threat model
● Manage customer and community relationships
● Be ready for when problems arise
Don’t connect or collect unless you need to

- It’s pretty hard to hack a device that’s not connected to a network
- It’s pretty hard for devices to give up information you never collected
- It’s pretty hard for servers to give up information you never sent them

I had to say it …
Build a threat model, and manage your threats

There are many threat modeling approaches. They all involve modeling the system, deciding what the important problems are, and mitigating them in priority order.

Shostack’s approach in *Threat Modeling: Designing for Security* is applicable to IoT development workflows, proven in practice, and concrete.

Start small and iterate. Remember the anti-patterns. Keep researching vulnerabilities, both in your market segment and elsewhere. Apply what you learn.

(If this presentation were a single slide, this would be it.)
A maze of twisty little passages

Lots has been written and done, as you might imagine.

- **Security Engineering (Anderson)** (everyone will tell you to read it, and it is great, but it’s also 1000+ pages and covers topics with limited relevance to IoT)
- **Application Threat Modeling (OWASP)** (you’ll run into this especially if you’re making web apps; it’s also good, but has a top down, waterfall focus)
- **Toward a Secure System Engineering Methodology (Schneier)** (threat trees were intended as the end-all method, but can be hard to get started with)
- Attack lists and tooling ([OWASP top 10](https://www.owasp.org/index.php/Category:Top_10), CAPEC, Metasploit, Kali, …)
- Methodologies ([OCTAVE](https://www.octave.org/), PASTA, VAST, STRIDE, …)
Choosing a threat modeling methodology for IoT

The “Microsoft methodology” as documented in Shostack’s *Threat Modeling* book is useful for securing IoT products - with some tweaks. YMMV.

- Starts lightweight and easy (there’s even a card game!)
- Describes how to improve in an evolutionary way
- Readable, concrete opinions and advice on what to do
- Battle-tested

In short, it maps cleanly to the often messy, iterative, and underspecified IoT lifestyle. Teams with and without security experts can benefit from it.
The Microsoft methodology in *Threat Modeling*

**In rough and condensed form:**

1. **Model your software system**
2. **Decide what to protect**
3. **Protect it as best you can**
4. **Validate (and hopefully ship)**

(We tweaked the software-centric approach.)

Provides a minimal starting point:
- "Model": whiteboard drawing
- "Decide": file a bug based on what you see
- "Protect": fix it
- "Validate": add a regression test case

Also describes how to get more sophisticated as you learn and invest more.

Feedback edges happen as you revisit requirements and revise your priorities. This can help structure meetings between engineering, PM, executives.
Applying the Microsoft methodology to IoT

Some more details, and antipatterns redux. You’ve already avoided Do Nothing!

- **Model your system**
  Make a data flow diagram of your system, considering trust boundaries. (Another tweak for IoT: also look at your schematic or block diagram.)

- **Decide what to protect**
  Choose concretely what to protect, going breadth-first across attack surfaces (avoid Just Add Crypto, Aim For Perfection). Add issues and tasks in the normal bug tracker, to inherit your team’s scheduling and prioritization workflow (this is also about buy-in); document as you go.

- **Protect it as best you can**
  Use standard, peer-reviewed methods that really preserve dependability where you decided it matters (avoid Do It Yourself, Security Grab Bag).

- **Validate (and hopefully ship)**
  Test protections in QA (avoid Simon Says Security). Make an informed judgement about if it’s good enough to ship (avoid Aim For Perfection).
An Example Dataflow Diagram

The canonical diagram is usually a cloud database. This is an example for IoT.

Make your own; adding processes, boundaries, etc. as needed.

Enclosed device

SoC

Application

RTOS kernel

Storage driver

Chipset driver

Chipset driver

IP stack + L2

EEPROM

On-board IC

On-board IC

Network

External phenomenon or controlled device
Use hardware to model assets and entry points

Shostack argues that asset-centric thinking is not useful for threat modeling software.

Our needs, teams, and abilities are a bit different, making assets more useful.

Updating FW in flash via SWD is easy to leave out of a DFD.
Be ready for issues; they will come

Remember Release And Forget?

- Document your security policy and support period somehow, and give people an easy way to file bugs. Respond to bug reports.
- Make sure you can make changes and reproduce the build throughout the support period. This may involve support contracts with binary blob vendors (or just accepting threats to binary components you can’t change).
- Ensure you can securely roll out updates, noting when they are security updates. Updates are part of your system; use standard methods and threat model the results!

What about Kill The Messenger?

- You might not fix everything that gets reported, but don’t just deny issues or make threatening responses.
Parting thoughts

“Although the picture is gloomy, it doesn’t justify despondency. [...] As protection requirements and mechanisms become more part of the working engineer’s skill set, things gradually get better. Security may only be got right at the fourth pass, but that’s better than never — which was typical fifteen years ago.”