The S in IoT Is For Security

Baking Security Into Your IoT Devices

Andrew Watts

Shawn Corey

Outline

- OWASP Threat Modeling Overview
- OWASP Threat Model on Fictitious Devices
- Hardware Hardening
- Firmware Signing
- Secure Remote Firmware Updates
- Operating System Hardening
- Device Application Hardening
- Transport Security
- Backend Application Security
- Lifecycle Management

OWASP Threat Modeling Overview

Threat modeling helps security professionals identify security controls needed to properly secure a system and prioritize security protections within the system.

Identify the following:

- Data stored or processed and what someone could do with that data
- Trust boundaries within the system
- Data flows both within the trust boundary, and that cross it
- Persistent data stored in and out of the trust boundary
- Threat actors and skill level required to exploit
- Impact a threat actor may cause if successful
- Vulnerabilities found at the trust boundary
- Security controls needed to mitigate the identified risk

What are you protecting?

Security is about protecting data and the story that data tells

What is your data?

- Are you collecting location information about people?
- Are you collecting data that identifies a person?
- Are you collecting or processing financial information?
- Can the data be used to attack critical infrastructure or people?
- Would you be concerned if the data was associated with your kids??

Enumerate your threat actors

Who are you worried about?

- Nation states? (Good luck)
- A competitor wanting to steal your design?
- Hacktivist looking to make a statement or cause chaos?
- Travis Goodspeed and a bottle of nitric acid?? (Again…good luck)

What are their motives?

- Espionage? Sabotage? Just curious how things work?
- Someone that wants to repurpose or bring new life to an old product?

What if the device is exploited?

How difficult is it to compromise the device?

- Team of highly trained security experts with millions of dollars? (High level of effort)
- A competitor with a small team of reverse engineers? (Moderate level of effort)
- Someone tinkering in their garage? (Low level of effort)

What is the impact if the device is compromised?

- Data leakage? Personal Safety? Internet wide botnet?

Who will be impacted?

- A town of people? An industry’s revenue stream? The Internet?

Identify your attack vectors

Where can an attacker submit input?

- Is user input added to commands you execute?
- Are you validating input or just blindly accepting it as truth?
- Are your services listening? If so, what interfaces are they configured to listen on?
- Are your applications and libraries up to date?

How does your device get data out?

- Are you initiating connections from the device to backend?
  - Web sockets are your friends
- Can your data be MitM?
- Are you using secure protocols? Is encryption bi-directional?

Case Study: Voting Machines

A voting machine performs the following functions:

- Accepts voting information and ensures one person, one vote
- Collects voting information and pushes to a centralized server
- Tally votes taken on the device
- Ensure voter information is taken anonymously

Case Study: Voting Machines

What expectations do we have from a voting machine?

- Ensure that votes are collected accurately
- The security of the device is sound and able to detect tampering
- Ensure data cannot be manipulated in transit
- Ensure that voter privacy is protected
- Ensure that backend systems are not susceptible to attacks or data manipulation
- Ensure that a vote is counted once and only once

Voting Machine Security Concerns

How could the voting machine be compromised?

- Hardware is tampered with, invalidating votes stealing an election
- Software is tampered with, invalidating votes or stealing an election
- Input fields used as an attack vector
- Communications intercepted and manipulated in transit
- Back end services are attacked changing voting results

Voting Machine Security Concerns

Who am I protecting my system from?

- Nation states, hacktivists and people wanting to change the outcome of an election

What are the trust boundaries

- The device should be trusted. We should easily be able to detect if someone tampers with the device.

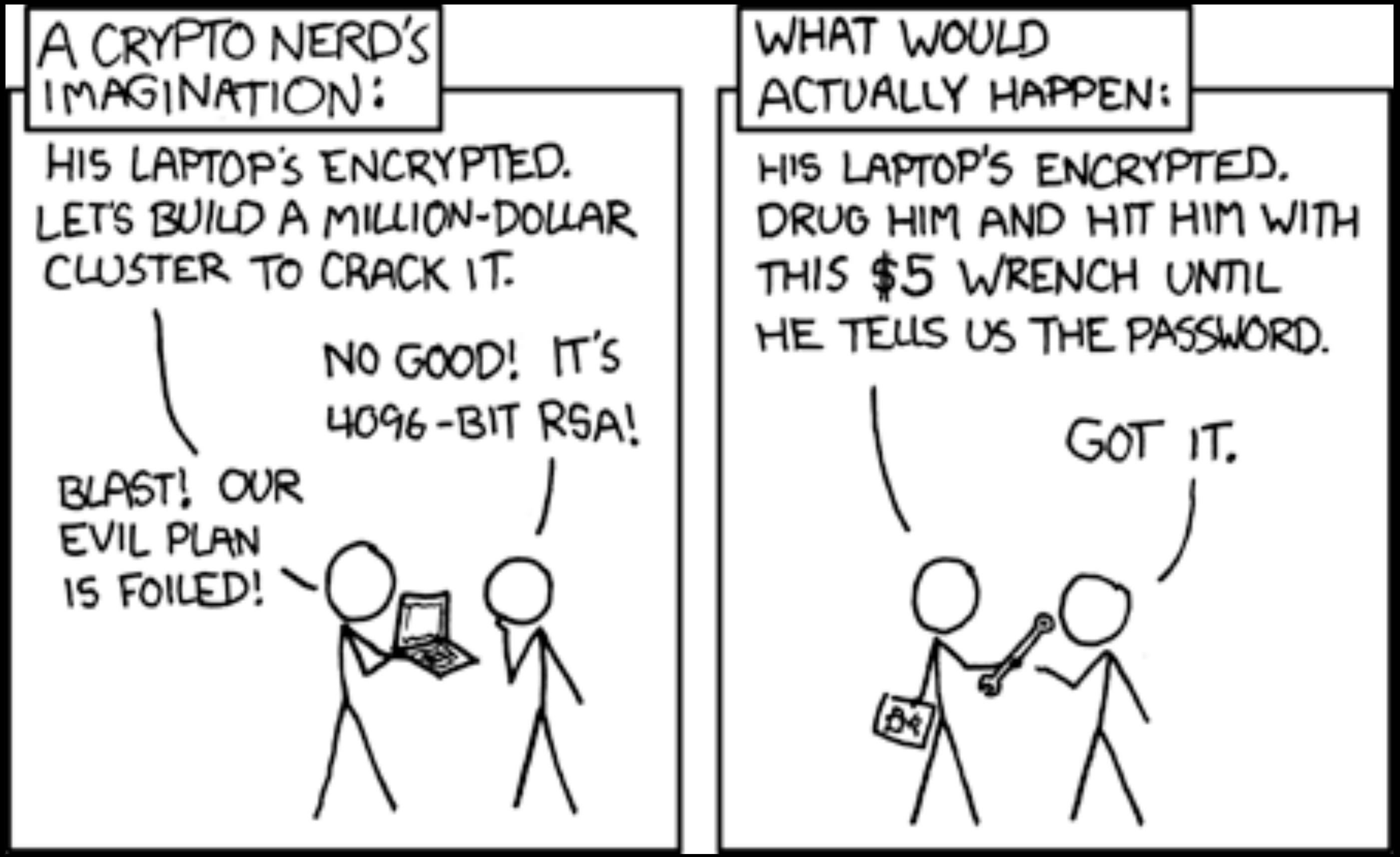
What are their motives?

- Sway an election that is favorable to an attacker

What is the impact?

- The will of the people is ignored and the outcome of an election goes to the attacker

Remember
Not everything can be protected


A CRYPTO NERD'S IMAGINATION:
HIS LAPTOP'S ENCRYPTED. LET'S BUILD A MILLION-DOLLAR CLUSTER TO CRACK IT.
NO GOOD! IT'S 4096-BIT RSA!
BLAST! OUR EVIL PLAN IS FOILED!
WHAT WOULD ACTUALLY HAPPEN:
HIS LAPTOP'S ENCRYPTED. DRUG HIM AND HIT HIM WITH THIS $5 WRENCH UNTIL HE TELLS US THE PASSWORD.
GOT IT.

https://xkcd.com/538/

Hardware Hardening

- Disable JTAG, SWD, SWI and all other debug interfaces in production units
  - Corollary, don’t even have the ports broken out on production devices, unless needed. It can simplify the board layouts and maybe get you smaller footprints
  - Disable UART, or at least the RX line and don’t print anything sensitive on the TX line
- If possible encrypt flash storage
  - Many devices support this natively. For instance the ESP32 implements it with just enabling a flag bit in firmware
  - Use per device encryption keys, don’t use a single encryption key for all devices
- Enable hardware Secure Boot if available.
  - Again, natively supported by many devices
- For sensitive devices implement tamper switches/traces/etc., can be combined with some battery backed SRAM and a device unique stored key to detect if a device has been opened
- Never underestimate an attacker’s creativity and always plan for someone smarter than you

Firmware Signing

- On device and FOTA updates need to be signed
  - Use at least 2048 bit RSA, preferably 4096 bit, or ECDSA key
- Leverage the hardware crypto modules for verification if possible, if not use a well known library/implementation. Don't roll your own!
- If combined with Secure Boot offers more protections
- Don't rely on encryption to "sign" updates or hide secrets in firmware, it will be broken
  - if your firmware can't be looked at without revealing secrets you're doing it wrong

Secure Firmware Updates

- Keep your dev/build environments up to date so your FOTAs get the latest fixes from upstream
- Use HTTPS targets for firmware downloads and validate server certs
  - Use cert pinning if you can
- Don’t allow firmware downgrades
  - Make use of eFuses/immutable storage changes to track firmware updates installed and have firmware refuse to run if the hardware expects a newer version
- All firmware updates must be signed
- Actually validate signature in your firmware install/update routines
  - have actually seen signed firmware that is not validated during install…

Firmware/Operating System Hardening

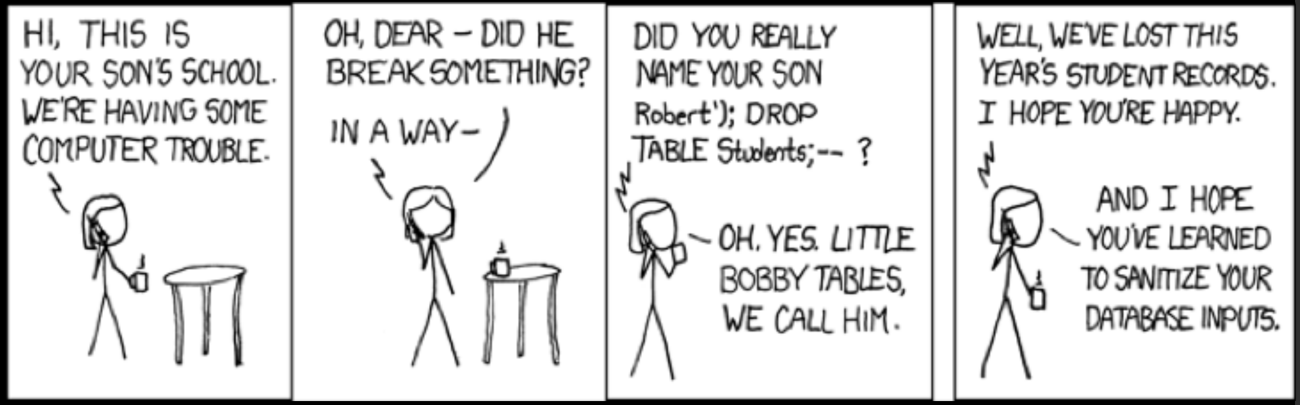
- Start with the newest baseline available, i.e. use the latest vendor SDK version or if there is a Linux 4.x base available for your platform don’t use 2.x
- Limit the installed components to a bare minimum
  - BusyBox does most of what you could need, just remove unneeded components in build
  - It will reduce attack surface, and may save on the BOM cost for flash
- For Linux bases, if possible, compile a custom kernel with only the features needed
- Don’t expose ports for services unless you need to
  - Also only expose them on the interfaces they need to be on, so don’t put admin portal on cellular if it only needs to be on WiFi
- Disable remote shells/logins. If not possible:
  - No TELNET… Ever… Like, seriously, never
  - Disable remote root logins
  - Require either a physical or authenticated remote trigger to enable the remoting, then disable after a timeout
  - Allow users to set their own passwords
- Moving services around to non-standard ports is not security, nmap sees all

Firmware/Operating System Hardening

- No “development” or “debugging” backdoors in release firmware, they will be found and exploited
- Build released firmware to release targets, not debug.
  - Use compiler optimizations and strip as many symbols as possible. Slows down RE attempts.
  - Put debug functions and logging in “ifdef”s that don’t get built in release builds
- Remove all debug and development tools from release builds
  - You don’t need netcat or gcc in release firmware

Device Application Hardening

- Standard rules apply, think about what you would do on a regular server
- Validate/sanitize all inputs, then do it again
- Don’t run as root unless absolutely necessary
- Expose the minimum of functionality to allow your device to fulfill its purpose
- Enable all the security options platform allows
  - Canaries, tagging/NX-bit, runtime bounds checks, etc.
- Authenticate and authorize all connections/requests to your app


HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

https://xkcd.com/327/

Transport Security

- TLS v1.2+ or DTLS (v1.2 preferred) if possible
  - Certificate pinning for (D)TLS connections to help prevent MITM
- For stateless UDP at least use AES with a per device key
  - If possible renegotiate the key on a regular basis
- Typically best to leverage the libraries/firmware from the SoC/chip vendor instead of adding your own
  - Most vendors will optimize the libraries to leverage any hardware acceleration or helpers
- If traffic cannot be encrypted at least have signed/verifiable data
  - Sensitive data should always be encrypted though

Backend Application Security

- See Device Application Hardening…
- For all requests ensure authentication/authorization is being done properly
  - Have seen multiple occasions where any authenticated user can read info for all devices if they know its ID
  - Have also seen cases where any auth'd user can manipulate another user's account by using their own session token and other accounts ID
- Use generic errors for all error responses
  - Don't have separate errors for incorrect username vs incorrect password
  - Don't have separate errors for device not found vs not authorized for device
  - For all returned errors, unless there is a valid reason, return the most generic error possible. HTTP/418 is a good choice ☺
- Only allow clients to connect using a supported level of encrypted connection
  - If TLSv1.2 is desired don't allow SSLv3
  - Restrict cipher suites allowed, don't allow NULL encryption TLS connections
- Follow OWASP/OASIS/W3C/NIST/other best practices and standards for all services

Lifecycle Management

- Clearly state intended support timeframe for released products
  - Ideally allow for 1-2 years after last sale to release updates, if not possible at least clearly state EOL/end of support
- Have a clear update cycle
  - Release updates only on regular cycle, unless there is an emergency patch required
  - Can be monthly/quarterly/yearly/whatever, but make it known and stick to it
- Start to refuse connections to devices with software older than X versions back or that have severe known issues after a sufficient warning period
- Give devices auto-update capabilities, and enable by default for low risk devices
- Have associated mobile apps or web pages bug the users to update to new firmware
  - Good place to message when device will be cut off for out of date firmware
  - If possible provide a single click option to kick off updates in the background
- If targeting enterprise/industrial/government provide a detailed changelog, probably good for regular consumer devices also

Voting Machine Security Control Considerations

- Hardware Hardening
  - Disable JTAG and other unused I/O, epoxy flash, tamper circuitry, Trusted/Secure Boot
- Software and OS security
  - Leverage signed firmware, encrypted storage, mandatory access controls, hardened libraries
  - Input should avoid free form fields
- Secure communications
  - Bi-directional TLS with certificate pinning, per session keys
- Secure Backend
  - Full disk encrypted storage, strong AAA, detailed auditing of events
- Lifecycle management
  - Defined change management and review
  - Defined firmware update requirements/schedule
  - Strong vulnerability scanning/monitoring and patch management, for both device and backend
- Bug bounty
  - Work with community and even provide devices for researchers/pentesters
- Other
  - Consider paper trails that humans can ready and verify results

Questions??