Handling Top Security Threats for Connected Embedded Devices

OpenIoT Summit, San Diego, 2016
Jeep Cherokee hacked in July 2015

- Presented at Black Hat USA 2015
  - Charlie Miller
  - Chris Valasek

- Remote exploit giving full control of the car

- Clearly demonstrates physical safety risk

- No way to fix remotely

- 1.4 million cars recalled
About me

- **Eystein Stenberg**
  - CS/Crypto master’s
  - 7 years in systems, security management
  - eystein@mender.io

- **Mender.io**
  - Over-the-air updater project for Linux/Yocto
  - Under active development
  - Open Source

- Reach me after on email or exhibitor hall
Jeep Cherokee Head Unit with Wifi

- Cherokee customers can buy wifi subscription as an add-on (~$40/month)
- Connect devices in the car to the car’s wifi to get online (phones, tablets, ...)
- Wifi is password protected
Wifi-based Breach: Short-range

- Wifi password based on system time after provisioning
- January 01 2013 00:00 GMT +- 1 minute
- Multimedia system breached due to software vulnerability
- Scope: Control music player/radio/volume and track GPS coordinates when within wifi range
Cellular-based Breach: Country-wide

- Scope: Control music player/radio/volume and track GPS coordinates countrywide
- Can also select a specific Jeep based on its GPS-coordinates
The Controller Area Network (CAN) bus

- The CAN bus connects ~70 electronic control units (ECUs), including engine control, transmission, airbags, braking.
- V850 chip is designed to only read from the CAN bus, to isolate components.
Unauthorized update to write to the CAN bus

- The head unit can **update the firmware** of the V850
- Firmware **update authenticity not checked** properly
Putting it together

Lessons

- Wifi hotspot password was predictable
- Remotely accessible service (in head unit) was vulnerable (and not updated)
- Firmware update (for V850) did not have proper authenticity checks
- The only way to fix the vulnerabilities is through a manual update (by customer or dealership)
Security basics protects against most attacks

- Devices more software-defined & connected (IoT)
- Consider which types of attacks your products can be vulnerable to
- Reduce risk of your products being compromised
Session overview

- Our goals
- Anatomy of an attack
- Security strategies
- The patching problem
- Strategies for patching embedded devices
Attacker motivation

- Why would someone attack your product?
- Can someone *make money* from a compromise? How much?
- All crime starts with a *motive*
Your goal is to lower attacker Return on Investment

● It is always possible to compromise

● Lower Return on Investment (ROI) for attacker
  ○ Decrease value of successful attack
  ○ Increase cost of successful attack

● Focus on increasing cost of attack in this session
Decreasing value of attack can be effective too
<table>
<thead>
<tr>
<th>Action</th>
<th>Desired outcome</th>
</tr>
</thead>
<tbody>
<tr>
<td>Reconnaissance</td>
<td>→ Discover vulnerabilities</td>
</tr>
<tr>
<td>Intrusion</td>
<td>→ Initial access</td>
</tr>
<tr>
<td>Insert backdoor</td>
<td>→ Ongoing access</td>
</tr>
<tr>
<td>Clean up</td>
<td>→ Avoid detection</td>
</tr>
</tbody>
</table>
Remote-access services helping attacker with 1 & 2

1. Port scan to discover
   ○ Which services are running
   ○ Versions

2. Match versions against known vulnerabilities/defaults

Pro tip: disable remote-access services that you don’t need
Devices are also susceptible to physical attacks

- In potentially hostile environments
- Device firmware/software can likely be analyzed
  - Access to device storage
- Compromise one device = compromise all?
- Pro tip: *Don’t share secrets (keys, passwords) between devices*
DNS Amplification attack (DDoS) enabled by IoT

Attacker

Compromised Devices: “Amplifiers”

Spoofed DNS requests: origin=victim

Open DNS servers

DNS responses

Victim’s server/network
Security design principles

● Principle of least privilege
  ○ Does component need to run as root?

● Separation of privilege
  ○ Does component need access to network?
  ○ Does it also need access to sensitive data?
  ○ Pro tip: *Don’t share keys between devices*
  ○ Pro tip: *Network segmentation* (between customers/internal/...)

● Kerckhoff’s principle
  ○ Assume only keys are secret, not designs/algorithms
  ○ Pro tip: *Only rely on industry-standard communication/crypto protocols*
Security patching is done too late

Cumulative Probability of Exploitation

5-10 days: <10% probability it is exploited

60 days: >90% probability it is exploited

110 days: remediation time avg.

Source: How the Rise in Non-Targeted Attacks Has Widened the Remediation Gap, Kenna Security
Why security patching happens too late

- Invisible until too late

- Too costly/risky
  - Manual
  - Requires downtime of production?
  - Risk of breaking production?

- Politics

- How often do you patch?
  - Do you have a way to do it? A process?
  - Often not a core competence and not a priority to develop updater
Patching connected devices is harder

- **No/expensive physical access**
  - Need failure management

- **Unreliable power**
  - What if power disappears in the middle of patching?

- **Unreliable (wireless) network connectivity**
  - Handle partial downloads
  - Ideally resume downloads in expensive networks like 3G

- **Public and insecure (wireless) networks**
  - Can someone inject arbitrary code during the update process?
  - Verify authenticity of update
Embedded client patching process overview

- Detect update
  - Poll server
- Download
  - Checksum
- Integrity
  - Signature verification
  - May not need this
- Authenticate
  - Important, but not trivial
- Install
  - Extract
  - Decrypt
- Roll back?
  - Important, but not trivial
<table>
<thead>
<tr>
<th></th>
<th>Full image</th>
<th>Package (opkg, …)</th>
<th>tar.gz</th>
<th>Docker/Containers</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Download size</strong></td>
<td>Large*</td>
<td>Small</td>
<td>Small</td>
<td>Medium</td>
</tr>
<tr>
<td><strong>Installation time</strong></td>
<td>Long*</td>
<td>Short</td>
<td>Short</td>
<td>Short</td>
</tr>
<tr>
<td><strong>Rollback</strong></td>
<td>Yes (dual partition)</td>
<td>Hard</td>
<td>Hard</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Consistency</strong></td>
<td>Yes</td>
<td>Medium</td>
<td>Hard</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>Design impact</strong></td>
<td>Bootloader, Partition layout</td>
<td>Package manager</td>
<td>tar, …</td>
<td>Kernel, docker</td>
</tr>
</tbody>
</table>

* Can mitigate with compression or binary diffs
Strategies to reduce the risk of bricking

- **Integrity checking**
  - This must be done
  - Easy to implement

- **Rollback support**
  - This should be a requirement: power loss, installation error, etc.
  - Could be hard depending on update type (tarball, package)

- **Phased rollout**
  - I.e. don’t deploy update to all devices in one go
  - Most do this to some extent: test & production environments
  - Can be more granular on device population (1%, 10%, 25%, 50%, ...)
  - Used in most large infrastructures
Security basics protects against most attacks

- Segment the network
- Disable remote-access services
- Don’t share keys between devices
- Patch software vulnerabilities quickly and securely
- “80-90% of security breaches can be prevented”
  - Center of Internet Security
Many possible ways in, but let’s fix the obvious