Safety Vs Security
A tale of two updates

Jérémy Rosen
This talk is about Philosophy and culture

I will talk mainly about Industrial embedded systems.

All projects are different. No project have all the constraints

My definitions

- **Safety**: Anything related to reliability
- **Security**: Anything related to hostile takeover

We will discuss “Why embedded systems suck at security”

But just a small part.
Show of hands: who's who

People with both hats

Safety people

Security people
Show of hands: who's who

Safety people
Show of hands: who's who

- People with both hats
- Safety people
- Security people
Show of hands: who's who

People with both hats
Safety is here to ensure that the system “always works as expected”

- Correct is not enough. You need to prove it.
  - Software
  - Hardware
  - Tools (compilers)

- No dynamic memory allocation
- Proofreading the generated Assembly code

- It is easier to prove that a bug has no consequence than to prove that a fix is correct
- Any change is a safety change
- All assumptions must be documented and checked at every level.

Safety people are paranoid freaks

But our planes and trains are incredibly safe.

---

1. Machine learning is going to be... interesting
The security brainwashing

Security is here to ensure that the system “can’t be used out of its purpose”

- Everything is an attack vector
- Any little hole is potentially a leap-frog to a whole exploit
- Security is a race
  - Find the weakness before the malevolent
  - Find a fix as fast as possible, temporary breakages are OK.
  - Deploy as fast as possible.
  - Embargoes are OK.
- The whole world is out to get you

Security people are paranoid freaks

But attacks are a real thing
and the security culture has measurable results
Safety and security: Compared cultures

**Safety**
- Code must be proven and certified

**Security**
- Must react quickly to attack
Safety and security: Compared cultures

Safety
- Code must be proven and certified
- Usage range is clearly defined

Security
- Must react quickly to attack
- Must protect from hostile behaviours

Safety trumps everything else
Security trumps everything else
Safety and security: Compared cultures

**Safety**
- Code must be proven and certified
- Usage range is clearly defined
- Bug likeliness goes down with time

**Security**
- Must react quickly to attack
- Must protect from hostile behaviours
- Threat models evolve and adapt

Safety trumps everything else

Security trumps everything else
<table>
<thead>
<tr>
<th>Safety</th>
<th>Security</th>
</tr>
</thead>
<tbody>
<tr>
<td>- Code must be proven and certified</td>
<td>- Must react quickly to attack</td>
</tr>
<tr>
<td>- Usage range is clearly defined</td>
<td>- Must protect from hostile behaviours</td>
</tr>
<tr>
<td>- Bug likeliness goes down with time</td>
<td>- Threat models evolve and adapt</td>
</tr>
<tr>
<td>- A known bug with no consequence should be ignored</td>
<td>- All bugs are potential exploits and must be fixed</td>
</tr>
</tbody>
</table>

Safety trumps everything else

Security trumps everything else
Safety and security: Compared cultures

Safety
- Code must be proven and certified
- Usage range is clearly defined
- Bug likeliness goes down with time
- A known bug with no consequence should be ignored
- Upgrade only as a last resort

Security
- Must react quickly to attack
- Must protect from hostile behaviours
- Threat models evolve and adapt
- All bugs are potential exploits and must be fixed
- Always use the latest version

Safety trumps everything else
Security trumps everything else
Safety and security: Compared cultures

Safety

- Code must be proven and certified
- Usage range is clearly defined
- Bug likeliness goes down with time
- A known bug with no consequence should be ignored
- Upgrade only as a last resort

Any change is a risk and needs to be justified

Security

- Must react quickly to attack
- Must protect from hostile behaviours
- Threat models evolve and adapt
- All bugs are potential exploits and must be fixed
- Always use the latest version

Any bug is a potential security weakness and needs to be fixed
Safety and security: Compared cultures

**Safety**
- Code must be proven and certified
- Usage range is clearly defined
- Bug likeliness goes down with time
- A known bug with no consequence should be ignored
- Upgrade only as a last resort

Any change is a risk and needs to be justified

**Security**
- Must react quickly to attack
- Must protect from hostile behaviours
- Threat models evolve and adapt
- All bugs are potential exploits and must be fixed
- Always use the latest version

Any bug is a potential security weakness and needs to be fixed

Safety trumps everything else

Security trumps everything else
The particularities of embedded system: Upgrades

System upgrades in the embedded world

- Upgrades must be robust and deal with failures on their own
  - No access to the product
  - Bad blocks
  - Conflicting configuration files
  - Invalid user configuration
  - Kernels need to be upgraded too...
The particularities of embedded system: Upgrades

System upgrades in the embedded world

- Upgrades must be robust and deal with failures on their own
  - No access to the product
  - Bad blocks
  - Conflicting configuration files
  - Invalid user configuration
  - Kernels need to be upgraded too...

- Some systems can't stop.
The particularities of embedded system: Upgrades

System upgrades in the embedded world

- Upgrades must be robust and deal with failures on their own
  - No access to the product
  - Bad blocks
  - Conflicting configuration files
  - Invalid user configuration
  - Kernels need to be upgraded too...

- Some systems can't stop.
- Old hardware can't be phased out
The particularities of embedded system: Upgrades

System upgrades in the embedded world
- Upgrades must be robust and deal with failures on their own
  - No access to the product
  - Bad blocks
  - Conflicting configuration files
  - Invalid user configuration
  - Kernels need to be upgraded too...
- Some systems can’t stop.
- Old hardware can’t be phased out
- Deployment time is controlled by the user

You can’t trust your subcontractors to survive
You can’t trust your technologies to survive
You can’t trust your engineers to survive
System upgrades in the embedded world

- Upgrades must be robust and deal with failures on their own
  - No access to the product
  - Bad blocks
  - Conflicting configuration files
  - Invalid user configuration
  - Kernels need to be upgraded too...

- Some systems can’t stop.
- Old hardware can’t be phased out
- Deployment time is controlled by the user
- (Very) long term support
  - You can’t trust your subcontractors to survive
  - You can’t trust your technologies to survive
  - You can’t trust your engineers to survive
The particularities of embedded system: Security

Security in the embedded world

- Physical access can't be restricted.
  - Secure boot is a requirement
  - Each product must have a unique key at factory-time
  - You might need a unique image per product.
Security in the embedded world

- Physical access can't be restricted.
  - Secure boot is a requirement
  - Each product must have a unique key at factory-time
  - You might need a unique image per product.

- No return to a trusted state
  - Bootloader attacks are a thing
  - JTAG attacks are a thing
  - ROM are expensive
The particularities of embedded system: Security

Security in the embedded world

- Physical access can't be restricted.
  - Secure boot is a requirement
  - Each product must have a unique key at factory-time
  - You might need a unique image per product.

- No return to a trusted state
  - Bootloader attacks are a thing
  - JTAG attacks are a thing
  - ROM are expensive

- No upgrade culture
  - *Ship and forget* philosophy (hardware makers)
  - No long term maintenance team (startup culture)
The particularities of embedded system: Security

Security in the embedded world

- Physical access can't be restricted.
  - Secure boot is a requirement
  - Each product must have a unique key at factory-time
  - You might need a unique image per product.

- No return to a trusted state
  - Bootloader attacks are a thing
  - JTAG attacks are a thing
  - ROM are expensive

- No upgrade culture
  - *Ship and forget* philosophy (hardware makers)
  - No long term maintenance team (startup culture)

You have to choose...
Security in the embedded world

- Physical access can't be restricted.
  - Secure boot is a requirement
  - Each product must have a unique key at factory-time
  - You might need a unique image per product.

- No return to a trusted state
  - Bootloader attacks are a thing
  - JTAG attacks are a thing
  - ROM are expensive

- No upgrade culture
  - *Ship and forget* philosophy (hardware makers)
  - No long term maintenance team (startup culture)

You have to choose…
Bricked or Pwned?
How often should you publish security updates?

- “As needed” is not realistic
  - **Android**  Monthly security updates
  - **Windows** Monthly security updates
    - **Linux**  Variable, but usually a rolling release. (Debian: automated daily updates)
    - **iOS**  As needed (monthly)
- Monthly seems to be the current best-practice
How often should you publish security updates?

- “As needed” is not realistic
  - Android: Monthly security updates
  - Windows: Monthly security updates
  - Linux: Variable, but usually a rolling release. (Debian: automated daily updates)
  - iOS: As needed (monthly)
- Monthly seems to be the current best-practice

Yes but... It takes more than a month to re-certify
How often should you publish security updates?

- “As needed” is not realistic
  - Android  Monthly security updates
  - Windows  Monthly security updates
  - Linux    Variable, but usually a rolling release. (Debian: automated daily updates)
  - iOS      As needed (monthly)

- Monthly seems to be the current best-practice

  Yes but... It takes more than a month to re-certify

  Yes but... What about our vulnerability window?
So...To summarize the problems

Both sides have very strict process requirements
- That are justified by years of good practices
- That need to be strictly followed to be effective
- That are effective at what they are meant to do

Those requirements are completely opposite
- Speed critical Vs Confidence critical
- Proactive Vs Reactive
- Preventive Vs Proven

It is impossible to reconcile both sides.

Let's look at ways to mitigate the problem.
So... To summarize the problems

Both sides have very strict process requirements
- That are justified by years of good practices
- That need to be strictly followed to be effective
- That are effective at what they are meant to do

Those requirements are completely opposite
- Speed critical Vs Confidence critical
- Proactive Vs Reactive
- Preventive Vs Proven
So...To summarize the problems

Both sides have very strict process requirements
- That are justified by years of good practices
- That need to be strictly followed to be effective
- That are effective at what they are meant to do

Those requirements are completely opposite
- Speed critical Vs Confidence critical
- Proactive Vs Reactive
- Preventive Vs Proven

It is impossible to reconcile both sides.
Let's look at ways to mitigate the problem.
How to mitigate that problem

You can't completely solve the problem... But you can mitigate

Avoid the problem entirely

- Not all products are safety critical, but all products need to care about security.
- You still need a robust upgrade system
How to mitigate that problem

You can't completely solve the problem...But you can mitigate

Avoid the problem entirely
- Not all products are safety critical, but all product need to care about security.
- You still need a robust upgrade system

Accelerate re-certification
- Automated testing should be part of the certification.
- Have a fast-path in your re-certification process.
- Minimize the safety critical perimeter and update it separately
How to mitigate that problem

You can’t completely solve the problem…But you can mitigate

Avoid the problem entirely
- Not all products are safety critical, but all products need to care about security.
- You still need a robust upgrade system

Accelerate re-certification
- Automated testing should be part of the certification.
- Have a fast-path in your re-certification process.
- Minimize the safety critical perimeter and update it separately

Separate safety and security
- Containers
- Hypervisors
- Hardware separation
How to mitigate that problem

You can't completely solve the problem... But you can mitigate

Avoid the problem entirely

- Not all products are safety critical, but all product need to care about security.
- You still need a robust upgrade system

Accelerate re-certification

- Automated testing should be part of the certification.
- Have a fast-path in your re-certification process.
- Minimize the safety critical perimeter and update it separately

Separate safety and security

- Containers
- Hypervisors
- Hardware separation

Plan for security updates

- Include an update agenda in your maintenance process
- Plan an End of Life for your products and document it
Thank you!
Questions?