Ostro OS security architecture

An IoT OS security architecture that is so boring that you can sleep soundly at night

Ismo Puustinen, Intel Finland
What is Ostro OS?

- https://ostroproject.org
- IoT operating system, based on Yocto project
- Suitable for devices of various sizes
  - Current HW targets: Galileo 2, Edison, NUC
- Not meant to be an end-user operating system
  - Ostro Project offers only pre-built development images
  - Ostro OS customers typically create IoT devices (ODMs, ...)
- Rolling releases
Ostro OS security goals

✧ Scalable security
  ✧ Customers can decide which protection mechanisms to use

✧ Let customers to focus on things that add value
  ✧ Applications, cloud integration, ...

✧ Try to keep the end-user devices up-to-date
  ✧ Make it as painless as possible for customers to provide timely updates

✧ Secure against network threats

✧ Vulnerability mitigation
System updates

- System updates are pushed to end user devices using Clear Linux software update mechanism
  - Stateless
  - The devices with the same "release number" are guaranteed to have the same versions of software – only configuration differs
Systemd

- Ostro OS security is heavily based on systemd

- System services
  - Removed all non-necessary privileges
  - Only system update service can write to root fs
  - Run as non-root if possible (ambient capabilities in systemd 229)
  - Permission checks based on Unix group membership

- Applications
  - Service files generated from application manifests
  - DAC or container technologies used to separate applications
  - Not complete separation due to the nature of DAC, use containers if needed
Firewall

✧ Restrictive default firewall rules (IPv4 and IPv6)
  ✧ Iptables
  ✧ Services and applications need to open holes for themselves

✧ In the future go over to nftables
  ✧ Declarative - services can drop firewall configuration files
Secure boot

- UEFI secure boot – optional
  - Protection against both offline and online attacks
  - Kernel, initramfs and kernel command line in one signed UEFI blob
  - IMA/EVM initialized from initramfs

- IMA hashes file content and stores the hash in security ima xattr with the file
  - Possible to sign the hash using a secret key (image build time)
  - Kernel contains CA with the public key → file content is secure

- EVM helps protect against offline attacks against the xattrs
  - The xattrs are signed in security evm with inode number (to prevent copying xattrs from one inode to another)
  - Not possible to calculate EVM hashes offline, thus need to be signed using TPM
Mandatory Access Control

- Smack – optional
  - More fine-grained permission handling than DAC
  - Three-domain model (System, User, _) inherited from Tizen
**Filesystem layout**

<table>
<thead>
<tr>
<th>Directory</th>
<th>Description</th>
<th>Protection Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>/</td>
<td>Conceptually read-only</td>
<td>All services (except software update) will use systemd’s ProtectSystem=full to make root fs appear read-only</td>
</tr>
<tr>
<td>/var</td>
<td>Persistent data</td>
<td>Kernel creates IMA/EVM creates hashes on-the-fly to provide some protection</td>
</tr>
<tr>
<td>/tmp and /var/run</td>
<td>tmpfs</td>
<td>Deleted every shutdown</td>
</tr>
<tr>
<td>/home</td>
<td>Persistent data</td>
<td>No IMA/EVM. Every application has its own home directory</td>
</tr>
<tr>
<td>/etc</td>
<td>Persistent data</td>
<td>Ostro OS is (will be) stateless - /etc is empty before first boot. IMA/EVM hashing like /var.</td>
</tr>
</tbody>
</table>