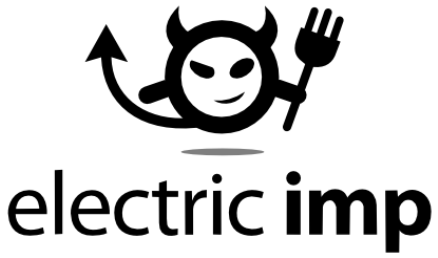

Electric Imp has sold over half a million 'Imps', all remotely upgradable and embedded in a variety of consumer and industrial devices.

An Imp consists of a microcontroller-grade processor and wifi radio. The processor hosts a virtual machine for the Squirrel scripting language, and arranges secure connectivity back to a cloud service. The cloud service runs an 'Agent' for the Imp -- which is a further Squirrel virtual machine that can communicate with the Imp and the Internet in general. Having an Agent allows the code running on the Imp to be suitably limited in scope so that it runs comfortably on a microcontroller.

The customer-provided code which runs in the Imp and Agent VM is instantly upgradable over-the-air. Because the VMs provide memory safety, bugs in the code they run do not prevent future changes. The platform further reports errors to the Agent, so local access is not needed during the software development process.

Underneath this we perform over-the-air, secure, crash-safe upgrades of the platform firmware running on the Imp. We produce and deploy these upgrades, meaning device manufacturers do not need security expertise or ongoing engineering effort to keep their devices secure and up-to-date over their lifespan. This aligns well with the economic interests and skills of device manufacturers. It also means our efforts to maintain Imp security are efficiently amortised across all devices, irrespective of application, market or device cost.

We'd like to see, initially, a common language for talking about the properties of embedded upgrade systems such as cryptographic security, device ownership, device separation, vulnerability reporting and management, and device lifespan.

Platform Upgrades

The first Imps, the imp001 and imp002, both have a STM32F205 microcontroller with 1MB of flash. This flash is split into ‘preboot’, ‘bootrom’, ‘improm’ and ‘bytecode’ sections, approximately 16KB, 192KB, 640KB and 128KB in size respectively.

The preboot is immutable and merely jumps to the improm if it is valid or to the bootrom otherwise. The bootrom does not have any functionality save the ability to upgrade the improm over the air. The improm contains the bytecode interpreter, hardware APIs, a TLS stack, etc. as well as the ability to upgrade the bootrom over the air.

Upgrade images are encrypted with AES (with a per-Imp-model key) and signed with RSA4096-PSS-SHA256. We find a 4096-bit public RSA operation to be an acceptable computational cost at ~44ms to ~180ms across our hardware models when considered alongside 1-3 second flash erase times that an upgrade also involves. Upgrade images are delivered over HTTP for local cacheability and bootrom size reasons; this unfortunately means our upgrade process doesn't keep some information private such as which devices are running which versions.

Upgrades are triggered by an upgrade request or demand delivered over the TLS-secured link between the improm and our cloud service. If the server requires a improm upgrade, the improm invalidates itself in flash, and then performs a hardware reset. The bootrom then runs and upgrades the improm. The very last operation performed in an upgrade is marking the new image as valid, so any failure up to that point does not leave an inconsistent state.

We control when a device receives a particular upgrade, but it typically involves discussion with the device manufacturer. The author of the bytecode running on the device can provide code that chooses to defer upgrade requests (but not demands) to a more suitable time. This capability was added for a device used only during live sporting events -- it is crucial that an upgrade is never performed during such an event.

Some devices are battery powered and this brings its own problems. We cannot upgrade devices with insufficient voltage to erase and rewrite their flash, so we check first and avoid doing so. Making a network connection requires a similar (slightly lower) voltage, so unupgradable devices soon become unable to connect and will have their batteries replaced or recharged.

Platform hardware requirements

Our least powerful hardware (as mentioned above) has 1MB of flash, 128KB of SRAM and a Cortex-M3 core running at 120MHz. Wifi is provided by a separate chip that has a further Cortex-M3; this code is provided by the vendor and loaded into the wifi device on demand. The bootrom and improm each have their own copy of this binary firmware blob, so these are included in our platform upgrade image.

Later Imp models have a requirement for an external SPI flash to offer more user bytecode storage, and accommodate larger platform and wifi firmware binaries.

We do not require a MMU/MPU, but in practice all of our hardware platforms have an MPU and we use this to enforce safety and security policies like W^X.

User Code

User code is written in Squirrel, compiled by the cloud service and delivered as bytecode over TLS to the improm. This is written to and interpreted from flash. Bytecode updates are under the control of the developer or device manufacturer; we are not involved operationally.

The Squirrel interpreter is a stack-based, memory-safe, garbage-collected virtual machine. We have not made major modifications to it, save for making the bytecode format more compact.

Notes: common requirements for IoT upgrade

I'd like to see a basic set of requirements for IoT device upgrade systems. I'm not confident that the field is coherent and general enough to get to a position like we have with network transport security, where one can say "just use TLS." I'd be happy if there were some document (perhaps in the format of RFC6888) which covered things like the following:

Security

These ideas are aimed at alleviating common security failure modes of embedded upgrade systems.

- Strong, modern cryptography to provide upgrade authenticity.
- No device contains secrets usable to compromise another (i.e., avoiding 'class breaks').
- Authenticity covers both binaries and all metadata used in processing an upgrade.
- Upgrade processed in one pass if read from untrusted media (i.e. avoiding TOCTOU bugs).

Device ownership and lifespan

Devices should be updated by an entity which has an economic interest in keeping them updated and secure, and for a suitable period. Customers may demand escrow agreements in the event that this entity ceases to exist, as is common in commercial software sales.

Vulnerability reporting and management

This could merely be a referential restatement of ISO29147.

Author: Joseph Birr-Pixton <joseph@electricimp.com>

More information

https://electricimp.com/docs/hardware/imp/modules/ - Hardware comparison.
https://electricimp.com/docs/hardware/imp/datasheets/ - Datasheets.