RC25095 (W1101-013) January 7, 2011
Computer Science

IBM Research Report

Friends in Low Places – Loading Firmware in the Field

Elaine Palmer
IBM Research Division
Thomas J. Watson Research Center
P.O. Box 218
Yorktown Heights, NY 10598

Tamas Visegrady
IBM Research Division
Zurich Research Laboratory
8803 Rüschlikon
Switzerland

Research Division
Almaden - Austin - Beijing - Cambridge - Haifa - India - T. J. Watson - Tokyo - Zurich

Friends in Low Places – Loading Firmware in the Field

Elaine Palmer and Tamas Visegrady, IBM Research Division

1 Introduction

A country song made famous by Garth Brooks in 1990 declares, "I got friends in low places," noting that one can always rely on ordinary people to help a friend in need. BIOS software is the friend in the "low places" of clients and servers. It is software on which these systems rely to verify the soundness of the hardware and to transfer control to subsequent software. It has full access to the resources of a system, including memory, processors, coprocessors, and fans. What, then, if this software were to become irreparably modified, whether by mistake or malice? This paper addresses the problem of reliably updating such firmware in the field, after a device has left the secure confines of a manufacturing facility.

2 The Device That Tested our Mettle

For many years, IBM has produced hardware security modules, also known as secure coprocessors, which are small, general-purpose systems inside a tamper-responding enclosure. In 1998 IBM's 4758 hardware security module and its firmware earned the first ever FIPS 140-1[1] overall level 4[1] (Certificate #35). That firmware, known inside IBM as "Miniboot," works in conjunction with its low-level friend POST (Power On Self Test). In addition to typical BIOS-like functions, Miniboot is responsible for verifying the state of security inside the secure enclosure. Miniboot is also responsible for handling software updates (more on that later). On successful inspection of the system, Miniboot transfers control to the embedded operating system and its applications, and does not run again until the secure coprocessor is reset. As required by FIPS 140-1 and -2 level 4, Miniboot was formally modeled and verified, rigorously tested, and inspected. To date, the design has stood the test of time, with only minor tweaks despite two major redesigns of the underlying hardware. (The current generation is the IBM 4765.)

3 Problems With Field Updates of Firmware

The security requirements of the device pose a difficult problem – how to securely update the firmware (and operating system and applications) in the field.

3.1 No hardware updates

During manufacture, after initial tests are completed, the system is wrapped in a tamper-sensing membrane and potted in a hard resin. At that point, the secure hardware can never be updated (except for replacing an external battery), without triggering a tamper event and permanently disabling the device. In that event, it becomes an expensive (but very secure) doorstop.

[1] Level 4 means that the device withstands all known logical and physical attacks without revealing the secrets (such as cryptographic keys) stored within.

3.2 No Configuration Tracking Back at the Ranch, No Trust in the Field

Keeping track of which software is in which device is out of the question, for example by keeping huge databases or affixing unique part numbers for each software configuration. The data and software inside the device vary by customer and by application, and can be updated frequently. To complicate matters, the host system in which the coprocessor is installed is not trusted, and the user is a potential adversary.

3.3 Multiple Authorities

There are multiple parties authorized to update the embedded software. IBM is solely responsible for the Miniboot firmware. However, any one or a combination of parties (IBM, OEM's, or customers) may furnish the operating system and the applications that run inside the secure device. Supporting this flexibility increases the complexity of the field update problem, but it decreases manufacturing and support costs by allowing one general-purpose device to be programmed and updated by multiple authorities for widely different security applications.

3.4 Inbound Updates, Outbound Authentication

From the time an IBM secure coprocessor leaves the factory, it must defend itself against malicious or mistaken firmware updates (inbound), while still permitting and applying field updates from properly authenticated parties. It must also prove to the outside world that it is an authentic IBM secure coprocessor (not a software clone) running a specific code stack (outbound authentication).

3.5 No Backdoors

Because of the extreme sensitivity of the applications that use the IBM secure coprocessor, no secret mechanisms to update the software or revive the device are permitted, even in the case of accidental tampers.

4 Initialization – The Origin of the Universe

How can a secure device be initialized with its very first credentials? There are at least three common ways: 1) imprinting, 2) installing temporary transport keys and initializing later, and 3) establishing permanent keys during manufacture.

4.1 Imprinting in the Field

Imprinting allows the first initializer of a device to establish its identity and its membership in an organization (or security domain). "Welcome to the world, device. I see you call yourself ABC, and now you are officially part of the XYZ power grid." The device may create its own initial keys, or they can be generated externally and injected. Those keys must be certified, lest the device be indistinguishable from other devices that are outside the security domain. Further attempts to imprint the device are either allowed (after wiping all secrets) or forbidden (by blowing a fuse or setting an unmodifiable bit). It is important that the imprinting process itself and the chain of custody of the devices be secured, for example, with trusted couriers. If not, then evildoers can create software clones, or imprint stolen hardware for nefarious purposes. In our case, these risks are unacceptable, but may be perfectly acceptable in other applications.

4.2 Temporary Transport Keys

Similar problems exist in the world of smart cards and mobile phone SIMs, where thousands of very low cost devices are manufactured, initialized (with a set of applications), and personalized (with a cardholder's information). Typically, chips are initialized during manufacture with a secret transport key that is common across a large batch of chips. It is known only to the manufacturer and the next organization to process the chips. When the number of organizations is relatively small and their identities are known in advance, such a scheme is feasible. Initialization and personalization are irreversible. In the case of JavaCards, some organizations allow the update and addition of applets in the field, but the initialization and personalization values remain constant once established. In the case of IBM's secure coprocessors, temporary transport keys are not an option, because at the time of manufacture, the identity of the "next" organization is not known.

4.3 Permanent Keys during Manufacture

IBM's secure coprocessors generate their own secret keys and either generate or import critical configuration parameters, such as trusted certificates. This operation takes place during a once-in-a-lifetime (of the device) initialization step, at the last stage of manufacturing, after the module has been encapsulated in its tamper-responding enclosure. At that time, Miniboot generates its own first keypair. IBM certifies the public key for that device, along with its own unique hardware id. The certificate attests that this unique, untampered device is the entity that knows the matching private key, and that it is running a specific version of Miniboot software.

5 Highlights of our Solution

This section describes features of IBM secure coprocessors that are related to firmware and firmware updates. A detailed description can be found online in [2].

5.1 Physical Separation and Decreasing Levels of Trust

Immediately after reset, POST and Miniboot run, then Miniboot transfers control to the operating system, which then invokes applications. Figure 1 shows the four layers of software that run inside the secure coprocessor.

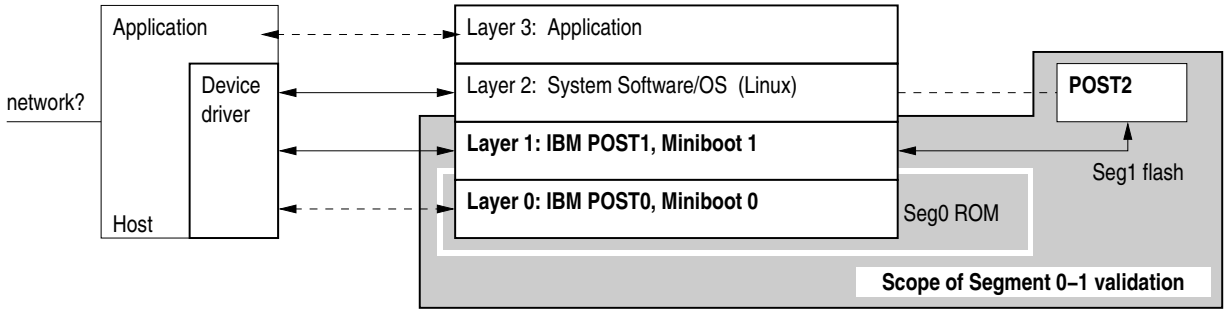

Figure 1 - Software Architecture

Miniboot is the most trusted software in the system and runs in a known state right after reset (a convenient assumption for formal proofs). Just as in conventional

systems, as more and more software is invoked, there is less and less trust that the software can maintain the secure state of the device. In typical computing environments, the operating system is trusted and the application is not. However, in the secure coprocessor, the all-powerful OS is not trusted.

In the original 4758 device, all software shares the same processor and address space. Therefore, a hardware lock microcontroller protects reserved segments of memory after each layer of firmware is completed. Higher layers cannot unlock lower segments under any circumstances. We use a term "trust ratchet" to describe how the system moves forward through the software, from trusted to untrusted, never going backwards, and never unlocking the hardware locks without first starting over at the known good state of reset. In more recent devices, we maintain the same concept of decreasing levels of trust, but use multiple processors with their own isolated memory subsystems instead of hardware locks on shared memory. One processor with its own memory is devoted exclusively to Miniboot. Another, with its own physically-separated memory is used for the OS and applications.

5.2 Officers

At the very core of Miniboot is the concept of "officers" (see Figure 2). Each officer has the authority to update a layer of code and its associated key storage, but nothing else. For layers N >=1, Miniboot authenticates a command from Officer N by verifying that the public-key signature on the command came from Officer N for that device.

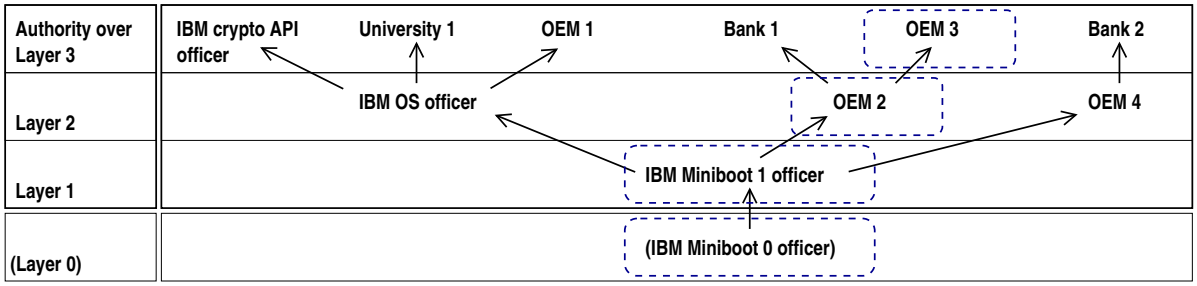

Figure 2: In this example, IBM is the officer that controls the two Miniboot layers, OEM2 is the officer that controls the OS, and OEM3 is the officer that controls the application

5.3 Commands

Miniboot has a very small number of commands, all listed in Figure 3. Software external to the device submits these commands to Miniboot over the primary communication interface between the untrusted host system and the secure coprocessor. The commands "Ordinary Burn 1" and "Ordinary Burn 2" request Miniboot to update the software burned in layers 1 and 2, respectively, in the persistent memory inside the secure enclosure. Some commands require that the requesting officer be authenticated, but others, such as simple queries, do not. Some commands are used only once, in the factory, such as "IBM Initialize". Other commands establish who is allowed to update a layer (who "owns" it), such as "Establish Officer 2." "Surrender Officer 2" relinquishes ownership of a layer.

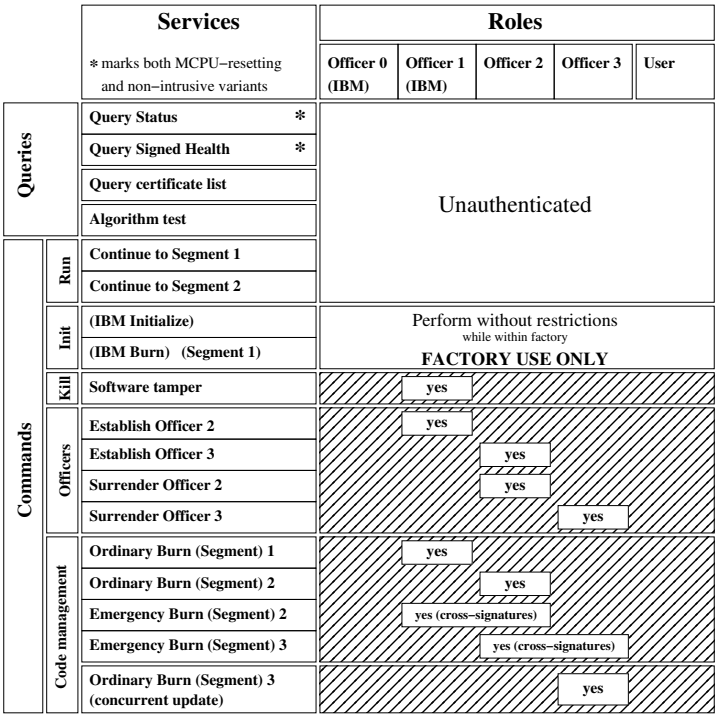

Figure 3: Miniboot commands and queries

5.4 Software Updates, Keys, and Outbound Authentication

Each time Miniboot 1 replaces itself, it generates a keypair for its successor and certifies the new public key with its current private key. This certificate establishes that if one trusted the current installation of Miniboot, then one can trust the next one. Each time the application configuration changes, Miniboot also generates and certifies a new keypair for the OS in Layer 2. (Miniboot also destroys the old Layer 2 private key.) This certification binds a keypair to a specific application configuration on a specific device. (The OS in Layer 2 can use this keypair to provide outbound authentication services to the application.) This binding, coupled with the trust chain for Miniboot's own keypair, permits parties to make accurate trust judgments about information coming out of the device.

6 Conclusion

A device exists that can securely update its own firmware in the field. It has been validated under FIPS 140-2 at level 4, meeting the strictest requirements for security. Its design has stood the test of time, for over 12 years. Aspects of its design may be applicable to the protection of firmware and firmware updates in other systems.

References

1. National Institute of Standards and Technology. Security Requirements for Cryptographic Modules. Federal Information Processing Standards Publication 140-1, 1994.

2. IBM eServer Cryptographic Coprocessor Security Module 4764-001 Security Policy, http://csrc.nist.gov/groups/STM/cmvp/documents/140-1/140sp/140sp661.pdf, 2007.