Adoption of the Least Privilege Separation Kernel (LPSK) for the Atom Platform

Paul C. Clark

October 2011
Porting the LPSK to the Atom Platform
This report was written in support of the NPS Military Wireless Communications Research Group. Any opinions, findings, conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect those of the sponsors.
Table of Contents

1 Introduction.......................................................................................................................1

2 Background .......................................................................................................................1
  2.1 Definitions ......................................................................................................................1
  2.2 Separation Kernels .........................................................................................................1
  2.3 Separation Kernel Protection Profile (SKPP) .................................................................3
  2.4 Trusted Computing Exemplar (TCX) ............................................................................3
  2.5 Least Privilege Separation Kernel (LPSK) .................................................................4
  2.6 LPSK System Architecture .......................................................................................4
  2.7 LPSK Configuration .....................................................................................................6

3 Progress Made...................................................................................................................7
  3.1 Hardware Platforms ......................................................................................................7
  3.2 Functional Specification ..............................................................................................9
  3.3 Boot Problem ...............................................................................................................9
  3.4 SATA Disk Driver .........................................................................................................9
  3.5 NIC Driver ....................................................................................................................10
  3.6 Memory Allocation ......................................................................................................11
  3.7 PC Hardware Clock .....................................................................................................11
  3.8 Software Clock ............................................................................................................11
  3.9 I/O Focus Efficiency ....................................................................................................12
  3.10 Configuration Tool and Vector ................................................................................12
  3.11 Audit .........................................................................................................................12

4 Current Functionality ........................................................................................................12

5 Future Work......................................................................................................................13

References............................................................................................................................15
Table of Figures

Figure 1 Complexity Relationships ................................................................. 2
Figure 2 Information Flow Between Partitions .................................................. 2
Figure 3 Virtual Machine Monitor (VMM) ........................................................ 3
Figure 4 LPSK System Architecture ................................................................. 5
Figure 5 Archos 9 PC Tablet (Taken from [9]) .................................................. 8
Figure 6 Acer Veriton N260G (Taken from [10]) ............................................. 8
Figure 7 HP rp3000 (Taken from [11]) ............................................................. 8

Table of Tables

Table 1 Definitions ............................................................................................. 1
Table 2 NICs on the Available Atom Platforms .............................................. 10
1 Introduction

This report provides a description of the FY11 work that was performed on the prototype Least Privilege Separation Kernel (LPSK) to adapt it to the Intel Atom processor. Section 2 provides the background information that may be necessary to understand the remainder of the report. Section 3 describes the progress that was made during the funded research. Section 4 summarizes the current functionality of the prototype kernel, while Section 5 describes potential future work.

2 Background

This section provides background information on the terminology and technology to be discussed in the remainder of the report.

2.1 Definitions

Table 1 provides a few definitions of security-related terms used in this report.

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>Principle of least privilege</td>
<td>The principle of least privilege states that a subject should only be given enough privilege to complete its assigned task, and no more. [1]</td>
</tr>
<tr>
<td>Secure attention key (SAK)</td>
<td>The secure attention key (SAK) is a non-spoofable user interface to the kernel.</td>
</tr>
<tr>
<td>Subject</td>
<td>A subject is an entity that causes information flows to occur.</td>
</tr>
<tr>
<td>Subject, trusted</td>
<td>A trusted subject is a subject configured to have the ability to cause information flows that are outside the normally allowed flows, but is trusted to not violate the intent of the policies being enforced.</td>
</tr>
<tr>
<td>Subject, untrusted</td>
<td>An untrusted subject is a subject that has no special privileges with respect to information flow.</td>
</tr>
</tbody>
</table>

2.2 Separation Kernels

A separation kernel is a small special purpose low-level executive whose general function is to partition resources, such as processes and memory, and then to control the flow of information between those partitions. The function of a separation kernel is even more primitive than a security kernel, which is more primitive than a general-purpose operating system, as graphically shown in Figure 1. [2]
A separation kernel enforces an information flow policy as configured by an administrator. The configuration describes how data is allowed to flow into or out of a partition. For example, if a subject is a subset of the resources in Partition A, the partition flow rules could be configured such that the subject is not allowed to read the resources allotted to partition B. Figure 2 shows an example where information can flow from partition A into B, but not from B to A. Information can flow from partition B to C, but not from C to B. In this example, information can therefore flow from partition A to partition C.

Because of its strong separation policies, a separation kernel is a good choice when building a virtual machine monitor (VMM) because of its ability to protect the resources of virtual machines running on the same platform, as shown in Figure 3. To be clear, separation kernels are not necessarily VMMs, and VMMs are not necessarily implemented as separation kernels.
The National Security Agency (NSA) sponsored the effort to specify the high assurance security requirements for a separation kernel within the Common Criteria (CC) framework [3]. This effort resulted in what is known as the Separation Kernel Protection Profile (SKPP) [4], which was completed in 2007.

Green Hills Software successfully evaluated its INTEGRITY-178B Operating System against the SKPP [5], while WindRiver’s MILS 2.0 product is currently under evaluation against the SKPP [6]. At least one other company claims that its product is compliant with the SKPP [7] without the actual evaluation to prove it.

2.4 Trusted Computing Exemplar (TCX)

Over the last 25 years very few products have been evaluated successfully at a high level of assurance, whether under the CC framework or the retired Trusted Computer Security Evaluation Criteria (TCSEC) framework [8]. It is hypothesized that one of the reasons for this failure is that there are few people who know how to take a product from conception to high-assurance evaluation, and much of that knowledge has not been shared, perhaps because it was viewed as proprietary. It is also hypothesized that there may be management at some companies who have contemplated entering the high assurance market, but see it as a big risk because of the unknown nature of high-assurance evaluation. These hypotheses are the basis for the Trusted Computing Exemplar (TCX) project at NPS.

The TCX project endeavors to reduce the risk of high assurance product development by publishing examples of the kinds of documentation, processes, and evidence that must be produced to meet high assurance requirements. As part of this effort, it is necessary to produce a component that undergoes high assurance evaluation to provide the “moral authority” for the information provided. It was therefore decided that the project would build a separation kernel that complies with the SKPP. The name given to this kernel is the Least Privilege Separation Kernel (LPSK).
2.5 Least Privilege Separation Kernel (LPSK)

The Least Privilege Separation Kernel (LPSK) is being designed and prototyped to meet the security requirements set forth in the SKPP. The SKPP is a high-level document that gives some leeway into how some of the mechanisms will be provided, as long as the minimal requirements are met. The LPSK enforces four policies, as described below:

- **Partition-to-partition flows**
  As described in Section 2.1, a partition information flow policy is a basic policy of separation kernels. An administrator configures which flows are allowed between declared resource partitions. This includes those flows that a trusted subject would be able to use.

- **Acyclic subset flows**
  The acyclic subset flows are similar to the partition-to-partition flows, but these flow declarations are meant to describe how the untrusted subjects of the system are restricted.

- **Subject-to-resource flows**
  These flows allow an administrator to comply with the principle of least privilege. It also provides the namesake for the LPSK. Without subject-to-resource flow rules, all subjects within a given partition have access to the same resources, even if they do not need those flows to perform their programmed duties. With subject-to-resource flows an administrator can declare finer-grained access control down to the resource level instead of the partition level.

- **Application Programming Interface (API) policies**
  The API policies allow an administrator to declare which kernel APIs are exported to each subject. An administrator can consider some APIs privileged and limit their use to only certain subjects.

2.6 LPSK System Architecture

The LPSK exports a basic API, but to provide a more useful and efficient application development environment additional services are required. The envisioned environment is shown in Figure 4. These services execute in separate hardware execution domains from the kernel.
The LPSK exports the partition abstraction, with security policies enforced upon all subjects. Between the LPSK and the applications exists a layer of software referred to as the Trusted Management Layer (TML). This layer either provides, or is envisioned to provide, or may provide the following services:

- **Scan code to ASCII conversion.**
  The kernel exports a keyboard device that emulates the hardware and therefore only returns scan codes. The TML returns data that is more generally expected by applications in the form of ASCII characters.

- **TCP/IP network stack**
  The kernel exports a network device that reads and writes Ethernet frames. The TML exports an interface to read and write packets with the ability to establish TCP connections.

- **File system**
  The kernel exports a disk interface that reads and writes disk sectors. The TML may provide a high-level file system for reading and writing files.

- **Human-Readable classification labels**
  If the declared policies enforce a mandatory access control (MAC) policy, then the TML may provide a mechanism to translate partitions flow rules into classifications in a human-readable format.

- **Console support**
  The kernel exports a text-based video device for randomly reading and writing ASCII characters to the screen. The TML combines the keyboard and video devices exported by the LPSK into a console device that allows users to see the typed characters on the screen, to backspace and delete information on the screen, and to scroll lines of text off the top of the screen as expected in a command-line environment.
• Identification & Authentication
  The TML provides the ability to log into the system with a user name and password.
• Standard C-library APIs
  Having standard C-library support allows for quicker application development and the potential for porting applications to the environment with less effort than otherwise.

One of the applications currently provided is the Trusted Path Application (TPA), which is a trusted subject that provides a user interface for security-relevant actions. It is in this application that users log in, and where they can change their password. It also provides the interface to select which of the partitions has exclusive access to user I/O via the keyboard and console, which is referred to as I/O Focus.

2.7 LPSK Configuration

The configuration of an LPSK instance is documented in what is referred to in the SKPP as a configuration vector. The SKPP requires the configuration vector to be in two forms: a human-readable form that an administrator reads and modifies, and a binary form that the kernel reads. The SKPP specifies that there shall exist a configuration tool that transfers vectors between these two forms.

A high-level view of how the LPSK is configured is given below:
• Declare the audit configuration
  o Turn auditing on or off
  o The size of the internal audit buffer
  o The action to take when the buffer gets full
• Run-time issues:
  o Configure how the kernel will display messages to the user
  o Reserve memory locations (e.g., memory required by drivers)
• Declare the partitions
  o The time it takes to execute all subjects across all partitions.
  o For each partition:
    ▪ Declare a partition active or passive (A passive partition cannot contain subjects in its set of resources)
    ▪ The percentage of the execution time that is allocated to subjects in this partition.
    ▪ The percentage of the system RAM allocated to the partition.
  o The partition that has the initial I/O focus.
  o The partition that receives I/O focus when the SAK is invoked
• Partition flow rules
  o Partition-to-partition flows
  o Acyclic flow rules
• Files to be imported during kernel initialization. For each file:
  o Location on disk
  o Assigned partition
  o Events that require auditing
Porting the LPSK to the Atom Platform

- RAM to be allocated
  - Size
  - Assigned partition
  - Events that require auditing
- Device declarations
  - Assigned partition
  - Device behavior (i.e., multiplexed between partitions or dedicated to a single partition)
  - Device-specific attributes (e.g., keyboard buffer size, or Ethernet MAC address)
  - Events that require auditing
- Process definitions
  - Assigned partition
  - The percentage of the execution time allocated to the partition
  - Subjects
    - Code location
    - Execution domain
    - Subject-to-resource flows allowed
    - Kernel APIs that can be used
    - Trusted or untrusted
    - Events that require auditing

3 Progress Made

The objective of the funded work was to port the original LPSK prototype to PC platforms that have the Intel Atom CPU, which is a small low-power CPU that will eventually make its way into small, embedded devices. Because the Intel Atom is x86-based, the effort was focused mainly on getting the LPSK to run on a modern PC architecture.

3.1 Hardware Platforms

Some funds were initially expended to purchase some Atom-based PCs. Three different makes and models were purchased that were seen to have different levels of potential success in the porting effort. The original LPSK prototype required a PS/2 keyboard, not the more common USB keyboard seen today. Buying PCs with PS/2 ports was seen as a less risky approach, though it limited the purchasing options. The purchased platforms are described in the following subsections.

3.1.1 Archos 9 PC Tablet

The Archos 9 PC Tablet, shown in Figure 5, was considered to be the most risky platform because it does not have a PS/2 port, but at the time of purchase its form factor was closest to that of a small handheld device, which is the focus of the sponsoring organization. It was hoped that either the BIOS provided PS/2-to-USB keyboard emulation to allow a USB keyboard to work with the LPSK, or that the porting effort would get far enough to provide the necessary USB drivers. Neither of those hopes was
Porting the LPSK to the Atom Platform

realized. The Archos 9 also did not have a separate NIC port, but rather depended on an
embedded wireless NIC for network connectivity or perhaps a USB-based NIC.

Figure 5 Archos 9 PC Tablet (Taken from [9])

3.1.2 Acer Veriton N260G

Of the Atom PCs with a PS/2 port the Acer Veriton N260G, shown in Figure 6, was the
smallest and, as it turned out, the most versatile of the three purchased. Even though it
has a PS/2 port it was always used with a USB keyboard because its BIOS provided good
PS2-to-USB keyboard emulation.

Figure 6 Acer Veriton N260G (Taken from [10])

3.1.3 HP rp3000

The HP rp3000, shown in Figure 7, also worked well for testing. It also has a PS/2 port
that turned out to be unnecessary because of a good USB emulation.

Figure 7 HP rp3000 (Taken from [11])
3.2 Functional Specification

The functional specification for the LPSK was started before this funded research began, but this research allowed Volume 1 of the LPSK functional specification to be completed. Volume 1 provides approximately 80 pages of prose that describe the exported interfaces of the LPSK. Volume 2, which provides a detailed description of the user interfaces and APIs of the LPSK, was started during this research, and is ongoing work.

3.3 Boot Problem

The first technical problem encountered during the porting effort was that the original LPSK prototype would not boot on the Archos 9 PC Tablet. The prototype kernel depends on a modified version of the grub boot loader [12] to install the kernel into memory and jump to its initialization point. Grub booted the kernel on the other two Atom platforms but not this one.

The lower 1MB of RAM is treated differently in the x86 architecture from the rest of RAM. After much frustration and debugging, it was found that grub was assuming that the lower 1MB of RAM was fully loaded, but the Archos 9 only has 640K of lower RAM. Grub was attempting to load some of the kernel into the small range between 640K and 1000K, which was failing. After this discovery, grub was modified to avoid the upper part of lower RAM when loading the kernel. Thereafter the LPSK booted on the Archos system.

3.4 SATA Disk Driver

The original LPSK prototype could only read and write to the Parallel-ATA (PATA) drives (also known as IDE drives), but because the Atom is a new CPU it often comes on a motherboard that only supports the newer Serial-ATA (SATA) drives. Therefore, a new LPSK disk driver was needed. The benefit of writing a SATA driver is that the SATA interface is a recognized standard; a driver that works on one system should work on another system, barring bugs in the hardware.

Portions of the SATA specification are not freely available and must be purchased, which was done with research funds. The higher-level specifications were obtained for free from Intel. A SATA driver was successfully developed using these specifications. Because there was little to no help on the Internet for writing a SATA driver, a small technical report was written as an aid to others who want to write their own driver from scratch.

The LPSK now detects the presence of a SATA controller, determines which SATA ports have a connected hard disk, and then probes each disk to determine the disk partitioning. Each SATA disk partition is exported by the LPSK as a SATA device. The driver can read and write sectors on all SATA controllers and drives upon which it was tested. The prototype test programs were also updated to include the SATA interfaces.
3.5 NIC Driver

The original LPSK prototype did not have a NIC driver, but because the intent of the sponsored research was related to communication, it was felt that a NIC driver should be developed. The problem with Ethernet controllers is that, unlike disk controllers, there is no standard interface to communicate with a NIC, so each NIC needs a different driver. At best, a family of NIC models may need only one driver, as long as a feature in a newer NIC is not required.

The NICs installed in the three platforms described in Section 3.1 are listed in Table 2.

### Table 2 NICs on the Available Atom Platforms

<table>
<thead>
<tr>
<th>Atom Platform</th>
<th>Network Interface Card</th>
<th>Interface</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archos 9 PC Tablet</td>
<td>Broadcom BCM4312</td>
<td>Wireless (b/g)</td>
</tr>
<tr>
<td>Acer Veriton N260G</td>
<td>RaLink Device 3090</td>
<td>Gigabit Ethernet</td>
</tr>
<tr>
<td>HP rp3000</td>
<td>Broadcom NetXtreme BCM5764M</td>
<td>Gigabit Ethernet</td>
</tr>
</tbody>
</table>

Two NICs were targeted for LPSK driver development: the AMD PCnet32 and the Broadcom NetXtreme BCM5764M. The AMD NIC was targeted for two reasons: 1) specifications are openly and freely available; and 2) because the NIC is exported by VMware it would allow the LPSK to access a network from within a VM. The Broadcom NIC was targeted for two reasons: 1) specifications were openly and freely available; and 2) Broadcom chips are common in NICs, so it was reasoned that one driver might work in many situations. Writing NIC drivers was the most difficult and the most time consuming of all the efforts described in this report.

The RaLink NIC was not targeted because it has no publicly available interface specification. The Broadcom wireless NIC was not targeted because there was no LPSK keyboard interface to the Archos 9 platform.

The first successful NIC driver was written for the AMD PCnet32. The first effort to get the driver working involved an analysis of the associated driver in Linux. Because of the size and complexity of the Linux driver and the simple programming interface for the AMD PCnet32, it was decided that it would not take much effort to write a simple driver from scratch. This did not prove to be true. After a lot of time, the strategy of writing a driver from scratch was abandoned to look for another alternative.

The second effort to provide an AMD PCnet32 driver involved an attempt to port a simple NIC driver from gPXE, an open source network bootloader [13]. However, the code was written to work on many different CPU platforms using odd C directives, pragmas, and macros as well as odd compiler and link options. It was very difficult code to work with, despite its relatively small size. After considerable experimentation, this effort was abandoned to look for another alternative for the driver.

After much searching, a very simple yet complete driver was found for a small open source OS called Sanos [14] that was focused solely on the x86 architecture. With
relatively little effort the code was successfully ported to the LPSK. After one NIC was
working, the code was expanded to work with several NICs at once. The LPSK test
program was expanded to test Ethernet drivers by first sending Address Resolution
Protocol (ARP) requests and processing the response, and then to send out an Internet
Control Message Protocol (ICMP) echo (ping) request and to process the response.

In truth, implementation of the Broadcom driver was attempted before the AMD driver.
First, the associated Linux driver was analyzed to determine the difficulty of porting the
Linux driver to the LPSK. Because of the size and complexity of the driver, its attempt
to include all Broadcom NICs in its scope, and the lack of comments made it undesirable
to port. Using the specification, which is much more complicated than the AMD
specification, an attempt was made to write a simple driver from scratch, but after
considerable effort the driver could not be made to work properly. The Linux code was
occasionally referenced for guidance, but it was difficult to discern the design from the
code. Had the time available to work on the project not expired, an attempt would have
been made to port a driver from the Sanos project (if it exists), which would probably
have succeeded in light of the earlier success with its AMD driver.

### 3.6 Memory Allocation

All the NIC drivers investigated in this study made use of a memory allocation routine
called `kmalloc` to obtain a chunk of unused memory of a requested size. Therefore, a
general-purpose heap manager was implemented in the LPSK to get and return chunks of
memory on demand.

### 3.7 PC Hardware Clock

A new clock interface for the LPSK was implemented to read the battery-powered PC
hardware clock. This exported clock interface allows applications to read the current date
and time. The test program was updated to include this interface in the test suite.

### 3.8 Software Clock

The hardware clock only provides granularity down to the second, which is not enough
granularity when the kernel needs to pause for small increments measured in
milliseconds. The solution to this problem was to use the CPU clock to measure elapsed
time. Modifying the kernel to use the CPU to measure time involved many pieces, as
described below:

- The Atom CPU has a register that counts the number of CPU clock pulses that
  have transpired since the CPU was reset. The Atom CPU also has an assembly
  instruction for reading that register.

- However, the CPU register mentioned above is 64-bits wide, and the LPSK is
  limited to a maximum variable size of 32-bits because of the compiler that is used
to generate the kernel. Therefore, to make use of clock-pulse functionality, a
  math library had to be constructed to perform 64-bit math operations using
  operands that are split into two 32-bit chunks.

- Measuring clock pulses is not enough because the kernel must know how many
  CPU clock pulses are issued per second. Therefore, the frequency of the CPU
  needs to be calculated by the kernel during initialization. Counting the number of
Pulses over a known period can be used to determine the frequency, but that once again requires an accurate clock.

- The last piece of the puzzle requires the kernel to use the PC real-time clock (RTC) to measure the number of clock pulses in one second. This final piece allows the kernel to calculate the number of clock pulses per millisecond.

Putting that all together allows the LPSK to measure the passage of time at a high level of granularity. That granularity is currently not exported at the kernel interface.

### 3.9 I/O Focus Efficiency

The I/O focus functionality requires the kernel to maintain a video buffer for each active partition. When I/O focus changes, the kernel copies the contents of the buffer for the new partition with focus into the video memory. The efficiency of this approach was a concern when the SAK was invoked because it necessitated a lot of work for the keyboard interrupt handler when interrupt handlers are supposed to keep their processing to a minimum.

It was found that the Atom platform provides seven separate buffers for video memory. Therefore, the kernel was modified to eliminate the buffering for the SAK-partition, and instead now writes all video data for the SAK partition into one of the additional video memory buffers. Now, when the SAK is invoked the kernel simply informs the video card to use the SAK buffer within the video memory, which significantly reduces the processing time for the keyboard interrupt handler.

### 3.10 Configuration Tool and Vector

A thesis student made some major changes in how the LPSK is configured [15]. The student created an XML schema to describe a valid XML representation of the LPSK configuration options that are described in the functional specification. The student then wrote a tool to transform an XML configuration vector into a binary vector, and vice versa. The student was not able to integrate a lot of this new functionality into the LPSK, but some additional use of this new configuration vector was made during the performance of this research. In addition, some bugs were fixed that were identified in the schema and the configuration tool.

### 3.11 Audit

A thesis student designed and implemented an audit subsystem for the LPSK [16]. The audit subsystem was integrated into the LPSK, but the student was only able to modify a small portion of the kernel to make use of the new subsystem. Some of that work was expanded during the performance of this research, but there remains much more audit integration before it can be considered a complete prototype.

### 4 Current Functionality

As brief listing of the functionality that is currently exported by the prototype LPSK is given below:

- Kernel configuration via an XML schema and configuration tool.
• Multitasking processes.
• Segmented memory.
• Device drivers for the following:
  o PATA disk controllers and disks
  o SATA disk controllers and disks
  o AMD PCnet32 Ethernet controller
  o PC hardware clock
  o Software clock
  o Text-based video
  o PS/2 Keyboard
• Inter-process communication using the following:
  o Eventcounts and sequencers
  o Signals
  o Shared memory
• Intra-process interrupt handling
• I/O focus switching
• Secure attention key
• Some event auditing

The following lists the functionality that is currently provided outside the run-time LPSK:
• Keyboard scan code to ASCII character translation.
• Console support
• Identification and authentication
• Trusted path application
• Test applications

5 Future Work

There is considerable work required to complete an LPSK prototype that conforms fully
to the functional specification. The following is a brief list that describes the potential for
future work in the LPSK prototype:
• Volume 2 of functional spec
  This work was started, but there is much more work to do to completely specify
  the programming interfaces for the LPSK.
• Kernel auditing
  Audit functionality needs to be pervasive within the kernel, but auditing events
  are currently only detected in a few modules.
• Implement all specified security policies
  There is still some work that needs to be done to fully implement all the specified
  security policies. The enforcement of the API policy must be started, and will
  require an update to the configuration tool so that when implemented it is user
  friendly.
• Multiple processes per partition
  The prototype LPSK is currently limited to one process per partition, but that
  needs to be expanded to a much bigger number. This will require a major change
to the prototype.
• Expand the device interface to deal with a control function.
The functional specification describes an interface for reading and modifying
device configuration information during run-time. This interface has not been
implemented.

• Provide additional NIC drivers
Only one NIC driver is currently supported. This is a problematic area that all
operating systems deal with because for each make and model of NIC that needs
to be supported, there may need to be a separate driver.

• USB driver
There is currently no support for any USB devices because there is no USB bus
driver. This will open up possibilities for many USB-based devices. A thesis
student has signed up for this task.

• USB keyboard driver
Once a USB bus driver is available then a USB keyboard driver needs to be
provided because fewer and fewer systems support PS/2 ports, and eventually the
ability to provide PS/2-to-USB emulation will disappear from the BIOS.

• Graphics driver
The current prototype only provides text-based video, but graphical interfaces
should be made available. VGA provides a standard graphics API at a relatively
low resolution, but there are no standards at higher resolutions. The situation with
modern graphics drivers is similar to NIC drivers where each card is unique. In
the short term, however, a VGA interface can be provided because all graphics
cards support this standard.

The following provides a list of potential future work outside the run-time LPSK:

• Provide the complete TCP/IP stack
The LPSK currently provides an interface to read and write Ethernet frames, but
to be useful in a network context there needs to be an interface that provides a
standard TCP/IP interface.

• Integrate C-library support
A thesis student implemented many standard C-library functions for the intended
use of applications running on top of the TML [17]. The student also ported some
simple C-based applications that depend on standard C-library functions. This
work was not integrated into the source trunk, and existing applications were not
converted to the C-library calls.

• Audit review application
There needs to be a design for an audit review subsystem and application. How
will audit records be read from the kernel in a timely fashion? Where will the
audit records be permanently stored? What is the user interface to a good audit
review application? Is this a service provided by the TML or a dedicated
application?

• New boot loader
The dependence on the grub boot loader needs to be severed, with a boot loader
that complies with the functional specification.

• New installation application
An LPSK-specific installation program needs to be developed that complies with the functional specification.

References


Porting the LPSK to the Atom Platform