Towards a certifiable MILS based workstation

Nils Agne Nordbotten and Tor Gjertsen

Norwegian Defence Research Establishment (FFI)

20 February 2012

FFI-rapport 2012/00049

1176

P: ISBN 978-82-464-2081-3
E: ISBN 978-82-464-2082-0

Keywords

Sikkerhet

Virtualisering

Operativsystemer

MILS

MLS

Approved by

Rolf Rasmussen	Project Manager
Anders Eggen	Director

English summary

This report is based on work at FFI towards the potential realization of a certifiable workstation for handling multiple security classifications. To realize such a solution based on the use of a MILS separation kernel, it is deemed essential to have a secure way to share the keyboard, mouse, and screen between partitions. We propose a design for this where it is ensured through the configuration of the separation kernel that data cannot flow between user partitions (i.e., classification levels), thereby simplifying the certification of such a system. The principal design can be generalized to handle all devices that are used either as purely input or purely output devices and it is also described in this report how the proposed solution can be applied to a touchscreen device.

A basic proof-of-concept prototype has been implemented and was demonstrated as part of the SOA Pilot at FFI in June 2011. This prototype adheres to the principal design, but does not represent a certifiable implementation.

The realization of the full potential of such a system depends on the existence of a separation kernel certified on suitable hardware. The last chapter of this report therefore provides an overview of the current status with regard to certification of separation kernels, before a discussion of potential ways forward.

Sammendrag

Denne rapporten omhandler arbeid ved FFI med tanke på en mulig realisering av en arbeidsstasjon for å håndtere flere graderingsnivå. For å kunne realisere en slik løsning basert på en MILS separasjonskjerne, er det nødvendig å kunne dele tastatur, mus og skjerm mellom partisjoner på en sikker måte. Vi foreslår en løsning for dette hvor konfigurasjonen av separasjonskjernen forsikrer at data ikke kan flyte mellom brukerpartisjoner (graderingsnivå), noe som dermed forenkler sertifiseringen av et slikt system. Det overordnede designet kan generaliseres til å håndtere alle enheter som benyttes som enten rene inn-enheter eller rene ut-enheter. Det beskrives også i rapporten hvordan den foreslåtte løsningen kan anvendes på en enhet med berøringsskjerm.

Det er implementert en prototype av den foreslåtte løsningen og denne ble demonstrert som en del av SOA piloten på FFI i juni 2011. Prototypen overholder det foreslåtte overordnede designet, men utgjør ikke en sertifiserbar implementasjon.

Det er en forutsetning for å kunne realisere det fulle potensialet i en slik løsning at en har en separasjonskjerne som er sertifisert på egnet maskinvare. Det siste kapitlet i denne rapporten gir derfor en oversikt over sertifiseringsstatus for separasjonskjerner, før det avsluttes med en diskusjon av mulige veier videre.

Contents

1 Introduction

In military systems there is a need for handling information of different security classifications. Traditionally this has been solved by using separate systems for the different security levels. However, this implies duplication in equipment and procedures, and is therefore often inefficient and unpractical in operation.

The need for a unit that can be accredited to handle multiple classifications is recognized particularly for field operations, where weight and power consumption is of great importance. There obviously is a limit to how much a soldier can carry on his back. Also in vehicles the space is often very limited, and it is important to be able to combine many functions in one unit.
In [1] we examined potential alternative approaches towards handling information of different classifications on the same physical terminal. Although several products were identified providing this feature, they were not applicable to scenarios requiring truly mobile high assurance equipment.

In order to facilitate such scenarios, an approach based on a MILS (multiple independent levels of security) separation kernel was identified as the best applicable software based alternative. Given the currently available technology, a software based approach has the benefit of providing a more flexible and space efficient solution.

The alternative based on the use of a MILS separation kernel is further examined in this report, based on our work with implementing a prototype of such a system. The aim has been to design a solution that would be relatively easy to certify, given a separation kernel certified on suitable hardware. Although the prototype implementation adheres to the proposed principal design, it has been created for demonstration and experimentation purposes and is by no means a certifiable high assurance implementation. The prototype has been implemented on a desktop type of system, but it is equally applicable to a laptop form factor. Furthermore, as will be discussed in Section 2.5, the principal design is also applicable to a touchscreen device such as a tablet.

The remainder of this report is organized as follows: Chapter 0 first presents the principal design of the solution, before providing an overview of the prototype implementation and a discussion of covert channels. The prototype implementation may be seen as a proof-of-concept with regard to the principal design. Because the prototype only implements the necessary basic functionality, potential additions and enhancements are discussed in the later sections of Chapter 0. Some example use cases are then presented in Chapter 3. Finally, in Chapter 4, we provide an overview of the current status with regard to certification of separation kernels before concluding with a discussion of potential ways forward.

2 Prototype workstation

The aim of the work with the prototype workstation has been to create an end-user solution for handling multiple classification levels on the same machine. As such, confidentiality in the form of maintaining separation between confidentiality levels has been our main focus. The resulting solution may also have other applications, however, as exemplified in Chapter 3.

In this chapter we first present the proposed principal design, which represents a further exploration of one of the designs that we first sketched in [1]. Afterwards, an overview of the prototype implementation is provided.

2.1 Design

One of the main design goals has been that the assurance with regard to the confidentiality security requirement should to as large extent as possible be based on the assurance provided by the underlying separation kernel. A direct implication of this is that as few trusted (i.e., security critical) additions as possible should be made. Also, when trusted additions are required, the consequences in the undesired case of these failing should be limited.

In order to realize such a design, we decomposed the system into several parts, as shown in Figure 2.1. There can be one or more user partitions (typically at least two). Each of these user partitions hosts an operating system (alternatively a special purpose user application) and typically corresponds to a given security classification. Strong separation between partitions is provided by the separation kernel.

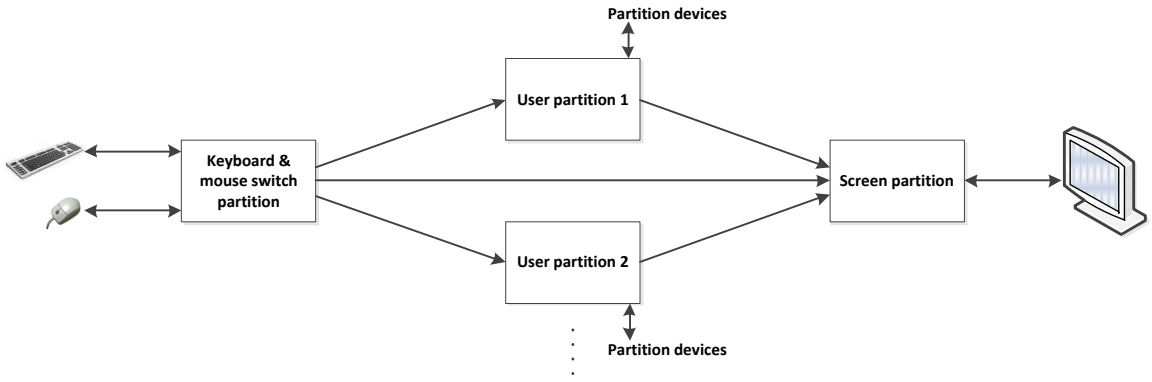

Figure 2.1 The prototype design illustrated in the case of having two user partitions. Each box represents a MILS partition, while each arrow represents a directed data channel.(The requirement for having a channel from the keyboard & mouse switch partition to the screen partition is implementation dependent.)

The screen partition, to the right, is responsible for displaying the output from the user partitions on the screen. This can be realized in several forms. One alternative is to only display one user partition at a time. A second alternative is to display the various user partitions concurrently

within pre-defined areas of the screen. A third alternative is to show each user partition within a separate window on the screen. This last alternative enables the user partitions to be displayed concurrently within separate windows, while allowing the windows corresponding to each user partition to be moved and resized (and potentially overlap) according to user preferences. The windows corresponding to each level may be distinguished through naming (i.e., text marking), different border colors, and/or different desktop themes and backgrounds. In the case where a user partition only runs a single application, this may also represent a distinguishing factor.

The keyboard & mouse switch resides within a separate partition and provides sharing of the mouse and keyboard between the user partitions. This way, the keyboard and mouse switch enables the user to interact with a single user partition at a time. Switching between different user partitions is performed using a keyboard combination that is intercepted by the keyboard and mouse switch software.[1] We will refer to the partition receiving the keyboard/mouse input at a given time as the active partition. If the display from each user partition is shown within separate windows on the screen, the keyboard & mouse switch partition should also be connected to the screen partition, as shown in the figure. The reason for this is to enable the user to move and resize the windows corresponding to each user partition. The screen partition can then be selected as the active partition in the same way as a user partition. In the case where only one user partition is displayed at a time, the channel from the keyboard & mouse switch to the screen partition is used for signaling a switch of active partition. Finally, in the case where each user partition is shown within a predefined area, this channel is not strictly required.

The use of one-way channels allows data to flow from the input devices (i.e., keyboard and mouse) to the selected user (or screen) partition and from the user partitions to the screen partition. In particular, it should be noticed that there is no direct or indirect channel from one user partition to another. This way, it is assured based on the configuration of the separation kernel that information cannot flow from one user partition to another user partition.

Because of this, neither the software in the user partitions nor in the screen partition needs to be trusted not to leak data to other parts of the system. This is a clear advantage as it allows these partitions to be based on untrusted commodity or legacy software. Nevertheless, it is important that the screen partition does not interchange data from the different user partitions when displaying on the screen.

[1] This may either be done using a single keyboard combination for cycling through the different partitions or by having a separate keyboard combination for each partition. Given a static configuration of user partitions, the latter approach seems preferable as it enables faster switching as well as labeling of the keys corresponding to the different partitions (e.g. with security classification). Alternatively, special purpose keys could be added to the keyboard or on a separate device connected to the switch partition.

Such mix-up in the display could potentially mislead the user to believe that some data is unclassified, while it in fact is classified. Still, any information leakage resulting from such would need to go through the human user and as such represents a more acceptable risk.

As a result, the keyboard & mouse switch constitutes the only trusted partition with regard to enforcing data flow to other partitions. More specifically, this partition is trusted not to leak user input intended for a given partition to another partition, and to correctly perform selection of the currently active partition. However, even in the case of malfunction, the keyboard & mouse switch is not able to leak information from one user partition to another. This reduces the potential consequences of a malfunction, as the user partitions may contain large amounts of data and potentially have access to other classified systems over a network. The keyboard & mouse switch is also difficult to attack for an attacker, especially without physical access to the machine,[2] as this partition only accepts input from the keyboard and mouse.
An actual implementation of the described design may chose to split one or more of the depicted partitions into multiple finer granularity partitions, as long as the high level design remains the same.

It should also be noticed that this design (used to share the keyboard, mouse, and screen) can be generalized to apply to all devices that can be used as either purely input or purely output devices. Such additional devices may optionally be handled within separate partitions.
As indicated in Figure 2.1, each user partition may optionally also be connected to one or more devices, such as a network interface or a disk drive (which are typically not used as purely input or purely output devices). Such a partition device is only accessible by the specified user partition and cannot be shared with other user partitions. [3]

2.2 Prototype implementation

This section provides an overview of our prototype implementation of the above design. As previously stated, this prototype implementation is not intended as a certifiable solution, but was created for demonstration and experimentation purposes. Several of the implementation choices would most likely be different for a production system.

[2] Withstanding attacks from a person with physical access to the machine has not been a goal of this work. (In that case, one would for instance be vulnerable to a keylogger attached to the physical keyboard connector or within the firmware of the keyboard. Such a keylogger might potentially replay input provided to a classified partition once switching to an unclassified partition.)

[3] The issue of sharing of a device, used for both input and output, is further discussed in Section 2.4.1.3.

It was considered an advantage to be able to reuse existing software for part of the prototype implementation. For that reason, it was decided to use open-source VNC (Virtual Network Computing) software as the basis for the screen/desktop handling. By using VNC, the desktop of each user partition is shown within its own VNC viewer window on the screen.

VNC was chosen because it is widely available as open-source for both Linux and Windows. Because we would be using the software in a nonstandard way, an open-source implementation was considered to be a plus as it would enable us to make any required changes. However, no such changes to the VNC software have been required.

The prototype implementation also makes use of a commercial separation kernel that was already available at FFI (i.e., LynxSecure 3.1). A paravirtualized Linux is used as guest operating system within each partition.

The remaining software modules, described in the following sections, have been implemented at FFI.

Figure 2.2 shows an overview of the prototype implementation. For simplicity it is shown with a single user partition, but additional user partitions can be included in a corresponding manner.[4] We will now provide a short description of each component.

[4] For connecting a new user partition to the keyboard & mouse switch, a separate message channel (MC) is added to the existing switch instance. For connecting to the screen partition, separate instances of the VNC viewer, VProxy, SHM driver, shared memory and interrupt(s) are added.

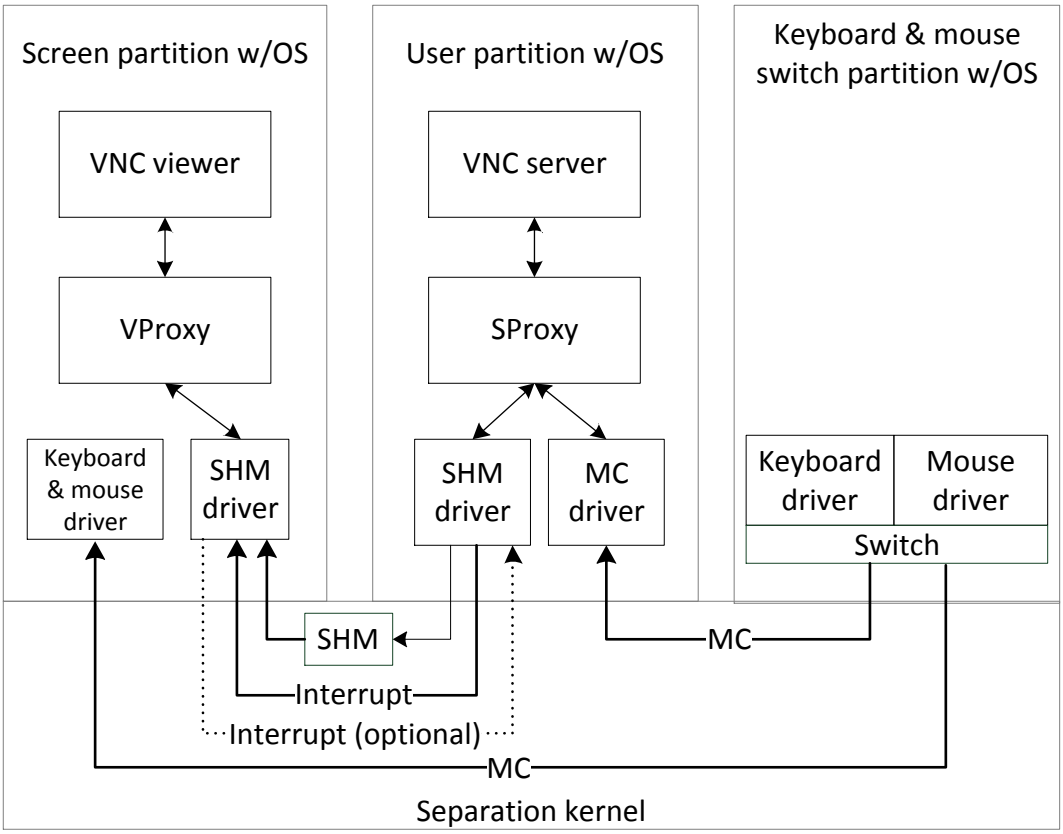

Figure 2.2 Overview of the prototype implementation. For simplicity only a single user partition is shown.

2.2.1 VNC viewer and server (screen partition and user partition)

As stated previously, standard VNC software was used. The VNC viewer (screen partition) displays the screen from the connected VNC server (user partition) within its window on the screen. The VNC viewer and server are connected to the VProxy and SProxy, respectively, using the RFB (remote framebuffer) protocol [2] over a local TCP connection.

2.2.2 VProxy (screen partition)

From the perspective of the VNC viewer, the VProxy resembles a standard VNC server. Because the VProxy has no return data channel to the true VNC server, however, the proxy itself is required to provide the correct RFB protocol responses to the viewer. As such, the proxy performs handshaking (including authentication of the client/viewer) and exchanges initialization messages with the client. After initialization the proxy forwards screen updates (i.e., RFB FramebufferUpdates) to the viewer upon request from the viewer. The proxy receives the screen updates through the shared memory (SHM) driver, which is accessed as a Linux device file.

2.2.3 SHM driver (screen partition and user partition)

The shared memory (SHM) driver provides access to the shared memory.[5] The SHM driver is implemented as a kernel module and utilizes the API provided by the separation kernel for accessing the shared memory. As shown in the figure, there are instances of this driver both in the reading partition (i.e., the screen partition) and in the writing partition (i.e., the user partition). While both partitions have read access to the shared memory, only the partition on the writing side (i.e., the user partition) has write access.

The SHM driver provides two alternative modes of operation. In both modes, the writing SHM driver signals the reading SHM driver with a synthetic interrupt each time it has finished writing to the shared memory.

In the default mode, the SHM driver on the reading side also sends an interrupt signal (to the SHM driver on the writing side) each time it is ready to read new data. A field at the start of the buffer is used to communicate the number of bytes written (i.e., to be read). The SHM driver basically provides a one-way byte stream in this mode, but with a sufficiently large buffer it resembles a one-way message queue with space for one message.

The alternative mode is based on the use of multiple buffering to avoid reading of partially written data, and therefore does not require interrupt signaling from the reader to the writer. In this mode, the shared memory is divided into two or more buffers each with enough space for an entire message (i.e., RFB FramebufferUpdate). At the start of the shared memory, there is an index that points to the last written buffer. When writing data, the writing SHM driver writes to the buffer with the oldest data. It then updates the index buffer so that it points to the newly written buffer, before it sends an interrupt signal to the receiving SHM.[6] The reader then reads the data from the buffer indicated by the index. Without additional time based synchronization, there is a risk of missing some screen updates using this method and it therefore requires sending the entire screen in each update (as opposed to the default of using incremental screen updates).

As mentioned, the SHM (driver) is accessed by applications (i.e., the proxies) as a Linux device file. The driver implements the poll method in order to support the use of select() in the proxies, facilitating synchronization between the proxy and SHM driver based on whether the shared memory has data ready to be read (on the reading side) or is waiting for data to be written (on the writing side).

---

[5] An alternative implementation could have been to use a message channel, instead of shared memory, for transmitting the screen updates. Extensive fragmentation would have been required, however, as the message channels (provided by the separation kernel) only support a limited message size.

[6] An alternative would have been to not to send an interrupt but have the receiving SHM driver poll the shared memory for new data, using sequence numbers to avoid duplicates.

2.2.4 SProxy (user partition)

The SProxy in the user partition is implemented in a similar manner as the VProxy in the screen partition. However, it has slightly different functionality and roles. More specifically, its main roles are to request screen updates from the VNC server, forward screen updates (i.e., RFB FramebufferUpdate messages) from the VNC server, and forward keyboard and mouse input to the VNC server. The latter includes taking keyboard and mouse input and forwarding it as RFB KeyEvent and PointerEvent messages respectively. This way, the VNC server does not see that the user input does not originate from the VNC viewer as it normally would. Compatibility between the configurations used by the SProxy and the VProxy, and thereby the VNC viewer and server, is ensured through static configuration (i.e., using identical configuration variables for corresponding instances of the VProxy and SProxy).

2.2.5 Keyboard/mouse driver and switch (keyboard & mouse switch partition)

The keyboard & mouse switch partition is the only partition with access to the physical keyboard and mouse devices. The keyboard/mouse driver in this partition provides interrupt handlers for the physical interrupts from these devices. Upon receiving keyboard input, it is first determined whether it is a key combination used for switching the active partition. If this is the case, it sets the active partition (not forwarding the input). If the input does not match a combination for switching partition, it is forwarded to the currently active partition for further handling. Mouse input is simply forwarded to the currently active partition.

The user receives implicit confirmation of the currently active user partition through observing the results of his/her actions in the corresponding window. A potential improvement would be to also send a notification to the screen partition, providing an explicit visual confirmation of the current partition. Such feedback could potentially also be provided by for instance flashing a light on the keyboard, corresponding to the specified partition, in which case one would not need to rely on the (untrusted) screen partition to provide this feedback. A one-way (ordered and lossless) message channel, provided by the separation kernel, is used for forwarding input to the currently selected partition.

The keyboard & mouse switch partition is trusted not to leak user input intended for one partition to another partition, and to correctly perform context switches. In order to enable assurance of the correct functioning of these mechanisms, this functionality should be implemented on top of the minimal runtime interface provided by the separation kernel (i.e., without a hosted operating system). Our sole motivation for including a hosted operating system within this partition was to reduce the prototyping effort, as this does not make a difference with regard to the principal design.

2.2.6 MC driver (user partition)

The MC driver is a kernel module enabling interaction between the SProxy and the message channel (MC). It makes use of the API provided by the separation kernel and is accessed by the SProxy as a Linux device file in a similar manner as the SHM driver.

2.2.7 Keyboard/mouse driver (screen partition)

The keyboard/mouse driver, in the screen partition, interacts with the message channel in the same way as the MC driver. However, keyboard/mouse events are not forwarded to a proxy, but are instead inserted to the guest OS as if they originated from a locally attached keyboard/mouse.

2.3 Covert channels

The configuration of the separation kernel ensures that there are no (direct or indirect) legitimate data channels between user partitions (i.e., when having a configuration with multiple user partitions). Likewise, there is no (direct or indirect) legitimate data channel from the screen partition to a user partition.

As one may recall, the default mode of our SHM driver makes use of synthetic interrupt signaling from the receiving partition to the sending partition (requiring this specific interrupt to be allowed in the configuration of the separation kernel). Allowing this interrupt signal provides a channel that can be misused by a malicious application within the screen partition to signal data to a collaborating application within a user partition.[7] Although this specific channel can be removed by using the alternative transfer mode, based on multiple buffering (in which case the separation kernel should be configured to not allow this interrupt), there are likely to be other covert channels between partitions.

In fact, the SKPP (Separation Kernel Protection Profile) [3] does not forbid the existence of covert channels in separation kernels. It is stated that the "SKPP allows storage and timing covert channels to exist in the TOE[8] implementation. ….. Therefore, if a TOE has covert channels, and it is deployed in an environment which cannot tolerate the risk to the IT assets associated with those channels, then the applications configured to run on the TOE should be of sufficient assurance (e.g., through CC evaluation or some other means) that they can be trusted to not exercise the covert channels."

As an example, the one-way message channel that is provided by the separation kernel, used by the keyboard & mouse switch partition to transfer user input to the active partition, also provides a covert channel in the opposite direction. More specifically, as this is a reliable message channel with limited buffer capacity, the receiving application can signal the sending application by waiting to receive messages, in which case the sender will be unable to send new messages. In order for this covert channel to be of use in this specific case though, it would also require an application in the keyboard & mouse switch partition to forward the signaled information to

[7] This requires the malicious application to influence the SHM driver's sending of the interrupt signals (or gaining access to kernel mode so that it can send the interrupts itself) and that the collaborating application in the user partition is able to (directly or indirectly) observe the received interrupt signals.

[8] Target of evaluation (TOE).

another user partition. Given that the keyboard & mouse switch partition is a trusted partition, it is reasonable that it can also be trusted to not exploit the covert channel given proper assurance.

However, an important aim of our design is that the same level of assurance should not be required for the other partitions. If covert channels are present in the separation kernel, there is therefore a risk that they may be exploited. Such a risk can be mitigated by restricting the applications that are allowed to run in partitions. For instance, not allowing any additional software to be installed within the screen partition greatly reduces the risk of a Trojan application being installed within this partition. Clearly specified interfaces between partitions (e.g., only FramebufferUpdates are to be sent to the screen partition) also simplifies input validation and reduces the attack surface.

Although the SKPP allows the existence of covert channels, this in itself does not mean that a product evaluated according to the SKPP necessarily has covert channels. As such, one needs insight into the performed covert channel analysis in order to evaluate the risk posed by potential covert channels. Also, according to [4], the Security Target for the PikeOS separation kernel has been generalized into a Protection Profile draft, where the main change relative to the SKPP is the claim of absence of covert channels. Clearly, the absence of covert channels is a desirable attribute of a separation kernel which would simplify the certification of sensitive systems making use of it.

2.4 Potential additions and enhancements

In this section we will first discuss some possible additions to the basic functionality provided by the current prototype implementation before considering potential performance enhancements.

2.4.1 Additional functionality

The current prototype provides basic functionality, and can as such be enhanced in several ways. In particular, it is possible to create many application specific additions to increase user friendliness. We will here briefly discuss the more general possible additions.

2.4.1.1 Cut and paste (or other data transfer) between partitions

In order to enable information to be moved between partitions, cut and paste between partitions would be highly practical in many situations. Let us first consider the least problematic case, i.e., cut and paste from a lower confidentiality level to a higher confidentiality level.

Assuming information is allowed to flow from low to high, as in the Bell-LaPadula model, this functionality can be implemented using a one-way channel from low to high. This one-way channel would be similar to those used for the screen updates and keyboard/mouse input in the current prototype (i.e., based on shared memory or a message channel). As any reverse signaling would provide for a potential covert channel between the untrusted user partitions, the use of a of a shared memory solution without reverse interrupt signaling may be preferable (i.e., similar to the multiple buffer based mode discussed in Section 2.2.3). To reduce the risk of malicious

software being transmitted to the high partition, one might also want to restrict the format of what is transmitted (e.g., only allowing plain text up to a given length). Other mechanisms for transmitting data between partitions (i.e., not by cut and paste) may be provided in a similar manner, but cut and paste does have the simplifying characteristic that it occurs at a relatively low rate.

It should be pointed out that support for drag-and-drop type of cut and paste, between partitions, is not possible under the chosen design. This is due to the fact that the different user partitions, and the screen partition, have different mouse pointer instances. One option could be to have an icon representing each target partition on the desktop of the source partition. The user might then drag the data to this icon to send it to the corresponding partition. Otherwise, the user would simply have to use a keyboard shortcut or a menu selection (e.g., in the contextual menu of the mouse). After copying the data and selecting the target partition, the user would then switch to the target partition and paste the data (potentially also selecting which partition to paste from).

Allowing copy and paste from a higher classified partition to a lower classified partition clearly involves a higher security risk. This would generally require a trusted review and release mechanism. Such a review and release mechanism could be implemented in an intermediate partition on the path from the high to the low partition, ensuring non-bypassability. If only plain text is allowed to be copied between partitions, the release mechanism might in itself be implemented with high assurance on top of the minimal runtime. However, the review part of this mechanism would need to trust the screen partition to correctly display the content to the user. As such, the acceptable risk would need to be considered towards the assured integrity of the screen partition. In our current implementation, the screen partition has been considered to be untrusted but the contents of this partition may in principle be replaced with a higher assurance solution.

2.4.1.2 Change of window size/resolution

Although the user can change the size of the windows representing each partition in the current implementation, the desktop size/resolution of the corresponding user partition does not scale to fit the size of the window. Because there is no return data channel from the screen partition to the user partition(s), it is not possible for the screen partition to initiate such a change in the desktop size. One possibility would be to have a menu in the user partition where the user can select the resolution (i.e., desktop size) for that user partition. This then requires that the screen partition is able to handle such changes (e.g., a VNC viewer supporting the DesktopSize pseudo-encoding of the RFB protocol).

2.4.1.3 Additional device sharing

The implemented solution provides sharing of the keyboard, mouse and screen, which we consider to be the most fundamental functionality of such a solution. As previously discussed, the same solution/design can be generalized to devices that are used as either purely input or purely output devices. A device that is used for both input and output cannot be shared between partitions in the same manner however. For that reason, each disk, network interface, and so on is

only assigned to a single partition, meaning that device duplication is required when multiple partitions are in need for the same type of device. Although not all partitions may require network connectivity or persistent storage, an approach based on device duplication may be impractical for some usages. This is especially the case if a higher number of user partitions are to be supported, in particular on a mobile device.

Device duplication can be avoided if device sharing is enabled in a secure fashion, providing the assurance level required for the intended usage. Given a secure software solution for sharing a single device that is used for both input and output, multiple user partitions could be connected to the partition providing the device sharing. We are currently not aware of any certified solutions for this however. Therefore, if sufficient assurance cannot be provided for the security span of the entire system, one option could be to share devices between some partitions while another partition may use separate devices.

An encryption module positioned between a user partition and the partition with the device to be shared may potentially be used to share a network interface or a storage device. Also, a solution for sharing of a single disk between multiple partitions [5] has been demonstrated by LynuxWorks in collaboration with Wave Systems Corporation. Their solution makes use of multi-banded self-encrypting disk drives, where separately encrypted bands are associated with different partitions.

2.4.2 Support for Microsoft Windows

The current prototype implementation has made use of Linux as a guest operating system (i.e., within partitions), although the separation kernel in use also supports Microsoft Windows. If the prototype implementation is to provide native support for Windows as a guest operating system, it is required to port the Linux drivers and proxy software used within user partition(s) to Windows.

In principle, it is also possible to indirectly support Windows using the current implementation. This can be done by having a minimal Linux partition, with the SProxy and drivers, connected to the Windows partition by a (bidirectional) virtual network. The VNC server running in the Windows partition can then connect to the proxy running in the Linux partition, thereby being connected to the screen (i.e., VNC viewer), mouse, and keyboard.

2.4.3 Performance enhancements

As noted previously, the prototype implementation was created for demonstration and experimentation purposes. For a production type of system, at least if considering a more resource limited device, performance improvements would be required. We will in this section briefly discuss possible performance enhancements.

The main overhead in the prototype system is due to the inter-partition screen handling and having multiple partitions with separate operating system instances. We will address both of these issues in the following discussion.

A main motivation for using VNC as part of the screen handling in the prototype implementation was to reduce the implementation effort through the use of existing software. Although this may be acceptable for basic (e.g., office type) applications, a lower level approach would improve performance. This could potentially be achieved through sending serialized OpenGL commands, as discussed in [6], enabling rendering to be performed in hardware.

Several performance enhancements are also possible when using VNC. Part of the motivation for using a proxy based solution for the prototype was that it provides a way to provide indirect support for Microsoft Windows, without developing Windows specific software (as discussed in Section 2.4.2). However, in terms of performance, VNC software modified to operate directly over the one-way shared memory (i.e., using the SHM driver) would reduce overhead.

When using the SHM driver in the mode based on multiple buffering (i.e., without using interrupt signaling in the reverse direction), where the entire screen is sent in each update, it would also reduce overhead to have the VNC server not send updates when there have been no changes since the last update. (The option in the RFB protocol to request the entire screen, instead of the incremental update, is intended to be used when the client has lost its current state. The server therefore sends a new update even when there have been no changes since the last update.)

Depending on the usage scenario and the resources available, one might also further reduce overhead by reducing the rate at which screen updates are sent for non-active partition(s).

With regard to having multiple partitions with separate operating system instances, the obvious way to reduce overhead is to limit the footprint and running services within each partition to what is actually required. Given a separation kernel that supports dynamic switching between pre-defined schedules, one might also switch schedule when switching the active partition. This way, the keyboard & mouse partition (or a partition which it notifies) would also have privileges to select the current schedule, so that the active partition receives relatively more processor time while the other partitions are suspended or receive less processor time. This would especially be beneficial if many user partitions (e.g., classification levels) are to be supported or when considering devices with limited processing or battery power.

2.5 Applicability to touchscreen devices

The current prototype implementation has been implemented on a workstation type PC, but is equally applicable to a laptop form factor. Its applicability to a touchscreen device, such as a tablet, is intuitively less clear and therefore needs some further discussion. Our current implementation is not intended for use on a touchscreen device. The objective of this section is therefore to establish whether the same principal design would still be applicable.

In our prototype implementation we chose to display each user partition within a separate window. Although this might still be a good solution for a large touchscreen display, it would likely become cumbersome on smaller devices. On such devices, it may be better to only show a

single user partition on the screen at a time or, on a highly specialized device, to designate a specific area of the screen to each user partition (where each user partition may correspond to some specific application(s)). This implementation issue does not affect the overall design however. For the discussion in this section, it may for simplicity be assumed that only the currently active user partition is shown on the screen.

With a touchscreen device, the touchscreen is basically used as a pointing device similar to a mouse. The physical keyboard is typically eliminated, potentially being replaced by an onscreen virtual keyboard. As discussed previously, our design is applicable to devices that can be used as purely input or purely output devices. A touchscreen display may appear as a combined input and output device. However, this is typically not the case. As illustrated in Figure 2.3, a touchscreen display is a layered construction with a display in the back and a touchscreen in front [7]. Consequently, there is a separate touch controller and a separate video controller. Referring to the prototype design, the video controller will then be under the control of the screen partition (as before), while the touch controller is under the control of what we have previously referred to as the keyboard & mouse switch partition (referred to as the input switch partition in Figure 2.3).

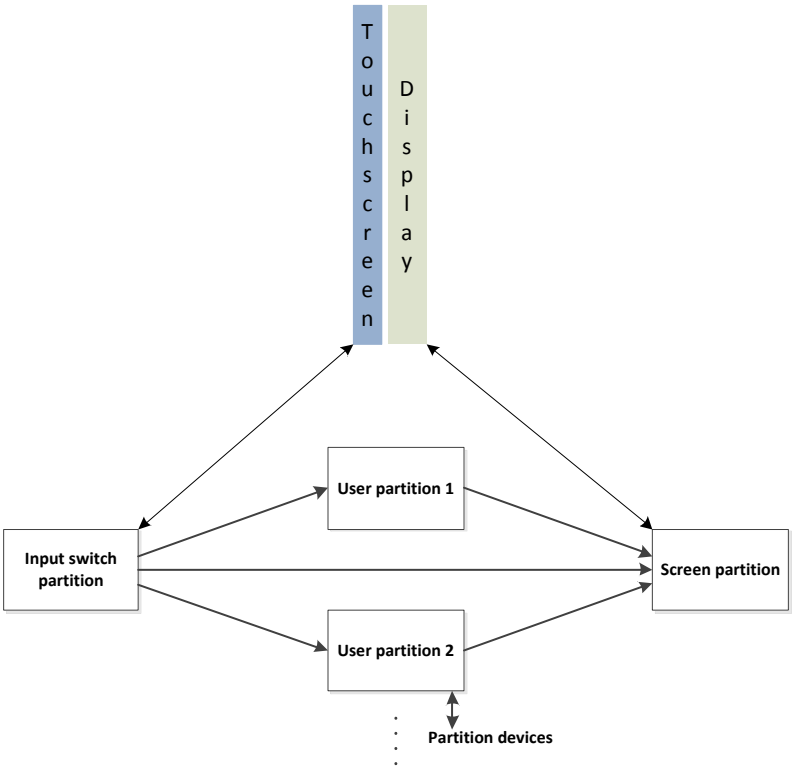

Figure 2.3 Incorporating a touchscreen display into the prototype design. (A cross-sectional view is used to illustrate the touchscreen display.)

Although the touchscreen and display can be included without changes to the design, the implications of lacking a physical keyboard must be considered. As previously described, our prototype implementation makes use of specific key combinations for selecting the active

partition. This might not be a preferable approach for a touchscreen device with a virtual keyboard. One reason for this is that the virtual keyboard is most easily displayed by the currently active partition. While the input switch partition would still be able to intercept touchscreen input provided by the user, the semantics of this input would depend on what is being displayed by the currently active partition. Such implicit trust in the currently active partition is not desirable. Although it would be possible to have the input switch partition to display the virtual keyboard, this would increase the complexity of this trusted partition and also provides less flexibility for user applications.

A better approach may therefore be to have some designated area(s) of the screen that is reserved for switching active partition. Such an area might be marked by imprinting on the surface of the screen or by print on the case (if the areas are located along the edge of the screen). Such an approach would fit nicely into the current design, as the input switch partition would only need to detect and intercept events within these specific areas of the screen. This may also be combined with a physical button, interconnected to the input switch partition, used in order to activate these areas for switching active partition. Alternatively one might use a physical button or sliding switch for switching partition altogether.

To summarize, it can be concluded that the prototype's principal design is also applicable to a touch screen device.

3 Example use cases

3.1 Handling multiple classifications – demonstration at the SOA Pilot

The prototype solution was demonstrated as part of the SOA Pilot at FFI in June 2011. The pilot showcased how a range of military operational systems can be interconnected, based on the principles of service orientation, and was played out as scenario story. The reader is referred to the main pilot report [8] for more details about the scenario and the pilot in general.[9]

As part of the pilot scenario, it was demonstrated how the prototype can be used for handling multiple security domains (i.e., classifications) on a single computer. A screenshot from the demonstration is shown in Figure 2.3, and the corresponding prototype configuration in Figure 3.2. The demonstrated configuration provided two desktops on the same screen, i.e., the red desktop marked Mission and the blue-green desktop marked National, each belonging to different security domains. This configuration is based on running the National system within the screen partition, while the Mission system runs in a separate user partition shown in a separate window. This configuration was chosen because information was assumed to be allowed to flow from the Mission system to the National system, but not in the opposite direction. As explained previously,

[9] There are also reports on other parts of the pilot, namely cross-domain information exchange using a guard [15], Web services [17], and semantic technologies [16].

the user can switch between the two levels using a keyboard shortcut and the Mission window can be moved or minimized when working within the National system.

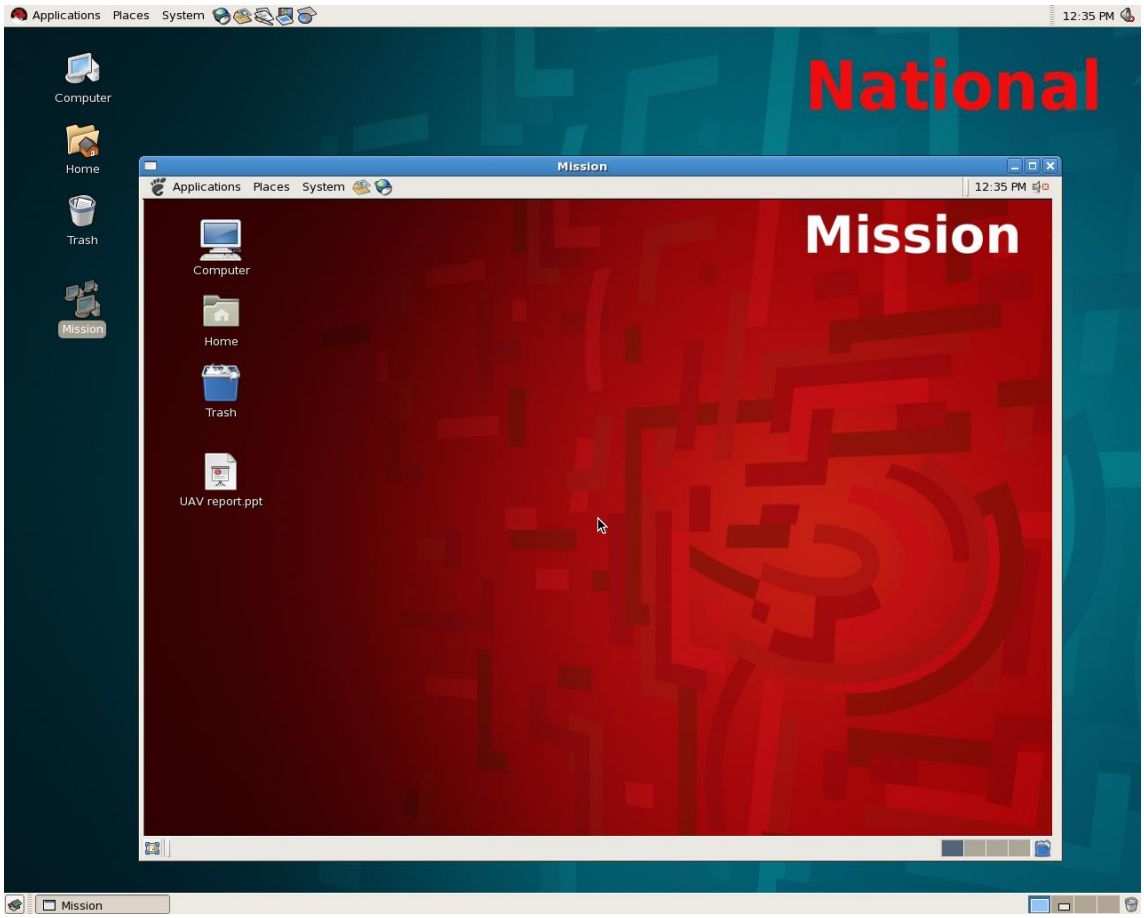

Figure 3.1 A screenshot from the prototype implementation as it was configured for the pilot. (The mouse cursor belonging to the National desktop is not shown, as the screen capture application excluded its own partition's mouse cursor when grabbing the screen.)

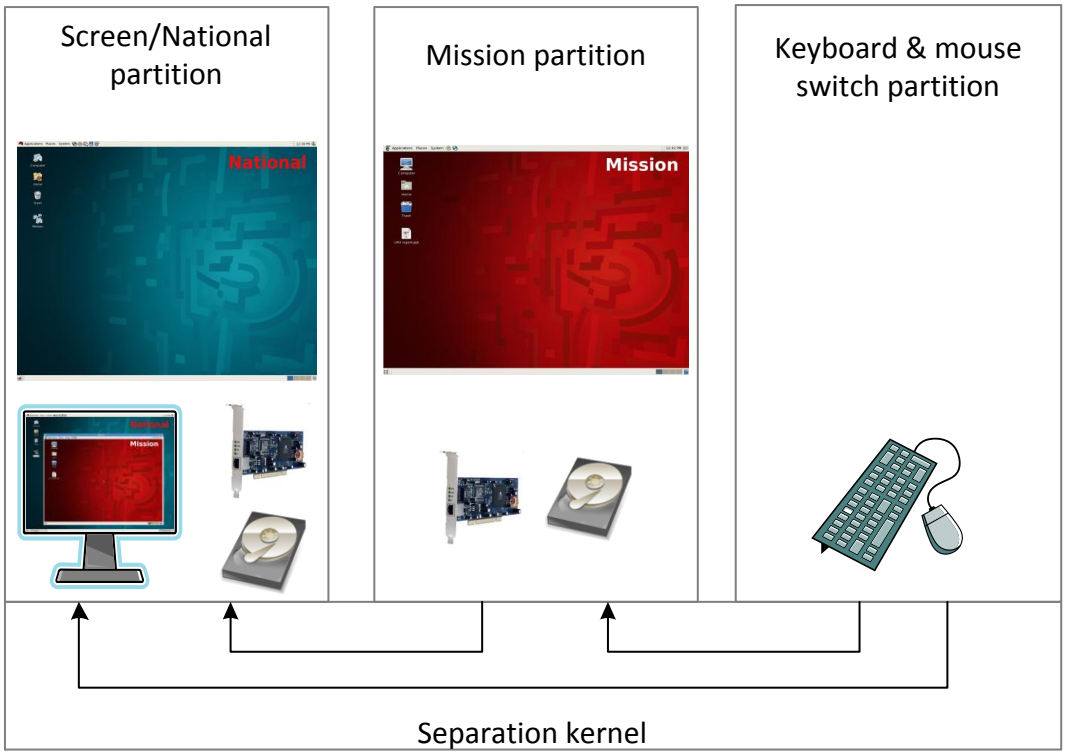

Figure 3.2 Prototype configuration used for the SOA Pilot. The Mission partition and the Screen/National partition were both assigned their own network interface and hard disk. An alternative configuration would have been to host the National system in a separate partition in the same way as the Mission partition.

In the pilot scenario, the prototype was demonstrated by using it to compare information belonging to the two different security domains. More specifically, the user had a coalition (i.e., Mission Secret) UAV image showing some suspicious objects. Wanting to compare this image with previous images from the same area, the user had used the national system to request a previous national (i.e., National Secret) surveillance image from the same area. The user received this previous image through e-mail within the National domain and compared it with the UAV image within the Mission domain, observing that the suspicious objects were not there previously. The user could now potentially take action on this new knowledge through interaction with the appropriate coalition entities using the Mission domain.

3.2 Providing separation for security critical or high risk applications (integrity protection)

Although our main focus has been on ensuring confidentiality when handling multiple classifications on the same computer, the prototype may also be used to partition a user system in order to protect the integrity of some part(s) of the system. This can be used to separate a part of the system that is deemed to have a high risk of being compromised (e.g., a web browser used for

accessing the Internet), in order to provide additional protection for particular parts of the system (e.g., an application with access to a critical database), or both.

In the case where a part of the system with high risk of being compromised is partitioned into a separate user partition, it may be desirable to run this from a non-writable source to ensure that it is recovered at system restart if it gets compromised. Still, in many situations, it is necessary to be able to apply security patches in an effective manner. It would therefore be desirable if another more protected partition, for instance during maintenance mode, was able to apply security patches and so on. We have not further explored this possibility however.

3.3 Supporting trustworthy binding of security labels

Another use case is found when considering the case for a solution to ensure that data is labeled with the correct confidentiality label. In order to have confidence in the confidentiality labels, it must be assured that classified data is not erroneously or maliciously labeled with a lower classification.

In this respect, consider a workstation with a high and a low partition (i.e., corresponding to a higher and lower confidentiality classification), where there is a one-way data channel from the low to the high partition.[10] Any information originating from the low partition is assumed to have a confidentiality classification of low or below. As such, when low confidentiality labels are attached to data within the low partition, one can be confident of not mislabeling high data.

The security of the cryptographic key used for signing the label and data is critical for such a solution. If this key was to be compromised, and end up within the high domain, it could be used to falsely label high data as low data, thereby leaking high data. Thus, although this key is used to label low data, we argue that the key itself needs to be protected according to the security requirements of the high data. Such protection may for instance be accomplished using a hardware security module connected to the low partition or by protecting the signature mechanism within a separate partition only accessible from the low partition.

It should be noticed that this scheme, in its basic form, does not provide a seamless way for downgrading high data to low as this would require the data to first be moved to the low partition (e.g., using a removable storage media). It does however effectively support a scenario where an operator makes use of information at the high level, in order to for instance send out some messages with a low classification. It could also enable a user at the high system to query some specific information at the low level or to control a low sensor (e.g., while being connected to a high network with a guard enforcing access to the low network).

[10] If there also is to be a one-way channel from the high to the low partition, this would need to go through a guard partition to ensure that high data is not released to the low partition.

4 Discussion

4.1 Separation kernel certification status

A separation kernel certified on a suitable hardware platform is a requirement for realizing the full potential of a system such as the one discussed in this report. At the time of writing, the only separation kernel that has completed Common Criteria evaluation is the INTEGRITY-178B Separation Kernel from Green Hills Software. This separation kernel has been evaluated to Evaluation Assurance Level (EAL) 6+ (High Robustness) on specific PowerPC platforms [9] [10] [11]. However, this hardware does not appear to be generally available.

There is also an ongoing EAL6+ evaluation of VxWorks MILS from Wind River, which was planned to be finished by the end of 2011 and as such may be expected to be completed any time soon. This evaluation also targets a PowerPC platform, and although this platform may be generally available the security target document has not yet been published. However, we expect this hardware platform to be better suited for an embedded type of solution (e.g., a guard) than an end-user system.

The INTEGRITY-178B and VxWorks evaluations have been performed claiming conformance to the U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness (SKPP) [3]. As of 2011 the SKPP has been "sunset" [12], meaning that no additional evaluations will be performed according to this profile. Although the evaluation of VxWorks MILS is allowed to finish, there will be no reevaluation of evaluated products on other hardware platforms (although minimal maintenance evaluations are permitted for up to two years). Furthermore, the U.S. National Information Assurance Partnership (NIAP) will now only focus on the lower evaluation assurance levels and issue protection profiles at EAL 1 and 2 [13]. Therefore, it cannot be expected that there will be evaluation of additional separation kernels, or evaluations on other platforms, according to the Common Criteria in the U.S. It is stated that separation kernels "continue to be sound design choices for security-critical systems" [13], but the Information Assurance Directorate will now focus on specific government systems making use of separation kernels instead of performing general evaluations.

As discussed previously, there is also ongoing work in order to perform a Common Criteria evaluation of PikeOS from Sysgo. However, we have no details on timeframes or specific platform(s). LynxSecure from LynuxWorks has also been developed to be evaluable at the highest levels of assurance. However, apparently due to the U.S. change of policy, an evaluation has not yet been performed.

4.2 The way forward

Although Common Criteria evaluations at these assurance levels are not automatically recognized between nations, the Common Criteria still represents the closest there is to a widely recognized security evaluation scheme for high assurance systems. Its wide recognition and defined process

also make it a relatively attractive option for industry, by providing predictability and recognition in a potentially large market.

Successful Common Criteria evaluation of a separation kernel on an x86 (Intel) platform would enable considerable reuse of legacy software on high assurance separation kernel based systems. This would also increase the applicability of the MILS architectural approach to many information assurance challenges.

The significant difficulties of evaluating a separation kernel according to the SKPP on commodity workstations (where a commodity workstation is defined as a typical x86 based desktop platform from vendors such as Dell, HP, or Apple) are discussed in [14]. These difficulties are to a large extent due to the complexity of the hardware and firmware (including BIOS) used in such systems. Nevertheless, the same report recommends continuing the support for application of separation kernels on commodity workstations as a path toward higher levels of robustness.

It should be noted that the complexity of commodity workstations is a general problem facing all attempts to run secure systems on these platforms, and not a problem specific to separation kernels. However, the assurance level often targeted with separation kernels is so high that the hardware/firmware platform becomes the weak link reducing the attainable overall assurance.

There are several potential ways forward. One option is to certify separation kernels on a simplified hardware that is suitable for running an end-user system such as the one described in this report. At a minimum, this requires a hardware platform suitable for connecting a keyboard and mouse (or alternatively a touchscreen), one or (preferably) more network interfaces, a relatively simple graphics processor/interface, and persistent storage (e.g., disk drive(s)). An x86 platform would be a significant advantage with regard to reuse of legacy/COTS software, but is strictly not a requirement for a Linux based system or more special purpose applications where a PowerPC platform can be used.

Certification of a separation kernel on the ARM platform could also be an interesting option, given its support by more recent commodity operating systems (e.g.., Android, Windows 8, and iOS) due to the widespread of smartphones and tablets. In addition to supporting the x86 and ARM platforms, Android is also supported on the MIPS platform.

In order to reduce evaluation costs, one might also consider lowering the assurance level to EAL 5 (as opposed to EAL 6), as this would likely be sufficient for many applications at least when viewed in context with additional protective measures and procedures. A lower assurance level could especially be advantageous if wanting to maintain certifications on relatively recent hardware. This could also be combined with a higher certification (possibly on simpler hardware) that is upgraded on a less regular basis for applications with higher assurance requirements.

To share the risks and costs of evaluation, a collaborative effort between for instance industry and NATO nations (or the EU) might provide for an economically more attractive way forward. A wider range of certified target platforms would clearly increase the applicability of separation kernels, and could as such help establish a more self-sustaining market.

MILS separation kernels provide less functionality than what is found in traditional MLS operating systems, thereby simplifying evaluation. For MILS to provide the functionality of an MLS operating system, however, evaluation of additional components is also required. However, for many applications the additional functionality provided by MLS operating systems is not required, and the simplicity of separation kernels provides flexibility to adapt separation kernel based systems to different usage applications (where the additional functionality of an MLS operating systems may not match anyway). In this sense, separation kernels can be seen as having the characteristics of a disruptive technology where the functionality provided can be expected to increase over time as complementary components become available.

References

[1] T. Gjertsen and N. A. Nordbotten, "Military operational systems in field - multiple levels of security," FFI-rapport 2009/01137, 2009.

[2] T. Richardson, "The RFB Protocol Version 3.8," 2009.

[3] Information Assurance Directorate, "U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness Version 1.03," 2007.

[4] H. Blasum, K. Degen, B. Langenstein, K. Müller, A. Nonnengart, M. Paulitsch, R. Schwarz, W. Stephan and S. Tverdyshev, "Scavenging a Common Criteria Protection Profile," [Online]. Available: http://www.embedded-world.eu/fileadmin/user_upload/call_for_papers/1316175796-15b3025ff2e3ee0158328040aa5a80ee.doc. [Accessed 24 January 2012].

[5] LynuxWorks, "LynuxWorks and Wave demonstrate first use of multi-banded self-encrypting drive combined with secure virtualization," 2011. [Online]. Available: http://www.lynuxworks.com/corporate/press/2011/wave.php. [Accessed January 2012].

[6] R. DeLong, "Protection Profile for MILS Console Subsystem in Environments Requiring High Robustness, Version 0.14G (DRAFT)," 2010.

[7] K. Maxwell, "Writing drivers for common touch-screen interface hardware," 2005. [Online]. Available: http://www.eetimes.com/design/embedded/4006455/Writing-drivers-for-common-touch-screen-interface-hardware. [Accessed February 2012].

[8] R. Rasmussen, B. J. Hansen and A. Eggen, "Experiment Report: SOA Pilot 2011," FFI-rapport 2011/02407, 2012.

[9] NIAP, "Validated Product - Green Hills Software INTEGRITY-178B Separation Kernel, comprising: INTEGRITY-178B Real Time Operating System (RTOS), version IN-ICR750-

0101-GH01_Rel running on Compact PCI card, version CPN 944-2021-021 with PowerPC, version 750CXe," 2008. [Online]. Available: http://www.niap-ccevs.org/st/vid10119/.

[10] NIAP, "Validated Product - Green Hills Software INTEGRITY-178B Separation Kernel, comprising: INTEGRITY-178B Real Time Operating System (RTOS), version IN-ICR750-0402-GH01_Rel (Version 4.2) running on Compact PCI card, version CPN 944-2021-021 w/PowerPC, v750CXE," 2011. [Online]. Available: http://www.niap-ccevs.org/st/vid10362/.

[11] NIAP, "Assurance Continuity - Green Hills Software INTEGRITY-178B Separation Kernel, comprising: INTEGRITY-178B Real Time Operating System (RTOS), version IN-ISP448-0100-SK_LMFWPCD2_Rel running on JSF PCD System Processor CCA, version 437140-007 w/PowerPC, v7748," 2009. [Online]. Available: http://www.niap-ccevs.org/st/vid10119/maint200/.

[12] National Information Assurance Partnership, "Archived U.S. Government Approved Protection Profile - U.S. Government Protection Profile for Separation Kernels in Environments Requiring High Robustness, Version 1.03," [Online]. Available: http://www.niap-ccevs.org/pp/PP_SKPP_HR_V1.03/. [Accessed 8 February 2012].

[13] NIAP, "SKPP Sunset Q&A," [Online]. Available: http://www.niap-ccevs.org/announcements/SKPP Sunset Q&A.pdf. [Accessed February 2012].

[14] Systems and Network Analysis Center Information Assurance Directorate, "Separation Kernels on Commodity Workstations," 2010. [Online]. Available: http://www.niap-ccevs.org/announcements/Separation Kernels on Commodity Workstations.pdf.

[15] R. Haakseth, "SOA Pilot 2011: Demonstrating secure exchange of information between security domains," FFI-rapport 2012/00117, 2012.

[16] J. Halvorsen and B. J. Hansen, "Integrating Millitary Systems using Semantic Web Technologies and Lightweight Agents," FFI-notat 2011/01851, 2011.

[17] K. Lund, F. T. Johnsen, T. H. Bloebaum and E. Skjervold, "SOA Pilot 2011: Service infrastructure," FFI-rapport 2011/02235, 2012.