# Security-Enhanced Darwin: Porting SELinux to Mac OS X

Christopher Vance *SPARTA, Inc.*
Todd C. Miller *SPARTA, Inc.*
Robert Dekelbaum *SPARTA, Inc.*
Andrew Reisse *SPARTA, Inc.*

## Abstract

Security-Enhanced Darwin (SEDarwin) is a port of access control elements derived from the National Security Agency's Security Enhanced Linux (SELinux) to Darwin, the Open Source core of Apple's Mac OS X operating system. Mac OS X combines both Open Source and proprietary technologies into a widely-used, production-quality UNIX-based workstation. Because Darwin is available under an Open Source license, it is possible to add mandatory access controls while still maintaining compatibility with the large base of user space applications, proprietary graphical components, and programming frameworks. While it was straightforward to port many of the SELinux components, Darwin's unique architecture meant significant new engineering was necessary in order to provide SELinux-style access controls consistently across the entire system. This paper describes the component technologies in Darwin, the areas of significant new development, and provides comparisons to SELinux.

## 1 Introduction

Apple's Mac OS X provides a number of security mechanisms, comprising both user space and kernel components[1]. The operating system kernel includes support for:

- UNIX-style discretionary access control checks: based on user and group identifiers.
- Mach port rights: besides controlling interprocess communication (IPC) channels, Mach port rights can protect tasks, threads, memory ranges, processors, and other devices.
- Audit support: generated audit logs use Sun's Basic Security Module (BSM) token stream format.
- BSD-style process accounting: records system accounting information for all processes.
- Encrypted virtual memory: AES algorithm used to encrypt virtual memory pages when stored on disk.
- Access Control Lists (ACLs): per-file fine-grained permission model implemented using extended attributes.
- Kauth: Apple's own in-kernel flexible, extensible security architecture for securing file access.

In addition to the kernel security components, Mac OS X provides a flexible user space security model. To hide some of the system complexity, Apple has added a layer of simpler security APIs to manage Keychain Services, secure network communication, provide certificate and trust services, and manage Authorization Services. Integral to this model, the Authorization Services APIs communicate with a user space Security Server to determine whether operations and accesses are permitted.

One of the shortcomings of Apple's security architecture is the lack of consistent, centralized access control. While there are discretionary access control (DAC) checks for BSD components, Mach port rights for the Mach subsystem, ACLs and Kauth controls for file access, and Security Services for user space services, there is no overarching design or single configuration point. Each component is configured and operates separately. The TrustedBSD Mandatory Access Control (MAC) Framework was added to Darwin to address this by adding access controls consistently throughout the kernel and by providing APIs for user space policy decisions. This Framework makes it possible to implement a single access control policy that provides protection for login sessions, processes, files, pipes, network communication, System V and POSIX IPC, and Mach IPC.

With the addition of the TrustedBSD MAC Framework, a full suite of kernel security mechanisms becomes available. Since the MAC Framework was designed to be orthogonal to existing security mechanisms, it does not replace any vendor access controls or grant privilege beyond what the base system permits. By design, the MAC Framework can only further restrict system access. The MAC Framework is similar in concept and construction to the Linux Security Modules (LSM) Framework. Each extends the base operating system security architecture by allowing pluggable security modules to further constrain applications.

With the addition of the MAC Framework, SELinux technologies can be incorporated to achieve consistent access control coverage. Apple's Mac OS X operating system uses a unique blending of component technologies that substantially distinguishes it from FreeBSD, Linux, Solaris, and other UNIX-like operating systems.

This architecture led to many challenges when porting the SELinux technologies.

This paper will discuss Apple's architecture, the application of the TrustedBSD MAC Framework to this architecture, the significant technical development that was necessary to secure the Mach IPC and IOKit subsystems of Apple's operating system, and the adaptation of the SELinux components for use on the Apple system.

## 2 Background

### 2.1 Mac OS X

Apple's Mac OS X is a commercially available operating system providing a graphical user interface on top of a UNIX core. Darwin, the Open Source core of Mac OS X, is composed of a variety of technologies, including facilities derived from the Mach 3.0 microkernel[2], operating system services based on FreeBSD 5, high performance networking facilities, and multiple integrated file systems. Like most UNIX systems, the Darwin kernel is engineered for stability, reliability, and performance[3, 4]. Layered on top of the UNIX core are a collection of graphics routines, a windowing system, a user interface, and an application object model, as shown in Figure 1. Together, these are the components that help make Mac OS X a commercially viable desktop system.

Apple makes the Darwin source code available under the Apple Public Source License, permitting third party inspection, modification, and extension of the operating system. This makes Darwin an attractive platform for operating system research, since researchers have the complete source code without the burden of non-disclosure restrictions. Through careful modification of the Darwin core, operating system behavior can be changed while still allowing the higher level application interfaces (Aqua, Quartz, Carbon, etc.) to function, thereby preserving the user experience. It was especially important to maintain compatibility with commercially developed and supported applications such as Microsoft Office, Adobe Digital Imaging Software, professional-grade audio and video processing software, and licensed (DRM-enabled) multimedia playback that help distinguish Mac OS X from other UNIX-based systems.

Darwin's kernel uses a Mach 3.0-derived core to manage processor resources (CPU and memory), handle scheduling, enforce memory protection, and implement interprocess communication. Mach provides most of the core OS services, including memory protection, preemptive multitasking, and advanced virtual memory.

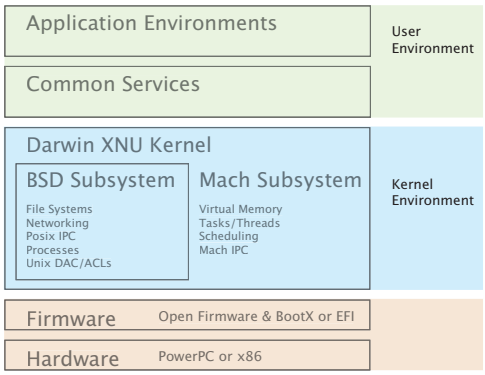


*Figure 1: Apple's OS X Architecture*

### 2.2 DTOS

The Distributed Trusted Operating System (DTOS) project was an enhanced version of the Mach 3.0 microkernel that included a Mach security server to secure process management, file objects, the network stack and Mach IPC. The project was a joint effort by the National Security Agency (NSA) and Secure Computing Corporation. This work developed a prototype secure Mach microkernel that provided strong, flexible security controls with good performance and application compatibility[5, 6]. The DTOS microkernel made enhancements to the Mach 3.0 design, the same version of Mach upon which Darwin is based. DTOS used the Lites[7] Mach server as its BSD subsystem. Unlike the BSD kernel code in Darwin, Lites runs as a user-level single server; it is not kernel-resident as it is in Darwin.

The DTOS security server utilized a Type Enforcement[8] policy that was a precursor to the Type Enforcement used by FLASK (described below). While the DTOS code was not directly used in SEDarwin, it was examined to help understand the complexities of introducing security into a Mach-based operating system. The goals of SEDarwin and DTOS are quite similar: develop a prototype that can be used to demonstrate that strong, useful security features can be introduced into an operating system without sacrificing desirable features such as performance or utility.

### 2.3 FLASK and SELinux

The Flux Advanced Security Kernel (FLASK) is an operating system security architecture that provides a flexible and fine-grained MAC architecture[9] using a generalized form of Type Enforcement. It was developed as a security-enhanced version of the Fluke microkernel-based operating system[10].

NSA's Security-Enhanced Linux[11] (SELinux) is a

version of the Linux kernel that includes a version of the FLASK security server. Policy enforcement logic is implemented using the interfaces provided by the LSM framework[11]. A wide range of security models can be implemented as security servers without requiring changes to any other component of the system. The FLASK architectural abstractions of the security server and access control vector cache are preserved in the SELinux implementation.

SELinux provided a logical base in securing the Darwin operating system because it is a mature, well-tested, and highly recognized mandatory access control implementation. Several SELinux-powered Linux distributions are currently in evaluation under LSPP/EAL4. The existence of a thriving community of SELinux users that continues to develop and enhance SELinux was also important. By showing that the same technology that powers SELinux can be used on Darwin, we hope to provide greater operating system security to a new set of users. Furthermore, the SEDarwin project has shown that the basic security architecture of SELinux is applicable to more than just the Linux platform, even though Darwin has features, such as Mach IPC and IOKit, that are not present on Linux.

### 2.4 MAC Framework

The TrustedBSD MAC Framework provides an extensible access control environment for the FreeBSD and Darwin kernels, allowing loadable kernel modules to instrument the kernel access control model. The MAC Framework's primary job is to act as the glue between the Darwin kernel and the policy modules. The MAC Framework manages a set of loadable security modules that collectively enforce the system's security policy.

The interface between policy modules and the Framework resembles the interface between the kernel and the Framework. Before the kernel uses an object, it consults the MAC Framework. The Framework calls the corresponding function (entry point) for each loaded module, composes the return values from each module, and returns a sensible error code to the kernel; all modules must return a non-error value for an access check to "succeed."

#### 2.4.1 Label Management

The MAC Framework is responsible for maintaining the association of security labels with kernel objects. Kernel subsystems notify the MAC Framework of relevant kernel object life cycle events, which are then forwarded to MAC policies. Policies maintain per-object state via a pointer in the security label, which may be used by the policy to reference statically allocated objects, per-object allocated memory, or reference-counted structures.

Security labels follow the existing allocation and usage cycles that the kernel uses to manage objects. For most kernel objects, this life cycle consists of allocation, initialization, use, and deallocation. While the Framework manages the association of labels with kernel objects, each policy defines the syntax and semantics of its own label data.

The MAC Framework passes labels to and from user space in the form of text strings. The Framework provides internalization and externalization routines to allow policy modules to convert between internal label representations and externally-visible strings. User space libraries provide policy-agnostic interfaces to read and update labels.

## 3 Moving SELinux to Darwin

Darwin includes a substantial quantity of code derived from the Open Source FreeBSD operating system. Leveraging this common source code heritage, the SEDarwin project has ported the MAC Framework to the Darwin platform, permitting Darwin's access control model to be similarly extended. Once the MAC Framework was available on Darwin, it was relatively easy to port SELinux components to the Darwin system.

For the kernel components, the porting effort primarily consisted of implementing the interface layer between the policy and the MAC Framework (analogous to LSM's "hooks") and adapting Linux-specific interfaces to their Darwin equivalents. This included system-specific constructs such as locking, memory allocators, and audit/logging. Since RCU-style locking is not available on Darwin, we had to use read/write locks for the access vector cache, similar to what previous versions of SELinux used. Aside from these things, we did not have to modify the inner workings of the security server itself, which is a testament to its portability.

Likewise, the user space SELinux components were easily adapted to Darwin. We added a compatibility library to emulate C library functions used by SELinux that are not present in the BSD libc and modified libselinux to use the MAC Framework library API and system controls (sysctls) in place of accesses to the SELinux pseudo-filesystem, selinuxfs. The associated SELinux tools to build the policy and change file and process contexts required few, if any, changes. By making as few changes as possible to the SELinux code base, we are able to update SEDarwin to newer revisions of the SELinux components with minimal

effort.

Although the core technology was simple to port to Darwin and was up and running on top of the MAC Framework in very little time, it was far from a complete solution. While the Darwin kernel was quite similar to FreeBSD and Linux in its BSD kernel subsystem, it also contained several components that were quite foreign, namely the IOKit device driver layer and the Mach IPC subsystem. These two components are not present in other modern operating systems and are worth discussing in detail. A solution to provide reasonable access controls for IOKit USB and Firewire devices proved to be straight-forward, but the Mach IPC subsystem required considerable effort to secure.

## 4 Controlling IOKit

Linux, FreeBSD, and Mac OS X each provide rich device driver APIs and support for encapsulating device drivers in loadable kernel modules. To provide complete access control coverage of the Darwin kernel, it was necessary to secure Apple's driver subsystem, IOKit. Since IOKit is significantly different than Linux's device driver implementation, this required evaluation and implementation, not a simple porting effort. Traditional UNIX-based operating systems often link hardware devices to a virtual file system (such as `/dev/`) and manage access via file system permissions. Darwin does not manage devices in this way, so an alternate approach must be taken to prevent access to hardware resources, especially removable devices.

Rather than secure each of the APIs individually, the SEDarwin project chose to manage loadable modules primarily via security policy. Like other loadable kernel modules, device drivers are present in the file system (typically in the `/System/Library/Extensions` directory), so file system access controls may be used to constrain their use. The kernel extension daemon ("kextd") and the kernel extension loader ("kextload") can be restricted to only load modules and device drivers that have the appropriate label.

For two classes of devices, it made sense to provide more flexible access controls. Both USB and Firewire are widely used to dynamically provide additional hardware at run time ("plug-and-play"). Mandatory access controls are typically not flexible enough to accommodate the wide variety of pluggable hardware that the systems support. Instead of creating a large number of additional APIs to support fine-grained access controls, the SEDarwin project modified the device driver matching system to give preference to a new filtering device driver (actually two drivers: one for USB and one for Firewire).

This new filtering driver encapsulates the device probe data and consults the MAC Framework to determine if the device should be permitted. If access is denied, the filtering device driver attaches to the device, consuming it and making it unavailable to the standard drivers. Since the filtering driver provides no interfaces to users, it effectively isolates the device from doing harm to the system. If access is permitted, the filtering device driver does nothing. In this case, the hardware probe continues and the device can be matched by a proper driver.

## 5 Controlling Mach IPC

The Mach IPC kernel subsystem is the most used messaging service on Darwin; thousands of messages per second are passed between applications, the window system, and the kernel. Mach IPC is essential to secure, but there was no corresponding subsystem on FreeBSD or Linux on which to base the access controls. The complexity of the Mach implementation and its interdependence on the BSD subsystem meant a great deal of effort was necessary to add consistent SELinux-style access controls. In securing the Mach kernel subsystem, there are three major areas to be covered: traps, messages and ports, and servers.

The entry point for Mach kernel services from user space is the trap, or system call. In Darwin, system calls with a negative number are Mach traps (positive system calls are used by the BSD subsystem). Compared to UNIX, Mach provides very few traps since most services are implemented by Mach servers that are accessed via Mach messages. Mach traps are used to implement message passing, virtual memory controls, BSD process ID to Mach task mapping, high resolution timers, IOKit, semaphores and locking primitives.

The fundamental IPC mechanism provided by Mach is message passing: a task sends a message to a "port," which may be received by another task or by the kernel. Mach ports are unidirectional; any reply to a received message requires a second port. Unlike Berkeley sockets, Mach messages are structured; they can contain several types of data specially interpreted by the kernel in addition to raw data. Messages can contain port rights, granting another task access to a port. Address spaces can also be shared via messaging. The receiving task receives mapping for the memory region and can access it directly, without requiring extra copies as generally happens in unstructured IPC systems. Figure 2 shows a high-level diagram depicting Mach IPC.

The Mach operating system provides other services, such as device abstraction, virtual memory, scheduling, etc., but those services are accessed via messages sent to kernel ports. The other services provided by the Mach kernel can be treated as user space servers for security purposes.

## 5.1 Native Mach Security

Mach security uses a capability-like model where ports have associated rights. Port rights protect access to the port and prevent an arbitrary Mach task from using a Mach port, even if it knows the port's "name." When a task possesses a port right, that task is permitted to perform the associated access on the port. A task can transfer a port right to another task by sending it a Mach message and attaching the right.

Mac OS X defines three basic port rights: send rights allow the holder to send messages, receive rights allow message receipt, and send-once rights allow the holder to send exactly one message to a port. Port send and send once rights can be generated by the task that holds the port's receive right. A task may also create a "port set" containing one or more receive rights. Port sets allow a task to receive messages from multiple ports without using a separate thread for each port, similar to how the select system call can be used with Berkeley sockets.

All port rights held by a task were either received in Mach messages, returned by other Mach system calls (e.g. `mach_task_self`), provided to the task at creation (the bootstrap namespace server is provided this way), or provided to a task through external use of its task port. Since Mach ports are unidirectional and only have a single listener, each port has a single receive right (which also connotes ownership of the port). However, multiple senders are permitted; all owners of send rights can send messages to the port.

When a Mach task is created, a port send right is returned (the receive right for the task is always owned by the kernel). This port send right permits the owner to start and stop the task, kill the task, manipulate the memory space and otherwise administer the task. These permissions exist outside the BSD permission model, the vendor implementation only requires local administrator access. The SEDarwin project must include additional non-bypassable security mechanisms to prevent misuse of Mach IPC services.

## 5.2 Implementing Mach Access Control Primitives

The Mach port rights model means that possession of a right could grant access to all operations on the

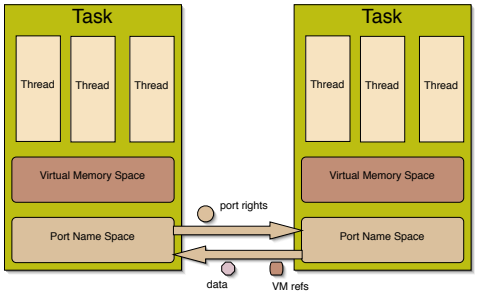


*Figure 2: Mach IPC Communication*

object. While this can be adequately controlled on a consumer desktop system by managing administrator access, it is insufficient for a trusted operating system. SEDarwin enhances the capability model by separating object access permissions from rights ownership. The MAC Framework adds labels to kernel objects and adds access control checks for Mach message sending and receiving, port right transfers (both send and receive) as well as relabel operations for ports and tasks. These additional controls on communications between tasks are desirable to control information flow between applications, as well as to control the types and nature of communications between applications; in particular a security policy may wish to attach a security label to tasks, messages, and ports.

Kernel changes made to support the MAC Framework for Mach IPC were fairly non-invasive, only affecting a few of the data structures and a small number of functions. The additions to the Mach portion of the Darwin kernel follow a model similar to the MAC Framework implementation in the BSD subsystem. Labeling operations are inserted into object creation and deallocation routines, and access decision calls are inserted into object usage routines.

Changes to the Darwin BSD subsystem were also necessary. Mach tasks and BSD processes must have separate labels because the Mach part of the kernel cannot acquire BSD-side locks. However, labels for Mach tasks and BSD processes need to be kept synchronized, as there really is only one subject (the Mach task_t and the BSD struct proc both include an opaque pointer to the other).

The policy-agnostic label structure also presents some implementation challenges compared to systems that assume a particular (often small or at least fixed-size) label format. Because policies differ in their label formats, the policy itself must handle label memory management. This means that duplicating a label is a potentially ex-

pensive operation compared to copying a small, fixed size security identifier. In addition, the MAC Framework does not currently permit policies to store pre-computed security decisions directly in the kernel object for later use as many common policies (Type Enforcement, MLS, capability models, etc.) might otherwise do. Policies can, however, implement their own caching mechanisms to speed up label computations or other operations. The access vector cache used by SELinux is one example of this.

## 5.3 Label Handles

The MAC Framework normally communicates labels to user space programs using text strings. For complex, security-aware servers wishing to perform their own internal access checks, performing repeated label to text conversions can be expensive. We introduce the "label handle" as an alternative to such repeated conversions: a kernel object with an attached label. In SEDarwin, label handles are represented to user space programs as Mach ports. The label handle may be created from a string representation or it may be computed based on a subject, object and service name. A task may also request the label handle of an existing port by using the labels trailer as described below.

## 5.4 Application Interfaces for Mach Access Control

A new kernel-resident Mach server has been added to handle security-specific requests. The new security server provides label operations and generic access control checks. Label operations supported by the server include the conversion of labels between text and internal formats as well as the creation of new label handles. Generic access checks, similar to interfaces exposed by `selinuxfs` in SELinux, provide a general way for a user process to query an arbitrary access control decision from the system's security policies. The functions typically take as arguments a task (usually the caller's), a subject label, an object label, a service name and a permission or method specific to that service. A boolean decision is returned.

### 5.4.1 Mach Message Trailers

Darwin has added a message component called the "message trailer" to the traditional Mach message structure. The trailer is located after the end of the message body and is used to include additional (out of line) information about the sender of a message. Two additional trailer types, `MACH_RCV_TRAILER_LABELS` and `MACH_RCV_TRAILER_AV`, were added to support security-aware programs. These extended trailers may be used alone or in combination with any other trailers supported by Darwin.

The `MACH_RCV_TRAILER_LABELS` trailer returns the sender's security labels in the form of a label handle. The caller may use the label handle to request access decisions from the security server or convert the handle to text form. For example, the bootstrap namespace server could ensure only a privileged process can examine the namespace created by a different loginwindow process. The returned label handle may be deallocated using the standard Mach port API functions.

The `MACH_RCV_TRAILER_AV` trailer provides a single access decision as a boolean value to the caller, computed by the loaded policies using subject, port, subsystem, and routine name arguments.

When the extended trailers are requested by a receiving task, calls are made to the loaded security policies and the result is copied out. Composition of these decisions is performed in the same manner as all other access controls managed by the MAC Framework: an operation is allowed only if all individual policies allow it.

Unlike the stock Darwin trailer fields, the additional trailers are not generated from existing data. The kernel must make additional calls to the MAC Framework to fill in the trailer data. Also, label handles (like regular Mach ports) must be released or destroyed when they are no longer needed. Since these additional trailers are only provided upon request, there is no performance penalty for unmodified callers.

An alternate approach, used by DTOS, is to add a modified `mach_msg()` trap for security-aware servers. Instead of requesting security information via message trailers, it can be passed back directly via the trap. We chose to use the message trailers as the changes required were smaller and less invasive than adding a new trap.

### 5.4.2 Using Security Trailers with MiG

The Mach Interface Generator (MiG) is similar to rpcgen on UNIX systems. MiG reads a specification of data types and method prototypes and generates client and server stubs that copy the arguments and a method ID into a buffer and send it as a Mach message. Both kernel and user space services make use of MiG. For example, the standard Mach subsystems for virtual memory, devices, and task management all use MiG-format messages. On Darwin, many user space systems, such as the namespace server, notification server, and pasteboard server use MiG-formatted messages as well. Using MiG whenever practical greatly reduces the development overhead of implementing messaging protocols.

This central role made it an ideal location to add additional access control support. MiG has been extended to support MAC Framework-specific information in the message trailers by providing an access decision or policy labels to the service function.

### 5.5 Adapting the SELinux Policy Language

Once support for Mach object labeling and access control was added to the MAC Framework, the SELinux policy language had to be extended to support the new Mach access controls. A new SELinux security class and appropriate permissions were introduced for Mach ports:

```
class mach_port
{
    relabelfrom
    relabelto
    send
    recv
    make_send
    make_send_once
    copy_send
    move_send
    move_send_once
    move_recv
    hold_send
    hold_send_once
    hold_recv
}
```

By default, port labels are derived from the task that created the port. A `type_change` rule in the policy can override this behavior. Ports can also be relabeled with the `mach_set_port_label()` kernel call. It is also possible for a TrustedBSD MAC Framework policy module to label kernel ports based on the type of kernel object associated with them (e.g. task, clock, memory object).

Mach servers register themselves using a specific subsystem identifier. Each method contained in the subsystem is numbered relative to the subsystem id. When an IPC message is sent to a method, there is no information in the message to indicate which subsystem contains the method. To work around this problem, SEDarwin keeps a mapping of method numbers to the subsystem that contains them. From this information we can compute the SELinux class that represents the subsystem as well as the associated permission bits. Using this information it is possible to query the security server for an access decision. To preserve compatibility with SELinux, the Mach IPC mapping is stored in a separate file that is loaded along with the binary policy.

## 6 Policy Rules

The system is nearly complete: the operating system kernel has been updated to include support for mandatory access control policies and the SELinux policy module has been ported and updated. However, without an appropriate policy configuration and rules, the system is no more secure than it was before.

During initial porting of SELinux controls and development of SEDarwin-specific extensions, only a basic policy was required in order to test system functionality and correctness. Now that the SEDarwin project has matured, a more robust and usable policy is required to make SEDarwin a viable, usable system. Rather than build a completely new policy, the Tresys-developed Reference Policy has been used as a starting point, as many high-level portions of this policy will be directly usable on SEDarwin.

There are, however, significant changes required to make the Reference Policy functional on SEDarwin. Our initial approach was to attempt to treat Darwin as just another Linux distribution, a "distro" in Reference Policy parlance. Because of the extensive differences in basic OS functionality however, this approach proved to be untenable. Therefore, the Tresys Reference Policy has been utilized as a starting point and we have reworked significant portions of the base policy for use with SEDarwin.

Many Linux-specific portions of the policy base have been removed as they have no close counterpart in Darwin. Darwin-specific policy modules have been grouped together in a "darwin" directory. The "selinux" base module has been modified in order to support SEDarwin's use of `sysctl` as opposed to `selinuxfs`. The "files" and "filesystem" modules also required modification. Many other Linux-specific policy modules (alsa, anaconda, apt, kudzu, ddcprobe, prelink, etc) have been entirely deactivated and some have been rewritten in order to achieve a fully functional and mediated Darwin system.

The Darwin model for system startup is significantly different from the Linux/BSD init-style model and required modification of the SELinux "init" module in order to allow system services to transition to their proper domains for execution. Darwin uses `launchd` rather than `init` to provide not only system startup services, but also system namespace mappings for Mach IPC and on-demand service provision (much like `inetd`). Due to this combined functionality, significant work to the "init" module was required to properly mediate system startup and service provision activities.

Policy modules to support Mach IPC mechanisms have been provided to allow mediation of these facilities which are not present in Linux. A new base policy module, "mach," has been developed to provide access controls for Mach messaging fundamentals such as control of messages between senders and receivers, transfer of Mach port rights and memory regions, etc. Interfaces for mediation of MiG services at the application method level are also supported by the SEDarwin policy, allowing for very granular access control of MiG-based applications.

## 7 Conclusion

While some aspects of Apple's architecture made it challenging, it was possible to adapt the SELinux components to apply consistent, fine-grained access controls across the complete operating system. SEDarwin can provide robust security while still allowing access to closed-source business tools. By starting with the mature SELinux kernel module and porting the TrustedBSD MAC Framework from FreeBSD to Darwin, it was possible to quickly secure the well understood kernel subsystems and focus development effort on the pieces that distinguish Darwin from other operating systems. With the addition of new access controls for Darwin's Mach IPC and IOKit device driver layers and by adapting the Reference Policy to the Darwin platform, we are able to provide strong, consistent security across the entire Mac OS X operating system.

## 8 Acknowledgments

This material is based upon work supported by DARPA under SSC SAN DIEGO Contract No. N66001-04-C-6019. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA or SSC.

## 9 Availability

This software is freely available for download on the SEDarwin web site:

`http://www.sedarwin.org/`

## References

[1] A. Singh, *Mac OS X Internals: A Systems Approach*. Addison Wesley, June 2006.

[2] K. Loepere, "Mach 3 kernel principles," 1992.

[3] "Apple Developer Connection: Mac OS X," Apple Computer, Inc., http://developer.apple.com/macosx/.

[4] "Mac OS X: An Overview for Developers," Apple Computer, Inc.

[5] S. E. Minear, "Providing policy control over object operations in a mach based system," in *Proceedings of the Fifth USENIX UNIX Security Symposium*, June 1995, pp. 141–156.

[6] D. Olawsky, T. Fine, E. Schneider, and R. Spencer, "Developing and using a policy neutral access control policy," in *Proceedings of the New Security Paradigms Workshop*, September 1996.

[7] J. Helander, "Unix under mach: The lites server," 1994.

[8] W. E. Boebert and R. Y. Kain, "A practical alternative to hierarchical integrity policies," in *Proceedings of the Eighth National Computer Security Conference*, 1985.

[9] R. Spencer, S. Smalley, P. Loscocco, M. Hibler, D. Andersen, and J. Lepreau, "The Flask Security Architecture: System Support for Diverse Security Policies," in *8th USENIX Security Symposium*. Washington, D.C., USA: USENIX, Aug. 1999, pp. 123–139.

[10] B. Ford, M. Hibler, J. Lepreau, P. Tullmann, G. Back, and S. Clawson, "Microkernels meet recursive virtual machines," in *Operating Systems Design and Implementation*, 1996, pp. 137–151.

[11] P. A. Loscocco and S. D. Smalley, "Integrating Flexible Support for Security Policies into the Linux Operating System," in *Proceedings of the USENIX Annual Technical Conference*, June 2001.