ACAPS – An Access Control Mechanism to Protect the Components of an Attack Prevention System

Joaquín García, Sergio Castillo, Guillermo Navarro, Joan Borrell
{jgarcia,scastillo,gnavarro,jborrell}@deic.uab.es
Information and Communications Engineering Dept.
Autonomous University of Barcelona, Bellaterra , Spain

Summary:

Current research in *Intrusion Detection Systems* (IDSs), targeted towards preventing computer attacks, is mainly focused on improving detection and reaction mechanisms, without preserving the protection of the system itself. This way, if an attacker compromises the security of the detection system, she may be able to disarm the detection or reaction mechanisms, as well as delete log entries that may reveal her actions. Given this scenario, we introduce in this paper the use of an access control mechanism, embedded into the operating system's kernel, to handle the protection of the system itself once it has been compromised by an attacker. We also show an overview of the implementation of such mechanism on a research prototype, developed for GNU/Linux systems, over the *Linux Security Modules* (LSM) framework.

Key words:

Network Security, Access Control, Attack Prevention, Intrusion Detection Systems, Linux Security Modules

Introduction

Recent network attacks are deploying distributed and coordinated techniques, which open the possibility to perform more complex attacks, such as distributed denial of service or coordinated port scans. These techniques are also useful to make their detection more difficult and, normally, these attacks will not be detected by exclusively considering information from isolated sources of the network. Likewise, Network Intrusion Detection and Response Systems also benefit from a distributed implementation. Different components of the system may look for different attack evidences, in order to detect this new kind of attacks.

We are currently working on a decentralized approach for achieving this distribution. Our solution uses a tuple space to communicate the different components within and with each other. Sensors, or middle-level analyzers place data (e.g. alerts) into a distributed tuple space, and higher-level analyzers consume those tuples to perform a detection and reaction process based on alert correlation [5].

The design of our prevention system has three main goals. The first goal is to obtain a modular architecture composed by a set of independent entities. These entities collaborate to detect when the resources where they are lodged are becoming part of an attack against the network where they are located or against a third party network. Once detected, they must be able to prevent the use of their associated resources to finally avoid their participation on the detected attack. The second goal is to achieve a complete independent relationship between the different components which form these cooperative entities. In this case, we distribute these components according to the needs of each resource we want to disarm. These two first objectives have been solved and discussed in [5].

The third goal is to obtain a system able to fulfill intrusion tolerance [4]. The system itself must maintain acceptable, though possibly degraded, service despite attacks in parts of the system, be them at network, application or system level. In order to achieve this third goal, we started out doing research on protection mechanisms to handle the security and strength of our prevention system's components. As a result of our current work, we present in this paper the development of ACAPS (which stands for An Access Control Mechanism to Protect the Components of an Attack Prevention System), a protection module integrated into the kernel of our research prototype's operating system, GNU/Linux, and implemented over the Linux Security Modules (LSM) framework [11].

The protection mechanism behind ACAPS consists of building a complementary kernel access control scheme, to handle the protection of the system itself once it has been compromised by an attacker. To do this, it intercepts and cancels unlawful system calls launched by the attacker. Thus, even if the attacker gains administrator permissions, she will not achieve her purpose. This security enhancement is solved without having to recompile the kernel, and with a high degree of flexibility and portability when compared to other proposals for the GNU/Linux kernel, such as [7] and [8].

Another important feature of our approach is that it allows to enforce the components' protection in an independently fashion. Hence, even if one of the components is compromised, the rest of them can continue to work in a trusted way. Although it will not be always possible to achieve the full independence between all the components, we show a first proposal for our work, by considering different protection levels. This way, we manage as much as possible the fulfillment of this requirement, and we offer the administrator in charge of the system the ability to perform component protection at different levels, such as application level, communication level, etc.

The rest of this paper is organized as follows. Section 2 describes the main properties of our proposed attack prevention system, as well as the main components to protect it through ACAPS. Then, we take a closer look to the protection scheme behind ACAPS in Section 3. Section 4 introduces a first implementation of ACAPS on our research prototype, through the Linux Security Modules (LSM) framework, as a kernel based access control mechanism. An evaluation concerning the efficiency, security, and usability of ACAPS is then presented in Section 5. Finally, Section 6 closes the paper with a list of conclusions and future work.

2. System Overview

The main purpose of our prevention system is to detect and react to coordinated or distributed attacks. By means of a set of cooperative entities which are lodged inside the network, the system avoids the use of network resources to perform coordinated attacks against third party networks. The aim of this system is not only to detect in-coming attacks against these entities, but also to detect when these nodes are the source of one of the different steps of a coordinated attack to avoid it.

Our approach is based on gathering and correlating information held by multiple sources. We use a decentralized scheme based on message passing to share alerts in a secure communication infrastructure [5]. The information exchange between peers is intended to manage a more complete view of the whole system. Once achieved, one can detect and react on the different actions of the corresponding attack.

As shown in Figure 1, each node of the architecture is made up of a set of analyzers (with their respective detection units or sensors), a set of alert managers (to perform alert processing and manipulation functions), and a set of local reaction units (or effectors). These components, and the interactions between them, are described in the following subsections.

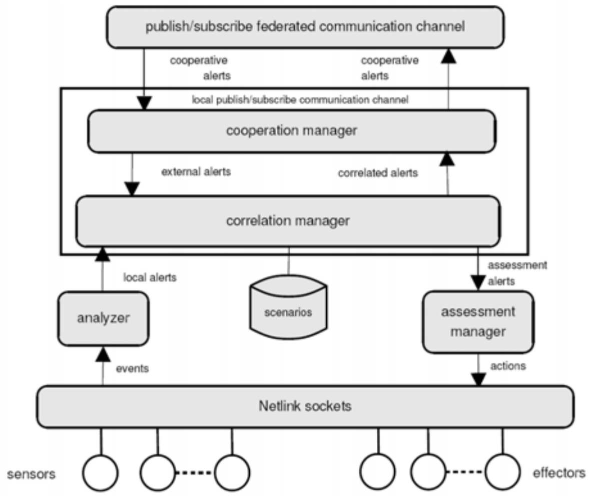

Fig. 1. Main components of each node

2.1 Analyzers

Analyzers are the local elements in charge for processing audit data. They process the information gathered by associated sensors, implemented as operating system's kernel modules, to infer possible alerts. Their task is to identify occurrences which are relevant for the execution of the different steps of an attack and pass this information to an alert correlation manager. The interesting occurrences are local alerts.

Each local alert is detected in a sensor's input stream and exchanged as an IDMEF message [2]. The *Intrusion Detection Message Exchange Format* (IDMEF) is intended to be a standard data format that automated intrusion detection systems can use to raise alerts about events that they report as suspicious. It allows analyzers and managers to assemble very complex alert descriptions. In accordance with the IDMEF format, each local alert has a unique classification, and a list of attributes with their respective types, to identify the analyzer that originated the alert, the time the alert was created, etc.

All possible classifications and their respective attributes are known by all system components (i.e. sensors, analyzers and managers) and all analyzers are capable of exchanging various instances of local alerts of one or more types.

Once formulated each local alert as an IDMEF message, using *XML* syntax, each local alert is published to the local communication channel via the publish/subscribe system, *xmlBlaster* [9]. A publish/subscribe system consists of brokers and clients that are connected to brokers. The brokers themselves form the infrastructure (notification service) used for routing the notifications. Clients can publish notifications and subscribe to filters that are matched against the notifications passing through the broker network. If a broker receives a new notification, it checks if there is a local client subscribed to a filter that matches this notification. If so, the message is delivered to this client. Additionally, the broker forwards the message to neighbor brokers according to the applied routing algorithm.

On the other hand, the exchanging of information between sensors and analyzers is based on a different communication mechanism, since sensors are working as modules in kernel space, and analyzers as daemon processes in user space. This communication is efficiently solved through the use of *Netlink sockets* [3], a Linux specific mechanism that allows us to perform communication between kernel modules and user space processes via the well known primitives from the socket treatment, and providing us transparency with the buffering mechanisms.

2.2 Managers

The use of multiple analyzers and sensors together with heterogeneous detection techniques increases the detection rate, but it also increases the number of alerts to process. In order to facilitate this process and to reduce the number of false negatives our architecture provides a set of cooperation and correlation managers, which performs aggregation and correlation of both, local alerts (i.e., messages provided by the node's analyzers) and external messages (i.e., the information received from other collaborating nodes). Just like for the communication between the analyzer and the correlation manager (c.f. Section 2.1), the communication between managers is also implemented through the use of the publish/subscribe system xmlBlaster [9].

The basic functionality of the cooperation manager is to cluster alerts that correspond to the same occurrence of an action. It also provides mechanisms to represent the information contained in the various alerts belonging to this cluster, applying aggregation and fusion techniques [5]. Each cooperation manager registers its interest in a subset of local alerts, by subscribing itself to the corresponding channel.
Similarly, the cooperation manager also registers its interest in related external alerts, and its interest in local correlated alerts. Once subscribed to these three filters, the alert infrastructure will notify the subscribed managers of all matching alerts. All the notified alerts are processed and, depending on the clustering and synchronization functions, the cooperation manager can publish some global and external alerts.

On the other hand, the main task of the correlation manager is the implementation and execution of the weighted alert correlation algorithm described in [5]. The correlation manager operates on the global alerts published by the local cooperation manager (i.e. it registers its interest in these alerts). Then, the alert infrastructure will notify of all matched alerts. Each time a new alert is received, the correlation mechanism finds a set of action models that can be correlated in order to form a scenario leading to an objective. At last, it includes this information into the CorrelationAlert field of a new IDMEF message and publishes the correlated alert.

The correlation manager is also responsible for reacting on detected security violations. The algorithm used is based on the anti-correlation of actions, to select appropriate countermeasures in order to react and prevent the execution of the whole scenario [5]. As soon as a scenario is identified, the correlation mechanism looks for possible action models that can be anti-correlated with the different actions of the incoming scenario, or even with the goal objective.

The set of anti-correlated actions becomes the set of countermeasures available for the observed scenario. The definition of each anti-correlated action contains a description of the countermeasures which should be invoked (e.g. hardening the security policy). Such countermeasures are included into the Assessment field of a new IDMEF message, and published through the alert infrastructure. Another manager, the assessment manager, will register and revoke its interest in these assessment alerts.

Once notified, the assessment manager performs a post-processing of the received alerts before sending the corresponding reaction to the local response units (or effectors). Just like for sensors and analyzers (c.f. Section 2.1), the communication from the assessment manager to the effectors (also implemented as Linux kernel modules) has also been implemented through the use of netlink sockets.

3. Protection Mechanisms

As described in the previous section, the entities of our platform cooperate to detect if the resources, where they are lodged, are taking an active part of a coordinate attack.

As it happens with a traditional IDS, with the proper manipulation of the processes associated to each node, an attacker could bypass the detection mechanisms. Thus, the intruder could make its way to hide the local part of the attack from the node. Furthermore, she could generate false alerts in order to cause a malfunction of the whole platform.

This problem leads to the need for introducing a protection mechanism on the different components of each node, keeping with their protection and mitigating or even eliminating any attempt to attack or compromise the platform and its operation. This way, even if an attacker compromises the security of the system, she would not be able to disarm the detection and reaction mechanisms.

Given the inherent characteristics of the design of our prevention platform, and according to [7], we consider the following two protection mechanisms: the *auto-protection* carried by the elements of each node, and the protection of the elements carried by the kernel of the operating system. In the first case, each component is responsible for its own protection, using mechanisms such hiding its processes, mobile agent systems, or cryptographic techniques associated to the system logs. Most of the current proposals in this field, such as [3, 14, 15], are inefficient against some attacks. For example, when the intruder is in a privileged position, she may interact without restriction with the components through the underlying operating system.

Therefore, the cancellation of processes associated with the detection system, or the deletion of logs, show the problems of these methodologies. This problem relies in two facts. First, the existence of privileged users (administrators) in most of the current operating systems, that can freely interact with the system. And second, the delegation of part of the protection to the operating system access control mechanism, which does not consider that an attacker could gain privileged user permissions from a bug or security failure.

In the second case, the kernel of the operating system provides the proper protection mechanisms, detached from the detection and reaction system. Protection is achieved by incorporating an access control mechanism into the kernel system calls. This way, one may allow or deny a system call based on several criteria such as the identifier of the process making the call, parameters of the call, etc. The kernel's access control allows to eliminate the notion of trust associated to privileged users, delegating the authorization for the execution of a given system call to the internal access control mechanisms. In addition, and contrary to the previous *auto-protection* mechanisms, it provides a unified solution, avoiding the implementation of different specific mechanisms for each component.

3.1 Proposed Scheme

In order to protect the components of our platform we propose an access control mechanism integrated into the kernel of the operating system. This way, even if an attacker gains administrator permissions, she will not be able to generate actions attempting on the node. Each unlawful system call to the components is intercepted and cancelled by the access control.

This methodology also allows us to provide a second level of protection. The mechanism provided by the kernel and the modularity based on components allows to enforce the compartimentalization principle [10].

This principle is based in the segmentation of a system, so several components can be protected independently one from another. This ensures that even if one of the components is compromised, the rest of them can operate in a trusted way. In our case, several components from the node can be executed as processes. By specifying the proper permission based on the process ID, we can limit the interaction between elements of the node. If an intruder takes control of a process associated to a given component (through a buffer overflow, for example), she will be limited to make the system call for this given process.

In our proposal the compartimentalization principle is used as follows. In each node we assume that the execution of the analyzers is isolated from the cooperation manager. The protection at kernel level avoids that potentially dangerous system calls (such as *killing* a process) could be produced from one component against another one.

For example, if a malicious user gains the control of a process associated to an analyzer, she will not be able to abort the execution of the cooperation manager.

Even so, it is not always possible to achieve a complete independence between the components. There is a need to determine which system calls may be considered as a threat when launched against an element from the node.

This requires a meticulous study of each one of the system calls provided by the kernel, and how can they be misused. On the other hand, we have to define the access control rules for each one of these system calls. Despite its complexity, we can consider three basic protection levels to classify the system calls:

- Critical process protection: comprises actions that can attempt on the proper execution of the processes associated to a node, either by interaction over them by signals, or the manipulation of the memory space.

  Some examples are: execution of a new application already in memory, cancellation or manipulation of the address space and process traces, etc.

- Communication mechanisms protection: comprises all the processes that allows an attacker to modify, generate or eliminate all kinds of messages interchanged between the node elements.

- Protection of files associated to the components: comprises all spiteful actions addressed to the files used by the components of the node, such as executable, configuration, or log files.

4. Implementation

In this section we outline the current implementation of ACAPS. In accordance with the protection scheme proposed in Section 3.1, it consists of a kernel based access control mechanism, and its development has been done over the *Linux Security Modules* (LSM) framework for *GNU/Linux* systems [11].

The LSM framework does not consist of a single specific access control mechanism; instead it provides a generic framework, which can accommodate several approaches. There are several hooks (i.e. interception points) across the kernel that can be used to implement different access control strategies. Such hooks are: *Task hooks*, *Program Loading Hooks*, *Filesystems Hooks* and *Network hooks*.

These LSM hooks, can be used to provide protection at the three levels commented in the previous section. Furthermore, LSM adds a set of benefits to our implementation.

First, it introduces a minimum load to the system when comparing it to kernels without LSM, and does not interfere with the detection and reaction processes. In the second place, the access control mechanism can be composed in the system as a module, without having to recompile the kernel. And third, it provides a high degree of flexibility and portability to our implementation when compared to other proposals for the Linux kernel, such as [7] and [8], where the implementation requires the modification of some features of the original kernel 2.6.x.

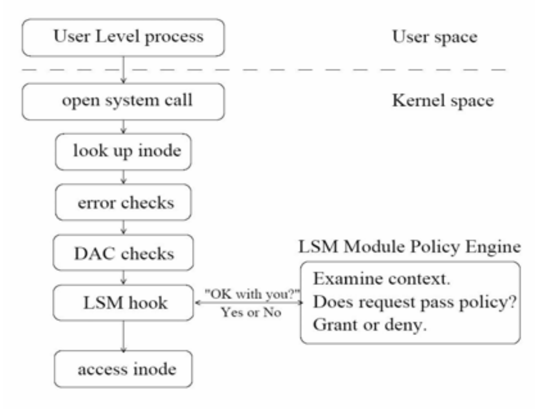

Fig. 2. Linux Security Module (LSM) Hook

The LSM interface provides an abstraction, which allows the modules to mediate between the users and the internal objects from the operating system kernel. To this effect, after accessing the internal object, the hook calls the function provided by the module and which will be responsible to allow or deny the access. This can be seen in Figure 2. There, a module registers the function to make a check over the inode*s* of the filesystem. At the same time, LSM allows to keep the discretionary access control (DAC) provided by Linux by standing between the discretionary control and the object itself. This way, if a user does not have permissions in relation to a given file, the DAC of the operating system will not allow the access and no call to the function registered by the LSM will be made. This architecture reduces the load of the system when compared to an access control check centralized in the operating system call interface, which always gets used for all the system calls.

The node components will be allowed to make operations only permitted to the system administrator (such as packet filtering, process or application cancellation, etc.). This implies that the system processes associated to the components will be executed by the root user. On the contrary, if we associate the processes to a non privileged user, the discretionary access control of Linux will not allow the execution of some specific calls. The internal access control mechanisms at the kernel is based in the process identifier (PID) that makes the system call, which will be associated to a specific component. Each function registered by an LSM module, determines which component is making the call from the PID of the associated process. It then, applies the access control constraints taking also into account the parameters of the system call. So, for example, a given component can access its own configuration files but not configuration files from other components.

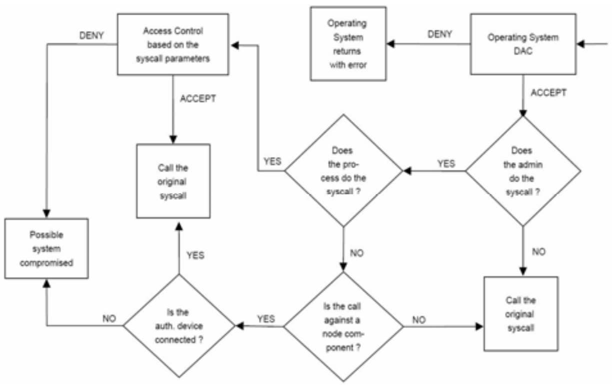
DENY
Access Control based on the syscall parameters
Operating System returns with error
DENY
Operating System DAC
ACCEPT
ACCEPT
YES
Does the pro-cess do the syscall ?
YES
Does the admin do the syscall ?
Call the original syscall
Possible system compromised
YES
NO
NO
NO
Is the auth. device connected ?
YES
Is the call against a node com-ponent ?
NO
Call the original syscall

Fig. 3. Access control example through ACAPS

An important issue in the implementation is the administration of the access control mechanisms and the management of each one of the nodes. As described in the previous section, the administrators should not be able to throw a system call, which may suppose a threat to the node. This prevents an intruder to do any harm to the node even if she could scale its privileges to the administrator ones. This contrasts with the administration of the node, if an administrator cannot interact with the components of the node, she will not be able to carry on any management or configuration process and activities. To solve this problem, we have introduced a temporal authentication process based on a cryptographic USB token [1]. While the device is connected to the system, the administrator will be able to hold the indispensable privileges to manipulate the node. When the device is retired, the access control enforcement will come to its normal operation. Figure 3 shows how a function, registered by ACAPS, allows the modification of a configuration file.

5. Evaluation

In order to show the applicability of our proposal, we evaluate in this Section the implementation of ACAPS from two different points of view. We first examine its efficiency, and we then discuss the security enhancement that it offers. This way, we can finally conclude whether the performance penalty introduced by ACAPS, in contrast to the security enhancement offered by, is acceptable to be deployed on a real system.

5.1 Efficiency

The efficiency of ACAPS can be decomposed into the following three classical perspectives: performance, scalability and modularity.

First of all, the overhead introduced by ACAPS can be considered as the impact in the system's performance introduced by the LSM framework and the access control hooks implementation. On one hand, and as pointed out in [11], the overhead of the LSM framework in the system is minimal compared to the standard Linux kernel, about 0-5%. On the other hand, a set of tests over our access control hooks implementation, through LMbench [6] and the OProfile system profiling support offered by the 2.6 series of the Linux kernel, also reveals that the performance impact is minimal.

From the point of view of scalability, our proposed protection mechanism can be extrapolated to other new components, by considering its own environment and the interaction with it in terms of access control. Thus, this protection methodology reinforces the modularity providing an easy and generic way to incorporate new components in the global architecture, without considering particular strategies to protect each one of them. In the same scope, the inclusion of new components guarantees a low degradation performance scalability, since the LSM framework and the access control implementation do not introduce complex computation.

5.2 Security

To be deployed on a real system, a protection mechanism must be both efficient and secure. However, efficiency and security are often a contrary criteria. Hence, we must also evaluate ACAPS from the perspective of the security enhancement that offers. The security introduced by our approach enforces the components' protection in an independently manner, and ensures that even if an intruder gets the administration privileges in a protected machine, she will not interact with the components of our prevention system. In contrast with other approaches that could offer similar solutions, the proposed access control provides us an additional level of security, reducing the impact caused if an attacker gets the control of one component. For reinforcement security purposes, the authentication token acts as a complement of the access control, translating the security authentication for administration to a physical element. This idea allow us to avoid some kind of logical attacks focused on getting the rights of the administrative entity, such as stolen password or buffer overflows.

6. Conclusions

Intrusion Detection Systems (IDSs) are currently very popular in corporate networks and successfully used by security staff and administrators to prevent, detect and react to security threats. Nevertheless, they may become a vulnerable point in the whole system. If the IDS or one of its components is compromised by an attacker, the consequences for the security of the system it is protecting may be disastrous. Protecting the components of an IDS is not easy, specially when dealing with distributed IDSs, made up of different components distributed over a network and with lost of communications involved.

In this paper we have presented an access control mechanism specially suited for a distributed prevention and reaction system. The distributed systems is made up of several components such as sensors, analyzers, managers, etc. that may be distributed in a network. We provide a solution for the protection of the components by making use of the LSM system in the Linux kernel.

The mechanism we have developed, called ACAPS, works by providing and enforcing access control rules at system calls. It is based on a protection module integrated into the operating system's kernel, providing a high degree of modularity and independence between components.

Summarizing the evaluation of our first implementation, we conclude that the low performance penalty introduced by ACAPS, the security enhancement offered by, as well as its scalability and modularity, is more than acceptable to be deployed and utilized on a real system.

We can also conclude that ACAPS offers a good degree of transparency to the administrator in charge, since the access control is integrated inside the operating system's kernel, and it does not interfere directly with user space's processes. At the same time, the token based authentication for administration purposes, provides a transparent and secure management of the platform.

As future work we are considering to continue our study about attack and intrusion tolerant mechanisms, to address the security of our proposed architecture from a wider and more global point of view.

Acknowledgments

This work was supported by the Spanish Government project *TIC2003-02041*, and the Catalan Government grants *2003FI126* and *2005BE77*.

References

[1] Aladdin Knowledge Systems. eToken – USB Token Authentication-Device, http://aladdin.com/etoken/, 2005.

[2] H. Debar, D. Curry, and B. Feinstein. Intrusion detection message exchange format data model and extensible markup language (xml) document type definition. Internet draft, January 2005.

[3] G. Dhandapani and A. Sundaresan. Netlink sockets overview. Technical report, The University of Kansas, September 1999.

[4] P. Esteves-Verissimo, N. Ferreira Neves, and M. Pupo-Correia. Intrusion-Tolerant Architectures: Concepts and Design. Technical report, Computer Science Department, University of Lisboa, Portugal, April 2003.

[5] J. García, F. Autrel, J. Borrell, S. Castillo, F. Cuppens, and G. Navarro. Decentralized publish-subscribe system to prevent coordinated attacks via alert correlation. In *6th International Conference on Information and Commu-nications Security*, October 2004.

[6] L. McVoy, and C. Staelin. LMbench – Tools for Performance Analysis http://bitmover.com/lm-bench/, 1998.

[7] T. Onabuta, T. Inoue, and M. Asaka. A Protection Mechanism for an Intrusion Detection System Based on Mandatory Access Control. In *13th Annual Computer Security Incident Handling Conference*), Toulouse, France, June 2001.

[8] A. Ott. The Role Compatibility Security Model. In *7th Nordic Workshop on Secure IT Systems*, Karlstad, Sweden, November 2002.

[9] M. Ruff. XmlBlaster: message oriented middleware. http://xmlblaster.org/xmlBlaster/doc/whitepaper/whitepaper.html, 2000.

[10] J. Viega, and G. McGraw. *Building Secure Software - How to Avoid Security Problems the Right Way*. Addison-Wesley, September 2002.

[11] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux Security Modules: General Security Support for the Linux Kernel. In *11th USENIX Security Symposium*, San Francisco, California, August 2002.