Proceedings of the 2004 IEEE International Conference on Communications (ICC 2004), Paris, France, June 2004.

Realizing a flexible Access Control Mechanism for Active Nodes based on Active Networking Technology

A. Hess, G. Schäfer
Telecommunication Networks Group, Technische Universität Berlin
Einsteinufer 25, 10587 Berlin, Germany
Email: [hess,schaefer]@tkn.tu-berlin.de

Abstract— **This paper presents a model and mechanism for flexible access control of loadable on-demand services in an active network, using code origin authentication and runtime supervision. During the development of the access control mechanism, we strongly focused on keeping the mechanism as efficient as possible, and to realize a modular design which allows to dynamically upgrade and configure the mechanism, making use of the active networking technology itself, while at the same time ensuring that mandatory security checks cannot be circumvented. Each service has to pass initial checks before it can be executed on an active node. Our approach provides access control that is dynamic, extensible and efficient, realizing a *demand-driven supervision* which avoids supervision of those actions that do not need to be supervised. Specific access control modules are realized as active services and activated when needed.**

Finally, we present results that have been achieved with a first prototype developed for the active networking platform AMnet [4] (Active Multicast Network) which are very promising.

I. Introduction

The programmable networking technology provides a framework for flexible and rapid service creation on top of existing networks. It uses enhanced nodes - so called active nodes - within the network for the provision of specific services. An active node is able to execute services which are loadable on-demand from a remote service module repository, and thus to enhance its functionality in a flexible manner. Application examples for this technology are media transcoding services, network security enhancing services [6] or overlay networks [1]. A service can either be a user space process and consist of one or more service modules which execute inside an Execution Environment (EE), or an extension to the Operating System (OS).

In this paper, we present an access control mechanism which exploits the possibilities provided by an active networking environment. Consequently, parts of the access control mechanism can be downloaded dynamically on an active node like a normal active service and the mechanism pre-adjusts itself corresponding to the upcoming security requirements which are specified in security policies. We show that the active networking technology provides an adequate framework for the realization of customized security applications, in our case a customized access control.

Challenging security issues arise due to the programmability and the location (directly connected to the Internet) of active nodes. The active node itself could be the victim of an attack, or the active node itself could be exposed to attack other end systems. Today, the following three approaches (and combinations of them) are considered to solve the above mentioned problem:

1) Identifying the code origin: Only code from which the origin and its integrity is known can be executed on an active node.
2) Prediction of code semantics: More or less a theoretical approach, as it is known that it is impossible to construct a generic algorithm to determine what a program does unless the programs are written in specific programming languages (for example [7]).
3) Restricting the functions the service can execute by runtime supervision.

Besides the arguments given above, the prediction technique is also impractical for active networks, as this would require the disclosure of the source code. Consequently, our solution is a combination of approaches one and three. Hence, only authenticated code can be executed on an active node and additionally, an access control mechanism, which is the subject of this paper, supervises the services at runtime. Additionally, we focused on the following requirements during the development phase of the architecture. The intention behind our access control mechanism is a security requirement specific supervision of the active services which are running on a node. The particular property of our access control mechanism is that it is widely realized as active networking services. For the realization of the access control mechanism as active networking services we focused on a dynamically extensible and adjustable architecture (new security requirements, new security holes, etc.). Another focal point was the independency of the programming language of the services (C, Java, etc.). Nevertheless, it should be possible to individually customize the authorizations of a service and an active node. Finally, we directed our attention towards efficiency. Therefore, the architecture is widely realized in kernel space and we followed a demand-driven supervision approach.

"Demand-driven" supervision means the supervision can be tailored at any time according to the aggregated supervision requirements of all active services on a node. Hence, the access control mechanism configures itself corresponding to the actual security requirements. The intention behind this is to secure the system while at the same time not degrading the performance to an unnecessary extent through the supervision of authorized actions which do not need to be supervised. Consequently, the access control mechanism consists of a basic core functionality which is responsible for the configuration of the mechanism, and of several supervision modules - so called access control kernel modules (ACKM) - which can be (de-) activated dynamically and which are realized as active services. Each supervision module is responsible for the supervision of exactly one resource or OS-service.

Considering the above mentioned aspects, measurements made with a first prototype show that a demand-driven access control mechanism can gain significant efficiency improvements. Additionally, the presented approach is very easily extensible for new security requirements.

The paper is organized as follows. The next section gives a short overview of the state of the art and afterwards we introduce and explain the concept of our access control mechanism. Section IV discusses the conducted experiments including the achieved results. Finally, the last section summarizes the paper.

II. Related Work

The Ariel Project [10] and the Naccio Project [3] present an access control mechanism for mobile Java code. Both solutions provide a mechanism to protect and control the local resources that can be accessed by Java programs only.

The Seraphim Project [2] also uses customized policies and provides the possibility to change or update them dynamically. In contrast to this, with our approach the policies are static during the execution of a service, but the structure of the enforcement engine can change, resulting in a smaller performance degradation.

Other approaches try to predict code semantics but there are difficulties in the realization of those systems [9], [8]. Also special programming languages are realized as [7] but service development poses difficulties as these languages are restricted, the languages must be learnt and consequently code reuse-ability is not given.

Summarizing, it can be stated that many approaches focus on Java and the Java Sand-box as [10] and [3], or special purpose programming languages [7]–[9]. However, the execution performance of code written in such languages is not sufficient for many kinds of applications (e.g media transcoding, intrusion detection, etc.). One major advantage of our approach in this respect is its programming language independency.

III. The Access Control Mechanism

A primary directive of our approach states that only authenticated code can be executed on an active node. A service can either be signed only by the author of the service or additionally, by the network administrator. The presence of the administrator's signature indicates that the service security policy has been inspected by the administrator and consequently he also specified a trust label for the service. The trust label is a means to specify the trustworthiness of a service which simultaneously determines the minimal access control mechanism configuration for this service (at least the set of access control kernel modules specified by the corresponding trust label must be loaded before the execution of the service can be started). The corresponding framework which allows to articulate the resource and authorization requirements for individual active nodes and services is described in detail in [5].

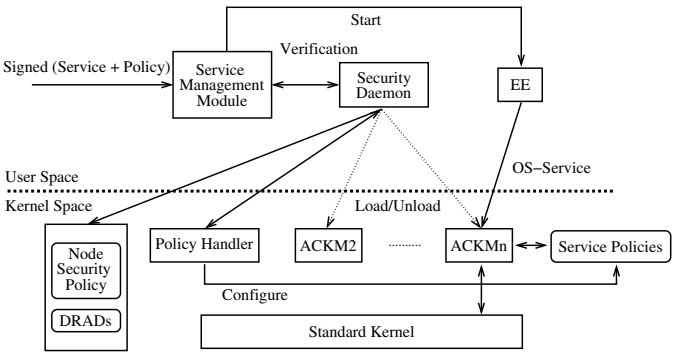

Fig. 1. The Architecture of the Access Control Mechanism

The framework (see figure 1) consists of a *security daemon*, a *policy handler*, a set of *access control kernel modules (ACKM)*, a *service revocation list* (SRL), a node security policy, a list of *default authorization and resource descriptions* (DRAD) and a database for the storage of the service security policies of the actual running services. One particularity of our approach is that security daemon and policy handler are the only two permanent entities. The access control kernel modules are realized as active networking service and thus activation occurs on demand. The service management module, the security daemon and a varying set of execution environments reside in user space. The service management module receives the signed active data (service + security policy), then it passes the complete data to the security daemon and waits for a verification result. The functionality of the security daemon is described in detail in section III-A. Assuming that the security daemon returns a positive answer to the service management module, then the service can be started inside an execution environment. Further on, the rest of the access control mechanism policy handler, service security database, node security policy, SRL, DRADs and the varying set of access control kernel modules ACKMs is placed in kernel space. Each ACKM is responsible for the supervision of one specific resource respectively one operating system service. The supervision is based on the technique of intercepting system calls.

A. The Security Daemon

The security daemon is responsible for the following tasks:

- verification of digital signature;
- comparison between service and node security policy;
- activation/deactivation of ACKMs;
- handover of service security policy to the policy handler module.

In the case of a service download from the service repository, the transferred data consists of one or two digital signatures (service author / administrator), a service security policy and the service itself. A service security policy consists of a *resource and authorization description (RAD)* specified by the service author and a *trust label* set by the administrator. In the case that the amount of trust labels to be set by the network administrator is too high, unreviewed services (those which are only signed by the service author) are handled as if they belong to the weakest trust category.

At the beginning, the security daemon splits the data into the above listed individual parts and next the digital signature is verified. If the verification was successful, it is still not clear whether the service can be executed on that active node or not. Therefore, the security daemon checks the service revocation list, the node security policy, and the list of default resource and authorization descriptions (for a more detailed description please refer to [5]). Assuming that all checks were successful, then the next task of the security daemon is to verify that all required ACKMs are loaded. If an ACKM is missing, the security daemon activates it before the service can start its execution. If the required ACKM is not available on the active node itself, then the ACKM is dynamically downloaded from the service module repository. Each ACKM is also digitally signed, and therefore it must pass the same initial test before being activated.

Afterwards, the service security policy is inserted into the local security database which is also realized in kernel space in order to minimize the policy lookup overhead when supervising specific access requests during the service execution phase. Finally the service management module starts the execution of the service in a newly created execution environment.

B. The Policy Handler

This section describes the policy handler, its tasks and the interconnection to the other entities of our approach. The policy handler is responsible for:

- supervision of the activation / deactivation of ACKMs
- supervision of the exit phase of services
- management of the service security database;

As already mentioned, kernel code cannot be supervised. In order to face this problem only the security daemon is capable to load or unload kernel modules. Therefore, the policy handler intercepts any user space request concerning the activation or deactivation of a kernel module, and only if the security daemon (verification of the process id) is the calling entity the request can be fulfilled.

Furthermore, the policy handler observes the exit phase of each service running on an active node. If a service exits, it must be guaranteed that also all child processes terminate. Hence, the policy handler keeps track of all child processes of each service. In case of a non-terminating child process of an already ended active service, the policy handler itself kills the child process.

The policy handler is responsible for the management of the service security database. A user space process is identified through its process identification number (PID). The PID and service security policy are linked and afterwards inserted into the database. If a service exits, it is the policy handler who removes the corresponding entry from the database. In doing so, the policy handler also verifies if an ACKM can be deactivated. Therefore, node security policy and service database must be consulted. An ACKM can only be removed from the kernel when no actual running active service requires the supervision of the corresponding resource or operating system service.

Finally, after the termination of a service, the policy handler adjusts the resource consumption of all active services in the appropriate data structure in the database (see also below).

C. An Access Control Kernel Module

By inserting a loadable kernel module, it is possible to extend the functionality of the kernel. In our case, it is possible to introduce more detailed decision criteria in the kernel to determine whether the desired action is allowed or not. The general method of system call interception is depicted in figure 2 and shows the interception of the *socket* system call. The user process uses the *socket()* command to create a socket for network communication. A process must execute a system call to gain access to the operating system services. Normally this is done by wrapper functions which are part of standard libraries. The wrapper function puts the variables to be submitted into the correct order, and then executes the proper system call. At the entry point into the kernel, the kernel uses a table – the so called system call table – for the forwarding of the incoming system calls to the corresponding functions. By changing the destination of a pointer inside the system call table, we can redirect a defined system call to another function. An ACKM intercepts exactly one system call. Consequently, for each resource and operating system service to be monitored, the corresponding ACKM must be loaded. An ACKM queries the database for the service security policy related to the requesting process, in order to check if the process is within its resource limits or if the service is authorized to use a specific operating system service. If this test is passed, the ACKM updates the appropriate data structure in the service security database to keep track of the resource consumption of each service. Additionally, the aggregate resource consumption by all running services (also stored in a data structure inside the database) is updated by each ACKM. Resources which are consumed by child processes are added to the resource consumption of their parents. Thus, parent and child process are evaluated together.

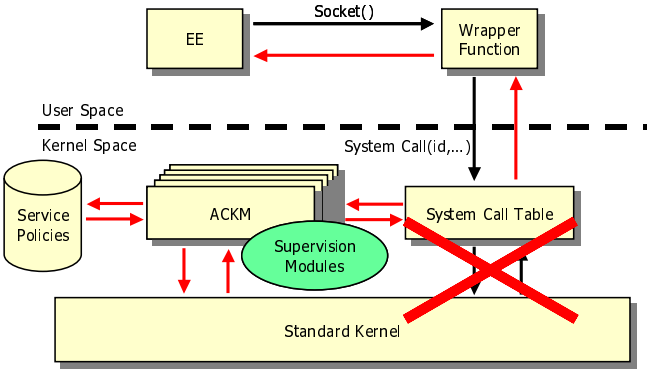

Fig. 2. Interception of a System Call

Next, the ACKM calls the standard kernel function belonging to the system call.

D. The Service Security Policy

A service security policy consists of two domains of responsibility. On the one hand, there is the author who specifies the resource and authorization requirements for his service in order that the service can be executed on an active node. On the other hand, the network provider specifies the supervision requirements for the execution of the service through the setting of the trust label and he has the possibility to specify an additional list of adaptive criterions for a service. Summarizing, a service security policy consists of a RAD, a trust label and a list of adaptive criterions (LAC). An adaptive criterion is a criterion which depends on changing condition(s) (see [5]).

E. Node Security Policy

At first, the node security policy defines which resources and OS-services are generally available to the services on that active node. Additionally, it also defines the complete quantity of each resource which can be maximally consumed in aggregate by all actual running services. Furthermore, the node policy provides a means to bound the set of services which can be executed on an active node (origin, trust label, etc.). The structure of a node security policy does not necessarily conform to the structure of a service security policy but the latter must be a subset of the node security policy.

Generally there are three possibilities to restrict the utilization of a resource inside a service / node security policy. First, a yes / no value, which indicates if a / any service is allowed to access the specified resource at all. Second, a quantitative value, which determines an upper limit of the specified resource that can be consumed by a / all service(s). And finally, a predefined value which is to be used for the specified parameter (e.g. destination address for a TCP-connection). Besides this, the framework also includes a service revocation list (SRL) as well as a list of default resource and authorization descriptions (DRADs). A service which is listed in the SRL must not be executed on any active node, and each DRAD entry specifies the authorization and resource limits for services of a specific trust label.

F. The Trust Label

One question of interest is, under which supervision a service must execute? Therefore, we introduced the so-called trust label. Through the setting of a trust label the network provider is able to articulate his trust in a service ranging from completely untrustworthy to completely trustworthy. One extreme possibility is a service which is labelled completely trustworthy due to a long experience with the author and extensive testing of the service. A service with such a label would not require any runtime supervision.

Services with a trust label apart from completely trusted require runtime supervision. Therefore, each active node additionally contains a list of DRADs.

G. Active Node Domains

Through the integration of node security policies we gain a great degree of flexibility, as the network administrator is able to specify what services may be executed on which active nodes. An active node inside the Internet requires certainly a different degree of supervision than an active node in a test-bed or inside an intra-net.

This flexibility enables us to generate several quality of service domains through different sets of node policies. The active nodes inside a domain have the identical node security policy. For example, inside a domain A only services of security category 1 could be executed as the nodes are best effort nodes. This means that the services on these active nodes receive no supervision. On the other hand, inside a domain B services of all security categories could be executed under a strict authorization checking. Certainly, this requires fast hardware and produces higher costs but therefore, a better availability is gained.

Furthermore, an upper limit for each resource which can be maximally consumed by all running services on that node can be specified in the node security policy, in order to secure a safe execution of the base services of the active node, e.g. routing tasks.

IV. Measurements

In this section, measurements achieved with a first prototype for the active networking infrastructure AMnet [4] are discussed. AMnet provides a platform for rapid and flexible service creation and uses active nodes (so-called AMnodes) within the network for the execution of services.

The access control consists of two phases: an initial test and the runtime supervision. The initial test consists of the verification of the digital signature and the examination whether the service can be executed on that active node. Afterwards, the runtime supervision adopts the responsibility for the service. The presented measurements focus on the runtime supervision.

The service used for the measurements creates a socket. Afterwards it steps into a loop of 1.000.000 cycles. In each loop cycle the service executes a *setsockopt()* command to

TABLE I
OVERHEAD PRODUCED THROUGH THE INTERCEPTION OF SYSTEM CALLS

Amount of service security policies	0	10	100	1000	10.000
Overhead in [μs]	0.068	0.139	0.190	0.244	0.345
Relative Overhead [%]	14.25	29.04	39.82	50.97	72.21

change a socket option. This command is intercepted by the corresponding ACKM_Network module. Intercepting a socket system call, the security kernel module makes a lookup in the local security database to see if the requesting service has got the authorization or not. We varied the number of stored security policies inside the corresponding database, and the position of the security policy inside the database of the requesting process was uniformly distributed. The security policies are stored in an ordered array (an indexed search over the process id), and to find the proper policy a binary search algorithm is executed.

Table I shows the results of our measurements. The first row of the table represents the number of service security policies stored inside the database. Thus, we see that for a number of zero policies, no lookup inside the database is required, we receive the overhead caused through the interception of the system call, which is about 0.0.68 μs. Additionally, we extended the number of service security policies stored in the database up to 10.000. The additional overhead which can be observed is caused through the lookup for the proper service requirements list. The last line of the tables represents the relative overhead. It can be seen that an additional overhead of 30 % is quickly reached. As the supervision is expensive, it is important only to supervise the actions which need to be supervised ("demand-driven security").

The measurements were made with a Pentium III/800 machine running Linux 2.4.18 kernel.

V. CONCLUSION

The demand-driven access control mechanism presented in this paper exploits the possibilities provided by an active infrastructure to upgrade and configure itself.

The realization of the access control mechanism in kernel space has many advantages and one drawback. Namely, if the trust label of a service requires the activation of an ACKM, then the access requests of all processes for that resource will be affected. But to our understanding, the kernel is the only place where all resource and OS-services requests can be supervised. Supervising system calls in user space, on the other hand, would require the supervision of every wrapper function, and still a wrapper function could be linked statically to a system service, enabling malicious or incorrectly linked executable code to circumvent the access control enforcement.

Our approach furthermore provides the possibility to execute unreviewed services under strict runtime supervision. Unreviewed services are treated as services of the lowest trust label. Additionally, it is possible to create different quality of service domains such that only services of similar trust labels could be executed on an active node.

Regarding the presented results, we see that the overhead produced through the runtime supervision is about 0.345 μs for one system call and 10.000 entries in the database. Concludingly, the supervision of services can get expensive, if many services execute a huge amount of supervised system calls within a short period of time. Thus, the automatic deactivation of non required ACKMs and the creation of QoS domains as outlined above helps to keep the performance degradation within tolerable limits.

REFERENCES

[1] Christian Bachmeir and Peter Tabery. PIRST-ONs: a service architecture for embedding and leveraging active and programmable networks technology. In *IEEE 10th International Conference on Software, Telecommunications and Computer Networks, SoftCOM*, October 2002.

[2] Roy H. Campbell, Zhaoyu Liu, M. Dennis Mickunas, Prasad Naldurg, and Seung Yi. Seraphim: Dynamic interoperable security architecture for active networks. Technical Report 2133, Department of Computer Science, University of Illinois at Urbana-Champaign, Urbana, Illinois, 1999.

[3] David Evans and Andrew Twyman. Flexible policy-directed code safety. In *IEEE Symposium on Security and Privacy*, pages 32–45, 1999.

[4] T. Fuhrmann, T. Harbaum, M. Schöller, and M. Zitterbart. Amnet 2.0: An improved architecture for programmable networks. In *Proceedings of the International Workshop on Active Networks IWAN*, 2002.

[5] A. Hess and G. Schaefer. A flexible and dynamic access control policy framework for an active networking environment. In *Proc. of Kommunikation in Verteilten Systemen (KiVS 2003)*, pages 321–333, Leipzig, Germany, February 2003.

[6] A. Hess and G. Schäfer. ISP-Operated Protection of Home Networks with FIDRAN. In *IEEE Consumer Communications and Networking Conference 2004*, Las Vegas, Nevada, USA, January 2004.

[7] Michael W. Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A packet language for active networks. In *International Conference on Functional Programming*, pages 86–93, 1998.

[8] George C. Necula. Proof-carrying code. In *Proceedings of the 24th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Langauges (POPL '97)*, pages 106–119, Paris, January 1997.

[9] George C. Necula and Peter Lee. Safe kernel extensions without run-time checking. In *2nd Symposium on Operating Systems Design and Implementation (OSDI '96), October 28–31, 1996. Seattle, WA*, pages 229–243, Berkeley, CA, USA, 1996. USENIX.

[10] R. Pandey and B. Hashii. Providing fine-grained access control for mobile programs through binary editing. Technical Report TR-98-08, 1998.