A dynamic and flexible Access Control and Resource Monitoring Mechanism for Active Nodes

A. Hess - Technische Universität Berlin, Germany
M. Schöller - University of Karlsruhe, Germany
G. Schäefer - Technische Universität Berlin, Germany
M. Zitterbart - University of Karlsruhe, Germany
A. Wolisz - Technische Universität Berlin, Germany

Abstract

Active and programmable networks are new paradigms in computer networking. Network nodes have the ability to load and execute special purpose programs called service modules (audio transcoder, traffic screening agent, etc.). The service modules are either stored on the node itself or dynamically downloaded from a service module repository and installed on the network node. These modules are designed and implemented by the provider or by a third party and as it is theoretically impossible to construct a generic algorithm to determine what a program does, the introduction of a quantitative access control mechanism seems to be a promising approach. Therefore, the standard security mechanisms of an operating system must be enhanced. This paper presents a generic mechanism to administratively limit the resources granted to service modules. The discussed technique is independent of the programming languages in which the services are implemented. The presented results, achieved with a first prototype developed for the active networking platform AMnet [4] (Active Multicast Network) are very promising.

1 Introduction

The programmable networking technology provides among other things a framework for flexible and rapid service creation on top of existing networks. It uses enhanced nodes within the network for the provision of individual application-specific services. This paper deals with the case that the programmable nodes can execute services which are loadable on demand from a service module repository and enhance the functionality of intermediate systems in a flexible manner. A service is composed of one or more service modules which are linked to a process chain. This chain is connected to an Execution Environment (EE) which hands packets to the first module in the chain and picks up the packet from the last. Those services cover areas such as media transcoding, semi reliable multicast, and congestion control.

The execution of services on nodes positioned in the Internet raises a huge security challenge. As it is theoretically impossible to construct a generic algorithm to determine what a program does, other possibilities to secure the execution of arbitrary code must be considered.

Our approach is the supervision of each process through a quantitative access control mechanism. This can be realized by supplying each application with an individual security policy which can be added at runtime and defines which resources are available for usage in which manner and size. The quantitative access control mechanism consists of two parts: an access control entity and a resource monitoring entity. The access control entity is responsible for the adherence of the security policies of the running services and the resource monitoring entity observes for every running service the amount of each resource consumed by it. Further on, it must be guaranteed that the quantitative access control mechanism is always able to react regardlessly of the circumstances.

All services for a programmable network differ in their resource requirements. They can be classified in two main categories: constant requirements and variable requirements. The first class contains modules like a stereo to mono audio transcoder where all packets are processed in the same manner. Such a module needs the same resources to process each packet.

Modules like an MPEG color to b/w transcoder also belong to the first class. Although the processing of a packet depends on the kind of frame the resource utilization has a constant upper limit. Semi reliable multicast and congestion control modules are directly dependent on external resources like link quality or system load on other network nodes and therefore belong to the second category. Such services are usually in stand-by mode occupying only a small amount of resources of the system. The reaction to special occasions contribute significantly to the resource requirements of these services.

This paper focuses on availability and access control as two major security goals for an active network node. This means that services are available and function correctly and further on, only authorized entities are able to access certain resources. Thus, the administrator must define a security policy for each service he wants to offer. The policy specifies which resources may be used by a service and also to what quantitative extent they may be used.

An overview of the architecture and a introduction to the concepts developed for access control and resource monitoring are given in the next section. Afterwards the paper is organized as follows. Related work is presented in section 3. Section 4 focuses on the implementation and performance measurements and section 5 concludes and gives an outlook on future work.

2 A dynamic and flexible Access Control and Resource Monitoring Mechanism

In this section, we present the concept of our dynamic and flexible access control and resource monitoring mechanism for an active node. One goal of the mechanism is that it should be as generic as possible, i.e. it should be usable independently of the type of services that are deployed on the active node. In other words, the execution environment provided by the active node for the execution of the service should have no influence on the control mechanism. Further on, the access control and resource monitoring mechanism should be easily extensible for future security and monitoring requirements and it should be fine-grained configurable. Additionally a realization of the concept without any changes of existing operating system code and with minimal performance degradation should be accomplished.

Due to the mentioned reasons we decided to split the access control and resource mechanism into a kernel space and a user space part. The user space part is a daemon which configures the kernel part of the mechanism and the kernel part is responsible for the access and resource control.

2.1 Architecture

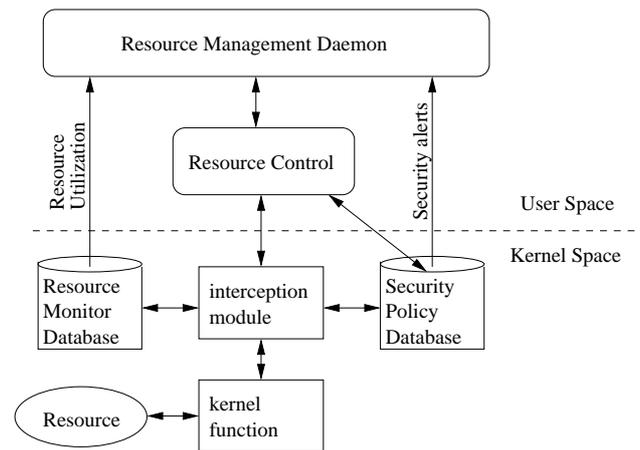The considerations mentioned before lead to the architecture shown in figure 1.

Figure 1: Components of the access control and resource monitoring mechanism

For every system call for which an access control policy exists or which must be monitored an interception module is inserted into the kernel. For access control the module contacts the security policy database to check the access rights of the process to the system call. If this test is passed the interception module updates the appropriate data structures in the resource monitor database. Both databases transfer their data to the resource management daemon. This is the central node management process. The resource control process is responsible for module handling and security policy setup. The interception module then calls the standard kernel function belonging to the system call. No new functionality to handle resources is introduced by the interception module.

In the next section the general technique of how to intercept a system call is explained.

2.2 System call interception

Most common operating systems make a distinct differentiation between application and operating sys-

tem. Each time a service requires an operating system service (e.g., to open a socket for communication) the service must send the proper system call to the kernel. The kernel checks the request of the process and then it decides whether to fulfill it or not.

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.
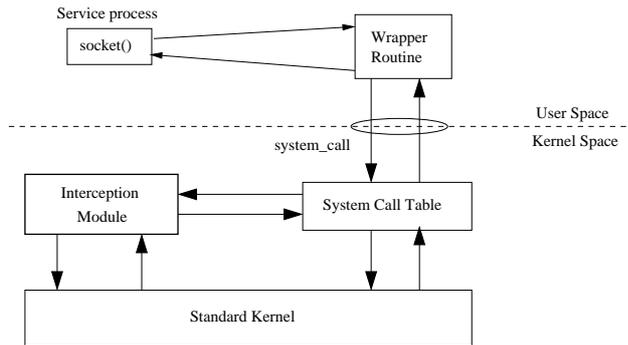

Figure 2: Interception of a system call

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. As mentioned, 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.

The entry point to the kernel is a good point to intercept certain operations. It would not be sufficient to intercept the command in user space by modifying the corresponding wrapper function, as in this case the programmer could link his own wrapper function statically into its program or do the system call directly.

2.3 Security - The Access Control Entity

The access control entity is responsible for the supervision of the adherence of the individual security policies for the running services. The security policy can specify one of the following restrictions for a resource:

1. Grant access: indicates, whether a service is allowed to access the specified resource at all;

2. Access limit: specifies the maximum amount of the specified resource that can be consumed by the service;

3. Predefined value: value to be used for a given parameter (e.g. destination address for a TCP-connection).

Items 1 and 3 are solely supervised by the access control. The second item is controlled by the resource monitoring mechanism. The security entity provides functions through which the upper consumption limit of a specified resource for a service can be prompted. If a service tries to consume a bigger portion of a resource than it is allowed to, the corresponding request is denied. The possibility to setup hard resource limits is necessary to prevent resource exhaustion of the system. If the access to a resource is once granted it is irreversible without interaction between the kernel module and the service process. Such an interface would prevent a generic mechanism.

The mentioned access control mechanism and further security aspects for active networking environments are described in detail in [5].

2.4 Resource monitoring

The security part of the mechanism checks the access rights to the requested resource and reports all illegal resource requests of services to the resource management daemon. The reaction to illegal actions is part of the configuration of the resource management daemon and is beyond the scope of this paper.

The resource monitor keeps track of all requests to a system resource and logs every allocation and deallocation. The resource monitoring module provides the current, the average, and the maximum resource allocation of each service to the resource management daemon. This information characterizes all services from the constant requirement class. After some time

of monitoring the values of average and maximum resource allocation get stable and are, therefore, characteristic for that service.

For all services from the variable resource requirements class the monitored data are just a snapshot information and can not be used for future predictions. As mentioned before, these service modules can contribute significantly to the resource utilization of the system and are easy targets of denial of service attacks due to their service characteristics. To provide availability, the system administrator might want to limit the maximal resource utilization of these services.

3 Related Work

The DARPA active network community [1] developed an architecture for active network nodes. The primary functional components are the Node Operating System (NodeOS) and the Execution Environments (EEs). The NodeOS is responsible for resource management. Bowman [7] is an extensible platform for active networks based on the DARPA architecture's NodeOS. It extends existing operating systems like SunOS or Linux to an active node but does not implement an own resource management. It completely relies on the resource control of the underlying OS. In contrast to our proposed approach the operating systems do not support fine-grain resource monitoring and limitations for service processes and impending overloads can not be detected.

The Resource Controlled Active Networking Environment (RCANE) [6] and the Darwin project [3] introduce a fine-grain resource monitoring and accounting mechanism. RCANE extends the Nemesis Operating System with special purpose schedulers for CPU, network I/O, and memory. To achieve this fundamental mechanisms of an operating system must be changed. The Darwin project focuses on bandwidth and user data access restrictions and accounting. Other resources are not regarded in their approach.

The Ariel Project [8] and the Naccio Project [2] 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.

4 Implementation

The concepts described in section 2 are implemented and tested for the AMnet Execution Environment [4]. The current version of AMnet runs as a software router on standard PC hardware. The operating system is Linux kernel version 2.4. The AMnet node consists of a signaling program, a resource controller, an EE and a resource management daemon. The service modules and the Security/Resource Monitor modules (SR modules) can be statically installed on the node or downloaded on demand.

The download of an SR module is invoked by the Resource Control Process on service startup. The module is identified by name and a version number. The version number can be omitted if any version of the SR module is sufficient or can be set to indicate that only versions later than the one specified are appropriate. The signaling process returns an error message to the Resource Control Process if the requested module can not be found in the service module repository and service startup is terminated immediately. A successfully downloaded module is put in the local SR cache and the process is notified that the module is now available. The Resource Control Process installs and initializes the SR module.

Service Module Repository   Active Node

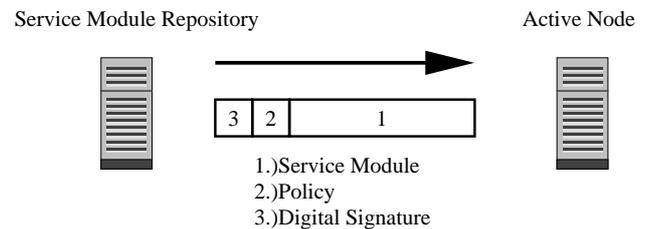| 3 | 2 | 1 |

1.)Service Module
2.)Policy
3.)Digital Signature

Figure 3: Transmission of active code

As depicted in figure 3, the arriving active code is supplied with a digital signature and a security policy. The daemon first splits the arrived data into three parts: active code, policy and digital signature. Then, the digital signature is verified. If the verification was successful the security policy of the service is compared with the local security policy of the active node, whether the active node is authorized to execute the service or not. If yes, the execution of the active code can be started.

All downloaded SR modules are also put in the local SR cache for faster access the next time the module is needed. The cache is managed by simple aging strat-

egy. For each module a timestamp is saved. When the SR gets unloaded the timestamp in the SR cache is updated. If a new SR module is requested and there is not enough free memory within the cache the SR module with the oldest timestamp is discarded from the cache. The procedure is repeated until the downloaded module can be put into the cache. This implies that even modules which are currently active can be discarded from cache. This is necessary to be able to run more SR modules than can be cached.

4.1 SR Modules

All SR modules are realized as kernel modules. The Linux kernel provides a standard interface to insert and delete kernel functionality at runtime. The configuration of such a kernel module can also be changed at runtime.

For flexibility reasons the SR module is split into a security and a monitoring part. Each part can be used independent of the other one. This allows security checks for system calls which are not monitored or vice versa.

The parameters of the function call are analyzed by the monitoring part and the new resource utilization is calculated. Thereafter, the system call is passed to the security part to do the security checks. The return value of the security part defines the proceeding of the monitoring part. If the access is granted, the standard Linux system call routine is called and the monitoring database is updated. Otherwise, an error is returned to the calling process.

An SR module does not notify the resource management daemon on every resource allocation. All collected data about resource allocations are stored in module variables. The content of these variables is accessible through the proc-file-system. Every SR module which provides a monitoring function must implement this interface for data access. The user space resource management daemon periodically polls this special purpose file to get the information about the resource utilization.

4.2 SR_Net: An Example of a SR Module for the control of network access

In this section an example SR module for the control of the network access is presented. Thus, we focus on the *socketcall()* system call. *Socketcall(int call, un-*

*signed long *args)* is the entry point for all existing socket operations, except *write()* and *read()* which can also be used to send and to receive messages using an existing socket. The operation requested is defined through the parameter *call*. Three interesting possible operations are:

1. *SYS_SOCKET*: The *socket()* function is used to create a socket of any supported protocol family.

2. *SYS_CONNECT*: The function is used to establish a connection between two communication endpoints.

3. *SYS_SENDTO*: The *sendto()* function allows to send a datagram and specify the destination address of the recipient at the same time.

By redirecting the *socketcall()* system call to our loadable kernel module we have the possibility to access all socket functions including the parameters passed. Thus, the possibility is given to decide corresponding to the security policy of the requesting process:

- Has the process the right to create a socket?

- If yes, what kind of socket (Stream, Raw, Datagram, ...) is the process allowed to create?

- Which destination addresses/domains are allowed for the transmission of data?

- The sizes of the used buffer

4.3 Measurements

In this section the measurements achieved with a first prototype for the active networking infrastructure AMnet running under Linux 2.4.18 / Pentium III/800 are discussed. The SR_Net module (see section 4.2) was utilized to intercept the *socketcall* system call. 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 uniform distributed. The security policies are stored in an ordered array and to find the proper policy a binary search algorithm was realized.

In table 1 the overhead caused through the *socket()* system call interception is depicted. The first column represents the amount of service security policies stored inside the database. The second and third column represent the absolute and relative overhead

Table 1: Overhead due to the interception of system calls

No. of security policies	absolute [s]	relative [%]
0	0.068	14.25
10	0.139	29.04
100	0.190	39.82
1000	0.244	50.97
10.000	0.345	72.21

Table 2: Time needed for the execution of 200 pings to the loopback interface

No. of security policies	No SMR	SMR loaded
0	3m18.975s	3m18.975s
10	-	3m18.975s
100	-	3m18.976s
100.000	-	3m19.064s

caused through the interception of a system call and the corresponding look up in the database whether or not the calling process has got the authorization.

The bigger the number of security policies the more time is needed in average to find the proper policy. If we look at the value for zero service security policies inside the database, we get an additional CPU-time needed to intercept and redirect one system call from about 0.068 μs.

The presented result focused on the CPU time. In a next experiment we measured the complete time needed to send and receive 200 ping packets to and from the loopback interface, i.e. we also measured the time during which the process is sleeping.

The figures in table 2 are the calculated mean values. The corresponding standard deviation is smaller than 5 ms per given value. Thus, we can conclude that up to 100 security polices stored, there is no remarkable performance degradation noticeable, at least for the described ping application. The security policy used for the measurement consists of a process identifier and a yes/no decision for network access.

5   Conclusion

We have presented an at runtime fine-grained configurable access control and resource monitoring mechanism for programmable networks which is based on a standard operating system. Further on, the presented mechanism is easily extensible for future security and resource requirements due to the modular design. The interception of system calls gives us the possibility to restrict the access to certain resources and to monitor the resource utilization on a per process basis, independent of the execution environment provided by the active node for the execution of the process.

Concluding we can state that the presented results show that the approach is very promising. A performance degradation is measurable but has no remarkable influence on the execution time of the service.

References

[1] K. Calvert (Editor). Architectural framework for active networks. In *DARPA AN Working Group Draft*, 1998.

[2] D. Evans. Flexible policy-directed code safety. *IEEE Security and Privacy*, 1999.

[3] J. Gao and P. Steenkiste. An access control architecture for programmable routers. In *Proceedings of the IEEE OPENARCH*, 2001.

[4] T. Harbaum, A. Speer, R. Wittmann, and M. Zitterbart. Providing Heterogeneous Multicast Services with AMnet. *Journal of Communications and Networks*, 3(1):46 – 55, March 2001.

[5] A. Hess. A dynamic and flexibel access control mechanism for active networks. Technical report, Technische Universität Berlin, 2002.

[6] P. Menage. RCANE: A Resource Controlled Framework for Active Network Services. In *Proceedings of the First International Working Conference on Active Networks (IWAN '99)*, volume 1653, pages 25–36. Springer-Verlag, 1999.

[7] S. Merugu, S. Bhattacharjee, E. Zegura, and K. Calvert. Bowman: A node OS for active networks. In *INFOCOM (3)*, pages 1127–1136, 2000.

[8] R. Pandey and B. Hashii. Providing fine-grained access control for mobile programs through binary editing. Technical report, University of California, 1998.