A Flexible and Dynamic Access Control Policy Framework for an Active Networking Environment

A. Hess , G. Schäfer

Telecommunication Networks Group
Technische Universität Berlin, Germany

Abstract. To provide security for active networking nodes with respect to availability and controlled access the introduction of an access control mechanism and consequently a policy framework are mandatory. We follow the approach of a scenario-tailored runtime supervision of the service. 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 could be executed on an active node. Furthermore, also service-specific adaptive criterions could be included into the initial check. This paper discusses the corresponding flexible and dynamic access control policy framework and we also present results achieved with a first prototype realized for the active networking environment AMnet.
Keywords: active networks, security, policy, access control

1 Introduction

In this paper we present a security policy framework for an active networking environment which is flexible and dynamically configurable using the possibilities given through the underlying active networking infrastructure. Our approach bases on a scenario tailored runtime supervision [7] of the services whereas the supervision entities decide corresponding to access control policies whether to fulfill specific requests. According to [3] a policy makes clear, what is being protected and why. It states the responsibility for that protection and it provides a ground on which to interpret and resolve any later conflict.

In this report we do not precisely define how to express a policy but we recommend to follow the Principle of Least Privilege [14], which holds that each principal be accorded the minimum access needed to accomplish its task. Our approach provides the possibility to supply each active node and service with an individual, special-purpose access control policy. Active services may pose a threat, as the general prediction of code semantics is impossible [9]. Thus, the presented approach is a combination of origin verification and policy-based access control. Origin verification means, that only authenticated code could be executed on an active node, on which in turn an access control mechanism supervises the execution of the service. The access control mechanism decides corresponding to a policy whether a request for a resource or an operating system service respectively is authorized or not.

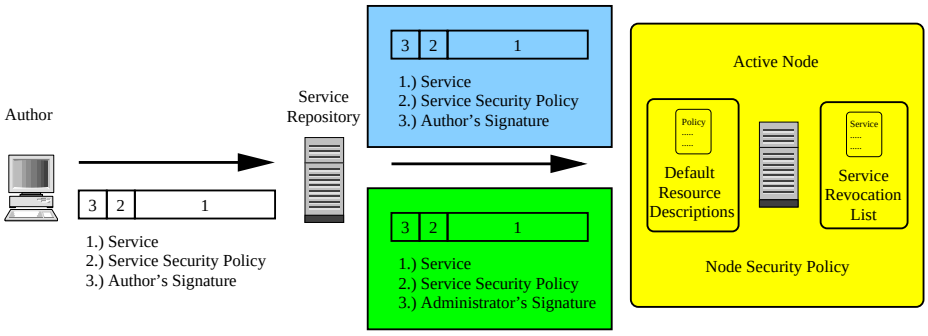

Fig. 1. The Transmission of a Service

For the development of the policy framework we posed the following requirements:

- An active node-specific protection: this requires the integration of a node security policy which can be configured dynamically at runtime.
- Service-specific supervision: this requires the integration of a service security policy.
- Polices must be easily extensible for future requirements due to:
  - new authorizations requirements
  - new known exploits, security holes
- Dynamic policy update in order to react on recognized security violations
- Possibility to execute untrusted / unreviewed services with a minimum set of privileges
- An efficient supervision: scenario-tailored runtime supervision
- Static and adaptive criterions in order to decide whether a service could execute on an active node
- Efficient policy management: the modification of a policy (e.g. adding or changing an entry of a service policy, etc.) should automatically propagate to the active nodes of a network in order to support policy updates for large networks.

Recapitulating, we state that our approach includes two different types of policies: a service security policy and a node security policy. In the following sections we will explain our approach in detail.

2 An Active Networking Environment

The active networking infrastructure consists of active nodes, a service repository (SR), a network provider and several users. An active node is able to execute services which are stored on the SR. If a new service is to be executed on an active node, the user / administrator is able to send the corresponding trigger signal to the active node, such that the active node downloads and installs the service. The installed service runs inside a so-called *execution environment (EE)* in user-space.

In figure 1 the transfer of a service between the host of the author and service repository and additionally, the transfer between service repository and active node is depicted. The transferred data consist as depicted of the service and the service security policy. Further on, regarding the transfer between service repository and active node the data are either signed by the author of the service or by the network administrator. A service security policy consists of a *resource and authorization description (RAD)*, a *trust label* and a *list of adaptive criterions (LAC)*. It is the responsibility of the network administrator to set the trust label and further on, to specify any required adaptive criterion (see section 3.2) for a service. If it is the case that the number of trust labels to be set by the network administrator is too big, the unreviewed services are handled as if they belong to the weakest trust category. The structure and functionality of the service security policy is discussed in detail in section 3.2.

Further on, the depicted active node in figure 1 is supplied with a node security policy, a *list of default resource and authorization descriptions (DRAD)* and a *service revocation list (SRL)*. The node security policy defines the security strategy of the active node, a DRAD specifies the limiting conditions for services of a specific trust label and the SRL is a listing of services which must not be executed on an active node.

3 The Policy Framework

In general a policy framework should protect the active nodes. It should detect misuse, attacks, bugs, etc. and afterwards itshould be able to react to the detected events. Our approach provides on the one hand a mechanism to configure the resources and authorizations which are available for services on an active node and on the other hand it provides also an mechanism to configure the resources and authorizations limits for a single service. Thus, the network provider could secure the execution of basic functions as routing tasks through the setting of reasonable authorization and resource limits on an active node (see also section 3.3). Further on, a mechanism is integrated to secure the execution of services which were not reviewed by the network administrator or which are labelled untrustworthy.

The policy framework consists of the following units:

- A service security policy, which consists of a resource and authorization description defined by the author of the service, a trust label and a list of adaptive criterions which are set by the network administrator.
- A node security policy which is defined by the network administrator and which specifies the security strategy of the active node.
- A list of default resource and authorization descriptions which are specified by the network administrator and each DRAD specifies the authorization and resource limits for services of specific trust label.
- A service revocation list, which is a listing of services which must not be executed on any active node and which is configured automatically by the active nodes and by the network administrator.

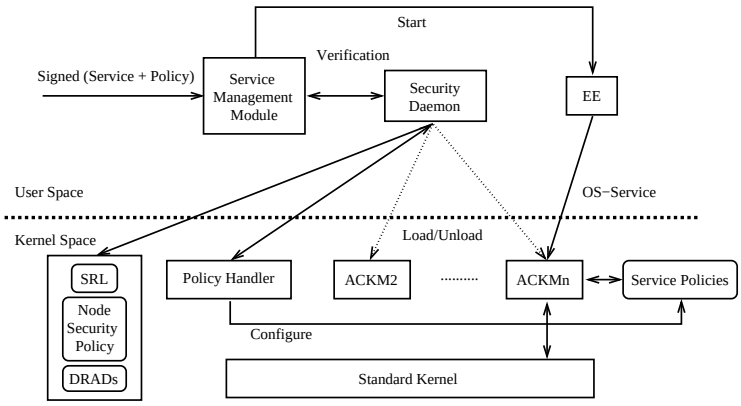

Fig. 2. Overview of the Architecture

3.1 The Policy Framework Architecture

In figure 2 the policy entities integrated into an active node are depicted. The policy framework consists of a *security daemon*, a *policy handler*, a set of *access control kernel modules (ACKM)*, an SRL, a node security policy, a list of DRADs and a database for the storage of the service security policies of the actual running services.

First we explain commonly used terms in the field of policies:

- Policy enforcement point (PEP) is the component that actually encounters the requests by services and is responsible for enforcement and execution of policy actions.
- policy decision point (PDP) is the component that is responsible for determining what actions are applicable to which services.

In our approach the functionality of PEP and PDP are realized together in so-called access control kernel modules. Each ACKM acts as PEP and PDP simultaneously for exactly one part of the service security policy, e.g. there is one ACKM for main memory requests, one for CPU cycles requests, etc. Our approach follows the goal of a scenario tailored supervision and through the setting of the trust label (see also section 3.2) the network administrator articulates his trust in a service or in other words he authorizes certain requests which could be posed by the service. Concluding, we do not want to supervise actions which the administrator labelled trustworthy and thus we follow the approach of a modular supervision of services. In figure 2 a security daemon is depicted. Security daemon and policy handler are the only two entities that must be running at any moment as they are responsible for the actual configuration of the policy framework structure. Before a service could be executed on an active node, security daemon and handler execute the initial check ($\Rightarrow$ is the service allowed to execute on that active node?) and afterwards they load / unload the required ACKMs. The initial check is depicted in detail in figure 3 and it consists of signature, SRL, node security policy, DRAD verification and the evaluation of the adaptive criterions. Node security policy (DRAD)

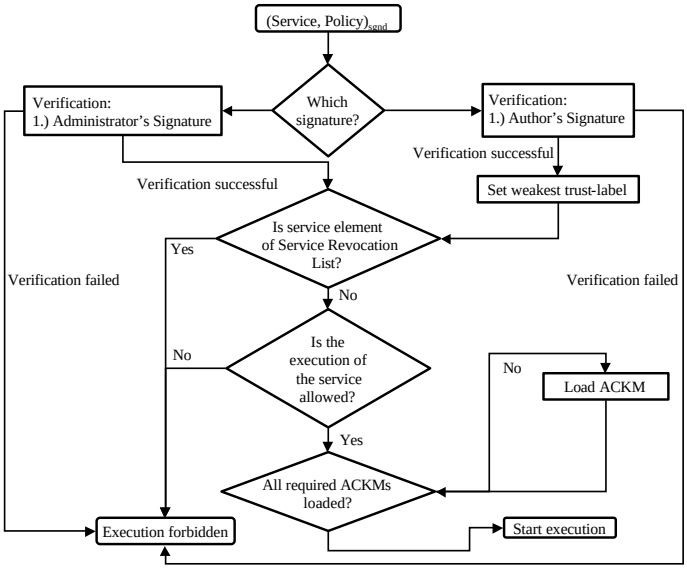

Fig. 3. The Initial Check

verification means a comparison between node security policy (DRAD) and service security policy in order to decide whether to allow the execution of the service. Therefore, the service security policy must be a subset of the node security policy (DRAD). How the adaptive criterions are evaluated is described in detail in section 3.2. If all verifications were successful, the security daemon and policy handler must take care about the required ACKMs. The security daemon is the only entity with the authorization to load / unload kernel modules and to modify the kernel variables as SRL, node security policy and DRAD. The procedure is described in more detail in section 3.6. Once all required ACKMs are active the service could be started inside an execution environment as depicted in figure 2.

3.2 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 could execute on an active node, but it is not said that the service obtains these. 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).

The Resource and Authorization Description Each service is supplied with a resource and authorization description which is specified by the author of the service as he is the person who knows best what kind and amount of resources and authorizations the service requires during its execution. The author describes in detail, what the service requires. In the following we give examples of required network authorizations:

- Specification of the resources to which access is required:
  - Network
- Specification of the quantity of required resource:
  - What bandwidth
- Specification of required parameters:
  - What kind of socket (RAW, DGRAM, STREAM, ...)
  - What port
  - Which destination addresses (it is also possible to define an address space through the usage of wild-cards)

The RAD is added to the service and then the author signs the service and RAD. Afterwards the signed data is transferred to the service repository. Now the signed data is in the area of responsibility of the network administrator (see also figure 1).

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. The ideal case is a service which is labelled completely trustworthy due to a long experience with the author or extensive testing of the service. A service with such a label does not require any runtime supervision.

Services with a trust label apart from completely trusted require runtime supervision. Besides the node security policy each active node is additionally supplied with a list of DRADs for partly / completely untrusted services which are defined by the network administrator. It depends on the strategy of the administrator if he defines one set of DRADs for all or groups of active nodes or if he defines an individual set for each active node. On each active node a DRAD for each trust label is present. A DRAD defines the limiting conditions (resources and authorizations) for services of a specific trust label. Summarizing we state that the trust label defines the supervision under which the service must execute and further it indicates the DRAD which specifies the limiting conditions for the service. Furthermore, a mechanism is provided by our approach to integrate dynamically a new trust label in order to provide a required intermediate trust label. If the security daemon receives a service which is supplied with an unknown trust label the security daemon contacts the service repository and downloads the DRAD corresponding to the unknown trust label.

Adaptive Criterions An adaptive criterion is a criterion which depends on a / set of changing condition(s). For the evaluation of an adaptive criterion a specified set of queries / calculations must be performed in order to know whether the criterion is fulfilled. An example for an adaptive criterion is an expiration date. This means that first

the actual date must be evaluated and then, only if the expiration date is yet not reached the service could be executed on that active node. Another example is a performance test and only if the active node is performant enough the service could be executed. Or if two services could not execute on the same active node at the same time, this could be prevented through defining an adequate adaptive criterion and creating the corresponding active service.

Each adaptive criterion consists of input value(s) and an unique active service name which identifies the active service required for the evaluation of the adaptive criterion. This means the required input values and the unique name of the active service corresponding to the adaptive criterion are specified directly in the LAC. The security daemon depicted in figure 2 is responsible for the verification of the adaptive criterions. Whenever the download and installation of an active service for the evaluation of an adaptive criterion is required the security daemon initiates this. Then the service is fed with the corresponding input values and the adaptive criterion is evaluated. It must be mentioned that the main part of active services which are required for the evaluation of the most common adaptive criterions are already present on each active node and must not be downloaded from the service repository.

3.3 The Node Security Policy

The node security policy is used for the following aspects:

- Defining which resources and authorizations are generally available for the active services on that active node. Additionally, it also defines the complete quantity of each resource which could be maximally consumed by all actual running services on that active node.
- Part of the initial check: could the service be executed on that active node?
- Could be used for the creation of different quality of service nodes (see also below).

The node security policy is defined by the network administrator and could dynamically be changed at runtime including consequences for the actual running services. Through the definition of the node security policy the limiting conditions for the active services which could be executed on that active node are set.

The first task of the node security policy is the definition of resources and operating system services which are available to the services. Hence for each resource and operating system service the authorization could be given or negated. Additionally, upper limits for the resources which are available to the services in complete are defined. For example, an active node provides 500 MByte of main memory for the execution of services and furthermore, does not allow network access for any service whereas other active nodes could be configured differently. An active node located inside the Internet requires certainly another security strategy than an active node which is located inside an Intra-Net. For example only services originating from a specified list of authors or services with a specific trust label could be executed on a specific active node.

A component of the initial test (see also section 3.1) is the comparison of node security policy and service security policy. The service security policy must be a subset of the node security policy in order to allow the execution of the service.

Furthermore, a great degree of flexibility is gained through the integration of node security policies, as the network administrator specifies which services may be executed on which active nodes. Thus, the possibility is given to create quality of service domains through different sets of node security policies. For example inside a domain A only services of the highest trust label could be executed, this means that the services on these active nodes receive no supervision. Whereas inside a domain B services of all trust labels could be executed under a strict policy checking which requires fast hardware and produces higher costs but therefore, a higher availability is gained.

A third example could be “best effort” nodes, that perform only runtime supervision for network communications but not for other resources (main memory, CPU cycles, ...). This means, that it is granted that not any unauthorized service is able to send / receive any data besides the data stream it is acting upon and further on, no guarantee could be given that a specific service will execute with a specific performance guarantee on a best effort node as no further resource checking is performed.

3.4 The Default Resource and Authorization Descriptions

Each active node holds a list of DRADs and each DRAD specifies the authorization and resource limits for services of a specific trust label. Concluding for each trust label a DRAD is present. Further on, the DRAD list could be dynamically modified according to new circumstances which includes consequences for the running services. Thereby, the network administrator possesses the ability to change the limiting conditions for partly / completely untrusted services at runtime. The DRAD is also consulted during the initial test. Inside the node security policies the trust labels which are accepted by the active node are specified. Thus, if an active node allows principally the execution of services of trust label 1-3, it is still not said that all services of trust label 1-3 could be executed on that node. First a comparison between the RAD of the service and the DRAD belonging to the corresponding trust label must be performed and only if the RAD is a subset of the DRAD the service could be executed.

3.5 The Service Revocation List

Our approach includes also a reaction mechanism. Corresponding to the revocation lists that are known from certificates we integrated a service revocation list (SRL) in our framework. The SRL stores the services which never should be executed on an active node. Members of the SRL are services which tried to violate against the service security policy or services which the network administrator manually added. If on an active node a new service is inserted into the SRL, a mechanism is started which synchronizes the SRLs of all active nodes, thus that all SRLs are in the same state. After synchronization each active node verifies that the service that has been added to the SRL is not actually running on it. If yes, the service is instantly terminated. Additionally, the network administrator is able to add / delete (due to debugging) services manually to / from the SRL. In future work the reaction mechanism will be refined. A mechanism which classifies the violations against a policy into different offense levels will be implemented and further on, corresponding to the offense level of the violation the following reaction will differ.

3.6 The dynamic upgrading of Node Security Policy, DRAD and SRL

Node security policy, DRAD and SRL are realized in a manner that they can be configured dynamically at runtime according to changed circumstances. Furthermore, the elements are realized as structures which are stored in kernel space due to security and performance reasons. The security daemon is the only entity which is authorized to load kernel modules or to modify the mentioned kernel variables and therefore, a shared secret is established between security daemon and policy handler. The policy handler intercepts any user-space request to load a kernel module or to access a kernel variable and only if the security daemon (verification with the help of the shared secret) is the calling entity the request could be fulfilled. The shared secret thus establishes the mutual trust between security daemon and policy handler.

The configuration of an active node with its individual node security policy is possible through specifying its individual node name inside the node security policy. The security daemon compares the given node name with its own and only if they equal the node security policy could be updated. Of course, each transferred node security policy is digitally signed and supplied with a time stamp (version number) to prevent replay attacks. The dynamic upgrading of the node security policies in order to prepare the system for new authorization requirements is done in a similar manner as described above. The provision of new authorization requirements conducts into a new structure of node security policy, DRAD and requires a new ACKM for the supervision of the new policy entries. The modification of the RADs is done by the authors of the services which require at least one of the new authorizations. Whereas it is the responsibility of the network administrator to extend the node security policies and to realize the required ACKM. As described above, the network administrator adds the new authorization requirements to the actual node security policy and afterwards he updates it. If a service arrives at an active node with a service security policy in which authorizations are specified which the active node does not know the active node contacts the SR and verifies if there is a new node security policy for it. If yes, the new node security policy is downloaded and the initial check is executed. If not, the execution of the service is rejected.

Summerazing, we state that through the provision of the above explained mechanism new policies (node and service) respectively new authorization requirements could be easily integrated in an active networking environment.

3.7 Implementation and Measurements

In this section the measurements achieved with a first prototype for the active networking infrastructure AMnet [5] 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.

Most common operating systems make a distinct differentiation between application and operating system. 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.

The presented measurements focuses on the runtime supervision, the time spent on the initial check was left aside. The service used for the measurements created a socket. Afterwards it stepped into a loop of 1.000.000 cycles. In each loop cycle the service executed a *setsockopt()* command to change a socket option. This command was intercepted by the corresponding ACKM_Network. After intercepting a socket system call the security kernel module made a lookup in the local security database 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 and to find the proper policy a binary search algorithm is executed. [1] Table 1 shows the results of our measurements. The first line of the table

Table 1. 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
rel. overhead [%]	14.25	29.04	39.82	50.97	72.21

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 0.068 μs. Additionally, we extended the number of service security policies stored in the database up to 10.000. The additional overhead which could be observed is caused through the lookup for the proper service security policy. The last line of the tables represents the relative overhead. It can be seen that an additionally overhead of 30 % is quickly reached.

4 Related Work

Application-dependent and special-purpose policies are increasingly important. Certainly Java [4, 11, 16] must be mentioned, which supports mobile code including policies. But the execution performance of services realized in Java is not efficient enough compared with other programming languages and thus not adequate for many networking services, whereas our approach supports all programming languages. The Naccio project of MIT [2] uses also the mechanisms provided by Java. Joust [6] is a Java active OS implemented in Scout [12] and provides a Java virtual machine for the execution of active services.

[1] The measurements were made with a Pentium III/800 machine running Linux 2.4.18 kernel and AMnet version 1 [17]

Regarding active networking projects, the Seraphim project [10, 13] must be mentioned. It follows the approach of dynamic policies. A dynamic policy is a program consisting of a set of guards and actions. For the realization so-called active capabilities are used, which are actually executable Java-byte-codes.

PLANet [8] is another active networking project. The project uses a type-safe, resource limited, functional programming language with dynamic type verification.

The key idea behind history-based access control [1] is to maintain a selective history of the access requests made by individual programs and to use this history to improve the differentiation between safe and potentially dangerous requests. Handlers are integrated to maintain the history, but the bigger the numbers of services which execute on an active node the bigger becomes the amount of data to be stored.

Enforceable policies by Schneider [15] are specified through the usage of a so-called Security Automata. But still there are difficulties in generating the security automata.

To our understanding Java seems not to be the language of choice for many active networking services, therefore we provide a mechanism which is applicable for any language. In addition, through the realization of the ACKMs, functions corresponding to the adaptive criterions and the node security policies as active networking service, the mechanism is easily extensible and can be upgraded.

5 Conclusions

The presented policy framework is flexible and provides mechanisms to react dynamically to changing circumstances which is a very important aspect considering the needs of a network. The framework includes two types of policies, with the node security policy dominating over the service security policy. Furthermore, services which have not been reviewed by the network administrator or which are labelled partly / completely untrusted could still be executed on active nodes, in addition, the modular approach provides a flexible mechanism to integrate easily new authorizations requirements.

As shown in section 3.7 the runtime supervision of services, although widely implemented in kernel space, could get expensive if a service executes a huge amount of system calls which are intercepted within a short period of time. Through the approach of a scenario-tailored supervision and through the possibility to create QoS active nodes the performance degradation caused through the runtime supervision could be minimized so that our approach allows to realize demand-driven access control for performance-critical active services. Furthermore, the supervision of access requests of active services in kernel space is highly efficient and additionally allows to control applications written in any programming language. Through the modular approach of Access Control Kernel Modules, node security policies and services belonging to an adaptive criterion which are realized as active networking services the possibility is given to refine a service or to integrate a completely new service into the infrastructure dynamically. Finally the policy framework is designed in a manner that it requires little maintenance work. If the structure of an element (node policy, DRAD, ...) of the policy framework must be changed this modification must be made only once. This allows for efficient policy management in order to support policy updates for large networks of active nodes.

Bibliography

[1] Guy Edjlali, Anurag Acharya, and Vipin Chaudhary. History-based access control for mobile code. In *ACM Conference on Computer and Communications Security*, pages 38–48, 1998.

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

[3] S. Garfinkel G. Spafford. *Practical UNIX & Internet Security*. O'Reilly, 1996.

[4] Li Gong. Java security: present and near future. *IEEE Micro*, 17(3):14–19, 1997.

[5] Till Harbaum, Anke Speer, Ralph Wittmann, and Martina Zitterbart. Amnet: Efficient heterogeneous group communication through rapid service creation.

[6] John J. Hartman, Peter A. Bigot, Patrick Bridges, Brady Montz, Rob Piltz, Oliver Spatscheck, Todd A. Proebsting, Larry L. Peterson, and Andy Bavier. Joust: A platform for liquid software. *Computer*, 32(4):50–56, 1999.

[7] A. Hess, M. Schoeller, G. Schaefer, M. Zitterbart, and A. Wolisz. A dynamic and flexible access control and resource monitoring mechanism for active nodes. In *Proc. of OpenArch 2002, Short Paper Session*, New York, USA, June 2002. IEEE.

[8] Michael W. Hicks, Jonathan T. Moore, D. Scott Alexander, Carl A. Gunter, and Scott Nettles. PLANet: An active internetwork. In *INFOCOM (3)*, pages 1124–1133, 1999.

[9] A. J. Kfoury, R. N. Moll, and M. A. Arbib. *A Programming Approach to Computability*. Springer, Berlin, 1986.

[10] Z. Liu, R. Campbell, and M. Mickunas. Securing the node of an active network, 2000.

[11] N. V. Mehta and K. R Sollins. Expanding and extending the security features of Java. In *7th Usenix Security Symposium*, pages 159–172, 1998.

[12] David Mosberger and Larry L. Peterson. Making paths explicit in the scout operating system. In *Operating Systems Design and Implementation*, pages 153–167, 1996.

[13] R. H. Campbell P. Naldurg and M. D. Mickunas. Developing dynamic security policies. 2002.

[14] Jerome H. Saltzer and Michael D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, September 1975.

[15] Fred B. Schneider. Enforceable security policies. *Information and System Security*, 3(1):30–50, 2000.

[16] Dan S. Wallach, Dirk Balfanz, Drew Dean, and Edward W. Felten. Extensible security architectures for Java. In *16th Symposium on Operating Systems Principles*, pages 116–128, 1997.

[17] Ralph Wittmann and Martina Zitterbart. AMnet: Active multicasting network. In *COST 237 Workshop*, pages 154–164, 1997.