

# PLUGGABLE ACTIVE SECURITY FOR ACTIVE NETWORKS

ZHAOYU LIU, PRASAD NALDURG, SEUNG YI, ROY H. CAMPBELL, M. DENNIS MICKUNAS

Department of Computer Science  
University of Illinois at Urbana-Champaign  
Urbana, IL 61801  
{zhaoyu, naldurg, seungyi, roy, mickunas}@cs.uiuc.edu

## ABSTRACT

*Security is of critical importance to the success of active networking. In addition, we argue that active security based on active networking principles can offer a wide range of opportunities to build better security systems. This paper describes the integration of active security into a software system implementing the active network architecture. The paper demonstrates that an extensible, reconfigurable security architecture based on active networking is flexible and accommodates a wide variety of security policies and mechanisms. The active security provides users the ability to dynamically create and enforce highly customized and situational policies for their applications. The active security also permits security systems to react to intrusion and can aid the application of the "need-to-know" security principle to network software and application security.*

**Keywords:** active networks, security, reconfigurable, active capability, interoperability

## 1 INTRODUCTION

An active network provides a software framework that enables network applications to customize the processing of their data [1, 2]. Active applications inject capsules that contain programs (along with data) into the network. Active routers dynamically install these programs and execute them on the data. Though this facilitates fast protocol and service deployment it also makes the routers vulnerable to attacks from arbitrary user-code. Securing the routing infrastructure against threats and exposures remains a major challenge in this paradigm [3].

Traditional networks rely on the underlying operating system to implement security mechanisms and policies. The traditional definition of security in a network environment includes authentication, access control, and encryption. Applications and routers establish a basis for trust by mutual authentication. To protect the integrity of the contents of the capsules, encryption and digital signatures can be employed. Access control mechanisms or policies are defined and enforced to

provide controlled access to the router resources. In addition, active network routers have to provide support to

- prevent malicious behavior of arbitrary user code and
- protect the user code and data from malicious routers

Though a wide range of policy types [4] and mechanisms [5] have been proposed, underlying operating systems implement only a static subset of these policies and mechanisms. Applications that want to use sophisticated or customized policies have to make do with lesser or weaker security guarantees. The overhead associated with adding new policies and mechanisms can also be prohibitive.

In order to exploit the active network flexibility, we have developed a dynamic, fully extensible, interoperable security architecture based on and built into the underlying active network architecture [6]. We term this approach active security [7]. The security architecture enables both static and runtime application-aware reconfiguration [8]. Adaptation allows the security provisions of the network to meet specific individual security requirements within different application scenarios. Applications can request specific security policy instantiations on specific parts of the network, distributing the relevant security policies on a "need-to-know" basis. Alternatively, changes in the security policies for the network can be triggered by the invalidation of a trust model, perhaps by the detection of intrusion or other abnormal behavior.

In this paper we describe the integration of active security into a software system (Bowman and CANEs [13, 14]) implementing the active network architecture [12] to showcase the above claimed advantages. Our active security system is composable and can be easily plugged into current active network systems. The integration demonstrates that the active security can provide users the ability to dynamically create and enforce highly customized and situational policies for their applications. It also shows that the active security can permit security systems to react to intrusion and can

aid the application of the "need-to-know" security principle to network software and application security.

The rest of the paper is organized as follows. Section 2 overviews our Seraphim active security architecture. Section 3 describes the integration of our architecture into a software system implementing the active network architecture. Section 4 presents an application example to show the flexibility of the active security based on the current implementation. Section 5 shows the preliminary performance measurement. Section 6 describes the future plan of the integration. The last section concludes this paper.

## 2 SERAPHIM: ACTIVE SECURITY ARCHITECTURE

Seraphim is a dynamic, flexible, and application specific security architecture that exploits the active, dynamic functionality provided by active networking using an active capability (AC) [6, 9]. Essentially an AC is an executable Java code, which concisely represents dynamic security policies and mechanisms. ACs are evaluated by a security guardian in a secure sandbox environment and the security guardian enforces the security requirements of AC evaluation results. We describe the architecture in more detail next.

### 2.1 ACTIVE CAPABILITY

Active capabilities are used to support flexible distributed dynamic security policies and services control employing the same active principles as active networks [6, 9]. Unlike a traditional capability, which is merely a static authorization credential that encodes the principal and the permissions associated with the principal, an active capability is a customized piece of code that encodes the type of access control policy and other constraints used in the access control decision making process. In our implementation, an AC is an executable Java code that concisely represents dynamic security policies and mechanisms. In addition, an active capability is protected by digital signatures, resides in user space and can be freely passed around.

An active capability can carry all the decisions policy information in its code. This way of implementation is not modular, elegant and efficient. A better way is to have a generic policy framework that supports different various policy types. ACs use the framework to implement specific policies. An application presents an active capability along with its regular data or protocol capsules to the active router's security guardian at execution time. The access control policy type and user credentials are extracted from the capability. The remote router's security guardian recreates the context of the policy type within its policy framework. If at any point during this process the policy framework discovers that it does not have an implementation for the type of the

policy, it downloads the code dynamically into the framework, using the underlying active network. It then instantiates the run-time parameters associated with the application in its sandbox-like environment and executes the active capability in this environment. Based on the result of the evaluation of this active capability, the access control decision is enforced.

The principal of the active capability, which can be a user, a role, or other principal, must be authenticated by a trusted authority. The trusted authority acts as the policy server in our system. The policy server is responsible for generating ACs, serving ACs to applications and keeping track of ACs. Usually one or more policy servers are associated with each protection domain. Application programs contact their nearest or least-loaded server and obtain the active capability dynamically.

### 2.2 POLICY FRAMEWORK

The policy framework is an object-oriented and coded in Java. This allows users and commercial organizations to specify policies tailored to their specific operational needs. The framework itself is a hierarchy of classes as shown in Figure 1.

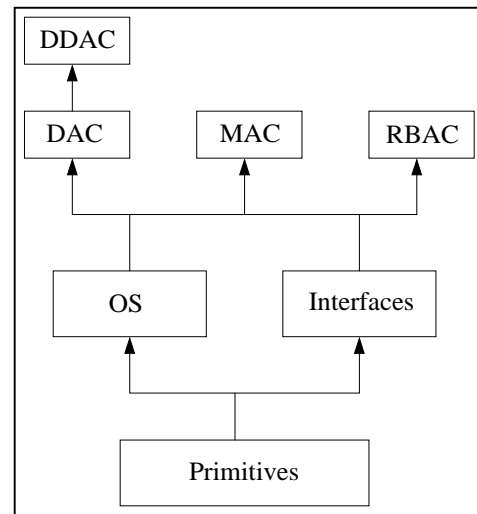


Figure 1: Component-level Map of the Policy Framework

The framework is dynamically configurable and extensible. The classes at the bottom of the framework are mostly abstract and are mainly used to represent mathematical concepts such as sets and mappings. These classes form the basis for a hierarchy of successively incremented specialized classes representing concepts such as labels and access control lists. Finally, at the top of the framework are classes that can be used to represent a variety of generic policy forms.

The policy framework supports the following common types of access control: Mandatory (MAC),

Discretionary (DAC), Double Discretionary (DDAC), and Role-based (RBAC) [10]. More application specific access control policy systems can be easily extended from this object-oriented framework ([11] provides several good examples). In our model, we can specify not only the traditional  $\langle \text{subject}, \text{object}, \text{operation} \rangle$  access control triple, but also include a resource limit on usage, situational decision rules, constraints and dependences, e.g., based on current time of the day or current role of the principal. The main policy type we use for active networks is RBAC because of its flexibility. We will describe its usage in more detail later.

### 2.3 SECURITY GUARDIAN

The security guardian in the architecture supports AC evaluation and enforcement. The security guardian's functionality is similar to a traditional reference monitor. All accesses to node resources must go through the security guardian. The security guardian uses the security library services to verify the signature on the active capability. To carry out the intended security operations specified by ACs, an evaluation engine and an enforcement engine are included in the security guardian. The evaluation engine evaluates ACs in a secure sandbox. The enforcement engine interacts with other NodeOS components to enforce faithfully the security operations, using the security library services. The enforcement engine can initiate security actions based on ACs requirements. So the security guardian may trigger or initiate security actions. The triggers can be intrusion detection alarms, or explicit requests by execution environments (EEs) or applications that use active networking features. For example, the security guardian can initiate installing firewalls dynamically [6].

## 3 INTEGRATION OF SECURITY INTO ACTIVE ARCHITECTURE

This section describes the integration of the above security system into a software system implementing the active network architecture [12]. The software system has two parts: the Bowman NodeOS and the CANEs execution environment [13, 14]. We first briefly overview the Bowman and CANEs systems, and then present the integration.

### 3.1 OVERVIEW OF BOWMAN AND CANES

The Bowman node operating system is built to support the CANEs EE. It is designed around three key abstractions: channel, a-flow, and state-store. A channel is the primary abstraction for communication and an a-flow is the primary abstraction for computation. The state-store provides a mechanism for a-flows to store and retrieve state that is indexed by a unique key. The Bowman is layered on top of a host operating system that

provides lower level services. To make the elementary Bowman channel, a-flow, and state-store abstractions more useful for users, Bowman provides an extension mechanism that is analogous to loadable modules in traditional operating systems. Using extensions, the Bowman NodeOS interface can be extended to provide support for additional abstractions such as queues, routing tables, user protocols and services ([15] provides a more complete NodeOS API).

The CANEs EE is built on the top of the Bowman NodeOS. It provides a composition framework for active services based on customizing a generic underlying program by injecting code to run in specific points called slots. The composition model basically has two parts. The first part, the underlying program, is a fixed part for uniform processing applied to every packet. The second part, the injected program, is a dynamic part that provides user-specific functionality for routing and processing the packets. The injected program is dynamically executed at the appropriate specific points (*slots*) in the underlying program. CANEs uses signaling messages to control the injected programs.

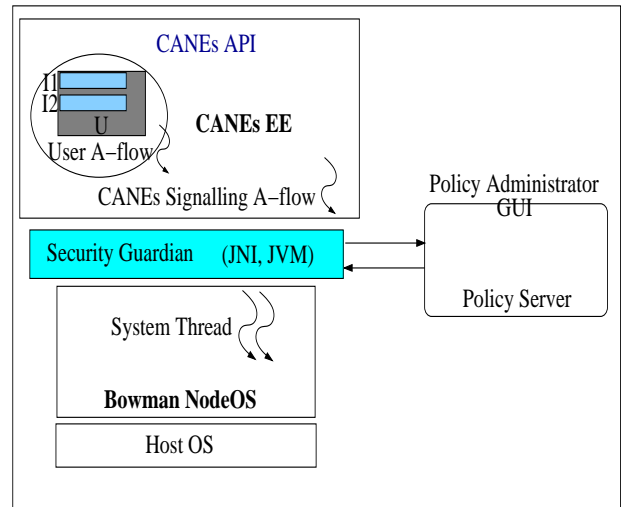


Figure 2: Integration of Active Security into Bowman and CANEs

### 3.2 INTEGRATION

The integration of active security, CANEs and Bowman is shown in Figure 2. The security guardian is a thin layer between the Bowman NodeOS and the CANEs EE. The Bowman NodeOS interfaces are replaced by the security interfaces. The security guardian does the following security checkings [16]:

- Authentication: It verifies the identification and the signature of the request messages. We use X.509 certificates [17] and a simple public key infrastructure (PKI) for authentication.

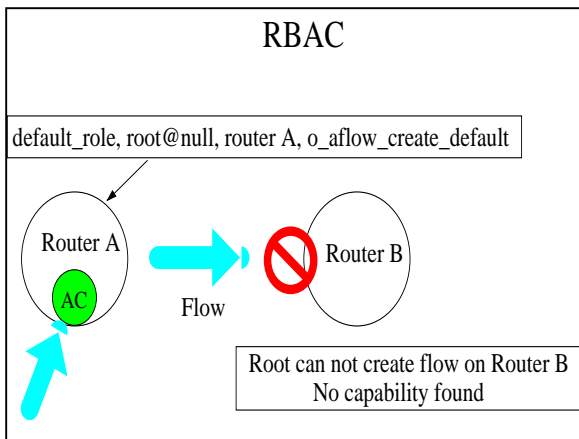


Figure 3: First Demo Scenario

- Authorization: If authentication succeeds, it then checks the access permission for the requests. This requires fetching and evaluating ACs.

Since Bowman and CANEs are written in C to obtain high performance and the Seraphim architecture is implemented in Java for interoperability and security purpose, we use JNI (Java Native Interface) in Bowman and CANEs to invoke the security guardian in Java. When Bowman starts, it starts the security guardian component that invokes Sun JVM. Each security check from CANEs to the Bowman NodeOS security interface is attached to the Sun JVM as a Java thread. After the checking, the Java thread is detached and destroyed.

The security guardian obtains ACs through a secure channel from the policy server. The policy administrator uses a GUI that allows users or system administrators to create and define policy specific attributes and to generate active capabilities. The GUI allows the administrator to create role definitions and associate users and permissions with the role, and supports other functionality (see [10] for more details).

#### 4 APPLICATION EXAMPLE

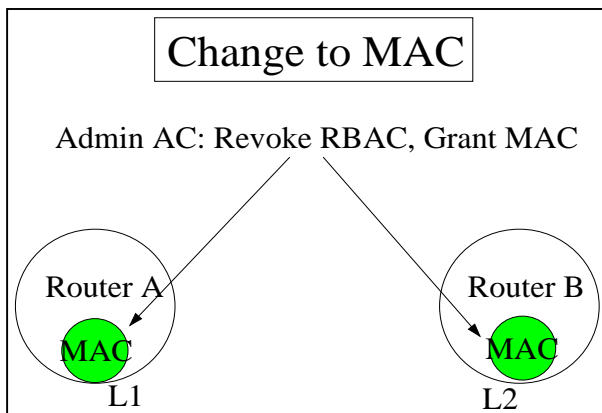


Figure 5: Third Demo Scenario

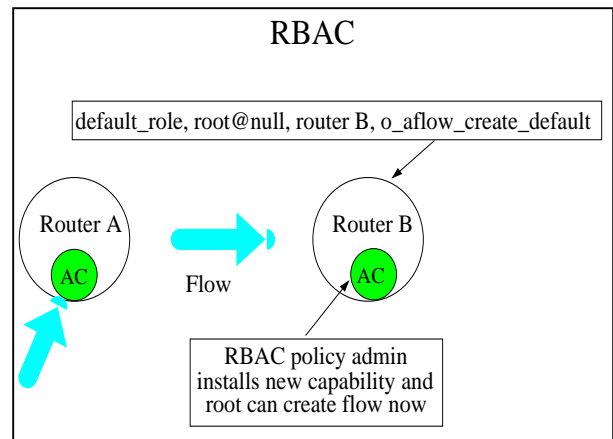


Figure 4: Second Demo Scenario

We have implemented a preliminary version of the authorization part of the integration. Based on the authorization part, we have developed an example application scenario, which is shown in Figure 3, 4, 5 and 6. Figure 3 shows that on behalf of user *root@null* of role *default\_role*, the CANEs EE can create an a-flow on the router A, but not on the router B. In order to have a complete flow path from the user *root@null* to the router B, we can dynamically create a new AC through the policy GUI and install it at the router B. Now the CANEs EE can create an a-flow on both router A and B on behalf of the user *root@null* of role *default\_role* (Figure 4). The policy type of Figure 3 and 4 is RBAC. If we want to have a stricter and less flexible policy we can dynamically change RBAC to MAC (Figure 5). In this case, the trigger for the policy type change may be an intrusion detection alarm. In MAC policy every entity is assigned a security level. A hierarchy is defined in terms of these levels. Subjects with lower levels cannot read from objects of higher levels and subjects with higher levels cannot write to objects of lower levels. We assume that the MAC level L1 is higher than MAC level L2. This means that the router B has lower security level than the router A. So if user *root@null* is also at security level L1, then user *root@null* can create an a-flow on only the router A since user *root@null* cannot write to router B (Figure 6).

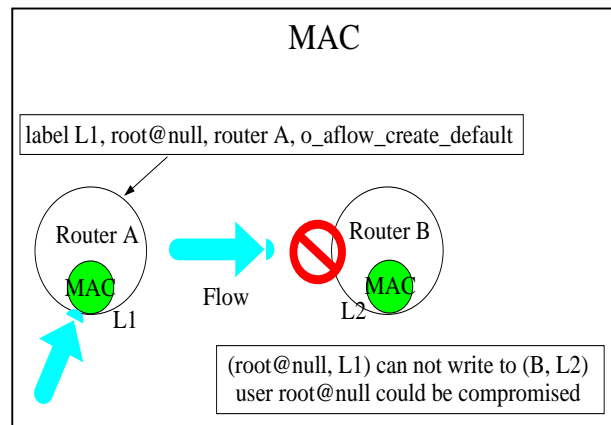


Figure 6: Fourth Demo Scenario

## 5 PERFORMANCE

The overhead that the integration introduces includes the JNI invocation overhead and the regular security overhead. The regular security overhead, which includes AC fetching and evaluation, is necessary for flexible access control and has been studied previously [6]. We used a simple active ping application (*atraceroute*) between two machines A and B to measure the JNI invocation overhead. Machine A is a Sun Ultra-5 machine, and machine B is a Sun Ultra-2 machine. Both A and B are on the same 100Mbps Ethernet LAN. Machine A sends an *atraceroute* to machine B that is running the Bowman NodeOS. We measure the round trip time (RTT) of the *atraceroute* command with and without JNI invocation (When with JNI invocation, we let security guardian simply return a *true* value in order not to include the regular security overhead). Without any optimization, the RTT is about 2400ms without JNI invocation and about 9100ms with JNI invocation.

In order to improve the performance, we plan to have a leaner JVM replace the current Sun JVM. A possible choice is Kaffe JVM [18]. A more dramatic improvement would be to use a simpler language than Java for ACs. The sandbox evaluation engine of the security guardian of the simpler language must be efficient.

## 6 FUTURE PLAN

We plan to extend the current integration implementation to provide security checking for all CANEs signaling messages. We plan to add authentication and dynamic revocation to the integration, using the security NodeOS API [16]. We also plan to integrate the Denial of Service prevention features into the system. Finally we will install an experimental setup for the flexible, secure, and composable demanded video distribution application [19] to demonstrate the secure composable services for active networks.

## 7 CONCLUSION

This paper describes the integration of the Seraphim active security into a software system implementing the active network architecture [12]. The active security architecture is dynamic, fully extensible, interoperable and is based on the underlying active network principles. The integration demonstrates that the active security architecture can be easily plugged into the active network architecture such as Bowman and is flexible and accommodates a wide variety of security policies and mechanisms. We show that active security can provide users the ability dynamically to create and enforce highly customized and situational policies for their applications.

We also show that the active security can permit security systems to react to intrusion and can aid the application of the "need-to-know" security principle to network software and application security. We believe that exploiting active security is a step in the direction of designing a comprehensive and flexible framework to integrate various security mechanisms and services into the active network architecture.

## 8 ACKNOWLEDGEMENTS

We would like to thank Matt Sanders, Ken Calvert and Ellen Zegura to help us understand the Bowman NodeOS and CANEs execution environment system.

This research is supported by DARPA F30602-98-1-0192.

## REFERENCES

- [1] D. Wetherall, J. Guttag, and D. Tennenhouse. ANTS: A Toolkit for Building and Dynamically Deploying Network Protocols. In *OPENARCH'98*, 1998.
- [2] D. Wetherall, U. Legedza, and J. Guttag. Introducing New Internet Services: Why and How. In *IEEE Network Magazine*, July 1998.
- [3] S. Murphy, ed. Security Architecture Draft. AN Security Working Group. Draft.
- [4] Ravi Sandhu. Role-Based Access Control. In *Advances in Computers*, Vol. 46, Academic Press, 1998. Also at <http://www.list.gmu.edu/articles.htm>
- [5] The SwitchWare Project Homepage. <http://www.cis.upenn.edu/~switchware/>
- [6] Zhaoyu Liu, Prasad Naldurg, Seung Yi, Tin Qian, Roy H. Campbell, and M. Dennis Mickunas. An Agent-based Architecture for Supporting Application Level Security. In *the DARPA Information Survivability Conference and Exposition*, Hilton Head Island, SC, January 2000.
- [7] Roy H. Campbell, Zhaoyu Liu, M. Dennis Mickunas, Prasad Naldurg, and Seung Yi. Seraphim: An Active Security Architecture for Active Networks. Tech. Report 2137, Department of Computer Science, University of Illinois at Urbana-Champaign, November 1999.
- [8] Roy H. Campbell, Zhaoyu Liu, M. Dennis Mickunas, Prasad Naldurg, and Seung Yi. Seraphim: Dynamic Interoperable Security Architecture for Active Networks. In *IEEE OPENARCH 2000*, Tel-Aviv, Israel, March 2000.
- [9] Roy H. Campbell, M. Dennis Mickunas, Tin Qian, and Zhaoyu Liu. An Agent-based Architecture for Supporting Application Aware Security. In *the Workshop on*

*Research Directions for the Next Generation Internet*,  
Vienna, VA, May 1997.

[10] Vijay Raghavan. On the Design and Implementation of a Security Policy Administration for a Dynamic Security System. Master's Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, May 1999.

[11] Tim Fraser. An Object-Oriented Framework for Security Policy Representation. Master's Thesis, Department of Computer Science, University of Illinois at Urbana-Champaign, December 1996.

[12] K. Calvert, ed. Architectural Framework for Active Networks. AN Architecture Working Group. Draft.

[13] S. Merugu, S. Bhattacharjee, Y. Chae, M. Sanders, K. Calvert, and E. Zegura. Bowman and CANEs: Implementation of an Active Network. In *Proceedings of 37<sup>th</sup> Annual Allerton Conference*, Monticello, IL, September 1999.

[14] The CANEs Project Homepage.  
<http://www.cc.gatech.edu/projects/canes/>

[15] L. Peterson, ed. NodeOS Interface Specifications. AN NodeOS Working Group. Draft.

[16] Zhaoyu Liu, Roy H. Campbell, and M. Dennis Mickunas. Securing the Node of an Active Network. In *Active Middleware Services*, Salim Hariri, Craig Lee, and Cauligi Raghavendra (editors), Kluwer Academic Publishers, Boston, MA, September 2000.

[17] C. Adams and S. Farrell. Internet X.509 Public Key Infrastructure Certificate Management Protocols. RFC 2510, March 1999.

[18] The Kaffe Homepage. <http://www.kaffe.org/>

[19] The PANAMA Project Homepage.  
<http://www.tascnets.com/panama/>