Strong Security for Active Networks

S. Murphy, E. Lewis, R. Puga, R. Watson, and R. Yee

Abstract—
Active networks are an exciting development in networking services in which the infrastructure provides customizable network services to packets. These custom network services can be deployed by the user inside the packets themselves. Furthermore, the custom network services require that the infrastructure perform much more sophisticated operations on packets than the traditional forwarding. Consequently, there are heightened concerns from users and network operators about security. This paper discusses security requirements and issues in active networks with respect to authentication and authorization in a node. We describe our prototype implementation of a solution to those issues. We go on to describe a security architecture derived from our experience and relate our prototype to the architecture.

Keywords—**active networks, authorization, authentication.**

I. Introduction

Active networks have been in development for the last half decade [1]. This is a promising new technology which has two primary features: it permits rapid deployment of new network services and it provides for complex computations to be performed on packets within the infrastructure. The ability to deploy new services can even be granted to the user and tied to the transmission of user packets. The infrastructure computation can modify or generate packets on the fly.

However, both these features of active networks heighten security concerns of clients of the technology. Network operators have concerns about allowing arbitrary code, particularly arbitrary code supplied by end users, to execute in their nodes. Users have concerns about the effect of the infrastructure computations on the data they are transmitting through the network.

Despite significant energy devoted to security research in active networks [2], [3], [4], [5], [6], [7], [8], the issues of security are by no means solved. This paper attempts to describe the security requirements in active networks and the challenges of meeting those requirements (Section 2). We describe our own implementation of a solution to a subset of those challenges (Section 3). We present a security architecture that is suggested by our solution and relate that architecture to our implementation (Section 4). We finish with a description of some related work and a summary of the results.

S. Murphy, E. Lewis and R.Watson are affiliated with NAI Labs at Network Associates in Glenwood, MD, sandy,lewis,rwatson@tislabs.com. R. Puga is affiliated with MyCIO.com in Glenwood, MD, rpuga@mycio.com. Richard Yee is affiliated with Sun Microsystems, in El Segundo, CA, richard.yee@west.sun.com.

This work was supported by Defense Advanced Research Projects Agency (DARPA) under contracts N66001-97-C-8514 (Secure Active Network Prototypes) and N66001-98-D-8508 (New Cryptographic Techniques for Active Networks), using the Space and Naval Warfare Systems Center (SPAWARSYSCEN) as the contracting and technical agent.

II. Security Requirements and Challenges

A. Active network description

An active network transmits active packets which contain active code or references to code. An active node extracts or retrieves the code and executes the code in the node with the packet as input. The result of the execution can be a modification of the state of the node, a modification of the packet, or the transmission of one or more packets. Legacy packets may be forwarded as before or may themselves be the recipient of custom network services.

The DARPA Active Network community has defined an architecture for an active node [10] that depicts a node as comprising a NodeOS and one or more Execution Environments. The Execution Environments (EE's) provide a programming interface or virtual machine that can be programmed or controlled by the active packets.

The NodeOS interface document [9], [11] of the community describes the resource abstractions of an active node operating system as thread pools (computation), memory pools (memory), channels (communication), files (persistent storage) and domains.[1] Domains are the primary abstraction for resource management and are allocated a set of channels on which messages are transmitted, a memory pool and a thread pool. Channels are the primary abstraction for communication of packets into and out of the node. Some channels are anchored in an EE, with packets arriving on an in-channel and transmitted on an out-channel. Packets are assigned to an in-channel on the basis of a packet filter key provided in the creation of the domain. Channels not anchored in an EE are cut-through channels and involve no processing of active code. A cut-through channel can be created by a domain running in an EE from an existing in-channel and out-channel and continues to draw on the resource limits (e.g., memory, bandwidth) of the creating domain.

A domain may create a subdomain at any time. The principal assigned to a domain is established at creation time and governs the activities of the domain thereafter. The node begins operation with one or more domains assigned to EE's. An EE is permitted to start subdomains with more explicit packet filters when and as it wishes. Therefore, the EE makes its own decision as to whether the active code in a packet will run as part of the EE's domain, with the EE's principal privileges and resources, or in its own separate domain, with the packet's principal privileges and separately allocated resources.

[1] "Domains" were termed "flows" in earlier publications within the active network community. The term was changed because of the possible confusion of semantics with the usual use of "flow" in networking.

The active packet arrives, is assigned by the node to a in-channel of a domain (either anchored in a EE or a cut-through channel) and is processed. If the active packet processing occurs in an EE, then the active code executes with access to the EE services and resources. The EE interface might pass the NodeOS interface through to the active code or might replace or augment that interface with its own services. The processing results in active packets being transmitted on an out-channel. The transmitted packets might be forwarded input packets, modified input packets, or newly inserted packets. The processing of the packet might also result in persistent changes in the node state.

Active packets can be transmitted through the Internet by using the Active Network Encapsulation Protocol (ANEP). ANEP provides its own header format which includes a Type ID field that identifies the EE to which the packet should be transferred. The ANEP header also provides for options defined as type-length-value fields. Options defined in the ANEP protocol include source, destination, integrity and end-end authentication.

B. Security requirements

¿From the description of the active network, one can see that there are many entities in an active network that have assets that they would want to protect. The end user at the source and destination, the active node itself, the execution environments and the active code/domain all have security concerns.

The end user retains the traditional concerns about the authenticity, integrity, and confidentiality of the packet's payload data as it traverses the network. As the active code may create persistent state in the active nodes it traverses, the end user will have the same concerns about data created in the infrastructure as well as concerns over access to that data.

The active node's security concerns are likely to be concentrated on authorization of use of the node's services and resources, in order to maintain availability of use. It will, of course, also be concerned about the integrity and confidentiality of its own state. The EE has the same concerns for its own services, resources and state.

The active code (standing as proxy to the end user who launched the packet) has security concerns that are related to access to its services (e.g., access to the domain in which it is executing) and access to sharable persistent state it creates.

C. Trust model

End User Viewpoint. The end user would rather not have to trust all active nodes, execution environments, and other active code in the active network. Therefore, it must view the nodes, EE's and other active code/domains as potential threat sources.

Baring results from certain research areas regarding mobile agents (running on untrusted hosts), there are few ways to assure the end user that its data will be protected from attacks (exposure, unauthorized use or modification, etc.) by the node or the EE in which its packets are processed in the clear. The end user may apply end-end cryptographic protections against these attacks and not make the node or EE privy to the cryptography, so that the data is not in the clear in the node. While end-end cryptographic protections limit the damage that the node or EE can cause to the data, they do limit the network services that can be performed for the packet. The only other assurance the end user has that it can be protected against attacks by the node or EE comes from an ability to direct the active code to avoid transmitting the packet to untrusted nodes or execution environments. This presupposes that the end user has some method of identifying nodes and EE's that are trusted and authenticating the nodes and EE's the active packet encounters, and that the packet transmission will be under the exclusive control of the active code.

The picture for protection against unauthorized access or use attacks on the end user's data from other active code/domains in the infrastructure is a bit more reassuring. The (trusted) node and EE can provide enforcement of the end user's authorization policy, as long as they have the ability to authenticate the principals associated with each active code/domain and are provided the end user's policy.

Node Viewpoint. The node has its own view of the threat sources in the active network. It should not be necessary for the node to completely trust the EE's it executes. It would certainly be unwise to architect the system so that it must trust the active code it runs or the end users who generate packets. Therefore, the node would view the EE's, the active code, and the arriving packets as potential threat sources.

Because the node architecture grants the EE the right to start subdomains if and when it wants, requests from the EE for NodeOS services might be intended to provide services for a packet not assigned by the EE to a subdomain. Such a request will be judged by the node on the basis of the privileges granted to the EE's domain. The NodeOS must trust the EE to properly use its own privileges on behalf of active code that is *not* assigned to a subdomain. That is, the NodeOS must trust that the EE is adhering to the node authorization policy in requesting NodeOS services on behalf of an active code.

Because the node has control over the allocation of resources and privileges to an EE's domain, it has the opportunity to mitigate the possible damage from an EE. It can balance the trust it holds in an EE with a judicious allocation of resources and privileges. Fully trusted EE's might be provided with more resources and more powerful privileges than less trusted EE's.

The threat from active code can be controlled because the node has the opportunity to enforce its own authorization policy for the actions of any domain. Finally, countering clogging attacks from arriving packets is a research area of its own. In short, protection against clogging attacks requires that the node's neighbors cooperate with traffic limits and that the node establish traffic limits with its neighbors that in the aggregate do not exceed its capacity.

EE Viewpoint. The EE sees the same threats from

active code and arriving packets as the node sees. It has the same opportunities to control the threat from active code by enforcing its own policy governing access by active code. The EE can rely on the node to enforce the EE's policy governing acceptance of arriving packets, as long as its required authentication of the packets is within the capabilities of the node, i.e., does not require some EE specific authentication mechanisms, and the node is provided with the EE's policy. The EE sees potential attacks from other EE's through shared persistent state or access to its services. As these access methods must be provided by the NodeOS, the EE must rely on the node to enforce the EE's policy.

Active Code Viewpoint. The active code itself would rather not have to trust all the nodes, EE's and other active code in the network. Unfortunately, it is in the same situation regarding trust in the node and EEs in which it runs as the end user is. The active code must trust the nodes and EE's on/in which it executes and avoid those it does not trust.

The active code sees potential attacks from other active code through access to shared persistent state or services. These access methods are provided by the EE and so the active code can rely on the EE to enforce its policy.

Protection Techniques. The active network community employs two ways of ensuring that possible attacks are avoided. The first is to limit the possible actions to those that would be safe for any entity in the system to perform. These are language based approaches, involving type-safe and namespace limiting languages. This is a low cost technique with a large payoff.

But it is not always possible to eliminate dangerous activities entirely. There will be some actions that some, but not all, entities should be permitted to perform. The second class of techniques associates a principal with each request for an action and enforces a policy that states which principals are permitted to perform which actions. These are authorization based approaches.

In our work we have concentrated on authorization enforcement to protect active networks and the authentication to support authorization enforcement.

D. Authentication Challenges

Some of the most challenging aspects of securing active networks concern the authentication support for authorization. Authorization decisions require the authentication of the entity making a request. Authentication normally implies the use of cryptographic techniques. But the application of existing cryptographic techniques to the active networks environment presents certain challenges.

First, the identification of the principal itself in active networks is challenging. Existing Internet interactions are typically client/server, where the explicit individual identity of the client and server are important. The Internet community has begun to move away from explicit individual identities to attribute based identities (X.509 Attribute Certificates, KeyNote and PolicyMaker, etc). Even so, the client and server have a common understanding of the identities or attributes that are important.

In active networks, the aspects of the principal's identity that are important may change radically as the packet traverses the network. Within the end user's enterprise network, the individual's identity or company role may be important. But beyond the immediate network of the end users, it is not likely that the individual identity of the end user will be important. Aggregate security attributes will be more likely to be used, which may be labels, groups, etc. Furthermore, the aggregate attributes may themselves differ in different domains. The first level ISP may have a local identification of the principal and attributes. The second level ISP that transits the packet may only be interested in authenticating that the traffic came from by a peer. Consequently, there may be multiple and varying principal identities or attributes that are important.

Second, the choice of an authentication mechanism presents challenges in active networks. Existing mechanisms for providing authentication protection of a packet are rooted in the existing Internet paradigm of client and server based communication (e.g., IPSEC, TLS, Kerberos, IKE, even API's such as GSSAPI). These will not be sufficient in an active network environment where the packet needs to be authenticated at source and destination and potentially every node in between. The existing solutions can be used hop-hop in the path but that provides little in the way of end source authentication. If all nodes in the active networks can be trusted and the edge node correctly identifies the end source, then hop-hop protection provides sufficient authentication. However, experience in the Internet (e.g., wide-spread Internet outages caused by one faulty router) has provided ample proof that it would be folly to trust all nodes as a set. Existing solutions can also be used to set up multiple security associations, one between the end user and each node on the path. The latency and bandwidth requirements to establish each association and multiply protect the packet would be prohibitive.

Even though hop-hop protections do not provide strong end-end authentication, hop-hop integrity protections are still important. Integrity protection between neighboring active nodes provides protection against attacks from outsiders, and should include protection against replay, modification and spoofing. This first level of protection is particularly important in neighbor to neighbor exchanges or signalling.

When hop-hop protections do not provide sufficient end-end authentication of the principal associated with a packet, we can employ end-end protections. However, the use of end-end cryptographic techniques is also a challenge in active networks. Symmetric techniques could be used if a key associated with the principal could be installed at each node of the packet's path through the network. The packet modifications at each node could be protected anew with the shared key. However, this has a similar trust drawback as using hop-hop protection: every node on the path must be implicitly trusted. Also, the assurance of authenticity of the principal, derived from the shared key, is diluted if the key is not unique to the principal and the path. For

the strongest assurance, each new communication would require key distribution or agreement among the nodes of the path. This expensive operation would be unsuitable for a datagram model of communication and would motivate not only a connection oriented model for communication but a virtual circuit model, where all packets in a flow of packets are protected by the same key and transit the same nodes.

Asymmetric techniques (i.e., digital signatures) can operate in a datagram model but have difficulty protecting packets that change. Signing a packet with a digital signature provides a cryptographic association from the signer to every potential verifier of the future. Therefore, authentication by digital signature is suited for a datagram model of communication, where the packet may decide in route what nodes it will visit. The digital signature protection provides the strong end source authentication that we wanted. However, the asymmetric private key that is used to produce the signature should be kept secret by the signer to maintain the security features of a digital signature. This means that the private key would not be known to the infrastructure nodes that process the packets. Consequently, infrastructure nodes cannot produce a new end source signature if the packet is modified in transit. It might be possible to have each modifying node sign just the modifications it makes, but such a scheme produces massive packet growth. Finally, the performance issues with asymmetric cryptographic techniques in terms of both performance and bandwidth are well known.

III. SANTS

We choose as our problem to prototype a secure active network that could provide authorization enforcement to the nodes, EE's and active code and integrity protection to the packet, with a distributed authentication mechanism that provided for retrieval of identities and attributes in a wide area network and dynamic assignment of attributes to a packet as it traversed the network.

A. Components

We created an execution environment supporting strong security by extending the MIT ANTS Execution Environment [13]. Our implementation, called SANTS for Secure ANTS, adds the following to ANTS:

- X.509v3 certificates
- DNSSEC for storage of the credentials
- Java Crypto API and Java Cryptographic Extensions crypto provider
- Keynote policy system
- Java 2 security features
- a separation between EE and Node classes
- a shared data capability, which we called BulletinBoard

We used the X.509v3 certificates as our globally unique principal credentials. We used some standard fields of the X.509v3 (like the organization field in the distinguished name) as security attributes. X.509v3 certificates also can include extensions that provide a mechanism to store attributes that are not represented in the standard fields. The Java Cryptographic Extensions package was used to apply and check the cryptographic protections.

We stored our certificates in DNS CERT records, protected by the DNSSEC security features [12]. This provides a secure distributed storage and retrieval mechanism for the certificates. Each certificate can be located by the fully qualified domain name of its CERT record. Therefore, we included in each certificate the fully qualified domain name of its issuer's certificate, to aid in distributed certificate chain processing. Consequently, as an active packet traverses the network, it is always possible to locate the certificates needed to fully verify the certificate signatures.

We used the Keynote policy system both as a ubiquitous policy language and as a policy evaluator. This allowed the end users to include their own authorization policies (e.g., for state created in the infrastructure) in the active packets they create, and expect that the NodeOS and EE in each node could understand and enforce that policy.

We used the concept of permission objects and protection domains of the Java 2 Security Architecture to provide the authorization policy enforcement. Active code was loaded with a class loader which we constructed. The class loader created the protection domain for the active code with a permission object containing the credentials associated with the active packet. The permission object *implies* method made a call to the Keynote policy evaluation engine. Methods that needed to be protected called the Java 2 Access Controller to inspect the stack and ensure every protection domain on the stack was authorized to perform the method.[2]

We built a shared data capability because we were interested in exploring the issues associated with creating shared data in the infrastructure. Consequently, the BulletinBoard feature is not sophisticated, but exhibits the abilities that we wished to protect, i.e., to create and access items. This feature was used in an application based on work in the University of Pennsylvania Switchware project, called Flow Based Adaptive Routing. In this application, "scouts" are sent out to find a best path through the network. The scouts flood themselves through the network, posting hints in the Bulletin Board of the best path found. In our adaption, the scouts also carry policies (expressed in Keynote) that govern access to their BulletinBoard items. The policies are based on the attributes carried in the X.509 certificates, such as the role or organization of the requester.

B. Authentication

As a first level of protection of the active network, we used HMAC-SHA1 integrity protection between each pair of neighboring nodes. This would prevent spoofed packets and modification of packets between neighbors.

[2]The Java 2 protection domain associates permissions with code, rather than with the execution of the code. Our implementation, then, means executions of the same code by different principals result in multiple copies of the code. The Java Authentication and Authorization Service (JAAS) associates principals with a thread, but was made public too soon before the end of the development of this prototype to be incorporated.

We decided to use the digital signature form of end-end authentication and integrity protection. To support the ability to provide multiple principal attributes and identifiers to the packet for use in different domains as the packet transits the network, we added a credential option field to the packet that contains a list of credentials references. Each reference is the globally unique X.509 identifier (issuer distinguished name and serial number) as well as a hint of the CERT location in the form of a fully qualified domain name. We also added an option to contain the digital signatures associated with the credentials referenced. The digital signatures are independent, that is, one signature does not cover the other signatures. Our intent was that the credentials should be authorized independently as well, so that if one credential carried attributes that authorized a service, then the request for the service would succeed.

We modified the packet format to separate the payload into a static area (covered by the authentication protection) and a variable area. The classic ANTS packet contains a header and the data payload. Some fields of the header are modified in each node, such as a network resource bound that is decremented by each node (much like a TTL). Some fields remain static throughout the packet's travels, such as the MD5 hash identifier of the active code. The static area of our packet includes the static portions of the EE header, in particular the code identifier, and the static portions of the data payload. The variable area of our packet includes the variable fields of the EE header and the variable portions of the data payload. Note that there is no provision for the active code to itself be modified in transit. Given that the variable area of our packet is integrity protected only by the hop-hop integrity protection, we felt that the user-supplied active code could provide additional integrity assurance through customized integrity checks of the variable data. We therefore prohibit variation in the active code itself, so that the user's checks could be ensured to remain intact.

The resultant format is:

ANEP header	
Credential Field	(list of credentials)
Static Payload	(EE header and Data)
Origination Signatures	(each signature covers two previous fields)
Varying Payload	(EE header and Data)
Original ANEP Options	(i.e., Source Identifier Destination Identifier, Integrity Checksum, or N/N Authentication)
Hop Integrity	(covers everything)

On arrival, the credential references are extracted from the packet. The associated X.509v3 certificates are retrieved from DNSSEC and their signatures verified (both the DNSSEC signature of the CERT record and the certificate signature). This might require the recursive retrieval of certificates associated with the issuer of the certificate. When the certificate and the certificate chain have been checked, the packet signature can be verified. If all this succeeds, the packet is authenticated.

C. Authorization

In our prototype we added authorization enforcement checks in the node services, the EE services and in the access functions to the BulletinBoard shared state.

The authorization developed for the prototype is based on the Java 2 security architecture, with modification to support the special needs of the active network environment. The Java 2 security architecture performs authorization checks by combining stack inspection and a protection domain associated with every executing class. Each class is provided a protection domain by its class loader. The protection domain contains a permission object holding the class's *granted* permissions. Any protected object specifies the *required* permissions that must be held by a caller and then calls the Access Controller. The Access Controller checks the required permissions against the granted permissions stored in the protection domain of every class on the execution call stack. This stack inspection enforces an intersection of permissions, so that active code cannot circumvent protections by calling a class in a protection domain with greater permissions than it holds itself. We also benefit from the namespace controls and type safety in Java, in that the protection domain is non-forgeable and not modifiable by the loaded class.

When a SANTS ClassLoader is called upon to load active code, it binds the active code class to a SANTS ProtectionDomain object that contains exactly one SANTS permission object. This SANTS permission object is instantiated and initialized with a credential set belonging to the principals associated with the SANTS ClassLoader. In turn, whenever the SANTS permission object is called upon by the AccessController to render an access decision, it consults the SANTS policy engine, currently Keynote, to determine whether the requested resource access is implied by the Keynote access policy. If any of the principals are authorized according to the policy, the access succeeds. To summarize, any Java class loaded by a particular SANTS ClassLoader will be explicitly bound to that ClassLoader's associated principals via the ProtectionDomain, and it will have authorizations based upon the Keynote access policy.

In our implementation we explored the problem of composition of multiple policies. It is always the case that policies can be provided by different sources (e.g., national, company, and local policies). In active networks, the problem is exacerbated by the fact that the end user's policies carried in the active code travel through the network encountering different administrations with their own policies. In our implementation, the EE provides a BulletinBoard shared data service to incoming active code. The active code carries a policy stating the authorizations for access to its BulletinBoard items. But the EE may have its own policy governing access to the shared data. We enforced a mandatory access control so that the EE policy superseded any more lenient policy established by the active code, while permitting the active code to restrict

access in cases where the EE granted access.

We found it difficult to express the composition of the policies within Keynote. We found it necessary to call Keynote twice within the enforcement decision and then perform the composition (conjunction) of the two policy evaluations. We expect that more complex compositions of policy should be possible and should be represented in a meta-policy language. Further work in this area is needed.

IV. A Security Architecture

¿From this experience, we believe that an active network security architecture would include:

Naming Globally unique credentials that represent aggregate security attributes (e.g., X.509v3 certificates)

Packet Format A packet format that provides for

- a separation between portions of the packet that are covered by the authentication protection and those that are not
- a list of the appropriate credentials or references
- a list of authentication protections that bind those credentials to the packet
- hop-hop integrity protection between neighbors

Policy Language A ubiquitous policy language for the statement of authorization policy for node, EE or end user (e.g., Keynote).

Security Support System to include

- a cryptographic engine (e.g., JCE)
- a credential system (e.g., DNSSEC):
  global storage: a distributed secure system for the storage, retrieval and/or dissemination, and revocation of credentials
  local storage: an engine for storage, receipt and/or retrieval, validation and revocation of local copies of credentials
- a key management system (e.g., the Java Keystore):
  generation, retrieval, exchange, agreement, etc. of keys;
  storage, retrieval, format translation, removal, etc., of keys
- a policy management system:
  store, retrieve, revoke, etc. policy components

Enforcement Architecture to include:

- a non-forgeable security context (e.g., protection domain)
- a mechanism for binding a security context to an execution (e.g., class loader)
- an enforcement engine (e.g., AccessController)
- a policy evaluation engine (e.g., Keynote)

The components of the architecture that can be of common use should be placed in the NodeOS layer with access to the functions through the NodeOS API. This means that the entire security support system should be placed in the NodeOS layer. Communication to and from off-site locations (e.g., for the retrieval of keys or credentials) could be native to the NodeOS or could be through a Security-Management EE, allowing the security communication to take advantage of the powerful features of active networks. Note that each EE could have its own customized version of the security architecture, if needed for its own unique requirements.

The authentication handling should occur in the NodeOS so that access to cut-through channels can be authorized according to the correct domain and so that the EE is not clogged with un-authorized packets. But some EE's may choose to process all packets within their own domain resources, i.e., not create subdomains. All EE destined packets would be assigned to the EE domain but would retain their end user principal credentials and authentication protections. We suggest (but did not implement) that the creation of a new domain should include an access control policy controlling which principals should be allowed to access that domain, with a generous EE's policy permitting access to anyone. If the EE feels that it is under a clogging attack, it could restrict that policy. So there must be NodeOS API calls to modify a domain's access control policy. A special exception should be allowed to packets destined for a generous EE (one with an "anyone" access policy) so that authentication in that case can be skipped.

Some modifications are needed to the NodeOS API to support the security processing. The domain creation NodeOS call must include an authentication policy and an access control policy. There must be a NodeOS API call permitting a domain to modify its access control policy or its authentication policy. There must be access to all the NodeOS layer security support system from the EE. And, finally, a call is needed to permit a domain to activate or deactivate certain credentials in its security context. Work is ongoing in the Active Network community to modify the NodeOS API. Existing API's for cryptography, authorization, authentication, etc., may be adopted. It is not clear that existing API's for authentication, which are rooted in the existing Internet paradigm of end-end client to server communications, will be completely suitable for active networks.

The security processing in the NodeOS would follow the following sequence of actions:

- receive packet
- verify hop-hop integrity
- assign packet to existing domain
- extract credential list
- check credentials authenticity according to authentication policy for the domain
- check credentials against access control policy for domain
- deliver entire packet to the domain, including the credentials, authentication protection fields, etc.

The security processing in the EE would include (not necessarily in sequence):

- receive a packet including credentials
- create a subdomain, providing the security context parameters for the domain (e.g., the credentials to be associated with the domain) and the access control and authentication policies for the domain
- modify the access control policy of a domain, the authentication policy of a domain or the security context of a domain
- add or remove cryptographic protections to user data

V. Related Work And Conclusion

The Switchware project's approach to security [2], [3], [8] is two-fold. First, their language for active code, called PLAN, has restricted functionality to "safe" functions that are defined as being available to anyone . In particular, the language is guaranteed to terminate and there are no features for inter-packet communication. Hence, many PLAN packets have no need for authentication. PLAN does have the ability to call local services called "switchlets" and some of those switchlets are privileged. Packet are granted access to a thinned interface of the available switchlets. Those that need to call privileged services must authenticate the principal associated with the packet and the authenticated identity is bound non-forgeably to the packet thread. Then the privileged service is added to their service symbol table and the privileged service can authorize access based on the authenticated identity. They mention use of digital signatures or HMAC-SHA1 (based on a shared key negotiated between the principal and the node) for authentication. They do not discuss how they identify the principal's key for the digital signature in the packet or how they negotiate shared secrets for a packet that traverses multiple nodes. Neither do they address the problems of authenticating packets that are modified in transit. Mechanisms to remove services from the symbol table when access is no longer authorized are not described. In some languages such revocation is difficult, such as revoking access to an object in Java once an reference to the object has been supplied.

Seraphim [4], [5] uses active capabilities that compute the authorization decision. These active capabilities can be carried in the packet or retrieved from a policy server by the local policy engine or by a stub active capability. Active capabilities are executable code and are executed within an evaluation engine in the node. This provides a flexible mechanism for distributing policy and expressing policy with the full power of an executable language. While there are references to a "principal" associated with the packet and to authenticating that principal based on X.509 certificates and a PKI, there is no mention of how the packet signatures are generated or verified. There is also no mention of how the active capability is bound to the packet so as to preclude cut-and-paste attacks. As the paper [5] mentions specifically that the code is backwards compatible with the ANTS code, which does not include cryptographic authentication, the provision for cryptographic authentication (particularly of portions of the packet that ANTS changes in route) is not clear. Finally, composition of policies from different sources (e.g., active capabilities in the packet plus active capabilities from local policy servers) is not mentioned.

In SANTS we have provided a worked example of a flexible mechanism for providing strong security in an active network. SANTS provides strong end to end authentication and integrity protection and per service authorization of access. SANTS allows for the inclusion of authorization information in the active packet itself so that the packet can cross multiple administrative domains and still be properly controlled. SANTS provides enforcement of each individual node's authorization policy for access to its services and resources, providing the nodes with the assurance that their assets can be protected. Through global credential identifiers and a ubiquitous policy language, SANTS provides for end user control over authorization of access to its created state in the network. This assures the end user that its data and service can be protected according to its wishes. SANTS also provides that the node policy will take precedence over the active code policy, assuring the node that data can be protected according to its wishes even in the face of lenient policies from the visiting active code.

We believe that this effort, besides providing a worked example of a security solution, also has exposed important security issues regarding the protection of active networks.

Acknowledgments

The authors would like to acknowledge the valuable suggestions of the anonymous reviewers.

References

[1] David L. Tennenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden, "A survey of active network research," *IEEE Communications Magazine*, vol. 35, no. 1, pp. 80–86, January 1997.

[2] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith, "Security in active networks," in *Secure Internet Programming: Issues in Distributed and Mobile Object Systems*, Jan Vitek and Christian Damsgaard Jensen, Eds., vol. 1603 of *Lecture Notes in Computer Science State-of-the-Art*. Springer-Verlag Inc., New York, NY, USA, 1999.

[3] D. Scott Alexander, William A. Arbaugh, Angelos D. Keromytis, and Jonathan M. Smith, "Safety and security of programmable network infrastructures," *IEEE Communications Magazine, issue on Programmable Networks*, vol. 36, no. 10, pp. 84–92, October 1998.

[4] 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, Eds. Kluwer Academic Publishers, Boston, MA, September 2000.

[5] 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.

[6] J.M. Smith, K.L. Calvert, S.L. Murphy, H.K. Orman, and L.L. Peterson, "Activating networks: a progress report," *Computer*, vol. 32, no. 4, pp. 32–41, April 1999.

[7] AN Security Working Group, "Security architecture for active nets," July 1998, available online at ftp://ftp.tislabs.com/pub/activenets/ secrarch2.ps.

[8] Michael Hicks and Angelos D. Keromytis, "A Secure PLAN," in *Proceedings of the First International Working Conference on Active Networks (IWAN '99)*. July 1999, vol. 1653 of *Lecture Notes in Computer Science*, pp. 307–314, Springer-Verlag, available online at http://www.cis.upenn.edu/ switchware/papers/secureplan.ps.

[9] AN Node OS Working Group, "NodeOS interface specification," January 2000, available online at http://www.cs.princeton.edu/nsg/papers/nodeos.ps.

[10] K.L. Calvert, "Architectural framework for active networks, version 1.0," University of Kentucky, July 1999, available online at http://www.cc.gatech.edu/projects/canes/papers /arch-1-0.ps.gz.

[11] Larry Peterson, Yitzchak Gottlieb, Mike Hibler, Patrick Tullmann, Jay Lepreau, Stephen Schwab, Hrishikesh Dandekar, Andrew Purtell, and John Hartman, "An OS interface for active routers," *IEEE Journal on Selected Areas of Communications*, to appear 2001.

[12] D. Eastlake, "Domain name system security extensions," RFC 2535, IBM, March 1999.

[13] D.J. Wetherall, J.V. Guttag, and D.L. Tennenhouse, "ANTS: a toolkit for building and dynamically deploying network protocols," in *Open Architectures and Network Programming 1998*, San Francisco, CA, April 1998, pp. 117–129, IEEE.