A Secure Plan

Michael Hicks and Angelos D. Keromytis*

Distributed Systems Lab
CIS Department, University of Pennsylvania
200 S. 33rd Str., Philadelphia, PA 19104, USA
{mwh,angelos}@dsl.cis.upenn.edu

Abstract. Active Networks promise greater flexibility than current networks, but threaten safety and security by virtue of their programmability. In this paper, we describe the design and implementation of a security architecture for the active network *PLANet* [HMA+99]. Security is obtained with a two-level architecture that combines a functionally restricted packet language, PLAN [HKM+98], with an environment of general-purpose service routines governed by *trust management* [BFL96]. In particular, we employ a technique which expands or contracts a packet's service environment based on its level of privilege, termed *namespace-based security.* As an application of our security architecture, we outline the design and implementation of an active-network firewall. We find that the addition of the firewall imposes an approximately 34% latency overhead and as little as a 6.7% space overhead to incoming packets.

1 Introduction

Active Networks offer the ability to program the network on a per-router, per-user, or even per-packet basis. Unfortunately, this added programmability compromises the security of the system by allowing a wider range of potential attacks. Any feasible Active Network architecture therefore requires strong security guarantees. We would like these guarantees to come at the lowest possible price to the flexibility, performance, and usability of the system.

This paper presents the design and implementation of a security architecture for *PLANet* [HMA+99], an active internetwork based on PLAN, the Packet Language for Active Networks [HKM+98]. Our approach is to partition the problem into two levels: language-based security for PLAN programs, complemented by *namespace-based security* for more general router services, governed by trust management. We briefly discuss PLAN and its role in this architecture, but focus more attention on service security. We present both architecture and implementation, and conclude with some applications of our approach, including a simple firewall that 'filters' active packets. [HK99], an extended version of this paper, contains more detailed motivation and performance analysis.

* This work was supported by DARPA under Contract #N66001-96-C-852, with additional support from the Intel Corporation.

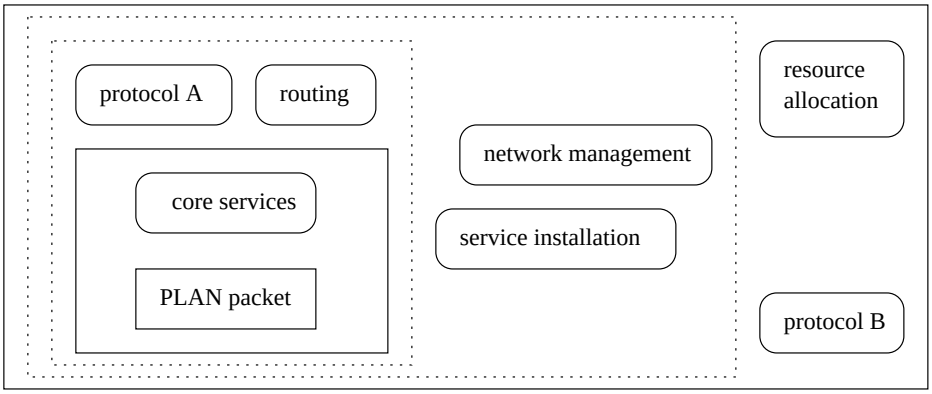

Fig. 1. PLANet's security architecture.

2 Architecture

Our security architecture is illustrated in Figure 1. The solid boxes define the two levels of the architecture: the contents of the central box define the *PLAN level* which is usable without need of credentials, while the remaining area forms the *service level*. This architecture falls along functional boundaries: *all* PLAN programs, by their nature, are safe (as defined below) and so may run unauthenticated, while, in general, service routines are unsafe, and must be partitioned by level of trust, visualized by the dotted boxes. We augment the PLAN level with a fixed set of 'core services' which are known to be functionally safe.

This architecture is designed to guard against the standard threats to computational resources and their contents [AAKS99]. In particular, we defend against attacks that would deny service, seek to obtain unauthorized content, and misrepresent (spoof) identity. We explain PLAN's role in defending against these attacks below.

2.1 PLAN

PLAN [HKM+98] is a small functional language with syntax similar to ML [Ler, MTH90]. To express remote computation, it includes a primitive `OnRemote` (among others) that evaluates an expression at a remote node. Invoking `OnRemote` will result in a newly spawned packet.

By design, the language has properties that prevent some attacks. PLAN is resource- and expression-limited, thus preventing CPU and memory denial-of-service attacks. For example, all PLAN programs are guaranteed to terminate[1], since PLAN does not provide a means to express non-fixed-length iteration or recursion. Additionally, PLAN programs are isolated from one another since there is no means of direct communication among them, and because the language's

[1] PLAN programs terminate as long as the services called also terminate.

strong typing and garbage collection prevent indirect means, such as through pointer swizzling or buffer overflows. Finally, a network resource bound counter, similar to IP's "Time to Live" (TTL) field, is used to bound network resources.

3 Service Security via Trust Management

Because of their general-purpose nature, service routines may perform actions which, if exploited, could be used to mount an attack. A radical approach to this problem would be to prevent *any* service routine from being installed that could potentially harm the node. However, this would preclude the addition of service routines—for example, network management operations—that should be available to *trusted* users. We thus employ security mechanisms which allow authorized programs to access potentially unsafe service routines.

3.1 Trust Management

In determining the form of these security mechanisms, we arrived at some basic requirements. First, the mechanisms should be simple to understand and employ. Second, security policies should be modifiable as needed, while the system is operating. Furthermore, policy mechanisms should be flexible enough to anticipate future application needs. Finally, security mechanisms must scale to support increasing numbers of principals and their trust relations. To meet these requirements, our service security relies on *trust management* [BFL96, BFIK99].

Trust management assigns some level of privilege (or trust) to a user, or *principal*, of the system. In particular, if a running PLAN program wishes to invoke a privileged service routine or alter a service parameter, the principal associated with the packet must be authenticated, and then the operation must be authorized. If either step fails, the operation is denied. We consider the question of policy and mechanism for authorization below; details about our particular implementation of authentication and authorization are presented in the next section.

3.2 Policy and Mechanism

Before applying trust management, we must consider what sorts of policies we would like to express, and what particular mechanisms we shall use to enforce these policies. For our system, we want our policies to express what services, above the core services, are available to certain users. We also find it convenient to indicate which services should be unavailable for a particular user; this will be motivated in Section 5. For purposes of simplicity and scalability, we choose to map *sets* of principals to *sets* of services. We also need to manage delegation policies with regard to these mappings. For example, we might specify that the services in set s may be accessed not only by principal p, but also by those principals authorized by p. In keeping with our requirements, this policy should scale to include many nodes, principals, services; and be alterable on-the-fly.

Furthermore, we want to specify not only *whether* a service routine may be invoked, but *how* it may be used. For example, a *resident state* service which allows packets to leave state on the routers might apportion different amounts of space to different users. We should also be able to specify general resource usage parameters, such as CPU and memory use.

To enforce security policy we require strong principal authentication, and use a *policy manager* on every node; more details are given in the next section. In our system, packets must authenticate themselves at some point before accessing privileged services; at this time, the appropriate services are added to (or subtracted from) the packet's current service symbol table. We call this approach *namespace-based security*. Since PLAN is strongly typed and looks up services on an as-needed basis, programs are incapable of invoking code outside of this updated table.

Additionally, we allow those services which may require policy-based parameterization to query the policy manager as necessary during their execution. For example, the resident state service mentioned above would query the local policy to determine how much memory the current principal was allowed to occupy.

We feel there are some compelling advantages to this approach. First, namespace-based policies are simple to formulate and easy to change. Second, because namespace-based security is centrally-administered, individual service routines may be written without concern for security, and policies may change dynamically without worry of inconsistency. Furthermore, unauthenticated programs may access the core services without additional performance penalty. Finally, because namespace-based security is not by itself sufficient, we allow services to formulate their own usage policies.

There is still some work to be done in our current system. Namespace-based security only applies to PLAN service routine calls, not calls between service routines. This is slightly more difficult, but entirely possible, since Caml, our service implementation language, provides a mechanism which may be used to implement namespace-based security: *module thinning*. The use of module thinning has been explored for active networks in [Ale98] and for mobile agent systems in [LOW98]. Also, while we have experimented with mechanisms for enforcing resource usage, we have yet to arrive at ones that are sufficiently lightweight. Relevant details may be found in [Hic98].

4 Implementation

4.1 Authentication

Before a PLAN program may invoke a trusted service, its associated principal must be determined; this is the process of authentication. Authentication is typically done in a public-key setting by verifying a digital signature in the context of some communication (*e.g.,* a packet). In PLAN, one obvious link between communication and authentication is the *chunk*.

A chunk (or **c**ode **hunk**) may be thought of as a function that is waiting to be applied. In PLAN, chunks are first-class—they may be manipulated as

data—and consist internally of some PLAN code, a function name, and a list of values to be used as arguments during the application. A chunk is typically used as an argument to `OnRemote` to specify some code to evaluate remotely. A chunk may also be evaluated locally by passing it to the `eval` service, which resolves the function name with the current environment, performs the application, and returns the result.

We have added a service called `authEval` which takes as arguments a chunk, a digital signature, and a public key. `authEval` verifies the signature against the binary representation of the chunk, and if successful, the chunk is evaluated.

There are two key advantages to this approach. One is that a principal signs exactly the piece of code he wants to execute, and may only have extra privilege while executing that piece of code. Second, only those programs which require authorization will have the extra time and space overheads. However, there is no protection against replay attacks, and public key operations are notoriously slow. Furthermore, authentication is only unidirectional (principal to node), thus providing less confidence to the caller. We mitigate these problems by using a variant of the mutual authentication protocol described in [AAKS98].

4.2 Authorization

As our policy manager, we have chosen to use the Query Certificate Manager (QCM) [GJ98], which provides comprehensive security credential location and retrieval services, employing a distributed ACL. While in this paper we are making use of QCM, our architecture is designed so that other policy managers be used instead. In particular, we are also experimenting with the KeyNote [BFIK99] trust-management system.

QCM is used to specify the services to be added to or subtracted from the default service-environment by associating certain *thicken* and *thin* sets of services with a principal or set of principals. Once a principal has been authenticated, these sets are used to modify the default environment. The resulting service environment is then used during subsequent chunk evaluation. As an optimization, we can cache this environment for future reference, thus avoiding repeated invocations of QCM and reconstructions of the environment.

A key advantage of using QCM is that it can be used for more than just specifying sets of principals on a per-node basis. In particular, sets described in a distributed manner impose no additional query complexity. For example, a node A may define a set which partially resides at another node B:

`l =` { p_1, p_2, ... , p_n } `union` $B\$m$

If the authorization service on A makes a membership test on set `l`, QCM will automatically query B if necessary. QCM may also make use of certificates, which are signed assertions about set relationships, to short-circuit remote queries. These may be passed as additional arguments to `authEval`, or may be obtained during node-node authentication. This allows QCM to implement both *push*- and *pull*-based information-retrieval.

5 A Simple Active Firewall

As an application of our architecture, we implemented a simple active firewall. Typically, firewalls filter certain types of packets, such as all TCP connection requests on certain port numbers. Usually such packets are easily identified by their protocol headers. However, in PLANet, and indeed in any active-packet system, there is no quick way to assess a packet's functionality.

Our approach is that rather than filter packets at the firewall, we associate with them a *thinned* service environment in which any potentially harmful services are removed. The packets may then be evaluated inside the trusted network using only those services. While this may seem to contradict our premise stated in Section 2 that the default environment should consist only of 'safe' services, in the context of a trusted Intranet, we would expect that the default privilege allowed to local packets exceeds that of foreign packets. Furthermore, we would not want to impose the overhead of authentication and authorization on local packets in the general case.

To thin the environment of foreign packets, our firewall associates them with a *guest* identity that has the appropriate policy. To do this, the firewall encapsulates each packet with a small wrapper which calls `authEval` with the original chunk, using the guest identity. In general, this would require the firewall to sign all incoming packets. However, because the guest environment will provide less privilege than the default environment, we should be able to conceivably avoid the cryptographic cost: any authenticating principal whose environment is thinned and not thickened can be 'taken at his word.'

In the base PLANet implementation, a two-hop ping takes 2.13 ms for a minimally-sized packet (80 bytes) and 3.06 ms for a maximally-sized one (1500 bytes). Changing the middle node to the 'signing firewall' adds 37% and 32% to the round-trip times, respectively, raising them to 2.91 and 4.03 ms. Between 1/3 and 1/2 of this overhead is attributable to signing and verification, depending on the packet size. For the firewall, the remaining overhead is due to encapsulation costs (which requires extra marshalling and copying), while for the end-host it is due to decapsulation and additional interpretation costs. Parallelism and special-purpose hardware can further reduce cryptographic costs and improve latency and throughput. If we eliminate the cryptographic operations, we reduce the end-to-end ping times to 2.55 and 3.41 ms for minimal and maximal payload, respectively. This reduces the firewall-induced overhead to 20% and 11%. A smarter PLAN interpreter would also considerably improve overall performance.

The firewall also imposes a fixed 101-byte space overhead due to the extra code and signature that is attached to the incoming packets. This translates to 126% and 6.8% space overhead for the minimal and maximal payload packets respectively. One way of mitigating this overhead is for PLAN to support code caching and language-level *remote-references*. Since all PLAN values are *immutable*, the contents of a remote reference may be safely cached without the need for a coherence protocol.

6 Related Work

Research in the area of security for active networks is in its early stages. The SANE [AAKS98] architecture is part of the SwitchWare Project [AAH+98] at the University of Pennsylvania. SANE is currently used in conjunction with the ALIEN architecture [Ale98]. Security is achieved in ALIEN through a combination of module thinning and type safety. Similar approaches have been taken in [LR99, BSP+95, vE99]. Other language-based protection schemes can be found in [BSP+95, CLFL94, HCC98, LOW98, Moo98]. The main difference between this work and SANE lies in that we can depend on a provably safe language (PLAN) for those packets that do not require special privileges. Furthermore, programming constructs available in PLAN (*e.g.,* chunks), considerably ease the task of implementing security abstractions.

A working group within the Active Networks project has been defining a common security meta-architecture [Mur98]. However, this architecture has not become concrete enough for implementation.

Secure PLAN is currently being extended to support validation and verification [NL96, Nec97] for active extensions.

We have demonstrated that our architecture addresses possible threats while still preserving the flexibility and usability of the system. This architecture is based on language safety, authentication, and trust management. We discussed the practicality and acceptable performance of our approach experimentally, in the context of an active firewall.

References

[AAH+98] D. S. Alexander, W. A. Arbaugh, M. Hicks, P. Kakkar, A. D. Keromytis, J. T. Moore, C. A. Gunter, S. M. Nettles, and J. M. Smith. The SwitchWare Active Network Architecture. *IEEE Network Magazine, special issue on Active and Programmable Networks*, 12(3):29–36, 1998.

[AAKS98] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. A Secure Active Network Environment Architecture: Realization in SwitchWare. *IEEE Network Magazine, special issue on Active and Programmable Networks*, 12(3):37–45, 1998.

[AAKS99] D. S. Alexander, W. A. Arbaugh, A. D. Keromytis, and J. M. Smith. Security in Active Networks. In *Secure Internet Programming* [VJ99].

[Ale98] D. S. Alexander. *ALIEN: A Generalized Computing Model of Active Networks.* PhD thesis, University of Pennsylvania, September 1998.

[BFIK99] M. Blaze, J. Feigenbaum, J. Ioannidis, and A. Keromytis. The Role of Trust Management in Distributed Systems Security. In *Secure Internet Programming* [VJ99].

[BFL96] M. Blaze, J. Feigenbaum, and J. Lacy. Decentralized Trust Management. In *Proceedings of the 17th Symposium on Security and Privacy*, pages 164–173. IEEE Computer Society Press, Los Alamitos, 1996.

[BSP+95] B. Bershad, S. Savage, P. Pardyak, E. G. Sirer, M. Fiuczynski, D. Becker, S. Eggers, and C. Chambers. Extensibility, Safety and Performance in the SPIN Operating System. In *Proceedings of 15th Symposium on Operating Systems Principles*, pages 267–284, December 1995.

[CLFL94] J. S. Chase, H. M. Levy, M. J. Feeley, and E. D. Lazowska. Sharing and Protection in a Single-Address-Space Operating System. In *ACM Transactions on Computer systems*, November 1994.

[GJ98] Carl A. Gunter and Trevor Jim. Policy-Directed Certificate Retrieval. `http://www.cis.upenn.edu/~qcm`, 1998.

[HCC98] C. Hawblitzel, C. Chang, and G. Czajkowski. Implementing Multiple Protection Domains in Java. In *Proceedings of the 1998 USENIX Annual Technical Conference*, pages 259–270, June 1998.

[Hic98] Michael Hicks. PLAN System Security. Technical Report MS-CIS-98-25, Department of Computer and Information Science, University of Pennsylvania, April 1998.

[HK99] Michael Hicks and Angelos D. Keromytis. A Secure PLAN. Technical Report MS-CIS-99-14, Department of Computer and Information Science, University of Pennsylvania, May 1999.

[HKM+98] Michael Hicks, Pankaj Kakkar, Jonathan T. Moore, Carl A. Gunter, and Scott Nettles. PLAN: A Packet Language for Active Networks. In *Proceedings of the Third ACM SIGPLAN International Conference on Functional Programming Languages*, pages 86–93. ACM, 1998.

[HMA+99] Michael Hicks, Jonathan T. Moore, D. Scott Alexander, Carl A. Gunter, and Scott Nettles. PLANet: An Active Internetwork. In *Proceedings of the Eighteenth IEEE Computer and Communication Society INFOCOM Conference*, pages 1124–1133. IEEE, 1999.

[Ler] Xavier Leroy. The Caml Special Light System (Release 1.10). `http://pauillac.inria.fr/ocaml`.

[LOW98] J. Y. Levy, J. K. Ousterhout, and B. B. Welch. The Safe-Tcl Security Model. In *Proceedings of the 1998 USENIX Annual Technical Conference*, pages 271–282, June 1998.

[LR99] X. Leroy and F. Rouaix. Security properties of typed applets. In *Secure Internet Programming* [VJ99].

[Moo98] J. Moore. Mobile Code Security Techniques. Technical Report MS-CIS-98-28, University of Pennsylvania, May 1998.

[MTH90] Robin Milner, Mads Tofte, and Robert Harper. *The Definition of Standard ML*. The MIT Press, 1990.

[Mur98] Security Architecture for Active Nets, June 1998. Draft available at `http://www.ittc.ukans.edu/~ansecure/0079.html`.

[Nec97] George C. Necula. Proof-Carrying Code. In *Proceedings of the 24th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 106–119. ACM Press, New York, January 1997.

[NL96] George C. Necula and Peter Lee. Safe Kernel Extensions Without Run-Time Checking. In *Second Symposium on Operating System Design and Implementation*, pages 229–243. Usenix, Seattle, 1996.

[vE99] T. von Eicken. J-Kernel a capability based operating system for Java. In *Secure Internet Programming* [VJ99].

[VJ99] Jan Vitek and Christian Jensen. *Secure Internet Programming: Security Issues for Mobile and Distributed Objects*. Lecture Notes in Computer Science. Springer-Verlag Inc., New York, NY, USA, 1999.