Action Amplification: A New Approach To Scalable Administration

Kostas G. Anagnostakis
Internet Security Lab, Institute for Infocomm Research
21 Heng Mui Keng Terrace, Singapore
kostas@i2r.a-star.edu.sg

Angelos D. Keromytis
CS Department, Columbia University
1214 Amsterdam Ave., New York, NY 10027, USA
angelos@cs.columbia.edu

Abstract— **We present a systems-management approach that enables administrators to effectively handle the challenge of increasing numbers of hosts, routers, users, and services in the networks to manage. Our approach is to map the actions of an administrator on a single host (such as creating a new user account) to the network at large, while maintaining the exact same interface. Our system *amplifies* the administrator's actions appropriately throughout the network, and confirms the correct propagation of all configuration changes throughout the distributed system. We argue that this approach allows administrators to easily manage several aspects of a large domain, because it provides a familiar and intuitive interface. Such a system can be used as a front-end to any other automation system used to manage large domains. To determine the feasibility of our approach, we implemented it on the OpenBSD system. We discuss the prototype implementation, along with the limitations to our approach that it exposes.**

I. Introduction

Effective systems management is a problem faced by all organizations. The difficulty of managing such networks increases with their scale and complexity. Since the capacity of administrators remains constant over time, other approaches must be used to lighten this burden. These include using more administrators and using increasingly more sophisticated workflow and service provisioning tools that automate repetitive actions and minimize the time needed to complete any task [12].

Unfortunately, there are some limitations inherent to these approaches. Although automated tools can help considerably, they often suffer from non-intuitive interfaces, which negate much of the benefit derived from the automation (since the administrator has to spend some time determining how to achieve the task at hand). This applies to systems that are based on graphical user interfaces (GUIs) as well as language-driven ones. Increasing the number of administrators does not scale well, and introduces the potential for conflicting configurations.

We propose a new approach to managing large-scale systems. We specifically focus on system administration and network configuration, and do not consider other management functions (such as monitoring) in this paper. Our starting point is the observation that administrators can handle small systems fairly easily[1]. Our approach is to use the same interface administrators use to manage such a small system, *amplifying* all relevant actions such that they take effect throughout the network using autonomic computing techniques to monitor and validate the correctness of the resulting configuration. Thus, we can map the creation of a new user in a unix system to a series of actions that, for example, create the new user in the corporate LDAP repository, issue the appropriate X.509 certificates, create VPN configuration entries on the firewall such that the user can tele-commute, grant access to the corporate document database, *etc.* From the administrator's point of view, the only action necessary was to add the appropriate line in the */etc/passwd* file.

[1]Whether this is because of more intuitive interfaces, better training, or some other reason, is not important for the purpose of our discussion.

To determine the feasibility of our approach, we wrote a prototype implementation using the OpenBSD system. We used *systrace* [15], a mechanism originally devised for sandboxing applications through system-call monitoring, to capture all "interesting" actions undertaken by the administrator (such as editing */etc/passwd* or configuring network interfaces). Our back-end processing then analyzes these actions and issues the appropriate directives. Using a graphical user interface, the back-end can notify the administrator of inconsistencies or provide other important information. Our conclusion is that such a system can easily capture many of the tasks administrators are responsible for in a large network.

Our prototype also exposes some limitations in our approach. Perhaps the most important is the fact that certain tasks (*e.g.*, deploying and running a new service on a large number of servers) do not map well to the file-based interface used in unix-like system administration. Furthermore, tasks that require intermediate state can complicate the processing logic in our system. Despite these limitations, we believe our system offers administrators a natural approach to large-network management.

A. Paper Organization

The remainder of this paper is organized as follows. Section II presents our approach. Section III describes our implementation using OpenBSD and `systrace`. Section IV gives a brief overview of related work, and we conclude the paper with Section V.

II. Our Approach

As we discussed in the previous section, our goal is to enable administrators to easily manage arbitrarily large installations by presenting them with a familiar and intuitive user interface. The system can then interpret and amplify the administrator's actions. Arguably, the best interface that fits these criteria is the one used to manage a single host: user addition/deletion, network interface and packet filtering/firewall

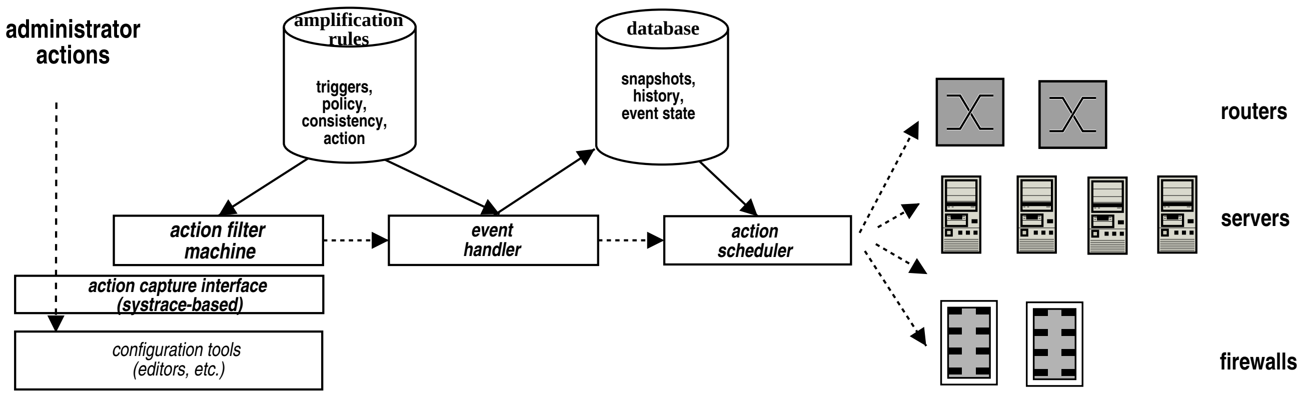
administrator actions
amplification rules
triggers, policy, consistency, action
database
snapshots, history, event state
routers
servers
firewalls
action filter machine
event handler
action scheduler
action capture interface (systrace-based)
configuration tools (editors, etc.)

Fig. 1. Overview of the Action Amplification Architecture.

configuration, process (in our case, network daemons) management, user quotas, *etc.* Administrators have been using such management interfaces, either directly or through shell scripts and other tools, for many years; our empirical observations (and our own experience in system administration) indicate that single-host management tasks are one of the few pieces of knowledge shared by all administrators.

Our architecture then, as shown in Figure 1, captures single-host management actions and amplifies them to network-wide operations. Several components are necessary to realize such a system. First, we need a suitably configured host (either as a stand-alone system or running inside an emulator, such as VMWare) that is explicitly used as the "amplifying engine". On this host, we need an *action capture interface* that monitors all interesting actions undertaken by the administrator. The actions are filtered according to a set of pre-defined rules, which weed out uninteresting actions (such as opening a file that has no significance from a management viewpoint, or simply reading `/etc/passwd`). Those actions that are deemed interesting are conveyed to the *event handler,* which consults the network-wide configuration and issues a number of configuration directives for the relevant hosts and services. At this point, various consistency checks are performed on the request and, if any problems are detected, the administrator is notified through a management interface. Otherwise, the directives are passed on to the *action scheduler,* which is responsible for distributing and applying them to the affected network components and services, and for reporting any problems to the administrator. We assume that the systems and services that must be configured are either known *a priori* (*e.g.,* recorded in a database) or can be discovered dynamically; the details are not pertinent to our discussion, so we will assume the former case.

We call the action filters, the system policy, and the consistency checks, all of which are organization-specific, the *amplification rules*. These rules are kept on a database that resides on the amplifying engine. This database is modified as new nodes and services are added, new user roles are created, and the organization's policy changes. Effectively, the amplification rules capture the organization's management policy for operations such as adding new users, removing existing users, *etc.* Multiple such databases (and rules) may exist inside an organization, if the management task is somehow partitioned (*e.g.,* with different administrative entities responsible for different branches of the network).

Note that we have not posed any requirements on the actual single-host management interface, other than the fact that it be auditable (*i.e.,* we can monitor all administration actions). Thus, it should be possible to create different administration front ends, *e.g.,* a Windows-based and a unix-based one, allowing us to better suit the system to an administrator's job experience.

In the remainder of this paper, we assume a unix-like management interface. That is, most of the administration effort involves editing various files, such as `/etc/passwd, /etc/groups, /etc/inetd.conf` and so on. This allows us to model administrator actions as **differences** between two versions of a file: comparing the contents of the file before and after an administrator has edited it reveals the desired action. For example, if a new line has been added to the */etc/passwd* file, the system can determine that a new user has been added to the system, and (through the event handler) issue the appropriate network-wide configuration directives (*e.g.,* creating the user in the LDAP directory, creating an RSA public/private key and X.509 certificate for use with the corporate web site, installing appropriate policy entries in the VPN firewall so that the user can tele-commute, *etc.*). Similarly, if an existing line has changed, the appropriate action for that user can be taken (*e.g.,* if the group ID has changed, then the appropriate ACL entries in the web server will be updated). The fact that most such files in unix are simply text-based makes the implementation easier, since we can use tools like `diff` and `awk` to implement much of the back-end of our system, as we shall see in the next section.

Note that certain administrator actions cannot be derived from file-version differences, *e.g.,* configuring a network interface or installing some packet filtering rules. Thus, we need to be able to capture more than file-based operations. Furthermore, we do not wish to directly modify tools such as `adduser,` since not all administrators use them to add a user; instead, we want a more general mechanism that can capture all potentially interesting actions and act on the subset of those identified by the action filters.

Furthermore, our description of the architecture so far implies an *information push* model: actions are undertaken as a result of administrator-issued events, and information (in the form of configuration entries) is pushed to the relevant hosts and services. While this can capture much of the administration task, we can also use the system in an *information pull* method. For example, the system dynamically instantiates the contents of a file (such as `/etc/passwd`), by retrieving the necessary information from all the relevant databases (*e.g.*, a network-wide LDAP directory) on demand. In that case, the management files can be considered as simple *views* of the relevant information, which itself is distributed among any number of systems and databases. Although we note the potential of our architecture to operate in this way, we will focus on the information-push aspects in this paper.

In terms of synchronization, our system must make sure that conflicting simultaneous actions undertaken by different administrators do not occur, or can be resolved. Fortunately, most single-host management interfaces already utilize some form of locking to prevent simultaneous access, *e.g.*, to `/etc/passwd`. For the remainder, we simply have to ensure serialization, *i.e.*, if two administrators configure a network interface in different ways, the configuration issued second should take effect. Fortunately, this is fairly straightforward when done in the context of a single operating system.

Actions that span multiple files (*e.g.*, creating a new group, then creating a new user belonging to that group) can be viewed as independent actions that have a particular temporal ordering. Thus, we do not need to consider system-wide action serialization and ordering. Should the situation arise in the future, we can treat such actions as part of a larger database transaction and perform rollback as needed. We do not investigate this option further in this paper; when inconsistencies arise (as in the case where a new user is created as belonging to a non-existing group), we notify the administrator through a management interface and abort the latest operation (*i.e.*, overwrite the file with the previous version).

In the next section, we describe our prototype implementation of the architecture.

III. Implementation

Our OpenBSD-based prototype implements the components presented in Figure 1. In this section, we present these components in more detail.

A. Action Capture Interface

We use `systrace` [15], as shown in Figure 2, for intercepting relevant administrator actions. `systrace` is an operating system facility for intercepting and processing *system calls*, *e.g.*, calls by applications to operating system facilities such as file, network or memory management operations. This facility has many uses in performing policy compliance checks for applications, detecting intrusion attempts and logging

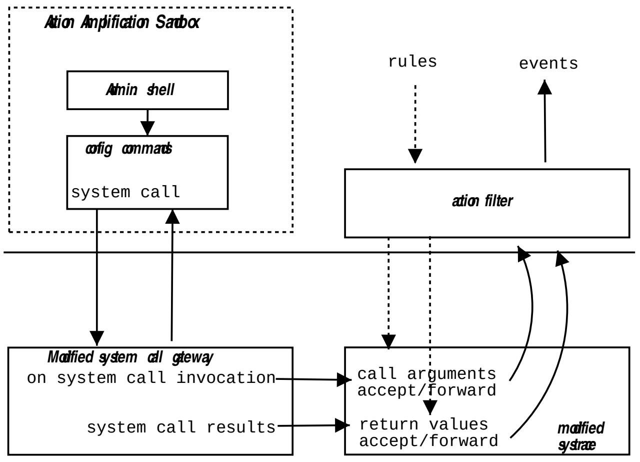

Fig. 2. Adaptation of *systrace* for the Action Capture Interface.

system events. The operating system kernel is instrumented to intercept system calls and trigger the kernel-level systrace facility for determining whether the call should be permitted, denied or if further processing is necessary. The kernel-level decisions are based on simple rules that can be evaluated fast enough *e.g.*, for system calls that should be always accepted or always denied without costly analysis of their arguments. If further processing is necessary the kernel part forwards the relevant information to a user-space daemon. This distinction between *fast-path* and *slow-path* processing ensures that `systrace` operation does not add significant processing overheads to the system. Communication between kernel-level and user-level parts of `systrace` is performed through the `/dev/systrace` pseudo-device. When the kernel defers system call processing to the user-level daemon, it suspends the process until it gets a response.

Since `systrace` was designed with the goal of performing policy checks, examining system calls upon entry is usually sufficient. For example, policies relating to file system operations can usually be checked when a file is opened by examining the system call arguments. Any subsequent attempt to read or write are assumed to be permitted since the `open` call was authorized. The case of action amplification is slightly more complicated because it is necessary to examine *sequences* of related operations. For example, tracking modifications to a configuration file requires matching a call to `open` indicating the beginning of a configuration action with a call to `close`, indicating that the action is complete. The name of the file is passed as an argument to `open` and can therefore be easily intercepted and checked against a list of files associated with configuration actions. However, subsequent operations on the file are performed through a file-descriptor instead of the filename and can therefore not be intercepted unless the file-descriptor is known to the action filter. Since this information is only available when the system call operation is complete, we have modified `syscall` to also intercept the *results* of system calls. The filtering process is the same as for system call entry: the system first consults a small in-kernel ruleset to

```
rule pwd_seq =
{
 native-open:
  CONDITION { filename=="/etc/passwd" &&
   (mode & (O_CREAT|O_RDWR|O_WRONLY)) }
  ACTION { state_add(pwd_seq,fd,pid); },
 *:*,
 native-close:
  CONDITION { state_lookup(pwd_seq,fd)==pid }
  ACTION { passwd_action(); }
}
```

Fig. 3. Example action filter rule for user account management.

determine if the event may be of interest, and either ignores it or notifies the user-level daemon.

For the purpose of action amplification, we have implemented our own user-space daemon that analyzes system call sequences to intercept administration actions that need to be amplified. The daemon is configured using a set of *CONDITION/ACTION* rules as shown in the example of Figure 3. Note that any element in the intercepted system call sequence can be associated with an action. If the *CONDITION* part becomes true, then the daemon notifies the event handler to process the *ACTION* part.

B. Event handler

The event handler receives notifications from the action capture interface and is responsible for deciding how to amplify the intercepted actions. For convenience, our current system uses `awk` scripts for action processing, combining a familiar scripting language with the ability to trigger external scripts or system commands if necessary – this is important as it allows the event handler to interact with a configuration database for bookkeeping, consistency checks, *etc*. Although the amplification actions could be performed by the event handler, the need for action serialization and aggregation requires the introduction of a separate “action scheduler” for orchestrating the actual execution of the amplification actions. The event handler’s task is therefore restricted to submitting job requests to the scheduler.

C. Action scheduler

The action scheduler receives action requests from event handlers and manages their execution. There are three main tasks for the action scheduler. First, it resolves potential conflicts between different actions, such as two administrators resetting a user’s password at approximately the same time. Second, it aggregates many actions for the same resource into batch requests, and regulates the rate of job execution to avoid overloading the network or the target resource. Third, it monitors the progress of jobs and reports to the administrator (e.g., through a console pop-up window) on possible failures.

D. Portability

Although we have implemented the action amplification system on OpenBSD, porting to other operating systems should be straightforward. `systrace` is also available for the Linux, Mac OS X and NetBSD operating systems and the modifications needed for tracing the results of system calls are platform-independent and can therefore easily be incorporated in a kernel patch.

```
adminboxes:*:3000:root
bigcluster:*:3001:bob,alice
roadwarriors:*:3002:uday,qusay
wifitest:*:3003:root
workstations:*:3004:user000,user001,user002
```

Fig. 5. Example `/etc/group` configuration.

E. Examples

a) User account management: We can track modifications to the UNIX password file by intercepting a sequence of system calls that starts with an `open` system call to write `/etc/passwd` and ends with a call to `close`. The `open` call is associated with an internal action of the system that takes a snapshot of the password file before the modifications are committed. The `close` call is the last call in the sequence and triggers the rest of the amplification procedure. An example filter rule for intercepting modifications to the password file is shown in Figure 3. The system compares the snapshot of the file before the change with the current image of the file using `diff` and determines what type of action was performed *e.g.*, account creation or deletion, password change, *etc*. An example `diff` is shown in Figure 4. Depending on the type of action, the system then determines how to amplify it by consulting the amplification rules.

When a new user is added to the system, we create accounts on the Windows Domain Server, the NIS server, and the RADIUS server. We create an RSA public/private key pair, issue an X.509 certificate, and push that information on the organization’s LDAP directory. Depending on the group(s) the user belongs to (or is added to), access control lists on the web server, the file server, the corporate database, and the print server are modified, and appropriate quotas (for disk space, CPU time, and pages printed) are enforced. Finally, the firewall configuration is changed to allow remote VPN connections from this user (identified by the X.509 certificate that was just created).

Likewise, when a user is deleted, the appropriate actions are undertaken (*e.g.*, the X.509 certificate is added to the Certificate Revocation List posted on the LDAP directory, and any running user processes are terminated on all systems).

Group membership (e.g., as specified in the `/etc/group` file) is used for more complicated configurations where access to certain systems is restricted, and cases where a certain group of users is allowed remote access through the firewall, as shown in Figure 5.

b) Network interface management: Our amplifying engine contains a number of virtual network interfaces (along with one or more real ones, that permit it to connect to the various servers) that represent the various firewalls, important routers, and connections to the public network. Configuring each interface “up” or “down” (via the `ifconfig` command)

```
--- passwd.orig Tue Aug 12 21:03:28 2003
+++ passwd Tue Aug 12 21:03:58 2003
@@ -18,3 +18,4 @@
 proxy:*:71:71:Proxy Services:/nonexistent:/sbin/nologin
 nobody:*:32767:32767:Unprivileged user:/nonexistent:/sbin/nologin
 angelos:*:7709:7709:Angelos D. Keromytis:/home/angelos:/usr/local/bin/bash
+anagnost:*:7805:7805:Kostas Anagnostakis:/home/anagnost:/usr/local/bin/bash
```

Fig. 4. Difference between original and modified `/etc/file` indicating the addition of a user.

enables or disables the relevant link[2].

More interestingly, installing packet filtering or Network Address Translation (NAT) rules on an interface will instantiate them on the relevant router(s) or firewall(s). Note that a pseudo-interface may correspond to more than one such systems. For example, the virtual interface `fr-all0` may correspond to all the firewalls in the organization, whereas `rt-cisco1` may represent all Cisco routers; the interface names can be named such that it is easy to determine their scope. The different sets of nodes represented by the various virtual interfaces need not be disjoint, *i.e.,* the same router may be affected by an action on `rt-cisco1` and on `fr-all0`.

c) VPN configuration: On most unix systems, most VPN configuration (using IPsec [8] and IKE [7]) is done through a combination of file editing (*e.g.,* `/etc/isakmpd/isakmpd.conf` on OpenBSD) and command-line directives (on OpenBSD, via the `ipsecadm` command). We can capture these actions and translate them to the appropriate configuration directives for the organization's VPN gateways. Security-configuration management for a web server, such as Apache, can be implemented in a similar manner.

d) Network services management: Administrators can select which services to run by editing files like `/etc/services`, `/etc/protocols`, and `/etc/inetd.conf`. One potential problem here is the fact that it is impossible to determine which subset of hosts should be affected by any particular change in some of these files, *e.g.,* `/etc/inetd.conf`. As a result, we use multiple different versions of such files for the different classes of hosts we are interested about, such as web servers. We differentiate between these files by using a naming convention of `file.''class''`, *i.e.,* we use a mnemonic name for each class as the file-name suffix.

e) Host management: The amplifying engine can assist in managing hosts on the network by tracking modifications to the `/etc/hosts` file. When a new host is added, the amplification engine can configure the DNS server and add client declarations to the DHCP server configuration so that the host can automatically obtain an IP address. Since DHCP can also provide network boot services, we can easily set up an automatic install procedure for new machines on the network.[3]

f) Software management: Finally, the administrator can centrally maintain file system images for different groups of hosts, and push new software and software patches by simply installing the software or moving files to the right directory. We must note that this can be different and much more complicated than simple mirroring. The flexibility of our amplification architecture allows the system to perform more complicated update tasks like license management, registry editing, *etc.*

IV. Related Work

Most efforts in automating system and network administration tasks are orthogonal to action amplification as presented here. To the best of our knowledge, this paper is the first to introduce and analyze the action amplification approach as a means of improving configuration management.

Early work on automating system management tasks for large-scale system installations has focused mostly on user account management for servers and workstations. The Moira system developed for Project Athena at MIT [16] maintains a centralized relational database of user accounts and other configuration information and periodically generates new configuration files customized for each managed host. After performing a series of necessary consistency checks, the new files are pushed to the hosts.

Sun's Yellow Pages (YP) and Network Information Service (NIS) [18] have been widely used for administering system access, naming of resources, etc. in large scale computing facilities. YP/NIS maintains a set of files called *maps* containing site-wide configuration information on centralized servers called YP masters. Remote clients need to communicate with YP masters when the configuration information they need to access is not maintained locally.

Increasing size, heterogeneity and complexity of system installations has led to more sophisticated configuration tools [3], [5]. For instance, `cfengine` [3] takes a *language-based approach* [17] for specifying abstract administration tasks and policies and mapping them to configuration actions. The introduction of a system administration language provides significant benefits in terms of simplifying repetitive tasks as well as allowing customization through an object-oriented interface.

[2]Note that it is possible that disabling a managed router's or host's interface will make it impossible to enable it again, because connectivity to the relevant router or host will have been lost. It is difficult to discern intent in this case, *i.e.,* whether the action is mistaken or intended. Our system can detect such ambiguous directives and prompt the administrator for confirmation prior to undertaking the requested action.

[3]The `/etc/hosts` file typically only specifies host-name and IP address, but we can overload the comment field for indicating any additional parameters such as the host operating system and the physical ethernet address.

The network management community has developed a range of technologies for managing heterogeneous networks, including models, mechanisms and standards for efficiently managing networks, including performing low-level configuration tasks on managed elements [2], [4], [6], [14], [1], [11], [13], [19]. This technology is usually combined with higher-level GUIs and is therefore subject to the same set of problems outlined in Section I.

V. Concluding Remarks

We have presented a management system that enables administrators to effectively handle the scalability challenge in network management. Our approach is to map administration actions in a single-host environment to the network at large, amplifying the effects of such simple operations as user addition/deletion, network interface management, *etc.* We described our prototype implementation, based on OpenBSD and using the `systrace` facility, which was originally developed as a mechanism for monitoring the behavior of critical applications and enforcing a security policy. Our implementation, which extends `systrace` in some minor ways, captures interesting operations on "interesting" files and maps them to a series of configuration directives for the appropriate network elements and services. Although we focused on a unix-like environment, the same principles can be applied on a Windows-based interface.

We believe that our approach is simultaneously simple and powerful enough to capture a large number of management tasks common to a large network. Its novelty lies in leveraging a management interface that is familiar to administrators, to make many common operations intuitive and easy to perform. Our examples of use demonstrate the flexibility of our approach, which can be used to handle such diverse tasks as management of users, routers/firewalls/VPN gateways, network services, host configurations, and software. Our system is easily expandable, can be tailored to a number of environments, and can be combined with many of the management tools currently in use. Our plans for future work include integration with such a system, and combining it with the [9], [10].

References

[1] L. Artusio. Profi le-based subscriber service provisioning. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2002.

[2] J. Boyle, R. Cohen, D. Durham, S. Herzog, R. Raja, and A. Sastry. The COPS (Common Open Policy Service) Protocol. RFC 2748, http://www.rfc-editor.org/, January 2000.

[3] M. Burgess and R. Ralston. Distributed resource administration using cfengine. *Software practice and experience*, 27:1083, 1997.

[4] J. D. Case, M. Fedor, M. L. Schoffstall, and C. Davin. Simple Network Mangement Protocol (SNMP). RFC1157/STD0015, http://www.rfc-editor.org/, May 1990.

[5] L. Cons and P. Poznanski. Pan: A High-Level Confi guration Language. In *Proceedings of the USENIX Large Installation System Administration Conference (LISA)*, 2002.

[6] G. Goldszmidt and Y. Yemini. Distributed management by delegation. In *Proc. of the 15th International Conference on Distributed Computing Systems*, pages 333–340, 1995.

[7] D. Harkins and D. Carrel. The Internet Key Exchange (IKE). Request for Comments (Proposed Standard) 2409, Internet Engineering Task Force, Nov. 1998.

[8] S. Kent and R. Atkinson. Security Architecture for the Internet Protocol. Request for Comments (Proposed Standard) 2401, Internet Engineering Task Force, Nov. 1998.

[9] A. D. Keromytis. *STRONGMAN: A Scalable Solution to Trust Management in Networks*. PhD thesis, University of Pennsylvania, December 2001.

[10] A. D. Keromytis, S. Ioannidis, M. Greenwald, and J. Smith. The STRONGMAN Architecture. In *Proceedings, DARPA Information Survivability Confernce and Exhibition*, volume 1, pages 178–188. IEEE Press, April 2003.

[11] I. Khalil and T. Braun. Automated service provisioning in heterogeneous large-scale environment. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2002.

[12] A. Konstantinou, S. Bhatt, S. Rajagopalan, and Y. Yemini. Managing Security in Dynamic Networks. In *Proceedings of the* 13^{th} *USENIX Systems Administration Conference (LISA)*, November 1999.

[13] I. Lück, S. Vögel, and H. Krumm. Model-based confi guration of VPNs. In *Proceedings of the 8th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, April 2002.

[14] G. Pavlou. OSI Systems Management, Internet SNMP and ODP/OMG CORBA as Technologies for Telecommunications Network Management. In *Telecommunications Network Management: Technologies and Implementations, S. Aidarous, T. Plevyak, eds, IEEE Press*, pages 63–109, 1998.

[15] N. Provos. Improving Host Security with System Call Policies. In *Proceedings of the 12th USENIX Security Symposium*, pages 257–272, August 2003.

[16] M. A. Rosenstein, D. E. jr. Geer, and P. J. Levine. The athena service management system. In *Proceedings of the USENIX Winter 1988 Technical Conference*, pages 203–212, Berkeley, CA, 1988. USENIX Association.

[17] D. Spinellis and V. Guruprasad. Lightweight languages as software engineering tools. In *Proceedings of the USENIX Conference on Domain Specifi c Languages*, page 67, 1997.

[18] Sun Microsystems. The network information service. In *System and network administration*, pages 469–511, 1990.

[19] A. Tal, B. Rochwerger, G. Goldszmidt, and Y. Koren. Khnum - A Scalable Application Deployment System for Dynamic Hosting Infrastructures. In *Proceedings of IM 2003*, 2003.