Lockdown: Simplifying Enterprise Network Management with Local Context

Andrew Blaich, Qi Liao, Brian Sullivan, Aaron Striegel, Douglas Thain, and Tim Wright
Department of Computer Science and Engineering
University of Notre Dame
Email: {ablaich, qliao, bsulliv9, striegel, dthain, twright} @cse.nd.edu

***Abstract*— The administrator of an enterprise network has a responsibility to enforce the policies on the network. Yet, most security mechanisms do not map well to the intended policies. This has been due to the prevalence of simplistic tools that have poor enforcement but, yet are easy to manage. While advanced commercial solutions do exist that have stronger enforcement, they are significantly harder to manage. To that end, we propose Lockdown, a policy-oriented security approach that builds on the concept of local context to deliver a lighter weight approach to enterprise network security while striking a balance between the level of enforcement and level of management available to the network administrator. In this paper, we describe how the Lockdown approach streamlines the process of network security management from network auditing to visualization to policy mapping to enforcement to validation. We demonstrate the strength of Lockdown through detailed assessments of an enterprise university network to show how local context significantly improves network management for the system administrator.**

I. Introduction

Network security policy is complicated and difficult to fully manage in an enterprise setting. While the commercial solutions provide a rich variety of mechanisms, they lack a streamlined approach that would allow them to be setup and managed efficiently. In a recent Computer Crime and Security 2007 survey [1], the collected data showed that the commonly deployed security solutions are the simpler and less effective ones. For instance, the survey showed that the lower end tools which include traditional firewalls have near ubiquitous deployment, found in upwards of 97% of the networks surveyed. The higher end solutions, such as endpoint security client software/(NAC) have considerably less deployment with their deployment in 2007 at 27 % down from 31% in 2006.

In short, there appears to be two broad categories of solutions for network administrators:

1) simply policy enforcement with easier manageability
2) rich policy enforcement with complex management

Clearly as evidenced by [1], network administrators are choosing the first category of tools in favor of manageability over security. Notably, such tools often derive their ease of management by implicitly trusting that Layer 3 (IP) and Layer 4 (Port) map to user and application.

To further illustrate the problem with lower end tools in regards to enforcement, consider an example based on *iptables* as shown in Figure 1. In the example, the desired high level policy is to allow outbound connectivity for non-secure web browsing. The natural rule for *iptables* or any network firewall would be to allow outbound port 80 with state preservation. Critically, the reliance on inference of application from port number allows for other applications not originally intended in the high level policy to gain access out of the network. As a result, the security mechanism of *iptables* would allow not only web browsing but also applications such as Skype or Gnutella that can arbitrarily configure port numbers or tunnel directly over HTTP. Although increasingly popular solutions such as deep packet inspection can address those concerns, those solutions suffer in the presence of user-agent spoofing and altogether fail when faced with end-to-end encryption [2].

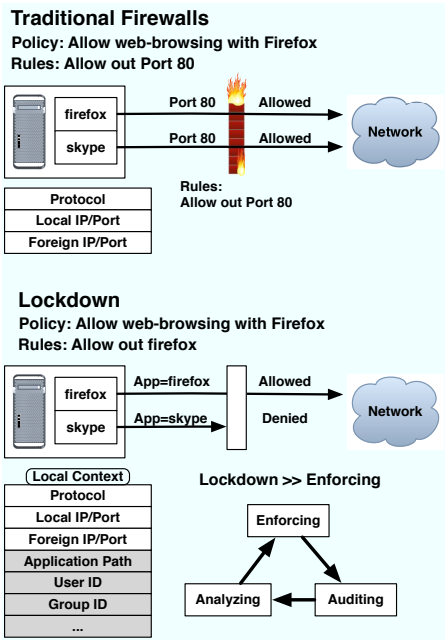


Fig. 1. Adding local context to streamline rule creation.

We posit that the perspective at which enforcement and monitoring occurs needs to be shifted in such a way that the *local context*, i.e. the why of a connection versus the where, forms the foundation of the security approach. By infusing enforcement with local context, high level policy can be followed more stringently as is seen in the "Lockdown" example of figure 1.

This work was supported in part by the National Science Foundation through the grants CNS03-47392 and CNS-05-49087.

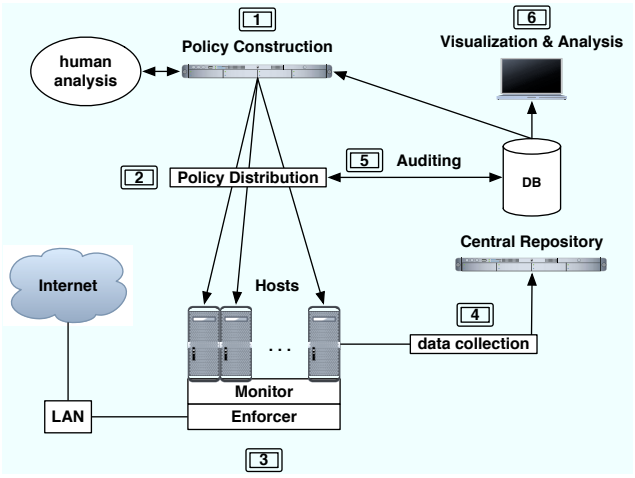


Fig. 2. Lockdown System Architecture, the number is indicative of the closed loop Lockdown operates in, i.e. 1,2,...,6,1,2,...,6

While enforcement with local context is certainly not a new concept, we argue that the monitoring (gathering, auditing, analyzing) of local context has profound implications for network management. Critically, local context strikes the ideal balance between ensuring proper policy enforcement and complexity of mechanism while also improving network management. In short, local context provides an extremely efficient representation of the *why* of a connection with minimal complexity increase over existing, well-deployed and well-understood tools. Moreover, local context meshes nicely with the reality of the enterprise network wherein resources for network security management are extremely limited or allocated only reactively rather than proactively.

To that end, we present the Lockdown security approach in this paper. Lockdown is a streamlined management approach for the enterprise network based on local context. Figure 2 gives an overview of the system architecture. Lockdown has several components that make up the system:

1) *Policy*: Lockdown improves the mapping of policy to mechanism by leveraging local context for rule construction. Local context allows for Lockdown to offer reasonable levels of expression while preserving clear observability from the policy statement which the rules originate from.
2) *Enforcement*: The Enforcer component allows for local context rules to be enforced through the use of a pluggable security module.
3) *Monitor*: The Monitor component enables the Lockdown system to gather local context natively from the hosts and forward it to a central repository.
4) *Auditing*: Auditing is used to validate that policy is being followed.
5) *Analysis & Visualization* These components allow for easy management of the network through visualization of the connections and discovering chains that occur as a result of user + application interactions. The data collected is in turn analyzed for patterns among resource usage and potential problem areas.

Lockdown provides an economy of mapping mechanism to policy that is expressive, but also lighter weight than high end host-based solutions. The ability to visualize and audit the network creates a platform to explore the local context gathered from all the hosts allowing for validation of policy as well as determining any sort of interesting anomalies occurring among user usage. Finally, the ability to identify problems and tie them to specific application easily are the key features of our system.

The core contributions of this paper are as follows:

- A systematic method for collecting and enforcing local context-based policy on hosts in a distributed fashion within an enterprise network.
- A framework for monitoring local context and enforcing related policy.
- A streamlined management system that includes analysis, visualization and auditing components.

Lockdown serves as a robust addition to an enterprise network infrastructure. Critically, we note that the goal of Lockdown is to complement, not replace the security infrastructure by plugging into existing work from data mining, anomaly detection, firewall analysis/management, and policy mapping.

Lockdown seeks to be complementary to the existing areas of work by supplementing network connectivity with additional information that can determine who the power users on a network are, what applications they are using, and other interesting tid-bits previously obscured. It also helps in determining the disconnect between policy stated and what is actually taking place in order to create a tighter more locked down network.

While no security is perfect, giving administrators and managers the details of exactly what is happening on their network leads to better security management principles and practices. Lockdown concerns itself with not creating an absolute perfect security solution, but rather how to manage a complex network system and the wide array of applications, users, and connections that occur on them.

II. Motivation & Background

To better motivate Lockdown, we explore several interesting cases drawn from the monitoring of hosts on the university network. Since April 2007, a prototype of the Monitor component of Lockdown has been running on roughly 250 hosts. The hosts are all *Linux*-based and are used for activities ranging from computation clusters to graduate student desktops to public Engineering lab machines. The monitor gathers the local context from each host which includes all network connections along with the related process, file, and user information associated with the connection. Further details about the Monitor can be found in Section III-C while details about the data collected is found in Section V.

Figures 3 and 4 demonstrates the notion that application cannot be inferred by port number. Both of these figures show that the client applications with established foreign ports are different than an administrator would expect to see. For instance, with port 22 in Figure 3 we can see that a majority of the connections are ruled by the typical *ssh* application

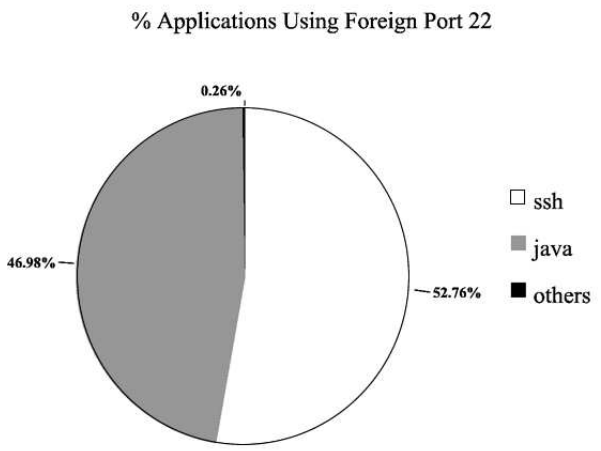


Fig. 3. Multiple Applications with Foreign Port 22 destinations

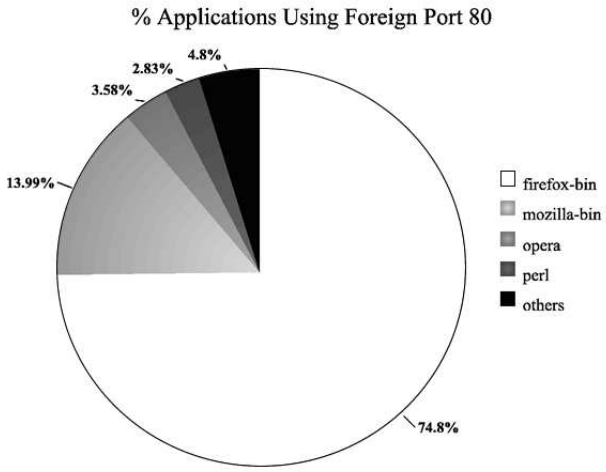


Fig. 4. Multiple Applications with Foreign Port 80 destinations

TABLE I
SINGLE CLIENT APPLICATION USING MULTIPLE FOREIGN PORTS

ssh	thunderbird-bin
22(ssh): 69.20%	993(imap): 85.31%
636(secure ldap): 28.46%	636(secure ldap): 10.42%
389(ldap): 1.63%	389(ldap): 1.77%
6010(x11): 0.58%	587(submission): 1.36%
7313(swx(ssh)): 0.09%	80(http): 0.29%
6012(x11): 0.04%	443(https): 0.27%
222(rsh-spx): 0.01%	25(smtp): 0.20%
	631(ipp): 0.19%
	6010(x11): 0.08%
	465(urd/igmpv3lite): 0.07%
	6011(x11): 0.02%

included in the *Linux* distribution, but a more intriguing case is the presence of *java* with just under half of the total connections. Upon further examination, the *java* application was discovered to be a program called *opennms* used to administer and manage the network in this instance by probing computers for *ssh* daemons.

Looking at Figure 4 one can see for port 80 there are the common web-browsers appearing: *firefox*, *mozilla*, and *opera* in addition to numerous other applications that are obtaining network connectivity through port 80. However, while there are only three primary application names, there are also multiple versions of each application being employed. The versions of *firefox* found on the campus machines correspond to: 1.5.0.10 and 1.5.0.12 while the most recent version available for download is at: 2.0.0.*. Upon further examination of the versions of *firefox* being used it was found that 19.47% of the hosts are running version 1.5.0.15 while the other 80.53% are using version 1.5.0.12. *Firefox* was additionally observed as being run locally on the hosts 94.95% of time while only 6.05% of the *firefox* instances were run off of a user space locate on a distributed file system. Restricting version number and application path is important to properly manage a multi-user network to ensure that only approved applications can be used.

Other issues occurring with inferring application based on port numbers can be seen in Table I. Most applications do not strictly use a single port, rather they use a multitude of ports for numerous different tasks. In Table I, *ssh* is shown as having used 7 different types of ports. Port 22 is the standard listening port which the *ssh daemon* runs on, 636 is used for *secure LDAP* lookup, 6010 and higher are used for *X11 forwarding* of the display back to a remote client. Applications typically require several ports to be open in order to fully work, yet these open ports are often not utilized 100% of the time enabling other applications to use them when idle. For a fully functioning application, all of these ports need to be opened, but it is necessary to ensure that the traffic utilizing these ports is the traffic which the administrator intended when he opened them.

Local context helps remedy the previous problems by providing what is actually occurring on the host. Through local context, hardened rule sets are formed and uploaded to the hosts for the enforcer to fully enforce policy that now has the notion of application, among other fields. By modifying the process of how hosts connections are monitored through a few small changes, the benefit in manageability and enforcement is increased tremendously. In addition to enforcement, administrators are aware of what is actually occurring on the network without having to install expensive hardware/software solutions.

Furthermore, in regards to enforcement, we are also motivated in improving application connectivity debugging when connections violate policy. Firewalls and related tools that enforce on a packet level, typically layer 3 or 4 of the network stack, will drop packets that are in violation of policy. The issue with dropping packets is that at the level in which the packets are dropped there isn't a way to alert the application *easily* of what is happening. Typically the user will end up

with a connection that appears to be doing something, but is just sending packets into a garbage bin, until either a timeout for the socket is reached or the user becomes frustrated and terminates the application. A better approach is to enforce at an entirely different layer within the computer so that explicit error messages can be returned for connections that violate policy. Section III-B discusses enforcement in depth.

III. Architecture

The Lockdown system architecture is a distributed system composed of host installable components in addition to a central repository for the database. Utilizing a distributed data-collection system of monitors and enforcers installed on every host the mapping of mechanism to policy is fully realized via use of the local context. Since the architecture is distributed, a global policy can be created for all hosts and then finely tuned for each host as individual needs may change over time in a closed loop.

The rest of this section will talk about each component in further detail, section III-A talks about how policy mapping to mechanism is achieved in Lockdown. Section III-B discusses the Enforcer module and how by using the Linux Security Module framework a locked down host can be achieved. Section III-C discusses the Monitor and how local context is gathered. Finally, section III-D describes the visualization and analysis components.

A. Policy

One of the most critical aspects of any enterprise security approach is how the enterprise policy is mapped to the network security mechanisms, be they end host or in-network mechanisms. In the ideal management case, the configurable aspects of the mechanisms (typically rules) map in an easily observable manner to the policy, be it one or more rules tied to a specific policy statement. The diverse array of work in the area of natural language processing with regards to security [3], [4] is a testament to the appeal of said aspect. In a similar vein, the mechanisms must be able to capture reasonable levels of expressiveness to enable reasonable confidence that the policy item is indeed addressed by the mechanism. Hence, the wide array of work on formal security expressiveness addresses this need [5]–[7].

However, as any administrator or researcher in systems will attest, the practical limits of systems and resources make complexity a natural enemy of robustness and security. Case in point from networking, the success of Ethernet can be largely attributed to its simplicity while other better performing but more complex solutions have fallen by the way side. To that end, we do not focus on the theoretical foundations of policy mapping but rather to focus on how local context offers a compelling economy of expressiveness, i.e. significant improvement to the efficacy of managing the network with negligible increases to complexity. In short, local context adds the ability to create rules that are cognizant of the normal UNIX user (akin to a network ACL) and application (name, path, arguments). In an oversimplified sense, the security mechanism offered by Lockdown would appear to simply be an enhanced firewall, i.e. *iptables++*. In contrast, as we will show in later sections, the consideration of local context can dramatically improve not only policy mapping but also network assessment, mechanism auditing, and debugging, i.e. streamlining the core management aspects expected of a system administrator.

```
# Policy Statement:
#   Only F-T employees can browse the Internet,
#   all employees can only access the intranet
#  (10.1.1.50), with Firefox as the only
#  browser
allow out to * when app=firefox
     AND group=FTEmploy
allow out to 10.1.1.50 when app=firefox
# Policy Statement:
#   Root cannot accept incoming connections
#   except for sshd
allow in from * when app=sshd
deny in from * when user=root
# Policy Statement:
#   Enable Condor (grid) functionality on
#   the LAN
allow out to 10.* when app=~condor/bin/*
     AND user=condor
allow in from 10.* when app=~condor/bin/*
     AND user=condor
# Remainder of policy
...
# Policy Statement:
#   Deny all other communications
deny all
```

Fig. 5. Example policy statements illustrating local context

To illustrate the expressiveness of local context, consider the three policy statements captured in Figure 5 regarding web access, external connectivity, and enabling grid applications. In the first case, the policy statement is that only full time employees are permitted to browse the Internet but all employees may access the internal company web server (intranet). Consider how such a policy might be enforced using current mechanisms. The traditional firewall would only be able to capture port numbers and IP addresses, only racheting down access to port 80. File-level ACLs could guard the usage of the application (*firefox*) if all web access was strictly limited to full-time employees via a group-wise ACL. However, the application itself can be used by multiple groups of users whose type of connectivity is limited by their employment status. User-dependent firewall rulesets could mitigate the issue but create management issues for maintaining consistency and debugging. User-authenticated application proxies also satisfy the policy statement, but have issues with encryption and crafty masquerading applications. In contrast, the addition of local context creates straightforward rules that provide sufficient expressiveness for the overarching policy. Moreover, multiple mechanisms are not needed allowing for a single rule consistent rule set to enforce the desired behavior.

In the second policy statement, the desire is to prevent root from directly offering services besides *ssh*. In short, the owner of any network services must be properly contained with its own individual policy statement reflecting a push towards

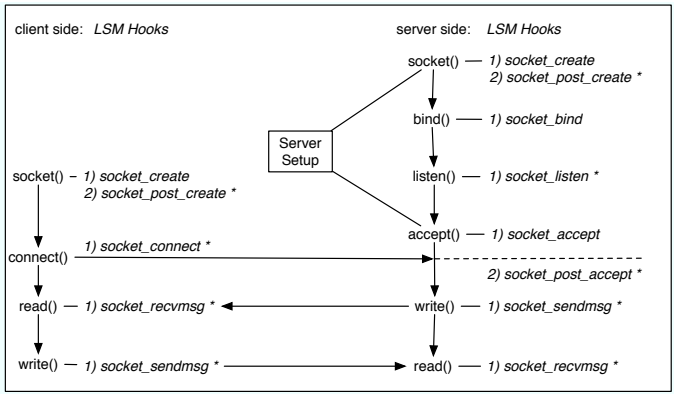


Fig. 6. Linux Security Module Hooks (LSM): * represents hooks where local context policy enforcement takes place.

RBAC-style management. Hence, developers are not precluded from offering services but can no longer simply bludgeon open access via root to at least force priviledge escalation exploits by attacking systems. The third policy statement in the example gives an example of such a network service. In the third example, the policy is to enable the functionality of Condor, a popular grid computing application [8]. A key property of Condor is that it embodies the properties that many emerging P2P and highly distributed applications share, namely that connectivity between nodes is essential but may be difficult to restrict to a specific port. With the entrance of local context, the adminstrator need no longer open a specific port or even consider ports for that matter. Rather, local context allows the rules to deal with the application in question and creates rules much more in line with the policy, i.e. make Condor work.

B. Enforcer

The job of the Enforcer component is to interpose when a network operation is attempted and either allow the operation to continue or to deny and shutdown the network attempt based on local context. Since sockets are the method in which the kernel manages network connectivity, it is a prime place to enforce on. Also, working at this level in the operating system allows direct access to the local context information. Figure 6 shows the necessary hooks that the Enforcer examines. With the Enforcer working at the system call level, it is able to solve the problem of acting on local context, providing interesting feedback to applications that have their sockets denied, and achieve an environment that is easily managed since it has a better view of what is actually occurring with network connectivity.

There are several paths that can be taken to achieve enforcement at the system call level. The first method involves system call interposition in the form of creating a kernel module and intercepting on system calls. This way involves creating your own version of the standard system call and overwriting the system call table to point to your code. This method is clunky, error prone, and has many traps [9], especially with the 2.6 version of the Linux kernel which was hardened against the ease in which someone could intercept systems calls in the 2.4 Linux kernel. The second way involves actually modifying the kernel source code. While this method proves one of the easiest without relying on frameworks, it requires that a custom kernel be compiled and loaded onto every system in the network. Custom kernels are un-manageable and time-consuming for an already heavily taxed IT department. The third method, the one selected for Lockdown uses the Linux Security Module framework

The Linux Security Module framework or LSM, standard in the 2.6 kernel, but available as a patch for the 2.4 version, has several hooks placed within an assortment of system calls that allow upcalls to loadable modules implementing the functions [10]. The Lockdown LSM module can be inserted dynamically at anytime, without the need for a kernel re-compile, and is responsible for enforcing the policy that is pushed out onto each of the hosts. While the concept of a pluggable security framework has been under fire by kernel developers, Linux creator Linus Torvalds says it is here to stay for the foreseeable future [11].

To enforce network connectivity we focus on the socket hooks, see Figure 6. We can determine whether certain sockets should be created before they are, or if an incoming connection on a listening host should even be accepted. If upon passing through the LSM the firewall or some sort of Intrusion Prevention/Detection System located down the line chooses to close the flow this can still be done since the LSM is an additional security feature and the best security measure is a layered approach.

Ideally a socket needs to be validated before it is created, allowed to connect, while it is listening, before it accepts a foreign connection, and/or when sending / receiving messages. There are however a few subtle points that need to be made clear as to how the hooks and the system calls interact with one another. (Although we enforce system calls, policy only refers to users, applications, and hosts.)

For a large majority of the hooks if an error code is returned i.e. -*EPERM*, then the system call that made the up call to the LSM hook will return an error as well and terminate. However, with hooks that have “post” within their name such as: *socket_post_accept*, the error code that is returned is not caught. The post hooks serve primarily as monitoring points and not enforcement points.

Socket post hooks:

- *socket_post_create*
- *socket_post_accept*

However, these two calls are essential in ensuring proper enforcement due to their implementation in the kernel. In both cases and more so in the *socket_post_accept* hook the foreign connection has yet to be established. The system is actually listening for connections in between the *socket_accept* and *socket_post_accept* hooks, as is illustrated in Figure 6. This leads to an interesting case of not being able to simply return an error code if the connection in the *socket_post_accept* needs to be denied. In order to deny the connection the socket needs to be shutdown through the *socket shutdown* operation.

For full enforcement, the hooks necessary to send and receive messages are needed. The *socket_recvmsg* and *socket_sendmsg* work at a higher level than where *iptables*

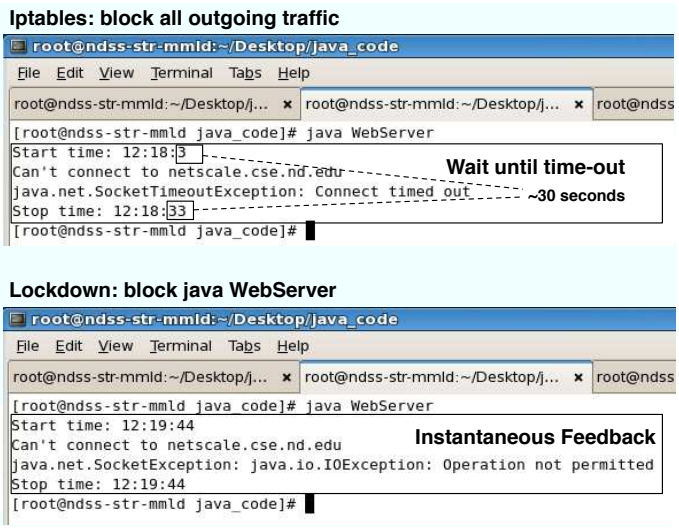


Fig. 7. The application view of enforcement comparing *iptables* (above) versus Lockdown (below)

validates on a packet by packet basis. When the application utilizing a socket wants to send a message these hooks are called and when finished, the data is passed to the network stack which is responsible for breaking down the information according to packet sizing and other related constraints. These hooks do introduce a small amount of latency as can be seen in the performance section later, but the level of security they provide is a necessary trade off. In cases where the LSM may become loaded after connections are established or if during a new policy push a connection is able to be established during the very brief switch, or if even the new policy suddenly disallows previously allowed applications that have sockets already established, are the reasons why we choose to use the send and receive socket hooks. The added security they provide ensures policy is fully enforced at all times.

The notion of being able to more effectively debug network connectivity issues is greatly enhanced with LSM enforcement. Figure 7 presents two cases. In the *iptables* case, all outgoing traffic is blocked as a *java* application we wrote to connect to our laboratory's web-server is initiated. The *java* application had a socket timeout value set to 30 seconds, where the default for *java* sockets is no time-out. Since *iptables* will simply drop packets and not inform the application of what is happening the application sat until 30 seconds had passed and then the user was informed of the *SocketTimeoutException*. Had there not been a timeout value set (default for *java*), the application would have sat indefinitely with no feedback for the user. In the Lockdown case, a rule was put in to block the *java WebServer* application. As soon as the application was executed the *IOException* was returned informing the user that the operation was not permitted. By using an LSM based enforcement approach the user/administrator is able to debug the problem easier than if packets were simply being dropped with no feedback of what is occurring.

LSM Installation:

By default there can be only two LSMs loaded onto a system, however in the case of the *Linux Fedora* distribution, *SElinux* comes installed along with the *linux capabilities* module. To install the Lockdown LSM, *SElinux* and *capabilities* need to be disabled so that Lockdown is able to register as the primary security module on a host. This fix is done by supplying two additional commands to the boot-loader configuration file.

```
LSM Installation Steps:
1) modify the boot-loader configuration file to
disable selinux and capabilities
2) reboot system
3) insert the Lockdown LSM
4) create device driver node (enables rules to
be transported from user-space to kernel-space)
5) run policy loader program to validate and
push new rule sets into Lockdown
```

Fig. 8. LSM Installation Steps

LSMs are capable of being "stacked", allowing multiple modules to be loaded onto the system, however by default this capability is not included in the kernel. The primary security module needs to supply the stacking ability or a third party module [12] needs to be loaded that takes care of the stacking for the LSMs that eventually will be be loaded.

Policy Deployment:

Policy is deployed from a central server to the hosts running Lockdown via a pull mechanism. The Lockdown LSM polls the central server at a standard interval and is responsible for checking if the policy file on the host is outdated. Based on the cluster in which a host belongs a different rule-set will be downloaded that reflects the unique set of behavior inherent to the cluster for which the host is a part of.

C. Monitor

The Monitor is deployed onto each host possible throughout the network. The purpose of the Monitor is to gather the local context related to network activity on the host. Since the only way to accurately know exactly what is occurring on a host is to monitor it by physically being at the machine, either in software or by some other means, this is why we deploy our monitor in a distributed fashion among all the hosts on the network instead of monitoring a central location. The information gathered from each host allows for accurate auditing and policy management in a closed loop.

While the monitor was prototyped as a shell script for simplicity in deployment and development time, a more robust version has been prototyped using an LSM module. The LSM version provides the same amount of information as the shell script, but also has the ability of instantly determining the direction of the connection (incoming/outgoing) without the need for detailed post processing within the database. The amount of data generated is roughly on the same order as the shell script, and all connections, including very short ones are caught, which is a limitation in continuously polling the same tools every few seconds looking for changes. The data from the LSM is produced as sockets are established and torn down allowing accurate connection logging. Other advantages

of using the LSM is that the data can be correctly formatted in such a way without having to rely on stream processors such as *sed* and *awk* in the shell script to re-format the outputs of the tools. The LSM version of the Monitor is incorporated with the Enforcer such that only one LSM needs to be loaded onto a system using Lockdown. As connections are allowed and denied these stats can be kept for a connection and used to further supplement the data collected.

The data collection from April to September 2007, has been using the shell script Monitor. At the time of writing, the agent is deployed on *Linux*, *Solaris*, and *OS X* platforms while a native *Windows* version is under development. The Monitor is a root installed *BASH* shell script that gathers and sends the local context of the monitored host to the central repository. The simplicity of the Monitor lies in its ability to run the commonly found tools *netstat*, *ps*, and *lsof* on any *Linux*/*Unix*/*BSD* based operating system.

These three tools are used instead of any single one in order to gather the entire local context which includes: *ppid, pid, uid, gid, foreign ip, foreign port, local ip, local port, full application path + arguments* among other information. No single tool, except *lsof*, gathers all of this data and glues it together, however the problem with *lsof* is the large amount of data it generates because *lsof* reports back all open files and libraries an application is using.

After *netstat/ps/lsof* executes, the output is properly formatted into a local buffer via a series of *sed* and *awk* commands. The currently buffered data is compared to the previously buffered data from the previous iteration of the tool by using the *diff* application. The diff'ing of the data enables the Monitor to capture the start and stop of each connection while avoiding needlessly collecting redundant data that results from multiple iterations of *netstat/ps/lsof*. The three tools are used to create a comprehensive view of information. *Netstat* provides the information relating to the (IP address, port number) tuple of the connection along with the PID, UID, and (short) program name responsible. We use *ps* to drill down even further and discover the full application's path along with all of the arguments supplied to run it. *Lsof* is checked against the *ps* data to ascertain if an application is spoofing it's path/name and to obtain any and all files a process may be using.

D. Visualization & Analyzer

In order to visualize the network connectivity among hosts within the Lockdown monitoring pool, we modified the source code for SoNIA [13] see Figure 9. By comparing the topological changes in the connectivity graph, Lockdown can compute the invariants and/or evolution of the monitored networks. Each host node is assigned a unique identification number and the edges between the nodes represent the established connections between them. Edges with higher weights (the magnitude/number of connections) are shown with a thicker line. Each node in the connection chaining graph is augmented to contain additional information shown in a pop-up window that the administrator can further investigate by clicking on the node in question. The additional information available for each node includes the network interfaces (IP) involved with the connection, the top local ports, applications and users that made the most connections on that node during the specific time interval, the in/out degrees of each node, edge weights, etc. For host-based chaining, dropdown menus are provided in the interface for selecting which *User* and *Application* type to view for the connections. The edges between host nodes are highlighted in different colors to reflect the change in user and application. The GUI interface represents the connectivity chaining at the host, user, and application levels. By stepping through each slice (a time window that is customizable to reflect the granularity level, for example hourly, daily or weekly), the system and network administrator can examine all connections between any pairs of monitored hosts that occurred during the time interval.

TABLE II

10000 CONNECTIONS TO A LOCAL WEB SERVER, (#) REPRESENTS THE RULE NUMBER

Test	Mean (seconds)	Standard Deviation
Base case	12.8	0.4216
Lockdown(1000)	14.8	0.4216
Lockdown(100)	14.9	0.3162
Lockdown(10)	14.8	0.4216
Lockdown(1)	14.9	0.3162
iptables(1000)	13.8	0.4216
iptables(100)	13.1	0.3162
iptables(10)	13	0
iptables(1)	13.1	0.3162

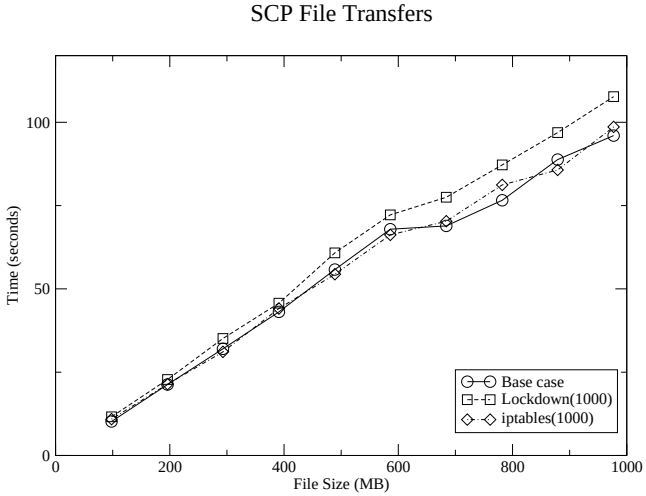


Fig. 10. SCP File Transfer micro-benchmark

IV. PERFORMANCE

Our goal for Lockdown is to improve management, but in order to do so we need to demonstrate that the overhead of such tools is in itself reasonable. The Enforcer was put through a battery of micro-benchmarks to ascertain the effect the LSM has on performance. Results are compared to a base-case with a computer that has neither the LSM or firewall loaded and one with only a firewall loaded.

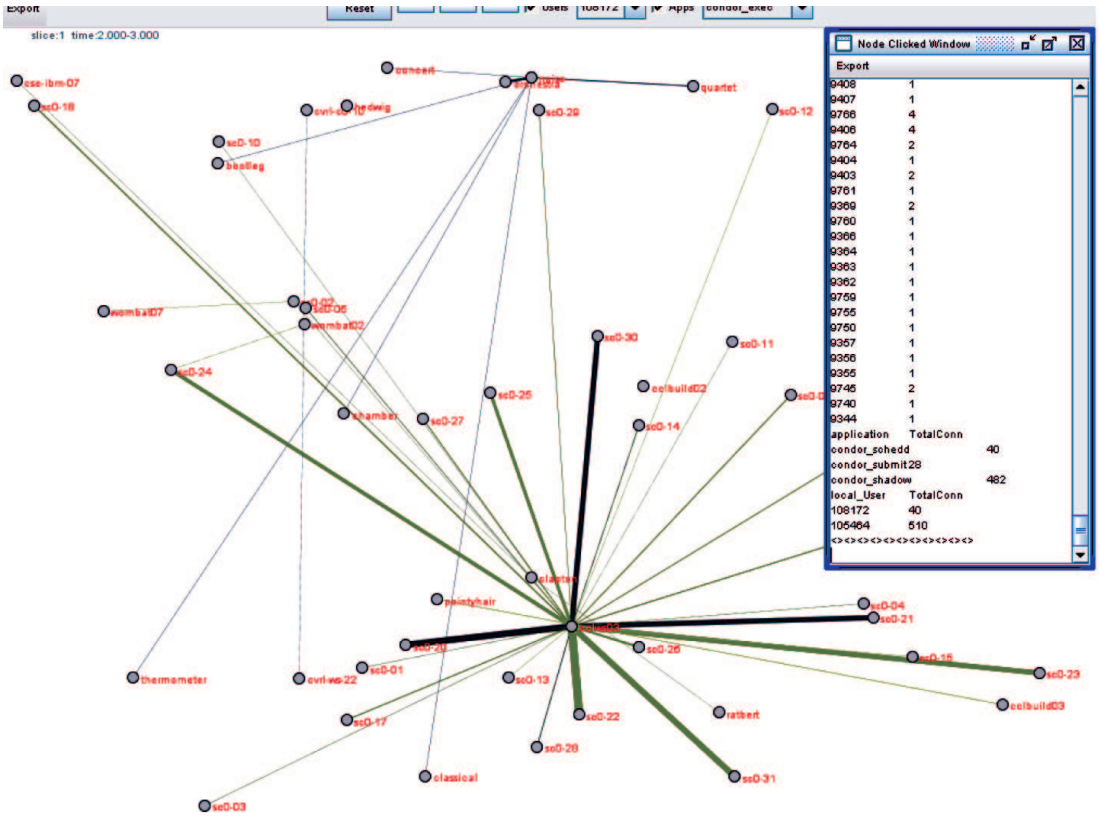


Fig. 9. Screen shots of SoNIA visualization tool.

Table II shows the results of the first benchmark that tested the performance of the LSM against creating 10,000 sequential connections to a web server on the same network. No data was sent or received, this benchmark simply tested the establishing of a socket to a web-server. The LSM and *iptables* were both tested with 4 different rule set sizes, where the rule that allows the connection through is the number represented next to the test, i.e. Lockdown(1000) means the 1000 rule allows the connection. Each test was run ten times on an x86 based Mac-mini [Intel Core Duo 1.66Ghz, 512MB RAM, & 40 GB HD] with Fedora Core 6 installed connected to a Netgear 100Mbps router on which an identically equipped Mac-mini running Fedora Core 6 was hosting a website. There is on average just over a second difference between using the LSM versus *iptables* spread out over 10,000 sequential connection attempts. The difference in performance is primarily because of the lookup associated with a large set of rules and the added enforcement the LSM provides through a form of system call interposition.

The second benchmark represented in Figure 10 shows the results of an SCP file transfer from the same two systems as outlined in test one. Once again each test was done 10 times for each file size ranging from 98MB to 977MB in increments of 98MB, numbers were chosen randomly, with the rule allowing the connection to be number 1000 in the list. Similar results were observed when compared to test one. *iptables* performs slightly faster, but the LSM overhead is not significant enough to cause performance problems. The LSM remained close to *iptables* in terms of performance time, but has a small overhead from the send and receive message hooks that are invoked every-time a block of data is sent from the application.

We conclude from the results of the benchmarks that the LSM's overhead is minimal when compared to the low-end of the tool spectrum, *iptables*. While further development on the LSM can lower the overhead the current status is acceptable for an actual production environment.

V. Experience using Lockdown

With the permission of our department's system administrators the Lockdown Monitor has been deployed and been running from April through September 2007. We have been analyzing, visualizing, and observing the behavior of users and applications associated with the network connections captured and reported back to the central server via the installed Monitors. In order to deploy the Enforcer onto our clusters we first need to determine how the network is being utilized so that when loaded with the rulesets the systems are still functioning according to the University's network policy.

This section describes the interesting analysis and results derived from the deployment of the Lockdown Monitor. It presents what information we were able to gain and what can be offered by the system in future use. Throughout this section, the following will be discussed:

- Storage and bandwidth requirements of a deployed Lockdown system.
- The number of connections made by hosts, users, and applications within our department.

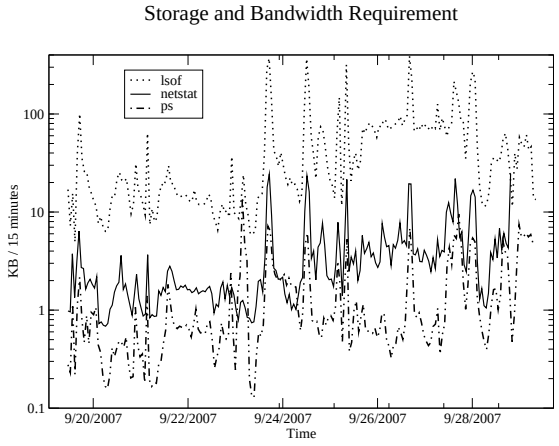


Fig. 11. The average sizes of raw data files uploaded by the Monitors for all hosts.

- Identifying the top users among the hosts and the applications used.
- The chaining of connections from monitored hosts and the applications + users responsible.

A. Storage and Bandwidth Requirement

To show that by turning on a system such as Lockdown the network does not become overloaded, we measured the storage and network bandwidth. The bandwidth metrics are derived from the raw data files uploaded by the Monitors from the hosts. Figure 11 shows the average file sizes over ten days from all hosts within the monitoring pool. It is seen that the average data size for *netstat* and *ps* has a minimum of about 2 to 3 KB every 15 minutes per host (interval in which the Monitor uploads data to the server), with occasional peaks at approximately 25 KB. *lsof* is the largest among the three files and oscillates depending on host usage. Roughly speaking if there are 500 monitored hosts with an assumed average of 100 KB for all three files, the storage on the central server reaches 4-5 GB per day and only about 0.5 Mbps on the local network.

B. Number of Connections Made by the Hosts, Users, and Applications

Figures 12 to 16 shows the hourly number of connections made by the users (local versus enterprise) on all monitored hosts. The UIDs are checked against the University's LDAP service to determine if they are enterprise users.

- Figure 12 shows the connection patterns made by users on the *cse-gw* hosts which solely consists of graduate student office desktop machines. The diurnal pattern can be clearly seen.
- Figure 13 shows the connectivity of users from the *helios* cluster of machines, which are public lab machines for all engineering students. The three spikes observed on the *helios* machines corresponds to unsuccessful `ssh` attempts from computers in Taiwan, Mexico and US during the hours 0-1am of 9/11, 7-8am of 9/14, and 2-3am 9/17 respectively.

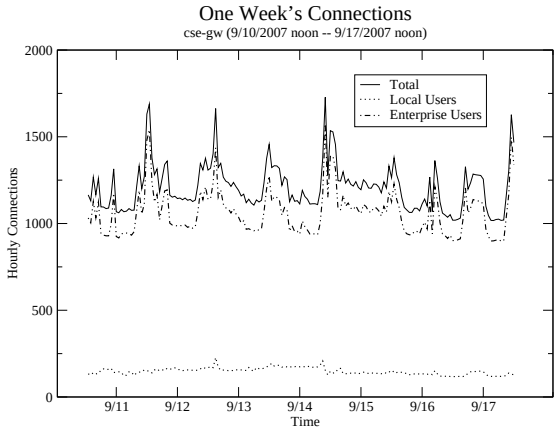


Fig. 12. Hourly Connection made by Enterprise Users and Local users: cse-gw cluster

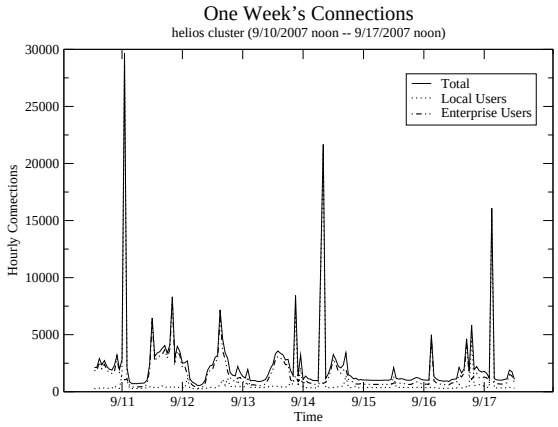


Fig. 13. Hourly Connection made by Enterprise Users and Local users: helios cluster

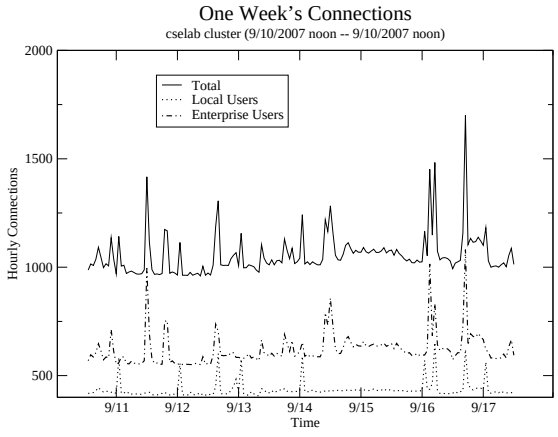


Fig. 14. Hourly Connection made by Enterprise Users and Local users: cselab cluster

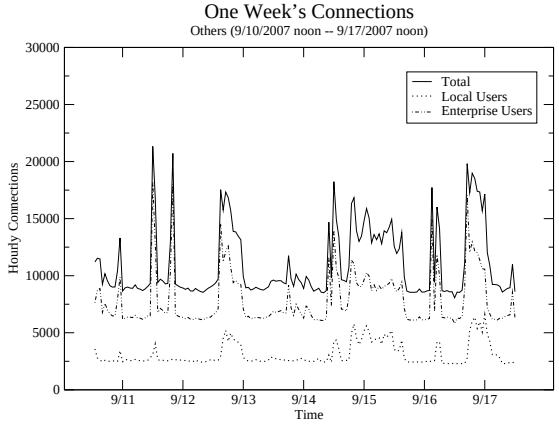


Fig. 15. Hourly Connection made by Enterprise Users and Local users: all other machines

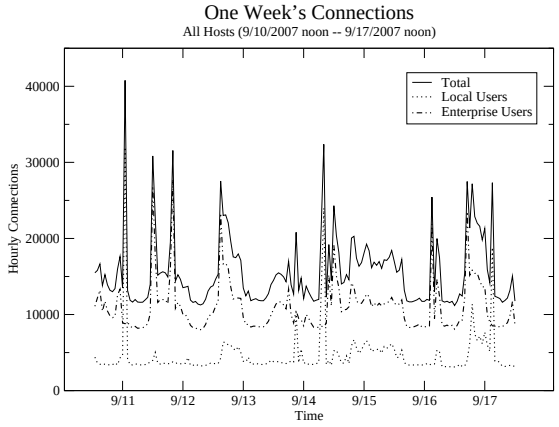


Fig. 16. Hourly Connection made by Enterprise Users and Local users: all clusters

- Figure 16 shows connections from *all* hosts in the monitoring pool. It is observed that the cyclic behavior occurring is driven by enterprise-scoped users, whose behavior is also less predictable, but nevertheless dominates the pattern of connections. An enterprise scoped user accessing their home directory with an application invoking network activity would create this type of behavior (i.e. queued grid jobs).

It is important to notice that with the help of the local context aware network management included in *Lockdown*, the information about the monitored connections has a very fine level of granularity. The level of detail on a connection is not only limited to what host was responsible, but rather ***who*** on the host was. Therefore, an identity of users in addition to the identity of hosts (IP/Port) is attached to traffic monitoring.

C. Top Users Having Most Hosts and Applications

From Figure 12, we can see that the enterprise users dominate the network connections. However, we do not know which enterprise users from it. Diving down deeper through the monitored data it is observed in Figure 17(a) which user's among the enterprise pool are responsible. Further drilling down with the gathered data we can see the top applications run by the user in question (e.g. uid 116670) that's responsible for the connections, as illustrated in Figure 17(b);

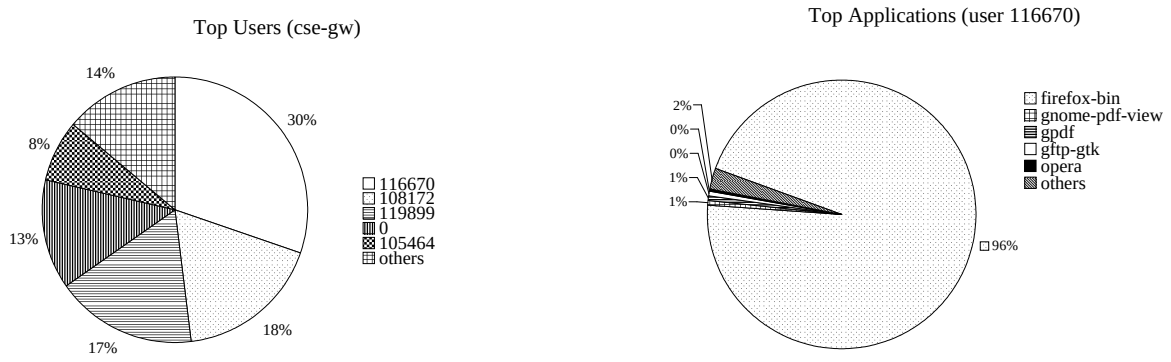
Another interesting property with the data *Lockdown* gathers is to watch the combination of *users*, *hosts* and *applications*. Figure 18(a) shows the top users (distinguishable by different machines they logged on) that had connected to the most unique foreign hosts over one week. Figure 18(b) shows the top users that had established connections using the most unique programs in the same week. While there are some CSE grad student users running experiments and the *Condor* system users appearing in Figure 18(a), the users that logged into the public computer lab shown up in the top list in Figure 18(b) suggests that the grid users in scientific computing contact many distinct hosts but, the variance among applications used on those hosts is few. Whereas in contrast, the physically active users, rather than user automated tasks (grid jobs), are more interesting in that the number of distinct applications used on those hosts to make network connections are diversified.

D. Connection Chaining

This section presents the context-aware connectivity chaining on the host, user and application level possible/discovered with Lockdown. Bipartite matching allows for Lockdown to show the applications and users at both ends (assuming both hosts are monitored) of the network connections in addition to the responsible hosts and port numbers. With *Lockdown*, it is possible to construct a connected graph on the level of hosts, users, and applications for the purpose of network analysis. The chains themselves can take multiple forms ranging from considering the complete chain (files, process/application, user, host) to considering high level topologies (user only, application only, user and application, host only). The topology can then be examined to assess both bottlenecks in performance (downstream dependencies) as well as areas of trust (*host1* indirectly trusts *host2* from its direct trust of *host3*). Moreover, the chaining of the various levels of context (user, application, host) extracts areas of trust relevant to inferring containment methodology (virus or exploit propagation), attack graphs analysis, risk management, and forensic analysis.

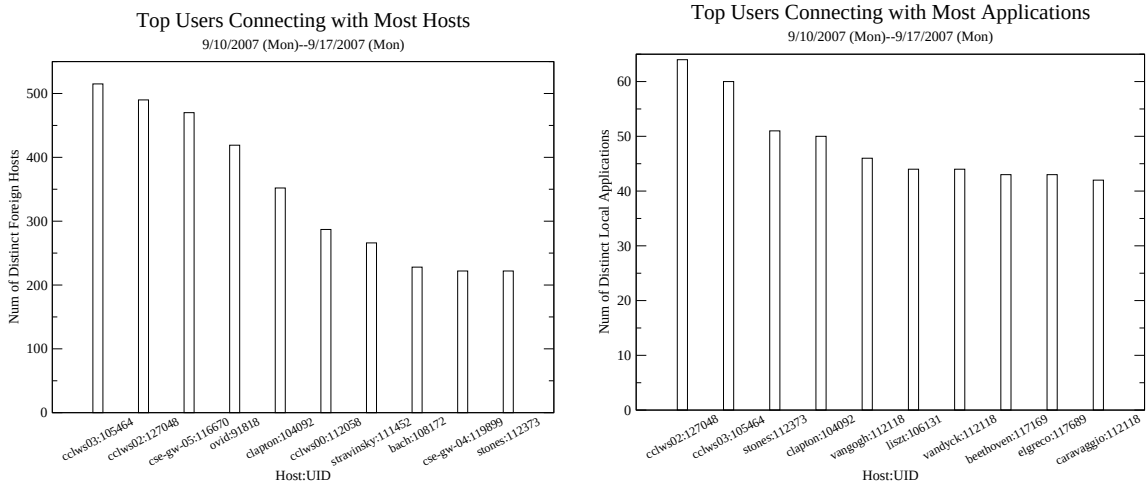
The algorithm we developed for bipartite matching uses two hashtables for linking the source and destination identifiers in linear time with regards to the number of total observed established connections. In its simplest form, a bipartite matching is found if an established connection recorded on Host A with source and destination identifiers src_A and dst_B matches another established connection record on Host B with src_B and dst_A within the same time frame. The time frame can be a varied by granularity of as low as a second to a much coarser one such as hours, days, weeks or months.

Table III shows an example of such connection matchings after the fusion of the data uploaded by the Monitors. Each new connection chaining record begins with the *start* and *stop* time of such a connection and is further divided into the left and the right part. The left part is the *local identity* in terms of host name, IP/Port pair, user, and application associated with the connection. Similarly, the right part is the *foreign identity* in the same format.



(a) Top Users (b) Top Applications

Fig. 17. (a) shows the top *users* in *cse-gw* cluster making the most connections, (b) shows the top *applications* being run by a selected user (116670) that's responsible for making the most connections during a week's monitoring period.



(a) Top users with most distinct hosts. (b) Top users with most distinct applications.

Fig. 18. Top users contacting most *distinct* hosts vs. top users making connections using most *distinct* applications. It suggests different connection behaviors between scientific grid users and desktop human users.

TABLE III

SELECTED OUTPUT OF BIPARTITE MATCHING OF ESTABLISHED CONNECTIONS. WITH THE HELP OF LOCKDOWN SYSTEM, WHICH APPLICATION RUN BY WHICH USER AT BOTH ENDS OF CONNECTIONS ARE IDENTIFIED.

Start	Stop	Local Host	Local Port(protocol)	Local User	Local Application	Foreign Host	Foreign Port(protocol)	Foreign User	Foreign Application
1177527137	1177527148	catbert	631(tcp)	0	cupsd	ratbert	34406(tcp)	97392	gnome-pdf-view
1177543303	1177543309	catbert	631(tcp)	0	cupsd	wally	35775(tcp)	92362	gedit
1177448975	1177449026	dustpuppy	54427(tcp)	105464	parrot	sc0-18	9094(tcp)	108172	chirp_server
1177391778	1177391807	noise	40096(tcp)	33	dumper1	concert	33084(tcp)	33	amandad
1177392075	1177392151	noise	40211(tcp)	33	dumper3	chamber	38429(tcp)	33	gzip
1177392075	1177392151	noise	40212(tcp)	33	dumper3	chamber	53342(tcp)	33	sendbackup
1177478126	1177478133	noise	41128(tcp)	33	dumper1	noise	41127(tcp)	33	chunker1
1177345841	1178341216	orchestra	32797(tcp)	27	ora_pmon_testd	orchestra	1521(tcp)	27	tnslsnr
1177345841	1179571284	orchestra	1521(tcp)	27	tnslsnr	orchestra	32797(tcp)	27	ora_pmon_testd
1177515292	1177515299	orchestra	36019(tcp)	317	httpd	orchestra	1521(tcp)	27	oracletestdb
1177610657	1177611222	sc0-16	9094(tcp)	108172	chirp_server	bomber	49857(tcp)	102744	condor_exec.e
1177610633	1177610638	sc0-17	9710(tcp)	108172	condor_schedd	bomber	9788(tcp)	108172	condor_startd
1177625404	1177625765	sc0-17	9314(tcp)	102744	condor_shadow	classical	9868(tcp)	108172	condor_starter
117748953	***1177548992***	***theresa***	***34479(tcp)***	***97464***	***ssh***	***dilbert***	***22(tcp)***	***0***	***sshd: root***
1177459056	1177459112	wally	34739(tcp)	92362	gedit	catbert	631(tcp)	0	cupsd
1177636992	1177636998	wombat02	9094(tcp)	108172	chirp_server	bootleg	38394(tcp)	97399	java

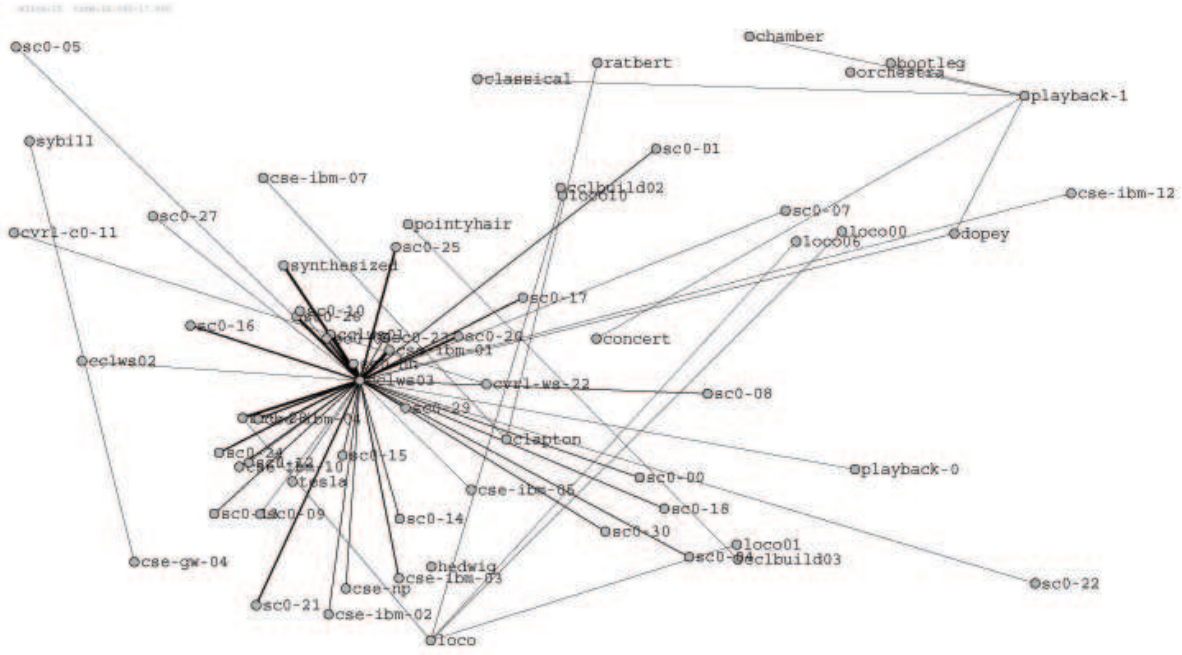


Fig. 19. Visualization of the bipartite matching from data collected on Saturday July 14, 2007 2pm-3pm EST.

Before, at one end of the connection (server side), the identity of ***who*** is connecting to the server is vaguely inferred from the *IP/Port* pair (assuming only user A can use that client machine). Now, the identity of ***who*** is connecting to a host can be precisely known from the bipartite matching without inferring from the IP/Port in terms of which *user* and what *application* are at *both sides* of the monitored connections. For example, in Table III, it is clear that from time *1177548953* to *1177548992*, user *97464* on host *theresa* using program *ssh* connected to the *ssh daemon* run by *root* on host *dilbert*. This is extremely useful for evaluating the effectiveness of the enforcement of the existing policy in the enterprise network, with a side benefit for forensic systems.

Figure 19 shows a snapshot of the visualization that Lockdown performs on the bipartite matching with the help of modifications done to [13]. Figure 19 is primarily showing a user on a specific machine fanning out a job onto Condor for a grid based computation.

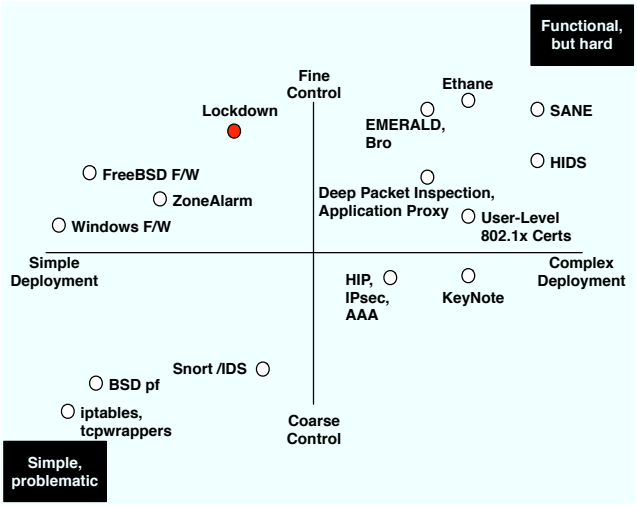


Fig. 20. Contextualizing Lockdown within existing network security mechanisms

VI. Related Work

In a broad sense, the work in this paper touches on the vast array of research already conducted with regards to firewall/policy analysis [14]–[18], intrusion detection [19]–[22], user/host authentication [23], [24], and recent clean slate design security efforts [25], [26]. Figure 20 attempts to capture where Lockdown lies on the axes of deployment complexity (x axis) and granularity of control (y axis). In some sense, the figure captures the range of solutions ranging from lightweight, simple firewall solutions to pervasive, heavyweight solutions that encompass the entirety of the enterprise. Notably, the figure focuses on standards body and research works with the primary discussion below regarding commercial solutions.

At the lower end of deployment complexity, traditional host-based and in-network solutions are located including firewalls (*iptables & OpenBSD-pf*) and flow monitoring solutions that act in an application-independent manner (*Snort*, classic anomaly detection systems, etc.). Notably, these devices suffer from the inability to understand the context of the connection, acting on Layer 3 / Layer 4 data or broad patterns of activity (signatures, anomalous traffic patterns, etc.). However, given that these techniques are often the initial mechanism for applying policy, significant research has explored how to validate policy mapping. Hamed and Al-Shaer [14] noted a taxonomy of conflicts in policy for network security devices with their previous work applying graph-based boolean function

manipulation to distributed policy analysis [15]. Guttman [16] constructed a global policy definition language with algorithms for verification while Bartal et al in [17] separated policy from topology along with a more modular architecture with Firmato.

Ioannadis et. al in [27] introduced KeyNote which addressed the issue of distributed firewall management. While the kernel enforcement mechanism of KeyNote shares aspects of the Lockdown enforcer component, KeyNote operates largely in the same rule domain as traditional IP firewall rules (IP, port) only with authentication (digital credentials) enforcing user identity. The recent clean slate efforts of SANE [25] and Ethane [26] move enforcement to the network switch itself with Ethane operating in a slightly less heavyweight manner than SANE. Similar to KeyNote, Ethane and SANE force the users through a centralized controller (digital certificates via IKE in KeyNote) to validate connectivity with the resulting authentication being a pre-requisite for proper LAN routing. Critically, the clean slate architectures of SANE/Ethane represent a sizeable cost in terms of changing network hardware. Moreover, we note that none of the noted works address management, focusing exclusively on the how of enforcement rather than how management of the network might be improved through visualization or auditing.

Deep packet inspection (DPI), i.e. application-specific proxying, trades processing speed for the ability to fully evaluate the state of the application-layer protocol. A typical deployment of DPI might involve an in-band application-aware IDS or forcing users to authenticate through an application-specific proxy (ex. web proxy). While this is marginally effective for the most basic of applications, DPI must continually react to application protocol enhancements and applications exploiting 'benign' operations to bypass filtering. Moreover, DPI offers little benefit when the traffic itself is encrypted (SSH, SSL, etc.). While work has been conducted on how to infer applications types despite encryption [28], the potential for widespread usage of encryption with IPv6 is problematic.

On the commercial side, numerous solutions exist across the entirety of the deployment complexity spectrum. Enhanced firewalls provide normal firewall rules with additional options for consideration of applications and for detecting changes to the application itself (Windows XP Firewall, ZoneAlarm). Management software such as Microsoft SMS (Server Management System) and others allows for management of the distributed policies. However as noted earlier, these tools can make security-based connectivity issues difficult to detect and offer little in the manner of validation or visualization of the network itself.

With regards to the high end, we note several prominent solutions including Cisco Security Agents, Endforce, Consentry, Alterpoint, and Elemental Secrity. In a broad sense, the solutions can be divided into three different groups. The first group are based on signature databases whereby application network accesses are analyzed by a host agent and compared to the signature database for possibly exploits and security warnings. The second group employs signature analyses to search for common security holes such as buffer overflows with data logged for future analysis. The third group of solutions such as Cisco's NAC employ a mixture of network-level and host-level control with user authentication to control network security. While these solutions are quite powerful, the pervasive commercial solutions are often time consuming to configure and manage requiring significant IT investments to employ effectively. Hence, as noted in the introduction, deployment of such solutions has largely been limited to a minority of enterprise environments [1].

Finally, we note related work in the area of intrusion detection. Specifically, Lockdown does not attempt to fill the role of a host-based IDS (HIDS) [21], [29], [30]. Notably, several works noted the need for local context [31], [32] for better policing but did not focus on how to gather or analyze the information. Conversely, other IDS works [21], [29], [33] have described approaches for the aggregation of host-based IDS information for centralized analysis. In contrast to the often heavyweight nature of host-based intrusion detection systems and their respective data gathering, Lockdown focuses on maximumizing benefit with minimal cost. As a result, Lockdown trades effectiveness of mechanism for that decision (impact of compromised host) but offers vastly improved management to the network administrator with minimal deployment cost.

Hence, Lockdown is placed in the middle of the spectrum from Figure 20. Lockdown does not purport to offer signature analysis for detecting zero day exploits nor claim to offer Tripwire-like functionality for detecting root-level host compromises. Rather, Lockdown is complementary to the vast body of existing work with assistance to the mapping of policy to mechanism and the addition of a process to further refine policy via streamlined monitoring and auditing.

VII. Summary and Future Work

There is no final solution to computer security and network manageability, the best approach is typically a layered one. Lockdown complements existing layered solutions by seeking to improve the ease in which an administrator can manage the network and ascertain what is occurring among the managed hosts in a streamlined manner. The inclusion of local context significantly improves the expressiveness of both enforcement and observed behavior to better map and validate high level policy to mechanism. Notably, the ability of Lockdown to deftly balance between economy of mechanism and complexity of mechanism allows Lockdown to stay lightweight but yet offer benefit via streamlined management through visualization and analysis. The full application of the Lockdown process offers a cycle of enforcing, auditing, and analyzing that creates a closed loop of manageability for a Lockdown deployed network without expensive restructuring. The analysis, auditing, and visualization tools help in unearthing user / application behavior among hosts within the enterprise network that previously was non-existent or buried in volumes of log data. Moreover, the components of Lockdown are modular, offering the ability for partial adoption of the components (i.e. only monitoring) rather than requiring pervasive deployment as an initial start point.

Future work on improving Lockdown's auditing, anlysis, and enforcment is currently split into several areas. First, we are in the process of developing a GUI/front-end tool

that enables on-demand exploration of the network data. The tool will allow for visual browsing of the data, i.e. following interesting graphs through various perspectives of the network. Second, we are expanding our agent coverage to robust implementations on Windows, Mac OS X, and Solaris. Third, we are exploring the potential for conveying local context during the connection setup phase and its implications for enhanced control and trust extensions.

ACKNOWLEDGMENT

The authors would like to thank Greg Allan for his work on the network visualization tools [13]. The authors would also like to thank Curt Freeland and James Rogers for their assistance with data gathering via the Lockdown monitor mechanism.

REFERENCES

[1] R. Richardson, "2007 csi computer crime and security survey," in *The 12th Annual Computer Crime and Security Survey*. Computer Security Institute, 2007.
[2] (2006). [Online]. Available: http://chrispederick.com/work/user-agent-switcher/
[3] J. Weeds, B. Keller, D. Weir, I. Wakeman, J. Rimmer, and T. Owen, "Natural language expression of user policies in pervasive computing environments," in *Proc. of OntoLex 2004 (LREC Workshop on Ontologies and Lexical Resources in Distributed Environments)*, 2004.
[4] E. S. Al-Shaer and H. H. Hamed, "Management and translation of filtering security policies," in *IEEE ICC*, 2003.
[5] A. Bandara, A. Kakas, E. Lupu, and A. Russo, "Using argumentation logic for firewall policy specification and analysis," in *IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM)*, Oct. 2006, pp. 185–196.
[6] A. El-Atawy, T. Samak, Z. Wali, E. Al-Shaer, S. Li, F. Lin, and C. Pham, "An automated framework for validating firewall policy enforcement," in *Proc. of IEEE Workshop on Policies for Distributed Systems and Networks (POLICY'07)*, Bologna, Italy, Jun. 2007.
[7] L. Zhao, A. Shimae, and H. Nagamochi, "Linear-tree rule structure for firewall optimization," in *Proceedings of Communications, Internet, and Information Technology*, 2007.
[8] "Condor software package," available at www.cs.wisc.edu/condor.
[9] T. Garfinkel, "Traps and pitfalls: Practical problems in system call interposition based security tools," in *Network and Distributed Systems Security*, 2003.
[10] C. Wright, C. Cowan, J. Morris, S. Smalley, and G. Kroah-Hartman, "Linux security module framework," in *Ottaw Linux Symposium*, 2002.
[11] (2007, September). [Online]. Available: http://kerneltrap.org/Linux/Pluggable_Security
[12] M. Quaritsch and T. Winkler, "Linux security modules enhancements: Module stacking framework and tcp state transition hooks for state-driven nids," in *Secure Information and Communication*, 2004.
[13] *Visualizing Network Dynamics*, January 2005.
[14] H. Hamed and E. Al-Shaer, "Taxonomy of conflicts in network security policies," *IEEE Communications Magazine*, pp. 134–141, Mar. 2006.
[15] E. Al-Shaer and H. Hamed, "Discovery of policy anomalies in distributed firewalls," in *IEEE INFOCOM*, Mar. 2004, pp. 2605– 2616.
[16] J. Guttman, "Filtering posture: Local enforcement for global policies," in *IEEE Symposium on Security and Privacy*, May 1997.
[17] Y. Bartal, A. Mayer, K. Nissim, and A. Wool, "Firmato: A novel firewall management toolkit," in *IEEE Symposium on Security and Privacy*, May 1999.
[18] S. Bellovin, "Distributed firewalls," *;login*, pp. 39–47, November 1999.
[19] "Snort: The open source network intrusion detection system," available at www.snort.org.
[20] M. Handley, V. Paxson, and C. Kreibich, "Network intrusion detection: Evasion, traffic normalization, and end-to-end protocol semantics," in *Proc. of USENIX Security Symposium*, Aug. 2001.
[21] P. Porras and P. Neumann, "EMERALD: Event monitoring enabling responses to anomalous live disturbances," in *Proc. of the 20th National Information Security Systems Conference*, Baltimore, Maryland, Oct. 1997, pp. 353–365.
[22] S. Axelsson, "Intrusion detection systems: A survey and taxonomoy," Chalmers University, Tech. Rep., 2000.
[23] R. Atkinson, "IP Authentication Header," *IETF RFC 1826*, Aug. 1995.
[24] R. Moskowitz and P. Nikander, "Host Identity Protocol (HIP) Architecture," *IETF RFC 4423*, May 2006.
[25] M. Casado, T. Garfinkel, A. Akella, M. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks," in *15th USENIX Security Symposium*, 2006.
[26] M. Casado, M. Freedman, J. Pettit, J.Luo, N. McKeown, , and S. Shenker, "Ethane: Taking control of the enterprise," in *SIGCOMM*, 2007.
[27] S. Ioannidis, A. D. Keromytis, S. M. Bellovin, and J. M. Smith, "Implementing a distributed firewall," in *ACM Conference on Computer and Communications Security*, 2000, pp. 190–199. [Online]. Available: citeseer.ist.psu.edu/ioannidis00implementing.html
[28] J. Horton and R. Safavi-Naini, "Detecting policy violations through traffic analysis," in *Proceedings of the 22nd Annual Computer Security Applications Conference*, 2006.
[29] E. H. Spafford and D. Zamboni, "Intrusion detection using autonomous agents," *Comput. Networks*, vol. 34, no. 4, pp. 547–570, 2000.
[30] R. Kemmerer and G. Vigna, "Hi-DRA: Intrusion Detection for Internet Security," *IEEE Proceedings*, vol. 93, no. 10, pp. 1848–1857, October 2005.
[31] M. Almgren and U. Lindqvist, "Application-integrated data collection for security monitoring," in *RAID '01: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*. London, UK: Springer-Verlag, 2001, pp. 22–36.
[32] M. G. Welz and A. Hutchison, "Interfacing trusted applications with intrusion detection systems," in *RAID '01: Proceedings of the 4th International Symposium on Recent Advances in Intrusion Detection*. London, UK: Springer-Verlag, 2001, pp. 37–53.
[33] H. Dreger, C. Kreibich, V. Paxson, and R. Sommer, "Enhancing the accuracy of network-based intrusion detection with host-based context," in *Conference on Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA)*, 2005.