WHITE PAPER

Security and openMosix

Securely deploying SSI cluster technology over untrusted networking infrastructure

CONTROL PAGE

Document Approvals

Approved for Publication:

Author Name: Ian Latter
8 December 2003

Document Control

Document Name: Security and openMosix; Securely deploying SSI cluster technology over untrusted networking infrastructure
Document ID: white paper - security and openmosix.doc-Release-1.2(257)
Distribution: Unrestricted Distribution
Status: Release
Disk File: C:\Documents and Settings_.NULL\Desktop\whitepaper\White Paper - Security and openMosix.doc

Version	Date	Release Information	Author/s
1.2	*08-Dec-03*	Release / Unrestricted Distribution	Ian Latter
1.1	*06-Dec-03*	Draft / Uncontrolled	Ian Latter
1.0	*05-Dec-03*	Draft / Uncontrolled	Ian Latter

Distribution

Version	Release to
1.2	Public Release
1.1	Macquarie University, Moshe Bar, Bruce Knox, Wim Vandersmissen, David Conran
1.0	Macquarie University, Moshe Bar, Bruce Knox, Wim Vandersmissen

Table of Contents

1 OVERVIEW 4

2 GOOD, BUSINESS SENSE. 5

2.1 WHAT IS AN "SSI"? 5
2.2 WHY WOULD I WANT AN SSI? 5
2.3 VISIBLE SSI DEPLOYMENT METHODOLOGIES 5
2.4 THE GREAT ROI HEIST 6
2.5 ALTERNATIVE DEPLOYMENT METHODOLOGY FOR SSI IN TODAY'S ENTERPRISE 6

3 COMMON, TECHNICAL SENSE. 8

3.1 HOW OPENMOSIX COMMUNICATES 8
3.2 HOW OPENMOSIX MAINTAINS ITS NODE DATABASE 8
3.3 HOW OMDISCD OPERATES 9

4 WHERE THE PROBLEMS ARISE 10

4.1 WHERE TO PLACE TRUST? 10
4.2 IMPLEMENTATION VULNERABILITY (NODE VULNERABILITY) 10
4.3 NETWORK AUTHENTICATION (CLUSTER NODE ADDITION) 11
4.4 NETWORK CONFIDENTIALITY AND INTEGRITY (CLUSTER COMMUNICATIONS) 11

5 MOVING FORWARD – MITIGATING THE RISKS 12

5.1 WHAT IS CHAOS? 12
5.2 WHAT IS CLUSTERKNOPPIX? 12
5.3 REDESIGNING THE AUTO DISCOVERY PROCESS (TYD) 12
5.4 IPSEC (NETWORK LEVEL ENCRYPTION AND AUTHENTICATION) 14
5.5 ESTABLISHING AN "INSIDE" AND AN "OUTSIDE" (DEFINITION) 15
5.6 IMPLEMENTING "INSIDE" AND "OUTSIDE" (PACKET FILTERING) 17
5.7 USING A FULLY-MESHED VIRTUAL TOPOLOGY 19
5.8 PUTTING IT TOGETHER – ROLLING YOUR OWN 20

6 CONCLUSION 21

7 REFERENCES. 22

7.1 PAPERS / PRESENTATIONS 22
7.2 DISTRIBUTIONS 22
7.3 SOFTWARE 22
7.4 FEAR, UNCERTAINTY AND DOUBT 23

8 CONTACT 24

8.1 ADDITIONS, MODIFICATIONS AND DELETIONS 24
8.2 CONSULTATION 24

1 Overview

The pinnacle of success for any Single System Image (SSI) based cluster, should be the achievement of a global deployment on commodity x86 computing equipment; leveraging both the public communications infrastructure and existing capital equipment (personal computing) expenditure.

In this white paper we will explore the openMosix network architecture, network level risk mitigation techniques for the redeployment of organizational infrastructure in "open" clusters, a practical application of those techniques in the CHAOS and ClusterKnoppix Linux distributions, and proposals for extending both the security model and the flexibility of the openMosix architecture.

2 Good, Business Sense.

2.1 What is an "SSI"?

An SSI is an acronym for a "Single System Image". It is a term used to describe a type of cluster technology that aims to create a large system that transparently consists of an enumerate quantity of smaller systems; the infrastructure equivalent to a fractal. The alternative cluster technology is Beowulf, which involves a Parallel Computer Emulation software layer being applied to an enumerate quantity of computer systems.

The primary difference between the two technologies is the focus. Beowulf creates a powerful parallel-capable cluster, on which application developers can write applications specifically for the cluster. SSI concentrates on extending the functionality of a single computer out into the cluster in a way that allows existing applications to operate without modification, but allowing them to take advantage of the additional resources as though they were local to a single host.

Note that while openMosix is not a true SSI, its trust model does operate as an SSI would.

2.2 Why would I want an SSI?

A true Single System Image runs unmodified software, on component hosts (or nodes) that are treated as dynamic resources. In this way, an SSI can provide availability, scalability and manageability to shrink wrapped or black box software.

Imagine having the ability to manage ROI by dynamically allocating and recalling hardware resources that are assigned to a single "virtual server"? Or, imagine having the ability to manage TCO by consolidating hardware resources into live pools (clusters), ready for mission critical application deployment? Further, imagine being able to align infrastructure expenditure to budgetary cycles, by upgrading one third of a virtual asset, per annum, over three years (or one twelfth over 12 quarters)?

What economically competitive information-age organization would not want SSI technology?

2.3 Visible SSI Deployment Methodologies

Publicly, the typical cluster deployments have been by organizations with either permanent, heavy and dedicated processing interests, or organizations with very small and temporary interests (even down to those who are just having an inquisitive look-see).

Technologically, this translates to organizations either running dedicated computing hardware and other permanently allocated resources (usually racks of current Dell, Compaq/HP or IBM hardware maintained in larger organizations), or those who have back-ended existing resources to provide the SSI as a secondary function (desktop PCs running Linux with X-Windows that are also part of a "background" SSI in smaller organizations or organization silos).

Neither of these methodologies is appropriate for the modern enterprise. The latter is immature, introducing a number of additional information and resource management

issues into the personal desktop; not the least of which is the insurmountable task (cost, or irretrievably lost investment) that most organizations would face in replacing the Microsoft Windows desktop and SOE interface with another. The former fails to maximize an organization's existing capital computer investments while further damaging its ownership costs, by abandoning existing computing infrastructure, and introducing yet another environment that requires its own upgrade path and shiny new components.

This paper discusses a third deployment methodology that has not been well explored.

2.4 The Great ROI Heist

The modern enterprise has an abundance of processing power held captive in its PC desktop investment.

Organizations make a one to five year PC asset investment based on a processor intensive workforce that operates for 9hours per day, 5 days per week – or, a total of 45hrs/wk for an asset that is technically available for 168hrs/wk. To put it bluntly, the sum of any organization's PC investment strategy – its support, training, operational costs, process streamlining, systems management, etc ad nauseam – have all gone into acquiring and maintaining that PC asset for a 100% of its "useful" life, which amounts to a little more than 25% of its operational life.

So, while a given organization may present a 24x7 face, it has a massive and unused capability that is waiting for 8am tomorrow morning. Therefore, there remains an opportunity to increase the ROI of the enterprise PC asset by as much as 300%.

Stealing this untapped asset is the foundation of the third deployment methodology.

2.5 Alternative Deployment Methodology for SSI in Today's Enterprise

Recall that one of the fundamental advantages of an SSI is its ability to dynamically add and subtract system resources, to or from the cluster pool. An SSI can be as small as a single node (host) or as large as tens of thousands of nodes. However, it's the SSI's ability to dynamically "grow" and "shrink" that allows the enterprise to completely waive the cluster's (virtual server's) bounding parameters, by deploying the SSI cluster with a single dedicated PC resource – a home node.

By having the enterprise's existing and available PC resources shutdown and reboot to an SSI node configuration, at a scheduled time, the home node will appear to grow in capacity, to the scale of the number of PC resources that are added to the cluster, at that time.

Depending on the scale of the enterprise, this can have two benefits;

1. Always-On SSI Cluster: aka "The Follow-the-Sun Supercomputer"

   Many global organizations are already providing services on a follow-the-sun basis. Dividing the world into three time slices, their regionalization strategy allows them to leverage operational staff in their regular office-hours in their local time zone, each providing 8hours of global service.

   In the same way, the other two zones are outside of office hours – leaving their PC infrastructure free to dynamically support the designated SSI's. While

only the home node(s) will remain permanently allocated to cluster(s), in this model, the cluster(s) will be permanently enabled at a maximum capacity.

2. Batch Capable SSI Cluster: aka "The 5-9 Supercomputer"

   Organizations that exist in a single time zone will not be fortunate enough to support a permanently populated SSI's using their PC infrastructure. However, such organizations would be able to support batch-capable SSI's – able to populate them dynamically, during the non-office-hours time periods.

Unfortunately, this methodology of dynamically scaling an SSI from a single dedicated home node, out into the unused PC infrastructure of an organization, doesn't come without risk.

3 Common, Technical Sense.

3.1 How openMosix Communicates

openMosix is not a true SSI, but it does communicate as one would.

All of the openMosix communications are client-server oriented. The service ports are not well documented, but they can be found in the openMosix FAQ (online at http://howto.ipng.be/openMosixWiki/index.php/FAQ);

```
Moshe says;
   “openMosix uses only UDP protocol ports in the address
range of 5000-5700."

linux/include/linux/mfs_socket.h
  MFS_MAIN_PORT 0xD302    == 723  (TCP)

linux/hpc/comm.c
  MIG_DAEMON_PORT 0x3412  == 4660 (TCP)
  INFO_DAEMON_PORT 0x3415 == 5428 (UDP)
```

This translates to a set of filterable communications rules, like thus;

```
Client 1024:65535  – UDP –>  Server 5000:5700
Client 1024:65535  – TCP –>  Server 723
Client 1024:65535  – TCP –>  Server 4660
```

However, because one of the feature benefits of an SSI is its “single system” concept, there is no such thing as a “slave” or a “master” node; each node in the cluster essentially operating as peer. Unfortunately for the communications architecture, this means that every node in the cluster must be able to communicate directly with every other node in the cluster, with the same set of communications rules. This architecture effectively rules out any type of dynamic NAT or “cluster gateway” concepts.

Some or all of the services listed above will be available, depending on the compile-time options used to create the openMosix-enabled kernel that is being executed.

The only native security applied to these services is a single compile-time option, which if enabled, will prevent local or foreign connections to these services by hosts not listed in the local node database. It is not clear at what level (or layer) this filtering occurs, or by what means packets/connections are filtered.

3.2 How openMosix maintains its Node Database

The openMosix kernel works with a cluster “map”. This map can account for up to 65,535 nodes, via a construction of up to 255 objects – each object containing an identifier, a base address and a count (of contiguously addressed hosts from the base).

The same map structure used in the Linux openMosix kernel is also used in the ascii file /etc/openmosix.map.

An example of this map file would be;

```
#
# Example /etc/openmosix.map
#
# 8 nodes; 192.168.1.10-14 and 192.168.1.24-26
#
# id     base             count
1        192.168.1.10     5
2        192.168.1.24     3
```

Natively, openMosix does not maintain this map itself. Instead, this is left to the operating system and the tools provided. It can be either a static management scheme via /etc/openmosix.map and "setpe" or a dynamic one via the auto discovery daemon "omdiscd".

What is important to note, is that any device in this map, is implicitly "trusted" by openMosix, and automatically assumed to be included into the SSI/cluster. Note, too, that once a host is entered into the kernel's map, the kernel will attempt to communicate with the new node.

3.3 How omdiscd Operates

The openMosix auto discovery daemon is omdiscd, and is distributed with the openMosix utilities. It works by sending a continuous stream of multicast "add me" messages, while listening for the same.

The address/group and port can be found in the omdiscd source (net.h);

```
#ifndef TESTING
#define NET_MCAST_GROUP "239.192.0.1"
#else
#define NET_MCAST_GROUP "239.192.0.2"
#endif
#define NET_MCAST_PORT 1334
```

As it finds nodes advertising on that multicast group, omdiscd reads in the kernel's openMosix map, adds the new node, and writes the map back into the kernel. In this way, it is virtually identical to the manual alteration of the /etc/openmosix.map file.

While the openMosix map uses arbitrary identifiers, the omdiscd process (which needs a more predictable assignment method) takes the last two octets of the base address, and uses that to assign a "unique" node id.

4 Where the Problems Arise

4.1 Where to Place Trust?

In questioning or addressing the security issues in and around an SSI, the first thing that must be understood is the trust relationships.

An SSI blurs the traditional boundaries between “inside” and “outside”. Inside ends up being the relationship between all nodes in the kernel’s openMosix map, and outside is everything else.

Before considering any other issue, be aware that applying this concept to the bigger picture (i.e. a context of the enterprise network, or even the Internet), assures that the inside becomes a complex set of trust relationships that must be established amongst a haze of anti-trust.

4.2 Implementation Vulnerability (Node Vulnerability)

It is well documented, within the Internet security community, that openMosix is not safe to expose to the untrusted (public) network;

```
Date Reported:      04/23/2002
Brief Description:  MOSIX malformed packet denial of service
Risk Factor:        Low
Attack Type:        Network Based
Platforms:          Linux kernel All versions, MOSIX 1.5.7,
                    openMosix 2.4.17
Vulnerability:      mosix-malformed-packet-dos
X-Force URL:
           http://www.iss.net/security_center/static/8927.php
```

In return, the openMosix developers have made their position equally clear;

```
Date: 2002-04-26 10:49
Sender: mosheb
Logged In: YES
user_id=6632

This is not a bug, but a feature of openMosix. As a
principle security has to be handled at the perimeter of
the cluster, not within the cluster.

Moshe
```

The denial of service attack listed here is still valid at the current release of openMosix (2.4.22-1 at the time of writing).

Similarly, incompatibilities between openMosix Linux distributions have seen hosts of the same kernel and openMosix versions consistently crashing one another. This was later attributed to differing compile-time options; which, while “solved”, demonstrates that a number of critical vulnerabilities capable of corrupting the running Linux kernel, still exist.

4.3 Network Authentication (Cluster Node Addition)

The only practical way to manage the deployment of a large and dynamic SSI is via auto discovery. However, omdiscd has two limitations that greatly impact upon the risk and scalability of the SSI, in the enterprise;

1. There is no authentication mechanism utilised by omdiscd, prior to adding a host to the kernel map. It is, therefore, a basic exercise to add an unauthorised IP address into the cluster trust relationships. This adds an intolerable risk to a processing device that is expected to handle an organization's critical information assets.
2. While not strictly a network authentication issue, but an issue that impacts upon auto discovery; omdiscd's use of multicast traffic creates two problems. In the outset, most enterprises are not immediately equipped to route multicast traffic, tending to limit auto discovery to a single LAN or VLAN. Additionally, as will be seen in the next section, not communicating with each node individually, at this early point, makes it difficult to add adequate security improvements later.

Both of these "features" are implemented by design. Unfortunately, neither is appropriate for the modern enterprise.

4.4 Network Confidentiality and Integrity (Cluster Communications)

As cluster network communications are really the stuff of dreams that are usually found on the buses and backplanes of ordinary server systems, it is a fundamental business requirement that they are very well protected in transit.

openMosix makes no effort to encrypt or validate data transferred between nodes. And, relatively speaking, this is understandable. As has been seen from the previous two issues, openMosix is openly designed without integrity checking, authentication or data privacy protection in any area of its operation.

While it has not been publicly proven, theoretically, data and data fragments can be sniffed and stolen from the wire, intercepted and corrupted, injected or modified.

5 Moving Forward – Mitigating the Risks

5.1 What is CHAOS?

CHAOS is a CD or PXE based Linux and openMosix cluster distribution.

CHAOS is a 9Mbyte Linux distribution, fitting on a single business card sized CDROM. The tiny disc boots any i586 class PC (that supports CD or PXE booting) into a working openMosix node, without disturbing the contents of any local hard disk. Ideal for large-scale ad-hoc clusters, once booted, CHAOS runs entirely from memory. CHAOS aims to be the most compact, secure and straightforward openMosix cluster platform available.

CHAOS is the supercomputer for your wallet.

5.2 What is ClusterKnoppix?

ClusterKnoppix is a modified Knoppix distribution, utilizing the openMosix kernel.

Knoppix is a Linux distribution that boots and runs entirely from CD. It runs a complete Linux distribution, based on Debian, which includes recent software and desktop environments, with programs such as Abiword, The Gimp, Konqueror, Mozilla, and hundreds of other quality open source programs; compressed from 1.7Gbytes to fit on a 700Mbyte CD. Its default windows environment is KDE.

5.3 Redesigning the Auto Discovery Process (tyd)

The best place to begin to remedy the security issues identified in the previous section is in the auto discovery process.

Ideally, the auto discovery daemon should act as a gatekeeper for the cluster. It should connect to every node in the cluster, individually, when joining or leaving the cluster, and should accept or decline requests from other nodes doing the same. Not only does this circumvent the limited distribution of multicast (by utilizing unicast), but it also provides an opportunity to negotiate authentication and encryption transforms.

The authors of the CHAOS distribution created tyd and the “Terrence and Phillip” protocol to resolve these very issues. Note though, that tyd does not perform all of these tasks itself.

tyd uses unicast to connect to each node in the cluster, to negotiate node additions and removals. This is done through the Terrence and Phillip protocol, where Terrence is a client and Phillip is a server. Phillip has been designed with the arbitrary port allocation of 3278/TCP – enabling a communications rule such as;

```
Client 1024:65535  – TCP –>  Server 3278
```

tyd starts as a Terrence. It connects to any node in the cluster (one that it is told about), and retrieves a copy of the openMosix map, from that node. Figure 1 demonstrates the Terrence and Phillip protocol in practice, with the new host requesting a copy of the cluster map from a given node in the cluster.

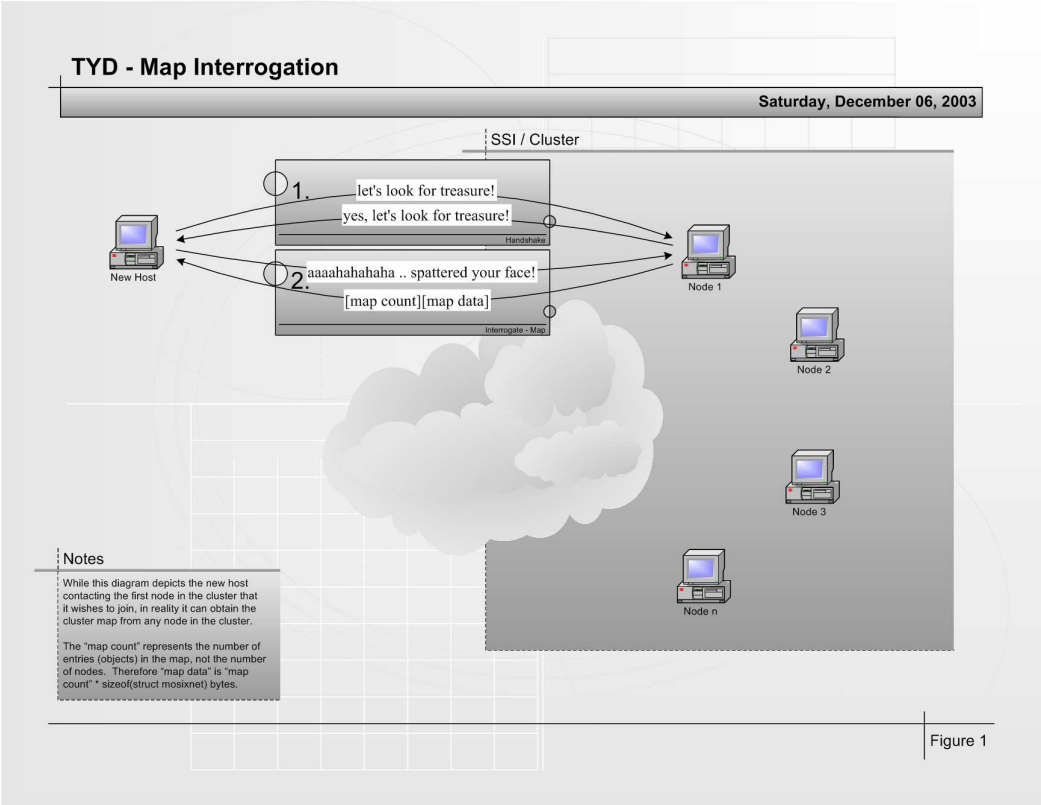


Figure 1

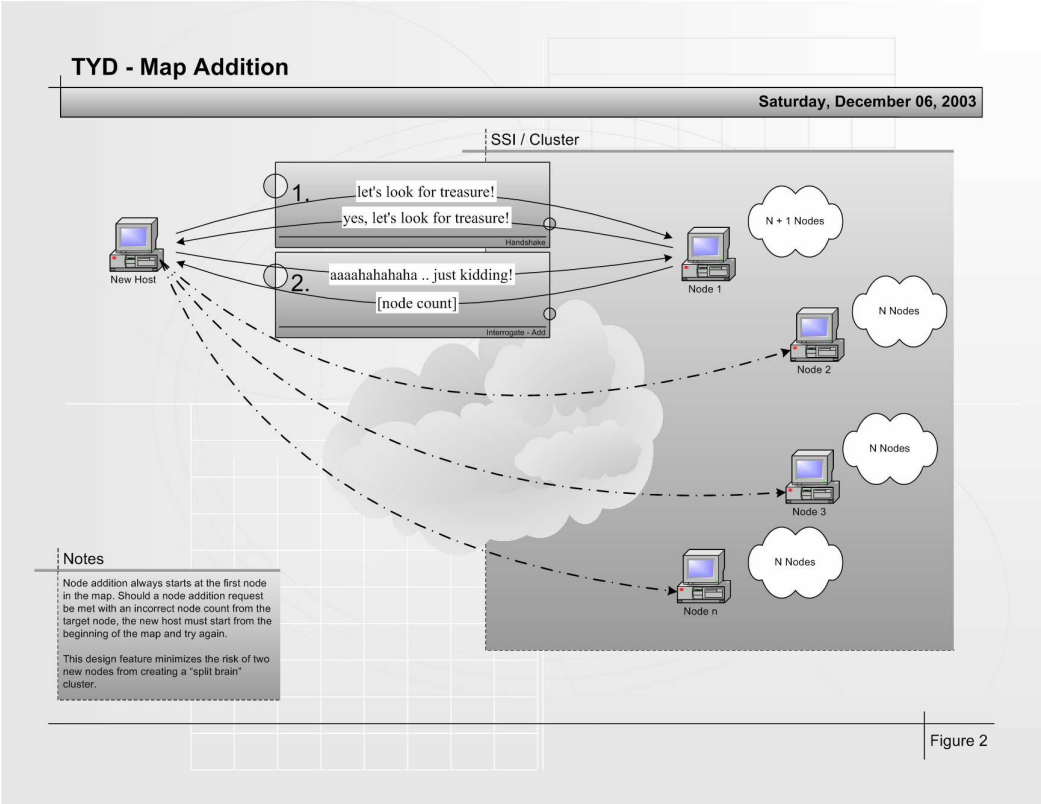


Figure 2

Once tyd has successfully acquired a copy of the map as Terrence, it then continues on to finish Terrence's work, by connecting to each node in the cluster database, asking to be added to the cluster.

Figure 2 shows how the new host requests to be added to the cluster, node by node. In the diagram, Node 1 has already added the new host, allowing the new host to move on to the next, and each, consecutive node until it has exhausted the cluster map.

After tyd has successfully added its node to the last node in the cluster database, tyd becomes Phillip. Phillip is completely passive; listening for new nodes (via their Terrence connections), serving the cluster database, and adding nodes as requested.

However, tyd does not negotiate encryption, authentication or any other security transforms. The Terrence and Phillip protocol makes no allowance for node authentication – and on its own, provides no additional value outside of unicast connectivity, over the existing omdiscd auto discovery process.

Instead, "per-host" hooks have been applied to sneakily offload these additional functions to other best-practice security components – allowing tyd to perform security actions, as nodes require them.

This offloading has made it relatively easy for other distributions, such as ClusterKnoppix, to incorporate the new auto discovery daemon, and to adopt the improved security features without significant impact.

5.4 IPSEC (Network Level Encryption and Authentication)

IP security (IPSEC) is probably the world's most common Virtual Private Networking (VPN) solution. IPSEC supports a wide variety of authentication and encryption transforms, and, in its encapsulation mode, is capable of tunnelling all layer-3 (IP) traffic between two endpoints.

The Linux 2.4 kernel series can be compiled with the FreeSWAN project source to enable IPSEC in the kernel. Both CHAOS and ClusterKnoppix include the FreeSWAN package within their respective kernels – along side of openMosix.

tyd enables network level authentication and encryption by dynamically working with IPSEC in three key steps;

1. tyd hands a cluster-wide pre-shared key (PSK) off to IPSEC for authentication on all connections.
2. tyd constructs a default "road warrior" IPSEC configuration, for new inbound connections. This includes an authentication transform of the PSK (above), an encryption transform of Triple DES (3DES), enables dynamic re-keying through perfect forward security (PFS), and enables the compression transform.
3. tyd creates one new IPSEC tunnel per Terrence connection (of the same transforms as above), routes it and initiates it.

The net result is that no new node can begin to communicate with any existing cluster node (even for the purposes of auto discovery) unless the new node is correctly authenticated and appropriately encrypted. IPSEC goes some way to providing node

(host) integrity assurance through authorised network level access, and a long way to assuring data confidentiality and integrity through network level encryption.

As can be seen in this section alone, by creating a unicast auto discovery process, it becomes very easy to add network level security, on a per-host basis. And, more importantly, this security is not user dependant; the default solution being a secure solution makes it highly likely that any user deploying the solution will deploy it securely.

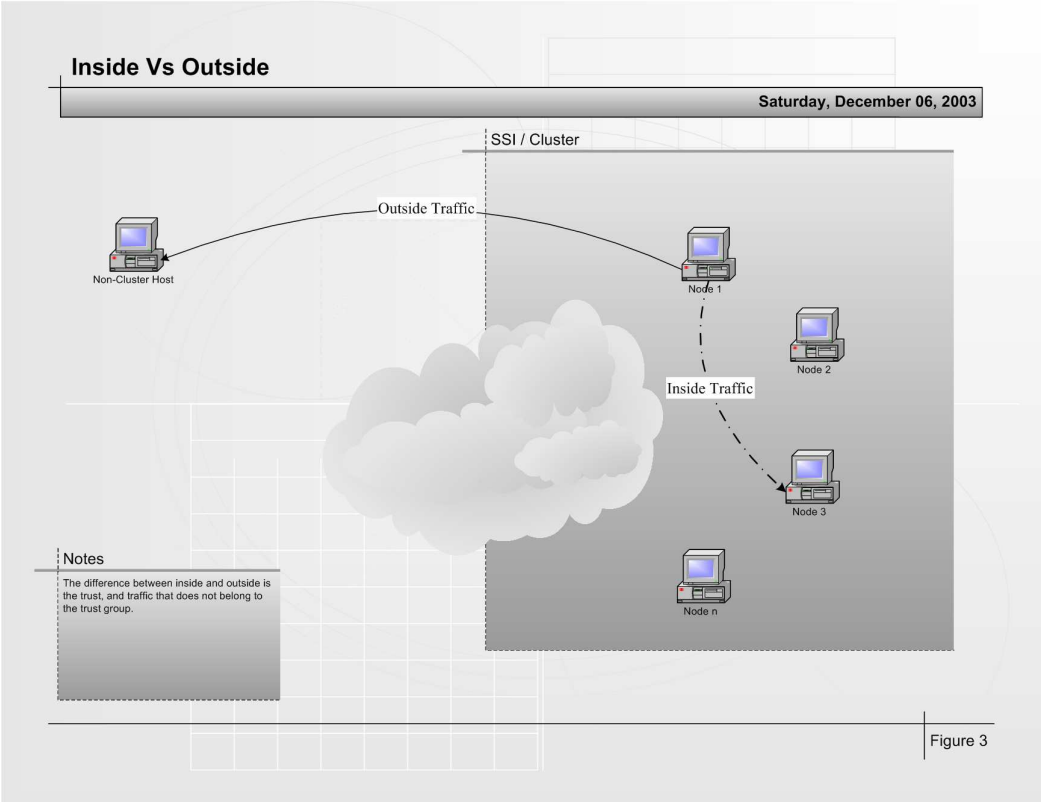
Note, too, that by "sealing off" (tunnelling) these inter-node communications, the first big step has been made in establishing "inside" versus "outside".

5.5 Establishing an "Inside" and an "Outside" (Definition)

While IPSEC has been used to create VPN tunnels between peers (nodes) in the cluster, there has still not been a clear separation between the inside and outside of the cluster.

What is inside? Recall that the node map (the cluster database) is a map of implicit trust relationships; and that those hosts in the map are inside the cluster, leaving all other hosts outside. The problem with this rule is that it is not defined specifically enough, and that it does not allow a safe entry point for untrusted hosts.

As can be seen in figure 3, hosts in the cluster communicate both "internally" – i.e. to other nodes in the cluster – and "externally" to other network connected devices. Therefore, not all communications that originate or terminate on cluster nodes are truly internal to the cluster as a whole.



The solution comes from section 3, where the cluster communications (or inter-node communications) are defined. Adding a second dimension to the definition, and exposing the real complication with the implicit trust relationships defined by openMosix, is this;

- Inside is all openMosix (cluster) traffic that occurs between nodes in the cluster map.
- Outside is everything else (including all non-openMosix traffic that occurs between nodes in the cluster map).

What does this mean in practice? That the nodes in the cluster must be configured such that only the known openMosix traffic is permitted between nodes in the cluster database, and that this traffic is routed via the authenticated and encrypted VPN tunnels;

```
Client 1024:65535  – TCP –>  Server 3278
Client 1024:65535  – UDP –>  Server 5000:5700
Client 1024:65535  – TCP –>  Server 723
Client 1024:65535  – TCP –>  Server 4660
```

Furthermore, enabling the gate-keeping functionality of the auto discovery process, an "air lock" style entry process can be created to allow new (untrusted) hosts into the cluster safely;

1. The new host must establish a valid IPSEC tunnel to the node it seeks to interrogate. Without this authentication, the new host remains untrusted, and unable to acquire network level access to the cluster internals (including auto discovery). This is a fundamental requirement for maintaining cluster integrity.
2. Only once a valid IPSEC tunnel is established, should nodes then negotiate the addition of the new host, into the cluster database. This second step is weak, as tyd provides no native authentication. However, even without this extension, the prerequisite of a negotiated entry into the cluster database above and beyond layer-4 connectivity, adds defence in depth; the second round of negotiations forcing any new host to completely satisfy entry requirements, minimising abuse (even from nodes that may have had their PSK compromised).

This process allows a successful host to first enter the internal network, gain visibility to the gatekeeper, and then to negotiate its place in openMosix's implicit trust relationship database.

By the same token, unsuccessful or unwanted hosts are left so far outside of the cluster internals, that they no longer have the ability to intercept, corrupt, inject or modify SSI backplane traffic, or critical SSI services.

5.6 Implementing "Inside" and "Outside" (Packet Filtering)

The practical application of this delineation is straight forward.

The Linux 2.4 kernel series incorporates the netfilter packet filtering engine. This engine is relatively sophisticated, incorporating layer-4 connection tracking and a mass of other features that are not required for this work.

For the application of this security suite, a host with a single interface (eth0) will be assumed. The FreeSWAN IPSEC software associates a virtual IPSEC interface with each physical interface – so an interface of ipsec0 will also be assumed. Every IPSEC tunnel to or from this physical interface will route via the virtual interface. This essentially means that the unencrypted traffic will enter and leave the local host via ipsec0 and the "outside of the tunnel" (encrypted traffic) will enter and leave via eth0.

By definition, the eth0 traffic, then, should only be IPSEC traffic;

```
Client 500  – UDP –>  Server 500
Client      – ESP –>  Server
```

Whereas ipsec0 should see all of the inter-node communications;

```
Client 1024:65535  – TCP –>  Server 3278
Client 1024:65535  – UDP –>  Server 5000:5700
Client 1024:65535  – TCP –>  Server 723
Client 1024:65535  – TCP –>  Server 4660
```

Included here is a dump of the rules that were derived from above, as executed via iptables from tyd, the auto discovery process.

1. First, flush the netfilter environment and add best practice rules;

```
-F
-F -t mangle
-P INPUT DROP
-P OUTPUT DROP
-A INPUT -p tcp ! --tcp-flags SYN,RST,ACK SYN -m state
         --state NEW -j DROP
-A OUTPUT -p tcp ! --tcp-flags SYN,RST,ACK SYN -m state
          --state NEW -j DROP
```

2. Add eth0 rules for IPSEC;

```
-A INPUT -i eth0 -p 50
          -s 0.0.0.0/0 -d 192.168.1.87/32
          -j ACCEPT
-A OUTPUT -o eth0 -p 50
          -s 192.168.1.87/32 -d 0.0.0.0/0
          -j ACCEPT
-A OUTPUT -o eth0 -p udp
          -s 192.168.1.87/32 --sport 500
          -d 0.0.0.0/0 --dport 500
          -j ACCEPT
-A INPUT -i eth0 -p udp
          -s 0.0.0.0/0 --sport 500
          -d 192.168.1.87/32 --dport 500
          -j ACCEPT
-A OUTPUT -o ipsec0 -p udp
          -s 192.168.1.87/32 --sport 500
```



```
          -d 0.0.0.0/0 --dport 500
          -j ACCEPT
-A INPUT -i ipsec0 -p udp
          -s 0.0.0.0/0 --sport 500
          -d 192.168.1.87/32 --dport 500
          -j ACCEPT
```

Note the exception; that FreeSWAN's IKE insists on transmitting via ipsec0.

3. Add ipsec0 rules for openMosix inter node communications;

```
-A INPUT -i ipsec0 -p tcp
          -s 0.0.0.0/0 --sport 1024:65535
          -d 192.168.1.87/32 --dport 3278
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p tcp
          -d 0.0.0.0/0 --dport 1024:65535
          -s 192.168.1.87/32 --sport 3278
          -m state --state ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p tcp
          -s 192.168.1.87/32 --sport 1024:65535
          -d 0.0.0.0/0 --dport 3278
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p tcp
          -d 192.168.1.87/32 --dport 1024:65535
          -s 0.0.0.0/0 --sport 3278
          -m state --state ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p udp
          -s 0.0.0.0/0 --sport 1024:65535
          -d 192.168.1.87/32 --dport 5000:5700
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p udp
          -d 0.0.0.0/0 --dport 1024:65535
          -s 192.168.1.87/32 --sport 5000:5700
          -m state --state ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p udp
          -s 192.168.1.87/32 --sport 1024:65535
          -d 0.0.0.0/0 --dport 5000:5700
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p udp
          -d 192.168.1.87/32 --dport 1024:65535
          -s 0.0.0.0/0 --sport 5000:5700
          -m state --state ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p tcp
          -s 0.0.0.0/0 --sport 1024:65535
          -d 192.168.1.87/32 --dport 723
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p tcp
          -d 0.0.0.0/0 --dport 1024:65535
          -s 192.168.1.87/32 --sport 723
          -m state --state ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p tcp
          -s 192.168.1.87/32 --sport 1024:65535
          -d 0.0.0.0/0 --dport 723
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p tcp
          -d 192.168.1.87/32 --dport 1024:65535
          -s 0.0.0.0/0 --sport 723
          -m state --state ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p tcp
          -s 0.0.0.0/0 --sport 1024:65535
          -d 192.168.1.87/32 --dport 4660
```



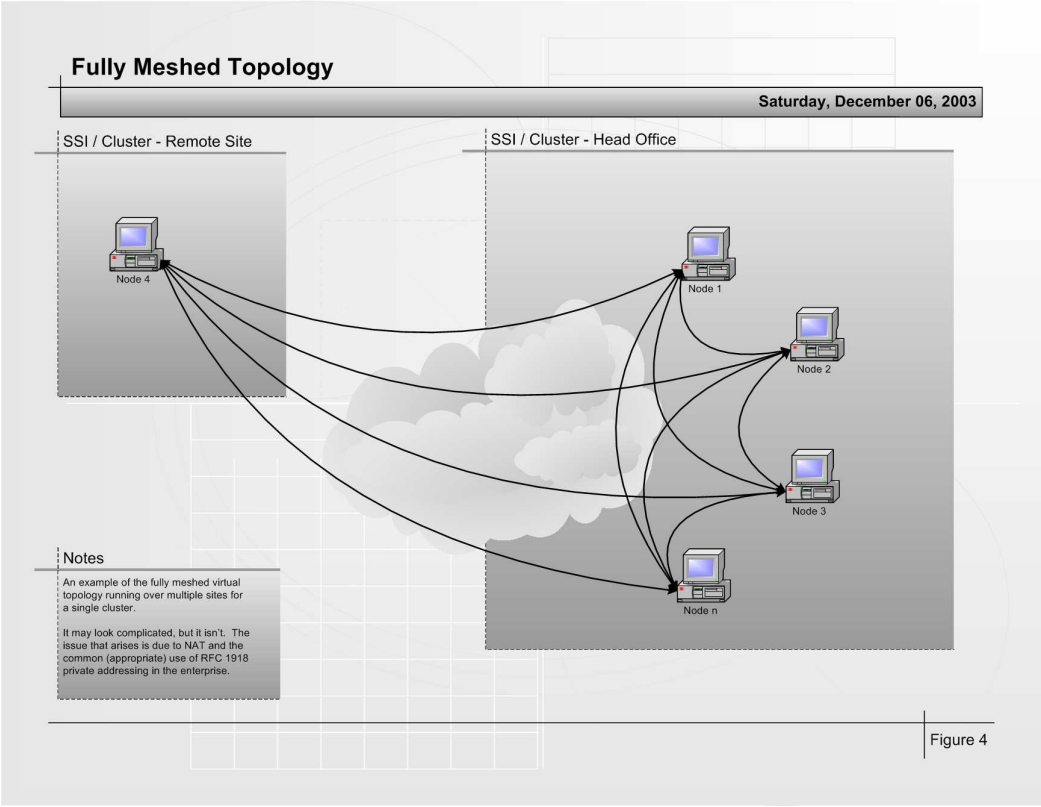
```
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p tcp
          -d 0.0.0.0/0 --dport 1024:65535
          -s 192.168.1.87/32 --sport 4660
          -m state --state ESTABLISHED -j ACCEPT
-A OUTPUT -o ipsec0 -p tcp
          -s 192.168.1.87/32 --sport 1024:65535
          -d 0.0.0.0/0 --dport 4660
          -m state --state NEW,ESTABLISHED -j ACCEPT
-A INPUT -i ipsec0 -p tcp
          -d 192.168.1.87/32 --dport 1024:65535
          -s 0.0.0.0/0 --sport 4660
          -m state --state ESTABLISHED -j ACCEPT
```

For further examples and rule set specifics, download, run and observe the CHAOS distribution in action.

5.7 Using a Fully-Meshed Virtual Topology

The native underlying communications architecture, utilised by openMosix, is a fully meshed topology. What this means is that while nodes may reside on differing subnets (or even differing continents within the public infrastructure), openMosix requires the ability to directly communicate from any node, to any node, on its regular communications services.

Figure 4 demonstrates this topology;



Typically, this isn't an issue. However, one of the problems that will arise in enterprise-wide SSI deployments, particularly those that are ad-hoc, will be the intervention required due to private (RFC 1918) addressing, and Network Address Translation (NAT)-based security solutions (such as CISCO's PIX).

Without the security methodology recommended in this paper, openMosix will assume that it can directly connect to the IP address that it has in its cluster database. This could be made to work, providing one of a number of static NAT options is available at the problem site.

With the security methodology recommended in this paper, the problem still exists, but an option to provide an additional layer of abstraction may exist also. It should be possible to extend both the operating Linux kernel, and tyd, to incorporate the Layer-2 Tunnelling Protocol (L2TP). L2TP could be used to provide a virtual class-b network – 65,535 addresses that would be RFC 1918 compliant – inside/on top of the IPSEC tunnel mesh. However, for the illusion to be complete, the auto discovery process would need to see what was going on behind the curtains (at least on the local node) so that it could facilitate the mirage.

Needless to say, that this problem has not yet been encountered, or resolved, within the community; but expect it to be there.

5.8 Putting it together – Rolling your own

The CHAOS distribution was designed to be elegant, simple and secure – to provide the security solutions discussed in this paper for ad-hoc enterprise-wide SSI clusters. Originally, CHAOS focused on CD distribution as the major deployment method, but quickly migrated toward network capable booting methods (such as PXE), vastly improving the deployment capabilities of the distribution, both in terms of automation and flexibility.

Enterprise roll-outs are likely to be deployed by utilising existing desktop and SOE management solutions, like IBM's "Remote Deployment Manager (RDM), Intel's "LANDesk", or Novell's "ZENworks",

However, for those interested only in *experimenting* with the considerations proposed in this paper, a better solution exists.

The ClusterKnoppix distribution has adopted the security solutions discussed in this paper, by compiling-in the additional Linux kernel resources and adding tyd as an alternate auto discovery process. ClusterKnoppix works well as a "home node"; in stark contrast to CHAOS, it is fully featured and very user friendly – including a self configuring X-Windows desktop.

With the amount of flexibility available in the SSI deployment methodology(s), any limitation in the practical deployment will probably stem from a given organization's operating environment, management tools or technical expertise. Modern equipment with current management tools and practices should yield success.

6 Conclusion

This white paper demonstrates the inherent risks that come from the SSI architecture, its dependencies and trust relationships. So too, does it show how applying existing best practice security methodologies, logically, to the SSI architecture, can mitigate these risks.

Single System Images are still young. Their greatest user base is founded firmly in the academic and research arenas; two institutions that are, traditionally, not security focused. By providing good quality, open source tools, such as CHAOS and ClusterKnoppix, these institutions will adopt good security practices unconsciously.

By making the security features second nature to the current user base, commercial organizations – including multinational enterprise – looking to deploy SSI clusters in untrusted enterprise networks, should be able to readily adapt and deploy SSI technology when the SSI technology, itself, is ripe for the picking; rather than having to wait for commercially sound practices to be applied to a technology that was developed for dedicated and isolated (trusted) deployments.

Single System Images will revolutionize application deployment – it is hoped that this paper has gone some distance to providing integrity and confidentiality assurance for those organizations willing to take the first steps.

7 References

7.1 Papers / Presentations

7.1.1 HPC Computing Applied to Business Applications

http://openmosix.sourceforge.net/Business_Applications_2003.pdf

Moshe Bar, 2003.

7.1.2 Turning a group of independent GNU/Linux workstations into an (Open)Mosix cluster in an untrusted networking environment and surviving the experience

http://www.democritos.it/events/openMosix/papers/cagliari.pdf

Giacomo Mulas, November 2002.

7.1.3 The El-Cheap-o Massively-Parallel Diskless After-5pm Super-Computer

http://office.sage-au.org.au/member-only/videos/sage-qld-1999-jun.avi

David Conran, June 1999.

7.2 Distributions

7.2.1 CHAOS

http://itsecurity.mq.edu.au/chaos/

7.2.2 ClusterKnoppix

http://bofh.be/clusterknoppix/

7.3 Software

7.3.1 FreeSWAN

http://www.freeswan.org/

7.3.2 Netfilter

http://www.netfilter.org/

7.3.3 openMosix

http://www.openmosix.org/

7.3.4 openSSI

http://www.openssi.org/

7.4 Fear, Uncertainty and Doubt

7.4.1 ISS: Security Center: X-Force Alerts and Advisories
http://xforce.iss.net/xforce/alerts/id/AS02-17
Internet Security Systems, April 2002.

7.4.2 Know Your Enemy: Statistics
http://project.honeynet.org/papers/stats/
HoneyNet Project, July 2001.

8 Contact

8.1 Additions, Modifications and Deletions

For changes to this document, please refer to the author and revision history blocks in the control page. Please report errors or omissions to the author.

8.2 Consultation

If you would like to discuss SSI security architecture or other concepts related to this white paper, then please contact the author;

Ian Latter

IT Security Officer

Macquarie University, Australia.

Email: Ian.Latter@mq.edu.au

