

Multilevel Filesystems in Solaris Trusted Extensions

Glenn Faden
Sun Microsystems
17 Network Circle
Menlo Park, CA, USA
1-650-786-4003

glenn.faden@sun.com

ABSTRACT

Multilevel security is typically implemented by assigning fine-grained security contexts, such as sensitivity labels to all subjects and objects. These extended security contexts require modifications to standard filesystems, and interfaces that affect system throughput and application compatibility. This trade-off between policy enforcement and performance tends to marginalize these systems to special-purpose environments. This paper describes a light-weight approach which avoids the requirement for customized filesystems or modified applications. Instead, the system is partitioned into labeled zones. Subjects and objects are associated with these zones from which they inherit their sensitivity labels. This structured approach to data separation makes it possible to implement mandatory access control on a mainstream operating system.

Categories and Subject Descriptors

D.4.3[Operating Systems]: File Systems Management, - *file organization*

D.4.4[Operating Systems]: Security and Protection – *access controls*

General Terms

Design, Security

Keywords

multilevel security, mandatory access control, file labeling

1. INTRODUCTION

In both corporate and government organizations, policies exist to limit access to information. A first step in establishing any access control policy is to evaluate the risks that are inherent in unauthorized disclosure of various types of data. Data labeling has been used for decades[12] to specify the sensitivity of information, and the risk of its disclosure. Access control policies can be expressed using these sensitivity labels to specify valid data flows. For example, a policy may state that data marked *Confidential* should not be distributed outside the organization.

When these policies are advisory in nature, they are referred to as Discretion Access Control (DAC). However, when such policies are enforced automatically and cannot be circumvented, they represent Mandatory Access Control (MAC). Two examples of MAC policies are Multilevel Security (MLS) and Domain Type Enforcement (TE)[5]. Both approaches use labels to partition objects and to define the rules for the flow of data.

Systems that implement MLS and TE have historically done so by attaching security contexts to all of the resources which they protect (objects) and to the users or processes which act on them (subjects). These security contexts are used by policy managers to determine whether a given subject has access to the objects. For example, when one label dominates another, data at the lower level can be read by subjects at the higher-level.

In order to uniquely label each object, storage must be allocated to retain the labels, and interfaces must be developed to maintain the label associations. Since these labels are used for mandatory access control, they must be rigorously protected from unauthorized modification. As required by the mandatory access policy, even the owner of the file is typically not authorized to change these labels. Only a special administrative role, such as an information security officer, is authorized to reclassify labeled data. Protecting a file's security context is at least as important as protecting the actual data in the file. This approach tends to require customized filesystems since standard filesystems don't provide the necessary protection model.

Modern commercial filesystems implement some kind of DAC policy which allows the subject who created the file (the owner) to restrict access to the file. But these filesystems don't have a notion of security beyond that associated with ownership. Therefore, operating systems that implement MAC have generally provided customized filesystems to support mandatory security contexts. For example, the Trusted Solaris™ OS extended the inodes of its UFS filesystem to support labels and other attributes[6, 11]. SELinux uses Extended Attributes (EA) for security contexts. According to J. Morris[7], for a filesystem to support SELinux security context labels, it needs EA support and a handler for the EA security namespace. Such filesystems currently include ext3, ext2, XFS and ReiserFS.

This paper describes a new approach in which a mandatory MLS policy is applied automatically, so that it can be used with any existing filesystem. This transparent approach has recently been integrated into the Solaris 10 OS and its open source counterpart, OpenSolaris[10]. The new MLS policy is enforced by enabling

Solaris Trusted Extensions[4]. The motivation for this transparent approach was to provide a single commercial OS with an optional MLS policy, that would support the full range of applications, patches, and hardware platforms.

2. GRANULARITY OF THE POLICY

A successful security policy should do more than prevent unauthorized access. It should be unobtrusive and straight-forward to minimize user frustration and confusion. Excessive complexity should be avoided in favor of strategies that are intuitive to users. Previous MLS systems have had limited success because their policies were invasive and incompatible with existing applications and procedures. Therefore it is worth reconsidering the benefits of fine-grained versus coarse-grained MLS policies.

Operating systems usually protect filesystem objects at the granularity of a single file. More fine-grained controls have been implemented in services such as multilevel versions of the X11 Window System [2] and per-row protections in Oracle Label Security [8], but these extensions are not interpreted by the OS kernel.

On the other hand, a less fine-grained approach makes sense for structured multilevel data. Although it may seem obvious that a MLS policy should be implemented at the same granularity as the traditional DAC policy, the protection policies for multilevel data involve special procedures and conventions that are distinct from unclassified data policies. For example, a user should not be able to determine that higher-level information even exists unless the user is cleared to read that data.

Just as in the physical world, sensitive digital information requires special handling procedures. For example, a folder marked Secret should not contain Top Secret data. The hierarchy of data classifications should be reflected in the organization of the data. Therefore, a more appropriate granularity for an OS-enforced MLS policy is at a directory (folder) level. But this model can be further optimized to handle just those directories whose labels differ from their parent directory. We can distinguish these special directories by making them the root nodes of mounted filesystems. In this way, only directories corresponding to mounted filesystems need to be explicitly validated by the MLS policy. All other directories and files can be labeled implicitly as an attribute of the mounted filesystem in which they exist.

3. FILESYSTEMS AND NAMESPACES

In the Solaris OS, new filesystems can be mounted over any arbitrary directory. In fact, filesystems can be mounted recursively on top of other filesystems with arbitrary complexity. However, there are a variety of restrictions which apply to the mount points. For example, only privileged users may mount (or unmount) filesystems. In this way the set of available pathnames is strictly controlled. We refer to the set of available pathnames as the file namespace, or simply *namespace*.

The Solaris kernel maintains an in-core table of mounted filesystem that is used to lookup pathname components which correspond to mounted filesystems. Some mounted filesystems correspond to actual mass storage, and some are simply aliases to directories in other filesystems. The term *filesystem* refers to both

on-disk formats and virtual types. For example, Personal Computer File System (PCFS), UNIX File System (UFS), and Zettabyte File System (ZFS) are all on-disk formats, whereas the Loopback File System (LOFS), Network File System (NFS), and Common Internet File System (CIFS), are mappings to other filesystems.

3.1 The Zettabyte File System

The Zettabyte File System [14] provides several new abilities which facilitate filesystem labeling. It provides Copy-on-Write semantics (COW) which enables almost instantaneous cloning of filesystem snapshots. It also allows an arbitrary number of filesystems to be created from an arbitrary number of pooled disks. The filesystems, known as *data sets* in ZFS parlance, include an attribute identifying their corresponding directories, and are automatically mounted as needed. This makes it convenient to use large number of data sets, such as associating a unique data set with each user's home directory.

3.2 The Loopback File System

The Loopback File System (called *bindfs* in Linux) is a special case in which a portion of an existing filesystem can be exposed with a new name and protection context within a different filesystem. For example, a portions of a filesystem can be exposed as read-only in a new context, even though it is writable in the original context. This concept will be explored in section 7, Labeled File Sharing.

3.3 The Network File System

Finally, the Network File System makes it possible to share or export arbitrary directories from local filesystems, such as ZFS data sets, so that they are remotely accessible. By enforcing the MLS policy on both trusted NFS clients and trusted NFS servers the MLS policy can be extended to apply to heterogeneous environments of trusted and untrusted clients and servers.

Together, these three filesystems facilitate the implementation of labeled filesystems without explicit labeled attributes. Although a wide variety of filesystems are supported, we will focus on these three filesystems in the remainder of the paper.

4. LABELS AND PATHNAMES

In order to implement a file MLS policy, the kernel must be able to unambiguously determine the sensitivity label of any file from its pathname. Since the Solaris kernel can identify the mounted filesystem corresponding to any pathname, this can be extended to determine the sensitivity label of any mounted filesystem.

However, this is not as simple as it may seem. A large Solaris system using network storage may have thousands of mount points. New filesystems, such as ZFS, reduce the distinction between directories and mount points. So a reliable and automated mechanism is essential to ensure that the mounted filesystems are labeled correctly and consistently.

Solaris Trusted Extensions eliminates the need to explicitly associate labels with mounted filesystems. Instead, the kernel derives the label of each mount point implicitly from other attributes. The system does not provide any interface for directly associating a label with a mount point, thereby eliminating an administrative burden and a source of human error.

5. LABELED ZONES

A unique namespace is established for each sensitivity label that is defined on the system. The namespace and its default label are tied together in a containment mechanism known as a *zone* [16]. A zone is a virtualized environment in which related applications run, and appear to those processes to be a complete operating system. However, each of these zones is just an isolated labeled execution environment sharing a single Solaris kernel. The isolation between labeled zones applies to all system services including networking, the window system, and interprocess communication. By default, no communication is possible between processes in differently labeled zones. However, the implementation is transparent to applications since they appear to be running in separate OS instances.

Although any two zones have a label relationship, the zones are more isolated than is normally required by an MLS policy. For example, the ability for a higher-level subject to read lower-level objects is not provided by default. Processes in higher-level zones can't observe lower-level processes, and lower-level files are not visible unless explicitly shared.

5.1 The Global Zone

A special global zone is used to administer these labeled zones, and to establish the data flow policies between them. From an MLS perspective, the global zone is a multilevel environment. Its processes run at the highest label and dominate all other labels. The global zone is inaccessible to normal users; restricted access is provided to administrative roles via the Role-Based Access Control [3]Faden, Glenn Faden, Glenn Faden, Glenn Faden, Glenn . Users and roles are assigned a range of sensitivity labels with which they may operate. For each such label, a corresponding labeled zone is instantiated, and users are constrained to run their processes in the corresponding zones.

5.2 The Trusted Path

Although the user can't start processes in the global zone, a *trusted path* menu is provided with which they may interact. The trusted path provides a reliable communication mechanism that can never be obscured or interfered with by untrusted processes. It provides a set of user interfaces and services to access labeled zones, allocatable devices, and administrative roles.

Users are initially authenticated using the window system in the global zone. Then they are presented with a set of trusted path interfaces to select an initial label or clearance. They may specify their labeling preferences and assign labels to individual desktop workspaces. The window system is able to initiate processes on behalf of users in the appropriately labeled zones.

For processes running in labeled zones, there are no interfaces to change their label or transition to another zone. However, an all-privileged global zone process can fork off a process in a labeled zone. To do so it must close any open files and relinquish any privileges beyond the limit established for the labeled zone. Once the transition occurs, the process and all of its descendants are contained within their labeled zone and cannot return to the global zone.

5.3 Namespace Hierarchy

Each zone has a unique root directory, which is a mounted ZFS data set. The mount point and all the files within it are labeled

with the zone's label; they are only writable within the zone. When a zone is configured it is assigned a set of filesystems. These filesystems either belong to the zone, or are imported from another zone. Directories imported from other zones retain the label of their owning zone. For example, most of the global zone's executable files and libraries are imported through a special read-only mount option called a loopback mount. Such files are protected at the lowest label on the system and are immutable in labeled zones.

Before any filesystem can be imported into a zone, the kernel performs a label comparison between the owning zone and the importing zone. For a read-only mount to succeed, the importing zone's label must dominate (have a higher sensitivity label than) the owning zone. An attempt to start a zone with an incorrect mount configuration will be logged and terminated. The MLS policy is always enforced and is not compromised by administrative errors. Once the mount configuration is corrected, the administrator can boot the zone.

A read-write mount is only permitted when the label of the zone equals the label of the filesystem to be mounted. Since each zone within a single system has a unique sensitivity label, the label of any locally exported filesystem will never equal the label of the importing zone and will never be mounted as writable. Therefore, no file is writable by more than one zone on any system. Zones with disjoint labels can never share their files.

However, remote filesystems accessed via the NFS protocol may be mounted for both read and write modes, if their remote label matches that of the importing zone. Otherwise, the MLS mount policy is identical for both remote and local filesystem imports.

The implementation imposes a natural hierarchical structure to the labels in the file namespace. The root of the filesystem has the highest label in the zone and sensitivity labels decrease monotonically as we traverse pathnames. Subjects cannot determine the existence of any higher-level filesystem objects since there are none present in their namespace.

5.4 Automatic File Labeling

One of the benefits of this implementation is that the MLS policy for read and write access to a file does not require explicit label comparisons. Instead of performing a MLS check every time a file is opened, the MLS check occurs when the filesystem containing the file is mounted. In general, the only time that a file's label needs to be determined is for human consumption, such as for auditing or for data archival. When a user queries the system to display the file's properties, or requests a printed copy of the file, the label is computed and displayed appropriately. Since looking up a file's label is rarely required, it doesn't need to be particularly efficient. So file labels are neither stored in the filesystem nor cached in the Solaris kernel.

The label of a file is invariant with respect to the label of the requestor. Since each zone has its own file namespace, the kernel first resolves all pathnames into a global zone representation before resolving any labels. In doing so it must deal with hard and soft links. Hard linking is a mechanism in the Solaris OS that provides the same underlying file node with multiple independent names in different directories. The system prevents hard links from spanning filesystems, ensuring that all hard links to a single file must have the same label. Soft links, also known as symbolic links,

may span filesystems so their label is resolved by following the links and determining the label of the final target.

The algorithm¹ for computing a file's label involves several steps. The global-zone relative pathname is computed and matched against the closest match in the mount table. If the entry corresponds to a local filesystem, the label of the zone owning the mount point is returned. For NFS mounts, if the remote server is a trusted host, the label is retrieved from the server. If the server is a single-level host, the label is determined from its network address.

The MLS policy prevents returning labels of files that are not dominated by the user's label. Since a user can only ask for a label of a file in its own namespace, the returned label will always be dominated by the requestor.

6. POLYINSTANTIATED DIRECTORIES

A traditional challenge in implementing MLS systems is the need to provide writeable instances of well known directories, such as `/tmp`, or user home directories. This is called *polyinstantiation*, since each label appears to have a unique instance of a directory with the same pathname. This problem has been solved in the past by customizing existing filesystem to provide special multilevel directories [11].

Since each zone has its own file namespace, multiple instances of the same pathname may exist in each labeled zone, but correspond to distinct directories. For example each zone will typically have a unique instance of a directory called `/export/home`. These directories are actually contained in different filesystems and are only writeable in the zone owning that filesystem.

Polyinstantiation of entire filesystems can also be accomplished by cloning ZFS snapshots. For example a ZFS data set (which is a type of filesystem) can be populated with lower-level information, and then preserved as a read-only snapshot. A unique instance of the snapshot can then be quickly cloned for each higher-level zone without copying the underlying data. Even though each zone initially shares a reference to the same underlying storage, the cloned instances are implicitly labeled by their owning zone. When a process in a labeled zone writes to any cloned filesystem object a unique instance of the file or directory is created for that zone.

7. LABELED FILE SHARING

A convention exists for exporting such directories in read-only mode so that they can be accessed by processes in higher-level zones. The convention is to mount such lower-level shared directories using the prefix `/zone/zonename` where *zonename* is the name of the lower-level zone. In this way the higher-level zone can read the shared portion of the lower-level zone's named space.

Figure 1 shows three labeled zones: *need-to-know*, *internal*, and *public*. The labels for these zones have a dominance relationship, where *need-to-know* strictly dominates *internal*, and *internal* strictly dominates *public*. All three of these zones have been configured by the policy set in the global zone to permit sharing of their instance of

the `/export` directory. Since the *internal* zone's label strictly dominates the *public* zone, a process in the *internal* zone can read the *public* zone's `/public` directory using the pathname `/zone/public/export`.

Meanwhile, a process in the *need-to-know* zone can also read the *public* zone's `/export` directory use the `/zone/public/export` pathname, as well as the *internal* zone's `/export` directory using the `/zone/internal/export` pathname. Neither the *internal* nor *public* zones can read from the *need-to-know* zone, even though its `/export` directory is shared, because their labels do not dominate the *need-to-know* zone's label. The global zone's `/usr` directory is shared for reading by all labeled zones.

The enumeration of shared filesystems is specified by a global zone administrative role. Since zones have separate namespaces, it is not possible for a process in one zone to directly mount any files owned by another zone. But the global zone includes the root directories of both namespaces, so a privileged global zone role can create a read only loopback mount from a lower-level zone to a higher-level zone. Such mounts are always consistent with the MLS policy. If the global zone administrator mistakenly attempts to create a loopback mount from a higher-level zone to a lower-level zone, the operation will be denied.

A second mechanism for file sharing is the use of NFS exports. Higher-level labeled zone processes may mount lower-level shared directories for read-only access via standard NFS protocols. An automount daemon in each zone does this dynamically, or each zone can be configured with its own instance of `/etc/vfstab`

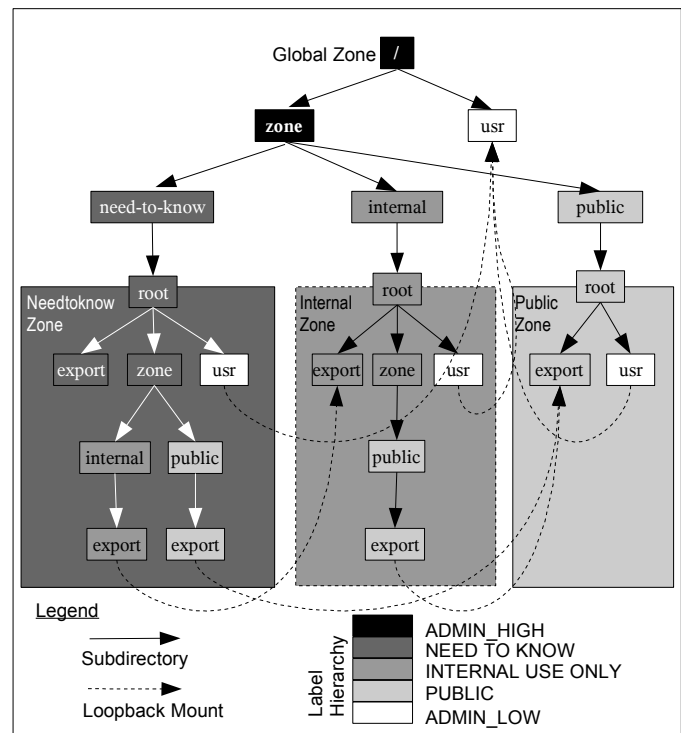


Figure 1. File sharing via loopback mounts.

¹The complete implementation can be viewed at <http://src.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/uts/common/os/tlabel.c>

to specify NFS mounts at boot time. Again, the MLS policy is enforced by the kernel to prevent inappropriate data flows.

Note that sharing is not transitive. A filesystem which is imported into any zone via an NFS mount cannot be exported from that zone. Each zone must import any shared filesystems from the entity that exported it. Therefore, the global zone's namespace includes only those portions of a labeled zone's namespace that were mounted from the global zone. Special privileges are required for global zone processes to observe the files or processes belonging to labeled zones.

A user in a zone can create symbolic links to files in these lower-level named spaces, so that they may appear at any point in the current zone's namespace to which the user has write access. The user may query the system to return the labels of any such files. Regardless of the path name used to refer to the file, the label will always reflect that of the zone from which it is imported.

Directories exported from the global zone are always assigned the label ADMIN_LOW. Note that any such lower-level mounts are always read-only. Higher-level mounts are prohibited.

8. UNIDIRECTIONAL DATA FLOWS

The ability to read lower-level files from a higher-level zone is called *reading down*. The lower-level process is only able to write at its own label, and cannot *write up*. However, two processes in separate zones can communicate using a FIFO conduit. A special rendezvous file, known as a *named pipe* can be created by one process, which then writes to the file. Another process reading from the file will receive the data written by the first process. The OS synchronizes these operations so that the reader will wait for data and terminate when the writer closes the file. This mechanism can be used by labeled zones to achieve a unidirectional data flow. If the writer creates the named pipe in a directory which is exported from a lower-level zone and imported into a higher-level zone, the named pipe will appear in the namespace of the higher level zone as a read-only object. It retains the label of the lower-level zone.

The apparent write up in this context is consistent with the MLS policy because the writer is simply writing to an object at its own label, and cannot identify the process at the other end of the FIFO. The reader in the higher-level zone cannot reply to the lower-level process. Unidirectional flows provide a mechanism for audit trails and similar logs that cannot be destroyed by processes within the zone. Even an all-privileged process within the zone cannot modify or erase the log once it has been recorded by a process in the higher-level zone.

9. LABEL MAINTENANCE

Although files are automatically labeled when mounted, the system provides for archiving labels, and for relabeling of existing files.

For offline storage a label-aware version of the `tar` program is provided to record the label of each file when it is archived. The labels are recorded in an ancillary file format which is backward compatible with Trusted Solaris labeled archives. The archived labels are used for validation when files are restored to ensure that the file labels are not changed. A file is not restored unless the

archived file's label is equal to the label of the zone into which it is being restored.

Authorized users may request that a the file label be reclassified. Separate authorizations are required for users to either upgrade or downgrade a file. Such requests are handled by a global zone service that validates the request, including a check to ensure the file is not currently in use. If the user is authorized and the file is unused, the file is relabeled by removing it from its current filesystem and copying it into a filesystem corresponding to the requested label. The pathname of the original file is adjusted to the namespace of the new filesystem.

An interesting side-effect of this operation is that it cannot be undone by the original subject, since the file disappears from its current directory in the original namespace.

10. RELATED WORKS

The Trusted Linux system proposed by Dalton and Choo [1] makes use of separate filesystem roots to achieve containment of web services. Their compartments are similar to the labeled zones in Solaris Trusted Extensions, but their labeling strategy does not include a hierarchical component. Therefore, there is no implicit policy for information flow other than complete isolation between compartments; explicit cross-compartment data flows are enumerated in a policy file. There is no concept of privileged processes within a compartment; transitions to `root` are prevented.

SELinux can be configured to enforce an MLS policy in addition to the primary TE Policy. To gain access to an object a subject must pass the traditional DAC policy, the TE policy, and the MLS policy. The following discussion is focused on the MLS filesystem policy. Both Trusted Solaris [11] and SELinux [9] provide extensions to the `mount` command to specify the default security context for a filesystem. This requires an administrator to determine the appropriate context, which is a source of human error. Furthermore, neither of these systems require that the label of the mounted filesystem be dominated by the label of the mount point. They permit higher-level filesystems or individual higher-level files to exist in directories that are accessible from lower-level subjects, and they permit authorized subject to upgrade a file so that its label dominates its directory.

Both Trusted Solaris and the SELinux MLS policy prevent normal subjects from reading higher-level files, but the file names and their existence represent sensitive information which should be hidden from the namespace. Filtering such multilevel information in a directory is problematic. Although the system call for reading directory entries, `getdents`, can be modified to hide upgraded file and directory names, a lower-level subject can infer their existence by attempting to create duplicate entries at its own label. SELinux does not provide such filtering so higher level file names are visible to lower level subjects. In Trusted Solaris, the policy for hiding upgraded file names is configurable because its enforcement introduces additional overhead affecting performance. This contrasts with the Solaris Trusted Extensions architecture, where upgrading a file is done by moving it to a higher-level directory in another filesystem. No filtering is required because directories contain no upgraded entries.

11. CONCLUSION

Much has been written about configuring flexible security policies and developing tools to validate them [5, 13]. This paper asserts that a useful security policy is best implemented when it is automatically consistent and easily understood. The use of labeled zones eliminates the need for explicitly labeled filesystems or for specifying labels as mount attributes. This approach provides a significant set of advantages:

- Since no modifications are required to any filesystem structures, all commercially available filesystems are compatible with the MLS policies enforced in the kernel.
- Unlike systems, where labeling attributes are specified as context mount [9] options, the mount point labels are automatically applied. This prevents a source of human error and inconsistency.
- Since labeled zones provide unique instances of directories and services at each label, unmodified applications can run at multiple labels concurrently.
- Since the zones are completely isolated by default and applications require no modification, it is unnecessary to maintain complex policy files.

This new MLS architecture is currently undergoing Common Criteria certification using the Labeled Security (LSPP) and Role-Based Protection Profiles (RBACPP). This approach is not a panacea, however. One of its weaknesses is that the underlying filesystems are only protected while the trusted OS is running. Of course, this is also true of systems which implement explicit security contexts. For example, an explicitly labeled filesystem can be mounted on another system which ignores the explicit labels.

Another tradeoff is the requirement to configure a unique zone for each label. Although zones can be provisioned efficiently using ZFS cloning, the administrator must determine the set of labels and zones that will be required by the user community.

A more robust solution to this problem requires labeled protection for data at rest. Cryptographic filesystems provide the missing functionality. Encrypted ZFS data sets are currently in development [15]. Each ZFS data set can be independently encrypted, so that the cryptographic keys can be associated with their corresponding labels. The MLS policy is applied to key management so that the filesystem cannot be mounted without providing the proper key for that label. As work is ongoing in this area, the details of this implementation will be the subject of another paper.

12. ACKNOWLEDGMENTS

The author is thankful for the the efforts of the Rampart project team whose efforts made this possible.

REFERENCES

- [1] Dalton C. and Choo T. S. *Trusted Linux: An operating system approach to securing e-services* Communications of the ACM, Volume 44, Issue 2. 2001
- [2] Faden, Glenn *Reconciling CMW Requirements with Those of X11 Applications*, Proceedings of the 14th National Computer Security Conference, Washington, D.C. 1991
- [3] Faden, Glenn *RBAC in UNIX administration* Proceedings of the fourth ACM workshop on Role-based access control, Fairfax, Virginia, United States, 1999, Pages: 95 – 101
- [4] Faden, Glenn, *Solaris Trusted Extensions: An Architectural Overview* 2006
<http://www.opensolaris.org/os/community/security/projects/tx/TrustedExtensionsArch.pdf>
- [5] Jaeger T., Zhang, X, and Cacheda F. *Policy management using access control spaces* ACM Transactions on Information and System Security (TISSEC) Volume 6 , Issue 3 , page 327-364. 2003
- [6] Mc Dougall, R. and Mauro J. *Solaris Internals*, Chapter 15 UFS File System, page 764, 15.4 Access Control in UFS, Sun Microsystems Press, 2005
- [7] Morris, James *Filesystem Labeling in SELinux* Linux Journal Contents #126, October 2004
<http://www.linuxjournal.com/article/7426>
- [8] Oracle Corporation *Data Classification with Oracle Label Security* 2005
http://www.oracle.com/technology/deploy/security/db_security/pdf/twp_security_db_ols_10gr2.pdf
- [9] Red Hat *Taking Advantage of SELinux in Red Hat Enterprise Linux*, Red Hat Magazine, Issue #6, April 2005
<http://www.redhat.com/magazine/006apr05/features/selinux>
- [10] Solaris Trusted Extensions
<http://www.opensolaris.org/os/community/security/projects/tx>
- [11] Sun Microsystems, *Trusted Solaris 8: A Technical Overview*, page 15. 2000 <http://www.sun.com/software/whitepapers/wp-ts8/ts8-wp.pdf>
- [12] US Department of Defense *Trusted Computer System Evaluation Criteria*, DoD 5200.28-STD. December 26, 1985.
- [13] Zanin, G. and Mancini L. *Towards a formal model for security policies specification and validation in the selinux system*, Proceedings of the ninth ACM symposium on Access control models and technologies, pages 136-145, Yorktown Heights, New York, USA , 2004
- [14] ZFS at OpenSolaris.org
<http://www.opensolaris.org/os/community/zfs>
- [15] ZFS on disk encryption support
<http://www.opensolaris.org/os/project/zfs-crypto/>
- [16] Zones at OpenSolaris.org
<http://www.opensolaris.org/os/community/zones>