MCS - adding MLS features to the targeted policy

Russell Coker ⟨rcoker@redhat.com⟩,
http://www.coker.com.au/selinux/
Red Hat

Abstract

It has become apparent that many people want some of the benefits of MLS but in a way that is easier to use than the full MLS implementation.

MCS (Multi-Category Security) is a change to policy that applies to both strict and targeted policies (it is in strict and targeted in rawhide). It uses the categories of MLS but makes no use of the sensitivity field (all processes have sensitivity *s0*). The initial design of MCS is based around the targeted policy so it has the same initial aims (minimising the cost of implementation for end-users). However MCS is used in the strict policy as well and provides the same confidentialiy benefits for strict as for targeted.

MCS operates in addition to the *type enforcement* model and is designed to be used primarily for protecting data confidentiality while type enforcement will be used for protecting system integrity. It is an optional feature which is always enabled but will not have an impact on anyone who doesn't choose to use it.

1 Introduction

The primary characteristic of MLS [1] is that it has multiple security levels. Each level is comprised of a classification and an optional set of compartments. Names such as *Top Secret*, *Secret*, *Classified*, and *Unclassified* are often used for the classification. There is no technical requirement that there be four classifications (there could be more or fewer), or that those names be used. In the default SE Linux MLS policy there are 16 classifications with id codes from *s0* to *s15*, the translation table may map some of those to the traditional four names.

In a MLS system the primary aim is to prevent "read up" or "write down". "Read up" means reading data that is classified at a level higher than the process that is reading it, for example a process which has a clearance level of *Secret* can not read data from a file classified at the level of *Top Secret*. "Write down" means writing data into a level lower than the process that is writing it, for example a process which has a clearance of *Secret* writing to a file classified as *Unclassified*. Preventing "read up" and "write down" means that both intentional release of secret data to processes/people that lack appropriate clearance and accidental release via mistakes or trojan horse programs is prevented.

It is permitted to "write up" and "read down". In practice this means that a document can be written at a *Top Secret* classification which is based on data from *Secret*, *Classified*, and *Unclassified* categories. Also a process with *Unclassified* clearance could write to a file classified as *Top Secret*.

There is another issue of integrity which is addressed in SE Linux by using the *domain-type* model. SE Linux has not (and will not) implement the BIBA integrity model that is conceptually similar to MLS as it is felt that the domain-type model is a better solution to the problem.

Classifications are not adequate to fully describe the security needs of military agencies or the computer systems that they use. It is common that people will not be permitted to read data that is at a classification that they have clearance for but is in a different area (project, organization, etc). To implement this the MLS model includes a feature known as *Categories*. Every process in the MLS model has a clearance that includes zero or more categories. Each file that may be accessed will have a sensitivity which is comprised of the classification and zero or more categories. If the set of categories for which the process is cleared is not a super-set of the categories with which the file is classified then the file can not be read.

In a commercial environment you might have cate-

gories for each NDA agreement that you sign with another company, for each secret research and development project, and for information that can't be released due to insider trading laws.

2 SE Linux Implementation of MLS

SE Linux provides a MAC system that is a second layer of defense, Unix permissions are the first line of defense. So you may have multiple users who have the same SE Linux security context who keep their data secret from each other through Unix permissions.

The primary feature of SE Linux is the *domain-type* model. In this model every process (subject) has a *domain* associated with it and every object that a process may access has a *type* associated with it. At the core of SE Linux is a database which describes for every combination of domain and type the access that is to be granted at a fine granularity [2]. For example file access is divided into *create ioctl read getattr lock write setattr append link unlink rename*.

Until recently the vast majority of SE Linux users did not use the MLS features. The most popular implementation of SE Linux has been in the *Red Hat* distributions, *Fedora Core* and *Red Hat Enterprise Linux*. In Fedora we have two policies, *strict* and *targeted*. The strict policy aims to grant minimal privileges to each process while the targeted policy has a domain named *unconfined_t* that has no restrictions and runs all programs from login sessions in that domain. The targeted policy aims to reduce the cost (in terms of administration work) of using SE Linux as much as possible.

3 Problems with MLS on Linux

One major issue for implementing MLS is that it doesn't work well with the Unix design. A design goal of SE Linux has been to work well with unmodified versions of commonly used software as much as possible. It is important for the success of SE Linux that it is not considered to be unreasonably difficult for administrators, users, or programmers. Many programs perform operations such as opening a file as read/write when they only need write access. This is not seen as a problem and developers often are reluctant to change such code.

Bad coding practices such as this are a significant obstacle to the development of MAC systems on Unix, getting programs changed to work well with the strict policy has been a constant battle even though most of the changes required for that are generally agreed to be good based on community standards for Unix program design. One common example of a change that is required for the strict policy is inheritance of file handles. When a process running as root spawns another process running as root it's often not considered a big problem if all the file handles are allowed to be inherited, of course if they get inherited by a non-root process it becomes a problem. Also if a bug in a child process causes it to write to the wrong file handle it could interfere with the operation of the parent. Prior to the release of SE Linux there was no convenient way for the administrator to track which file handles were being inherited. SE Linux implements two permission checks for file handle inheritance, there is a *fd use* permission to determine whether the a process may receive a file handle from a process in a different domain (which may not be the parent, it may be a grand-parent process, or the file handle may have been transferred over Unix domain sockets). The other permission check is to determine whether the process inheriting the file handle is permitted the access (either read, write, or read/write) that was used when the file handle was opened. Due to these checks many bugs have been found and fixed in commonly used programs (one can only imagine the number of instances of inappropriate file handle inheritance in cases where there is no domain transition and therefore SE Linux does not log it).

The difference with MLS related changes is that people believe that there is little reason for implementing such changes in a non-MLS environment. In the traditional Unix design it's usually regarded that a process with write access to a file should have read access as well. The aim of access control in Unix has traditionally been primarily to protect system integrity not data secrecy.

The goal for protecting secrecy is that a process which can read everything should have write access to few files to prevent writing secret data to the wrong place, and that a process with wide write access should not be able to read secret data so that the public files it writes to will not contain any secrets. The traditional Unix design for protecting system integrity has been to have processes with high integrity that are permitted to do whatever they want.

Note that these problems do not impede our work on getting LSPP certification for SE Linux systems. Getting MLS working with the programs that are required for LSPP certification is not that difficult, it's getting it working with all the programs in Fedora which is so difficult as to be almost impossible!

The problems with getting wide spread developer support for the full MLS policy is the easy part of the problem. The most difficult part is in training users. Teaching the typical users and system administrators about the concepts involved with this will be difficult. Teaching them how to use it in practice with symbolic names

for MLS contexts and ranges for processes would be insanely difficult.

I am obliged to note that shortly before completing the final draft of this paper I gave demonstrations of MLS on Fedora Core 5 Test 2 to delegates at the *2006 Linux Conference Au* and it received positive reviews. Given that some people are describing MLS as *cool* I guess that there is a possibility for it to be used in home systems. There's no accounting for taste.

4 The Design of MCS

MCS is an extension to both the strict and targeted policies which was mostly designed for the targeted policy. It aims to allow categorising data without increasing the difficulty of system administration. The full MLS model is not suitable for most users for the reasons described above, but we believe that there is a need for something more than the domain-type model to protect data secrecy.

People who are used to using either the *targeted* or *strict* policies will have those options available. MCS can be considered an additional layer of protection applied after the controls of *targeted* or *strict* policies. MCS is also designed to be transperant, so a major design goal (which is fully achieved) is to have MCS make no impact on users who choose not to use it. It is possible to use the *type enforcement* features of SE Linux and ignore the existence of MCS.

To avoid the full cost of MLS it was decided that classifications would not be used. The SE Linux design permits using a policy that has no reference to MLS (no classifications or categories defined) which has been commonly used in the past. However there is no support for a policy without classification but only categories, so for MCS we just define a policy with only a single classification with code *s0*. For a MLS SE Linux policy the default configuration is to have classifications with codes *s0* to *s15* inclusive).

The other significant change in the MCS policy is that instead of preventing read-up and write-down we instead only permit reading or writing to a context that is dominated by the context of the process. This means that a process can read and write a file if it posesses a superset of the categories that are assigned to the file and will not get either read or write access otherwise. The general expectation among Unix users and administrators is that granting extra privileges to a process will only add to the set of actions that it is permitted to perform, as opposed to the MLS expectation that permitting a process to read from more secret data should also deny it from writing to less secret data. This change means that the MCS policy does not support the Bell La-Padula (BLP) security model or a sub-set of it, so MCS is not a cut-down MLS policy, it is an entirely different policy that uses the same kernel and policy language features!

This is required to permit operatiion in a manner that most system administrators expect. It is expected behavior that if you can write to a file then you can read it. In fact many programs open files with read/write access when they only want to write to them and thus fail to perform as expected on MLS systems.

In some ways MCS categories act in a similar manner to Unix supplemental group IDs with the added benefit that the user can drop categories that they don't need for a particular task. This fits well with the practices of Unix users while also adding some significant benefits. With Unix groups it is not possible for a non-root process to easily drop access to groups that are not needed. There is a limit of a small number of groups (we are working with a default of 256 categories for MCS and more can easily be added). Also there is no method of labelling a file on disk for supplemental groups, in Unix a file must have exactly one GID.

Of course the down-side of breaking the no-read-up and no-write-down convention of MLS is that under the MCS policy information leaks will be possible, a process that has categories *c0* and *c1* can read data from a file classified with category *c0* and write it to a file classified with category *c1*. In cases where it is necessary to stop accidental leaks it will be necessary to prevent one process from having both the categories in question. It is my observation that in most non-military environments this is not considered a problem, I belive that someone who is permitted to access two different categories of data will be trusted not to read data from a file of one category and write to a file of another. Note that MCS does not attempt to deal with the problem of a hostile user or a trojan horse program that wants to maliciously relabel files with fewer categories or copy data to a file that has fewer categories than the source file.

5 Uses of MCS

One of the primary design aims of the MCS policy was to support corporate environments where different categories of data are used. This is the most obvious real-world use for it.

Another possible use is for multiple instances of a program. Currently if you want to have two copies of a program running which are not permitted to interact with each other then you need to modify the policy to have two domains (there is work in progress to make this relatively easy, but at the moment it's inconvenient). For example you could start two copies of a program with

the commands *runcon -l s0:c200 program* and *runcon -l s0:c201 program*. This would spawn two instances of the program which would not be able to directly interact with each other and which by default would create files that the other instance could not read or write.

Finally I can imagine MCS being used on University shell servers. On such machines trojan horse attacks are common, so the ability for an unprivileged user to easily run a process with reduced access to the system so that it could not cause damage would be very useful. When I first started work on SE Linux sys-admins often asked me whether unprivileged users could use SE Linux to restrict the access of their own processes. At the time it was unfortunately unreasonably inconvenient for both the system administrator and the user to implement such systems. The only viable option for the administrator was to define a large number of roles for the user in question this would significantly increase the size of the policy and therefore the amount of kernel memory used. Now with the MCS policy every user can easily run programs with restricted access.

There are probably other uses of MCS that will be discovered when it is widely available in *Fedora Core 5*. The close ties between the design of MCS and the traditions of Unix system administration will make it easy for the typical sys-admin to work with. Because of this I expect that users will rapidly devise uses of MCS that I have not considered. Note that I expect more developments in this regard on the *Fedora* platform than on the *Red Hat Enterprise Linux* platform because *Fedora* machines tend to have a more experimental nature. The same also applies for the *Debian* and *Gentoo* distributions which also are popular among home users and students.

6 Mandatory aspects of MCS

The initial release of MCS was advisory. It permitted a process to launch a child process with more MCS access rights, the domain-type feature of SE Linux and Unix permissions were relied on for access control.

I have been working on making MCS enforcing. This means that a process can not launch a child with any MCS categories that it doesn't posess, it can't debug a process that has such categories, or kill it. It can however launch a child with less categories and there is no restriction on doing so.

The next issue regarding mandatory MCS is the default category assignment of files. With the MLS kernel code in SE Linux a sensitivity label may have a range. For labels on processes ranges are used and the low level of the range is used for the default level of files which it creates (ranges are not used for files and directories - the low and high levels are forced to be equal). The default sensitivity of a process for a user shell in MCS is *s0-s0:c0.c255* which means that it has a range from *s0* (no categories) to *s0:c0.c255* (all 256 categories). In this case the default sensitivity of files will be *s0*. However we can configure the MCS policy via the *semanage* tool to specify a different default low level of the sensitivity range. For example a user could have the range *s0:c0.c100-s0:c0.c200*, this means that it can access files with categories from the set *c0.c200* (all categories from *c0* to *c200* inclusive) and that the default sensitivity of any files it creates will be *s0:c0.c100*. It would be denied access to categories outside the set *c0.c200* but would be permitted to execute a child process with a set of categories which is a sub-set of that. Also it could execute a child process with a low level that is any sub-set of the high level. Valid sensitivity ranges that may be used for child processes include *s0:c0.c199-s0:c0.c200*, *s0:c0.c200-s0:c0.c200* (which is equivalent to *s0:c0.c200*), and *s0:c0.c5-s0:c0.c199*.

In the MLS policy there are restrictions on setting the low sensitivity of the range. I considered placing such restrictions on the MCS policy. What could be done is to prevent a user from launching a process with a low level which is lower than the low level of it's parent and also preventing a process from relabeling a file to a level that's below it's low level.

An example of using this could be to make the low level for users in the *HR* department include the *HR* category. This would mean that every process they launch would have the *HR* category in the low level (of course they could increase the low level and add other categories for which they are authorised) and they could not create any files which lack that category. This would mean that every file created by a user in the *HR* department would have the *HR* category.

I am not sure that the benefits of this outweigh the costs. At the moment my plans for MCS involve having the only restriction of the low level of a process be that it is dominated by the high sensitivity.

```
s0:c0=HR
s0:c1=Financial
s0:c0,c1=HR&Financial
s0:c0-s0:c0,c1=HR-HR&Financial
```

In a corporate environment the administrator might use the above in the *setrans.conf* file to name the *HR* and *Financial* categories. A user who works for the HR department might have a default MCS sensitivity range of *s0:c0-s0:c0.c1* which is known as *HR-HR&Financial*. This means that by default files will be created with the

sensitivity label of *s0:c0* AKA *HR* but that files with the *Financial* category (*c1*) can also be accessed. The user in question would also be permitted to request that a file they create be labeled as *HR&Financial* (useful for copying files), and they would be permitted to relabel files between *s0*, all combinations of the *HR* and *Financial* categories.

7 Migrating to the MCS Policy

One significant issue when developing new policy is to allow users to easily migrate to it.

When labelling files and directories on disk each file system object has an XATTR named *security.selinux* that contains the security context for the object. If an object has a security context which does not match the policy which is in use then it is given the type *unlabeled_t* which prevents access to all processes other than those run by the system administrator. In the previous versions of the SE Linux kernel code if a file had a security context without the MLS components then it would get the type *unlabeled_t* which made an upgrade from a non-MLS system to MLS or MCS more difficult than necessary. A recent change in the kernel code assigns a default sensitivity label of *s0* to such files. So when upgrading a machine to the MCS policy all files will have label *s0* which is exactly what we desire for an initial MCS installation.

This was the only kernel change related to the MCS policy, and it applies equally well to the MLS policy. Due to the flexibility designed into the SE Linux policy language the design of the MCS policy did not require any kernel code changes!

8 Integrating MCS into a Distribution

Supporting MCS in a distribution requires a lot more work than just adding it to the policy. It has to be possible to manage the labelling of files and processes in a manner that's not overly difficult. The basic support for changing the security contexts of files and preserving the context when copying files and backing them up has been in Fedora since Fedora Core 2. One of the issues with MLS is that only certain combinations of categories may make sense. So we need to have a method of translating MLS contexts to a human readable form. In rawhide we currently use the file */etc/selinux/X/setrans.conf* (where *X* is the name of the policy, either *targeted*, *strict*, or *mls* to store translations from MLS labels to human readable labels, below is an extract from the default file (in rawhide as of Jan 2006):

```
# s0:c0=CompanyConfidential
# s0:c1=PatientRecord
# s0:c2=Unclassified
# s0:c3=TopSecret
# s0:c1,c3=CompanyConfidentialRedHat
s0=
s0-s0:c0.c255=SystemLow-SystemHigh
s0:c0.c255=SystemHigh
```

If the comment characters are removed then this means that category *c0* maps to *CompanyConfidential* and the combination of categories *c0* and *c3* maps to *CompanyConfidentialRedHat*. With the current implementation a combination of the categories *c0* and *c1* will not be translated, a translation is only applied if there is an exact match from the MLS context to an entry in the mcs.conf file. This mapping is used by programs such as *ls* to display the context of files when the *-Z* option is used to display SE Linux contexts as we can see below:

```
# ls -aZ /tmp/foo
drwx------ etbe etbe etbe:object_r:\
staff_tmp_t:SystemHigh .
drwxrwxrwt root root root:object_r:\
tmp_t           ..
-rw-rw-r-- etbe etbe etbe:object_r:\
staff_tmp_t         foo
# ls -la
total 32
drwx------  2 etbe etbe 16384 Sep 28\
17:05 .
drwxrwxrwt 15 root root  4096 Sep 28\
17:22 ..
-rw-rw-r--  1 etbe etbe     0 Sep 28\
17:05 foo
```

In the above example you can see that the directory in question has the classification *s0* and the categories *c0* to *c127* inclusive.

Below is an example of the output of the *id* command:

```
# id -Z
etbe:staff_r:staff_t:SystemLow-\
SystemHigh
```

The above output shows that the clearance of the current process is *s0-s0:c0.c255*.

9 Further development of MCS/MLS

The MCS policy not only provides the benefits stated previously but also facilitates further development of the MLS features of SE Linux. Prior to the release of MCS hardly anyone was using the MLS features which

meant that numerous usability issues were not being addressed. The release of MCS means that many organizations which would not consider a full MLS implementation will now be using MLS features - and expecting them to work correctly! This of course means that software developers (people who work for the distributions of Linux, "upstream" free-software developers, and commercial application programmers) will be compelled to make their software work in an MLS environment. This addresses the greatest problem that has faced commercial use of MLS systems, the fact that almost no software was written specifically to support it and a large portion of the available software failed to work in some way when used on an MLS system.

When the MCS policy is widely released (Fedora Core 5 and Red Hat Enterprise Linux 5) it will be quite easy to convert an FC5 or RHEL5 machine to the full MLS policy. NB The full MLS policy will not be covered by the standard Enterprise Linux support contract.

The full MLS policy is designed to fulfill the requirements for certification under the *Labeled Security Protection Profile (LSPP)* [3]. Certification is a long process, merely having MLS support is far from enough to get certification.

References

[1] *Introduction to Multilevel Security*
Dr. Rick Smith,
University of St. Thomas Minnesota
`http://www.cs.stthomas.edu/faculty/resmith/r/mls/index.html/`

[2] *Configuring the SELinux Policy*
Stephen D. Smalley, NAI Labs, ssmalley@nai.com
`http://www.nsa.gov/selinux/policy2-abs.html/`

[3] *Labeled Security Protection Profile*
Information Systems Security Organization NSA
`http://niap.nist.gov/cc-scheme/pp/PP_LSPP_V1.b.pdf/`

[4] *A Brief Introduction to Multi-Category Security (MCS)*
James Morris, Red Hat, jamesm@redhat.com
`http://www.livejournal.com/users/james_morris/5583.html/`