Unlocking the Power of SELinux by Utilizing the CDS Framework IDE

Benjamin Stroud
CS591
Fundamentals of Computer/
Network Security
Fall 2009
bstroud@uccs.edu

***Abstract*-- SELinux (Security Enhanced Linux) is fast becoming a popular and powerful tool for large and small businesses as well as many government organizations. The CDS (Cross Domain Solution) Framework IDE allows developers and Information Technology professionals to harness this radical new power without having to delve deep within the architecture of the system. The overarching goal of this paper is to explore the CDS Framework IDE and present its capabilities and limitations. I will touch on the history behind the framework and give some background on SELinux in general. I cover some of the benefits of using the Framework. There will be a section exploring how to setup the framework and how to create policies as well as how the development environment translates a user's project into a functioning SELinux policy. I will explore alternatives to this solution and what new directions the creators are planning to take. Finally there will be a discussion of ideas my ideas for improvements to the Framework as well as a presentation of my overall conclusions.**

Keywords – CDS Framework IDE, SELinux, policy editor, Cross Domain Solution, Secure Linux, Linux, TreSys, Open Source Tool, GUI Policy Editor

I. INTRODUCTION

SE(Security Enhanced)Linux is a powerful tool for organizations concerned with information security. It is often seen, however, as extremely daunting from a management standpoint. Administrators believe that SELinux and its policies can only be used by security experts and that without an in-depth knowledge of the inner workings of SELinux one cannot hope to use it effectively. I will show through the exploration of the CDS(Cross Domain Solution) Framework IDE(Integrated Development Environment) that not only does the use of this framework make SELinux more accessible, but it also makes it more powerful by giving an administrator the ability to directly translate an intuitive understanding of how information should flow securely through a system, and then directly translate that understanding into usable SELinux policies.

In this paper I will discuss (1) what SELinux is (2) how policies are used and what their advantages and limitations include, (3) what a Cross Domain Solution is and how SELinux makes them possible, (4) what the CDS Framework is, (5) what its benefits include, (6) how it is used, (7) some available alternatives, (8) and some criticisms of Cross Domain Solutions in general. There will also be a final section reviewing and summarizing my findings as well as presenting ideas for future improvements to the project.

II. WHAT IS SELINUX?

A common misconception about SELinux is that it is a stand-alone operating system, when in fact it is a security enhancement to Linux which gives the administrator the ability to control which users and more importantly, which applications and services, can access which resources. Like most modern operating systems, Linux traditionally has used access controls such as file permissions or modes that are modifiable by various levels of users. SELinux, however, gives us the ability to define access controls via a system wide policy that can't be changed by careless users or malicious applications[1].

SELinux also gives an administrator a much more fine-tuned ability to restrict access. For example, rather than being able to just control who can read , write or execute a file, SELinux policies allow an administrator to specify if someone (or something) can move, only append to, or unlink a file. Policies apply not only to files on the system, but almost any other system resource, such as network resources and IPCs (Inter-Process Communications)[1] such as sockets, pipes, or signals[2]

III. WHAT IS A CROSS DOMAIN SOLUTION?

A CDS is essentially a way to transfer data from one domain to another. Within the context of the CDS Framework domains are defined as containers within a system that form groups of entities (processes, resources, etc) with the same security level and access to the same resources. So a CDS gives an administrator the ability to define a secure environment in which data from one security level can be transmitted to another[3].

A. Example Architecture

In the following section I present a typical scenario that will illustrate when a CDS can be of use, and through this example I will clarify what a CDS is. The goal of the administrator in this example is to connect a machine with sensitive data (Domain B) to the Internet (Domain A) for the purpose of receiving emails[3].

The administrator or system architect must filter the traffic through several gatekeeper processes that perform various inspections of the incoming data, without allowing any compromise of the trusted information zone's integrity due to the transfer of untrusted data. The traffic must travel through a virus filter, a spam filter, a content filter, as well as the basic mail service running on the system as seen in figure (1). Each of these services has its own unique task and requirements to function within the system [3].

The framework is a plug-in to Eclipse[5] which is a popular open source java based IDE[6]. The CDS Framework was created by Tresys Technology who also built SLIDE (SELinux Policy IDE)[6] which is the basis for the framework[5] and will be discussed in the following section.

A. SLIDE as it relates to the CDS Framework IDE

SLIDE, like the CDS Framework IDE, was designed as a plug-in for Eclipse. It is a text based, rather than graphics based IDE. Much like the CDS Framework IDE, SLIDE allows for a user to create and edit SELinux policies[5]. Before the CDS Framework can output an SELinux binary file the graphical representation of the policy must first be converted into a language that SLIDE can understand, at which point SLIDE is responsible for actually compiling the higher level output of the Framework into a usable SELinux policy[7]. I give a more detailed description of this process in a later section.

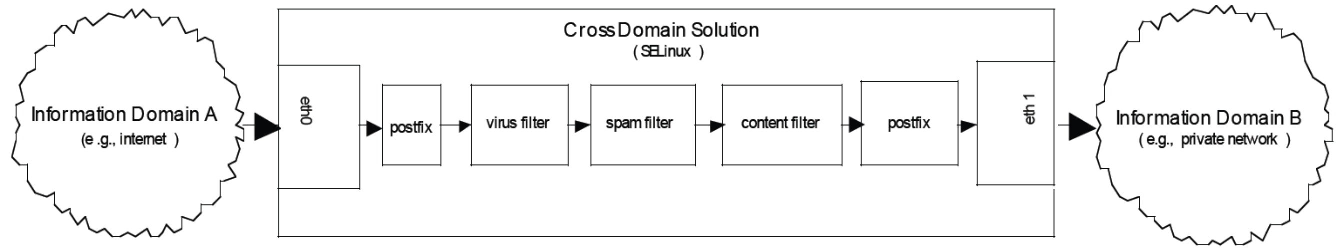


Figure 1: Example CDS Architecture[3]

B. Goals of the Example Architecture

The administrator's goal is to minimize the amount of responsibility and therefore trust that is given to the intermediate processes. If any of these processes were to become compromised a properly designed CDS would minimize the amount of damage that could be done to the system by controlling the data flow from one zone to another and restricting what each process can do. The combination of SELinux and the CDS Framework IDE is what gives the administrator the power to create this solution [3].

IV. WHAT IS THE CDS FRAMEWORK IDE?

The CDS Framework is an open source tool that allows system administrators to create Cross Domain Solutions and edit SELinux policies without requiring a deep and intimate knowledge of the underlying details of SELinux policy creation or editing. The CDS Framework is GUI driven software solution that allows a system designer to drag and drop objects representative of system components into a workspace and arrange those objects in an intuitive way that is then translated into a functional SELinux policy[4]. The arrangement of the design components in the CDS Framework project represent how data should flow through a system and where the domain boundaries lie[5].

B. CDS Framework Concepts and How They Relate to SELinux

Within this paper I use SELinux related terms (such as domains, policies, etc.) from the perspective of the CDS Framework and how they are defined by the creators and used within the tool since the Framework is my primary focus. In this section I want to take a moment to point out some of the differences between how the CDS Framework and its creators define - or even add additional - terms that refer to concepts that originated within the realm of SELinux.

1) Domains: The CDS Framework IDE creators define domains as containers within a system that form groups of entities (processes, resources, etc.) with the same security level and access to the same resources[5]. In SELinux the concept of domains are that they are simply types associated with particular processes which control what access a process has to the system[8].

2) Decomposing Domains: This paper makes reference to decomposed domains within the CDS Framework. Decomposing a domain is defined as breaking domains into sub-domains. Sub-domains are bound by the same rules as their parent domains and can only interact through shared resources (defined below). Decomposing domains helps to further isolate access to objects within the domain. SELinux does not have any concept of decomposing domains[5]. This is due to the difference in the way domains are defined within the context of the Framework and SELinux as discussed above.

3) *Shared Resources*: The CDS Framework uses a higher conceptual idea to define shared resources than SELinux. Shared resources are entities shared across different domains which allow for information to pass between them. Within the Framework a single shared resource can include multiple system objects (pipes, sockets, etc.) which are treated as one entity. In SELinux the single conceptual object would be broken down into each of its system object classes. While each of these objects individually would still be referred to shared resources in SELinux, the higher level grouping does not exist[5].

4) Access: In general accesses are the interaction between domains and their shared resources. They define the way data can flow through the separate domains. In the CDS Framework accesses are limited to read, write, and readwrite. SELinux allows for much more flexibility as mentioned in section II. The higher level accesses used by the CDS Framework must be mapped to multiple SELinux permissions in order to capture the intuitive meaning of the access granted. When dealing with a file for example, write in the CDS Framework maps to both random access writing and appending to a file in the actual SELinux policy. Different resource accesses will mean different mappings. The mappings for a socket will be different than the mappings for a file [5].

5) Base Domains and Base Resources: The CDS Framework allows a user to use the concept of Base Domains and Base Resources to interact with the SELinux Reference Policy[5]. An SELinux Reference Policy is a generic system policy for SELinux that all the custom policies in the environment are built upon[9].

6) Entrypoints: An Entrypoint is a concept unique to the CDS Framework. It is a way for the user to define what is known in SELinux as a domain transition. A domain transition allows the domain type of a process to change during execution, giving it more or less privileges. A classic example is a user of the SELinux enabled system imputing a password when prompted. If the password is correct, the process is promoted to a higher access level.

The entrypoints in the CDS Framework consist of a source domain, a target domain, and the entrypoint itself which is positioned between the source and target in the graphical editor[5].

V. THE BENEFITS OF USING THE CDS FRAMEWORK IDE

For many the primary reason for using the CDS Framework IDE is the ability to take advantage of the higher level abstraction that the GUI based editor provides. As

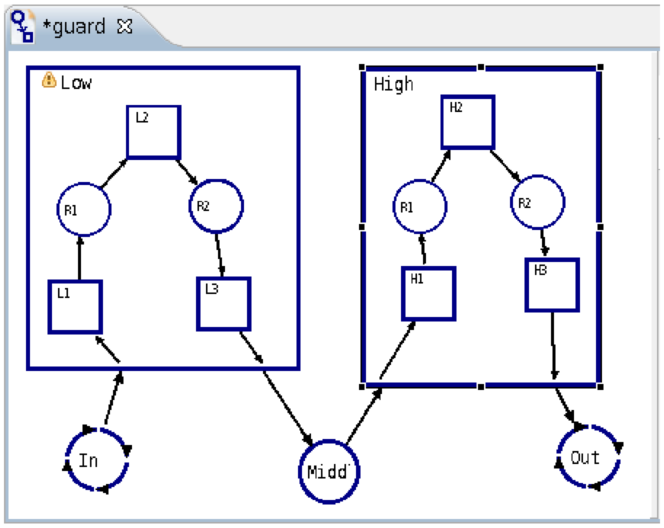


Figure 2: Example of CDS Framework Project[5]

shown in figure (2), a user can create a very intuitive CDS that will be directly translated into an SELinux policy. In figure (2) we can very quickly grasp how data flows through the defined accesses (the arrows) as it passes from resource to resource and from domain to domain. The flow of information starts from the network resource (denoted by the circular arrows) labeled “In” into the domain (represented by a square) labeled “Low”, first flowing to sub-domain “L1”, then the shared resource (circle) “R1” then to domain “L2” and so on until it reaches the resource shared between the two main domains “Midd” which then passes the data into the parent domain “High”. The information travels through High's various sub domains and internal shared resources until the data exits the CDS through the network resource “Out”[5]. So as the reader can plainly see with just a simple key decoding the meanings of the symbols we can ascertain at a glance how the data flows through the CDS.

The ease with which a system designer can understand the language of the CDS Framework IDE leads us to the next major reason people are interested in using the CDS Framework. A system administrator can begin using the Framework to create and edit complex SELinux policies without an in depth or extremely detailed understanding of the nuances of policy manipulation[5]. Many people who are new to the world of SELinux are daunted by the amount of background knowledge needed to begin making the simplest policy changes. With the breakthrough of the CDS Framework IDE the knowledge needed to begin effectively utilizing the power of SELinux is significantly reduced.

VI. HOW TO USE THE CDS FRAMEWORK IDE

A. Overview

The CDS Framework has the familiar look and feel of many popular IDEs. As stated above it is an Eclipse plug-in so anyone familiar with Eclipse, or any modern IDE, should be comfortable working with the CDS Framework. A CDS has to be part of an Eclipse project so a user must create a new project or work from an existing project. Within the project one or more systems will be defined. Once the project and its related systems are setup the user can begin modifying the CDS through the graphical editor before finally compiling it into an SELinux policy binary[5].

B. Installation

As explained above, the CDS Framework is built upon several layers all of which must be installed before the IDE will be usable. I list the required dependencies below. For a detailed description of the installation process please see [10].

- Eclipse 3.4.1 or later
- SLIDE version 1.3.14 or later
- SETools version 3.3.2.2 or later
- Base Reference Policy
- CDS Plug-in[10]

C. Major Components of the IDE and How They are Used

1) Framework Navigator: The Navigator is essentially a hierarchical tree based view of the Framework that the user is manipulating. As shown in figure (3) the framework navigator consists of (1) the Project, (2) the System, (3) the Policy Items, and (4) the Custom Additions folder.

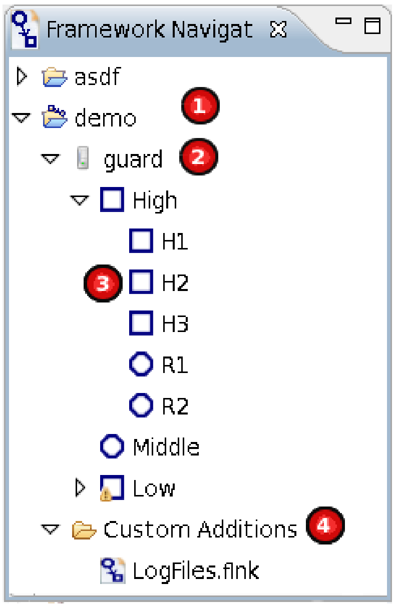


Figure 3: The Framework Navigator[5]

As mentioned in part A of this section the project being modified can contain several systems and each system is made up of all the domains and resources – or policy items – of the CDS. In this example the system "guard" has two parent domains, "High" and "Low". High is expanded and we see its sub-domains and internal shared resources. Low has also been decomposed but the elements remain collapsed within the navigator. The "Custom Additions" folder in the navigator view displays the custom base domains, base resources, and abilities of the members of the domains included in the project. Custom additions are links to the SELinux Reference Policy[5] defined in section IV subsection B part 5.

2) Graphical Editor: The Graphical Editor is presented in Figure (2). It is the work area of the IDE where items from the Palette (discussed below) can be placed and arranged to define the details of the CDS being created[5].

3) Tool Palette: The Palette component of the IDE presents the user with a list of graphical representations of the Framework components mentioned throughout this paper. It also gives the user two modes of selecting the components within the editor. To add a component to the editor, first select a component by clicking on it and then click within the Graphical Editor. The Access and Enter items are used to form connections between the components within the Editor[5]. The Palette is shown in Figure (4) below.

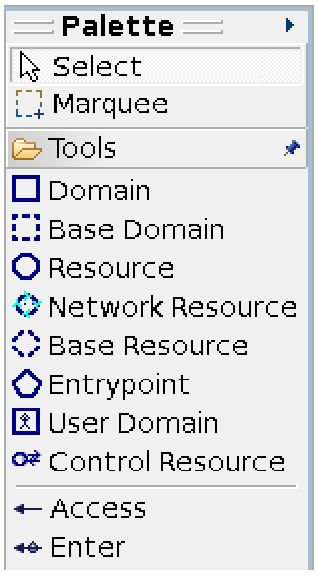
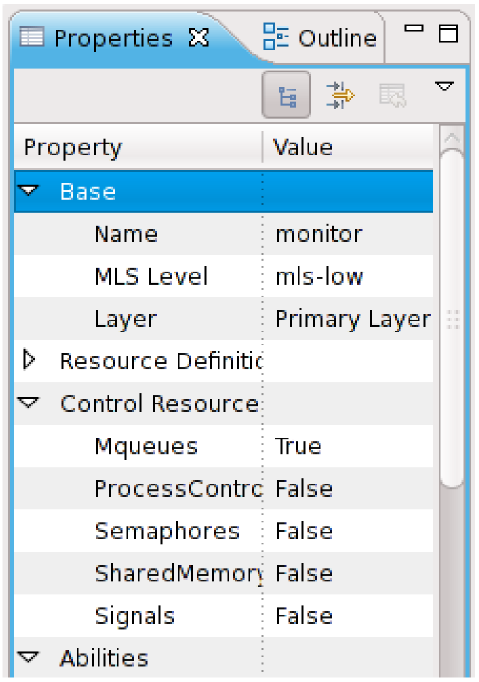


Figure 4: Tool Palette[5]

4) Properties: In Figure (5) the reader can see the Properties View for the IDE. It displays all the properties for any selected component in the Graphical Editor. The item's properties can be edited using this view. As with most IDEs the properties displayed will change depending on the item selected. A user has the ability to select multiple items in the Graphical editor and modify all the properties those items share simultaneously [5].

5) Problems: The Problems view in the CDS Framework IDE functions much like the error outputs available in most advanced IDEs. It displays to the user all errors detected when the user attempts to compile his or her CDS into a binary policy. Selecting an error within the problems view will automatically select the item within the



Property	Value
Base	
Name	monitor
MLS Level	mls-low
Layer	Primary Layer
Resource Definitic	
Control Resource	
Mqueues	True
ProcessContrc	False
Semaphores	False
SharedMemor	False
Signals	False
Abilities	

Figure 5: Properties View[5]

Graphical Editor that is the source of the error. This view also offers a "quick fix" option for some common problems[5]. A sample error output is displayed in Figure (6).

B. Converting the CDS graphic into SLIDE input

The graphical layout of the policy is also represented by a text based Framework Policy (.fpol) file[5]. This file is generated based on the rules within the CDS textual policy language. This text file is the bridge between the visual policy created by the user and the input required by the translator [7].

The CDS Translator takes the text file generated by the CDS Framework IDE and converts it into a SLIDE compatible text file. It does this by referencing the Dictionary (.fdic)[5] file[7].

C. Converting SLIDE file into Binary SELinux Policy

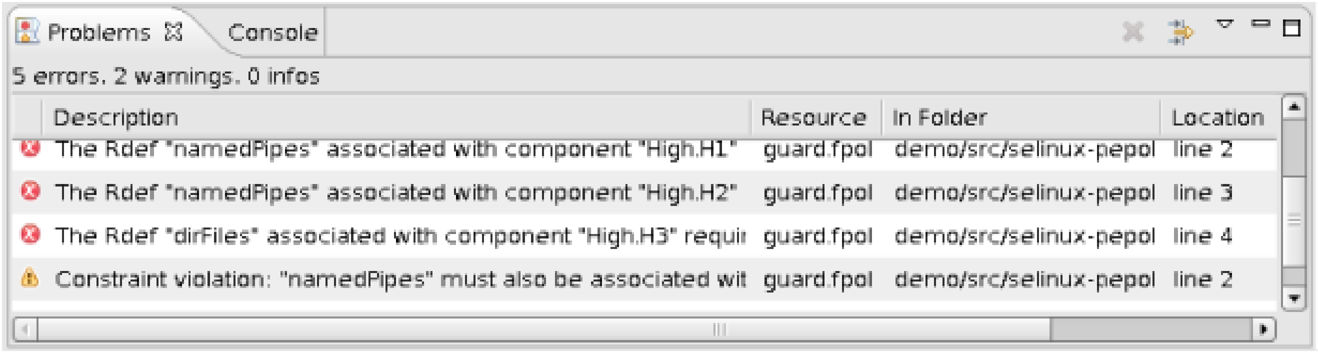
The SLIDE build tools within the SLIDE SDK are responsible for creating the Binary SELinux policy. This is done using the same process as when a designer builds a policy using SLIDE directly. SLIDE calls "make" using the SELinux reference policy which creates the binary policy[7].

An overview of this processes can be seen in Figure (7) on the following page.

VIII. ALTERNATIVES TO THE CDS FRAMEWORK IDE

A. SLIDE

Instead of using the more abstracted CDS



5 errors, 2 warnings, 0 infos

Description	Resource	In Folder	Location
The Rdef "namedPipes" associated with component "High.H1"	guard.fpol	demo/src/selinux-pepol	line 2
The Rdef "namedPipes" associated with component "High.H2"	guard.fpol	demo/src/selinux-pepol	line 3
The Rdef "dirFiles" associated with component "High.H3" requi	guard.fpol	demo/src/selinux-pepol	line 4
Constraint violation: "namedPipes" must also be associated wit	guard.fpol	demo/src/selinux-pepol	line 2

Figure 6: Problems View[5]

VII. HOW THE GRAPHICAL REPRESENTATION OF THE CDS IS TRANSLATED INTO SELINUX POLICIES

A. Overview

The basic concept of how the Framework translates the graphical specifications of the CDS into an SELinux policy is as follows. The CDS Framework converts the layout into a format acceptable to the SLIDE plug-in. This is done via the CDS's translator module. SLIDE is then responsible for compiling this output into an SELinux binary file[7]. I explore the details of this process in the next several sections.

Framework IDE a system administrator might utilize the SLIDE plug-in that the Framework is built upon. The advantage of this approach is that the user gains a finer control over the details of the policy being generated while still having access to all the tools included with a full featured IDE[11].

The disadvantage of this approach is that the background knowledge required to effectively use SLIDE is greater than that of the Framework IDE[11]. The textual input is less intuitive and therefore it may take longer to implement the desired CDS.

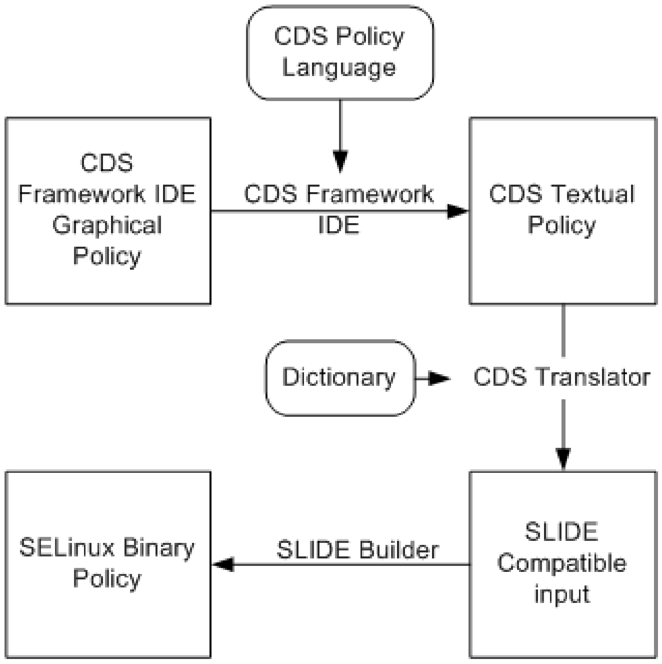


Figure 7: Process of translating CDS Graphic into Binary Policy

B. SELinux Policy Editor (SEEdit)

SEEdit is an alternative IDE that, like SLIDE, is designed to assist users wishing to edit or create their own SELinux policies[12]. Again, with this approach the user gains more control over the details of the policy being created while retaining the beings of using an IDE, but loses the intuitiveness of the CDS Framework. SEEdit does however, provide a layer of abstraction which hides some of the details of raw policy editing. This is done via SEEdit's custom policy editing language SPDL or Simplified Policy Description Language[13].

C. Create or Edit the Policy in a Text Editor

The administrator also has the option of opening the plain text policy in an editor, making changes, and then compiling the policy from the command line using the makefile in the policy source directory[8].

The advantages of this solution are that the administrator has absolute control over exactly what the makeup of the policy will be and does not have to learn how to use a potentially unfamiliar IDE.

The major disadvantage of this approach lies in the difficulty involved. To create a complex CDS using this method would take large amounts of background knowledge and experience in SELinux. In addition the chance for error is greatly increased using this approach. The safety nets the IDEs provide in the form of error and warning outputs are lacking. As with the IDE based approaches the ability to conceptually design a CDS and directly translate it into a policy is lost.

IX. CRITICISMS OF THE CDS CONCEPT

There are some who say that using a CDS is a detriment to the security of the overall system and should be avoided. These critics make the argument that a CDS is contrary to the original idea of Mandatory Access Control (MAC) which tries to force rigid security rules without room for different interpretations depending on the situation. A CDS by its very definition adds additional rules to the secure environment to allow the transfer of data from another domain which may be potentially insecure[14].

Critics point out that implementing a CDS allows the well meaning administrator to push through system modifications while focused narrowly on the current problem at hand, potentially overlooking larger system wide security concerns. The micro view of trying to move data from point A to point B could compromise the macro system and should just be avoided altogether[14].

Towards the center of the issue are people who are not completely apposed to using a CDS, but state that it should only be used as a last resort when there are no alternatives for accomplishing a critical task[14]. If what the administrator is trying to achieve is not critical, or if there is a way of reaching his or her goal within the boundaries of MAC then they maintain that using a CDS should be avoided.

X. CONCLUSION

A. Summary

In this paper I have explored the CDS Framework IDE and shown how someone who is interested in SELinux but may not be an SELinux expert can use the Framework accomplish their goals quickly and safely. I have given an overview of SELinux as well as the concept of the Cross Domain Solution. I have also covered some of the alternatives to the CDS Framework and discussed some criticisms of the CDS concept.

B. Advantages and Disadvantages

The main appeal of the CDS Framework IDE to someone considering implementing a CDS is the fact that the graphics based language is so intuitive. The designer can translate a basic data flow diagram and translate it directly into an SELinux policy without having a huge wealth of background knowledge of SELinux and management of its policies. The major disadvantage to using the Framework is that an administrator may not be able to make use of the fine grain control that SELinux has to offer.

C. Future Work

1)Allow for Finer Access Controls: One of the major disadvantages of using the CDS Framework mentioned in this paper – the lack of fine tuned control - could be mitigated significantly by giving the access objects in the Framework the ability to use all the accesses available in SELinux. For example, rather than just read and write ability an object could have the ability to read and append only.

2)A Framework Policy Analysis Module: The creators of the framework have suggested adding a Policy Analysis Module that will guide the policy creator during the editing process and inform him or her as to whether the policy being created conforms to relevant security principles[15]. This addition would lead to the creation of a more secure system for less experienced users.

3)Macros for the Policy Editor: The developers have also put forth the idea of adding a Macro recorder to the IDE so that repetitive actions necessary to create a policy can be captured once and then repeated at the discretion of the user[15].

4) Automatic Graph Layout: Another idea for the IDE is to allow the user to select an automatic layout for the diagram that has been created[15]. Often when making a complex policy the layout of the diagram can become cluttered and difficult to decipher. The automatic layout feature would allow the user to potentially skip the arduous process of attempting to rearrange all of the elements into a more aesthetically pleasing configuration. The IDE would attempt to rearrange the graph based on simple user input such as "circular graph," "spiral," or "tree". The IDE would then apply a layout algorithm to the existing graph. This could speed up the policy creation process even further and lead to more clear and reusable CDS layouts.

REFERENCES

[1] "What is SELinux," SELinux Project .Org, [Online]. Available: http://selinuxproject.org/page/Main_Page [Last Accessed: Dec. 6 2009].

[2] D. Rusling, "The Linux Kernal," The Linux Documentation Project, [Online]. Available: http://tldp.org/LDP/tlk/ipc/ipc.html [Last Accessed Dec 7 2009]

[3] K. MacMillan, et al., "Lessons Learned Developing Cross-Domain Solutions on SELinux," treysys.com, March 2, 2006. [Online]. Available: http://www.tresys.com/pdf/Lessons-Learned-in-CDS.pdf. [Last Accessed: Dec. 5 2009].

[4] "CDS Framework IDE," [Online]. Available: http://oss.tresys.com/projects/cdsframework/. [Last Accessed: Dec. 5 2009].

[5] Tresys Technology, LLC, "CDS Framework Toolkit Documentation," treysys.com, May. 7, 2009. [Online]. Available: http://oss.tresys.com/projects/cdsframework/chrome/site/helpfiles/webdocs.html. [Last Accessed: Dec. 6 2009].

[6] "What is Eclipse?" Eclipse.Org, 2009 [Online]. Available: http://www.eclipse.org/home/newcomers.php [Last Accessed: Dec. 8 2009].

[7] Personal Email Communication, Dave Sugar, Dec 12, 2009. Tresys Technology, LLC.

[8] F. Coker "Guide to Writing SELinux Policy," Linuxtopia.Org, 18 March 2004. [Online]. Available: http://www.linuxtopia.org/online_books/writing_SELinux_policy_guide/index.html [Last Accessed: Dec. 6 2009].

[9] C. PeBenito, F. Mayer, K MacMillan ., "Reference Policy for Security Enhanced Linux," Tresys Technology, 2006. [Online]. Available: http://selinux-symposium.org/2006/papers/05-refpol.pdf. [Last Accessed: Dec. 5 2009].

[10] "CDS Framework Download and Installation Instructions," [Online]. Available: http://oss.tresys.com/projects/cdsframework/wiki/download. [Last Accessed: Dec. 5 2009].

[11] "SLIDE Introduction," [Online]. Available: http://oss.tresys.com/projects/slide/. [Last Accessed: Dec. 5 2009].

[12] Y. Nakamura and Y. Sameshima, "SELinuxfor Consumer Electronics Devices," Linux Symposium, July 23rd, 2008. [Online]. Available: http://ols.fedoraproject.org/OLS/Reprints-2008/nakamura-reprint.pdf. [Last Accessed: Dec. 5 2009].

[13] "SELinuxPolicy Editor," SEEdit, Aug 27, 2008. [Online]. Available: http://seedit.sourceforge.net/ [Last Accessed: Dec. 6 2009].

[14] "Cross Domain Solutions," Academic dictionaries and encyclopedias, [Online]. Available: http://en.academic.ru/dic.nsf/enwiki/3986437 [Last Accessed: Dec. 6 2009].

[15] "Active Tickets," Tresys Technology[Online]. Available: http://oss.tresys.com/projects/ cdsframework/report/8. [Last Accessed: Dec. 12, 2009].