The Design and Implementation of a Guard Installation and Administration Framework

Rev 1.0
26 Jan 2007

Boyd Fletcher
USJFCOM J9 &
SPAWAR Systems Center San Diego
boyd@spawar.navy.mil

Chris Roberts
General Dynamics
christopher.roberts@je.jfcom.mil

Kurt Risser
Dataline
kurt@risser.net

Abstract

The Guard Installation and Administration Framework is a set of applications and processes to reduce the development costs for building installation and maintenance subsystems for SE Linux based cross domain guarding solutions. This paper discusses the issues with the development of SE Linux based guards and our solutions to them.

1. Overview

In Spring of 2004, USJFCOM J9 started the Cross Domain Collaborative Information Environment (CDCIE) to develop a standards based, non-proprietary, open source, secure, scalable collaborative information environment (CIE) to enable cost-effective information sharing in both single and cross domain environments. The CDCIE consists of the following capabilities:

1. Cross Domain Portal and Portal Applications
   - Provide a portal and suite of commonly used portal applications that are classification labeling aware
2. Cross Domain Collaborative Tool
   - Provide a secure and scalable collaboration tool for DoD that solves the tactical chat, cross domain, full functional (minus video) collaboration requirements
3. Security Enhanced Office Automation Suite
   - Provide a method to safely redact documents for release to lower classification levels & external entities.

To implement these components we decided to develop a number of special purpose trusted gateways to front end an existing general purpose XML Guarding Solution. We[i] are developing the following gateways:

1. Collaboration Gateway (CG) – Bi-directional cross domain XMPP based text chat with language translation and whiteboarding
2. Web Services Gateway (WSG) – Bi-directional cross domain XML/SOAP based web services with or without human review
3. Streaming Data Gateway (SDG) – Bi-directional transfer of streaming audio data.

These gateways provide user account management, a highly scalable platform for user (or externally) facing services, protocol translation and termination, and a second layer of a three-layer defense in depth strategy[ii]. To implement the gateways we needed an operating system with the following capabilities:

1. Ability to run Java very efficiently
2. Capable of being used in a trusted computing environment
3. High degree of installation configurability including the ability to build custom installation DVDs.
4. Ability to build a guard based on Assured Pipeline design.

We evaluated a number of trusted operating systems and determined that SE Linux best met our needs. We chose SE Linux for the following reasons:

1. Type Enforcement (TE) and its ability to do Assured Pipelines simplifies guard development. Assured Pipelines are typically implemented as series of content filters that are staged together in sequence with single entry and exit points. The one-way information flow in the Assured Pipeline is enforced by the operating system kernel and is non-by-passable. The content filters tend to be much smaller applications than are developed in

traditional trusted operating systems, which make it easier to evaluate the correctness of their behavior.

2. Most of the other available trusted operating systems base their architecture on the work of Bell, LaPadula and Biba (BLB). We felt that the BLB approach was more suitable to building Multi-Level Security (MLS) workstations, sometimes called compartmented mode workstations (CMW), then in building guards. Guards need to pass data between networks (or systems) of different classification levels. This breaks the Bell, LaPadula and Biba security model.

For our solutions we chose the Red Hat Enterprise Linux (RHEL) AS version of Linux because of its Common Criteria EAL 4[iii] evaluation and robust support for SE Linux. During the development of these trusted gateways we have developed a Guard Installation and Administration Framework to facilitate the installation and maintenance of a guarding solution. Our guard framework attempts to solve the following problems:

1. Installation Complexity – Most existing guarding solutions take a day or longer to install and configure. We had a goal of 30 minutes from insertion of the DVD to system being operational. Installation must be automated and should minimize data input to only site specific configuration data.
2. Unfriendly User Interface - The typical Department of Defense (DoD) system administrator is not a Linux/Unix expert. Access to the command line increases the risk that the system could be put into an insecure configuration. Guards should provide a "Windows-like" user interface.
3. Lack of a Common Administration Interface - Users are familiar with a Microsoft Management Console (MMC) type of approach for system administration. Most guards use a combination of separate tools for administration of the system.
4. Lack of Centralized Logging – Existing logging systems on Unix or Linux use difficult to secure communications methods (like sockets, common files, shared memory). They also have a tendency to use a large number of log files, which makes troubleshooting and security log inspection time consuming and error prone.
5. Lack of low level Linux support in Java - Java lacks, and for good reason, many of the low level functions to manipulate the operating system that are required for secure communications within a guard.

2. Software Installation

During the development of our first guard we identified the following software installation related problems:

1. Lack of ability to automatically mount a CDROM/DVD in the Linux installation change root environment during the operating system installation. This is required so that customized software can be installed and configured during the operating system installation.
2. Running applications during the installation that required user feedback proved very problematic because the not all ttys created during the installation process were created with the proper terminal settings.
3. Lack of an extensible installation wizard framework to prompt users for configuration information.
4. Lack of good Operating System security lockdowns (covered in section 3.5)

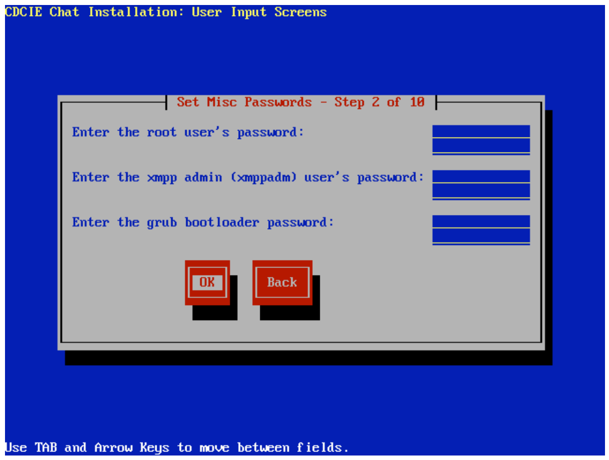
Our custom installation process starts by inserting a DVD into the computer and rebooting the system. If the hardware is configured to boot from DVD the installation will automatically start. The user will be prompted to verify if they really want to start the installation and then the user will be prompted to agree to a license agreement or disclaimer. At this point the disk is repartitioned and RHEL is installed. Once the operating system has been installed, a change root environment (the new o/s' root directory) is available to install and configure the software. It is at this point where we run into our first problem – the change root environment is unable to access the DVD drive.

In order to mount the DVD in the installation change root environment, you essentially have to force a MAKEDEV on the SCSI and IDE device trees then force and mount on all the created devices until you find one that succeeds and contains a file that is only found on the installation DVD. We implemented an inline Perl script in the ks.cfg that implements the above process. Once the correct DVD device is identified and the DVD mounted, the processing of the ks.cfg continues non-interactively until the custom installation wizard is started.

Normally during a RHEL installation the user interacts with the ttys 1-3. TTY 1 is used as the primary display and interaction terminal, tty2 is used to display any errors during the installation, and tty3 is used for any custom scripts. However, tty3 is not created as a login terminal and during the development of our custom installation wizard we found that the console font and terminal settings used in the non-login did not support ASCII line art. Since the Perl/Newt based installer uses ASCII line art to present a familiar interface to the user we were forced to create a new tty as a "login" terminal in the ks.cfg (Kickstart Configuration File) so that the correct terminal settings were set. To accomplish this, we did the following:

1. Installed the open-1.4.rpm package
2. Switched to TTY 9 (which was not used):
   ```
   chvt 9
   ```
3. Created the shell as a login shell and ran our installation program (Perl/Newt):
   ```
    open -c 9 -s -w -l -- bash -l -c
   "/mnt/cdrom/cdcie/cdcieinstall.pl"
   ```
4. Created a tty for displaying the error messages from the installer:
   ```
   open -c 8 -l -- tail -f
   /cdcie/log/cdcie_install_log.txt
   ```
5. When the custom Perl/Newt installer exited, we return the user to tty 3:
   ```
   chvt 3
   ```

Once the proper ttys are created, the Perl/Newt based custom installer starts. This installer is designed to be highly customizable and can be easily extended to support any number of configuration screens. From the point of view of the operator it behaves like an installation wizard. Those familiar with installing applications on MS Windows will be familiar with this approach. It is designed to be easy and efficient to use. It is however text based (to support all classes of hardware devices). The Perl/Newt combination are particularly well suited for installers because Perl provides exceptional text processing capabilities (useful for editing configuration files) and Newt provides an easy to understand API syntax for developing text based user interfaces that resemble graphical user interfaces. The installer comes with a wide assortment of input validation modules including ones for strong passwords, IP address field validation, hostname field validation, and LDAP field validation. In order to make sure the strength of the users password matches the operating system's strength characteristics the installer actually applies the new password to the users' accounts before switching to the next page. The installer includes canned screens for settings users' passwords, setting the system time, and setting the base classification of the device. Lastly the installer stores the configuration values in a key/value based configuration file that is then used by a number of scripts in the ks.cfg to configure things like the firewall, network configuration, and custom application settings. After the user exits the installer, the rest of the installation process is non-interactive.


CDCIE Chat Installation: User Input Screens

Set Misc Passwords - Step 2 of 10

Enter the root user's password:

Enter the xmpp admin (xmppadm) user's password:

Enter the grub bootloader password:

Use TAB and Arrow Keys to move between fields.

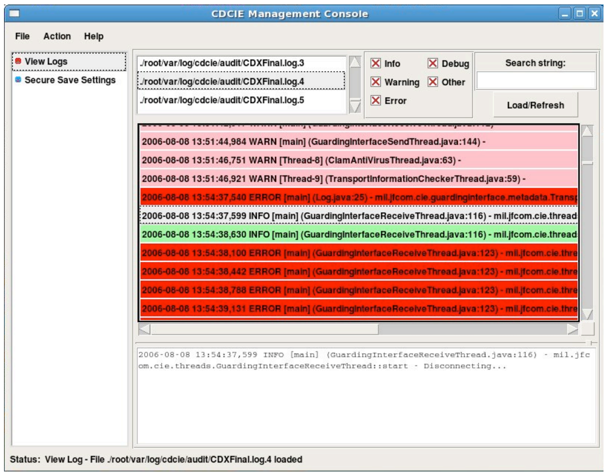
Figure 1 - Screenshot of the Custom Installation Wizard

3. Administration Framework

While researching the requirements for our cross domain guarding solution we examined the installation and maintenance procedures of a number of existing solutions. One of the common deficiencies we found in those systems was the lack of consistency in the guard administration applications. Existing guards typically required the administrator to use the command-line, text based user interfaces, and graphical interfaces all on the same device. Additionally, we had discussions with a number of organizations using cross domain solutions and found that in the DoD today most organizations have a high system administration proficiency in MS Windows and a low to moderate proficiency in Linux/Unix.

3.1 Guard Management Console

In order to reduce the administrative complexity of the guard, we developed an extensible Perl/Tk based administration framework called the Guard Management Console (GMC). The GMC enables guard developers to rapidly build administrative graphical user interfaces for guards. The GMC implements per user (role-based) security by mapping, in its XML based configuration file, which users can access which management modules (plugins). SE Linux's Type Enforcement is then used to implement mandatory access control (MAC) on management modules that are accessible to each user or role.



```
./root/var/log/cdcie/audit/CDXFinal.log.3
./root/var/log/cdcie/audit/CDXFinal.log.4
./root/var/log/cdcie/audit/CDXFinal.log.5

2006-08-08 13:51:44,984 WARN [main] (GuardingInterfaceSendThread.java:144) -
2006-08-08 13:51:46,751 WARN [Thread-8] (ClamAntiVirusThread.java:63) -
2006-08-08 13:51:46,921 WARN [Thread-9] (TransportInformationCheckerThread.java:59) -
2006-08-08 13:54:37,540 ERROR [main] (Log.java:25) - mil.jfcom.cie.guardinginterface.metadata.Transp
2006-08-08 13:54:37,599 INFO [main] (GuardingInterfaceReceiveThread.java:116) - mil.jfcom.cie.thread
2006-08-08 13:54:38,630 INFO [main] (GuardingInterfaceReceiveThread.java:116) - mil.jfcom.cie.thread
2006-08-08 13:54:38,100 ERROR [main] (GuardingInterfaceReceiveThread.java:123) - mil.jfcom.cie.thre
2006-08-08 13:54:38,442 ERROR [main] (GuardingInterfaceReceiveThread.java:123) - mil.jfcom.cie.thre
2006-08-08 13:54:38,788 ERROR [main] (GuardingInterfaceReceiveThread.java:123) - mil.jfcom.cie.thre
2006-08-08 13:54:39,131 ERROR [main] (GuardingInterfaceReceiveThread.java:123) - mil.jfcom.cie.thre

2006-08-08 13:54:37,599 INFO [main] (GuardingInterfaceReceiveThread.java:116) - mil.jfc
om.cie.threads.GuardingInterfaceReceiveThread::start - Disconnecting...

Status: View Log - File ./root/var/log/cdcie/audit/CDXFinal.log.4 loaded
```

Figure 2 - Screenshot of the Guard Management Console's Log Viewer Module

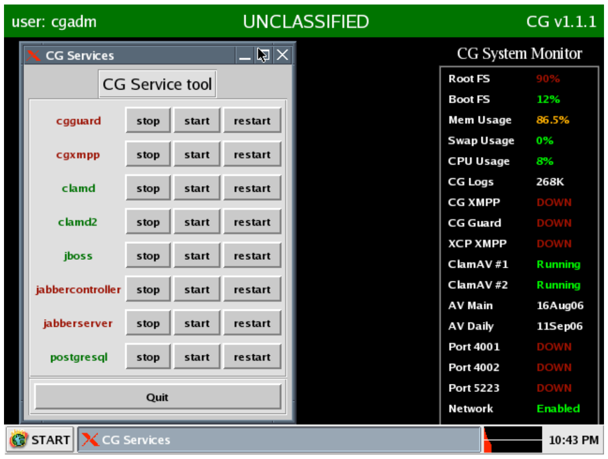
The GMC includes user account and certificate management, log file viewer (with support for log4j and standard Unix log files), system monitoring, service start/restart/stop panel, anti-virus file updates for ClamAV, and system backup modules. We chose Perl/Tk for a number of reasons including:

- Perl/Tk is a robust toolkit for rapidly creating graphical user interfaces (GUI)
- Perl is exceptional at manipulating text files (i.e. configuration files), which is a common function on guards.
- Perl was developed on Unix/Linux and has been used extensively for years as a scripting tool for Unix/Linux systems.
- There is a huge repository of Perl Modules covering everything from databases to XML to LDAP.

The code snippet below is the basic structure of the required elements of a GMC module.

```
package "ModuleName";
sub new {
        # This registers the module with the GMC.
When called the module will become available on the
left side module selection screen.
}
sub load {
        # This loads the module into the GMC and dis-
plays it in the right side of the GMC window.
}
sub unload {
        # This will unload the module from the GMC.
}
sub help {
        # This will return a help dialog for display
to the user
}
-1
```

The GMC also includes a separate Status Monitor that displays current disk usage, CPU load, memory usage, and specific TCP/IP port bind statuses. This status monitor is displayed on the background of the user's desktop. The status monitor is intended to provide the system administrator with a quick and easy to understand overview of the current operational status of the system. The status monitor can easily be modified to display the status of different ports and processes.



CG System Monitor

Root FS	90%
Boot FS	12%
Mem Usage	86.5%
Swap Usage	0%
CPU Usage	8%
CG Logs	268K
CG XMPP	DOWN
CG Guard	DOWN
XCP XMPP	DOWN
ClamAV #1	Running
ClamAV #2	Running
AV Main	16Aug06
AV Daily	11Sep06
Port 4001	DOWN
Port 4002	DOWN
Port 5223	DOWN
Network	Enabled

Figure 3 - Screenshot of the Guard Management Console's Status Monitor (on right)

In order to reduce exposure to the command-line all users except root are automatically placed in a graphic user environment on login and we used the MS Windows-like JWM window manager to implement a user interface that many users are already familiar with. A logoff/shutdown/restart capability has been configured prompt the user to enter a reason for rebooting or shutting down the system. This reason is logged to system log file.

3.2 Backup & Recovery

Backup and Recovery is a critical capability that all production systems must have and the CDCIE gateways were no exception. However, SE Linux, or trusted systems in particular, poses some interesting problems for backup and recovery. In typical production server environment, sites implement two levels of backups: full and incremental (or differential). Incremental backups present a number of problems for guards including:

- reduced reliability if one of the incrementals in the chain fails since the last full backup;
- doing a partial system restore to a production system potentially places the system into a unknown/unstable configuration since other parts of the system have been changing since the incremental backup was completed.

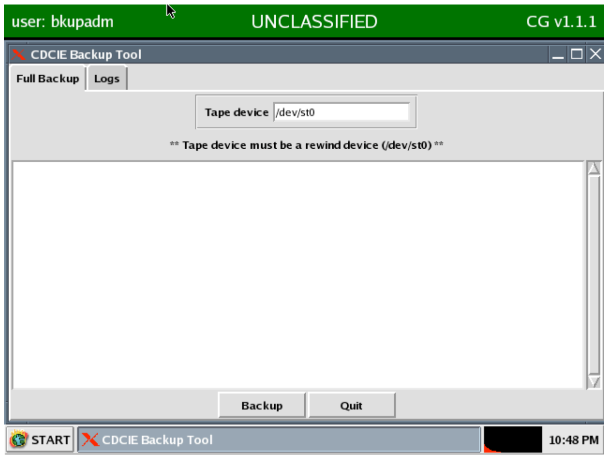


Figure 4 - Screenshot of the Guard Management Console's Backup Tool

The GMC Backup module allows a user to start a full backup of the entire system and view the backup logs for previous backup jobs. Since the GMC Backup module only supports full system dumps the operating system cannot be running during system recovery. Recovery operations require a booting from the installation DVD media with a recover option specified. Once the kernel has started, the user is asked select which device to restore from and to confirm start of the recovery operation. Currently the GMC Backup module supports backup to tape only. Upon initial reboot after a system restore the SE Linux file labels are reapplied to the system.

3.3 Software Integrity

There are three aspects of software integrity management in cross domain solutions:

1. Managing of the integrity of communications with the guard
2. Verifying the integrity of the operating system and application files
3. Verifying the integrity (or authenticity) the distribution media.

The GIAF provides solutions for the later two. After completion of the DVD build process an md5 checksum is created of the ISO. This checksum is then provided to anyone receiving (via DVD or downloaded from our website) the media to verify that the media received is the original copy. This md5 checksum can also be provided to certification and accreditation authorities to provide a unique ID record of the contents of the DVD to make sure that installations of the guarding solution are only using an officially approved version.

During the DVD build process an md5 checksum is generated for all binaries on the system and the checksum file is written to the disk to be included in the final DVD image.

In order to verify the integrity of the system, the system is rebooted using the DVD with the integrity option specified at the boot prompt. The md5 checksums of all binaries on the system is compared is the record store on the distribution media.

The integrity system has the ability to write md5 checksums of all configuration files on the system to either floppy or USB Drive. During the integrity checking process you have the option to either generate new configuration file checksums or verify that the existing ones match those stored on external media (floppy or USB Drive).

3.4 Centralized Logging

In a production server environment, logging of system state changes is important; in a cross domain solution it is critical. Unfortunately, in most Linux systems the two most common methods of logging are not well suited for guards. In traditional Linux, most systems either log directly to a log file or write via sockets to syslogd. SE Linux has problems with enforcing a true one-way data flow using sockets so a possibility of back channels exists. Writing directly to log files has concurrency and locking problems if more than one process writes to the log file, so many developers just use individual log files. While individual log files are definitely more secure, separate log files can make troubleshooting technical problems or doing forensics after a security incident very time consuming and error prone.

So to solve both of these problems we have implemented the concept of a trusted central logging daemon

(CLD). The CLD is based on the Apache Log4J project's Simple Socket Server. But instead of using sockets for communication, we have created a new Log4J appender class that uses System V Message Queues to guarantee (via SE Linux policy) that the communication between multiple processes and the logging daemon is a one-way connection. The assured pipeline processes have write access to the Message Queue, while the CLD only has read access.

The CLD benefits include:

1. Log4J and its C, C++, Perl and .Net counter parts are extremely popular in the developer community
2. The Apache Logging infrastructure supports a large number of logging features like log rotation, custom log file naming, log appending, logging to a database, customizable log file formats, etc…
3. The Apache Logging infrastructure supports multiple levels of debugging from FATAL to DEBUG and can be extended with custom levels. The CLD is configured to always log FATAL and ERROR messages. This guarantees that critical system stability (FATAL) and security (ERROR) messages are always logged.

CLD clients use the custom Appender to log messages, which are placed on the System V Message Queue. The actual Central Logging Daemon is a separate process that takes log messages from the System V Message Queue, then writes them via a standard log4j RollingFileAppender to a text file on the file system.

Messages to be logged may be objects rather than simple text. Whenever the logger is called with an object, the content, if any, is written to the filesystem. The content is persisted as a file to a designated area on the file system according to the logging level used when the message was logged. Objects logged with severe error levels are persisted to a separate directory from those logged with informational levels. A custom log4j Appender class is used by the CLD to accomplish this. Typically these are large binary objects like files that are being transferred.

SE Linux is used to control what actions the CLD can to do. These actions include granting the ability to create log files, ability to append to log files, and the ability move the active log into the archive directory. The CLD is not allowed to delete or truncate a log file. So now that there is a common logging subsystem for the assured pipeline filters and the multitude of other applications on the system, we are able to create a GMC logging module that not only can read the normal Linux syslog files but can also read Apache Logging infrastructure based log files. The logging view module also has the ability to save the log files off to floppy disk or USB Drive and delete archived logs. SE Linux policy is used to ensure that only the deletion of rotated logs is allowed.

Some examples of how to use Log4J and the CLD are below:

An example of logging a simple text message:

```
Logger log = (Logger)
  Logger.getLogger(App.class);

log.info("Application initialized success-
fully.");
```

An example of logging a document object and an associated message:

```
Logger log = (Logger)
  Logger.getLogger(App.class);

TransportInformation ti =
  new TransportInformation();

ti.setStatusMessage("This TransportInformation
object includes a secure document as DATA.");

ti.setData(CDGLogHelper.readFileIntoByteAr-
ray("confidential.doc"));

log.warn(ti);
```

3.5 User Account and Certificate Management

The GMC has modules for managing a number of aspects of user account and certificate management including:

1. Resetting of users passwords
2. Management of user certificates in a Java Key Store. This includes importing and exporting of certificates from/to USB Drive or Floppy.

A specific user password management administrative user conducts passwords resets. The root user can reset the password for the password management user.

3.6 Operating Systems Lockdown

Any guard installation would not complete without a comprehensive lockdown of the core operating systems. During the installation process, we have tried to reduce the number of places that a human could make a mistake and have eliminated human interaction in all portions of the actual software installation and security lockdown. The system is installed in a secure configuration whether you want to or not. The operating system security lockdown is derived from the security requirements and lockdown guidance from government sources, Center for Internet Security, Bastille Linux Project, and number of other external sources and focuses on a number of key areas including:

1. Installation of only required subsystems
2. Removable of subsystems required for installation but not for operation
3. Installation and configuration of iptables – the Linux firewall.
4. Configuration of user account and password policies
5. Setting of file and device permissions to more conservative settings.
6. Disabling of unneeded or insecure functions (or services) like: disabling DNS lookups, talk, inbound ping, tftp, most of inetd, Finger, automount, portmapper, etc…
7. Binding internally facing servers to the loopback address so that they cannot be attacked externally. Sometimes iptables was used to "enforce" this binding to the lookback address.

The GIAF provides several scripts that implement the above and additional lockdowns.

3.7 Java-To-UniX Library

During the architecture discussions for the development of the CLD and our Web Services guard, it was decided that the most secure and efficient method for communicating between processes (i.e. in the assured pipeline) was to use System V Message Queues. However Java lacked the capability to interact directly with System V Messages Queues. After some investigation into the problem, it was decided to use the open source Java-To-UniX (JTUX) library. However JTUX did not fully support System V Message Queues nor did it support RHEL 4.1 so a number of modifications were made to it to add that support. When using System V Message Queue in assured pipeline or just as secure communications mechanism as in the CLD, the Queues themselves should be created by an external process during system boot-up to avoid the possibility of a back channel in queue creation.

4.0 Conclusions

4.1 GIAF Conclusions

During the development of CG and our other gateways, we have found that the GIAF has significantly decreased the time required to create administrative user interfaces from weeks to days. The use of an automated installation process, even early-on in the development process, has allowed us to do full system testing earlier in the development cycle and conduct more accurate unit level testing since the applications are being deployed into a realistic (very close to end state) environment.
During the installation of a guard, we are required to go through a process called Security, Testing, and Evaluation (ST&E). This process is done at each site where a guard is installed. Because the GIAF automates all of the guard's installation and security lockdown, we are able to cut the ST&E from a typical week to around 2 days with most of that focused on site-specific configuration.

Additionally, because the GIAF automates so much of the installation process, it makes our installations very repeatable in a highly consistent manner. This allowed the security testers of the system to spend their time looking at the security implementation instead of fighting to get and keep the system up and running.

The GIAF is constantly being improved as USJFCOM J9 develops its next generation of guards. Portions of the GIAF (minus the GMC) were used in the CDCIE Chat Collaboration Gateway when it completed Certification, Test, and Evaluation (CT&E) in October of 2006. The GIAF will be used in CDICE Chat version 2.0 and in our upcoming Web Services and Streaming Data Gateways.

4.2 SE Linux Conclusions

Early on in the project we decided, due to the complexity of SE Linux policy development, to contract out the policy development to a software engineering firm that specializes in that type of work. This proved to be good decision for several reasons: it reduced out development costs and timeline since we did not have to spin-up our developers to be SE Linux Policy experts and the outside engineers were able to objectively look at our design and point out design flaws before the code had been written. We did, however, send our engineers to SE Linux class so that they would be able communicate efficiently with the policy developers,

During the development and testing of the guard's policy, we found that a strong understanding of Unix/Linux inter-process communications and low-level system calls is critical to debugging of policy files. Initially this proved to be somewhat of a problem for us since most of our developers are Java programmers and they generally do not interact with the operating system at such a low-level.

Luckily the SE Linux community realized that policy development was too complex and has undertaken a number of projects to simplify its development. One of the first successful projects to appear is called Reference Policy. Work is currently underway to convert the Collaboration Gateway's policy to Reference Policy and all the gateways currently under development are using Reference Policy.

One interesting side effect of using SE Linux is that we now have a much better understanding of how other people's applications along with our own interact with the operating system. When developing in virtual machine environments like Java, developers sometimes do not realize the amount of low-level interaction with the operating system that is really going on. This increased understanding of the interactions contributes significantly to the development of more efficient and secure programs. Through the use of SE Linux policy debugging, we have also found that a number of commercial and open source applications are not very well behaved. For example, many applications frequently access files read/write (like /etc/passwd) when they only need read. We frequently denied access to a variety of systems calls in these applications, just to determine whether it would have any negative affect – many times the application had no change in execution behavior with the restrictions in place.

So after spending the last three years successfully developing guarding solutions using SE Linux, we have come to the following conclusion about SE Linux: The ability to very tightly secure the communication flows between processes and isolate the processes from one another was absolutely critical to the successful completion of the CT&E for our CDCIE Chat solution.

5.0 References

Center for Internet Security - www.cisecurity.org
Tresys SE Linux Information - www.tresys.com/selinux
NSA SE Linux Information - www.nsa.gov/selinux
Comprehensive Perl Archive Network – www.cpan.org

[i] The "We" includes USJFCOM J9, Air Force Research Lab, Trident Systems, General Dynamics, Dataline, EG&G Technical Services, and Tresys.

[ii] The Defense in Depth layers for CDCIE consist of a Firewall, a Gateway, and an XML Guard. There is a firewall and gateway in each classification domain.
[iii] Government agencies are required to use Common Criteria evaluated operating systems. http://niap.nist.gov