Setroubleshoot: A User Friendly Tool to Diagnose AVC Denials

John Dennis
Red Hat

Abstract

Practical experience with the deployment of SELinux has shown it is often disabled in the field negating its promise. Anecdotal evidence suggests software developers and end users consider it difficult to use, comprehend, and the root cause of mysterious failures. Perceived as productivity barrier the expedient solution is to disable it. A new tool has been developed which exposes AVC denials in real time and interacts with a user presenting information in a friendly manner explaining the current denial and suggesting possible solutions. Replacing the hidden, often obscure failures in software which only occur when SELinux is enabled with friendly notification of SELinux's actions will improve the use experience fostering adoption.

1 Introduction

The adoption of SELinux along with its potential to vastly improve computer security has been frustrated by the perception it is more an impediment than a solution. Software developers, system administrators, and end users quickly learn enabling SELinux will introduce anomalous behavior into an otherwise functioning system. This arises when SELinux denies access via an AVC denial as part of its Mandatory Access Control (MAC). Unfortunately these denials are typically hidden or obtuse when discovered.

AVC denials can occur because the configuration of the security policy does not match how the software is being used, because there are bugs in the security policy, because of problems in the software requesting the resource, or because a genuine intrusion was thwarted. SELinux is still a young technology and experience has shown most AVC denials during this infancy period have been the result of one of the first three causes which can be mitigated provided they are known about.

The AVC denial manifests itself as a software failure which in practice is either not reported or reported incorrectly. This is because most current software and users are only aware of the traditional UNIX Discretionary Access Control (DAC), otherwise generically known as “permissions”. With luck the AVC denial may be reported as a permission error by the failing software. However the problem investigator will often be confounded by the realization there is no DAC permission problem. It is just as likely the software will fault with no error messages compounding the frustration. During problem investigation the advice may be offered to disable SELinux with the consequence the system is restored to a functioning state.

The actual cause of the problem may have been recorded in a system log as an AVC message such as this:

```
denied  { execmod } for  pid=14366
comm="acroread" name="AcroForm.api"
dev=hda2 ino=5619873 scontext=user_u:sys-
tem_r:unconfined_t:s0 tcontext=system_u:ob-
ject_r:lib_t:s0 tclass=file
```

Very few people after a mysterious failure are trained to scan log files for AVC denial messages. Even less have the technical expertise to interpret the message and translate it into a comprehensible understanding of the problem leading to a solution. The practical consequence of the hidden nature of the failures and the obtuse audit report is to disable SELinux because it is perceived to be more of an impediment than a valuable security enhancement.

2 Problem Solution

We recognized a need to bring AVC denials out of their hidden recesses into a much more visible and immediate form. It would be a clear advantage if the AVC denial was visibly displayed at the moment it occurred, this helps users correlate a software failure they are encountering with the AVC denial that triggered it. We also recognized simply reporting the AVC denial, while an improvement over just log file entries, was still insufficient to help users. It would also be necessary to perform automated analysis of the AVC denial in the context it occurred in order to explain to the user in friendly terms what occurred and what a possible solution might be. We also wanted an alert mechanism should there be a genuine security attack in progress which SELinux was actively thwarting in order to present to the user in real time a comprehensible interpretation of the violations.

We established several overarching goals:

- Plug-in architecture for analysis modules
- Flexible alert mechanism
    - GUI popup notification
    - Email notification
    - System monitoring integration
- Both local & distributed monitoring
- Easy review of alerts
- No dependencies outside of core Linux
- Integration with bug reporting
- Query if this alert represents a known problem

3 Architecture

The architecture of setroubleshoot is a traditional client/server model with a dependency on the audit subsystem for monitoring. The decision to divide setroubleshoot into client and server components was motivated by the need to run the monitoring component at a higher level of privilege needed for monitoring. There also needed to be a central agent which could determine if an incoming event was new or an instance of an event previously seen with the ability to store this information in a persistent database and then finally to disseminate the alert to interested listeners. The server component runs as a daemon thus it is named setroubleshootd following UNIX convention.

The server loads a set of analysis plug-ins at start up. Each plug-in is designed to recognize one or more classes of AVC denials and to provide explanatory text and suggested solutions relevant to the denial it recognized.

Clients running as a unprivileged process make a persistent connection to the server identifying themselves as a unique user. Via an internal RPC protocol clients can query the server for alerts stored in its database, receive immediate notification when a new alert fires and have the server attach per user metadata to the alert such as a desire to filter alert notifications.

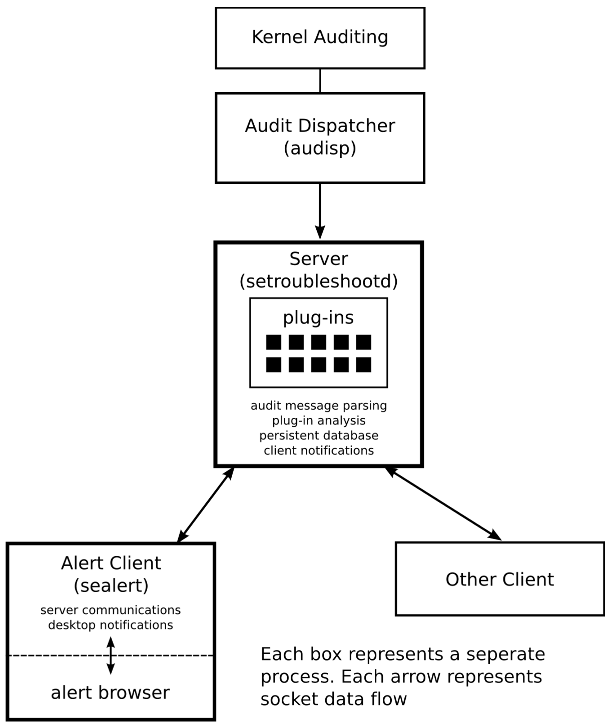


Figure 1: Architecture

4 Implementation

Setroubleshoot is written in Python and limits its dependencies to the core packages found in a standard Red Hat distribution. XML is used to store and transport all data making it amenable to processing by other tools. The entire system is highly asynchronous, both in the data capture area and the network protocol exchanges. As a consequence both client and server components are based on event loops and callbacks. The plug-ins are also written in Python with a conscious effort to make authoring them as simple as possible. The entire system is internationalized and has been translated into a variety of languages.

4.1 Receipt of AVC Denials

The setroubleshootd server receives information about AVC denials in real time by making a client socket connection to the audit dispatcher (audisp). Audit messages arrive in pieces and are assembled into a single audit event. If the assembled audit event is an AVC denial the event is placed on a queue for later analysis in a separate thread. It is also possible to feed the analysis system with audit events read from a log file. When the is event placed into the analysis queue it is tagged with it's disposition so after analysis it can be directed to a database where all events from the audit message source are collected. Alert databases permit aggregation according to the source of the messages, thus there can be a database for real time messages from the audit subsystem on a local node, a database built as the result of scanning a log file, or a database comprising alerts from all the nodes in a managed group.

4.2 Audit Message Processing

The system receives audit information in real time by connecting to the audit dispatcher (audisp). Audisp allows for tools other than setroubleshoot to monitor audit messages via a socket connection. The task of audit monitoring was a non-trivial part of the setroubleshoot implementation due to the following issues:

There is a limited amount of buffering available for audit messages being emitted from the kernel. The audit listener must dequeue the messages quickly or it runs the chance of losing data therefore the part of setroubleshootd which listens for audit messages executes in its own thread because the main setroubleshootd thread might be busy running an analysis on previous audit events or be busy servicing any of it's clients.

The audit system does not emit complete information for an event all at once, rather it emits partial messages which are tagged with an event identifier. Each one of these messages may describe one aspect of the event (e.g. path information, AVC, etc.). There is no guarantee the messages associated with a single event will arrive in order, individual audit messages may be interleaved with messages from other events. There is no termination marker indicating when all the messages belonging to an event have been emitted.

It then becomes the job of the listening software to reassemble the audit messages into complete events. To do accomplish this messages upon receipt are placed in a message cache grouped by their event identifier. The event identifier which is shared by every message in the event includes a time stamp. The message cache is processed (swept) when either its size grows beyond a defined limit or an interval of time has elapsed, whichever comes first. Each set of messages sharing a common event identifier are marked with a time-to-live when they enter the cache. When the cache is swept any event whose time-to-live has expired is assumed to be complete and it is anticipated no more messages will arrive for that event. The messages are then flushed from the cache as a complete audit event. This solves the problem of out of order messages and the lack of an event termination delimiter.

If the audit system is rapidly producing messages many events may be flushed from the cache at once during the sweep operation. Complete events are then checked to see if they are of interest to setroubleshoot (the audit system audits many aspects of the kernel, not just those related to SELinux and AVC's). If the event is of interest the thread listening for audit messages places the complete event on a queue so that it may immediately return to the task of accepting audit messages without blocking. The main setroubleshootd thread will sometime in the future dequeue the event and run analysis on it.

4.3 Processing an AVC denial

The analysis thread removes audit events from the queue and iterates over the set of loaded analysis plug-ins passing the event to each plug-in in succession. Each plug-in can register its priority to control in what order it will be called relative to other plug-ins.

Most plug-ins make heavy use of the source and target contexts of the AVC but are free to examine and consider all data in the audit event. If the plug-in recognizes the denial it will generate a signature used to identify this alert and attach a report to it. This report contains descriptive information such as a summary, a verbose description, a verbose explanation of possible solutions, and zero or more exact shell commands which can be run to alleviate the problem. In addition the plug-in gathers as much environmental information as possible such as the software package which triggered the denial and its version, the SELinux configuration, SELinux policy version, OS version, etc.

The same condition which triggered an AVC denial may occur multiple times, once for each instance the triggering software violated the security policy. Each instance must be recognized as belonging to the same class of event so that individual event instances can be coalesced into a single alert report with a tally of how many times the event has triggered, the date/time it was first triggered and the date/time of it's most recent firing.

4.4 Alert Signatures

A mechanism is needed to label the event the plug-in recognized so that it can be treated as a specific instance of a denial and then referred to throughout the rest of the system by this label (e.g. a “handle” or “key”). It is also vital this label be portable across systems so the same label will refer to the same issue no matter where it is referenced. The label is referred to as the signature of the event. It is the responsibility of each plug-in to generate a signature when it provides a report. This is because only the plug-in can know the particular circumstances which uniquely define the conditions common to all event instances of this class.

The signature is a subset of the event data. It is stripped of any information specific to an event instance (e.g. the process id or inode) but contains enough information to distinguish it from any other event class. The signature is represented as an XML document which structures the event data. To further avoid possible collisions between signatures generated by independent plug-ins the signature contains an “analysis-id” making it unique to the plug-in that generated it. Plug-ins are free to use as many different analysis-id's as they wish and to generate different reports based on the event input. In practice most plug-ins use their own name for every analysis-id. The descriptive report is a template in which information specific to the event will be substituted.

5 Disposition of the alert report

If the plug-in returns a report this signals the event has

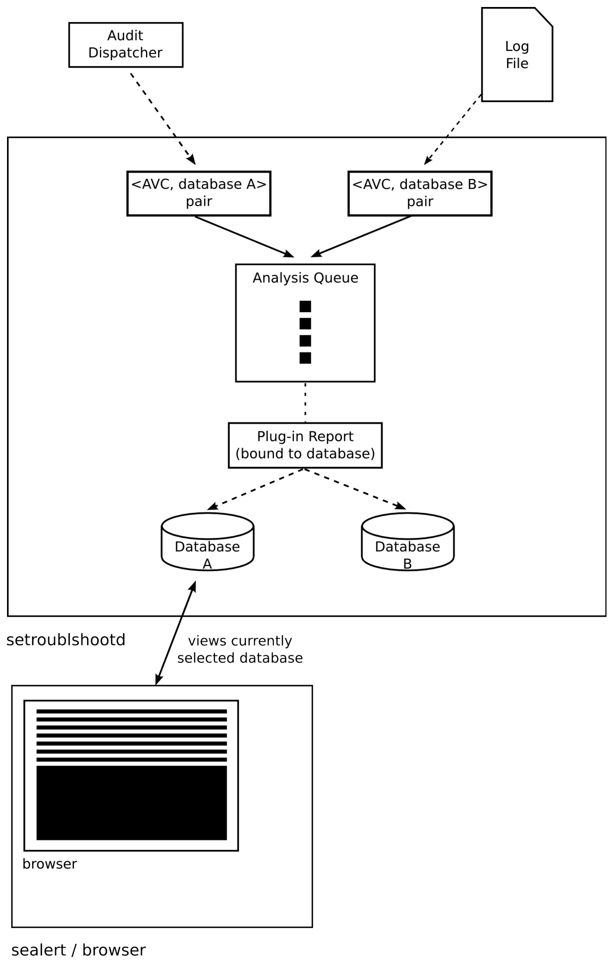


Figure 2: Dataflow

been recognized and the analysis thread terminates the plug-in iteration. The <signature,report> pair is then passed to the database object currently bound to the analysis request. The database object uses the signature as a key to perform a lookup. If the lookup fails a new object (siginfo) is created in the database indexed by the signature. If the lookup succeeds returning an existing siginfo object then that siginfo object is updated by incrementing its report count, recording the current date/time, and copying in the descriptive information in the report. The descriptive information is replaced so only the most current information is presented to the user.

The database object then signals to the server a new alert has been recorded and the server will be able to broadcast the alert to each connected client.

5.1 Client notification

Clients maintain a persistent connection to the server so they may receive alerts as they occur. An important role for the alert client is to ascertain if the user wishes to be notified of this particular alert or whether the user has elected to filter the alert. The filtering information for whether a particular alert for a specific user is filtered is maintained in the database attached as metadata to the alert. This is done for two reasons. Firstly because this information must persist between client sessions. Secondly, there are asynchronous timing issues between when the user interacts at his leisure with the first of multiple alerts which may be arriving at the client and being queued. Because of hysteresis neither the server nor the client can fully know the disposition of any alert in the pipeline. The solution is to update the filtering state in the server at the moment the user sets it. Then when the next alert is pulled off the queue the client queries the server asking if the alert in question is currently filtered, this synchronizes the state between the client, the user, and the server.

5.2 GUI notification of new alert

If the server responds the alert is not filtered a notification is presented on the user's desktop. The notification consists of a status icon indicating there is something pending the user needs to address. The status icon is unobtrusive so it does not interrupt the user's current task or attention. However, because the presence of the status icon in the status bar is unobtrusive the user may miss it's arrival. To address this problem at the moment the status icon is first presented a temporary notification balloon is attached to it calling the users attention to it's presence. The notification balloon will timeout and be removed after a short duration (use of the notification balloon can be configured).

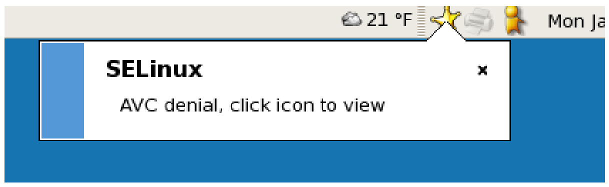


Illustration 1: Alert Notification

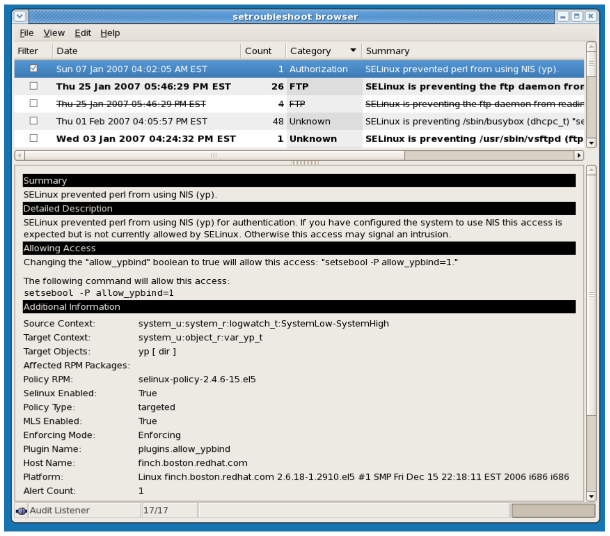
5.3 GUI alert presentation

We experimented with different GUI's to present to the user with when he clicked on the status icon for further information. Originally the idea was to present a popup dialog with just the current alert displayed. Experience suggested this was suboptimal. What happened if there were multiple alerts pending? The user would have to dismiss the dialog only to re-invoke it again. So we tried adding navigation buttons to cycle forward and back through the alert list, but this was awkward. The status icon is removed once the alert is viewed so how could the user go back to a previously viewed alert once the popup dialog had been dismissed? How could the user change the filtering once an alert had been filtered and dismissed? Some form of browser seemed necessary to allow the user to view multiple alerts simultaneously, to sort, group, and search alerts, to change the filtering, or to delete alerts which were no longer of interest.

We experimented with several forms of browsing including via HTTP served by the setroubleshootd server and a custom browser acting as another client of the server. We finally concluded the alert popup dialog was an unnecessary redundant UI component and that a single browser would be presented when the user clicked on the status icon. If the status icon was absent the browser could be brought up via a desktop menu.

It is necessary for a process with a connection to the setroubleshootd server to be continuously running to receive alerts. The server connection, the alert notifications, and the browser GUI would all be contained in a single process with the GUI components being visible only at select moments.

5.4 Alert Browser



Filter	Date	Count	Category	Summary
✓	Sun 07 Jan 2007 04:02:05 AM EST	1	Authorization	SELinux prevented perl from using NIS (yp).
☐	**Thu 25 Jan 2007 05:46:29 PM EST**	**26**	**FTP**	**SELinux is preventing the ftp daemon fron**
☐	~~Thu 25 Jan 2007 05:46:29 PM EST~~	4	~~FTP~~	~~SELinux is preventing the ftp daemon from readir~~
☐	Thu 01 Feb 2007 04:05:57 PM EST	48	Unknown	SELinux is preventing /sbin/busybox (dhcpc_t) "se
☐	**Wed 03 Jan 2007 04:24:32 PM EST**	**1**	**Unknown**	**SELinux is preventing /usr/sbin/vsftpd (ftp**

Summary

SELinux prevented perl from using NIS (yp).

Detailed Description

SELinux prevented perl from using NIS (yp) for authentication. If you have configured the system to use NIS this access is expected but is not currently allowed by SELinux. Otherwise this access may signal an intrusion.

Allowing Access

Changing the "allow_ypbind" boolean to true will allow this access: "setsebool -P allow_ypbind=1."

The following command will allow this access:
```
setsebool -P allow_ypbind=1
```

Additional Information

Source Context:	system_u:system_r:logwatch_t:SystemLow-SystemHigh
Target Context:	system_u:object_r:var_yp_t
Target Objects:	yp [dir]
Affected RPM Packages:	
Policy RPM:	selinux-policy-2.4.6-15.el5
Selinux Enabled:	True
Policy Type:	targeted
MLS Enabled:	True
Enforcing Mode:	Enforcing
Plugin Name:	plugins.allow_ypbind
Host Name:	finch.boston.redhat.com
Platform:	Linux finch.boston.redhat.com 2.6.18-1.2910.el5 #1 SMP Fri Dec 15 22:18:11 EST 2006 i686 i686
Alert Count:	1

Audit Listener 17/17

Illustration 2: Browser

Illustraion 2 shows the current implementation of the browser. It is modeled after many popular email clients. The window is divided into two panes, in the top pane is a list of alerts with columns showing the time the alert was last triggered, a count of how many times the alert has triggered, the general category alert belongs to and the summary of the alert. In addition there is a check box to toggle filtering of the alert. Similar to an email client the alerts which have not been viewed yet are highlighted in a bold font. Once the alert has been displayed in the detail pane for a number of seconds the alert is permanently marked as having been viewed by the user and the highlight is removed.

All of the columns are capable of being sorted so the user can view alerts by age, by frequency, or grouped by category. The category designation is useful for scenarios when the user suspects SELinux might be affecting a particular software component such as his web server or ftp server, thus he can group by category and look for AVC denials associated with that component.

The bottom pane shows the detailed information for the currently selected alert much like an email message would be displayed in an email client. At the bottom of the window is a status area for messages, an icon displaying the connection status with the server, and a progress bar for long duration operations such as loading data or scanning log files. Each alert is capable of being sent to a printer, saved to a file, or deleted.

5.5 Connection states, databases

The sealert client which contains the browser must maintain a persistent connection to the setroubleshootd server in order to receive alerts. It is possible for either end of the connection to go down, this is especially true if the setroubleshootd service is restarted. The sealert client detects the connection loss and periodically attempts to reconnect. These connection attempts are displayed in the status area of the browser along with an icon showing the connection state. When the connection is lost the browser clears all of its data to further reinforce to the user nothing is known at this moment.

The browser presents a view of a single alert database, the browser is said to be visiting that database. The database of alerts received from the audit system is the default database. It is possible to scan log files looking for AVC denials as well. In this scenario a new database is created to store the alert reports generated during the log file scan thus aggregating them as an independent set of alerts. Equally the browser can open an alert database on another node. Selection of which alert database is currently being viewed in the browser is analogous to the selection of a folder in an email client.

5.6 Email alerts

In addition to being notified of alerts via a desktop GUI the system may be configured to send alerts via email to designated recipients. Email alerts may be preferable when a desktop session is not present, when the user does not want a GUI interaction, when he wishes to catalog alerts (i.e. in a mail folder), or when he wishes to monitor a remote system. The email is formatted in both HTML and plain text.

6 Remote monitoring

The architecture supports remote monitoring such as might be desired by a system administrator who manages a collection of servers or a system administrator who may wish to remotely diagnose problems reported by one of their users.

There are two ways remote monitoring can be performed. Alerts can be dispatched via email where a variety of email tools can be used to aggregate, filter, or further disseminate the information. Or, because the architecture is inherently a client/server model a remote client via the same browser interface can be connected to a setroubleshootd server running on another node. By default the setroubleshootd server only listens on UNIX domain sockets thus it can only be connected to from a client on the same node. This choice of default assures greater security, however, the setroubleshootd server can be configured to listen on inet sockets thus affording greater flexibility.

It should be noted the client/server model also allows for the possibility of authoring a module for any number of system monitoring tools. The module would simply connect to the setroubleshootd server as would any other client and then translate the alert into a format suitable for the monitoring tool. The fact that all data in the setroubleshoot architecture is stored and transported in XML should help make data integration with other components easy.

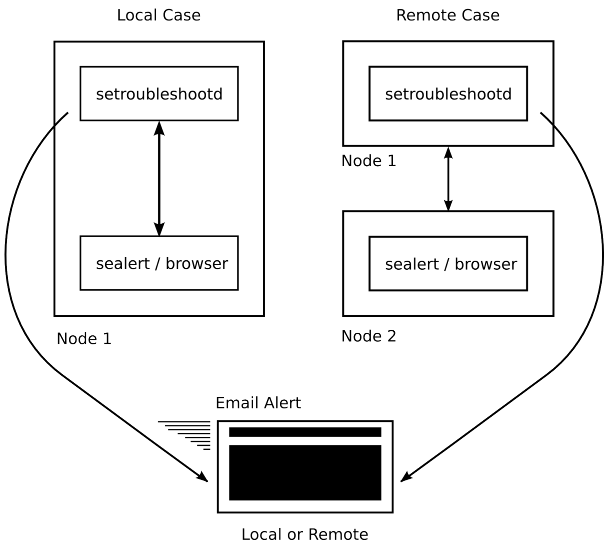


Figure 3: Operational Modes

7 Future work

The design of setroubleshoot and it's model of labeling classes of AVC denials with a signature portable across databases and systems sets it up for future functionality. For example when the AVC denial is suspected to be a bug in either the policy or an application a central server could be contacted to query if this issue has been seen before and to answer some questions. If it has been reported then what bug reports are open against it? Is there a known resolution to the issue? Are there available updates which could be applied to resolve the issue? If the issue had not been reported then a bug report could be automatically generated containing the contents of the plug-in report. The sealert client could periodically query the central server for alerts resident in its database to see if a resolution has been made available and then alert the user he may wish to apply the update. All of these features are dependent on the notion of a portable signature not tied to any given machine.

The current set of analysis plug-in's are geared toward covering the entire set of policy booleans. This is because a very common case of undesired AVC denials is the ignorance of how to tailor the policy to a particular installation. A richer set of analysis plug-in's will hopefully be developed by the community to broaden the coverage and and fine tune the granularity of informational messages. It is also anticipated that as bugs in either applications or the policy are diagnosed a plug-in will be authored to identify that issue helping to smooth the path for the next person who stumbles on the same problem.

8 Summary

An intuitive tool has been developed which communicates with a user in a friendly manner to alert them in real time when the SELinux system has denied access to some resource. Armed with this awareness the user can then take corrective action. The tool is very flexible in it's deployment model and is amenable to extension. We hope setroubleshoot will enhance the use experience and drive the adoption of SELinux by removing some of the barriers to it's acceptance.

9 Acknowledgments

Daniel Walsh contributed enormously with his insights and implementation. Karl MacMillian was a source of SELinux knowledge, contributed code, and was instrumental in keeping the effort focused. The entire set of analysis plug-in's were authored by both Dan and Karl whose knowledge of SELinux operation and experience with AVC denials were key in providing the necessary end user information.

http://hosted.fedoraproject.org/projects/setroubleshoot