SELinux tutorial
Hardening web servers with SELinux

OWASP

EU Summit, Portugal, November 2008

Introduction

$whoami

Pavol Lupták

- OWASP Slovakia local chapter leader
- Big open source and Unix/Linux fan
- Owner of security company Nethemba s.r.o.
- Security consultant with more than 10+ years of practical experience in security with many business certificates (CISSP, CEH, ...) and MSc in Security/Computer Science

Why to use SELinux for Web servers?

- The most secure Linux hardening
- Opensource (everybody can see its code)
- High granularity (full control which syscalls are allowed for every user process)
- With current GUI tools it is not difficult to configure
- Complete segregation of web server from the rest of system
- You can create SELinux policy for any web server or web script

Agenda

- SELinux history
- DAC vs. MAC approach
- DTE, RBAC, MLS models
- SELinux Flask Architecture
- SELinux policy
- Auditing
- Types of policies
- Apache SELinux policy
- SELinux modules: Apache
- Booting SELinux (optional)

Differences between this 2-hours presentation and 1-day training course

- All practical exercises have been removed
- All non-Apache SELinux policy examples have been removed
- No SELinux/CentOS training images are provided

Recommended SELinux platform

- Any Linux distribution (the best support is probably for Redhat EL/CentOS, targeted policy works OK also on Ubuntu, Debian, Gentoo)
- Usually there is no commercial support for strict/MLS policy
- TrustedBSD (SEBSD module)

SELinux history I.

- Originally a development project from the National Security Agency (NSA)
- Implementation of the Flask operating system security architecture
- The NSA integrated SELinux into the Linux kernel using the Linux Security Modules (LSM) framework (Linus Torvalds, who wanted a modular approach to security instead of accepting just SELinux into the kernel)

SELinux history II.

- Originally, the SELinux implementation used persistent security IDs (PSIDs) stored in an unused field of the ext2 inode
- The next evolution of SELinux was as a loadable kernel module for the 2.4.<x> series of Linux kernels. This module stored PSIDs in a normal file
- Finally, the SELinux code was integrated upstream to the 2.6.x kernel, which has full support for LSM and has extended attributes (security.selinux in xattrs) in the ext3 file system. SELinux was moved to using xattrs to store security context information.

DAC (Discretionary Access Control)

- Users can change security attributes at request
- Subject with a certain access permission is capable of passing the permission to any other subject
- Users: administrators vs non-administrators
- Unix DAC - ability of the owner of a file or directory to grant or deny access to other users (chown, chmod, chattr, ..)

Standard Linux Access Control

- Uses an Unix DAC
- Subjects are processes with real and effective user group IDs
- Objects are files, directories, pipes and devices with access mode in inode: rwx r-x --- uid gid
- Access rules are hard-coded in the kernel, checked on syscall call

Standard Linux Security Problems

- Access is based on user's access
- Example: Your firefox (if it is compromised) can read your ssh keys!
- Example II: It is possible to gain root shell through exploiting root process
- **Kernel does not distinguish applications from users**
- **Processes can change security properties**

MAC (Mandatory Access Control)

- Users can not change security attributes at request (non-discretionary)
- A corporate policy or security rules is enforced
- User programs have to work within the constraints of these access rules
- Follows the principle of least privilege
- Subjects vs. Objects

SELinux Access Control

- Uses Flask architecture, DTE, RBAC and MLS security models
- The subjects and the objects remain the same, SELinux assigns to every subject and object a security context (SID) combined from a SELinux user, role, type and MLS level
- Configurable via policy language
- All access is denied by default

Domain Type Enforcement (DTE) model

- Considers domains associated with subjects (processes) and types associated with objects (file, directory, device, ..)
- Domain Definition Table (DDT) - represents allowed access modes between subjects and objects (e.g. read, write, execute)
- Domain Interaction Table (DIT) - represents allowed access modes between domains (e.g. signal, create, ..)
- All acccess is denied unless explicitly allowed

Role Based Access Control (RBAC) model

- Associates the permissions to the roles, not directly to the users
- Each user (or subject) is associated with one or more roles
- Each role contains the permissions that are needed for its correct operation
- 4 classes of RBAC models (core, hierarchical, constrained, unified)

Multilevel Security (MLS) model

- Based on the Bell-La Padula (BLP)
- Multi-level subject - its low level differs from its high level, it is trusted to handle data at any level in its range while maintaing proper separation among the different levels
- Multi-level object - used for the private state of multi-level subjects and for data sharing between multi-level subjects
- Does not care about integrity of data, least privilege, or separating processes and objects by their duty, and has no mechanisms for controlling these security needs

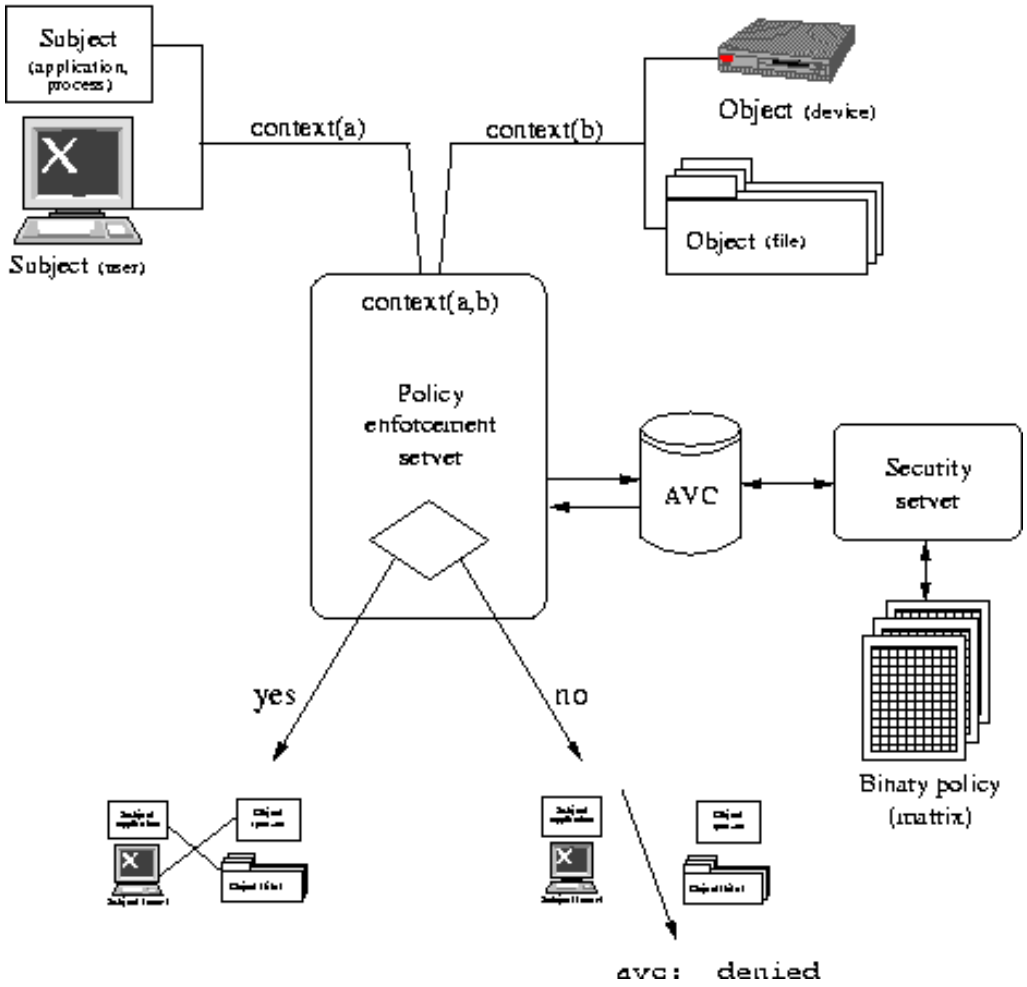
Bell La Padula (confidentiality model)

1. The *-property (read star-property) states that a subject at a given security level must not write to any object at a lower security level (no write-down).
2. The Simple Security Property states that a subject at a given security level may not read an object at a higher security level (no read-up).
3. The Discretionary Security Property uses an access matrix to specify the discretionary access control

Flask architecture

- Provides flexible support for MAC control policies
- Separates the definition of the policy logic from the enforcement mechanism
- Provides an access vector cache (AVC) that stores the access decision computations provided by the security server
- Focuses on the concept of least privilege
- Specifies the interfaces provided by the security server to the object manager that enforce the security policy (DTE, RBAC, MLS)

Flask architecture



SELinux, implemented Flask

- LSM module, using the LSM hooks in the kernel to control and label (Because of the abstraction layer provided - SELinux is highly configurable and modifiable)
- Differences in the specific way SELinux implements Flask in the Linux kernel compared to traditional Flask:
- Under traditional TE, there is a distinction between types and domains. In SELinux, domains are processes that have the attribute process
- The security server, the AVC, and the policy engine are now all parts of the kernel.

What is policy?

- A set of rules that guide the SELinux security engine
- Defines types for file objects and domains for processes, uses roles to limit the domains that can be entered, and has user identities to specify the roles that can be attained, ...

Where is policy?

- /etc/selinux/<policyname>/policy/ — the binary policy and runtime configuration files
- /etc/selinux/<policyname>/src/policy/ — policy sources
- /etc/selinux/<policyname>/contexts/ — location of the security context information
- /etc/selinux/<policyname>/modules/booleans.active
- The configuration file /etc/selinux/config

File System Security Contexts

- SELinux stores file security labels in xattrs
- Label every file system object (all files) with an individual security attribute
- **Mount support:** mount -t nfs -o context=user_u:object_r:user_home_t <hostname>:/shares/homes/ /home/
- fscontext= sets the overarching file system label to a specific security context
- defcontext= overrides the value set for unlabeled files in the policy and requires a file system that supports xattr labeling

SELinux integration – example I

- id -Z
- root:system_r:unconfined_t
- ls -dZ /tmp
- drwxrwxrwt root root system_u:object_r:tmp_t /tmp/
- touch /tmp/foo
- ls -Z /tmp/foo
- -rw-r--r-- root root root:object_r:tmp_t /tmp/foo

SELinux integration - Example II

- ps -Z, ps auxZ
- ld -Z
- ls -laZ
- lsof -Z
- netstat -Z
- find / -context=

SELinux integration – Example III

- chcon (fundamental utility used to change a files context)
- restorecon
- tar --selinux
- star -xattr -H=exustar -c -f
- rsync –X -xattr

SELinux integration – Example IV

- setenforce [0 | 1]
- getenforce
- getsebool named_disable_trans
- setsebool named_disable_trans 1
- setsebool httpd_enable_homedirs 1
- /etc/selinux/config
- selinuxenabled (for scripts)
- matchpathcon

Object Classes

- SELinux defines a number of classes for objects in order to group certain permissions by specific classes (e.g.filesystem for file systems, file for files, and dir for directories).
- Each class has it's own associated set of permissions (filesystem: mount, unmount, get attributes, set quotas, relabel,.., file:read, write, get and set attributes, lock, relabel, link,rename, append, .. netif: tcp_recv, tcp_send, udp_send, udp_recv, rawip_recv, rawip_send)

SELinux Permissions

- Permissions are the actions that a subject can take on an object, if the policy allows it. These permissions are the access requests that SELinux actively allows or denies
- There are several common sets of permissions defined in the targeted policy, in $SELINUX_SRC/flask/access_vectors (EXERCISE)

TE Rules - Attributes

- Identify as groups sets of security types that have a similar property (e.g. httpdcontent, file_type, netif_type, port_type, and node_type, fs_type, exec_type, mta_delivery_agent, domain, reserved_port_type)
- a type can have any amount of attributes, and an attribute can be associated with any number of types.
- attribute domain; attribute netmsg_type;

TE Rules – Type Declaration

- **Syntax:**
- type <typename> [aliases] [attributes];
- ## Examples
- type httpd_config_t, file_type, sysadmfile;
- type http_port_t, port_type, reserved_port_type;
- type httpd_php_exec_t, file_type, sysadmfile, exec_type;

TE Rules – Type Transitions

- results in a new process running in a new domain different from the executing process, or a new object being labeled with a type different from the source doing the labeling
- ## General syntax of a transition>
- type_transition <source_type(s)> <target_type(s)> : <class(es)> <new_type>

TE Rules – Domain Type Transition

- ## Domain transition syntax:
- type_transition <current_domain> <type_of_program> : process <new_domain>
- type_transition httpd_t httpd_sys_script_exec_t:process httpd_sys_script_t;
- type_transition initrc_t squid_exec_t:process squid_t;
- **Macro:** domain_auto_trans(initrc_t, named_exec_t, named_t)

TE Rules – Object Labeling Transition

- ## New object labeling syntax:
- type_transition <creating_domain> <parent_object_type> :<class(es)> <new_type>
- type_transition named_t var_run_t:sock_file named_var_run_t;
- file_type_auto_trans(named_t, var_run_t, named_var_run_t, sock_file);

TE Rules - Access Vectors

- rules that allow domains to access various system objects
- <av_kind> <source_type(s)> <target_type(s)>:<class(es)> <permission(s)>
- **allow** named_t sbin_t:dir search;
- **auditallow** unconfined_t security_t : security { load_policy setenforce setbool };
- **dontaudit** named_t root_t:file { getattr read };

TE Rules - neverallow

- neverallow <source_name(s)> <target_name(s)> : <class(es)><permission(s)>
- neverallow domain ~domain:process transition;
- These assertions are checked by the policy compiler, checkpolicy, when the policy is built, but after the entire policy has been evaluated, and are not part of the runtime access vector cache.

Understanding AVC

- Disallow an operation -> denial message is generated:
- Jan 14 19:10:04 hostname kernel: audit(1105758604.519:420): avc: denied { getattr } for pid=5962 exe=/usr/sbin/httpd path=/home/auser/public_html dev=hdb2 ino=921135 scontext=root:system_r:httpd_t tcontext=user_u:object_r:user_home_t tclass=dir

Understanding SELinux log messages

- AVC Messages can get created for a variety of reasons:
- A mislabeled file
- A process running under the wrong context
- A bug in policy
- Basically an application goes down a code path that was never tested by the policy writer and gets an unexpected AVC
- An intruder

audit2allow and audit2why (tools)

- **audit2allow** - generate SELinux policy allow rules from logs of denied operations
- audit2allow -a -l -o domains/misc/local.te
- audit2allow -a -l -M domains/misc/local
- **audit2why** – translates SELinux audit messages into a description of why the access was denied

SELinux Troubleshoot Tool

- setroubleshoot - service listens to audit daemon for AVC messages, then processes plugin database for known issues /usr/share/setroubleshoot/plugins
- Displays knowledge base of how to handle avc message
- sealert command can launch browser or analyze log files
- Can be configured to send mails /etc/setroubleshoot/setroubleshoot.cfg

Auditing

- Audit system receives SELinux Events
- No auditd running -> AVC in /var/log/messages and dmesg
- auditd running -> AVCs in/var/log/audit/audit.log
- CAPP – Controlled Access Protection Profile
- EAL4+. - E Assurance Level (Level of testing and documentation)
- cp /usr/share/doc/audit-1.0.12/capp.rules /etc/audit.rules

Enable Kernel Auditing

- Sometimes applications fail with no AVC messages (dont audit rules sometimes cover up Real errors)
- Append the parameter audit=1 to your kernel boot line
- RHEL 4: Install selinux-policy-targeted-sources
- make -C /etc/selinux/targeted/src/policy enableaudit load
- RHEL 5: semodule -b /usr/share/selinux/targeted/enableaudit.pp
- semodule -b /usr/share/selinux/targeted/base.pp

Policy Macros

- SELinux uses m4 macro language
- policy.conf contains exploded macro policy codeadmfile;
- # can_exec(domain,executable)
- define(`can_exec',`allow $1 $2:file { rx_file_perms execute_no_trans };')
- define(`rx_file_perms', `{ read getattr lock execute ioctl }')

SELinux Users

- Different than UNIX identities
- Not currently used in targeted policy: In the targeted policy, processes and objects are system_u, and the default for Linux users is user_u
- Linux UIDs and SELinux user identities should match because login and similar applications will try to look up the match. If it fails to find a match, it will fall back to user_u

SELinux Roles

- Define which SELinux user identities can have access to what domains (but simply being in a role is not enough to allow domain transition)
- role <rolename> types <domain(s)>;
- role sysadm_r types ldconfig_t;
- allow user_r sysadm_r;
- role_transition sysadm_r $1_exec_t system_r; (rarely used, only in strict policy)
- Used in strict and MLS policy

TE Rules - Constraints

- Provide final and overarching constraints on the use of permissions that are enforced during runtime by the kernel security server
- Are in the form of Boolean expressions. The expression must be satisfied for the given permission to be granted.
- constrain process transition (u1 == u2 or t1 == privuser);
- constrain process transition (r1 == r2 or t1 == privrole);

Special interfaces & Filesystems

- /proc/<PID>/attr
- current — current security context.
- prev — the context prior to the last exec
- exec — the context to apply at the next exec
- fscreate — the context to apply to any new files created by this process.

Types of policies

- **Strict** - every subject and object are in a specific security domain, with all interactions and transitions individually considered within the policy rules
- **Targeted** - every subject and object runs in the unconfined_t domain except for the specific targeted daemons. The objects on the system that are in the unconfined_t domain are allowed by SELinux to have no restriction

Strict Policy

- A system where everything is denied by default
- SELinux designed to be a strict policy.
- The policy rules only have allows, no denies
- Minimal privilege's for every daemon
- Separate user domains for programs like GPG,X, ssh, etc
- Difficult to enforce in general purpose operating system
- Not Supported in RHEL

Targeted Policy

- System where processes by default are unconfined - only targeted processes are confined
- By default user processes run in unconfined domains (unconfined_t)
- System processes run in initrc_t
- Unconfined processes have the same access they would have without SELinux running
- Daemons with defined policy transition to confined domains

Targeted Domains

- In RHEL4: 15 targets defined (httpd, squid, pegasus, Mailman, named, dhcpd, mysqld, nscd, ntpd, portmap, postgresql, snmpd, syslogd, winbindd)
- In RHEL5: 200 targets defined (every program shipped by Red Hat and started on boot should have a domain defined)
- All system space is confined
- Limited confinement for user space (20 unconfined domains)

MLS Policy

- Strict policy with Bell-LaPadula Support
- Supported in RHEL 5 with special license.
- Server only operating system
- No X-windows support
- Limited package set
- HP/IBM working towards getting EAL4+/LSPP certification

httpd_selinux policy I

- httpd_sys_content_t – data content which is available from all httpd scripts and the daemon
- httpd_sys_script_exec_t – CGI scripts that are allowed to run
- httpd_sys_script_ro_t - CGI scripts in httpd_sys_script_exec_t can only read these files
- httpd_sys_script_rw_t – CGI scripts in httpd_sys_script_exec_t can read/write these files
- httpd_sys_script_ra_t – CGI scripts in httpd_sys_script_exec_t can read/append these files

httpd_selinux policy II

- httpd_unconfined_script_exec_t – CGI scripts in this contenxt can run without any SELinux protection (should only be used for a very complex httpd scripts, after exhausting all other options)
- public_content_t, public_content_rw_t – for sharing files with multiple domains (Apache, FTP, rsync, Samba, ..)
- You need to enable httpd to write to public_content_rw_t by “setsebool -P allow_httpd_anon_write=1” or “setsebool -P allow_httpd_sys_script_anon_write=1”

Booleans in SELinux Policy

- /selinux/booleans/
- echo “1 1”> /selinux/booleans/....
- echo 1 > /selinux/commit_pending_bools
- setsebool [-P] boolean value | bool1=val1 bool2=val2 ...
- getsebool [-a] boolean ...
- / etc/selinux/targeted/modules/active/booleans.local

httpd booleans I

- Online interactive customization of SELinux policy (setsebool)
- httpd_enable_cgi
- httpd_enable_homedirs (chcon -R -t httpd_sys_content_t ~user/public_html)
- httpd_tty_comm (prompt for a password to open cert.file)
- httpd_unified (all files labeled as httpd context can be read/write/execute)
- httpd_builtin_scripting (turn on/off internal (e.g. PHP) scripting)

httpd booleans II

- httpd_can_network_connect
- httpd_suexec_disable_trans (disable suexec transition)
- httpd_disable_trans (disable whole SELinux protection for httpd)

Understanding the File Contexts Files

- # Syntax of file context description
- regexp <-type> (<file_label> | <<none>>)
- Type -d means to match only directories, the -- means to match only files
- /usr(/.*)?/java/.*\.so(\.[^/]*)* -- system_u:object_r:shlib_t
- ifdef(`dhcp_defined', `', ` /var/lib/dhcp(3)? -d system_u:object_r:dhcp_state_t define(`dhcp_defined') ')

Common SELinux macros

- init_daemon_domain, init_system_domain, domain_file, domain_entry_file
- can_exec
- corenet_tcp_sendrecv_all_if, corenet_udp_sendrecv_all_if, corenet_raw_sendrecv_all_if, corenet_tcp_sendrecv_all_node, corenet_udp_sendrecv_all_node, corenet_raw_sendrecv_all_node, corenet_tcp_connect_all_ports
- domain_auto_trans, domain_trans
- files_tmp_file, file_pid_file

Understanding Roles in Targeted policy

- system_r - role is for all system processes except user processes
- user_r - default user role for regular Linux users
- object_r - all objects have the role object_r
- sysadm_r - system administrator role in a strict policy

Assigning Object Types

- Configuration file specifies default context
- Inherited from containing directory at runtime
- **Applications can explicitly set context:**
- chcon: utility to set contexts
- passwd: maintains context on /etc/shadow

Assigning Process Types

- (default) inherited from parent process [bash (user_t) -> ls (user_t)]
- set by policy (type transition rule) [init (init_t) -> httpd_init_script (initrc_t) -> httpd (httpd_t)]
- set by application (e.g., login) [login (login_t) -> bash (user_t)]

Policycoreutils

- genhomedircon, fixfiles, restorecon, restorecond, setfiles, chcon, chcat
- audit2allow, audit2why (See Understanding SELinux log messages)
- secon, sestatus
- semodule, semodule_deps, semodule_expand, semodule_link, semodule_package
- load_policy
- run_init (only in MLS, strict)
- semanage, system-config-selinux
- setsebool, getsebool
- newrole (only in MLS, strict)

Managing File Labeling

- **restorecon** (Used to set a file back to the system default policy)
- **setfiles** (Used to initialize a system. Used at the File system level. Require to specify file_context file)
- **fixfiles** (Script that wraps setfiles/restorecon with several useful features)
- **genhomedircon** (Used to generate file_contexts.homedir)

SELinux Modules

- In RHEL 5 /Fedora Core 5 and later, the concept of Policy Modules was introduced
- **The semodule command:**
- Copies the policy package (pp) files to
- /etc/selinux/targeted/modules/active/modules
- Compiles all installed pp files into new policy file /etc/selinux/targeted/policy/policy.21
- Creates new file_context file and file_context.homedirs
- Loads new policy

SELinux Modules II

- semodule -l ; List all modules currently loaded
- semodule -b /usr/share/selinux/targeted/enableaudit.pp
- semodule -b /usr/share/selinux/targeted/base.pp
- semodule -i myapache.pp
- semodule -r myapache

Policy Modules

- Policy modules consists of three files:
- Type Enforcement File (te) (Contains the allow rules and interface calls associated with the confined domain)
- File Context File (fc) (Contains all of the labeling file context for the policy module)
- Interface File (if) (Contains all interfaces used by other domains to interact with this confined domain)

semanage framework

- Allowing Apache to listen on port 81:
- In RHEL4: required custom policy, policy sources and tools
- In RHEL5: semanage port -a -t http_port_t -P tcp 81
- Other use: semanage user -a guest_u
- semanage fcontext -a -t httpd_bugzilla_script_exec_t /usr/share/bugzilla/cgi(/.*)?

Writing New Policy for a Daemon

- policygentool mydaemon /usr/sbin/mydaemon
- make -f /usr/share/selinux/devel/Makefile
- semodule -i mydaemon.pp
- restorecon -v /usr/sbin/mydaemon
- setenforce 0
- service mydaemon restart
- audit2allow -R -f /var/log/audit/audit.log

References

- https://www.redhat.com/docs/manuals/enter
- http://people.redhat.com/dwalsh/SELinux/Pr
- http://en.wikipedia.org/wiki/Bell-LaPadula_m
- http://hq.alert.sk/~wilder/SELinux-hysteria

Thank you!

Pavol Lupták
pavol.luptak@nethemba.com
http://www.owasp.org
http://www.owasp.org/index.php/Slovakia

Kernel Boot Parameters

- Kernel parameters override /etc/selinux/config settings
- selinux=0 Boots the kernel with SELinux turned off (All files will no longer get created with file context)
- Enforcing=0 Boots the kernel in permissive mode (File labeling continues)

Booting SELinux I

1. The initial process is assigned the predefined initial SID kernel (before the policy is loaded)
2. /sbin/init mounts /proc/, then looks for the selinuxfs
3. If init does not find SELinux in the kernel, finds it is disabled via the selinux=0 boot parameter, or if /etc/selinux/config specifies that SELINUX=disabled, boot proceeds with a non-SELinux system
4. init sets the enforcing status if it is different from the setting in /etc/selinux/config (parameter enforcing is passed during boot)

Booting SELinux II

1. The kernel checks /selinux/policyvers for the supported policy version (/etc/selinux/config)
2. If the binary policy is not the version supported, init attempts to load the previous version policy
3. Init modifies the policy in memory based on the local booleans settings
4. Initial SIDs are mapped to security contexts in the policy
5. init then re-executes itself so that it can transition to a different domain
6. At this point, init continues with its normal boot.

Booting SELinux III