Implementing SELinux Support for NFS

James Carter
jwcart2@epoch.ncsc.mil
Information Assurance Research Group
National Security Agency

The Goal

- Goal is to allow SELinux to work over NFS
  - Same security controls as a local filesystem
    - Correctly handle file and process labels
    - Make access decisions based on those labels
- Goal is NOT to solve all the security problems of NFS

What is Needed

- Client and Server must still work with normal NFS.
- Server must be able to get the security context of the requesting process on the Client and use it to make security decisions.
- Client must be able to get and set the security context of files accessed through NFS.

Interoperability with Normal NFS

- Client needs to specify whether to use SELinux or normal NFS.
  - Modified mount to add “selinux” option and NFS_MOUNT_SELINUX flag
- Server must recognize the difference between a SELinux and normal NFS request.
  - SELinux NFS uses a different RPC program number (100006 instead of 100003)
  - Modified RPC layer to handle more than one program number

Passing the Security Context of the Requesting Process

- Security context and its length is added to the end of the RPC header.
  - Easy, Fast, and at the RPC layer
- Other ways it could have been done
  - Use RPC Authentication flavor – limited to 400 bytes and auth_unix was hardcoded at one time.
  - Add to each NFS procedure – not at RPC layer

Using the Security Context of the Requesting Process on the Server

- Added a fssid to struct task_security_struct
- Modified appropriate hook functions to use fssid, if it is set.
- RPC layer sets the fssid based on the security context sent by the Client, processes the request, and then clears the fssid.
- The Server is trusting the Client

Getting and Setting the Security Context of Files

- Implemented Extended Attribute support for NFS
- Added getxattr and setxattr NFS procedures
- Extended Attributes over NFS are limited to 1024 bytes, not the normal limit of 65536 bytes.

How Things Work on the Client

- Based on the mount flag:
  - Superblock security behavior is specified as using xattrs so the security server will know to use them
  - nfs creates a client with the program number and SELinux nfs procedures
- For every NFS operation, the SELinux NFS program number is used and the security context of the client process added to request.
- The SELinux NFS procedures add getxattr and setxattr to the normal NFS procedures

How Things Work on the Server

- Receives a request and recognizes the SELinux NFS program number
- Gets the client process' security context from the header and sets the fssid
- Processes the request using the SELinux nfsd procedures
- Clears the fssid
- Returns the result

Other Issues

- nfs getattr procedure does not use vfs_getattr, so it skips the security_inode_getattr check
  - Added security_inode_getattr check to NFSD's getattr procedure and to the encode_post_op_attr function
- Client needs to revalidate the security contexts of inodes
  - Modified _nfs_revalidate_inode()

Other Issues

- Server will not check permissions for the owner if the MAY_OWNER_OVERRIDE flag is set
  - Added security_inode_permission() check
- Credential cache on client can cause problems
  - Store sid in struct rpc_cred and compare the current sid to the stored sid when searching the cred cache

Testing

- Test all NFS procedures three ways
  - Test allow and deny using separate files
  - Test allow then deny using the same file
  - Test deny then allow using the same file
- All tests work as they should as long as either the Client or the Server is in enforcing mode.

Testing – Improper Behavior

- Caching
  - Permissive-Enforcing, Allow–Deny:
    - nogetattr, noreadlink will fail
  - Can create cases were caching will cause the wrong behavior on read and writes, even with the Client and Server in Enforcing mode.
- fscreate
  - If setfscreatecon() is used to create a file with a specific security context, there will be a short window where the security context of the file on the server will be the default security context.

Testing – No Credential Cache Fix

- Enforcing - Enforcing, Deny-Allow:
  - access, read, and write fail
- Permissive – Enforcing, Allow-Deny:
  - noloookup, noaccess, noread, nowrite, and noreaddir succeed
- Permissive – Enforcing, Deny – Allow:
  - access, read, and write fail
  - noreaddir succeeds

Numbers

- Setup for the procedure tests involves creating 32 files, 9 directories, and 2 symlinks
  - Normal NFS requires 354 NFS procedures to setup
  - SELinux NFS requires 199 additional NFS procedures (156 getxattr and 43setxattr)
- Running through the procedure test
  - SELinux NFS: 86 access and 20 getxattr
  - No Credential support: 6 access and 20 getxattr
  - No Credential Cache: 142 access and 20 getxattr
  - No Revalidation: 86 access and 5 getxattr
- Overall ~1% performance hit

Future

- NFS version 4
- Other network filesystems