Restricted Community Accounts

Securing Science Gateways at the Account Level

Kevin J. Price

June 14th, 2006

NCSA

Background

- Many Science Gateways use community accounts, a form of shared account
- Shared accounts are a potential weak point in resource security
  - Increased risk of attack
  - Greater degree of anonymity
- The assumed model is a relatively fixed list of applications being used in predictable ways
  - This model applies to Gateways making use of community accounts, but can also apply in other scenarios, such as server-server interactions

NCSA

Goals

- Increase resource security by decreasing potential damage from an attack made through a community account
  - Give resource administrators some peace of mind with respect to community accounts
- Provide a framework that is easy for a resource administrator to implement and configure on a wide variety of systems
  - A POSIX-compliant or otherwise platform independent solution is ideal

NCSA

Our solution

- Gateway provides a policy to the resource administrators
  - List of applications being used
  - Expected parameters for each application
- Policy is converted into configuration files by the resource administrator
  - Resource administrator can choose which control methods to implement for each community account

NCSA

Our solution

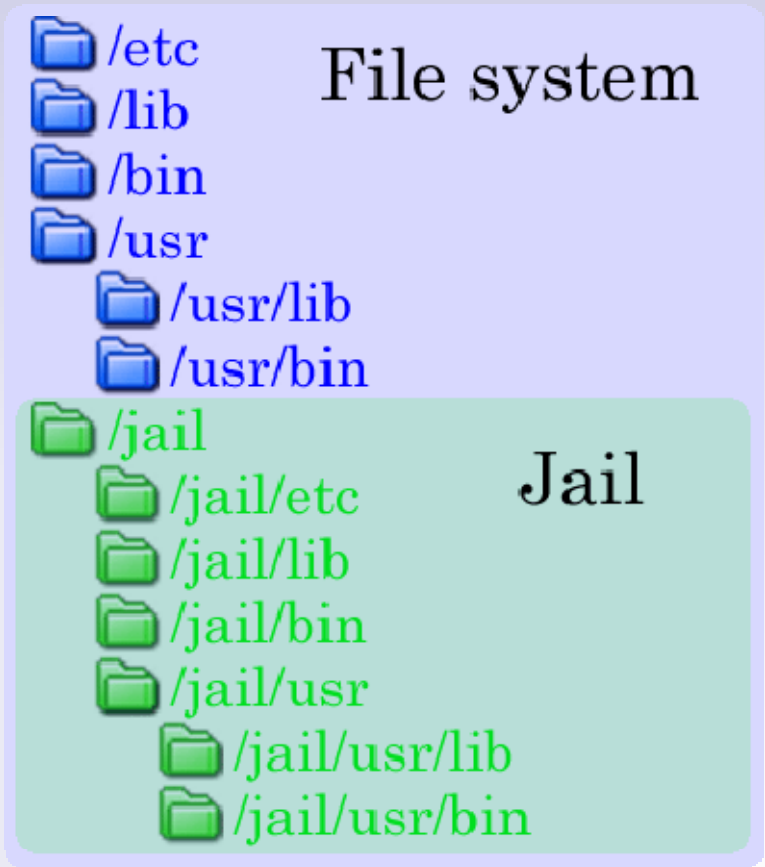
- Utilize a restricted shell to implement either or both of two control methods
  - Command restriction
    - Control the applications an account can run, and the parameters that can be passed to those applications
    - Weaker control, but easier to implement
  - Chroot jailing
    - Sandbox an account at the file-system level, restricting the user to a specific set of applications
    - Stronger control, but harder to implement

Restricted shells

- A UNIX-style shell that can either replace or supplement existing shells (e.g. bash)
- Limit an account's activities
  - Command restrictions
  - File access restrictions
- Provide sandboxing of the environment
- Log user activity

Chroot jailing

- A chroot jail is a subdirectory within a file system that contains a bare-bones version of a standard system install



NCSA

Chroot jailing

- A `chroot` system call redefines the root directory for the current process
  - Files outside of this new root cannot be accessed by the current process or by any child processes
- Calling `chroot` requires root access
- Breaking out of a chroot jail is possible, typically requiring root access as well

NCSA

Products in development

- commsh – Community Shell
  - Restricted shell that can restrict commands and command parameters and use `chroot` to sandbox the user
- chroot_jail – Jail construction utilities
  - Utility suite for constructing and maintaining chroot jail environments

NCSA

commsh

- A restricted shell that offers two distinct methods for securing an account:
  - Command restrictions
  - Secure chroot jailing
- Other methods may be added to future versions, such as dynamic account generation

commsh: Command restrictions

- A set of configuration directives describe which commands can be run
- Directives come in four flavors:
  - Commsh-specific matching format
  - Sudoers-style matching
  - Regular expression matching
  - Exact command and argument matching

commsh: Command restrictions

- Examples:
  - DirectAccess /usr/secure/bin/* **
  - DirectAccess /bin/cat *
  - DirectAccess /usr/bin/jobmgr indata-*.txt outdata-*.txt
  - DirectAccessRE /usr/gridcomm/app < datafile\d+ > outfile\d+$
  - DirectAccessExact /usr/gridcomm/app2 file*

commsh: Command restrictions

- If the command is allowed, the community shell launches the command directly; this occurs without any intervening shell
- If the command is not allowed, the shell logs the occurrence via syslog and terminates

commsh: Secure chroot jailing

- Secure chroot jailing represents a wrapper around the POSIX `chroot` function call
- When invoked, the routine first performs a security audit in the directory, checking possible ways a user might break out of a chroot jail
- If any of the checks fail, the shell logs the error via syslog and terminates

NCSA

commsh: Secure chroot jailing

- Checks performed can include checks for:
  - Setuid root files
  - Other setuid/setgid files
  - Block or character device files
  - UNIX sockets and FIFO pipes
  - Mount points
- These checks are configurable

commsh: Current limitations

- No sandboxing of individual users under the shared account
- Job auditing and accounting by individual user is ignored
  - Martin and Foster have modified GRAM to include this functionality

chroot_jail

- The chroot_jail utility suite is a set of Perl utilities that can be used to create and manage chroot jails
- Dependency detection and tracking features make it easier to manage jails
- Security auditing features help prevent users from breaking out of a chroot jailed environment

NCSA

chroot_jail: Dependencies

- Constructing a chroot jail can be tedious
  - Each executable file can have dependencies that it requires in order to run
  - Those dependencies might have dependencies of their own
- The chroot_jail utilities manage much of this dependency tracking automatically, adding needed files and removing any unnecessary ones

chroot_jail: Dependencies

- Files can have several different types of dependencies
  - Dynamically linked libraries
  - Runtime interpreters (e.g Perl, bash)
  - Symbolic links
  - Language-specific dependencies
    - Perl modules
    - Java modules / JAR files
  - Application-specific requirements
- The first three of these are currently checked for automatically by the utility suite

NCSA

chroot_jail: Security auditing

- When each file is added to the jail, an MD5 checksum of its contents, permissions, and ownership is stored in the database
- A included utility checks for any files that have a modified checksum, as well as for any known security threats, such as setuid binaries or libraries not owned by root

chroot_jail: Current Limitations

- There is no way to modify files within a jail that also modifies the metadata
- Packages cannot be installed directly into a jail
- Files within a jail cannot be grouped together except through dependency information

Command restriction limitations

- Some commands can launch other commands
  - Typically interactive applications
- Poorly coded applications can sometimes be tricked into running arbitrary commands even if they’re not designed to do so
  - Buffer overflows

Chroot jailing limitations

- Users can “break out” of chroot jails using a variety of methods
- Chroot jails only restrict file system access, not CPU or memory usage
- Chroot jailing is not supported by most job managers
- Accessing some file systems (e.g. GPFS) from within a chroot jail is a potential problem

Related work

- Jailkit – a set of utilities (akin to chroot_jail) for creating chroot jails
- Chrsh – a shell wrapper that invokes the `chroot` system call and then launches another shell within the jail
- Samhain – a file integrity checker
- Virtual servers (VMWare, Virtual Server)
- AppArmor and other Kernel modules

NCSA

Future work

- Integrating this solution with GRAM and other job managers
- Making use of the Community Authorization Service (CAS) to manage data file access
- Implement or integrate dynamic account generation into commsh

Project information

Source code for these products, other information about the project and the full paper for this presentation are available at the project website:

http://security.ncsa.uiuc.edu/research/commaccts/

This work was funded by the NCSA NSF CORE award. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author and do not necessarily reflect the views of the National Science Foundation (NSF).

NCSA