Our Problem

Practical Request: Provide a secure (and safe) CVS server, that

- conforms to our local network security policy (e.g. encryption, . . .)
- work reliably for at least 40 internal and external users
- migration of existing (local) repository (ca. 2GB of data)
- provides an easy to maintain access control
- no need for a separated server (extra hardware)

Our Proposal

A CVS server with cvsauth extension and a special setup, providing:

- role based access control (discussed in this talk)
- encrypted data transfer (via cvsauth, not discussed here)
- a (secure) anonymous access

Research Work/Challenges

- verify mapping of roles and users
- verify security/safety/access control properties
**Research Work/Challenges**

- verify mapping of roles and users
- verify security/safety/access control properties

**We provide this using:**
- standardized modeling language, namely Z
- a compiler to Isabelle/HOL-Z
- standard data refinement notions à la Spivey
- special tactics for this type of proofs

---

**Concepts of CVS**

- concurrent (and cooperative) versions management system
- provides a central database: the *repository*
- provides merging for different versions of files (not discussed here)
- every client has a local copy: the *working copy*
Concepts of CVS

- concurrent (and cooperative) versions management system
- provides a central database: the **repository**
- provides merging for different versions of files (not discussed here)
- every client has a local copy: the **working copy**

**Problem:** limited access control via file system

**Our extensions provide:** role-based access control over an insecure network (non-standard)

---

### CVS Server Architecture

**CVS Server Refinement: Group Setup**

| High-level request: | Low-Level Implementation:
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>group</td>
</tr>
<tr>
<td>staff</td>
<td>admin staff staff friend</td>
</tr>
<tr>
<td>students</td>
<td>admin staff students friend</td>
</tr>
<tr>
<td>friend</td>
<td>admin staff friend public</td>
</tr>
<tr>
<td>public</td>
<td>admin staff students friend</td>
</tr>
</tbody>
</table>

Who can write to a file with the following access attributes:

<table>
<thead>
<tr>
<th>admin:owner</th>
<th>friend:group</th>
<th>other</th>
</tr>
</thead>
<tbody>
<tr>
<td>r x</td>
<td>r x</td>
<td>w</td>
</tr>
</tbody>
</table>

Only the users **students** and **public** can write to it.

---

**The System Architecture:**

- **Abstract Data Type for Permissions**
  
  $$[\text{CvsPerm}]$$

  Permissions must be organized in a hierarchy

  $$\text{cvs}_\text{admin}, \text{cvs}_\text{public} : \text{CvsPerm}$$
  $$\text{cvs}_\text{perm}_\text{order} : \text{CvsPerm} \leftrightarrow \text{CvsPerm}$$
  $$\text{cvs}_\text{perm}_\text{order} = \text{cvs}_\text{perm}_\text{order}^*$$

  $$\forall x : \text{CvsPerm} . (x, \text{cvs}_\text{admin}) \in \text{cvs}_\text{perm}_\text{order}$$
  $$\forall x : \text{CvsPerm} . (\text{cvs}_\text{public}, x) \in \text{cvs}_\text{perm}_\text{order}$$
  $$\forall x : \text{CvsPerm} . (\text{cvs}_\text{admin}, x) \notin \text{cvs}_\text{perm}_\text{order}$$
  $$\forall x : \text{CvsPerm} . (x, \text{cvs}_\text{public}) \notin \text{cvs}_\text{perm}_\text{order}$$

---

Achim D. Brucker and Burkhart Wolff  
A Case Study of a Formalized Security Architecture
Refinement and Security

- e.g. hierarchic role-based access control
- e.g. configuration of POSIX groups, users, and file permissions

CVS-Server: High-Level Architecture

Security Properties: access control, authentication, non-repudiation

CVS-Server: Low-Level Architecture

Security Properties: access control

The Abstract CVS-Server Model

- Data:
  - clients with their states (a table of files)
  - server with its state
  - roles, authentication, permissions
  - role hierarchies

- Abstract Operations:
  - login
  - commit
  - update
  - checkout
names and data
\[
\text{[Abs\_Name, Abs\_Data]}
\]

modeling the working copy
\[
\text{ABS\_DATATAB} \equiv \text{Abs\_Name} \mapsto \text{Abs\_Data} \\
\text{ABS\_ROLETAB} \equiv \text{Abs\_Name} \mapsto \text{Cvs\_Perm}
\]

modeling the client state (the security context):
\[
\begin{align*}
\text{ClientState} & \quad \text{RepositoryState} \\
\text{wfiles} & \in \text{P Abs\_Name} \\
\text{wc} & \in \text{ABS\_DATATAB} \\
\text{wc\_uidtab} & \in \text{ABS\_UIDTAB} \\
\text{abs\_passwd} & \in \text{PASSWD\_TAB}
\end{align*}
\]

\[
\begin{align*}
\text{abs\_up} & \equiv \Delta \text{ClientState} \\
& \equiv \text{RepositoryState} \\
& \equiv \text{files?} \in \text{P Abs\_Name} \\
\text{wc}' & = \text{wc} \uplus \{ n : \text{wfiles} \cap \text{files}? | n \in \text{dom rep} \land n \in \text{dom wc\_uidtab} \\
& \quad \land (\text{wc\_uidtab}(n), \text{abs\_passwd}(\text{wc\_uidtab}(n))) \text{ is valid in rep} \prec \text{rep} \\
\text{wc\_uidtab}' & = \text{wc\_uidtab} \uplus \{ n : \text{wfiles} \cap \text{files}? | n \in \text{dom rep} \\
& \quad \land n \notin \text{dom wc\_uidtab} \cdot n \mapsto \text{choose valid rolename(rep\_permtab, n)} \\
\text{abs\_passwd}' & = \text{abs\_passwd} \land \text{wfiles}' = \text{wfiles}
\end{align*}
\]

\[
\begin{align*}
\text{client needs sufficient permissions} \\
\text{non-blocking, files to which the client has no permissions are ignored} \\
\text{the permission table in the working copy is updated}
\end{align*}
\]
Concrete CVS-Server Model

- The POSIX Layer:
  - names, paths
  - POSIX permissions (DAC model)
  - state of a filesystem
  - state of the process
  - operations cd, mkdir, chmod, umask, cp, ...

- The CVS-Server Layer:
  - Operation cvs_login
  - Operation cvs_ci
  - Operation cvs_up
  - Operation cvs_co

The Refinement

- The concrete state:
  - System invariant describing allowable UNIX permissions on the user accounts and the repository. (formalizing 'the administrators job')

- Abstraction relation R:
  - abstract client state are mapped onto files with suitable file permissions
  - roles are mapped onto UNIX configurations (groups, unique uid’s, sticky bits, ...)

System Architecture: Security Properties

Any sequence of CVS operations starting from an empty working copy does not lead to a working copy with data to which the client has no permission (unless he was able to "invent" it).

\[
\begin{align*}
\text{InitAbsState1} & \equiv \text{AbsState} \land [\text{wc} : \text{ABS_DATATAB} \mid \text{wc} = \emptyset] \\
\text{ReachableStates} & \equiv \text{AtransR}[\text{InitAbsState1}] \\
\text{ReadAccess} & \equiv \forall \text{ReachableStates} \bullet \text{ClientState} \land \text{RepositoryState} \\
& \quad \land [\text{wc} : \text{ABS_DATATAB}; \\
& \quad \quad \text{rep} : \text{ABS_DATATAB}; \\
& \quad \quad \text{rep_permtab} : \text{ABS_PERMTAB} | \\
& \quad \quad \forall n : \text{dom wc} \bullet (n, \text{wc}(n)) \in \text{Ainvents} \lor \\
& \quad \quad \quad (((\text{wc}(n) = \text{rep}(n)) \land (\exists m : \text{Aknows} \bullet \\
& \quad \quad \quad \quad (\text{rep_permtab}(n), \text{authtab}(\text{rep})(m)) \in \text{cvs_perm_order}))]
\end{align*}
\]
Security Analysis

We study two levels of possible attacks:

- Attacks against the **abstract model**:
  \[
  \text{trans} = (\text{login} \lor \text{add} \lor \text{commit} \lor \text{update})^* \\
  \text{(change data in wc only to invent data)}
  \]

- Attacks against the **concrete model** (POSIX):
  \[
  \text{trans} = (\text{login} \lor \text{add} \lor \text{commit} \lor \text{update} \\
  \lor \text{chmod} \lor \text{umask} \lor \text{cp} \lor \ldots)^* \\
  \text{(not being root)}
  \]

Summary

- Architecture modeling is an important abstraction level in security analysis: we investigate security models and their relation (and not code)

- ... technique to analyze tricky system administration issues formally

- POSIX/Unix-model reusable, (validated against POSIX and Linux)

- Method applicable for a wide range of practical security problems

Practical relevance (Application)

- over 80 users in 5 different roles

- over 3 GB of versioned data

- used on a daily basis (in mission critical projects)

- used for over two year without problems