A Case Study of a Formalized Security Architecture

Achim D. Brucker
ETH Zürich, Switzerland

Burkhart Wolff
Albert-Ludwigs Universität Freiburg, Germany

June 5, 2003
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 $\mathcal{Z}$
- a compiler to Isabelle/HOL-$\mathcal{Z}$
- standard data refinement notions à la Spivey
- special tactics for this type of proofs
Roadmap

- Concepts of CVS

- CVS Server Refinement
  - Example: Group Setup (Roles)
  - The CVS Server Architectures

- Security as a Refinement Problem

- Security Analysis
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
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 Refinement: Group Setup

High-level request:

- admin
- staff
- students
- friend
- public

Low-Level Implementation:

```
( /etc/group )
```

<table>
<thead>
<tr>
<th>group</th>
<th>users</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin</td>
</tr>
<tr>
<td>staff</td>
<td>admin staff</td>
</tr>
<tr>
<td>friend</td>
<td>admin staff friend</td>
</tr>
<tr>
<td>students</td>
<td>admin staff students</td>
</tr>
<tr>
<td>public</td>
<td>admin staff students friend public</td>
</tr>
</tbody>
</table>

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

```
admin:owner     friend:group     other
r  _  x         r  _  x         _  w  _
```
CVS Server Refinement: Group Setup

High-level request:

- admin
- staff
- students
- friend
- public

Low-Level Implementation:

```
/etc/group
```

<table>
<thead>
<tr>
<th>group</th>
<th>users</th>
</tr>
</thead>
<tbody>
<tr>
<td>admin</td>
<td>admin</td>
</tr>
<tr>
<td>staff</td>
<td>admin staff</td>
</tr>
<tr>
<td>friend</td>
<td>admin staff friend</td>
</tr>
<tr>
<td>students</td>
<td>admin staff students</td>
</tr>
<tr>
<td>public</td>
<td>admin staff students friend public</td>
</tr>
</tbody>
</table>

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

- admin:owner  friend:group  other

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

- Only the users **students** and **public** can write to it.
The System Architecture: Group Setup

- Abstract Data Type for Permissions
  
  \[ Cvs\_Perm \]

- Permissions must be organized in a hierarchy

  \[
  \begin{align*}
  \text{cvs\_admin, cvs\_public} & : Cvs\_Perm \\
  \text{cvs\_perm\_order} & : Cvs\_Perm \leftrightarrow Cvs\_Perm \\
  \text{cvs\_perm\_order} &= \text{cvs\_perm\_order}^* \\
  \forall x : Cvs\_Perm & \implies (x, \text{cvs\_admin}) \in \text{cvs\_perm\_order} \\
  \forall x : Cvs\_Perm & \implies (\text{cvs\_public}, x) \in \text{cvs\_perm\_order} \\
  \forall x : Cvs\_Perm & \implies (\text{cvs\_admin}, x) /\in \text{cvs\_perm\_order} \\
  \forall x : Cvs\_Perm & \implies (x, \text{cvs\_public}) /\in \text{cvs\_perm\_order}
  \end{align*}
\]
Refinement and Security

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

- system architecture (+ security model)
  - implementation architecture (+ security tech.)

- security requirements
  - security technology

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

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

- names and data
  \[\text{Abs\_Name, Abs\_Data}\]
The System Architecture

- names and data
  \[\text{[Abs\_Name, Abs\_Data]}\]

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

- names and data
  \[\text{Abs}_\text{Name}, \text{Abs}_\text{Data}\]

- modeling the working copy
  \[
  \text{ABS\_DATATAB} \equiv \text{Abs}_\text{Name} \rightarrow \text{Abs}_\text{Data} \\
  \text{ABS\_ROLETAB} \equiv \text{Abs}_\text{Name} \rightarrow \text{Cvs\_Perm}
  \]

- modeling the client state (the \textit{security context}):
  \[
  \text{ClientState} \\
  \text{wfiles} : \mathbb{P} \text{Abs}_\text{Name} \\
  \text{wc} : \text{ABS\_DATATAB} \\
  \text{wc\_uidtab} : \text{ABS\_UIDTAB} \\
  \text{abs\_passwd} : \text{PASSWD\_TAB}
  \]
The System Architecture: Operations

\[ \text{abs_up} \] 
\[ \Delta \text{ClientState} \]
\[ \Xi \text{RepositoryState} \]
\[ \text{files?} : \mathbb{P} \text{Abs_Name} \]

\[
w_{c}' = w_{c} \oplus \{ n : w_{\text{files}} \cap \text{files}? | n \in \text{dom rep} \land n \in \text{dom } w_{c} \text{uidtab} \\
\quad \land (w_{c} \text{uidtab}(n), w_{c} \text{passwd}(w_{c} \text{uidtab}(n))) \text{ is_valid_in } \text{rep} \} \triangleleft \text{rep} \]
\[
w_{c} \text{uidtab}' = w_{c} \text{uidtab} \cup \{ n : w_{\text{files}} \cap \text{files}? | n \in \text{dom rep} \\
\quad \land n \notin \text{dom } w_{c} \text{uidtab} \bullet n \mapsto \text{choose_valid_rolename}(\text{rep \ permtab}, n) \} \]
\[
\text{abs_passwd}' = \text{abs_passwd} \land w_{\text{files}}' = w_{\text{files}}
\]

- client needs sufficient permissions
- non-blocking, files to which the client has no permissions are ignored
- the permission table in the working copy is updated

Achim D. Brucker and Burkhart Wolff  
A Case Study of a Formalized Security Architecture
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).
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{InitAbsState}_1 & \equiv \text{AbsState} \land \[wc : \text{ABS\_DATATAB} \mid wc = \emptyset\] \\
\text{ReachableStates} & \equiv \text{AtransR}(\text{InitAbsState}_1) \\
\text{ReadAccess} & \equiv \forall \text{ReachableStates} \bullet \text{ClientState} \land \text{RepositoryState} \\
& \land \[wc : \text{ABS\_DATATAB}; \\
& \ rep : \text{ABS\_DATATAB}; \\
& \ rep\_permtab : \text{ABS\_PERMTAB} \mid \\
& \forall n : \text{dom}\ wc \bullet (n, wc(n)) \in \text{Ainvents} \lor \\
& \ ((wc(n) = rep(n)) \land \exists m : \text{Aknows} \bullet \\
& \ (rep\_permtab(n), \text{authtab}(rep)(m)) \in \text{cvs\_perm\_order})]
\end{align*}
\]
Security Analysis

System architecture (+ security model) → Security requirements

Implementation architecture (+ security tech.) → Security technology
Security Analysis

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

A Case Study of a Formalized Security Architecture
Security Analysis

Attack

login
commit
update

. . .

system architecture
(+ security model)

implementation architecture
(+ security tech.)

security requirements

security technology

new security requirements

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

We study two levels of possible attacks:

- Attacks against the **abstract model**:

  \[ \text{trans} = (\text{login} \lor \text{add} \lor \text{commit} \lor \text{update})^* \]

  (change data in wc only to invent data)

- Attacks against the **concrete model** (POSIX):

  \[ \text{trans} = (\text{login} \lor \text{add} \lor \text{commit} \lor \text{update} \]
  \[ \lor \text{chmod} \lor \text{umask} \lor \text{cp} \lor \ldots)^* \]

  (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 years without problems