# FMICS 2003

# *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 Z
  - a compiler to Isabelle/HOL-Z
  - standard data refinement notions á la Spivey
  - special tactics for this type of proofs





# Roadmap

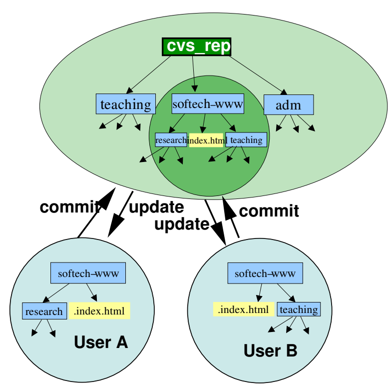
- **Concepts of CVS**
- **CVS Server Refinement**
  - Example: Group Setup (Roles)
  - The CVS Server Architectures
- **Security as a Refinement Problem**
- **Security Analysis**





# 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***

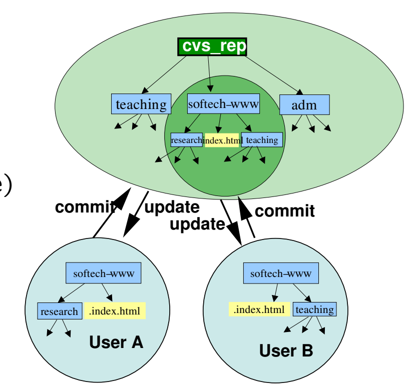





FMICS 03 CVS Concepts 5

# Concepts of CVS

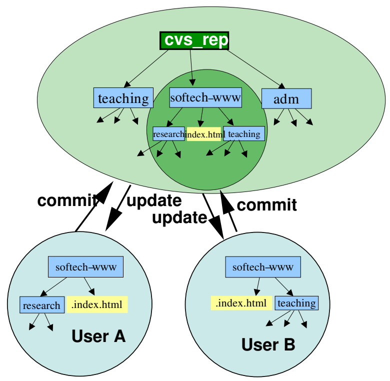
- concurrent (and cooperative) versions management system
- provides a central database: the ***repository***
- provides merging for different versions of files (not discussed here)
- every client has a local copy: the ***working copy***
- **Problem**: limited access control via file system




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

# 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)

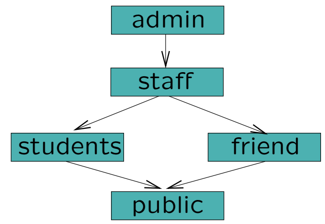







# CVS Server Refinement: Group Setup

**High-level request:**




**Low-Level Implementation:**
(/etc/group)

| group | users | | | | |
|---|---|---|---|---|---|
| admin | admin | | | | |
| staff | admin | staff | | | |
| friend | admin | staff | | friend | |
| students | admin | staff | students | | |
| public | admin | staff | students | friend | public |

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

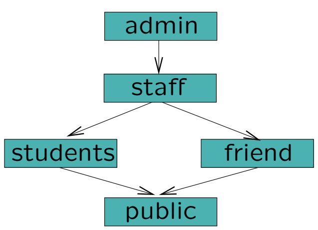
| admin:owner | friend:group | other |
|---|---|---|
| r _ x | r _ x | _ w _ |





# CVS Server Refinement: Group Setup

**High-level request:**




**Low-Level Implementation:**
(/etc/group)

| group | users | | | | |
|---|---|---|---|---|---|
| admin | admin | | | | |
| staff | admin | staff | | | |
| friend | admin | staff | | friend | |
| students | admin | staff | students | | |
| public | admin | staff | students | friend | public |

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

| admin:owner | friend:group | other |
|---|---|---|
| r _ x | r _ x | _ w _ |

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





# The System Architecture: Group Setup

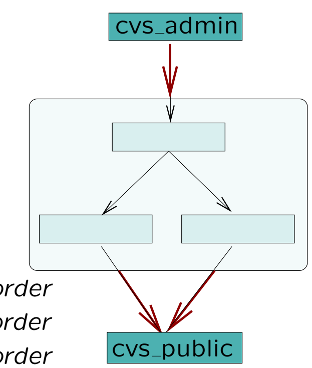
- Abstract Data Type for Permissons

  $[Cvs\_Perm]$

- Permissions must be organized in a hierarchy

$$cvs\_admin, cvs\_public : Cvs\_Perm$$
$$cvs\_perm\_order : Cvs\_Perm \leftrightarrow Cvs\_Perm$$

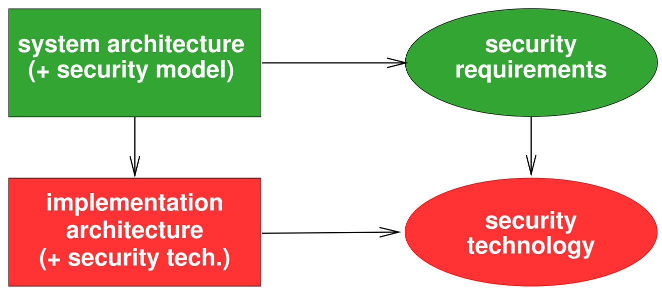
$$cvs\_perm\_order = cvs\_perm\_order^*$$
$$\forall x : Cvs\_Perm \bullet (x, cvs\_admin) \in cvs\_perm\_order$$
$$\forall x : Cvs\_Perm \bullet (cvs\_public, x) \in cvs\_perm\_order$$
$$\forall x : Cvs\_Perm \bullet (cvs\_admin, x) \notin cvs\_perm\_order$$
$$\forall x : Cvs\_Perm \bullet (x, cvs\_public) \notin cvs\_perm\_order$$



## Refinement and Security

e.g.
hierarchic
role–based
access control

e.g.
configuration of
POSIX groups,
users, and
file permissions

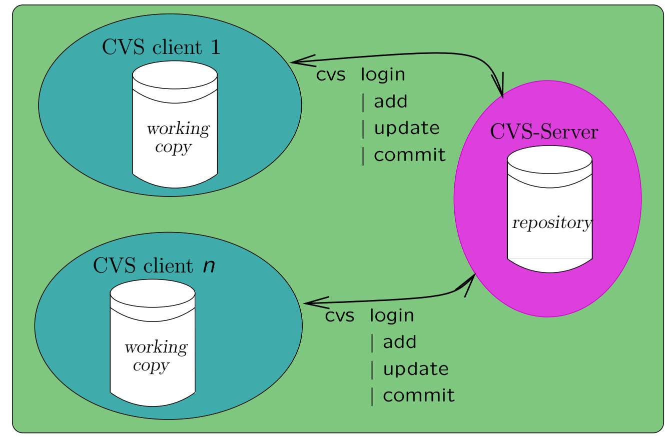







## CVS-Server: High-Level Architecture

Security Properties: access control, authentication, non-repudiation

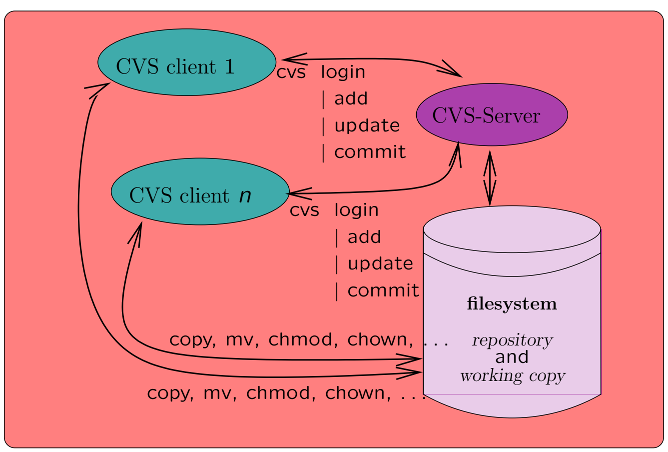







## CVS-Server: Low-Level Architecture

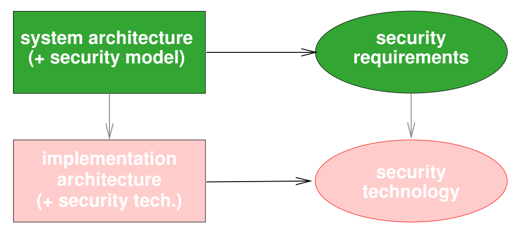
Security Properties: access control








## 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

## The System Architecture

- names and data
  $[Abs\_Name, Abs\_Data]$





## The System Architecture

- names and data
  $[Abs\_Name, Abs\_Data]$

- modeling the working copy
  $ABS\_DATATAB == Abs\_Name \nrightarrow Abs\_Data$
  $ABS\_ROLETAB == Abs\_Name \nrightarrow Cvs\_Perm$





## The System Architecture

- names and data
  $[Abs\_Name, Abs\_Data]$

- modeling the working copy
  $ABS\_DATATAB == Abs\_Name \nrightarrow Abs\_Data$
  $ABS\_ROLETAB == Abs\_Name \nrightarrow Cvs\_Perm$

- modeling the client state (the *security context*):

*ClientState*

$$wfiles : \mathbb{P}\, Abs\_Name$$
$$wc : ABS\_DATATAB$$
$$wc\_uidtab : ABS\_UIDTAB$$
$$abs\_passwd : PASSWD\_TAB$$





## The System Architecture: Operations

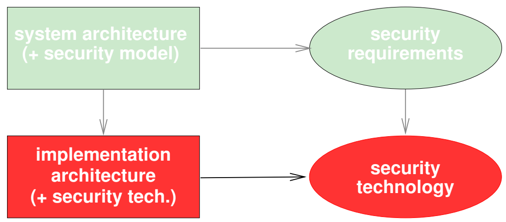
*abs_up*

$$\Delta ClientState$$
$$\Xi RepositoryState$$
$$files? : \mathbb{P}\, Abs\_Name$$

$$wc' = wc \oplus \{n : wfiles \cap files? \mid n \in \mathrm{dom}\, rep \land n \in \mathrm{dom}\, wc\_uidtab$$
$$\quad \land (wc\_uidtab(n), abs\_passwd(wc\_uidtab\, n))\ \mathrm{is\_valid\_in}\ rep\} \lhd rep)$$
$$wc\_uidtab' = wc\_uidtab \cup \{n : wfiles \cap files? \mid n \in \mathrm{dom}\, rep$$
$$\quad \land n \notin \mathrm{dom}\, wc\_uidtab \bullet n \mapsto choose\_valid\_rolename(rep\_permtab, n)\}$$
$$abs\_passwd' = abs\_passwd \land wfiles' = wfiles$$

- 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

## Concrete CVS-Server Model

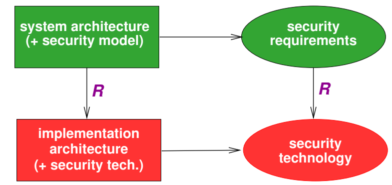



- **The POSIX Layer:**
  - names, paths
  - POSIX permissions (DAC model)
  - state of a filesystem
  - state of the process
  - operations cd, mkdir, chmod, umask, cp, ...
- **The CVS-Server Layer:**
  - Operation cvs_login
  - Operation cvs_ci
  - Operation cvs_up
  - Operation cvs_co





## 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).



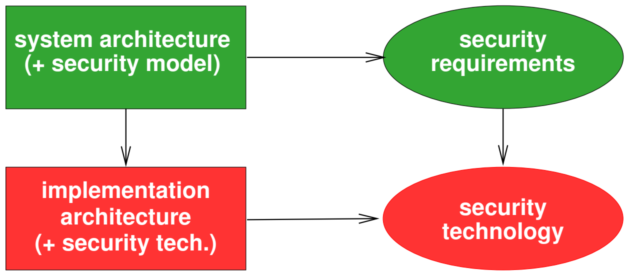


## System Architecture: Security Properties

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).

$$
\begin{array}{ll}
\mathit{InitAbsState1} & == \mathit{AbsState} \land [\mathit{wc} : \mathit{ABS\_DATATAB} \mid \mathit{wc} = \varnothing] \\
\mathit{ReachableStates} & == \mathit{AtransR}(\!| \mathit{InitAbsState1} |\!) \\
\mathit{ReadAccess} & == \forall\, \mathit{ReachableStates} \bullet \mathit{ClientState} \land \mathit{RepositoryState} \\
& \quad \land\ [\mathit{wc} : \mathit{ABS\_DATATAB}; \\
& \qquad \mathit{rep} : \mathit{ABS\_DATATAB}; \\
& \qquad \mathit{rep\_permtab} : \mathit{ABS\_PERMTAB} \mid \\
& \qquad\quad \forall\, n : \mathrm{dom}\ \mathit{wc} \bullet (n, \mathit{wc}(n)) \in \mathit{Ainvents} \lor \\
& \qquad\qquad ((\mathit{wc}(n) = \mathit{rep}(n)) \land (\exists\, m : \mathit{Aknows} \bullet \\
& \qquad\qquad\quad (\mathit{rep\_permtab}(n), \mathit{authtab}(\mathit{rep})(m)) \in \\
& \qquad\qquad\qquad \mathit{cvs\_perm\_order}))]
\end{array}
$$

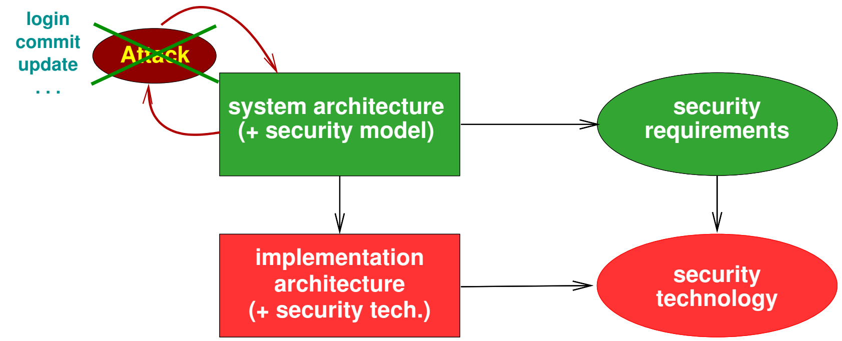
# Security Analysis







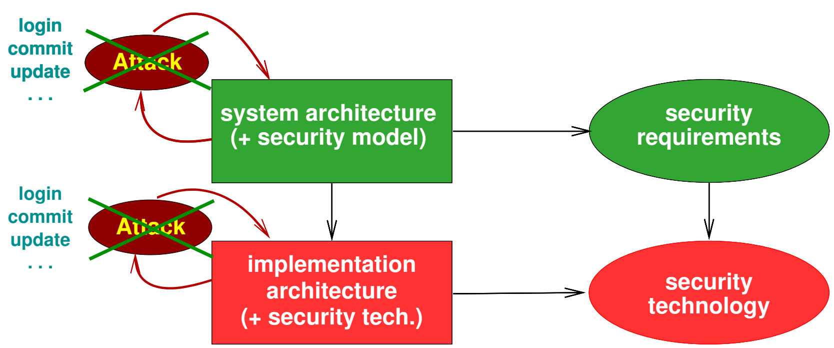

# Security Analysis







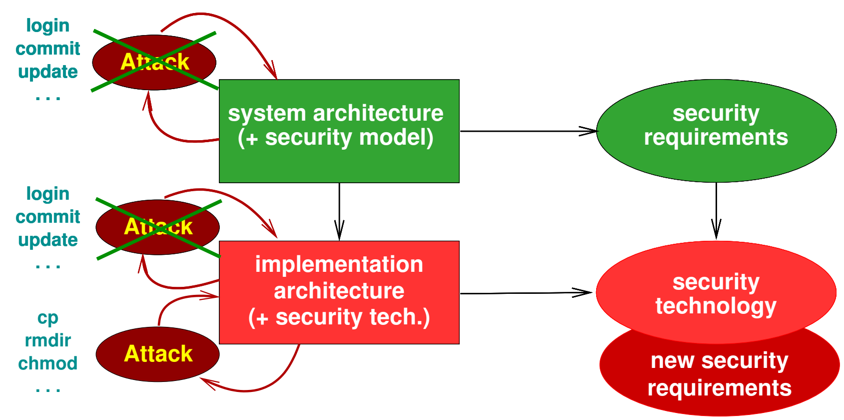

# Security Analysis








# Security Analysis



# Security Analysis

We study two levels of possible attacks:

- Attacks against the **abstract model**:

  $trans = (login \lor add \lor commit \lor update)^*$

  (change data in wc only to invent data)

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

  $trans = (login \lor add \lor commit \lor update \lor chmod \lor umask \lor cp \lor \ldots)^*$

  (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 year without problems