FSEM 1111 Computer Security – from a Free Software Perspective

Christian Grothoff
christian@grothoff.org

http://grothoff.org/christian/

Collaboration

- Essential: problems are often too hard for just one person
- Different people contribute different skills
- Meeting in person is costly (time, travel, low productivity)

$\Rightarrow$ Internet-supported collaboration

Key Tools

- Communication: E-mail, IRC, VoIP, ...
- Data management: Version Control Systems
- Issue tracking: Forums, Bugtracking Systems
- Knowledge integration: WWW pages, Wikis, Forums

Version Control Systems

Key Features of VCS include:

- Content Distribution
- Access Control Mechanisms
- Data Backup / Recovery / Rollback
- Branching and Merging

Content Distribution

- "Latest" version is in the VCS repository
- Any authorized user can obtain this version – possibly multiple times
- Before work starts, checkout latest version from repository
- Periodically during the session (and at the end), commit to repository

$\Rightarrow$ Easy way to keep data synchronized between multiple machines!

Access Control

- Anonymous read-access: anyone can read the data
- Individual read access: specific users can read
- Individual write access: specific users can update
- Group access: simplify management by creating groups
- Partition repository: different rules for different directories

Authentication is usually done using username and password.

Example: Subversion Access Control

```
[/fsem1111/w2007]
grothoff = rw
* =
[/fsem1111/w2007/andrew]
andrew = rw
grothoff = rw
[/fsem1111/w2007/barney]
barney = rw
grothoff = rw
[/fsem1111/w2007/ateam]
andrew = rw
barney = rw
```

Example: Subversion Access Control

The "passwd" file contains lines like this:

`grothoff:N2FHEWsLcoPto`

The SVN client transmits the password P. The server then computes $H(P + Salt)$ and compares with the hash code in "passwd".

Versioning

- Each `commit` operation creates a new revision
- VCS enables accessing all past revisions
- Subversion gives each revision a unique number (per repository)
- VCS attempts to minimize space overhead for storing revisions
- VCS enables concurrent editing and attempts to merge changes

Example: Concurrent Editing

1. Alice creates an initial text T_A and commits to the VCS (R1)
2. Bob retrieves T_A from the VCS and begins to edit
3. Carol retrieves T_A from the VCS and also edits it
4. Bob commits his updated text T_{AB} to VCS (R2)
5. Carol completes her edits (T_{AC}), but her commit fails: she edited R1, but the latest version is R2 (and she edited R1)

Example: Concurrent Editing

6. Carol retrieves Bob's changes $(T_{AB} - T_A)$ using VCS
7. The VCS automatically attempts to produce $T_{ABC} = T_{AC} + (T_{AB} - T_A)$.
8. If the VCS is not certain that it succeeded, it may require Carol to verify T_{ABC} manually.
9. Carol commits T_{ABC} as R3.
10. Alice requests the latest updates from the VCS, obtaining T_{ABC}.

Automatic Merging

- $T_{AB} - T_A$ is computed line-by-line
- Each change is stored with some context (lines before, lines after, offset in file, etc.)
- If changes apply to different lines and are at least a line apart in the document, automatic patching should succeed
- Otherwise, SVN produces a document with both versions

A Merge Conflict

```
Alices text.
<<<<<<<<<<
Bob inserted this text.
==========
Carol inserted this text.
>>>>>>>>>>
More text from Alice.
```

Edit the text to resolve the conflict, then use `svn resolved filename` to tell Subversion that all conflicts in the file have been addressed.

Branching

- Branches enable parallel development of closely related works
- Branches are created (forked) from a common starting point
- The starting point is often the current version, but does not have to be
- Each branch can make progress independently of the others
- VCS can help with merging branches

More on Branching

- Subversion *suggests* placing branches in `project/branches/` and the HEAD in `project/trunk/`
- Creating branches in `svn` is done by copying the entire `trunk` to the `branches/` directory
- Note that this does *not* create an actual copy in the repository (for efficiency's sake)

Viewing Changes in the Branch

- `svn diff -c NUMBER PATH` lists the changes made in the directory PATH since revision NUMBER
- If `NUMBER` is the revision of the branch creation and `PATH` is the path of a branch, then this will show the changes made in the branch

Merging

- You can automatically merge (conflict-free) changes using `svn merge`
- Enter the directory where you want to apply the changes (this can be the branch directory to apply changes from `trunk` or `trunk` to apply changes from a branch)
- Execute `svn merge -c NUMBER https://svn/PATH/` to merge changes made to PATH since revision NUMBER to the copy in the current directory
- Run `svn commit` afterwards to commit the merge (after resolving conflicts)

SVN Merge vs. applying patches

- The `svn diff` and `diff` commands can also show differences between versions
- The `patch` command can be used to apply those versions against an existing version
- These commands *can not* handle file additions and renaming operations
- `svn merge` can handle these!

A Warning

- `svn` does not record branching information

⇒ You must manually track what branches were created from what version and merged into what other versions.

⇒ Keep a text file documenting branching operations – in your repository!

Branches are important for larger projects with multiple releases; however, many projects can do just fine without them.

Questions

?

Group Projects

1. Orientation
2. Division of Labor
3. Assessment
4. Presentation

Orientation

- You need to know the other members of your group
- Exchange and record contact information (phone, e-mail, addresses, etc.)
- Be aware of abilities and limitations of group members (skills, time constraints, motivation)

Division of Labor

- Ensure that the entire group is clear about the project requirements – do not assume that you all agree on those to begin with!
- Break up the project into smaller tasks, establish dependencies between tasks
- Agree on a timetable and individual responsibilities; include contingency plan(s)
- Schedule group meetings

Assessment

- Use meetings to review progress and revise plans
- Provide feedback to material produced by other members
- Plan on doing most outside of meetings (except, possibly for meetings in groups of two)

Presentation

- Plan the presentation only after you have completed your research
- Decide on who presents and how the presentation should be done
- Use multimedia to support the presentation (for example, to visualize data), but the talk itself must stand on its own
- Practice the presentation with the group

Presentation Tips

- *never* put long paragraphs on slides
- More pictures and figures is always better
- Do *not* use these lecture notes as a good example[1]
- Avoid reading verbatim from the slides

[1] These slides are mostly talking points for me and notes for you; they are acceptable for a lecture – but would be terrible for a professional presentation.

LaTeX

- The presentation must be done in LaTeX
- We will talk about presentations in LaTeXin the next lecture
- For now, form groups and study the topic!