Credit Card Transactions, Version Control Systems

CS174

Chris Pollett

Nov. 12, 2008.

Outline

- Credit Card Transactions
- Version Control Systems

Credit Card Transactions

- One use for HTTPS is so that you can do credit card transactions on the web.
- One way to do this is to get a merchant account with a bank and use the bank's API to handle the online transaction. This involves sending data to the bank you are are a merchant of (the acquiring bank) which in turn communicates with the card issuing bank.
- Another way is to use a third party merchant which then communicates with a bank.
- The latter companies usually have two different ways of allowing you to do the transaction:

  (1) You can send the client on to the 3rd party merchant with the data needed for the transaction. The transaction gets handled on the 3rd party merchant's site.

  (2) You can use SSL and communicate with the 3rd party merchant on the server-side. The client never leaves your site.

Authorize.Net

- We will consider the second of these two approaches in the context of Authorize.net, a 3rd party payment gateway.
- Info about their api and sample code can be found at: http://developer.authorize.net/
- We will now consider some of the sample code downloaded from their site.
- The basic key to how this works are the curl_init, curl_exec, and curl_close commands which are used to open a connection from your server to Authorize.net

Version Control Systems

- Another use for https is for version control such as Subversion.
- A version control system solves the problem of how many people can work on the same code base at the same time.
- There are many different such systems on the market, the oldest is Source Code Control System developed at Bell Labs in 1972.
- It was subsequently supplanted in popularity by RCS (Revision Control System) which in turn was supplanted by CVS (Concurrent Version System).
- Nowadays, there are many different versions control systems on the on the market: BitKeeper, Perforce, Mercurial, etc: http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- We will look at Subversion because: (a) it uses http and https as its means of accessing the repository; (b) it is open source andhas been around along enough that it seems for real; and (c) it is reasonably popular.

Subversion

- Subversion was created in 2000 by CollabNet, Inc.
- To get subversion (it often comes with *nix systems) you can go to:

  http://subversion.tigris.org/
- A manual for subversion can be found:

  http://svnbook.red-bean.com/nightly/en/index.html
- I'll often call Subversion svn.
- Common ways to set up svn on a network are to use the svnserve that comes with it or to Apache.
- To begin though, we probably just want to experiment with creating a repository on our local machine, which can be done with a lines like:

  svn create /path_to_repository

  svn co repository_uri local_dir ; cd local_dir

  #svn import trunk file:///path_to_files_want_in_initial_repository

  #if importing from a different repository

  svn mkdir trunk #don't do if did above

  svn mkdir branches

  svn mkdir tags

Trunk, Branches, Tags

- The organization above is not mandatory; however, it is a common way to organize an svn repository.
- Basically, the trunk directory is used to hold the main version of your code.
- Incremental work on the code is usual done and maintained in trunk.
- If someone needs to implement a large feature which might make everything not work until it is ready, then they might create branches from trunk and work on those.
- Often it is useful to have "frozen versions" of the code base which are known to be completely working and don't change much.
- You might use the tags directory for these. For instance, you might maintain a production tag.

Basic Work Cycle

- Typically, a programmer who is going to work on a bug, checks out a version of the repository:

  svn checkout url_or_path_to_repository local_dir

  Or

  svn co url_or_path_to_repository local_dir
- This makes a copy of the repository in local_dir.
- If we cd into local dir, we can see what's there by typing the command:

  svn list
- If you actually do an ls -a on the directory in question, you will see there is a hidden directory .svn in each subdirectory of your local copy. This is how svn keeps track of what things have been updated.
- You can now edit the files to fix your bugs using whatever editor you like (such as vi).
- When you have made your changes, you can do:

  svn update

  or

  svn up

More Basic Work Cycle

- svn up will get the most recent versions of files you haven't changed from the repository into your local store, it will try to merge the changes in the files you have changed with those in the repository again outputting to your local store, finally, it may say there is a conflict on some files which svn couldn't resolve.
- For the conflict files you need to look at the files and fix the conflict. Once its fixed you can type:

  svn resolved filename_with_conflict.
- You can then check in your code (this make your changes take effect in the repository rather than your local copy) with a line like:

  svn ci -m "Fixes BugXXXX, r=so-and-so"

Add, Delete, Diff

- Sometimes in fixing a bug, you need to add a file to the local copy of the repository.
- After you have made the file in the editor, you can then tell svn to add it to the local copy using a line like:

  svn add filename
- Similarly, sometimes you might need to delete a file from the local copy of the repository:

  svn delete filename
- Sometimes you need to create a patch consisting of your changes to the repository. This could then be reviewed by someone else.

  svn diff > my_patch.txt
- Someone could then apply the patch by checking out the code from the repository and then doing a line like (in the correct directory:

  patch -p0 < my_patch.txt

Bugzilla

- Typically, you would post patches you make to some kind of bug tracking tool for review before they are checked-in.
- One such tool is Bugzilla which can be downloaded from: http://www.bugzilla.com
- It has been around since 1998 and is reasonably easy to use.
- Here is a list of other tracker that exist:

  http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems

svn copy, merge

- Creating a tag and a branch are essentially the same thing:

svn copy trunk_url prod_url -m "Create production tag"

- If trunk changes and you want to incorporate the changes into production one can, check out a copy of production and do a line like:

svn merge -r###:HEAD trunk_url branch_name

then commit the result.

- If you don't want to emphasize from which version to which you can leave off -r###:HEAD
- To reintegrate a branch with trunk, you check out trunk, and do a line like:

svn merge -r###:HEAD branch_url trunk

on the local copy, then commit the result.