CS144: Web Applications

Project Part 5: Enabling Secure Transaction
All Parts due: Sunday, March 12, 2017 11:00 PM

- **Please note the hard submission deadline.** No submission will be accepted after 2 day of deadline even you have more grace period left, 11:00PM.
- **Submission deadline:** Programming work is submitted electronically and must be submitted by the deadline. However, we recognize that there might be last minute difficulty during submission process, so as long as you started your submission process before 11:00PM, you have until 11:55PM to completely upload your submission. After 11:55PM, you will have to use grace period as follows.
- **Late Policy:** Since emergencies do arise, each student is allowed a *total* of four unpenalized late days (four periods up to 24 hours each) for programming work, but the final assignment can be more than two days late. If you **"run out of"** your grace period, you will get penalized 25% for 1 day delay, and you will be penalized 50% for 2 day delay. No programming work will be accepted more than 48 hours late (48 hours is a hard deadline).
- **Honor Code reminder:** For more detailed discussion of the Honor Code as it pertains to CS144, please see the Assigned Work page under Honor Code. In summary: You must indicate on all of your submitted work *any assistance* (human or otherwise) that you received. Any assistance received that is not given proper citation will be considered a violation of the Honor Code. In any event, you are responsible for understanding and being able to explain on your own all material that you submit.
- **Reminder:**
    - Projects must be completed individually or by a team of two.
    - You **CANNOT** disclose and discuss any security strategy you plan to use/implement in this project to others. You can only ask for clarification if something is unclear in spec. Any violation may cause penalty to your grade or might be considered as a violation of the Honor Code. If you're not sure whether you should "announce" your question/answer to google group or not, send a private email to TA before you post through google group.

Enabling Secure Transaction

In Project 4, you developed search and browse interfaces to help users access items on eBay. In this project, you extend your Project 4 to allow users to pay for an item by inputting their credit card number. Before you move on, read the following two tutorials to learn how to enable SSL on tomcat and how to keep information through a sequence of pages using http session:

- Enabling SSL on Tomcat
- Creating a sesson-enabled servlet

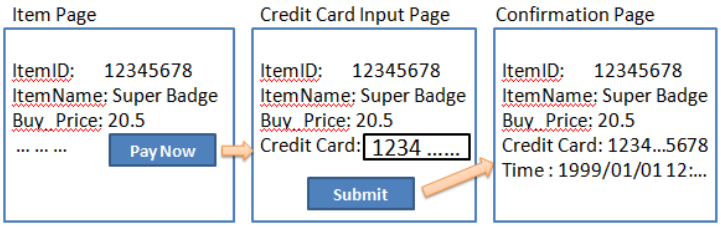
Your task for this part of the project is to enable the following sequence of steps on your tomcat server to obtain the user's credit card number:

Transaction workflow

1. On an item page (i.e., a page that is returned from the `ItemServlet` class in Project 4) if the item has "Buy_Price", you should display a "Pay Now" link.

2. When the user clicks on the "Pay Now" link, your server should lead the user to the "credit-card-input page" that shows the ItemID, Item Name and Buy_Price (and optionally other information) of the item that the user is paying for. In addition, the page should contain one input box to let the user type a credit card number. For simplicity, we assume that a credit card number is enough for authorizing a transaction (i.e., card holder name, expiration date, etc. are not needed).
3. Once the user presses the "Submit" button on the credit-card-input page, your server should show the final "confirmation page", where the user can see the ItemID, Item Name, Buy_Price, credit card number, and transaction time (and optionally other information that you think is relevant).

The image below illustrates the sequence of the pages that the user will see through these steps:



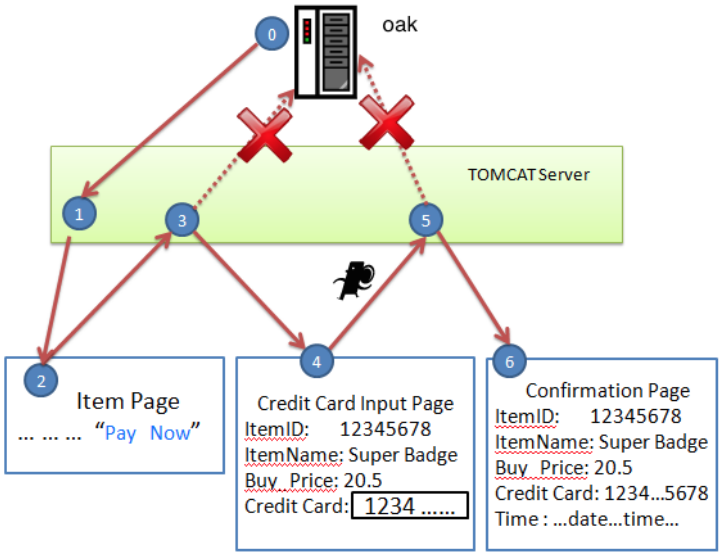
Your task for this project is to implement the relevant Java servlet classes and JSP pages to support the above transaction workflow. Of course, you are taking a serious security risk when you enable online transaction by the users, so you have to be extra careful. In particular, you are working with the following important requirements for this project:

Basic requirements

1. Protecting the credit card number from a malicious third party is essential. No one should be able to eavesdrop on your communication and obtain your customer's credit card number while it is in transit over the Internet.
2. There should be no possibility for a malicious (and technically savvy) user to purchase an item for a price other than its Buy_Price.

For simplicity, other requirements that may be important for a real-world Web site are not enforced in this project. For instance, we do not worry about whether the user-provided credit card number is fraudulent or not, and we do not check whether the user who pays for an item is the same user who won the auction or not. Only the above two requirements need to be guaranteed for this project.

You may think that you can meet the above requirements simply by (a) encrypting every communication between the tomcat server and the user's browser with HTTPS and (b) obtaining Buy_Price directly from oak whenever it is displayed to the user. Unforutnately things are a bit more comlicated due to a few additional preformance-related constraints. We refer to the following figure to illustrate these constraints and data flow among the machines.



Important constraints

1. SSL communication is computationally very expensive, so you need to protect your commuication through HTTPS *only when the credit card number is in transit*. All other communication between your server and the browser should be done through plain HTTP, not HTTPS.

   Note that if you want to encrypt a communication from the browser to the server through SSL, you simply need to make the beginning of the request URL "`https://...`" (as long as the URL has the correct port number for the SSL connection). Then both the request to and the response from the server will be encrypted. If you want to make sure that your communication is not encrypted, start your request URL with "`http://...`". You may find the following methods of the `HttpServletRequest` class helpful to avoid hardcoding the server name and the path at which your page will be deployed: `getServerName(), getServerPort(), getContextPath(), isSecure().`

2. Due to the heavy load on the DBMS server at oak, you have to *minimize* the number of requests sent to oak. Clearly, you cannot avoid contacting oak to obtain the item information while you generate the "item page" (the communication from (0) to (1) in the above figure), but beyond that, the tomcat server CANNOT contact oak for the rest of the transaction. In particular, you are NOT allowed to send a request to oak while you are generating the "credit-card-input page" or the "confirmation page" in order to obtain the item information that is being purchased (i.e., the communication from (3) to oak or from (5) to oak is not allowed).

Given the second constraint, it is clear that the Tomcat server somehow has to "remember" the item information obtained through the communication from (0) to (1) for the rest of the transaction until the final confirmation page (6) is displayed. This can be done in one of the following two ways:

1. The Tomcat server can "push" the information to the client as HTTP cookies, hidden input form fields or as part of a URL link, so that the information is transmitted back to the server when the future request is made. This is essentially asking the browser to "remember" the item information.
2. The Tomcat server itself can remember the information by starting a new HTTP session when the item page is generated and by associating the item information to the HTTP session. Once the session is established, any subsequent HTTP request from the same browser will be part of this session, so Tomcat will be able to retrieve the associted information to process the request.

Due to the potential risk of the "client-state-manipulation" attack of the first approach, we decide to take the second approach in this project by using the `javax.servlet.http.HttpSession` interface. If you are still not clear on how to use an HTTP session to remember some information through a sequence of pages, read our tutorial on HTTP session in servlet again. As you implement new servlet class(es) for this project, make sure that you add relevant entries in the `web.xml` file for for the URL mapping of your servlet class(es).

Since you will need to enable SSL on tomcat, you will need to modify `server.xml` in `/etc/tomcat7/` as well. *When you enable SSL, make sure that you enable it at the port 8443 (the default value given in the existing server.xml file).*

Other Notes

1. Some credit card number might be a counterfeited one. In order to validate a credit card number, two common ways are the "Luhn" and "Mod 10" check. (Reference: How To Generate Valid Credit Card Numbers). In this project, you are **NOT** asked to implement either one. If you desire, you may want to make sure that the credit card number includes only *numbers(0~9), dashes, and spaces* and is not empty using Javascript, but even this is not a mandatory requirement of this project.

What to Submit

This project is an extension to your Project 4, so your submission zip file **project5.zip** should include *all your files for Project 4* and a few additional files that you implemented for Project 5. The structure of the submission zip file should be essentially the same as that of Project 4 as we show here:

```
project5.zip
 |
 + team.txt
 |
 + README.txt
 + build.xml
 |
 + WebContents
 | + any Web files (*.html, *.jsp, *.css, *.js, image files)
 | + WEB-INF
 |   + web.xml
 + src
 | + java source codes for the servlet (with your own naming/structure)
 |
 + lib
   + external java libraries used (not available in our VM)
```

In the README file, please include the answers to the following questions: (A *succint answer* is enough. Please limit your answer to 100 words per question.)

- **Q1**: For which communication(s) do you use the SSL encryption? If you are encrypting the communication from (1) to (2) in Figure 2, for example, write `(1)→(2)` in your answer.
- **Q2**: How do you ensure that the item was purchased exactly at the Buy_Price of that particular item?
- Note: If you get help from any source other than those mentioned in this page, at the end of your README, please clearly cite all references you use, and breifly explain how they help you, such as which portion(s) is/are particularly helpful.

The `team.txt` is a plain-text file (no word or PDF, please) that contains the UID(s) of every member of your team. If you work alone, just write your UID (e.g. 904200000).
If you work with a partner, write both UIDs separated by a comma (e.g. 904200000, 904200001). **DO NOT put any other content, like your names, in this file!**

Please make sure that the `build.xml` in the zip file have the target "build" that builds your web site into a single `eBay.war` file, and the target "deploy" that deploys that .war file on the Tomcat server pointed to by the environment variable `$CATALINA_BASE`. That is, **we should be able to simply unzip your submission and run "ant build" and "ant deploy" to deploy your web site on our machine.**

Your final submission should include *all* files to make your Web site up and running. Add any additional notes or comments that you think will be helpful to the README.txt file.

As always, remember to allow sufficient time to prepare your submission once your work is complete.

Testing of Your Submission

The "grading script" for both Parts A and B submissions is the same file p5_test, which can be executed like:

```
cs144@cs144:~$ ./p5_test project5.zip
```

Add the path to the zip file if necessary after downloading the script and setting its permission appropriately.

You **MUST** test your submission using the script before your final submission to minimize the chance of an unexpected error during grading. When evertything runs properly, you will see an output similar to the following from the grading script:

```
Stopping tomcat server if it is running...
Running 'ant build' to build your war file...

 ... output from ant ...

Removing existing eBay application files on Tomcat...
Deploying your eBay application...

 ... output from ant ...

Now your Tomcat server is running with your application.
Please access your application through your browser.
Make sure that all application functionalities are working fine.
Don't forget to stop Tomcat server once you are done.
```

After you run the script, make sure to check the functionality of your site using the Firefox browser in the VM and stop the Tomcat server when you are done.

Once your submission file **project5.zip** is ready, submit via CCLE.

Grading Criteria

- Successful build and deploy without any error (20%)
- Answers to the questions in readme file (20%)
- Meet the spec(60%, each part is 30%)

Reference

- JSP & Servlet: http://java.sun.com/developer/technicalArticles/javaserverpages/servlets_jsp/