Evaluation of WebGoat

ANTO CVITIC
and KRISTOFFER SVENSK

KTH Computer Science
and Communication

Bachelor of Science Thesis
Stockholm, Sweden 2010

Evaluation of WebGoat

A N T O C V I T I C
a n d K R I S T O F F E R S V E N S K

Bachelor's Thesis in Computer Science (15 ECTS credits)
at the School of Computer Science and Engineering
Royal Institute of Technology year 2010
Supervisor at CSC was Mads Dam
Examiner was Mads Dam

URL: www.csc.kth.se/utbildning/kandidatexjobb/datateknik/2010/
cvitic_anto_OCH_svensk_kristoffer_K10037.pdf

Royal Institute of Technology
School of Computer Science and Communication

KTH CSC
100 44 Stockholm

URL: www.kth.se/csc

Evaluation of WebGoat

Abstract

This essay evaluates WebGoat as a teaching platform intended for computer science students. The problems of teaching many diverse technologies and their interactions pose a challenge to teachers as well as students. Today's web applications are built on technologies and standards never intended for the modern day usage. The complexity has grown and so has the security impact for any organization which finds itself a victim of web application vulnerabilities. Many organizations work on improving web application security, one of which is OWASP - the producer of WebGoat. The key in preventing security holes is education. Here WebGoat shines with a wide span of lessons. In this essay we evaluate how well the application achieves its goals.

Utvärdering av WebGoat

Sammanfattning

Denna uppsats utvärderar WebGoat som ett inlärningsverktyg för studenter inom datalogi. Att lära ut flera olika tekniker och deras samspel är problem som utgör en utmaning för både lärare och studenter. Dagens webbapplikationer är uppbyggda på tekniker och standarder som inte är menade för användning i samma utsträckning som de är idag. Komplexiteten har ökat och det har även säkerheten för alla organisationer som ser sig som ett offer för sårbarheter i sina webbapplikationer. Många organisationer arbetar med att förbättra säkerheten i webbapplikationer, ett av dem är OWASP - skaparna av WebGoat. Nyckeln till att bekämpa säkerhetsluckor är utbildning. Det är här WebGoat kommer in med t brett utbud av lektioner. I denna uppsats utvärderar vi hur väl applikationen uppnår sina mål.

Table of Contents

1. Introduction

We evaluate WebGoat as a teaching platform for web application security and propose improvements of both a didactic and structural nature. We focus on injections, XSS, and AJAX vulnerabilities since these are common in web applications and are difficult to detect.

WebGoat is a deliberately insecure web application developed for educational purposes. It is aimed at students who wish to get a better understanding of the risks and vulnerabilities found in modern web applications and to learn how to avoid security breaches in their own web development.

The application contains the most common types of security holes and it is in use by companies such as Google for educating their employees. Each lesson has a plan explaining the background of the problem and hints which guide the student to a solution; the students' task is to finish the lesson by exploiting the vulnerability. Due to web application security encompassing a wide area of technologies which often interact, it is a difficult process even for users with programming experience to complete the lessons given in WebGoat. The variety of technologies and standards make it a difficult subject to teach.

2. Purpose

Evaluate the suitability of using WebGoat as a teaching platform by various testing. The purpose of this paper is to propose improvements of WebGoat and web application security teaching in general.

3. Method

Using the information provided in WebGoat for a specific lesson we tried to learn the techniques set out and took notes on our progress. We suggest improvements for those lessons we found are lacking in hints or lesson plans according to the criteria below. For the lessons which required using other sources we took note of this and suggest to include this information in the lesson.

We evaluate how well the WebGoat lesson taught us the specific security issue by checking how many times we had to resort to other sources and how long it took to complete a lesson. We take notes every time we resort to hints, this way we have a metric on the minimal required assistance.

- Evaluate lessons based on our learning experiences, and comparing with other sources
- Evaluate problem-solving approach in computer science from a didactics point of view
- Propose improvements to current lessons

The criteria for a lesson to be considered of good quality are the following:

- Focus, what is to be learned in this lesson.
- Intent, clearly specify the goal.
- Hints, the lesson should provide hints to solve the lesson, but not give the answer in full too early.
- Show the student how to prevent the flaw in his/her own programming.
- Solution, describe in detail the activities to be performed to complete the lesson.
- Obstructiveness, the application should not interfere with the lesson.

Focus and intent correspond to the *understanding* part of the problem solving model explained below. Here the student forms the conceptual idea of the task at hand using the lesson plan and background information on the subject. Hints correspond to the *design* phase*;* they help look at similar problems and solutions. Obstructiveness corresponds to the *writing* phase*;* which is when the student attempts a solution, the web application should not fail or crash. Solution and prevention corresponds to *reviewing/feedback* in the model.

4. Background

Web application security in general is a challenging task since it encompasses many varying technologies - from HTTP protocols to Java and PHP programming. An error in each of these could have a sever security impact. For an organization, vulnerabilities in web applications could be costly since any investments in firewalls and intrusion detection systems are bypassed by the nature of web applications. Web applications are by nature exposed to a wider audiance.

4.1 Problem based learning in computer science

One approach to teaching computer science as well as other fields is a problem solving method. The following is a description of this method and how it has been used in other schools.

The problem solving method has been applied successfully in fields such as medicine and mathematics. In computer-science it has been investigated and implemented successfully in UK for foundational courses. A general description:

> "*The principal idea behind problem-based learning is that the starting point for learning should be a problem, a query or a puzzle that the learner wishes to solve… [...]. Information on how to tackle the problem is not given, although resources are available to assist the students to clarify what the 'problem' consists of and how they might deal with it.*"[6]

A study from the university of Kent in Canterbury gives another overview of problem-based learning (PBL).[3] It includes four phases which a student passes when acquiring knowledge of both the problem and the solution. These are in order: *understanding*, *design*, *writing* and *reviewing*. The *understanding* part is to ask "what if" and see the problem from as many angles as possible. During *design* one looks to related problems for possible solutions and how they might be familiar. During *writing* one turns to the chosen solution and puts it to action.

In *reviewing* one looks back at the solution. The aim is to consolidate and appreciate lessons learned so they can be reused when facing other problems in the design phase.

A long standing debate in teaching is concerned with the level of *guided assistance* necessary. One side argues that students learn best with minimum required assistance. The students are left to discover or construct the solution for themselves rather than being presented with crucial information from which they can solve a problem. The other side argues novices to a field should not be left alone and instead be told or instructed about the problems, concepts and solutions in the field. PBL fits in the minimum guided side. The underlying assumption for PBL is that students learn best by walking the path of professionals and acquiring experience by performing the same tasks themselves.[7]

PBL has been criticized for letting students solve the puzzles with minimum required assistance. This method does not really teach the students the main reasoning behind the solutions and how to apply it to similar problems. Merely they get a slight speed increase in solving puzzles but not as fast as students who read a set of problems and their explanations. However why this is is not explained in the given studies. Students who are asked to read a set of problems and their solutions tend to do better and faster when given similar problems. This is a major criticism of a problem-solving method. The reason for this difference may lie in the way the problem is put forth and the assistance given. If the problem is way out of the students knowledge-zone any amount of hints will not make her solve the problem, unless he first gains a fundamental understanding of the problem and background at hand.[7]

This is a major limitation of both PBL and WebGoat. However the limitation can be bypassed if crucial instructions for each lesson are given. Crucial instruction is defined here as the background for a lesson explaining the concepts and ideas behind the vulnerability. This instruction can be in the form of either a hint, in the lesson background or the solution of the lesson.

4.2 WebGoat as a teaching platform

WebGoat consists of lessons in web application security. The lessons are split up into categories depending on type of threat. Each lesson contains a problem and a description in form of a lesson plan as well as a solution. Assistance and guidance is available in the form of hints. It is possible to observe parameters, cookies and other data sent to and from the application by using a web proxy. From the given information a student is expected to successfully complete the task on her own. As described above this approach corresponds to problem-based learning and students are to solve the problems with minimum guidance. Students become aware of the vulnerabilities but also gain first-hand experience in exploiting them.

In many lesson it is required to have third-party software. For example a web proxy is frequently needed. WebGoat has an introductory lesson in which the developers recommend WebScarab which is an open-source software. Other useful tools are Firebug, IEWatch and WireShark. It is recommended to complete the introductory lesson before beginning to work with WebGoat.

4.3 Vulnerabilities for investigation on WebGoat

A short overview of selected types of attack which we are going to evaluate in WebGoat is in order.

Injection flaws are a type of flaw in which a parameter supplied to the web application goes unchecked or is unsanitary and then used by the web application as ordinary data or even code. In this way a malicious user may enter or inject code to be run as part of the web application. The most common type is SQL-injection where the supplied user input is performed on the database without checking for special characters or otherwise escaping the input, thus leading to execution of arbitrary user SQL statements with the possible result of stealing passwords or other sensitive information. A common method to escape this kind of attack is to strip the input from special characters which control the statement such as `'` and `;`. However this technique of mitigating the attack is not the best since the stripping has to be performed on every occurrence of user supplied input with the database. This can be difficult and is easily missed. Another flaw in this mitigation technique is the wide gap left even after stripping of almost all special characters. Since most SQL statements can be rewritten using only letters. One does not need `=` but can write `LIKE` instead. A better way is to allow input only from a whitelist, however this method is also difficult to maintain and the preferred method to mitigate against SQL-injections is to use prepared statements. With a prepared statement the user supplied input cannot interfere with the coded SQL statement since it is already in the SQL engine. The user supplied input is merely inserted into the existing statement. Another advantage is that the prepared statement will be parsed only once by the SQL engine, so a speedup is achieved as well.

This type of attack might give the ability for an attacker to run her own code, and to access system files. Whenever an input to the web application requests to access the file system, if the input is not sanitized a malicious user may give `../../../etc/passwd` as input and access the `/etc/passwd` file.

This type of attack also allows for another common security issue, namely **reflected Cross-Site Scripting (XSS)**. This is a form of attack where input provided by one user is supplied to another user visiting the same web application. For the other user this seems to come from the legitimate source and the malicious users' code can be executed on this users machine to steal, for example, session cookies. This can happen if the web application is showing what one user entered to another user. This could be a user profile, a web forum or similar. The malicious user may enter his/her JavaScript to perform requests to the web application from the other users web browser. To mitigate against this type of attack the web application should sanitize the user input and not allow code to be entered as part of user input.

Asynchronous JavaScript and XML (AJAX) allows web clients to perform asynchronous requests to web servers using JavaScript. This means a web application can serve the client a document with code to make new requests in the context of the served document. The client can request either data or new code to run with the same privileges as the first document. The injection and cross site flaws are still around when using this technique, however using AJAX allows for these to be performed without the client knowing since the requests are sent in the background from the current context. AJAX is used to update a page without performing a new HTTP request.

Already we notice that many attacks are intertwined and depend on each other, that is why it is important to reach a deep understanding of the underlying web technologies. Only with this understanding and good coding practices one can escape the risks.

5. Evaluation of WebGoat Lessons

5.1 Overview

WebGoat is a broad application in the sense of the amount of lessons included. Because of a limited time frame, we had to make a selection and narrow down the lessons evaluated. We chose eight of the eighteen categories of vulnerabilities and out of these two-three lessons each.
The lessons are the following:

- **AJAX Security**
  - Silent Transaction Attacks
  - Dangerous use of evaluation
- **Concurrency**
  - Thread Safety Problems
- **Cross-site Scripting**
  - XSS LAB: Cross site scripting
  - Cross Site Request Forgery (CSRF)
- **Denial Of Service**
  - Denial of Service from Multiple logins
- **Injection Flaws**
  - LAB: SQL-injections
  - Modify data with SQL-injection
  - Command injection
- **Malicious Execution**
  - Malicious File Execution
- **Parameter Tampering**
  - Bypass JavaScript Client-side Validation
- **Session Management Flaws**
  - Hijack a session

5.2 AJAX Security

5.2.1 Silent Transaction Attacks

In the given lesson there are two input fields, a submit button and a statement of "account balance". The goal is to try to bypass the user's authorization and silently execute the transaction.

The background to the problem is given in detail. The first hint told the student to look at the source page where a user can find the method that submits the transfer and the JavaScript function. With enough programming skills a user can see that the function performs client-side validation and then sends the request to transfer money as a HTTP GET parameter. The lesson is completed by using the JavaScript function, submitData, which sets the amount of money in some account directly. An attacker can enter this in the address field of the browser to execute the transfer silently. The request will be fulfilled with no validation or confirmation since the attacker bypasses the validation function.

Summary
Students learn from this type of attack that any server which silently processes client side request in a single submission is dangerous to the client. It is possible to change the state of the server-side application and data with a single statement. For AJAX it is worse since no feedback is given to the user hence the 'silent' in the lesson name. Exactly how these vulnerabilities are programmed is left as an exercise. By looking at the source code one needs to sort out the information relevant to the teaching tool and to the JSP as such to find where the vulnerability is introduced.

All criteria for a good lesson are fulfilled.

5.2.2 Dangerous use of evaluation

The goal for this lesson is to submit some input containing a script. The script should bypass a server-side validation of user-supplied data and be reflected back into the browser. The lesson plan explains that the application is using a JavaScript function `eval()` to validate the input. The solution is given in a hint explaining the student must use `alert()` on document.cookie in order to successfully complete the attack. It is also mentioned that this is a reflected XSS attack. The lesson is built-up using a table representing a shopping cart and two input fields for credit card information.

The information given at this point provides a clear perspective on how to solve the problem. It involves a JavaScript which uses the `alert()` function on document.cookie. As a first approach the student could try submitting a JavaScript with standard tags `<SCRIPT>alert(document.cookie);</SCRIPT>`.

This solution is not accepted because as mentioned in the lesson plan, the input is used in conjunction with a function called `eval()`. A student familiar with JavaScript would suspect that the script tags are not necessary. Being aware of this, the security flaw is easily exposed by closing the `eval()` and then invoke `alert(document.cookie)`. This information is also included among the hints and eventually even the solution will be given.

Summary
The lesson is well structured and informative, focus is achieved. The lesson plan gives a good background to the problem and leaves minimal room for misunderstandings. The hints are within an appropriate range and not too revealing. The lesson fulfills all criteria since the prevention of this is learned in other lessons, where the solution is to sanitize user input.

5.3 Concurrency

5.3.1 Thread Safety Problem

The goal for the student to exploit the thread concurrency vulnerability by viewing the login information for another user that is attempting the same function at the same time. The lesson states that this will require the use of two browsers.

The lesson plan explains that web applications handle many HTTP requests simultaneously and developers can use variables that are not thread safe. This means a variable can be maintained across multiple threads and so exposing itself across threads. A malicious user can then access a variable which is set by another user. A student must be familiar with threads for this lesson to be meaningful and understand what thread safety means. The first hint "Web applications handle many HTTP requests at the same time." is redundant considering it is already stated and explained in the lesson plan.

Reading the lesson introduction "...view login information for another user that is attempting the same function at the same time" it is apparent a user could just attempt the same function at the same time from different browsers, or if the timing is tight use a timer to set off the requests. From this information the solution is implied, use two browsers, Opera and Firefox, enter Dave in Opera and Jeff in Firefox. Push the submit button almost simultaneously, Firefox also returned daves information just as Opera. Showing the source code the student can see that the timing is 1500ms which is enough time to switch browser and push the button.

Summary
The lesson maintains focus, shows clear intent and the hints are within range. After completing the lesson the user should be aware of the problems of thread concurrency and not use variables, especially user data, across multiple threads.

5.4 Cross-site Scripting

5.4.1 LAB: Cross Site Scripting

Stage one - Stored XSS
The goal is to execute a stored XSS attack as 'Tom' against the Street field on the Edit Profile page and verify that 'Jerry' is affected.

The lesson plan is a concise introduction to XSS. If one users unvalidated input is retrieved by another user, the victim browser can be caused to load another undesirable page or run other malicious code. This flaw can also be exploited when unvalidated user input is used in an HTTP response. A reflected XSS attack means an attacker can craft a URL with the attack script and post it to another website for the victim to click.

Using only this information it is unreasonable that a student will get the full depth of cross-site scripting. The lesson plan is very concise and if the student does not already possess some understanding of XSS, other sources are required such as OWASPs own article on XSS.

The instruction to execute the XSS attack as 'Tom' against the Street field on the Edit Profile page leaves the student wondering how if not already familiar with XSS. A hint is needed to push the student in the right direction. The first hint does exactly this, “You can put HTML tags in the form input fields”. This suggests the student can edit his profile and there add his code. When another user views the profile, the code will execute. Updating the Street field as 'Tom' by adding a simple `<br />` tag and later viewing it as 'Jerry' confirms it is possible to enter any tags in the field which do get executed as normal code by other users. This is a stored XSS attack.

The lesson code in WebGoat does not check for any kind of tags or code entered in the input fields, it only checks specifically for the following `<script language="javascript" type="text/javascript">alert("document.cookie");</script>`. This solutions is given as a fourth hint, so even if the student performed the XSS, the lesson code would not detect it as successful. The lesson falls here on the obstructive criteria. This could be improved in the lesson code by checking if the input field contains < or > and a list of valid html tags and so set the student as having completed the lesson.

This lesson will not however teach a student everything there is to know about XSS, for this the student needs to read more on the subject from OWASP or other sources or continue the lab.

Even after reading the concise explanation, a user still needs to think exactly how to perform this type of attack. This gives weight to the value of a problem-solving approach. The student gains experience, not only knowledge.

Stage two - Block stored xss using input validation

This lesson only works with the developer version of WebGoat. The goal of the lesson is for the student to block the stored xss in the previous stage.

There are hints for this stage saying “Many scripts rely on the use of special characters such as <”. This suggest for the student to block these and possible other special characters used in the previous stage, and so complete the lesson. The second hint says “Allowing only a certain set of characters (whitelisting) is preferred to blocking a set of characters (blacklisting)”. The third hint was also helpful, which read “The java.util.regex package is useful for filtering string values.” The student can then begin the design phase, actually implementing the whitelist in the Java code.

With these hints, a student proficient in Java programming can complete the lesson by allowing only letters in the input fields and throwing ValidationException if failed.

Looking at the source for the lesson one can find the handles for completing the lesson and it is as good as the other lesson of the same type "block previous vulnerability" in injection flaws.

There is even a comment in the right place to validate the input using a parsing component and an empty method - doParseEmployeeProfile.

Stage three & four
The goal here is to verify that 'David' is affected by a previously stored XSS attack in 'Bruce' profile. A simple check finishes the lesson.

What the student learns here is that only validating input does not help against already stored XSS. The fix from stage two only disallows entering XSS into any field, if the XSS is already in place it can still affect users.

In the last stage the goal is to fix stage three by using OutputEncoding to strip any invalid characters after it is read from the database.

Summary
Focus, intent and prevention are all achieved with this lab. The warning about this lesson being only in the developer version needs to be removed. A minor flaw in the lesson is lack of information or hints on tomcat deployment. Without this information a student cannot finish the lesson even if he/she knows how to add the solution and in the right place.

5.4.2 Cross Site Request Forgery

The task is to send an email containing malicious content to a newsgroup. The attack uses a HTML request disguised as a image in 1x1 pixels, making it invisible for the reader. The purpose is for example, to transfer funds from the readers' bank-account if the person has an authenticated session to their bank in the browser.

When the email is unsealed the HTML code will try to retrieve an image but instead activate the malicious link hidden within the code. In the lesson it is a link to the WebGoat application but it has an extra parameter for a HTML request and thereby making it malicious. The user is given one input field and a text area. After submitting an e-mail, it is will appear further down on the page under a list of incoming emails. By submitting an email with the malicious code the lesson is completed.

Summary
The lesson plan is rich with information and gives an good background to the problem. The goal of the lesson is clear from the start and no advanced coding experience is needed to complete the task. Using basic knowledge of HTML a student can successfully complete the lesson by including a malicious link in the text area. The hints provide enough information to get started. So if a student is struggling eventually he gets a hint to push in the right direction. Although the solution is not given too early and still gives the student a chance to complete it on their own.

All criteria are fulfilled.

5.5 Denial of Service

5.5.1 Denial of Service from Multiple Logins

Denial of Service is a form of attack which can prevent access to a web applications database that is restricted to only a few connections at the time. The attacker can block access for other users by filling the database connection pool which could give devastating consequences for a business due to both time and money are wasted. The purpose of this lesson is to retrieve a list of valid users and use that information to login and fill the connection pool. The pool allows a maximum of two connections and has to be filled with logins from three different accounts. The user is given two input fields and a submit button.

The lesson plan does not contain any more information than the one given above and more is not necessary as focus and the intent of the lesson is explained in detail.

If a student previously carried out lessons under the 'Injection Flaws' category, she might suspect that this is a SQL injection attack. If not, it might be difficult to understand how the problem should be handled and a lot of time will be spent on figuring out how to start without using the hints.

It is revealed in the first hint to use SQL injection. By tampering with the input the user is able to access a table containing login information for all users in the database. Although, to ask a question in SQL you need to include the name of the table you want to access. That information is first revealed after submitting any data on the site and the whole question to the database is shown for the user. This information is also presented in the second hint but that hint is in essence almost a complete solution to the problem which leaves no room for the user to formulate an own solution. Once the list of accounts are exposed, the lesson is easily completed by logging on using three of the acquired accounts.

Summary
For a user familiar with SQL injection this lesson is very straightforward and easy to finish. Although there is no information given about the attack involved which can cause confusion on how to get started. This is given in the first hint but we can not access any data without a name of the table in the database and still leaves confusion for the student.

This confusion can be avoided by moving the lesson into the 'Injection Flaws' category. Due to this form of SQL injection is a substantial part in the lessons given there. It is also recommended that the lesson plan should provide more information on the attack itself so the user can get a good enough understanding of threat. The improvement would fulfill the requirements of the *understanding* part of the PBL process. It is an interesting problem in web application development and should contain a follow-up lesson in how to prevent this type of attack in some form of a lab exercise.

Because of lack on intent and prevention this lesson does not fulfill the criteria.

5.6 Injection Flaws

The most common attack according to OWASP top ten list is SQL injection flaws. Mitigation techniques include stripping all special characters from any user input such as `'` and `"` in order for the user input to not be treated like a part of the SQL query or file access and not get executed. The severity from these types of attack ranges from low to severe depending on the context.

5.6.1 LAB: SQL Injection

We chose to do a lab containing four stages, each building on the next. The goal for the first stage is to bypass authentication without using a correct password by injection of SQL. In the lesson plan it is stated that one will also implement code changes in the web application to defeat these attacks, however this is in stage two. The background given is of interest to a really wide audience. It describes what SQL injection is in general terms and is not quite enough to get started with the lesson unless the student has previous experience with SQL. This needs to be corrected, by at least a link to more on SQL and injection attacks, otherwise the student is stuck.

It is apparent that one needs to enter a string into the password field that will bypass the authentication. Without SQL knowledge, a hint is needed. The first hint says the application is taking your input and inserting it at the end of a pre-formed SQL query. This information could be in the background, nothing new here to push the student in the right direction. Looking at the source one can see the input field has a limit of only 8 characters, which is not enough for exploitation. Now a familiar user with HTTP should realize that this limit is only enforced by the web browser and so can by bypassed by telnet or a proxy interceptor. If the user is not familiar with this, one more hint will reveal this.

A second hint showing the query the application builds, pushes the student in the right direction. Now it should be apparent that injecting code using the special characters `'` to infiltrate the query and issue a statement that always returns true such as `OR '1'='1';` Still if the student does not know SQL this is of little help. A link to an external SQL tutorial should be available in the background or as a hint.

Using the web proxy WebScarab which is recommended by OWASP to be used together with WebGoat, one can intercept the HTTP headers, and modify the password field to more than 8 characters, and thus execute the statement which always returns true. The stage is then completed. The full solution is given in a hint which is `txt' OR '1' = '1`. This solution in the form of a hint is acceptable since by now a student should have tried a myriad of variations on the same theme from the previous hints, thus gaining experience with SQL injections.

What is learned by the user at this stage is how to exploit injection attacks successfully using *trial and error*. What is needed to improve this lesson is a link to more on SQL for students without a basic understanding of it.

Stage two

The goal is to prevent the injection in the previous stage by modifying the lesson to use a parameterized query, or so called prepared statement.

The lesson plan provides no new information on this stage, and the background is the same. Here we think it is much preferred to point the student to tomcat deployment documentation or suggest how the changes the student makes can take effect. Without knowing this, a student is stuck, and the hints do not provide any help. The student needs to know where the file is, how to change it, how to recompile it and how to make tomcat reread the class file.

The one hint given is to "search for PreparedStatements in the other lessons and so see how one can change the lesson code to use it". We suggest a few more hints such as "The file to change is `WebGoatINSTALLDIR/tomcat/webapps/webgoat/WEB-INF/classes/java/org/owasp/webgoat/lessons/SQLInjection/Login.java` and this compiled file goes to `WebGoatINSTALLDIR/tomcat/webapps/webgoat/WEB-INF/classes/org/owasp/webgoat/lessons/SQLInjection/Login.class`. Notice the omission of java in the path". And "Tomcat redeploys applications from the .war file found in the webapps directory automatically when it notices a change, add your new class file to the webgoat.war file". In the lesson plan of stage two we suggest a link to *http://tomcat.apache.org/tomcat-5.5-doc/deployer-howto.html* is in order with a note to read the "Deploying on a Running tomcat server" part.

This stage is very good since it teaches how to prevent SQL injection flaws in real code. The lesson needs more hints on how to change lesson code and text on tomcat deployment procedures.

Stage three

Here the goal is to use 'numeric SQL injection' to bypass authorization. You get login credential to login and view your own profile, but the lesson is finished only when you view the profile of another user with id 112. Since the page sends an 'employee_id' together with requested action 'view' it seemed natural to intercept the current users id and give it 112 instead, but this gave a strange error page only saying "An error has occurred" with no information in the logs. A hint gave away the answer which is `101 OR 1=1 ORDER BY salary desc`. Which means there is a check for if the currently logged in user is the requested profile id. This check is bypassed by using the above string. It gives the currently logged in users ID, 101, but appends something which is always true and asks to sort it by salary, which happens to return another user first than the current users id.

Improvement for this lesson is to tell the student "Nice try" instead of the strange error, as the error might lead the student to check if his/hers installation of WebGoat is working correctly. Another improvement is to tell the user to select an ID to view if the button 'ViewProfile' is pushed with no user selected instead of the current solution which points to an error page.

Stage four

The goal is to fix the previous lesson so it is not allowed to bypass authorization. This is done by using PreparedStatements in the getEmployeeProfile method in `ViewProfile.java`. The lesson is very similar to lesson two and serves only to further exercise using PreparedStatements and to notice the difference in exploit ability compared to regular statements.

Summary
This lab is a good exercise on how to exploit injection flaws and also teaches how to prevent them by fixing the code.

This lab needs better background information with the intent to form a better understanding of SQL. Either as a link to an external source, such as OWASP, or internally in the lesson background text. A student needs to know SQL to finish these lessons. Another suggested improvement is to tell the student how the changes he/she makes to the lessons code can take effect.

The lab fulfills all criteria.

5.6.2 Modify data with SQL Injection

The goal of this lesson is to modify the data of a relational database. The lesson lacks a lesson plan, but the hints are good, from the first "SQL Injections accept more statements", second "Use a `;`" to the next-to-last hint which gives a link on how to use update statements. It is possible to complete the lesson without using the hints if a student is familiar with SQL. The last hint gives the solution.

What a student learns from completing this lesson is that for any SQL injection it is possible to add more statements than just the current one by appending a delimiter `;` to the injection and writing another statement which is executed as well.

By looking at other SQL injection type flaws we found the lesson plan states the background to the problem and in terms how to avoid injection attacks in your own application, to sanitize user input. How this is done however, is left as an exercise. We believe the lessons of this type of flaws need more code, either in the lesson plan or provided on a course in web application security. How to sanitize user input is not given in the lesson. This needs to be corrected.

Summary
The lesson is a nice exercise on how to use the update statement to modify data. It needs a lesson plan, preferably keeping the student on the task by explaining the update statement. Move the solution from the hint to the solution page.

5.6.3 Command Injection

The goal is to execute any command on the hosting operating system. The lesson contains a scroll-list of files from which the student can select to view in the page. Since the default files are help or lesson files for other lessons it is confusing to see which text that belong to the current lesson and how to continue.

The lesson is finished when intercepting the request and changing the parameter of the filename to view to a system command, such as netstat. This happens because the lesson code

uses a system command to show the contents of a file, the system command is a parameter so it can be changed to run any other command.

Here we learn of another type of injection flaw aside from SQL injections. However we do not learn how to avoid it. To do so the student has to look at the source and make sense of the flaw herself.

Summary
This lesson lacks in focus and intent since the default files in view are plans from other lessons. The lesson also lacks in the prevention criteria, no code is shown on where and why this vulnerability happens. The main idea however is shown.

5.7 Malicious Execution

5.7.1 Malicious File Execution

The goal is to upload and run a malicious file. It should create a new file 'guest.txt' in a specific directory on the local computer when executed. You are given an input field and an upload button. Once an image has been uploaded it will be shown in the lesson page. The lesson lacks a lesson plan which leaves limited understanding of the problem. The hints phowever rovide a good guidance toward a solution. The first hint encourages the student to find out where the uploaded files are stored and to if you can browse them directly. By uploading an image and checking the source code the destination of the file has been discovered and is easy to browse. If the application accepts file types other than images it would be possible to upload a executable file.

The second hint suggests a .JSP file that creates an instance of the class `java.io.File` and calling the `createNewFile()` method of that instance. The third hint provides external references on how to create .JSP files and to the Java API for the `java.io.File` class. Using the provided information it is possible to construct a .JSP file that will create 'guest.txt' when executed. Because the lesson opens the file after it has been uploaded, the site will run the program and the malicious file is created.

Summary
How to prevent this attack is not discussed. However it would be reasonable not to allow executable files to be uploaded. A possible solution could be to check the file extension. Also mark the upload directory of user files in your container as non-executable. This information could be added to the congratulations page that is shown when the lesson is completed. The lesson plan is not necessary as the hints are good.

Lesson does not fulfill the criteria because it lacks of information on how such an attack is prevented.

5.8 Parameter Tampering

5.8.1 Bypass Client Side Script Validation

The lesson is about bypassing a client-side validation. The validation is built-up using seven input fields which accept different types of data. All of the validators have to be broken at the same time. The lesson plan has a good background and the goal is very well defined.

The first hint explains that the validation happens in the browser. If the validation is done in the browser then the solution is to pass the server unvalidated data using a proxy to intercept the connection or similar tools. The second hint suggests using a proxy and trying to modify the values after they leave the browser. By using WebScarab to intercept requests we can catch the values entered in the input fields and modify them to include unwanted characters. The client-side validation has thereby been bypassed and invalid values have been sent to the server. Lesson successfully completed. Principals of client-server validation, never trust the client. The client can always send bad data to the server.

Summary
This lesson is very well written and fulfills all criteria. No improvements necessary.

5.9 Session Management Flaws

5.9.1 Hijack a Session

The goal of this lesson is to authenticate using a spoofed cookie that is already authenticated. The session cookie is not randomly set and so allows to brute force a valid one and acquire a session without having provided valid credentials. The lesson plan explains why these kind of flaws exists, due to neglect of application developers to provide fully random session data. More than this is needed, or a link to read more on hijacking a session cookie since it may not be apparent for students what tools to use to see the cookie pattern.

The first hint is good, "the server skips authentication if you send the right cookie". This means don't bother with the input fields and guessing passwords, just try to guess the cookie. Selecting 'Show Cookies' from the menu link shows a WEAKID cookie and its value. A downside of this lesson is that the weak cookie is saved the first time it is received by the browser. Without using a web proxy such as WebScarab the same cookie will be sent by the browser and the student has no chance to see that the pattern of new cookies. For users familiar with session cookies this is a trivial task, however this needs to be explained in the lesson plan.

The second hint “Is the cookie value predictable? Can you see gaps where someone has acquired a cookie?” Still no luck without seeing how the cookie changes. A better hint would be "Your browser saves the cookie, use a web proxy for this". Using WebScarab one can see that for every subsequent request the same cookie is set by the browser. One way to request new cookie is done by deleting the old one in the header. This would have been a very good hint, for example “Your browser saves the first cookie, to see new cookies you have to unset the previous one. Find a way to delete it for every request you make so you can see the

pattern”. Another hint would be "Try WebScarab". Lets see the third hint since the second one did not help. “Try harder you brute.” Does not really help. But the fourth hint “The first part of the cookie is a sequential number, the second part is milliseconds”. Now we have a pattern.

Instead of these vague "hints" we could have told the student about cookies, and to use WebScarab to fetch for example 50 new cookies and analyze these, then see the pattern themselves.

The student should learn how to use WebScarabs built-in session id analyzing tool. This tool will help reveal the pattern and acquire many cookies. By removing the WEAKID from the request header, it will be set for any subsequent requests. It is then possible to acquire 50 new cookies and analyze these. Using the second hint “can you see gaps where someone has acquire a cookie?” the student can see in the analysis part where the sequence-number skips one, then try to guess the milliseconds in between those sequence numbers with a brute forcing tool such as Jhijack or crowbar.

Summary

This lesson is very difficult since few good hints are given and the lesson plan explains the reasons for the flaw but does not push the student in how to solve the lesson. A prevention for this flaw is to use more random data. Randomization of data is a delicate subject and is left as an exercise for the user to research for her favorite programming language.

This lesson is completely problem-solving, taking much time and resources of the student to solve a simple task. We suggest better hints and background material to improve the lesson.

6. Conclusion and General Improvements

In general the WebGoat application fulfills the understanding, design and write phases of the PBL model. As we have seen the majority of the lessons have an informative lesson plan and well defined goals. The focus and intent criteria are satisfied. The hints correspond to the design phase and this criteria is also fulfilled. Although many of these hints could be overlooked, they are like a nice guide which lead the student in a good progression through the lessons. Good solutions are available in most of the lessons. But when it comes to the reviewing phase, it is a recurring problem that the lessons lack in describing or showing how to prevent the attack. This lack breaks the cycle of the PBL four-phase cycle. In order to complete that cycle in WebGoats teaching approach we suggest the following improvements.

One improvement would be to show 'background code' for any lesson which shows the main threat under investigation. It is now possible to show the source code of any lesson. However the lesson code contains other methods and definitions which are not needed to understand the threat and more importantly how to mitigate it. We suggest further that this 'background code' gives the secure way of performing the same actions without the vulnerabilities, in clear code.

In accordance with a problem-solving approach and a behaviouristic approach to teaching this 'background code' should only be made available when the student completes the lesson. This is an important step to focus on the reviewing part of the problem solving cycle.

WebGoat should be used in conjunction with a course in web application security if the student wishes to possess deeper knowledge of web applications. It is enough to use it to learn how to exploit vulnerabilities, but it takes more to learn how to avoid them. The 'background code' is of significant importance here, as well as reading about the specific vulnerability in other sources to give more details into how the problem is introduced and how to avoid it, with code-examples.

With this simple addition of 'background code' the WebGoat teaching platform can move away from being only a problem-solving approach. The students are then given the choice to study the solutions in detail and prepare for similar problems in their programming.

If WebGoat is to be used as basis for a web application security course at a university it is important to provide discussions on the problems and solutions. In particular the reviewing phase of the problem-solving approach needs more care in such a course, as WebGoat does not provide it. We suggest the students are asked to write explanations for what they did and why the lessons worked as they did.

7. Discussion

Evaluation of the chosen lessons in WebGoat has given us interesting results and new insights in teaching a difficult subject. As the application is based on a problem based approach it falls short on teaching and showing the students how to prevent the flaws found. For most of the lesson the prevention is implied, however we think an approach with crucial instructions on prevention will give better results in improving web application security overall than learning how to exploit successfully. This criticism coincides with the major criticism of PBL. The effect of this teaching method gives the student a mere increase in speed when solving these problems. Students who recieve crucial instruction do as well and in some cases even better. Our suggestions for a future course in web application security is to include WebGoat together readings and discussions of the problems. The reviewing phase needs to be strenghtened and one way to do this is by discussions with other students and teachers.

8. References

[1] Huang, Yao-Wen & Huang, Shih-Kun & Lin, Tsung-Po, *Web Application Security Assessment by Fault Injection and Behavior Monitoring.* Taiwan: Academia Sinica, Institute of Information Science 2003.

[2] Barg, Mike & Fekete, Alan & Greening, Tony & Hollands, Owen & Kay, Judy & Kingston, Jeffrey H, & Crawford, Kathryn, "Problem-based Learning for Foundation Computer Science Courses", In: *Computer Science Education,* Routledge 2000, Volume 10, Issue 2, pages 109-128.

[3] Barnes, David J & Fincher, Sally & Thompson, Simon, "Introductory Problem Solving in Computer Science". In: *5th Annual Conference on the Teaching of Computing,* Daughton, Goretti & Magee, Patricia (ed.) Dublin: Dublin City University 1997, pages 36-39.

[4] Kaasbøll, Jens J, *Exploring didactic models for programming,* University of Oslo, Department of Informatics 1999.

[5] Stuttard, Dafydd & Pinto, Marcus, *Web Application Hacker's Handbook - Discovering and Exploiting Security Flaws.* Indianapolis: Wiley Publishing Inc. 2008.

[6] Boud, David & Feletti, Grahame, *The Challenge of Problem-Based Learning*, Second Edition. London: Kogan Page 1997.

[7] Kirschner, Paul A. & Sweller, John & Clark, Richard E.,"Why Minimal Guidance During Instruction Does Not Work: An Analysis of the Failure of Constructivist, Discovery, Problem-Based, Experiential, and Inquiry-Based Teaching". In: *Educational Psychologist*, New Jersey: Lawrence Erlbaum Associates, Inc. 2006, Volume 41, Issue 2, pages 75-86.