Exploiting Web Applications - Instructor's Guide

by Bruce Leban, Google Inc.

The Exploiting Web Applications codelab (aka the "Gruyere codelab") is a self-paced and self-contained course teaching students how attackers exploit web applications and how software developers can protect their applications. Students work through a series of exercises covering a wide variety of different vulnerabilities and are guided to find exploits for them.

The codelab is built around Gruyere, a small yet full-featured microblogging application with lots of security bugs. The source code to Gruyere is published under a Creative Commons license that allows you to incorporate excerpts from the code in your course materials and allows students to use the code in white box hacking exercises, to code review and to try and fix bugs.

The codelab is suitable for incorporating in computer science curricula on security, software engineering or general software development. Because the codelab necessarily includes the answers to the exercises, it is not suitable as a graded assignment. The best way to incorporate the codelab in a class is to present complementary lecture material on web security. Google Code University has published course materials entitled "What Every Web Programmer Needs To Know About Security" by Arkajit Dey and Neil Daswani and "Introduction to Web Security" by Neil Daswani which are suitable for this purpose.

After the codelab, you may want to assign students one of the companion exercises on the next page.

To get started, visit http://google-gruyere.appspot.com

Companion exercises for Exploiting Web Applications codelab

(Difficulty level: ☆ to ☆☆☆☆)

☆ (1) Write a function to convert a user name containing arbitrary characters into a safe name to use for a file system directory or HTML identifier. Your function should preserve readability of alphanumeric parts of the name and be invertible.

☆ (2) Write a function to sanitize a file system path provided by a user so that it can safely be passed directly to the open function.

☆☆ (3) Implement a better password storage mechanism for Gruyere.

☆☆ (4) Write a program to crack Gruyere's cookies. Given three valid cookies and without knowing the cookie secret, your program should generate a new valid cookie for any user.

☆☆ (5) Implement a better, more secure, cookie mechanism for Gruyere.

☆☆☆ (6) Fix or rewrite Gruyere's HTML sanitizer. It should accept all the HTML tags listed in jsanitize.py.

☆☆☆ (7) Write unit tests that verify Gruyere's HTML sanitizer is correct. Your tests should be more than just string comparisons of bad input to good output, as it should not depend on specific implementation of the sanitizer. You can assume that non-dangerous HTML is left alone and that dangerous HTML is removed or replaced, but should not depend on the specific way that the transformation is performed.

☆☆☆☆ (8) Fix or rewrite Gruyere's template system.

☆☆☆ (9) Write unit tests to verify that Gruyere's system is correct.

In several of these cases (2, 6, and 8 in particular), after students submit their solutions they can be challenged to break each other's solutions. You could, for example, assign problems (6) and (7) in combination and then apply each student's unit tests to the other students' fixed sanitizers.

You can also use these as live coding challenges ("iron coder" style): students (or teams of students) are given a fixed amount of time to write the code and then a fixed amount of time to break each other's code.