IMPACT OF AUTO-GRADING OF AN INTRODUCTORY COMPUTING COURSE

AUTHOR MCAUTHORSON, AUTHOR MCAUTHORSON, AUTHOR MCAUTHORSON, AUTHOR MCAUTHORSON, AND AUTHOR MCAUTHORSON

1. Introduction

2. Technical Description of Bottlenose

Our web application, Bottlenose, was initially developed to support the teaching of a "flipped" course, where students would watch video lectures online before class to prepare for classroom questions and discussion. The design of Bottlenose also called for online submission and grading of programming assignments, which turns out to be a useful piece of functionality even for traditional courses.

Bottlenose is built using the Ruby on Rails[1] web application development framework. This framework has allowed the application to be built rapidly, but also provides built in automated testing infrastructure which should help the application stay high quality and maintainable as it grows. The application follows standard Rails conventions. A PostgreSQL[2] database is used to store most application state, although student submissions are stored on the file system.

A simple process for online submission of assignments is provided. Each student is emailed an authentication link which brings the to their list of assignments and identifies them to the application. Assignments are submitted by uploading the programming code directly in their web browser. Both single-file assignments and more complicated assignments (submitted as a gzipped tarball) are supported. The automated grading process begins immediately when an assignment is submitted, giving students feedback within a few seconds. Students may attempt submissions multiple times.

In order to automatically grade student programs, submissions are compiled and run on the server. Allowing students to run arbitrary code on the server is clearly a potential security issue, so Bottlenose uses a sandbox mechanism to prevent student programs from causing problems. Five major techniques are used to isolate student programs from the rest of the system:

(1) **Separate system user** - Each student program is run under a separate system user with minimal Unix permissions.
(2) **Run in a "chroot"** - Student programs can only access specific, whitelisted parts of the file system.
(3) **Resource limits** - The "setrlimit" system call is used to set limits on the use of a variety of resources, including RAM, child processes, and created file size.
(4) **Isolated working directory** - Each program is executed in a separate "tmpfs" filesystem which ceases to exist when the grading process finishes.
(5) **Watchdog timer** - A grading process is terminated if it lasts more than five minutes.

[1] http://rubyonrails.org/
[2] http://postgres.org/

This sandbox mechanism isn't foolproof, but it has worked reasonably well at preventing the grading server from being disrupted by common student mistakes like infinite memory allocation loops.