Spring 2014

Automatic Evaluation of Python and C Programs with codecheck

Kiet Nguyen
San Jose State University

Automatic Evaluation of Python and C Programs with codecheck

A Thesis

Presented to

The Faculty of the Department of Computer Science

San José State University

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

Kiet Nguyen

May 2014

The Designated Thesis Committee Approves the Thesis Titled

Automatic Evaluation of Python and C Programs with codecheck

by

Kiet Nguyen

APPROVED FOR THE DEPARTMENT OF COMPUTER SCIENCE

SAN JOSÉ STATE UNIVERSITY

May 2014

_____

Dr. Horstmann Department of Computer Science

_____

Dr. Austin Department of Computer Science

_____

Dr. Tseng Department of Computer Science

Table of Content

Abstract

This project enhances the codecheck autograder by implementing automatic evaluation of C and Python programs. Two security approaches are implemented and analyzed in order to complete this goal. The first approach involves isolation by using virtualization and the second approach involves hardening of the host operating system. I describe both implementations and measure their performance levels to see which approach is more efficient.

1. Introduction

Codecheck (http://horstmann.com/codecheck/) is a “convention over configuration” unit test library for automatic evaluation of student programs. It helps minimize the authoring work of an instructor. Currently, codecheck can only evaluate Java programs. The goal of this project is to enable codecheck to also evaluate Python and C programs.

Like Java, Python and C are very popular and powerful programming languages that are taught in many colleges and universities. But unlike Java which runs inside a Java Virtual Machine (JVM) that offers built in security features to avoid malicious Java programs from corrupting the operating system, Python and C programs don’t have that kind of built in security and thus running them when grading students’ programs poses a security issue.

This project will be focused on implementation and comparison of two approaches used to securely evaluate Python and C programs. The first approach is to use a Linux container such as Docker (https://www.docker.io/) to securely execute arbitrary programs. In this approach, each program will be executed inside an isolated Linux container to protect the host system. The second approach is to harden the host operating system. In this approach, a set of rules are used to prevent malicious code from performing unwanted operations on the host system. These two approaches differ in performance, resource usage, and level of effectiveness. The focus of this project will be to implement and explore the strengths and weaknesses of these two approaches.

2. Pre-existing Work

According to [6], automated assessment of programming assignments has been explored for decades. Beginning with Forsythe and Worth's work in 1960's, many automatic grading systems have been developed such as Kassandra on scientific computing [6], Ceilidh on grading ML programs [6], Leal and Moreira worked on discovering student problems early [6], and TRAKLA2 facilitated teaching data structures and algorithms [6]. More recently, new systems have been developed such as LabRat which grades Java program snippets [11], CodeLab [4] provides interactive grading of programming exercises, and Web-CAT [17] grades students on how well they test their code.

Int the following sections I describe some of the work that has been done in the subject of automatic evaluation. Refer to [6] for a more intensive list of pre-existing work.

2.1. Web-CAT

Web-CAT is an open source automated grading system developed by Stephen Edwards. It grades students on how well they test their own code. Web-CAT utilizes a web interface to provide all of its capabilities. With a web browser, the user can control all submission activities, give feedback, and view results and grading activities. Web-CAT grades assignments by coming up with three scores, namely the code completeness, test completeness, and test validity scores.

Web-CAT offers the ability to detect erroneous student programs and protection from malicious code. It also provides data integrity through the use of system security policies and a relational database. It was written to have a high degree of portability. By packaging under a Java servlet application, it will run under any compatible servlet container, for instance, Apache Tomcat. Besides, Web-CAT itself is completely language neutral. And it has been used to handle submissions in Java, Prolog, Scheme, C++, Standard ML, and Pascal.

A downside of Web-CAT is that it is highly geared towards unit testing and thus every assignment is required to have a JUnit test. Therefore it takes quite some time to author a

new problem. This makes it very time consuming for instructors to create weekly assignments.

2.2. Codelab

CodeLab works with Python, Java, C, C++, C#, JavaScript, Visual Basic, and SQL and it is also a web-based system like Web-CAT. According to the vendor, it was first offered in 2002 to help with teaching multiple programming languages, it is now a seasoned system that has been used in over 350 institutions worldwide and has analyzed over 60 million exercise submissions from more than 150,000 students.

CodeLab has over 800 short exercises. And each exercise is focused on a particular programming idea or language construct. The student types in code and the system provides instant feedback regarding its correctness. It even offers hints when the submission is incorrect so the students can fix their code. This process is designed to help the student with the semantics, syntax, and common usage of the language elements.

There are a few limitations to Codelab. First of all, there is no way to author new problems. Besides, each CodeLab problem is very simple and can be solved with a few lines of code. Therefore it does not contain enough logic as in a typical homework assignment. It was designed for students to explore simple programs and practice their coding skills instead of for instructors to use. Figures 1 and 2 demonstrate how Codelab works.

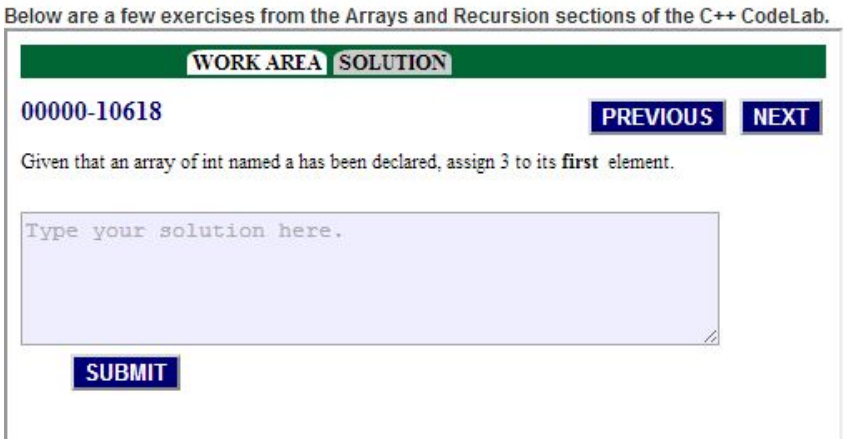


Figure 1: A sample Codelab exercise in C++

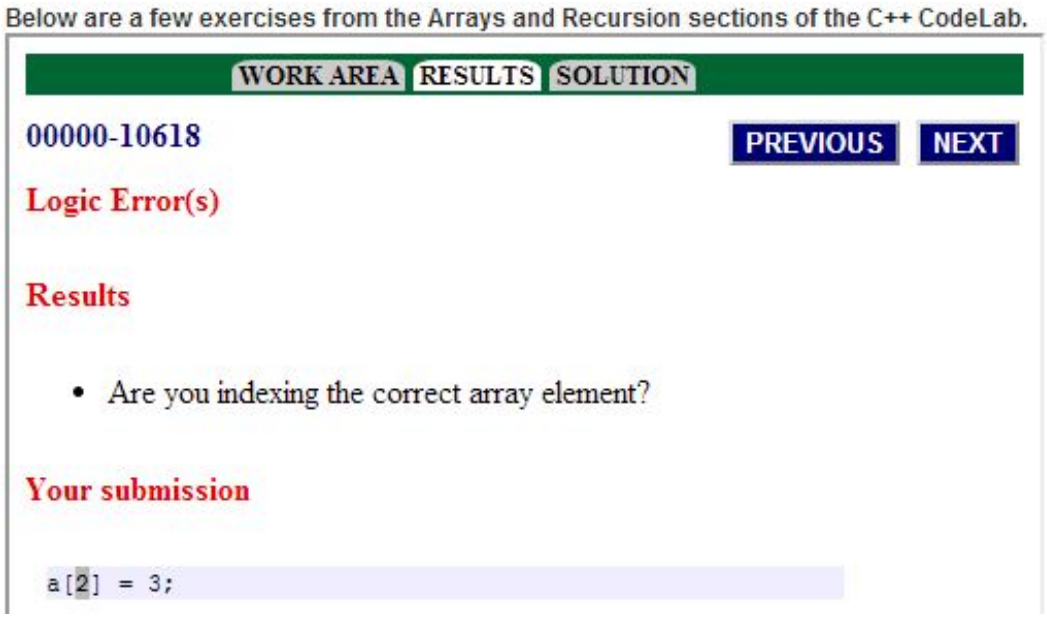


Figure 2: A sample result from a C++ exercise on Codelab

From those two figures it is visible that Codelab also provides feedback regarding logic errors in the student's code.

2.3. CodingBat

CodingBat is a free site of live coding problems to build coding skill in Java and Python. It was created by Stanford computer science lecturer Nick Parlante. The problems on CodingBat are short and provide immediate feedback in the browser. A limitation to CodingBat is that it is only available for Java and Python, not C. There are currently 13 Java problem groups and 8 Python problem groups available on CodingBat. Each problem group has a couple dozens exercises. Figure 3 and 4 show how CodingBat looks and what it does.

The parameter weekday is true if it is a weekday, and the parameter vacation is true if we are on vacation. We sleep in if it is not a weekday or we're on vacation. Return true if we sleep in.

sleepIn(false, false) → true
sleepIn(true, false) → false
sleepIn(false, true) → true

Go ...Save, Compile, Run Show Solution

```
public boolean sleepIn(boolean weekday, boolean vacation) {
  if (!weekday || vacation) {
    return true;
  }

  return false;
}
```

Go

Show output only (no red/green)

Figure 3: A sample Java exercise on CodingBat

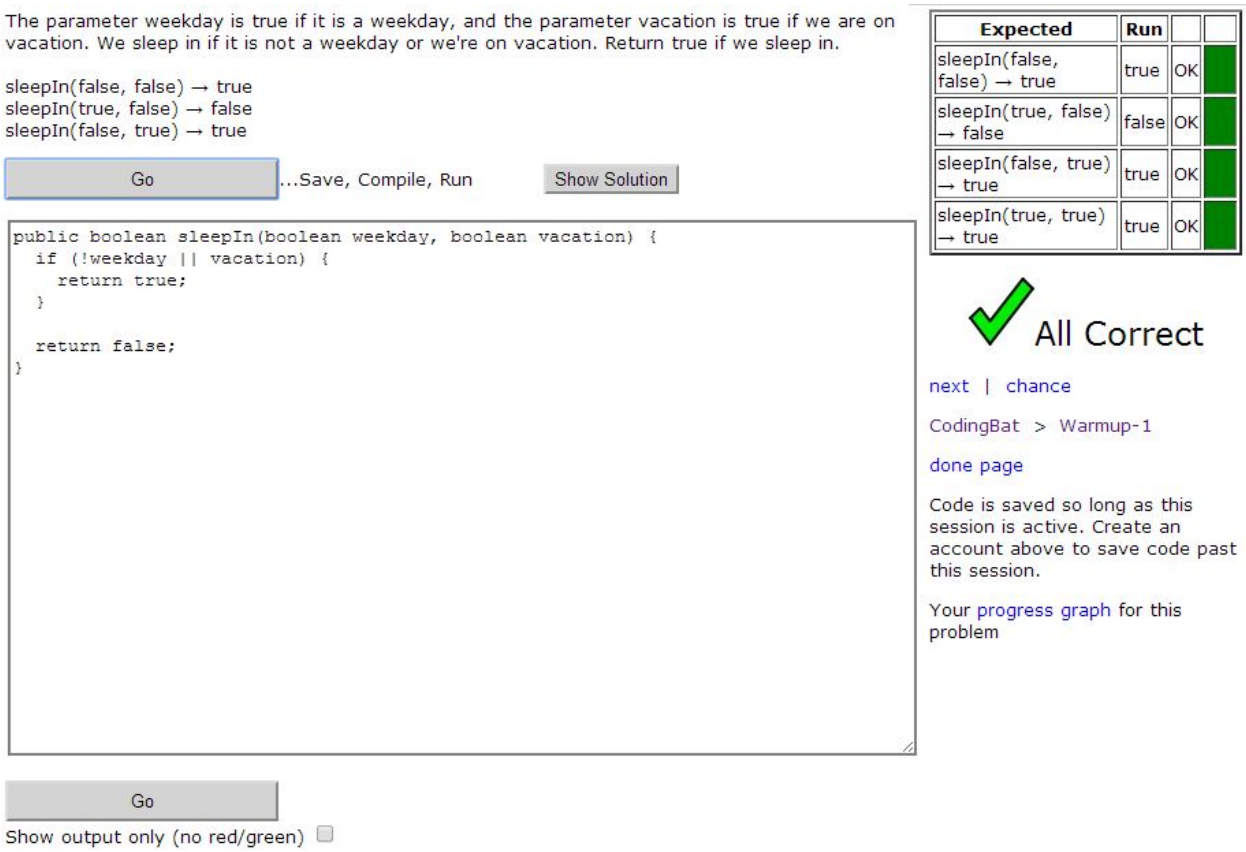


Figure 4: A sample submission from a Java exercise on CodingBat

To author a new CodingBat problem, the user supplies the documentation, some hints, tests to test the student code, the correct code, and tags to customize the user interface, for instance, showing solution. To author a Python problem, the process is similar as described except the `lang:python` tag is needed.

3. Codecheck

Codecheck (http://horstmann.com/codecheck/) is a “convention over configuration” unit test library for automatic evaluation of student programs. It helps minimize the authoring work of an instructor. Codecheck was developed by Cay Horstmann of San Jose State University.

For the instructors, codecheck provides an easy way to make an assignment. Specifically, the author of a problem provides a zip file that consists of an optional problem.html file and two folders named student and solution. In the student folder, codecheck expects an incomplete source code of the problem to display to the student in a textbox on a web page and files that are used for testing the student work (input files, tester programs, library files). And in the solution folder, codecheck expects the solution file.

After an assignment is submitted to codecheck, the instructor will receive a link pointing to the assignment. The instructor then distributes this link to the students. Figure 5 demonstrates how a codecheck problem looks like.

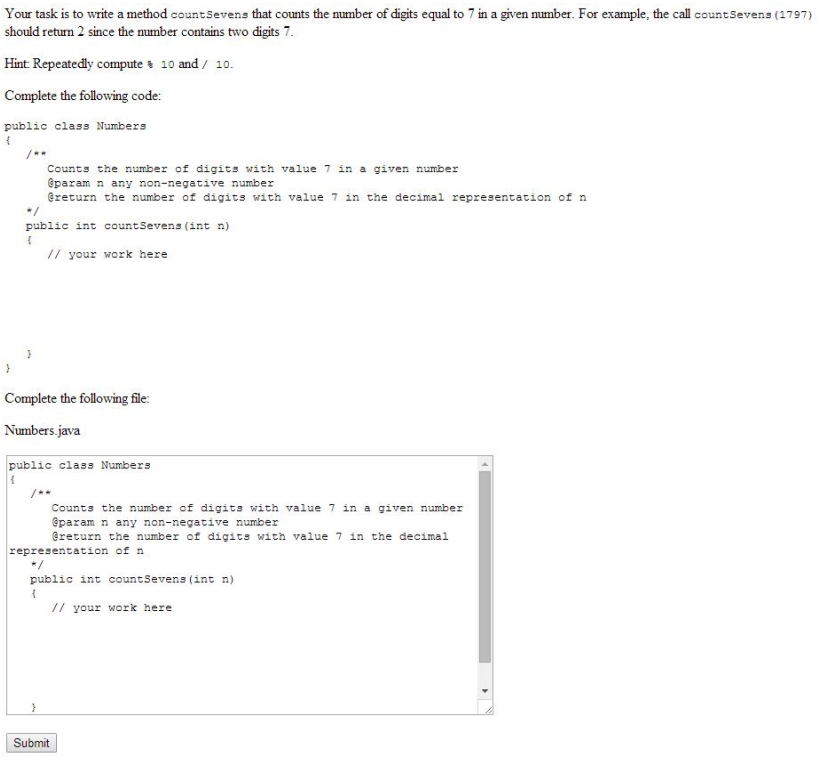


Figure 5: A sample codecheck problem

Once given the link to the assignment, the students fill in the code in the textbox and hit the Submit button. After that they will be given a score and a Download button to retrieve a signed zip file to submit to their instructor. Figure 6 shows how a codecheck result page looks like.

Calling method

Testing method countSevens

Arguments	Actual	Expected	
1797	2	2	pass
7777777	2	7	fail
1985	2	0	fail
7	2	1	fail
0	2	0	fail

Student files

Numbers.java:

```
1   public class Numbers
2   {
3      /**
4         Counts the number of digits with value 7 in a given number
5         @param n any non-negative number
6         @return the number of digits with value 7 in the decimal representation of n
7      */
8      public int countSevens(int n)
9      {
10        return 2;
11
12
13
14
15
16
17     }
18  }
```

Score

1/5

Download

Figure 6: A sample codecheck result page

One advantage of codecheck is that it requires no registration and thus eliminates the hassle of signing up and the inconvenience of logging in every time. For the implementor, this is also an advantage. There is no need to manage user accounts, passwords, or confidential information. The only persistent data is the problem database.

4. Implementation

In my project, I extend codecheck to handle C and Python. An extension to C++ can now be trivially done. In this section I will explain how the Docker approach and the apparmor approach have been implemented. But first let me explain the modifications I made to the original version of codecheck so that it can handle C and Python programs.

4.1. The Language Interface

First of all, Dr. Horstmann and I added a `Language` interface to codecheck. With the `Language` interface, any new language can now be added to codecheck. To do so, the `compile` and `run` methods of the new language need to be written. For Python, only the `run` method needs to be implemented since there is no compilation.

The code for the `Language` nnterface is given below

```
public interface Language {
    boolean isSource(Path p);
    boolean isTester(String modulename);
    boolean isMain(Path dir, Path p);
    String moduleOf(Path path);
    Path pathOf(String moduleName);
    boolean compile(String modulename, Path dir, Report report);
    String run(String mainclass, Path classpathDir, String args, String
        input, int timeoutMillis) throws IOException,
        ReflectiveOperationException;
    void writeTester(Path sourceDir, Path targetDir, Path file,
        List<String> modifiers, String name, List<String> argsList) throw
```

```
        IOException;
    String[] pseudoCommentDelimiters();
    Pattern variablePattern();
    String substitutionSeparator();
}
```

To enable codecheck to handle C programs, I implemented this `Language` interface to create two classes, namely `CLanguage.java` and `PythonLanguage.java`. The code for the `CLanguage.java` is given below

```
public class CLanguage implements Language {
    public boolean isSource(Path p) {
        return p.toString().endsWith(".c");
    }
    public boolean isTester(String modulename) {
        return false;
    }
    private static Pattern mainPattern = Pattern.compile("int|void\\s*main\\s+");
    public boolean isMain(Path dir, Path p) {
        if (!isSource(p))
            return false;
        String contents = Util.read(dir, p);
        System.out.println("CLanguage isMain(): " + (contents != null &&
            mainPattern.matcher(contents).find()));
        return contents != null && mainPattern.matcher(contents).find();
    }
    public String moduleOf(Path path) {
        String name = path.toString();
        if (!name.endsWith(".c"))
            return null;
        return name.substring(0, name.length() - 2);
    }
    public Path pathOf(String moduleName) {
        Path p = FileSystems.getDefault().getPath("", moduleName);
        Path parent = p.getParent();
```

```
    if (parent == null)
        return FileSystems.getDefault().getPath(moduleName + ".c");
    else
        return parent.resolve(p.getFileName().toString() + ".c");
}
public boolean compile(String modulename, Path dir, Report report) {
    String compilerMessage = "";
    try {
        Runtime r = Runtime.getRuntime();
        String compileCommand = "gcc -w -o " + dir.toString() + "/"  + modulename +
            " " + dir.toString() + "/" + modulename + ".c -lm ";
        System.out.println("CLanguage compile() $compileCommand = " +
            compileCommand);
        Process p = r.exec(compileCommand);
        p.waitFor();
        BufferedReader br = new BufferedReader(new
            InputStreamReader(p.getErrorStream()));
        while (br.ready()) {
            compilerMessage += br.readLine() + "\n";
        }
        if (!(compilerMessage.trim().equals(""))) {
            report.error("Compiler error", compilerMessage);
            System.out.println("CLanguage compile() $return = FALSE");
            return false;
        }
    }
    catch (Exception e) {
        report.error("Cannot compile", e.getMessage());
        return false;
    }
    return true;
}
public String run(String mainclass, Path classpathDir, String args,
        String input, int timeoutMillis) throws IOException,
        ReflectiveOperationException {
    String result, execCommand = "";
```

```
try {
    Runtime r = Runtime.getRuntime();
    try {
        Process p;
        if (input != null) {
            execCommand = classpathDir + "/" + mainclass + (args == null ||
                args.trim().equals("") ? "" : " " + args  + " 2>&1");
            System.out.println("CLanguage run() $execCommand = " +
                execCommand);
            p = r.exec(execCommand);
            p.getOutputStream();
            p.getOutputStream().write(input.getBytes(Charset.
                forName("UTF-8")));
            p.getOutputStream().flush();
            p.waitFor();
        }
        else {
            execCommand = classpathDir + "/" + mainclass + (args == null ||
                args.trim().equals("") ? "" : " " + args) + " 2>&1";
            System.out.println("CLanguage run() $execCommand = " +
                execCommand);
            p = r.exec(execCommand);
            p.waitFor();
        }
        BufferedReader br = new BufferedReader(new
            InputStreamReader(p.getInputStream()));
        while (br.ready()) {
            result += br.readLine() + "\n";
        }
    } catch (Exception e1) {
        e1.printStackTrace();
    }
}
catch (Exception e) {
    System.out.println("!!! Main run() ERROR !!! " + e.getMessage());
}
```

```
    System.out.println("CLanguage run() $result = " + result);
    return result;
}

public void writeTester(Path sourceDir, Path targetDir, Path file,
        List<String> modifiers, String name, List<String> argsList)
        throws IOException {
    String className = moduleOf(Util.tail(file));
    String returnType = "";
    String parameterType = "";
    List<String> lines = Util.readLines(sourceDir.resolve(file));
    List<String> studentLines = Util.readLines(targetDir.resolve(className +
        ".c"));

    // append codecheck to solution function
    int i = 0;
    while (i < lines.size() && !lines.get(i).contains(className)) {
        i++;
    }
    if (i == lines.size())
        throw new RuntimeException("Can't find function " + className +
            " for inserting CALL in " + file);
    lines.set(i, lines.get(i).replace(className, className + "codeCheck"));

    // find return type and parameter type
    String trimmedLine = lines.get(i).trim();
    returnType = trimmedLine.substring(0, trimmedLine.indexOf(" "));
    parameterType = trimmedLine.substring(trimmedLine.indexOf("("),
        trimmedLine.lastIndexOf(")"));

    // append student lines
    for (int x = 0; x < studentLines.size(); x++) {
        lines.add(studentLines.get(x));
    }

    // write main method
```

```
    lines.add("int main( int argc, const char* argv[] ){");
    for (int k = 0; k < argsList.size(); k++) {
        lines.add("   if (strcmp(argv[1], \"" + (k + 1) + "\") == 0) {");
        lines.add("      " + returnType + " expected = " + className + "codeCheck("
            + argsList.get(k) + ");");
        if (returnType.contains("char")) {
            lines.add("      printf(\"%s\\n\", expected);");
        } else if (returnType.contains("float")) {
            lines.add("      printf(\"%f\\n\", expected);");
        } else {
            lines.add("      printf(\"%d\\n\", expected);");
        }
        lines.add("      " + returnType + " actual = " + className + "(" +
            argsList.get(k) + ");");
        if (returnType.contains("char")) {
            lines.add("      printf(\"%s\\n\", actual);");
        } else if (returnType.contains("float")) {
            lines.add("      printf(\"%f\\n\", actual);");
        } else {
            lines.add("      printf(\"%d\\n\", actual);");
        }
        if (returnType.contains("char*")) {
            lines.add("      if (strcmp(actual, expected) == 0) printf(\"%s\",
                \"true\");");
            lines.add("      else printf(\"%s\\n\", \"false\");");
        } else {
            lines.add("      if (actual == expected) printf(\"%s\", \"true\");");
            lines.add("      else printf(\"%s\\n\", \"false\");");
        }
        lines.add("   }");
    }
    lines.add("}");
    Files.write(targetDir.resolve(pathOf(className + "CodeCheck")), lines,
        StandardCharsets.UTF_8);
}
```

```
    public String[] pseudoCommentDelimiters() { return new String[] { "////", "" };
    private static String patternString =
        ".*\\S\\s+(\\p{javaJavaIdentifierStart}\\p{javaJavaIdentifierPart}*)\\s*=
        \\s*([^\\s;]+)\\s*;.*";
    private static Pattern pattern = Pattern.compile(patternString);
    public Pattern variablePattern() { return pattern; }
    public String substitutionSeparator() { return ";"; }
}
```

In a typical call, the calling method first calls the `compile` method of the `language` class, if `compile` returns true then it proceeds to call the `run` method. This process is shown in the code snippet below. This code snippet has been taken out of the `doCalls` method in the `Main` class. Notice the `run` method is called in the third line of the code snippet.

```
if (compile(mainmodule)) {
    for (int i = 0; i < calls.getSize(); i++) {
        String result = language.run(mainmodule, workDir, "" + (i + 1), null, timeout);
        Scanner in = new Scanner(result);
        List<String> lines = new ArrayList<>();
        while (in.hasNextLine()) lines.add(in.nextLine());
        in.close();
        args[i][0] = calls.getArgs(i);
        if (lines.size() == 3 && Arrays.asList("true", "false").contains(lines.get(2)))
                {
            expected[i] = lines.get(0);
            actual[i] = lines.get(1);
            outcomes[i] = lines.get(2).equals("true");
         } else {
            expected[i] = lines.size() > 0 ? lines.get(0) : "???";
            actual[i] = lines.size() > 1 ? lines.get(1) : "???";
            outcomes[i] = false;
         }
         score.pass(outcomes[i]);
    }
    report.runTable(new String[] { "Arguments" }, args, actual, expected, outcomes);
}
```

4.2. Docker Approach

4.2.1. Rationale For Using Docker

Docker is an open-source engine that automates the deployment of any application as a lightweight, portable, and self-sufficient container that will run virtually anywhere. It is ideal for creation of lightweight, private Platform-as-a-Service environments. To achieve that goal Docker uses the Linux container technology. The biggest advantage Docker offers compared to using a full stack virtualization solution is performance.

For the purpose of this project, Docker seems like a natural approach to implement because doing so ensures that each student's program runs inside an isolated container. This is desirable because whatever happens inside a container stays inside that container and so it cannot affect the host system nor any other students' programs.Moreover, since a container is cheap to create, the system does not suffer too much from the security/performance tradeoff.

4.2.2. Cron Jobs For Docker Maintenance:

Since each request requires a new container to be created, it is crucial to kill containers that have exceeded the time limit and to remove containers that have finished running to remove the load on the system. To do that, two cron jobs have been created and are listed below.

1. Kill Docker Containers That Have Been Running Over a Timeout Limit

```
#!/bin/bash
docker kill `docker ps -a | grep 'Up.*minute\|Up [1-5][0-9] second' | awk
    '{print $1}'`
```

This script uses the Docker kill command to kill all containers that have been running longer than 10 seconds.

2. Clean Up Docker Containers That Have Been Killed or Have Exited

```
#!/bin/bash
docker kill `docker ps -a | grep 'minutes\|hour\|hours\|day\|days' | awk
    '{print $1}'`
docker rm `docker ps -a | grep 'minutes\|hour\|hours\|day\|days' | awk
```

```
    '{print $1}'`
docker rmi `docker images | grep '<none>' | grep
    'minutes\|hour\|hours\|day\|days' | awk '{print $3}'`
```

This script uses the `rm` and `rmi` commands of Docker to remove containers and untagged images that have been created for more than one minute. Untagged images are those created solely for the purpose of spawning a new docker container and they become useless after a container has finished its job. Therefore they need to be removed alongside the finished containers.

4.2.3. Dealing With Stack Overflow

It is common for students to run into stack overflow issue, for instance, making an infinite number of recursion such as one in the example given below. To deal with this issue, I used the `-m` option with value of 32m when invoking the `run` command to tell Docker to give the container a maximum of 32 megabytes of RAM.

```
#include<stdio.h>
int fib(int n) {
   return fib(n-1) + fib(n-2);
}
int main () {
  int n = 42;
  printf("%d", fib(n));
  return 0;
}
```

In this program the student made the mistake of not including the terminating condition which is

```
if (n <= 1)
   return n;
```

in the `fib` function. Therefore creating an infinite call stack. Since the amount of memory is limited to 32 MB, the program terminates when it exceeds the allotted memory.

4.3. Apparmor Approach

4.3.1. Rationale For Using Apparmor

AppArmor is a mandatory access control (MAC) system which is a kernel (LSM) enhancement to confine programs to a limited set of resources. Apparmor's security model is to bind access control attributes to programs rather than to users. AppArmor differs from some other MAC systems on Linux: it is path-based, it allows mixing of enforcement and complain mode profiles, it uses include files to ease development, and it has a far lower barrier to entry than other popular MAC systems.

For this project, Apparmor becomes the second natural choice because it provides a way to harden the host operating system from potentially malicious student programs. Since it does not use virtualization technology like the Docker approach, it runs much faster. This approach is followed from David Malan's hardening of his Fedora-based grading system with SELinux [13].

4.3.2. Security Measures

I used the `timeout` script (written by Pavel Shved) to limit memory usage to 32 MB in order to cope with stack overflow issue. I also configured apparmor to allow enough permissions for Java to run while disabling one or more of read/write/execute accesses to system files not required by codecheck.

The `timeout` script is written by Pavel Shved and is available at https://github.com/pshved/timeout. This script allows the user to specify a timeout or a maximum amount of memory when running a program. It has the following format:

```
timeout [options] command [arguments]
```

where

The basic options are:

- `-t T` - set up CPU+SYS time limit to T seconds
- `-m M` - set up virtual memory limit to M kilobytes

In my project I used

```
timeout -m 32768 /path/to/codecheck
```

And below is a snippet of an apparmor profile for Java that I used. It is based on one found at http://www.insanitybit.com/2012/08/27/apparmor-and-java/. The full profile is listed in Appendix A. I will explain the differences between my profile and the original profile.

```
# Last Modified: Mon March 31 01:46:38 2014
#include <tunables/global>

/usr/lib/jvm/java-7-oracle/jre/bin/java {
#include <abstractions/base>

/data/temp/ext/** rw,
/data/temp/ext/**/** rw,
/usr/lib/gcc/x86_64-linux-gnu/*/** rix,
/usr/bin/gcc* rix,
/usr/bin/python3 rix
}
```

First of all, since codecheck stores the problems in the /data/temp/ext/ folder, it needs to be able to read and write to that folder as well as its subfolders. And that's what the lines given below do.

```
/data/temp/ext/** rw,
/data/temp/ext/**/** rw,
```

For the remaining lines, since I need to use the gcc compiler and the python3 program inside Java, I need to give Java permissions to read and execute those programs.

```
/usr/lib/gcc/x86_64-linux-gnu/*/** rix,
/usr/bin/gcc* rix,
/usr/bin/python3 rix
```

One issue that needs to be addressed is how to avoid students from filling up the disk. One way to solve this problem is run the program as a user with a limited disk quota. Another way to do it is by using ulimit. With ulimit there is no need to set up a new user each time a program runs and removing the user after the program has finished executing. ulimit is also easier to use and in this project it is used in a process as follows.

```
ulimit -d 1000 -f 10 -n 100 -v 100000
```

Where `-d 1000` limits the maximum size of a process's data segment to 1000 kbytes, `-f 10` limits the maximum size of files written by the shell and its children to 10 blocks, `-n 100` limits the maximum number of open file descriptors to 100, and finally `-v 100000` limits the maximum amount of virtual memory to 100000 kbytes.

4.3.3. Dealing With Core Dumps

The segmentation fault error can be displayed in the program output for the students to see and fix their code because of an approach that I used which I will explain below. The output not only shows that there is a non-page error in the student's code but also points out what line causes the error.

Supposed the student program contains the following lines:

```
#include <stdio.h>
int main()
{
   char* p = 0;
   printf("%d", *p);
}
```

To deal with the segmentation fault error, we first start by compiling the code with option `-g` and then using ulimit to set the size of core files to unlimited. Then we execute the program and using gdb to tell us whether the segmentation fault occurred. The commands for the approach is listed below:

```
$ gcc -g test.c -o test
$ ulimit -c unlimited
$ rm -f core
$ ./test
Segmentation fault (core dumped)
$ ls core
core
$ echo q | gdb --core=core test| tail -4
Program terminated with signal SIGSEGV, Segmentation fault.
#0  0x0000000000400541 in main () at test.c:7
7           printf("%d", *p);
```

Figure 7 shows an output when such error occurs

Testing chars

Actual output	Expected output
Program terminated with signal 11, Segmentation fault. #0 0x0000000000400508 in main () at /tmp/codecheck.8BN87cfhXJ/chars.c:6 6 printf("%d", *p); (gdb)	AB

fail

Actual output	Expected output
Program terminated with signal 11, Segmentation fault. #0 0x0000000000400508 in main () at /tmp/codecheck.8BN87cfhXJ/chars.c:6 6 printf("%d", *p); (gdb)	CD

fail

Actual output	Expected output
Program terminated with signal 11, Segmentation fault. #0 0x0000000000400508 in main () at /tmp/codecheck.8BN87cfhXJ/chars.c:6 6 printf("%d", *p); (gdb)	EF

fail

Student files

chars.c:

```
#include <stdio.h>

int main(void)
{
     char* p = 0;
   printf("%d", *p);

    return 0;
}
```

Score

0/3

Download

Figure 7: Segmentation fault error displayed in the program output

5. Usage Data

This section will present and analyze the data gathered from the user as well as the stress tests and security tests performed on the system.

5.1. Data From The User

The system was placed under real world usage for a semester long to support two sections of CS49C, which is an introduction to C programming course taught at San Jose State University and it consists of about 60 students. A virtual machine of 4 CPUs and 4 GB of RAM was created to run the docker version of codecheck and another virtual machine with 2 CPUs and 2 GB of RAM was created alongside to run the Apparmor version of codecheck. No security concerns were discovered during the length of the semester.

I received positive feedback from the instructor of these two sections because codecheck had allowed him to double the amount of assignments he would have given to his students. It was possible because codecheck made grading more efficient and so it allowed for more assignments to be created without going over the grader's time limit. However, it should be noted that the amount of assignments for CS 49C is still only ⅔ the amount for CS49A/B which is the prerequisites.

Since more assignments can be created, the instructor was able to create a draft version and a final version for every assignment. The draft version is due a few days before the final version and is simpler than the final version. The idea is to get the students to think about the problem in advance. This way the students can and do ask questions for multiple days before the final version is due. This would be impossible to do without automation and instant feedback.

Table 1 lists all the assignments created for CS49C section 2 with due dates. All assignments utilized codecheck except for 8a and 9a. The reason for the last two assignments not using codecheck was because the instructor was not aware that codecheck could handle file input/output.

Assignment	Draft Due	Final Due	Used codecheck	Max Score
Programming 1a	02/12, 10:30 AM	02/17, 10:30 AM	Yes	4
Programming 1b	02/12, 10:30 AM	02/17, 10:30 AM	Yes	4
Programming 2a	02/19, 10:30 AM	02/24, 10:30 AM	Yes	4
Programming 2b	02/19, 10:30 AM	02/24, 10:30 AM	Yes	4
Programming 3a	02/26, 10:30 AM	03/03, 10:30 AM	Yes	4
Programming 3b	02/26, 10:30 AM	03/03, 10:30 AM	Yes	4
Programming 4a	03/05, 10:30 AM	03/10, 10:30 AM	Yes	4
Programming 4b	03/05, 10:30 AM	03/10, 10:30 AM	Yes	4
Programming 5a	03/12, 10:30 AM	03/17, 10:30 AM	Yes	4
Programming 5b	03/12, 10:30 AM	03/17, 10:30 AM	Yes	4
Programming 6a	04/16, 10:30 AM	04/23, 10:30 AM	Yes	4
Programming 6b	04/16, 10:30 AM	04/23, 10:30 AM	Yes	4
Programming 7a	04/23, 10:30 AM	04/28, 10:30 AM	Yes	4
Programming 8a	04/30, 10:30 AM	05/05, 10:30 AM	No	4
Programming 9a	05/07, 10:30 AM	05/12, 10:30 AM	No	4

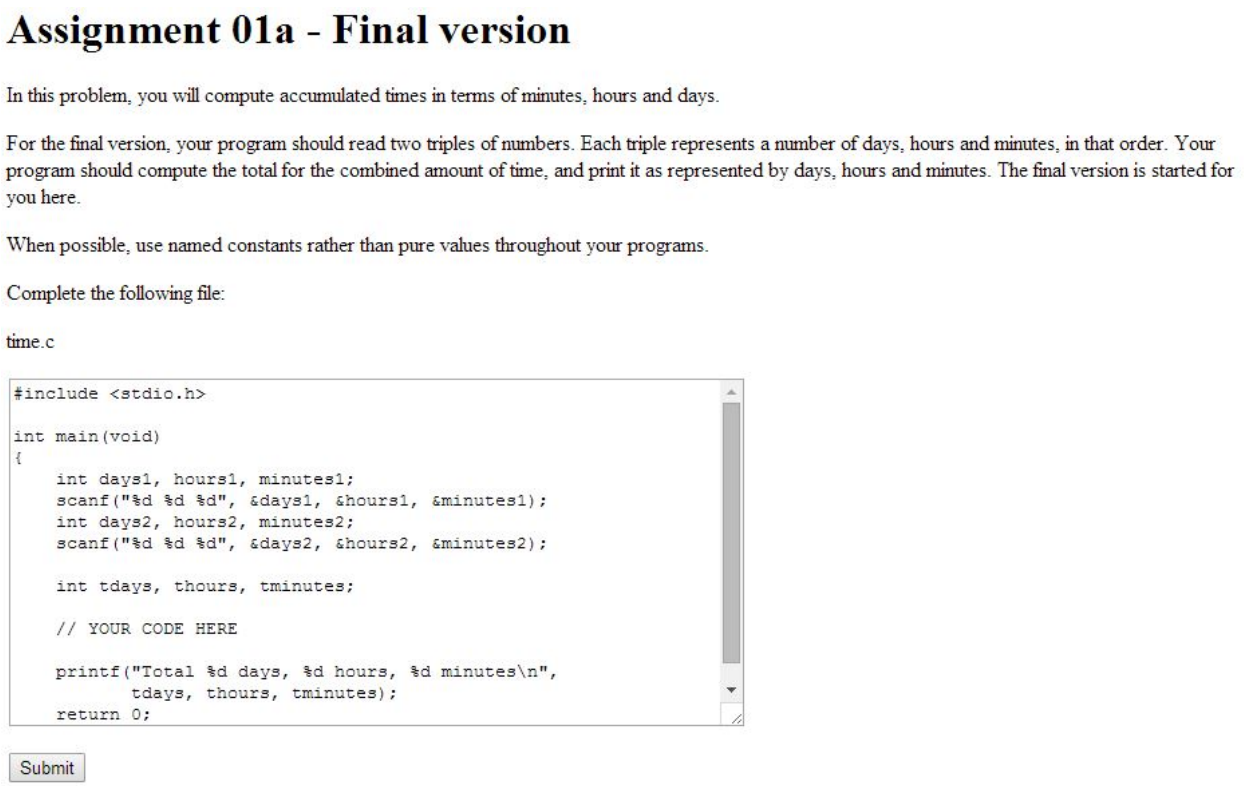
Table 1: List of assignments for CS 49A section 2 with due date

Table 2 lists the score distribution of the assignments listed above except for assignment 9a which has not been graded as of this writing.

	4/4	3/4	2/4	1/4	0/4	Average
Program 1a	56	5	0	0	1	3.85
Program 1b	48	7	3	0	4	3.53
Program 2a	52	8	1	0	1	3.66
Program 2b	51	7	1	0	3	3.66
Program 3a	48	8	0	0	6	3.48
Program 3b	40	11	0	0	11	3.11
Program 4a	43	15	0	0	4	3.5
Program 4b	44	14	3	0	1	3.61
Program 5a	53	6	1	0	2	3.74
Program 5b	40	11	1	1	9	3.16
Program 6a	41	13	3	1	4	3.38
Program 6b	44	15	0	0	3	3.56
Program 7a	34	15	2	0	11	2.98
Program 8a	30	11	4	3	14	2.64
Sum	624	146	19	5	74	

Table 2: Summary of scores of all assignments for both sections.

The next few figures will be used to show some of these assignments.



Assignment 01a - Final version

In this problem, you will compute accumulated times in terms of minutes, hours and days.

For the final version, your program should read two triples of numbers. Each triple represents a number of days, hours and minutes, in that order. Your program should compute the total for the combined amount of time, and print it as represented by days, hours and minutes. The final version is started for you here.

When possible, use named constants rather than pure values throughout your programs.

Complete the following file:

time.c

```
#include <stdio.h>

int main(void)
{
    int days1, hours1, minutes1;
    scanf("%d %d %d", &days1, &hours1, &minutes1);
    int days2, hours2, minutes2;
    scanf("%d %d %d", &days2, &hours2, &minutes2);

    int tdays, thours, tminutes;

    // YOUR CODE HERE

    printf("Total %d days, %d hours, %d minutes\n",
           tdays, thours, tminutes);
    return 0;
```

Submit

Figure 8: CS 49C assignment 1a final of little difficulty

Grade distribution:

4/4: 56 students

3/4: 5 students

2/4: 0 students

1/4: 0 students

0/4: 1 students

Assignment 04a - Final version

In this problem, we will use 1- and 2-dimensional arrays to create simple maps. Please read the problem carefully and make sure you understand it completely before writing your program.

Homer loves donuts. Every morning when he gets to work, his first stop is the coffee break room. Unfortunately sometimes the room is lights out, like whenever there's a power failure (which happens surprisingly often, considering they work at a nuclear power plant). So Homer asks Lenny to make him a glow-in-the-dark map of the break room, showing exactly where the donuts are located. That way, Homer can find the shortest path to the donuts, even in the dark.

Lenny draws the map using a square grid layout, represented by a 2-dimensional array. The break room is 10 by 10 square feet, and each square foot is represented by one element in the array. An integer value in each element indicates how many steps from that location to the box of donuts. The location of the donuts is represented by the number 0 (because it looks like a donut). The eight squares immediately surrounding the donuts are represented by values of 1, and the squares surrounding those are represented by values of 2, and so on. You can help Lenny by using your C skills to fill the whole array.

For the final version, Lenny needs a 2-dimensional array to make the map. The room grid is represented by an array with 10 rows by 10 columns, indexed from 0 to 9 in each direction. Your program will read two integers, which tells you the row and column of the donuts' location. Fill in the whole array, and print it out row by row with no spaces between the digits. Here's an example when the donuts are located at row 3, column 7:

```
7654333333
7654322222
7654321112
7654321012
7654321112
7654322222
7654333333
7654444444
7655555555
7666666666
```

You must keep the values in a 2-dimensional array. The final version is started for you here.

Complete the following file:

donuts_final.c

```
#include <stdio.h>

////REQUIRED \[.*\].*\=

int room[10][10];  // Map of the room

int main(void)
{
    int x, y;
    scanf("%d %d", &x, &y); // Read the donuts' location

    // YOUR CODE HERE

    return 0;
}
```

Figure 9: CS 49C assignment 4a final of medium difficulty

Grade distribution:

4/4: 45 students

3/4: 13 students

2/4: 0 students

1/4: 0 students

0/4: 4 students

Assignment 07a - Final version

In this problem, we will use character and string functions to extract a list of names. Please read the problem carefully and make sure you understand it completely before writing your program.

Gru, the reformed supervillian, received a lovely bouquet of forget-me-not-nots (yes that's a double negative) from an anonymous rival supervillian. Unfortunately after one sniff of the beautiful flowers, Gru could no longer remember the names of most of his Minion allies: in particular the ones whose names alphabetically followed his own name. Luckily, Dr. Nefario had prepared for just such a tragedy, because he had previously compiled a database of Minion names. Just to need to extract the names, print out the list and let Gru memorize it.

For the final version, you must scan a long text string to find Minion names. They are easy to find, because they are the only characters in upper-case letters. You will create a new single string which has each Minion name on a separate line, and this time, it must be capitalized as a proper name. (First letter in upper-case, the rest in lower-case). Oh, one other catch, you should only fetch the names that alphabetically follow "Gru". Those are the ones that Gru forgot. (So, Zeke would be included, but Aaron would not.) The new string will be printed at the end of the program.

You will probably find it convenient to use functions from the C standard library. Don't forget to include the header file(s) too. The final version is started for you here.

Complete the following file:

minions_final.c

```
#include <stdio.h>

int main(void)
{
    int i, j;
    char text[200];
    char minions[200] = "";
    char minion[10]; // useful for intermediate strings

    // Read the raw text
    scanf("%s", text);

    // YOUR CODE HERE


    // Print the string of Minion names
    printf("%s", minions);
    return 0;
}
```

Submit

Figure 10: CS 49C assignment 7a final of challenging difficulty

Grade distribution:

4/4: 34 students

3/4: 15 students

2/4: 2 students

1/4: 0 students

0/4: 11 students

Student Survey

1/ What do you think is the biggest advantage of code-checker?

- Instant feedback 90% IIIIIIIIIIIIIIIIIIIIIIIII
- Makes me start the work earlier 7% II
- Fairer grading 0%
- Easier to understand requirements 3% I
- Other (Fill in box 7) 0%

2/ What do you think is the biggest disadvantage of code-checker?

- Server unreliable 66% IIIIIIIIIIIIIIIIIII
- Classmates cheat 7% II
- Too much homework 7% II
- Other (Fill in box 7) 20% IIIIII

3/ Compared to the other programming classes, how much time did you spend programming?

- Much more 7% II
- More 7% II
- About the same 43% IIIIIIIIIIII
- Less 33% IIIIIIIII
- Much less 10% III

4/ How many hours did you spend on average on each

- Draft: _____ 1.5 Average
- Final: _____ 5.5 Average

5/ How do you rate the codecheck system for homework submission?

- Very useful 40% IIIIIIIIIII
- Useful 57% IIIIIIIIIIIIIIII
- Somewhat useful 3% I
- Not useful 0%

6/ If you were to take another class, would you prefer that an automated grading system like codecheck be used?

- Definitely 50% IIIIIIIIIIIIII
- Probably 40% IIIIIIIIIII
- Maybe 10% III
- Definitely not 0%

7/ Other (See appendix B)

From the survey responses, it appears that the students think the biggest advantage of codecheck is its ability to provide instant feedback. And the biggest disadvantage is the unreliability of the server which was due to the fact that the system was still in development while it was being used. About ⅓ the responses said that they spend less time programming with codecheck. 97% of the responses say that codecheck is either useful or very useful for homework submission. And 90% of the responses would prefer an automated system like codecheck to be used for their future programming courses.

5.2. Performance Tests

In this section I will present the series of stress tests I have done to find out the absolute limits of Docker and apparmor. In these stress tests I would like to find out how each system handles when all the students submit their homework assignments at the same minute right before the deadline. For instance, all the students hit submit at 11:59 PM when the deadline for their assignment is at 12:00 AM. Another performance metrics I would like to get out of these tests is how the systems would handle during the final hour of the submission deadline in which all the students are practically using the systems as their IDE when they continuously make small changes in their code and hit submit.

5.2.1. The Setup

Two virtual machines of identical configuration, namely cs43 and cs44, were created for these stress tests. Each machine runs Ubuntu Linux 12.04 64 bit and has 2 CPUs, 2 GB RAM, and about 8 GB of hard drive. One machine is installed with the Docker approach and the other one is installed with the apparmor approach. The tool that I used to perform these stress tests is JMeter, a popular open source stress test tool written in Java.

5.2.2. The Stress Test Without Infinite Loop

In this test it is assumed that all the students submit their code without infinite loops in them. This test will cover two scenarios as explained above. But first of all, let’s take a look of the programming assignment that was used in these stress tests

Assignment 02a - Final version

In this problem, we will use selection statements to make decisions in a game. Please read the problem carefully and make sure you understand it completely before writing your program.

Each week, two sisters named Minnie and Maxine visit their Grandpa, who has a jar of quarters (25-cent coins) to give to the girls. First, Grandpa tells them how many coins are in the jar. Then each girl can whisper in Grandpa's ear and tell him how many quarters she wants (up to all the coins). After Grandpa hears both requests, he decides how to split the coins between the two girls.

For the final version, Grandpa splits the coins in a particular way:

If the two girls' requests combined are no more than the number of coins in the jar, then they each get as many as they asked for. But in this case, any extra coins left over go to the girl who asked for fewer coins.

On the other hand, if the combined amount of their requests exceeds the number of coins in the jar, then the girl who asked for fewer coins gets the amount that her greedy sister asked for, and the girl who asked for more coins gets as many coins are left over.

In either case, if both girls asked for the same amount of coins, Grandpa splits the coins exactly evenly, while keeping one coin for himself if there was an odd number of coins.

The final version is started for you here. It reads three integers in the following order: the number of coins in the jar, Minnie's request, and Maxine's request. Your program should print out the amount of money (as dollars and cents) that each girl gets.

Complete the following file:

quarters.c

```
#include <stdio.h>

int main(void)
{
    int jar;              // the total number of coins in the jar
    int minnie, maxine;   // the number of quarters they each
request
    scanf("%d %d %d", &jar, &minnie, &maxine);

    // YOUR CODE HERE

    printf("Minnie gets $%4.2f \n", /* EXPRESSION */);
    printf("Maxine gets $%4.2f \n", /* EXPRESSION */);
```

Submit

Figure 11: the problem used for the stress tests is a homework assignment used by CS49C

In this test, JMeter is configured to send as many requests as possible to both servers for a duration of 5 minutes. After that the performance metrics are displayed in a summary report as shown in the next two figures.

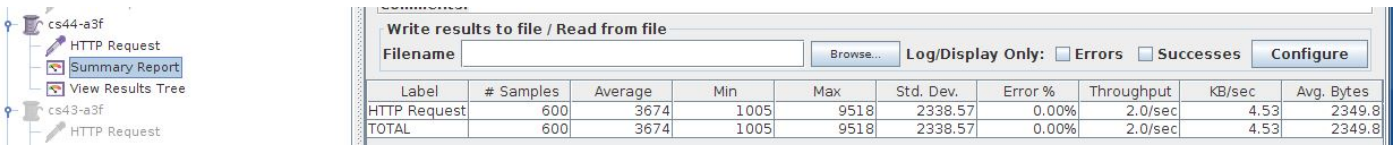


Figure 12: apparmor result for code without infinite loops

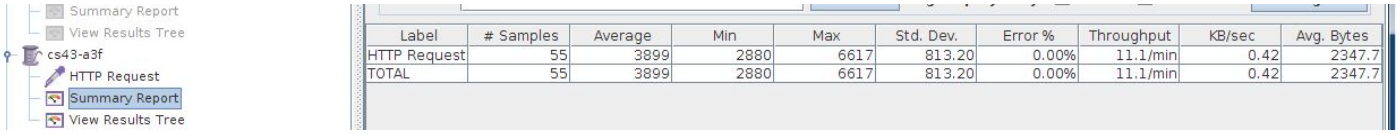


Figure 13: Docker result for code without infinite loops

The results from the figures can be summarized in the table below

	Throughput	**Error %**	**Ave. Time (ms)**	**Min (ms)**	**Max (ms)**	**Std. Dev. (ms)**
apparmor	72 per min	0	3674	1005	9518	2338
Docker	11.1 per min	0	3899	2880	6617	813

Table 3: Summary of stress tests without infinite loops

From these measurements it appears that with apparmor, codecheck is able to produce 6.5 times the throughput as compared to Docker. Even though the average response times seem to be about the same with each request takes about 3.5 to 4 seconds to complete.

5.2.3. The Stress Test With Infinite Loop

In this test it is assumed that all the students submit their code with infinite loops in them. This test will cover two scenarios as explained above.

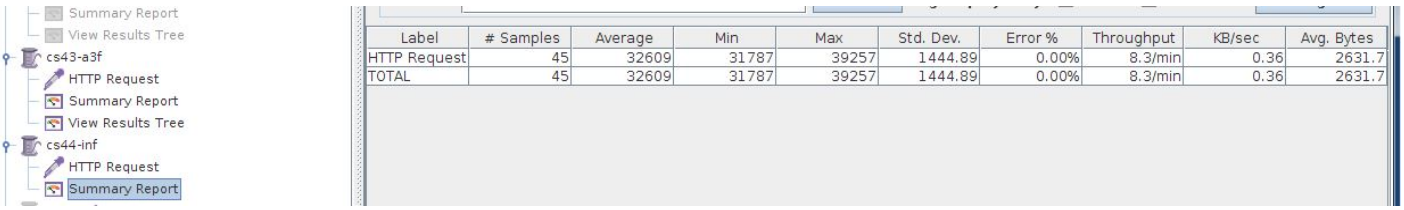
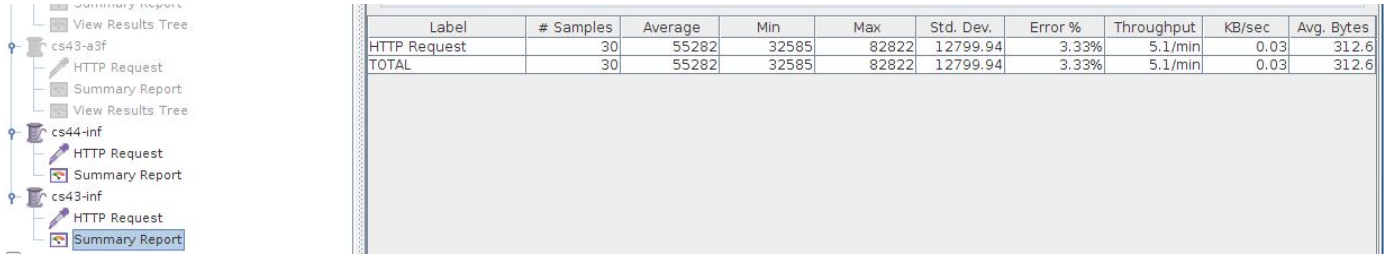


Figure 14: apparmor result for code with infinite loops



Label	# Samples	Average	Min	Max	Std. Dev.	Error %	Throughput	KB/sec	Avg. Bytes
HTTP Request	30	55282	32585	82822	12799.94	3.33%	5.1/min	0.03	312.6
TOTAL	30	55282	32585	82822	12799.94	3.33%	5.1/min	0.03	312.6

Figure 15: Docker result for code with infinite loops

	Throughput	**Error %**	**Ave. Time (ms)**	**Min (ms)**	**Max (ms)**	**Std. Dev. (ms)**
apparmor	8.3 per min	0	32609	31787	39257	1444
Docker	5.1 per min	3.33	55283	32585	82822	12799

Table 4: Summary of stress tests with infinite loops

From these measurements it appears that with apparmor, codecheck is able to produce 1.5 times the throughput as compared to Docker. For Docker, there is a 3.33% error rate which means some of the request came back with a HTTP 404 error. Timewise, Docker has a longer response time than apparmor but the standard deviation is also bigger which means some of the requests are taking longer than others to complete and thus skewing the average result. This is due to the fact that the cleanup script explained in chapter 4 runs every minute.

5.2.4. The Stress Test With Time-Consuming Code

This test was performed on the system with the Fibonacci code below which has an exponential execution time. The system will be subjected to a load test with the load being to calculate `fib(42)`.

```
#include<stdio.h>
int fib(int n) {
   if (n <= 1)
      return n;
   return fib(n-1) + fib(n-2);
}

int main () {
  int n = 42;
  printf("%d", fib(n));
```

```
  return 0;
}
```

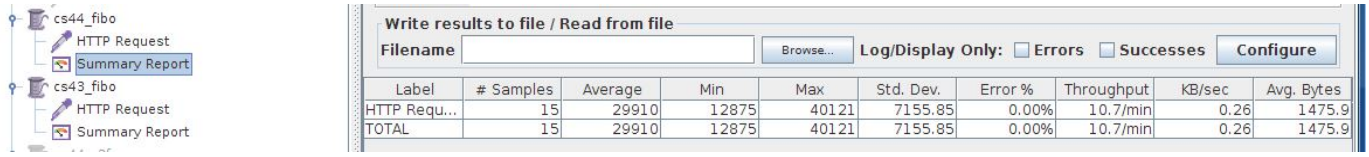


Figure 16: apparmor result for fib(42) code

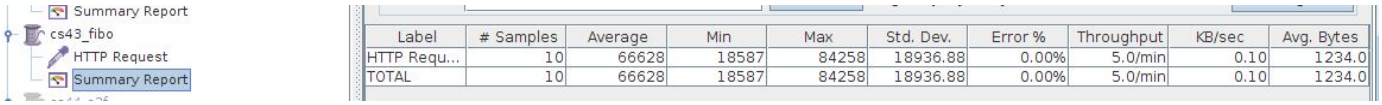


Figure 17: Docker result for fib(42) code

	Throughput	**Error %**	**Ave. Time (ms)**	**Min (ms)**	**Max (ms)**	**Std. Dev. (ms)**
apparmor	10.7 per min	0	29910	12875	40121	7155
Docker	5.0 per min	0	66628	10587	84258	18936

Table 5: Summary of stress tests with fib(42) code

From these measurements it appears that with apparmor, codecheck is able to produce 2 times the throughput as compared to Docker. Even though the average response times for Docker is longer than apparmor's. The reason for this is similar to the one explained in the previous test.

Even though the stress tests show that Docker cannot handle a load of 60 submissions per minute to serve the size of two sections combined. In reality, the two sections have different due dates for their assignments. Therefore the theoretical maximum is reduced by half to become 30 per minute. Moreover, many students submitted their assignments early. And besides, the server used for the two sections had several times more resources than the test servers.

5.3. Security Tests

These tests were conducted to expose the strengths as well as limitations of each approach.

5.3.1. Read/Write To a File

This test shows whether each system can prevent a student's program from accessing other students' submission files. As the test shows, the apparmor approach cannot prevent it but the Docker approach can. The reason for it is because with Docker each student program runs in a separate container and thus one student cannot read another student's file. With apparmor since the program has read and write access to the /tmp/codecheck folder, it can access all files in that directory.The program is as follows:

```
#include <stdio.h>
#include <dirent.h>

int main(void) {
  DIR *d;
  struct dirent *dir;
  d = opendir("/tmp/codecheck");
  if (d) {
    while ((dir = readdir(d)) != NULL) {
      printf("%s\n", dir->d_name);
    }
    closedir(d);
  }
  return(0);
}
```

The next two figures show the results after running the code

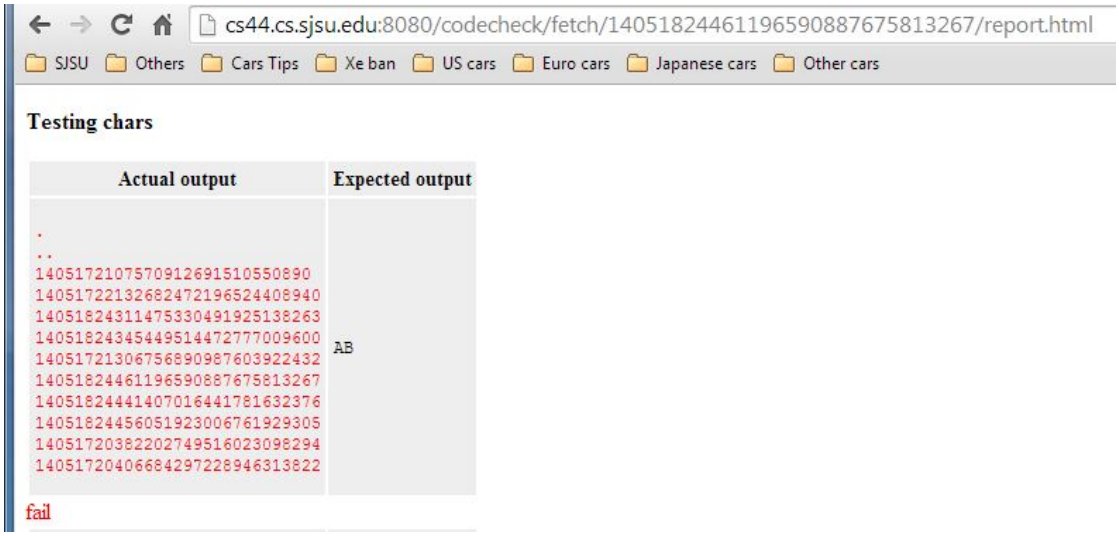


Figure 18: the apparmor approach cannot prevent students from accessing neighboring folders

Figure 19: the Docker approach can prevent students from accessing neighboring folders

5.3.2. Delete Files

This test shows whether each system can prevent a student's program from deleting files. As the test shows, the apparmor approach can prevent it but the Docker approach cannot. However, in Docker, the student program runs inside a container which is completely isolated from the host. Therefore whatever happens to it does not affect the host at all.The program is as follows. This program tries to delete /bin/sh, /bin/bash, and /bin/cat.

```
#include <stdio.h>
#include <dirent.h>
#include <sys/types.h>

int main(void) {
  remove("/bin/sh");
  remove("/bin/bash");
  remove("/bin/cat");
  DIR *d;
```

```
  struct dirent *dir;
  d = opendir("/bin");
  if (d) {
    while ((dir = readdir(d)) != NULL) {
      printf("%s\n", dir->d_name);
    }
    closedir(d);
  }
  return(0);
}
```

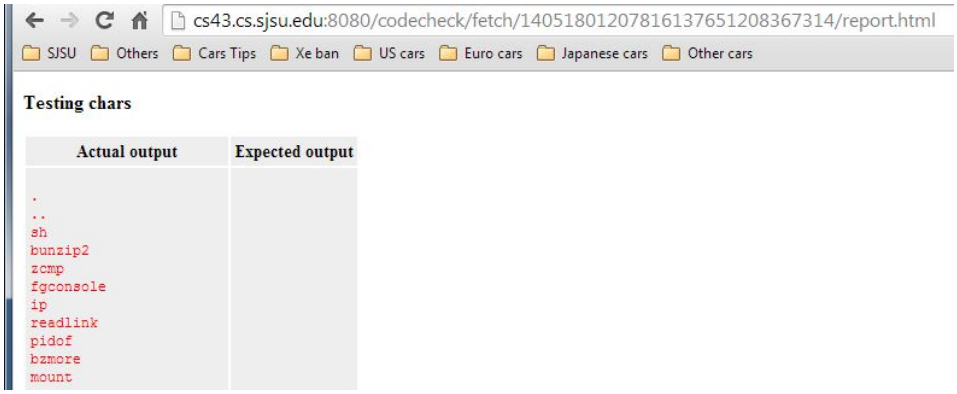


Figure 20: In Docker the malicious program can delete /bin/bash and /bin/cat but not /bin/sh.

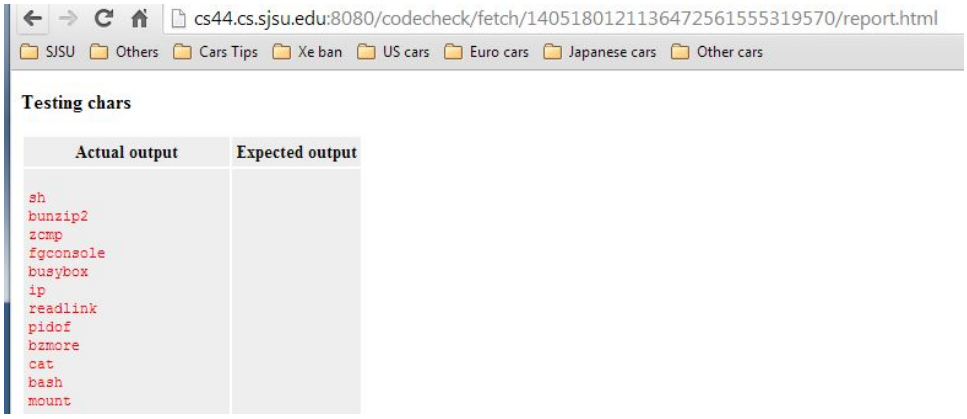


Figure 21: In apparmor the malicious program cannot delete any of the files

The kernel log shows that the calls to remove the system files were suppressed.

May 21 02:32:59 knub64-VirtualBox kernel: [361540.689707] audit_printk_skb: 6921 callbacks suppressed

6. Conclusion

In this project I adapted codecheck for C and Python programs. I also implemented isolation with Docker and hardening operating system with apparmor. To compare the performance levels of Docker and apparmor, I performed a series of stress tests and analyzed the results. In order to gather real data for my implementation, I worked with the CS 49C instructor to put the system under real use to support 60 students in two sections. And I also analyzed student performance and satisfaction by conducting a survey.

In conclusion, this paper presented two security approaches used for the purpose of secure automatic evaluation of students' programs. The first approach involves the use of Docker, a lightweight Linux container solution, to isolate each student program from the host system and from one another. And the second approach makes use of Apparmor to harden the host operating system from undesirable effects of running potentially malicious code. Both approaches appear have been successfully tested in a semester long of real world usage. Even though Docker provides a greater measure of isolation, in actual usage, the security provided by apparmor appears to have been sufficient Moreover, performance-wise, the Apparmor approach is preferable to the Docker approach as it does not incur the overhead of using virtualization. The test data shows that apparmor outperforms Docker by achieving two to six times the throughput.

Appendix A

This appendix lists the full apparmor profile that I used in this project.

```
# Last Modified: Mon March 31 01:46:38 2014
#include <tunables/global>

/usr/lib/jvm/java-7-oracle/jre/bin/java {
#include <abstractions/base>

network inet stream,
network inet6 stream,
/tmp/*/** rwix,
/data/temp/ext/** rw,
/data/temp/ext/**/** rw,
/usr/*/** rwix,
/usr/lib/gcc/x86_64-linux-gnu/*/** rwix,
/usr/bin/gcc* rix,
/usr/bin/python3
/anon_hugepage//deleted r,
/dev/snd/* rw,
/etc/.java/ rw,
/etc/.java/deployment/ rw,
/etc/fonts/** r,
/etc/host.conf r,
/etc/hosts r,
/etc/java-*/ r,
/etc/java-*/** r,
/etc/lsb-release r,
/etc/nsswitch.conf r,
/etc/passwd mr,
/etc/pulse/client.conf r,
/etc/ssl/certs/java/* r,
/etc/timezone r,
owner /home/** w,
/home/** mrk,
/home/*/.cache/dconf/user rw,
/home/*/.java/** rwk,
/home/*/.pulse-cookie rwk,
/proc/*/cmdline r,
/proc/*/coredump_filter rw,
/proc/[0-9]*/fd/ r,
/proc/asound/version r,
/proc/filesystems r,
/run/resolvconf/resolv.conf r,
/run/shm/ r,
```

```
/run/shm/* rw,
/sys/devices/system/cpu/ r,
/sys/devices/system/cpu/** r,
/tmp/ r,
owner /tmp/** m,
/tmp/** rw,
/usr/bin/env ix,
/usr/lib/j2*-ibm/jre/bin/java ix,
/usr/lib/jvm/java-&-oracle/jre/bin/java mr,
/usr/lib/jvm/java-*-sun-1.*/jre/bin/java{,_vm} ix,
/usr/lib/jvm/java-*-sun-1.*/jre/lib/*/client/classes.jsa mr,
/usr/lib/x86_64-linux-gnu/pango/*/modules/*.so mr,
/usr/local/share/fonts/ r,
/usr/share/alsa/** r,
/usr/share/fonts/ r,
/usr/share/fonts/** r,
/usr/share/glib-*/schemas/* r,
/usr/share/icons/** r,
/usr/share/themes/** r,
/var/cache/fontconfig/* r,
/var/lib/dbus/machine-id r,
/var/lib/defoma/x-ttcidfont-conf.d/dirs/TrueType/ r,
/var/lib/defoma/x-ttcidfont-conf.d/dirs/TrueType/* r,
@{PROC}/[0-9]*/net/if_inet6 r,
@{PROC}/[0-9]*/net/ipv6_route r,
@{PROC}/loadavg r,
}
```

Appendix B

This Appendix shows the students' comments about the codecheck system.

1/ I'd like it if code checker shows us what inputs they use for our programs. For instance what numbers they'll use or whatever, it'd help in the coding process.....
i.e.
"CC input DOAWDOO#$24#231"
"Your program output: MY PROGRAM SUX LOL"
"Expected output: Nice program!"

2/ The only disadvantage I see is that people will use it to debug their code instead of learning to do it themselves and rush to submit to code check instead of conducting their own test's

3/ its chill brah

4/ I really like the code-checker and draft/final format, but might need to change it up some time in the course to teach about unit testing and more in-depth debugging

5/ Code checking helped us to understand what the code needed to be implemented.

6/ code checker is awesome for beginner programming assignments like in the first few months of the class, but as the class progressed into harder problems i felt code checker was harder or more difficult to use due to the small nature of the site and limited space (but overall i very much rather have used codechecker than not throughout this class).

7/ I think code check is a great system. Maybe a few things need to change but for the most part I like code check.

8/ Something like an automated grading system needs a better, more stable server that doesn't crash so much. If it is achievable, then it is recommended.

9/ have a good summer!

10/ I think you should put a test code for the students in the codechecker so students will actually know what the output should be. It makes it easier to understand the problem and more convenient then just testing it out on codechecker first with printf(...).

11/ Code check disadvantages: for the programmer invisible characters are not shown such aa new line and null, making it sometimea hard to find the error... Otherwise my only other complaint would be to make it visually appealing.

12/ The only disadvantage for code-checker is that students rely on it too much. It is convenient, but as a programmer you should already be looking for the test cases and expect what the output should be.

13/ The occasional server unreliability was a major bummer.

14/ Code-Check server was unreliable many times during the semester.

15/ I have prior experience programming in C, so the assignments took less time to finish compared to other courses I have taken. However, taking into consideration other students may have no prior experience programming or familiarity in another language other than C the difficulty of assignments seems just right for this course.

16/ Code check is cool

References

1. The apparmor project (http://wiki.apparmor.net/index.php/Main_Page)
2. Salman A. Baset. 2012. Open source cloud technologies. In Proceedings of the Third ACM Symposium on Cloud Computing (SoCC '12). ACM, New York, NY, USA, , Article 28 , 2 pages. DOI=10.1145/2391229.2391257 http://doi.acm.org/10.1145/2391229.2391257
3. CodingBat project (http://codingbat.com/)
4. CodeLab project (http://turingscraft.com/)
5. Jackson, David, A semi-automated approach to online assessment, Proceedings of the 5th Annual SIGCSE/SIGCUE ITICSE conference, 164-167, 2000.
6. Xiang Fu, Boris Peltsverger, Kai Qian, Lixin Tao, and Jigang Liu. 2008. APOGEE: automated project grading and instant feedback system for web based computing. SIGCSE Bull. 40, 1 (March 2008), 77-81. DOI=10.1145/1352322.1352163 http://doi.acm.org/10.1145/1352322.1352163
7. Kay, David G, Scott, Terry, Isaacson, Peter, Reek, Kenneth, Automated grading assistance for student programs, Proceedings of the 25th SIGCSE technical symposium, 381-382, 1994.
8. J. Archer Harris, Elizabeth S. Adams, and Nancy L. Harris. 2004. Making program grading easier: but not totally automatic. J. Comput. Sci. Coll. 20, 1 (October 2004), 248-261.
9. Docker.io project (https://www.docker.io/)
10. Dawson-Howe, Kenneth, Automatic submission and administration of programming assignments, ACM SIGCSE Bulletin, Volume 28, (Issue 2), 40-42, 1996.
11. Labrat project (http://www.horstmann.com/bigj3/labrat-faq.html)
12. Macpherson, P.A., A technique for student program submission on UNIX systems, ACM SIGCSE Bulletin, Volume 29, (Issue 4), 54-56, 1997.
13. David J. Malan. 2013. CS50 sandbox: secure execution of untrusted code. In Proceeding of the 44th ACM technical symposium on Computer science education (SIGCSE '13). ACM, New York, NY, USA, 141-146. DOI=10.1145/2445196.2445242 http://doi.acm.org/10.1145/2445196.2445242
14. Robinson, Sally S., Soffa, M.L., An instructional aid for student programs, Proceedings

of the 11th SIGCSE technical symposium, 118-129, 1980.

15. Pavel, Shved, Limiting time and memory consumption of a program in Linux, https://github.com/pshved/timeout
16. Tremblay, Guy, Labonté, Éric, Semi-automatic marking of Java programs using JUnit, International Conference on Education and Information Systems: Technologies and Applications (EISTA '03), 42—47, 2003.
17. WebCAT project (http://web-cat.org/)