```
#####################################################################
# CS:APP Bomb Lab
# Directions to Instructors
#
# Copyright (c) 2003-2016, R. Bryant and D. O'Hallaron
#
#####################################################################
```

This directory contains the files that you will use to build and run the CS:APP Bomb Lab. The Bomb Lab teaches students principles of machine-level programs, as well as general debugger and reverse engineering skills.

****
1. Overview
****

1.1. Binary Bombs

A "binary bomb" is a Linux executable C program that consists of six "phases." Each phase expects the student to enter a particular string on stdin. If the student enters the expected string, then that phase is "defused." Otherwise the bomb "explodes" by printing "BOOM!!!". The goal for the students is to defuse as many phases as possible.

1.2. Solving Binary Bombs

In order to defuse the bomb, students must use a debugger, typically gdb or ddd, to disassemble the binary and single-step through the machine code in each phase. The idea is to understand what each assembly statement does, and then use this knowledge to infer the defusing string. Students earn points for defusing phases, and they lose points (configurable by the instructor, but typically 1/2 point) for each explosion. Thus, they quickly learn to set breakpoints before
each phase and the function that explodes the bomb. It's a great lesson and forces them to learn to use a debugger.

1.3. Autograding Service

We have created a stand-alone user-level autograding service that handles all aspects of the Bomb Lab for you: Students download their bombs from a server. As the students work on their bombs, each explosion and defusion is streamed back to the server, where the current results for each bomb are displayed on a Web "scoreboard." There are no explicit handins and the lab is self-grading.

The autograding service consists of four user-level programs that run in the main ./bomblab directory:

- Request Server (bomblab-requestd.pl). Students download their bombs and display the scoreboard by pointing a browser at a simple HTTP server called the "request server." The request server builds the bomb, archives it in a tar file, and then uploads the resulting tar file back to the browser, where it can be saved on disk and untarred. The request server also creates a copy of the bomb and its solution for the instructor.

- Result Server (bomblab-resultd.pl). Each time a student defuses a bomb phase or causes an explosion, the bomb sends a short HTTP message, called an "autoresult string," to an HTTP "result server," which simply appends the autoresult string to a "scoreboard log file."

- Report Daemon (bomblab-reportd.pl). The "report daemon" periodically
scans the scoreboard log file. The report daemon finds the most recent
defusing string submitted by each student for each phase, and validates these strings by applying them to a local copy of the student's bomb. It then updates the HTML scoreboard that summarizes

the current number of explosions and defusions for each bomb, rank ordered by the total number of accrued points.

- Main daemon (bomblab.pl). The "main daemon" starts and nannies the request server, result server, and report deamon, ensuring that exactly one of these processes (and itself) is running at any point in
time. If one of these processes dies for some reason, the main daemon detects this and automatically restarts it. The main daemon is the only program you actually need to run.

****
2. Files
****
The ./bomblab directory contains the following files:

Makefile	- For starting/stopping the lab and cleaning files
bomblab.pl*	- Main daemon that nannies the other servers & daemons
Bomblab.pm	- Bomblab configuration file
bomblab-reportd.pl*	- Report daemon that continuously updates scoreboard
bomblab-requestd.pl*	- Request server that serves bombs to students
bomblab-resultd.pl*	- Result server that gets autoresult strings from bombs
bomblab-scoreboard.html	- Real-time Web scoreboard
bomblab-update.pl*	- Helper to bomblab-reportd.pl that updates scoreboard
bombs/	- Contains the bombs sent to each student
log-status.txt	- Status log with msgs from various servers and daemons
log.txt	- Scoreboard log of autoresults received from bombs
makebomb.pl*	- Helper script that builds a bomb
scores.txt	- Summarizes current scoreboard scores for each student
src/	- The bomb source files
writeup/	- Sample Latex Bomb Lab writeup

****
3. Bomb Terminology
****

LabID: Each instance (offering) of the lab is identified by a unique name, e.g., "f12" or "s13", that the instructor chooses. Explosion and
diffusions from bombs whose LabIDs are different from the current LabID are ignored. The LabID must not have any spaces.

BombID: Each bomb in a given instance of the lab has a unique non-negative integer called the "bombID."

Notifying Bomb: A bomb can be compiled with a NOTIFY option that causes the bomb to send a message each time the student explodes or defuses a phase. Such bombs are called "notifying bombs."

Quiet Bomb: If compiled with the NONOTIFY option, then the bomb doesn't send any messages when it explodes or is defused. Such bombs are called "quiet bombs."

We will also find it helpful to distinguish between custom and generic bombs:

Custom Bomb: A "custom bomb" has a BombID > 0, is associated with a particular student, and can be either notifying or quiet. Custom notifying bombs are constrained to run on a specific set of Linux hosts determined by the instructor. On the other hand, custom quiet bombs can run on any Linux host.

Generic Bomb: A "generic bomb" has a BombID = 0, isn't associated with
any particular student, is quiet, and hence can run on any host.

****
4. Offering the Bomb Lab
****
There are two basic flavors of Bomb Lab: In the "online" version, the
instructor uses the autograding service to handout a custom notifying
bomb to each student on demand, and to automatically track their
progress on the realtime scoreboard. In the "offline" version, the
instructor builds, hands out, and grades the student bombs manually,
without using the autograding service.

While both version give the students a rich experience, we recommend
the online version. It is clearly the most compelling and fun for the
students, and the easiest for the instructor to grade. However, it
requires that you keep the autograding service running non-stop,
because handouts, grading, and reporting occur continuously for the
duration of the lab. We've made it very easy to run the service, but
some instructors may be uncomfortable with this requirement and will
opt instead for the offline version.

Here are the directions for offering both versions of the lab.

4.1. Create a Bomb Lab Directory

Identify the generic Linux machine ($SERVER_NAME) where you will
create the Bomb Lab directory (./bomblab) and, if you are offering
the
online version, run the autograding service. You'll only need to have
a user account on this machine. You don't need root access.

Each offering of the Bomb Lab starts with a clean new ./bomblab
directory on $SERVER_NAME. For example:

```
    linux> tar xvf bomblab.tar
    linux> cd bomblab
    linux> make cleanallfiles
```

4.2 Configure the Bomb Lab

Configure the Bomb Lab by editing the following file:

./Bomblab.pm - This is the main configuration file. You will only
need
to modify or inspect a few variables in Section 1 of this file. Each
variable is preceded by a descriptive comment. If you are offering
the
offline version, you can ignore most of these settings.

If you are offering the online version, you will also need to edit
the
following file:

./src/config.h - This file lists the domain names of the hosts that
notifying bombs are allowed to run on. Make sure you update this
correctly, else you and your students won't be able to run your
bombs.

4.3. Update the Lab Writeup

Once you have updated the configuration files, modify the Latex lab
writeup in ./writeup/bomblab.tex for your environment. Then type the
following in the ./writeup directory:

```
        unix> make clean
        unix> make
```

This will create ps and pdf versions of the writeup

4.4. Running the Online Bomb Lab

```
---

------
4.4.1. Short Version
------
From the ./bomblab directory:

(1) Reset the Bomb Lab from scratch by typing
    linux> make cleanallfiles

(2) Start the autograding service by typing
    linux> make start

(3) Stop the autograding service by typing
    linux> make stop

You can start and stop the autograding service as often as you like
without losing any information. When in doubt "make stop; make start"
will get everything in a stable state.

However, resetting the lab deletes all old bombs, status logs, and
the
scoreboard log. Do this only during debugging, or the very first time
you start the lab for your students.

Students request bombs by pointing their browsers at
    http://$SERVER_NAME:$REQUESTD_PORT/

Students view the scoreboard by pointing their browsers at
    http://$SERVER_NAME:$REQUESTD_PORT/scoreboard

------
4.4.2. Long Version
------

(1) Resetting the Bomb Lab. "make stop" ensures that there are no
servers running. "make cleanallfiles" resets the lab from scratch,
deleting all data specific to a particular instance of the lab, such
as the status log, all bombs created by the request server, and the
scoreboard log. Do this when you're ready for the lab to go "live" to
the students.

Resetting is also useful while you're preparing the lab. Before the
lab goes live, you'll want to request a few bombs for yourself, run
them, defuse a few phases, explode a few phases, and make sure that
the results are displayed properly on the scoreboard.  If there is a
problem (say because you forgot to update the list of machines the
bombs are allowed to run in src/config.h) you can fix the
configuration, reset the lab, and then request and run more test
bombs.

CAUTION: If you reset the lab after it's live, you'll lose all your
records of the students bombs and their solutions. You won't be able
to validate the students handins. And your students will have to get
new bombs and start over.

(2) Starting the Bomb Lab. "make start" runs bomblab.pl, the main
daemon that starts and nannies the other programs in the service,
checking their status every few seconds and restarting them if
necessary:

(3) Stopping the Bomb Lab. "make stop" kills all of the running
servers. You can start and stop the autograding service as often as
you like without losing any information. When in doubt "make stop;
make start" will get everything in a stable state.

Request Server: The request server is a simple special-purpose HTTP
server that (1) builds and delivers custom bombs to student browsers
on demand, and (2) displays the current state of the real-time
scoreboard.

A student requests a bomb from the request daemon in two
steps: First, the student points their favorite browser at
```

```
    http://$SERVER_NAME:$REQUESTD_PORT/
```

For example, http://foo.cs.cmu.edu:15213/. The request server
responds by sending an HTML form back to the browser. Next, the
student fills in this form with their user name and email address,
and
then submits the form. The request server parses the form, builds and
tars up a notifying custom bomb with bombID=n, and delivers the tar
file to the browser. The student then saves the tar file to disk.
When
the student untars this file, it creates a directory (./bomb<n>) with
the following four files:

```
    bomb*        Notifying custom bomb executable
    bomb.c       Source code for the main bomb routine
    ID           Identifies the student associated with this bomb
    README       Lists bomb number, student, and email address
```

The request server also creates a directory (bomblab/bombs/bomb<n>)
that contains the following files:

```
    bomb*        Custom bomb executable
    bomb.c       Source code for main routine
    bomb-quiet*  A quiet version of bomb used for autograding
    ID           Identifies the user name assigned to this bomb
    phases.c     C source code for the bomb phases
    README       Lists bombID, user name, and email address
    solution.txt The solution for this bomb
```

Result Server: Each time a student defuses a phase or explodes their
bomb, the bomb sends an HTTP message (called an autoresult string) to
the result server, which then appends the message to the scoreboard
log. Each message contains a BombID, a phase, and an indication of
the
event that occurred. If the event was a defusion, the message also
contains the "defusing string" that the student typed to defuse the
phase.

Report Daemon: The report daemon periodically scans the scoreboard
log
and updates the Web scoreboard. For each bomb, it tallies the number
of explosions, the last defused phase, validates each last defused
phase using a quiet copy of the bomb, and computes a score for each
student in a tab delimited text file called "scores.txt." The update
frequency is a configuration variable in Bomblab.pm.

Instructors and students view the scoreboard by pointing their
browsers at:

```
    http://$SERVER_NAME:$REQUESTD_PORT/scoreboard
```

4.4.3. Grading the Online Bomb Lab

The online Bomb Lab is self-grading. At any point in time, the
tab-delimited file (./bomblab/scores.txt) contains the most recent
scores for each student. This file is created by the report daemon
each time it generates a new scoreboard.

4.4.4. Additional Notes on the Online Bomb Lab
------
* Since the request server and report daemon both need to execute
bombs, you must include $SERVER_NAME in the list of legal machines in
your bomblab/src/config.h file.

* All of the servers and daemons are stateless, so you can stop
("make
stop") and start ("make start") the lab as many times as you like
without any ill effects. If you accidentally kill one of the daemons,
or you modify a daemon, or the daemon dies for some reason, then use
"make stop" to clean up, and then restart with "make start". If your

Linux box crashes or reboots, simply restart the daemons with "make start".

* Information and error messages from the servers are appended to the "status log" in bomblab/log-status.txt. Servers run quietly, so they can be started from initrc scripts at boot time.

* See src/README for more information about the anatomy of bombs and how they are constructed. You don't need to understand any of this to offer the lab. It's provided only for completeness.

* Before going live with the students, we like to check everything out
by running some tests. We do this by typing

```
    linux> make cleanallfiles
    linux> make start
```

Then we request a bomb for ourselves by pointing a Web browser at

```
    http://$SERVER_NAME:$REQUESTD_PORT
```

After saving our bomb to disk, we untar it, copy it to a host in the approved list in src/config.h, and then explode and defuse it a couple
of times to make sure that the explosions and diffusion are properly recorded on the scoreboard, which we check at

```
    http://$SERVER_NAME:$REQUESTD_PORT/scoreboard
```

Once we're satisfied that everything is OK, we stop the lab

```
    linux> make stop
```

and then go live:

```
    linux> make cleanallfiles
    linux> make start
```

Once we go live, we type "make stop" and "make start" as often as we need to, but we are careful never to type "make cleanallfiles" again.

4.5. Running the Offline Bomb Lab

In this version of the lab, you build your own quiet bombs manually and then hand them out to the students. The students work on defusing
their bombs offline (i.e., independently of any autograding service) and then handin their solution files to you, each of which you grade manually.

You can use the makebomb.pl script to build your own bombs manually. The makebomb.pl script also generates the bomb's solution. Type "./makebomb.pl -h" to see its arguments.

Option 1: The simplest approach for offering the offline Bomb Lab is to build a single generic bomb that every student attempts to defuse:

```
    linux> ./makebomb.pl -s ./src -b ./bombs
```

This will create a generic bomb and some other files in ./bombs/bomb0:

```
    bomb*        Generic bomb executable (handout to students)
    bomb.c       Source code for main routine (handout to students)
    bomb-quiet*  Ignore this
    ID           Ignore this
    phases.c     C source code for the bomb phases
    README       Ignore this
    solution.txt The solution for this bomb
```

You will handout only two of these files to the students: ./bomb and ./bomb.c

The students will handin their solution files, which you can validate
by feeding to the bomb:

```
    linux> cd bombs/bomb0
    linux> ./bomb < student_solution.txt
```

This option is easy for the instructor, but we don't recommend it
because it is too easy for the students to cheat.

Option 2. The other option for offering an offline lab is to use the
makebomb.pl script to build a unique quiet custom bomb for each
student:

```
    linux> ./makebomb.pl -i <n> -s ./src -b ./bombs -l bomblab -u
<email> -v <uid>
```

This will create a quiet custom bomb in ./bombs/bomb<n> for the
student whose email address is <email> and whose user name is <uid>:

```
    bomb*        Custom bomb executable (handout to student)
    bomb.c       Source code for main routine (handout to student)
    bomb-quiet*  Ignore this
    ID           Identifies the student associated with this bomb
    phases.c     C source code for the bomb phases
    README       Lists bomb number, student, and email address
    solution.txt The solution for this bomb
```

You will handout four of these files to the student: bomb, bomb.c,
ID,
and README.

Each student will hand in their solution file, which you can validate
by hand by running their custom bomb against their solution:

```
    linux> cd ./bombs/bomb<n>
    linux> ./bomb < student_n_solution.txt
```

The source code for the different phase variants is in ./src/phases/.