```
########################################################################
# CS:APP Buffer Lab
# Directions to Instructors
#
# Copyright (c) 2002-2012, R. Bryant and D. O'Hallaron
#
########################################################################

This directory contains the files that you will use to build and run
the CS:APP Buffer Lab.

The purpose of the Buffer Lab is to help students develop a detailed
understanding of the stack discipline on IA32 processors.  It involves
applying a series of buffer overflow attacks on an executable file.

This version of the lab has been specially modified to defeat the
stack randomization techniques used by newer versions of Linux. It
works by using mmap() and an assembly language insert to move the
stack pointed at by %esp to an unused part of the heap.

**********
1. Overview
**********

----
1.1. Buffer Bombs
----

A "buffer bomb" is an executable bomb, called "./bufbomb", that is
solved using a buffer overflow attack (exploit).  In this lab,
students are asked to alter the behavior of a buffer bomb (called
bufbomb) via five increasingly difficult levels of exploits.

The levels are called smoke (level 0), fizz (level 1), bang (level 2),
boom (level 3), and kaboom (level 4), with smoke being the simplest
and kaboom being the most difficult.

----
1.2. Solving Buffer Bombs
----
Each exploit involves reading a sequence of bytes from standard input
into a buffer stored on the stack. Students encode each exploit string
as a sequence of hex digit pairs separated by whitespace, where each
hex digit pair represents a byte in the exploit string. The program
"hex2raw" converts these strings into a sequence of raw bytes, which
can then fed to the buffer bomb:

    unix> cat exploit.txt | ./hex2raw | ./bufbomb -u <userid>

Each student works on an identical buffer bomb, but the solution to
the individual phases is a function of each student's userid.  Thus,
students must develop the solution on their own and cannot use the
solutions from other students.

The solution to each phase is unique for each student because it
typically involves the manipulation on the runtime stack of a unique
"cookie" computed from the userid by the "makecookie" program:

    unix> ./makecookie bovik
    0x1005b2b7

The lab writeup has extensive details on each phase and solution
techniques.

----
1.3. Autograding Service
----

We have provided the same stand-alone user-level autograding service
```

used by the Bomb Lab to handle all aspects of the Buffer Lab for
you. Students download their buffer bombs from a server. As the
students work on their bombs, they can submit successful exploit
strings to the server by running the buffer bomb with "-s" argument:

```
    unix> cat exploit.txt | ./hex2raw | ./bufbomb -u <userid> -s
```

The current results for each bomb are displayed on a Web "scoreboard."
As with the Bomb Lab 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 ./buflab directory:

- Request Server. Students download their bombs and display the
scoreboard by pointing a browser at a simple HTTP server called the
"request server."

- Result Server. Each time a student submits an exploit string
the buffer 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. The "report daemon" periodically scans the scoreboard
log file. The report daemon finds the most recent exploit string
submitted by each student for each phase, and validates these strings
by applying them to a local copy of the buffer bomb (unlike the Bomb
Lab, each student works on the same buffer bomb). It then updates the
HTML scoreboard that summarizes phases that have been successfully
solved for each bomb (identified by cookie to protect the student
privacy), rank ordered by the number of solved levels.

To avoid infinite loops during validation, the Report Daemon calls
each bufbomb in autograding mode, using the -g flag. This causes the
bomb to timeout after 5 seconds.

- Main daemon. 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 ./buflab directory contains the following files:

```
Makefile                - For starting/stopping the lab and cleaning files
buflab-handout/         - Contains the files handed out to each student
buflab.pl*              - Main daemon that nannies the other servers & daemons
Buflab.pm               - Buflab configuration file
buflab-reportd.pl*      - Report daemon that continuously updates scoreboard
buflab-requestd.pl*     - Request server that serves bombs to students
buflab-resultd.pl*      - Result server that gets autoresult strings from bombs
buflab-scoreboard.html  - Real-time Web scoreboard
buflab-update.pl*       - Helper to buflab-reportd.pl that updates scoreboard
handin/                 - Most recent exploits from each student and each level
log-status.txt          - Status log with msgs from various servers and daemons
log.txt                 - Scoreboard log of autoresults received from bombs
makebomb.pl*            - Program that builds a buffer bomb
scores.txt              - Summarizes current scoreboard scores for each student
src/                    - The buffer bomb source files, including a master
                                        - solver in ./src/solve the automatically generates
                                        - a solution string for any userid and level.
writeup/                - Sample Latex Buffer Lab writeup
```

**************************

3. Buffer Bomb Terminology
**********
Notifying Bomb: A buffer bomb can be compiled with a NOTIFY option
that allows the student to submit successful exploit strings to the
autograding service. Such bombs are called "notifying bombs."

Quiet Bomb: A buffer bomb that is not a notifying bomb is called a
"quiet bomb."

Cookie: Unlike the Bomb Lab, each student works on the same
binary. However, the solution to each phase is different for each
student because the exploit string typically must contain a 32-bit
"cookie" that is computed from the student's userid.

**********
4. Offering the Buffer Lab
**********
As with the Bomb Lab, there are two basic flavors of the Buffer Lab:
In the "online" version, the instructor uses the autograding service
to handout buffer bombs to each student on demand (each student gets
the same bomb program), 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 versions 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 Buffer Lab Directory
---

Identify the generic Linux machine ($SERVER_NAME) where you will
create the Buffer Lab directory (./buflab) and, if you are offering
the online version, run the autograding service. You'll only need a
user account on this machine. You don't need root access. Any desktop
with an internet connection will do.

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

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

---
4.2 Configure the Buffer Lab
---
Configure the Buffer Lab by editing the following file:

./Buflab.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 all 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 submit results from.  Make sure you
update this correctly, else you and your students won't be able to

submit their results. You should include $SERVER_NAME in this list,
along with any machines that your students will be submitting from.

---
4.3. Update the Lab Writeup
---
Once you have updated the configuration files, modify the Latex lab
writeup in ./writeup/buflab.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 Buffer Lab
---

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

(1) Reset the Buffer 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 ("make cleanallfiles") deletes all handin
strings, status logs, and the scoreboard log. Do this only during
debugging, or the very first time you start the lab for your students.

The ./src/solve directory contains a master solver script called
./solve.pl that you can use to automatically generate a nicely
commented solution exploit string for any userid and phase. See
./src/solve/README for details.

The ./handin directory contains the most recent exploit string
received from each userid and each phase

```
<userid>-<level>-exploit.txt
```

as well as a report showing the result from the validation

```
<userid>-<level>-report.txt
```

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

Students submit their successful exploit strings to the grading
service by calling the bufbomb with the -s option:

```
linux> cat exploit_string | ./hex2raw | ./bufbomb -u bovik -s
```

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

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

(1) Resetting the Buffer Lab. "make cleanallfiles" resets the lab from scratch, deleting all data specific to a particular instance of the lab, such as the status log, the scoreboard log, and the handin file. 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 handins. You'll have to ask them to submit their results again.

(2) Starting the Buffer Lab. "make start" runs buflab.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 Buffer 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 buffer 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 by pointing their favorite browser at

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

The request server makes the buflab-handout directory (if necessary), tars it up, 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 (./buflab-handout) with the following three binary files:

```
bufbomb*     Notifying custom bomb executable
hex2raw*     Converts a hex-encoded exploit string to a byte stream
makecookie*  Computes the cookie associated with a userid.
```

The request server also makes sure that there are bufbomb, hex2raw, and makecookie binaries in the ./buflab/src directory.

Result Server: Each time a student submits an exploit string, the buffer bomb sends an HTTP "autoresult string" to the result server, which then appends the message to the scoreboard log. Each autoresult string contains the userid, level, and exploit string.

Report Daemon: The report daemon periodically scans the scoreboard log (log.txt) and updates the scoreboard (buflab-scoreboard.txt). For each userid, it validates the most recently received exploit string, and updates the scoreboard. The update frequency is a configuration variable in Buflab.pm.

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

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

```
------
4.4.3. Grading the Online Buffer Lab
------
The online Buffer Lab is self-grading. At any point in time, the
tab-delimited file "./buflab/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.

The autograding service also updates the ./buflab/handin directory,
which contains information that will be very helpful to you when
students have questions about their submissions, especially when
they submit solutions that passed their bufbomb but failed when
evaluated by the grading server (common for the nitro phase).

For each student (userid), it keeps the last exploit string they
submitted for each phase (0-4) in a file called

    <userid>-<level>-exploit.txt

along with a report file called

    <userid>-<level>-report.txt

that shows the output from the buffer bomb on the corresponding
string.

Section 5 describes how to generate solutions for arbitrary userids
and levels.

------
4.4.4. Additional Notes on the Online Buffer Lab
------

* If you want to build your own notifying bomb and hand it out
  manually to your students (rather than having them use the request
  server), then you should use the "./makebomb.pl" script:

  unix> ./makebomb.pl -n

  This creates the three binaries, ./src/bufbomb, ./src/hex2raw, and
  ./src/makecookie, and copies them to ./buflab-handout/, which you can
  then tar up and hand out to students.

* 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 buflab/log-status.txt. Servers run quietly, so they
  can be started from initrc scripts at boot time.

* 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 buffer 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 solve and submit different
phases for different fake userids to make sure that they 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 Buffer Lab

---

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

Use the ./makebomb.pl script to build the quiet buffer bomb:

```
    unix> ./makebomb.pl
```

This creates the three binaries ./src/bufbomb, ./src/hex2raw, and ./src/makecookie, and copies them to the ./buflab-handout/ directory, which you can then tar up and hand out to students.

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

```
    unix> cd src
    unix> cat <userid>-<level>.txt | ./hex2raw | ./bufbomb
```