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

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

The purpose of the Attack Lab is to help students develop a detailed
understanding of the stack discipline on x86-64 processors.  It
involves applying a total of five buffer overflow attacks on some
executable files. There are three code injection attacks and two
return-oriented programming attacks.

The lab must be done on an x86-64 Linux system. It requires a version
of gcc that supports the -Og optimization flag (e.g., gcc
4.8.1). We've tested it at CMU on Ubuntu 12.4 systems.

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

----
1.1. Targets
----
Students are given binaries called ctarget and rtarget that have a
buffer overflow bug.  They are asked to alter the behavior of their
targets via five increasingly difficult exploits. The three attacks on
ctarget use code injection. The two attacks on rtarget use
return-oriented programming.

----
1.2. Solving Targets
----
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 be fed to the target:

    unix> cat exploit.txt | ./hex2raw | ./ctarget

Each student gets their own custom-generated copy of ctarget and
rtarget.  Thus, students must develop the solutions on their own and
cannot use the solutions from other students.

The lab writeup has extensive details on each phase and solution
techniques. We suggest that you read the writeup carefully before
continuing with this README file.

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

As with the Bomb and Bufer Labs, we have created a stand-alone
user-level autograding service that handles all aspects of the Attack
Lab for you: Students download their targets from a server. As the
students work on their targets, each successful solution is streamed
back to the server. The current results for each target 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 ./attacklab directory:

- Request Server (attacklab-requestd.pl). Students download their
targets and display the scoreboard by pointing a browser at a simple
HTTP server called the "request server." The request server builds the
target files, archives them 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 targets and their
solutions for the instructor in the targets/ directory.

- Result Server (attacklab-resultd.pl). Each time a student correctly
solves a target phase, the target 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" called log.txt.

- Report Daemon (attacklab-reportd.pl). The "report daemon"
periodically scans the scoreboard log file. The report daemon finds
the most recent autoresult string submitted by each student for each
phase, and validates these strings by applying them to a local copy of
the student's targets. It then updates the HTML scoreboard
(attacklab-scoreboard.html) that summarizes the current number of
solutions for each target, rank ordered by the total number of accrued
points.

- Main daemon (attacklab.pl). The "main daemon" starts and nannies the
request server, result server, and report daemon, 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 ./attacklab directory contains the following files:

```
Makefile                - For starting/stopping the lab and cleaning files
attacklab.pl*           - Main daemon that nannies the other servers & daemons
Attacklab.pm            - Attacklab configuration file
attacklab-reportd.pl*   - Report daemon that continuously updates scoreboard
attacklab-requestd.pl*  - Request server that serves targets to students
attacklab-resultd.pl*   - Result server that gets autoresult strings from targets
attacklab-scoreboard.html - Real-time Web scoreboard
attacklab-update.pl     - Helper to attacklab-reportd.pl that updates scoreboard
targets/                - Contains unique targets generated for each student, with
solutions
log-status.txt          - Status log with msgs from various servers and daemons
log.txt                 - Scoreboard log of autoresults received from targets
scores.csv              - Summarizes current scoreboard scores for each student
src/                    - Attacklab source files
validate.pl             - Called periodically by report daemon. Validates solutions
                          for each student, and updates scoreboard and scores files.
writeup/                - Sample Latex Attack Lab writeup
```

****
3. Solutions
****

TargetID: Each target in a given instance of the lab has a unique
non-negative integer called the "targetID."

The five solutions for target n are avalable to you in the
targets/target<n> directory, in the following files:

Phase 1: ctarget.l1,
Phase 2: ctarget.l2,
Phase 3: ctarget.l3,
Phase 4: rtarget.l2,
Phase 5: rtarget.l3,

where "l" stands for level.

****
4. Offering the Attack Lab
****

There are two basic flavors of the Attack Lab: In the "online"
version, the instructor uses the autograding service to handout custom
targets 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 targets 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 Clean Attack Lab Directory

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

The machine must be x86-64 and have a relatively recent version of gcc
that supports the -Og optimization flag (e.g., gcc 4.8.1).

Each offering of the Attack Lab must start with a clean new ./attacklab
directory on $SERVER_NAME. For example:

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

4.2 Configure the Attack Lab

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

./Attacklab.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.

./src/build/config.c - This file gives the course number and lists the

domain names of the hosts that targets are allowed to run on. Make
sure you update this correctly, else you and your students won't be
able to run your targets. You should include $SERVER_NAME (the machine
running the attacklab servers) in this list, along with any machines
that your students are allowed to submit from.

./src/build/driverhdrs.h - This file provides the SERVER_NAME where
the attacklab servers are running. It must be identical to the
$SERVER_NAME in Attacklab.pm.

4.3. Update the Lab Writeup

Modify the Latex lab writeup in ./writeup/attacklab.tex for your
environment. Then type the following in the ./writeup directory:

        unix> make clean; make

4.4. Offering the Online Attack Lab

4.4.1. Short Version

From the ./attacklab directory:

(1) Reset the Attack Lab from scratch once 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 targets, 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 their custom targets by pointing their browsers at
    http://$SERVER_NAME:15513/

Students view the scoreboard by pointing their browsers at
    http://$SERVER_NAME:15513/scoreboard

where $SERVER_NAME is defined in Attacklab.pm.

4.4.2. Long Version
------

(1) Resetting the Attack 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 targets 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 targets for yourself, run them, solve a few phases (using the solutions provided to instructors in the targets directory), 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 targets are allowed to run on in src/build/config.c) you stop the servers, fix the configuration, reset the lab, and then request and run more test targets.

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

(2) Starting the Attack Lab. "make start" runs attacklab.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 Attack 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 targets to student browsers on demand, and (2) displays the current state of the real-time scoreboard.

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

    http://$SERVER_NAME:15513/

For example, http://foo.cs.cmu.edu:15513/. 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 target with targetID=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 (./target<n>) with the following files:

README.txt: A file describing the contents of the directory.

ctarget: An executable program vulnerable to code-injection attacks.

rtarget: An executable program vulnerable to return-oriented
    programming attacks.

cookie.txt: An 8-digit hex code that serves as a unique identifier in
    the attacks submitted to the results server.

farm.c: The source code of this target's ``gadget farm,'' which are
    used in generating return-oriented programming attacks.

hex2raw: A utility to generate attack strings.

The request server also creates a directory (attacklab/targets/target<n>) that contains these files, along with the solutions for each of the five phases. See the targets/target<n>/README.txt for details.

Result Server: Each time a student successfully solves a phase, the target 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 targetID, the phase number, and the exploit string.

Report Daemon: The report daemon periodically scrapes the scoreboard log, validates the most recent submission from each target, and updates the Web scoreboard with the results rank-ordered by score and submission time. It also creates a CSV file with the scores for each student. The update frequency is a configuration variable in Attacklab.pm.

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

```
http://$SERVER_NAME:15513/scoreboard
```

4.4.3. Grading the Online Attack Lab

The online Attack Lab is self-grading. At any point in time, the CSV file ./attacklab/scores.csv 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 Attack Lab

---

* Since the request server and report daemon both need to execute targets, you must include $SERVER_NAME in the list of legal machines in your attacklab/src/build/config.c 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 attack/log-status.txt. Servers run quietly, so they can be started from initrc scripts at boot time.

* See the README files in src/build/ and src/solve for details about how targets are constructed and solved. Not necessary, but included 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 target for ourselves by pointing a Web browser at

```
http://$SERVER_NAME:15513
```

After saving our target to disk, we untar it, copy it to a host in the approved list in src/build/config.c, and then solve it using the solution files in targets/target<n>: ctarget.l1, ctarget.l2, ctarget.l3, rtarget.l2, and rtarget.l3.

Then we check the scoreboard to see that it's being updated correctly:

recorded on the scoreboard, which we check at

```
    http://$SERVER_NAME:15513/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. Offering the Offline Attack Lab

In this version of the lab, you build your own custom targets manually and
then hand them out to the students. The students work on solving
their targets offline (i.e., independently of any autograding service)
and then handin their solution files to you, each of which you grade
manually by feeding the exploit strings to the copy of the student's
targets.

You generate targets manually using the the src/build/buildtarget.pl
script, which must be run from the src/build directory. For example,
to generate target 2 for user bovik:

```
    linux> cd src/build
    linux> ./buildtarget.pl -u bovik -t 2
```

This will generate a file called targets/target2.tar, which you then
hand out to the student.

Students will need to run ctarget and rtarget using the -q option,
which tells the targets not to try to contact the (non-existent)
grading server.

The student will hand in up to five exploit strings: bovik-ctarget.l1,
bovik-ctarget.l2, bovik-ctarget.l3, bovik-rtarget.l2, and
bovik-rtarget.l3. You evaluate each exploit by feeding it to the
ctarget or rtarget programs in the targets/target2directory. For
example:

```
    linux> cat bovik-ctarget.l3 | ./src/hex2raw | ./targets/target2/ctarget -q
    linux> cat bovik-rtarget.l2 | ./src/hex2raw | ./targets/target2/rtarget -q
```