

HOMWORK 2

ENTS 689i Network Immunity
Fall 2008

Overview

In this assignment, you will utilize the course's virtual isolated network, `hack.er`, to perform a set of experiments and answer a list of questions about a specific buffer overflow in a particular program. The goal of this assignment is to become more familiar with the low-level steps involved in finding, exploiting, and mitigating such vulnerabilities.

Deliverables

The single deliverable for this assignment will be the answers to the bold, numbered questions embedded in the procedure below. **START EARLY!**

This assignment is due no later than **12:01am EDT on October 20th, 2008 (midnight Sunday night)**. All completed answers must be sent electronically to npetroni@cs.umd.edu in either **plain-text ASCII** or **PDF** format. All other formats will receive **NO CREDIT**. Any mail arriving later than 12:01am on Monday morning (as determined by my mail server's time stamp) will be considered late. The standard course grading policy will apply for late homework on this assignment. If you turn in several copies of this assignment, the last one received will be graded and all others discarded.

This assignment may be completed alone **or with one partner** from class. If you work with a partner, be sure to include **both names** on the answers that you turn in. Both partners will receive the same grade on this assignment. General familiarity with Linux, C programming, and debugging will be very helpful in this assignment, but are not strictly necessary. You may benefit from choosing a partner who has experience in this environment.

Using the Virtual Network

All experimental work on this assignment is to be done on the `hack.er` virtual network. To login to this network, you will need the credentials that were distributed in class on October 9, 2008. The same credentials will be used for the remainder of this course. Our virtual network runs on a single VMware ESX server. This is a very powerful machine, but it is also a shared resource. Please be conscious of how many resources you are using at a given time. For this assignment, there is no reason to ever have more than one VM (per team) running at a time. Additionally, you may be able to control the virtual machines of other students using the VMware interface. ***Please only modify the state of your own virtual machine, as described below.***

The `hack.er` network is completely isolated from the outside world. There is no way to transfer files out of it directly. Therefore, all assignment answers should be recorded **outside** of `hack.er`, on your regular Internet-facing machine and sent using the public email system.

To login to our network, you must use the VMware Infrastructure Client, which only runs on Windows systems. You can use your own Windows system or the ENTS laboratory, which has the VM Client already installed. You can download the VM Client from <https://129.2.94.253/client/VMware-viclient.exe>. You should expect a certificate error from your browser, which you can safely ignore.

Procedure

NOTE: only perform each step **ONCE** per team. You do not need two VMs (or answer sheets) if you are working with a partner. One is sufficient.

- I. Log into the virtual network.
 1. Obtain the VM Client program described in the previous section, or log into one of the ENTS cluster machines and use the client that is already installed there.
 2. Log into the class ESX server at IP address 129.2.94.253 using the username and password provided in class.
 3. Once you log in, you will see a tree of servers and virtual machines on the left-hand side of the GUI. If you do not, go to View->Inventory->Virtual Machines and Templates.
 - II. Clone your own copy of the HW2 virtual machine.
 1. In the left-hand tree of the GUI, click on the VM template labeled “hw2 template.”
 2. In the main GUI window, select “Deploy to new virtual machine.”
 3. In the resulting dialog, give your VM a name as follows: <username>-hw2. For example, if you logged into the virtual infrastructure with login *student1*, then you would name your VM “student1-hw2.” If working with a partner, name your VM with both usernames, e.g., “student2-student23-hw2.” Click Next.
 4. Select the server 129.2.94.252 and click Next.
 5. Choose the defaults for the remaining wizard items, clicking Next at each step until you are done. Your VM will take several minutes to be copied and created. You can monitor the status of your VM in the “Recent Tasks” section of the client GUI.
 - III. Start your virtual machine and log in.
 1. Once your VM has completed creation, click on it in the left-hand tree of the GUI. In the main GUI window, click on “Power on the virtual machine” to start your VM.
 2. Click on the console tab or launch the console viewer in a separate window. You can make your console full-screen by using the View menu of the console window.
 3. Your target VM is a Fedora 9 Linux machine. Once the VM is finished booting, you should see a Fedora login screen. Click on the user named “User” and login with the password “us3rpw”.
 4. Several steps in this procedure require superuser privileges. For those steps, we will say “as superuser.” Once in the VM, you can become superuser by typing `su` at any console window. The root password is “r00tpw”. Your prompt will change for that console. Note that **ONLY** that console is now operating with superuser privileges. If you want root in multiple windows, you will need to `su` in those as well or launch new terminals as root.
- NOTE: Your virtual machine will stay running until you explicitly shut it down, as described at the end of this procedure. Even if you log out of the VM client, you may still have a running VM. It is perfectly acceptable to leave your VM on and in a temporary state, but there is no guarantee that your work will not be lost due to power outage, system administrator action, etc. Therefore, the safest thing for you to do if you are unfinished is to create a “Snapshot” of your VM, or simply shutdown the VM and restart it later. If you do use snapshots, please do not keep more than one at any given time (that is, delete your old snapshots) so that we can save disk space.**
- IV. Download and build the required software packages.

1. Open a web browser (click the Firefox logo at the top of the desktop) and direct your browser to `http://www.hack.er`. On the main page of the website you will see links to several software packages that are required for this assignment, as well a copy of this assignment sheet and possibly other info. Download the following software to your VM:
 - a. The Art of Fuzzing (`taof-0.3.2.tgz`)
 - b. The tinyhttpd web server (`tinyhttpd.tar.gz`)
 - c. The tinyexploit exploit program (`tinyexploit.tar.gz`)
2. Open a command terminal by clicking on the terminal icon (top middle of the desktop).
3. Change directory into the folder where you downloaded the software above. If you chose the default, you need to go to the `Download` folder using the command `“cd Download”`. Once in this folder, type `“ls”` and you should see the three software packages.
4. Unpack each software package using the command `“tar xvfz <software package>”`. You have to run this command once for each program that you downloaded. If you run `“ls”` after unpacking the download, you will see a new directory created for it.
5. Enter the tinyhttpd directory (`“cd tinyhttpd”`) that you just created and build the software by typing `“make”`. You can ignore any compiler warnings (although there should be none). There are three versions of tinyhttpd, which will be referred to as `“safe,” “broken,”` and `“protected.”`
 1. `httpd-safe` is a correct program that does not have any known flaws in it.
 2. `httpd-broken` is a program that contains a low-level programming flaw.
 3. `httpd-protected` is the same program as `httpd-broken`, but has been built using some compiler protections that attempt to mitigate low-level flaws.

QUESTION 1: (5 pts) Briefly describe (no more than one paragraph) the low-level flaw that has been inserted into `httpd-broken.c`.

Hint: you can use the UNIX `diff` program to see which places differ between the broken and safe versions of the program.

6. All versions of `httpd` will bind to port 8888 on your local system. Therefore, you cannot run more than one version of `httpd` at once on a given VM, but you should not need to do this.
 7. As a test, run any version of `httpd`. For example, type `“./httpd-safe”`. The web server will print `“httpd running on port 8888”` and not return a prompt.
 8. Using web browser, connect to the web server that you just launched using the URL `http://localhost:8888`. Once you have confirmed it is working, you can shutdown your `httpd` (type Control-C in the console where you started it).
- V. Fuzzing for vulnerabilities.
1. Launch the safe version of `httpd` by running `“./httpd-safe”` in the `tinyhttpd` directory.
 2. In another console (click the button at the top of the desktop again), enter the `taof-0.3.2` directory that you unpacked above (step 4 in the previous section) and **as superuser** (`su` first) run Taof. To run Taof, run the command `“python2.4 taof.py”`. Note that there are multiple versions of Python installed in your VM. Taof requires version 2.4. If run correctly, you should see the Taof fuzzing GUI appear.

3. In the Taof GUI, click on “Data Retrieval.” You should see a new dialog box appear.
4. Click on “Network Settings.” Again, a new dialog should appear.
5. Enter the following information in the network settings dialog and click OK:
 1. Local Server: 127.0.0.1
 2. Local Port: 8889
 3. Remote Server 127.0.0.1
 4. Remote Port 8888
 5. Select tcp
6. Taof should now be ready to run, and will indicate this by printing “Please click start when ready.” Click the “start” button.
7. Open a web browser and go to the URL `http://localhost:8889`. This URL is Taof acting as a pass-through to the real httpd server. After making one or more requests in the web browser, go back to Taof and click the “stop” button. The capture dialog should disappear.
8. In the “Request List” section of the main Taof GUI, select the first packet (or the only packet, if there is only one) that you just sent. It will look like an HTTP GET request. Click the “Set Fuzzing Points” button.
9. In the new dialog that appears, select the entire HTTP request (the ASCII box, beginning “GET ...” and click the “+ Add” button. You should see a new entry in the “Fuzzing Points” table at the bottom of the dialog box. Once you have a single fuzzing point, click “OK” to dismiss the dialog.
10. In the main Taof window, click the “Fuzzing” button. The settings should be, as before, 127.0.0.1 and port 8888.
11. Click “Start” to begin fuzzing httpd.
12. After a couple of minutes, all tests will have run and the fuzzer will tell you that it is complete. Leave the fuzzer open, you will use it again shortly.
13. In a console window, look at the `taof-0.3.2` directory that you created before. You should notice a new directory labeled something like `fuzzing_session-XXX-XX.XX.XX`. (each X will be a number instead). Enter this directory and look at (for example, using the “less” program) the file labeled “debugging,” which is a log of all requests that were sent to the http server.

NOTE: in the same directory, Taof will leave a file called `fuzz.xml` that can be re-loaded to reuse the exact experiment details. If you need to close/re-open Taof, you may want to use this file.

QUESTION 2: (5 pts) Briefly (no more than one paragraph) describe the buffer overflow test that the fuzzer performed. What patterns of requests did it make and why?

14. Return to the console where you launched httpd and stop it from running (control-c). Now run the broken version, but this time do it using the debugger as follows:
 1. Type `'gdb ./httpd-broken'`
 2. At the gdb prompt, type `'handle SIGPIPE nostop noprint nopass'`. This step prevents the

debugger from stopping httpd between each fuzzing request.

3. At the gdb prompt, type 'run'. You should see the standard httpd start greeting.
15. Return to Taof and again click “Start.” This time, the fuzzer should stop before completing all tests and the program will crash, returning you to a gdb command prompt.

QUESTION 3: (10 pts) What happened? Why did the program crash? What is the value of eip after this test and why? What do you see in the Taof debugging file this time?

Hint: use gdb's “print \$eip” and “x/10 \$esp” commands.

16. Leave Taof open, you will use it again later on.

VI. Exploit httpd.

1. Re-run httpd-broken by again typing “run” in the gdb prompt.
2. In a new console, enter the directory `tinyexploit` that you created previously. Run the command “./validrequest.sh”. This generates a normal http request, like a browser, but discards the output. Look at the output of the httpd-broken program inside of gdb and look at the location of buf that was printed. Note that this is cheating. In the real world, the attacker would need to figure out this value or develop his exploit in a position independent manner.

QUESTION 4: (5 pts) At what address does httpd-broken say buf is located?

3. Build the tinyexploit program by typing 'make' at the command prompt.
4. Run the exploit by typing `./tinyexploit <bufaddr>`, where bufaddr is the location answered in the previous question. Note that you have to prepend the hex number with the characters '0x' just like they were printed. The target httpd program should not crash, but something else should happen.

QUESTION 5: (5 pts) What happened when you ran the exploit? What output did httpd give?

QUESTION 6: (20 pts) Draw a picture of the contents of the stack frame for the call to accept_request just BEFORE the attacker inserts his data (i.e., before the call to get_line). ASCII art is fine. Include, at a minimum, the locations of all local variables, the return address, the frame pointer, and the stack pointer. Your drawing should use abstract terms like “return address of accept_request” to describe the stack, not the actual addresses that are stored on it.

QUESTION 7: (20 pts) Draw a picture of the contents of the stack frame for accept_request just AFTER the attacker has inserted his data (i.e., after the call to get_line). The same guidelines apply for this stack drawing.

Hint: Re-run the exploit, but this time set a breakpoint at the call to accept_request (type 'break accept_request' before typing 'run'). You can then single-step the program to see what happens by running the 'next' command repeatedly until you are at the appropriate program point. You can then print the value of \$eip or \$esp using the “print” command and you can dump memory locations using the “x” command. For example, “x /20 \$esp” will print 20 doublewords from the stack. NOTE: You will need to quit gdb and restart it if the exploit was successful.

5. Quit gdb when you are finished.

VII. Compiler protections.

1. Run the “protected” version of tinyhttpd by running “./httpd-protected”
2. In the Taof GUI, restart the fuzzer until httpd-protected stops running.

QUESTION 8: (5 pts) What output did httpd-protected give? You don't have to give the exact output, just describe it.

QUESTION 9: (10 pts) Why did this happen? Name two protections that the compiler could have added and explain them.

3. Close Taof. You are now finished with it.

VIII.. Kernel protections.

1. In a console, and **as superuser**, type the following command:
`echo '1' > /proc/sys/kernel/randomize_va_space`
2. Re-launch httpd-broken and run the validrequest.sh command. Note the address of buf.
3. Repeat the last step (restarting httpd-broken AND re-running validaterequest.sh) at least five times.

QUESTION 10: (5 pts) What do you notice about the address of buf for each new run of httpd-broken? Why do you think this happened?

4. In a console, and **as superuser**, type the following command:
`echo '1' > /proc/sys/kernel/exec-shield`
5. Re-launch httpd-broken and run the validaterequest.sh command to get the address of buf.
6. Run tinyexploit using the address of buf from the previous step, attempting to again exploit httpd-broken as before.

QUESTION 11: (10 pts) Did the exploit succeed? What is the last kernel log message when you run 'dmesg' from the command line? Why do you think happened?

IX. Cleaning up.

1. Congratulations, you have complete HW 2! There are a few housekeeping items left to do.
2. First, shutdown your virtual machine using the normal shutdown mechanism:
System->Shutdown and click “Shutdown”
3. Once your VM is shutdown, right-click on your VM in the VMware client and click “rename”. Add the suffix “-finished”, e.g., “student1-hw2-finished”, to your VM's name.
DO NOT DELETE YOUR VM.
4. You can now close the VMware client.
5. Don't forget to send your answers to **npetroni@cs.umd.edu** in ASCII or PDF format.