Paxson Spring 2011	CS 161 Computer Security	Midterm 1

PRINT your name: ______________________, ______________________
(last) (first)

SIGN your name: ____

PRINT your class account login: `cs161-`__________

Your TA's name: ____

Your section time: ____

Name of the person sitting to your left: ________________ Name of the person sitting to your right: ________________

You may consult one sheet of paper (double-sided) of notes. You may not consult other notes, textbooks, etc. Calculators and computers are not permitted. Please write your answers in the spaces provided in the test. We will not grade anything on the back of an exam page unless we are clearly told on the front of the page to look there.

You have 80 minutes. There are 7 questions, of varying credit (100 points total). The questions are of varying difficulty, so avoid spending too long on any one question.

Do not turn this page until your instructor tells you to do so.

Question	Points	Score
Short Answer	18	
Memory Safety	15	
Denial-of-service	20	
Spoofing	15	
Surviving Hot Spots	12	
Javascript	10	
XSS Defense	10	
Total:	100	

Problem 1 ***Short Answer*** **(18 points)**

(a) (3 points) The Soda Hall elevator requires card key access to go to the 4th floor. The door connecting the west stairwell and the 4th floor opens without cardkey. What security principle is missing in this design?

CIRCLE the best answer, and briefly explain.

(i) Fail-safe defaults. (ii) Complete mediation.
(iii) Separation of responsibilities. (iv) Least privilege.
(v) Human factors matter.

Justification:

(b) (3 points) Agree or disagree, and briefly explain: *It's better to use library calls with rich functionality, like* `system()`*, than those with more narrow functionality, like* `execve()`*, because then your code that uses the library is simpler, and so more likely to be correct.*

(c) (3 points) Agree or disagree, and briefly explain: *When sanitizing inputs to avoid injection attacks, it's better to use a white-list approach than a black-list approach.*

(Con't)

(d) (3 points) Agree or disagree, and briefly explain: *You can tell whether a web page your browser displays represents a phishing attack on `mybank.com` by checking whether the URL in the address bar starts with `http://mybank.com`.*

(e) (3 points) Suppose you have both the source code and a binary corresponding to a 5,000,000-line C program. Explain a technique you could use to try to discover whether it has a memory-safety vulnerability.

(f) (3 points) List three problems with using CAPTCHAs.

Problem 2 ***Memory Safety*** **(15 points)**

For the following code, assume an attacker can control the values of `item` and `n` passed into `print_report()`. CIRCLE any memory safety problems in the code and briefly explain them; or, if you believe there are no such problems, explain why the code is safe.

```
/* Returns in "plural" the plural of the given string "str". */
void plural(char* str, char* plural)
{
    char* buf;
    size_t n;

    plural[0] = '\0';

    if ( str == 0 ) return;

    n = strlen(str);
    if ( n == 0 ) return;

    buf = malloc(n+1);
    if ( buf == 0 ) return;

    char last = str[n-1];
    if ( last == 's' )
        /* Assume it's already plural. */
        strcpy(buf, str);
    else if ( last == 'h' )
        /* fancy plural, like church -> churches */
        sprintf(buf, "%ses", str);
    else /* regular plural */
        sprintf(buf, "%ss", str);

    strcpy(plural, buf);
    free(buf);
}

void print_report(char *item, int n)
{
    char item_plural[256];
    plural(item, item_plural);
    printf("We sold %d %s\n", n, item_plural);
}
```

Reminder: `strlen(s)` calculates the length of the string s, not including the terminating `'\0'` character. `strcpy(dst, src)` copies the string pointed to by `src` to `dst`, including the terminating `'\0'` character. `sprintf` works exactly like `printf`, but instead writes to the string pointed to by the first argument. It terminates the characters written with a `'\0'`.

Problem 3 ***Denial-of-service*** **(20 points)**

An anti-spam company, GreenMail, uses a vigilante approach to fighting spam.[1] GreenMail's customers report their spam to GreenMail, and the company then automatically visits the websites advertised by the URLs in the spam messages and leaves complaints on those websites. For each spam that a user reports, GreenMail leaves a generic complaint. GreenMail operates on the assumption that as the community grows, the flow of complaints from hundreds of thousands of computers will apply enough pressure on spammers and their clients to convince them to stop spamming.

After a short while of operation, GreenMail's public web site comes under a massive DDoS attack that uses SYN flooding.

(a) (3 points) Briefly describe the type of traffic that an attacker sends to launch a SYN flooding attack.

(b) (3 points) Briefly describe how the attack can cause a denial-of-service.

(Con't)

[1]FYI, this is based on an actual company and its experiences.

(c) (6 points) Can GreenMail use a packet-filter firewall to defend itself against the DDoS that uses SYN flooding?

If so, describe what sort of rule or rules the firewall would need to apply, and what "collateral damage" the rules would incur.

If not, explain why not.

(d) (4 points) Explain how the GreenMail service could itself be used to mount a DoS attack.

(e) (4 points) Briefly describe one approach that victims could use to defend themselves against the attack you sketched.

Problem 4 *Spoofing* **(15 points)**

Suppose we could deploy a mechanism that would ensure IP source addresses always correspond to the actual sender of a packet—in other words, suppose it is impossible for an attacker to spoof source addresses. For each of the following threats, explain whether (and briefly why) the mechanism would either:

(i) Completely eliminate the threat.

(ii) Eliminate some instances of the threat, but not all of them.

(iii) Have no impact on the threat.

Here are the threats:

(a) (3 points) Reflected cross-site scripting (XSS) attacks.

(b) (3 points) Kaminsky's DNS cache poisoning attack.

(c) (3 points) DoS "amplification" where the attacker sends traffic to an Internet "broadcast" address.

(d) (3 points) Clickjacking.

(e) (3 points) Format string vulnerabilities.

Problem 5 ***Surviving Hot Spots*** **(12 points)**

You go into Javalicious, a coffee shop with free WiFi. You settle in for an afternoon of web-surfing and tweeting. You learn that the network sends all packets unencrypted. You find this disconcerting. Worse, you see Prof. Evil seated at the table next to yours, using a laptop connected to the same WiFi network.

For your HTTP web connections, consider the basic security properties of *confidentiality*, *integrity*, and *availability*. For each of these, Circle YES if Prof. Evil can undermine the given property, or NO if not. Justify your answer.

Confidentiality (4 points): YES, can undermine NO, can't undermine

Justification:

Integrity (4 points): YES, can undermine NO, can't undermine

Justification:

Availability (4 points): YES, can undermine NO, can't undermine

Justification:

Problem 6 ***Javascript*** **(10 points)**

Suppose a user turns Javascript completely off in their browser. For each of the following threats, indicate whether (and briefly explain why) doing so would either:

(i) Completely eliminate the threat.

(ii) Eliminate some instances of the threat, but not all of them.

(iii) Have no impact on the threat.

Here are the threats:

(a) (2 points) Cross-site request forgery (CSRF).

(b) (2 points) SQL injection.

(c) (2 points) Reflected cross-site scripting (XSS).

(d) (2 points) Stored cross-site scripting (XSS).

(e) (2 points) Phishing.

Problem 7 ***XSS Defense*** **(10 points)**

Suppose a browser attempts to protect the user from XSS attacks as follows. Any time the user clicks on a URL link on a page the browser is displaying, the browser scans the URL looking for any text that matches the pattern `/<[^>]*>/`. This regular expression will match an open-tag character (`<`), followed by an arbitrary number of characters other than a close-tag character, followed by a close-tag character (`>`).

If the browser finds a match, it remembers the associated text. It then scans the response it receives for the URL for any instance of that text. If it finds the same text in the response as it did in the match, then it will interpret the response as a pure HTML document, and refuse to execute any script that it contains.

(a) (5 points) Briefly explain to what degree this mechanism protects against reflected XSS.

(b) (5 points) Briefly explain to what degree this mechanism protects against stored XSS.