Paxson Spring 2017 — CS 161 Computer Security — Midterm 1

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

I am aware of the Berkeley Campus Code of Student Conduct and acknowledge that any academic misconduct will be reported to the Center for Student Conduct, and may result in partial or complete loss of credit.

SIGN your name: _____

PRINT your class account login: `cs161-`_____ and SID: _____

Your TA's name: _____

Your section time: _____

Exam # for person sitting to your left: _____

Exam # for 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, computers, and other electronic devices are not permitted.

You have 80 minutes. There are 5 questions, of varying credit (300 points total). The questions are of varying difficulty, so avoid spending too long on any one question. Parts of the exam will be graded automatically by scanning the **bubbles you fill in**, so please do your best to fill them in somewhat completely. Don't worry—if something goes wrong with the scanning, you'll have a chance to correct it during the regrade period.

If you have a question, raise your hand, and when an instructor motions to you, come to them to ask the question.

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

Question:	1	2	3	4	5	Total
Points:	64	62	58	56	60	300
Score:						

Problem 1 ***True/False*** **(64 points)**

For each of the following, FILL IN THE BUBBLE next to **True** if the statement is correct, or next to **False** if it is not. Each correct answer is worth 4 points. Incorrect answers are worth 0 points. Answers left blank are worth 1 point.

(a) Framebusting allows Javascript in an outer page to access the cookies associated with an inner page loaded in an `iframe`.

○ **True** ○ **False**

(b) `http://www.coolvids.com:3000/index.html` is in the same origin as `http://coolvids.com:3000/index.html`.

○ **True** ○ **False**

(c) Browsers apply the Same Origin Policy to determine what URLs can be loaded in `iframe`s.

○ **True** ○ **False**

(d) If Tyrion uses a browser with no code vulnerabilities and uses a unique, long password for every website he visits, then he will be safe against phishing attacks.

○ **True** ○ **False**

(e) A recommended defense against clickjacking attacks is for servers to include an HTTP `X-Frame-Options` header in its replies.

○ **True** ○ **False**

(f) HTTP-Only Cookies are designed to prevent CSRF attacks.

○ **True** ○ **False**

(g) An attacker can steal Alice's cookies for `www.squigler.com` by exploiting a buffer overflow vulnerability in Alice's browser.

○ **True** ○ **False**

(h) Executable Space Protection (e.g., DEP and W⊕X) is a defense against buffer overflow attacks.

○ **True** ○ **False**

(i) ASLR is a defense against buffer overflow attacks that requires operating system support.

○ **True** ○ **False**

(j) ASLR will prevent any attack that overflows local variables from executing injected code.

○ **True** ○ **False**

(k) Stack canaries are a defense against buffer overflow attacks that requires operating system support.

○ **True** ○ **False**

(l) Stack canaries will prevent any attack that overflows local variables from executing injected code.

○ **True** ○ **False**

(m) Stack canaries provide some protection against `printf` format string vulnerabilities, but do not protect against all such vulnerabilities.

○ **True** ○ **False**

(n) AMD's NX feature, and Intel's similar XD feature, provide protections against XSS attacks.

○ **True** ○ **False**

(o) If a web page from `abc.com` includes a script from `xyz.com`, the Same Origin Policy puts the script from `xyz.com` in the `abc.com` origin.

○ **True** ○ **False**

(p) The Same Origin Policy prevents XSS attacks if a browser implements it correctly.

○ **True** ○ **False**

Problem 2 ***Multiple Choice*** **(62 points)**

(a) (8 points) Many people lock valuables in a safe in their house in addition to locking the doors of the house. **Mark ONE** security principle that **best** fits with this approach:

- O Ensure Complete Mediation
- O Don't rely on security through obscurity
- O Defense in Depth
- O Privilege Separation

(b) (8 points) Bob places a duplicate key to his house under one particular stone in his front yard in case he forgets or loses his main key. **Mark ONE** security principle that **best** fits with his approach:

- O Ensure Complete Mediation
- O Don't rely on security through obscurity
- O Defense in Depth
- O Privilege Separation

(c) (10 points) Assume that an airport wants to achieve two security goals: **(a)** passengers can only board planes if they have boarding passes issued in their actual name, and **(b)** passengers cannot board planes unless they have undergone a security inspection by the TSA.

Consider the following design that an airport uses to try to meet these goals. The airport operators arrange that passengers can only board an airplane if they:

1. show a boarding pass and photo ID at a TSA security checkpoint, for which the photo on the ID matches the passenger presenting it, and the name on the ID matches that on the boarding pass
2. pass through a TSA security inspection at that checkpoint
3. present a boarding pass at the gate when actually going onto the plane

Also assume that the TSA correctly carries out its inspections of passengers, their boarding passes, and their photo IDs.

Mark ALL of the following concepts that are relevant to analyzing whether the airport's design achieves goals **(a)** and **(b)**. You should only consider the approach as described above. Do not consider any additional facts that you happen to know about how airport security actually works.

- O Code is Data and Data is Code
- O TOCTTOU vulnerability
- O Ensure complete mediation
- O Whitelisting
- O Injection vulnerability

(d) (12 points) Which of the following attacks can web servers protect against by sanitizing user input? **Mark ALL** that apply, even for cases where there are better ways to protect against the attack than sanitization.

- ○ XSS
- ○ Phishing
- ○ CSRF
- ○ Drive-by Malware
- ○ SQL Injection
- ○ Clickjacking

(e) (12 points) Which of the following are defenses against XSS vulnerabilities? **Mark ALL** that apply.

- ○ Use browsers written in a memory-safe language
- ○ Prevent webpages from being framed by other domains
- ○ Use Content Security Policy headers to turn off inline scripts
- ○ Make sure that browsers enforce the Same Origin Policy
- ○ Always set cookies using the "secure" flag
- ○ Use HTML escaping

(f) (12 points) Suppose that the Acme Browser ensures that the URL displayed at the top of a given window (in the "address bar") always accurately reflects the URL of the page the browser is displaying in that window. It is not possible for any script to alter what's shown as the URL, nor to overlay text on top of the address bar.

Alice, a security-minded user, runs Acme browser. She loads a page from `hohum.com`. The address bar displays `http://hohum.com/`*`path`*, where *`path`* contains a bunch of text (all of which fits into the address bar, and thus is visible). Alice has no information about the trustworthiness of the `hohum.com` site.

By carefully inspecting the URL in the address bar, for which of the following Web security attacks can Alice *at least partially* **defend** herself. **Mark ALL** that apply. Do not mark an attack if Alice can only possibly *detect* that the attack happened. Only mark attacks that she can (at least sometimes) keep from happening.

- ○ CSRF
- ○ Phishing
- ○ Clickjacking
- ○ Reflected XSS

- ○ SQL Injection
- ○ Stored XSS

Problem 3 ***Reasoning About Memory Safety*** **(58 points)**

Consider the following code:

```
1 /* Copy n characters from src into dst starting at
2  * dst's start_at'th character.
3  */
4 void copy_at(char *dst, char *src, int n, int start_at) {
5     for (int i = 0; i < n; i++) {
6         dst[i + start_at] = src[i];
7     }
8 }
```

(a) (12 points) Write down a *precondition* that must hold at line 6 to ensure memory safety.

(b) (16 points) Write down a *precondition* that must hold when the function is called to ensure memory safety. As usual, your precondition should not unduly constrain the operation of the function.

(c) (15 points) Write down an *invariant* that always holds just **before** line 6. You can assume that the precondition you specified in part (b) is true when the function is called. For simplicity, you can omit from your invariant any terms that appear in the precondition from part (b) that will be true throughout the execution of the function.

Your invariant should allow you to establish that the precondition you stated in part (a) will then also be true.

(d) (15 points) Write down an *invariant* that always holds just **after** line 6 executes (but before the loop iterates). The same as for part (c), you can omit terms from part (b)'s precondition if they will be true throughout the execution of the function.

Your invariant should allow you to establish that your invariant in part (c) will always hold when the loop iterates.

Problem 4 ***Browser Security*** **(56 points)**

Neo has decided to build a new web browser, BerkBrowser, which mimics the design of the Chromium web browser. BerkBrowser works by splitting the browser into two separate processes:

1. A *renderer* process, which is in charge of processing a website's code and resources (HTML, CSS, Javascript, images, videos) to generate the webpage's DOM and to then display the webpage's content to the user. It is also responsible for enforcing the Same Origin Policy (SOP).
2. A *kernel* process, which is in charge of managing the browser's persistent state (cookies, bookmarks, passwords, downloads) and mediating access to the user's local file system (e.g., downloads and uploads).

When a user visits a website using BerkBrowser, the renderer process parses and displays the webpage to the user. As the website's code makes various requests, the renderer performs SOP checks and allows/denies the actions as appropriate. Under this architecture, the renderer process cannot access the user's local filesystem, and must communicate with the kernel process via a small and bug-free API in order to access files on the user's machine. If a website wants the user to upload a file, the renderer will send an API call to the kernel process, which will display a dialogue window where the user can either choose to close the window (not upload anything), or select a file to upload. If a user needs to download a file from a website, the renderer will send an API call to the kernel process, which will display another dialogue window to the user that asks if the user would like to accept or reject the download. If the user accepts, the file is downloaded into the browser's Downloads folder. If the user rejects, the file download is blocked.

For parts A and B, mark your multiple choice answer and then write a **one-sentence explanation** in the space below each question.

For part C, write your answer in the space below the question; keep your response ≤ 4 sentences.

(a) (16 points) **Mark ONE** of the following security principles that **best** describes the BerkBrowser's architecture.

- ○ Consider Human Factors
- ○ Detect If You Can't Prevent
- ○ Don't Rely On Security Through Obscurity
- ○ Least Privilege Principle
- ○ Use Fail-Safe Defaults

(b) (20 points) **Mark ALL** of the following web attacks that this architecture is primarily designed to mitigate.

- ○ XSS Attacks
- ○ SQL Injection
- ○ Phishing
- ○ Drive-by Malware
- ○ CSRF

(c) (20 points) Suppose that Bob is using the BerkBrowser to surf the web. He visits his banking website, makes some transactions, but does not log out afterwards, so his cookie does not expire. He then navigates his browser to his favorite news website, but accidentally clicks on an ad that takes him to `evil.com`. This malicious website manages to exploit a vulnerability in the renderer process that allows `evil.com` to completely compromise and control the renderer, but not the kernel process.

Can the malicious website cause transactions to occur from Bob's bank account? If your answer is **Yes**, briefly describe how the attack would work. If your answer is **No**, explain why BerkBrowser's architecture prevents this attack.

○ **Yes** ○ **No**

Explanation:

Problem 5 ***I don't think you heard me . . .*** **(60 points)**

Assume the code below has been compiled to use randomized stack canaries, and is run with data execution prevention (e.g., DEP), and ASLR applied to the stack segment. ASLR is not applied to other memory regions.

How might an attacker exploit a vulnerability in this code to execute the command "/bin/rm -R /home/enemy/*", deleting all of the files of user "enemy"?

```
/* DEP will be enabled! :o */
void run(char* cmd) {
  system(cmd);
}

void print_twice_the_fun(char* x) {
  printf("%s %s\n", x, x);
}

int main() {
  // random unpredictable stack canary will be used!
  char first[32];
  void (*printfn)(char*);
  char second[4];

  printfn = &print_twice_the_fun;

  gets(first);
  gets(second);

  printfn(first);
}
```

Use the following assumptions:

1. The server is on an IA-32 platform with 4-byte words (recall it's also little endian).
2. The stack is aligned at word granularity.
3. Local variables of each function are placed on the stack in the order they appear in the source code.
4. The address of the first instruction of the *run* function is 0x11111110.
5. The address of the first instruction of the *print_twice_the_fun* function is 0x11111250.
6. The address of the first instruction of the *main* function is 0x11111500.
7. A randomized stack canary protects the *main* function's RIP.
8. Data execution prevention is enabled.
9. ASLR is enabled for the stack segment.

Answer the following:

(a) (30 points) For each defense mechanism below, describe why the code is still vulnerable even using this defense.

1. Randomized stack canary

2. DEP

3. ASLR on the stack

(b) (30 points) Give inputs (for both the first and second calls to `gets`) that an attacker could provide corresponding to a successful attack. You should indicate any hex characters such as 0xff (i.e., a single byte with value 255) by writing them between vertical bars, for example: "|0xff|". So if you want to indicate the attacker inputing 3 'A's followed by two 0x8d characters, you would write "AAA|0x8d|0x8d|". You do not need to indicate the newline that terminates each line.

i. First input:

ii. Second input: