Popa & Wagner Spring 2016 — CS 161 Computer Security — Midterm 1

Problem 1 ***True or False, and Fill-me-in*** **(12 points)**

In parts (a) and (b), circle True or False. In parts (c), fill in the blank. Do not justify your answer.

(a) TRUE or FALSE: A problem with iframes is that if a user visits an attacker's website, that website could load a bank website inside <iframe> tags and read sensitive data from this website.

> **Solution:** Explanation: The same-origin policy prevents this.

(b) TRUE or FALSE: A site that implements and requires a hidden validation token in a form value for requests, in addition to authentication cookies, but is vulnerable to XSS attacks, is safe from CSRF attacks.

> **Solution:** Explanation: You can use a XSS attack to learn the CSRF token and then mount a CSRF attack. See also the solution to problem 3(b).

(c) Consider the following setup: an operating system has access to sensitive files, a sandbox runs on top of the operating system, and a potentially-compromised application runs inside the sandbox. The sandbox does not allow the application to access any local files.

In preventing an attacker from gaining access to the sensitive files, the minimum trusted code base is [the sandbox and operating system.]

> **Solution:** Explanation: The sandbox has to be trusted, since it is responsible for restricting the execution of the application. The operating system also has to be trusted, since the filesystem code is part of the OS and any bugs in the filesystem could allow a malicious application to access sensitive files.

(d) An attacker tries to mount an XSS attack on a forum application. He inserts the string below into a comments form and clicks submit.

Type your comment here:

```
<script> document.write("You were attacked!"); </script>
```

SUBMIT

The web server correctly escapes all inputs it receives using HTML escaping, then

it stores them in a database. Alice comes to this forum and visits a page where she can see every comment including this one.

Write down the exact string Alice will see in her browser when she views the attacker's comment.

Solution: `<script> document.write("You were attacked!");</script>`

Explanation: After escaping, this will saved in the database and included in the HTML response as

`&lt;script&gt; document.write(&quot;You were attacked!&quot;);&lt;/script&gt;`

When the browser displays that on the screen, it will turn the `&lt;` into `<`, etc., *without* executing it.

Problem 2 ***Multiple choice*** **(12 points)**

(a) When you pay for something online using PayPal, the PayPal checkout form that shows the price and asks for your PayPal login always appears on its own page, never embedded in an iframe on the seller's checkout page. What threat is this defending against? Circle the best answer.

XSS	Integer overflow
CSRF	SQL injection
[Clickjacking]	Same-origin policy
Buffer overruns	Drive-by malware
None of the above	

(b) NinjaCourses.com is written in Python. Sheryl decides to rewrite it in C, ensuring her re-implementation behaves the same. Assuming Sheryl tries to make her C implementation have the same functionality as the Python implementation, what new security threats might Sheryl's C code face that aren't equally applicable to the original Python code? Circle all that apply.

XSS	CSRF
Clickjacking	[Buffer overruns]
Phishing	SQL injection
Separation of responsibility	Two-factor authentication
None of the above	

(c) After finishing her C code, Sheryl does CS 161's project 1 and learns for the first time about the ret2esp technique. Which of the following provides a complete defense against ret2esp attacks? Circle all that apply.

[x] Non-executable Stack	Referer validation
Content Security Policy (CSP)	Same-origin policy
[x] Memory-safe programming languages	Least privilege
Prepared statements	Two-factor authentication
None of the above	

Solution: Explanation: A non-executable stack prevents ret2esp, since ret2esp involves executing a `jmp *esp` instruction, which involves jumping into the stack. If the stack is non-executable, jumping there will fail (cause a segmentation fault). Memory-safe programming languages also prevent it, because they prevent overflowing buffers in the first place and thus there's no way to overwrite the return address to point to a `jmp *esp` instruction.

Problem 3 ***Web security*** **(15 points)**

Patsy-Bank learned about the CSRF flaw on their site described in discussion section, and they hired a security consultant who helped them fix it by adding a random CSRF token to the sensitive `/transfer` request. A valid request now looks like:

```
https://patsy-bank.com/transfer?to=bob&amount=10&token=<random>
```

The CSRF token is chosen randomly, separately for each user.

Not one to give up easily, Mallory starts looking at the welcome page. She loads the following URL in her browser:

```
https://patsy-bank.com/welcome?name=<script>alert("Jackpot!");</script>
```

When this page loaded, Mallory saw an alert pop up that says "Jackpot!". She smiles, knowing she can now force other bank customers to send her money.

(a) What kind of attack is the welcome page vulnerable to? Provide the name of the category of attack.

Solution: Reflected XSS

(b) Mallory plans to use this vulnerability to bypass the CSRF token defense. She'll replace the `alert("Jackpot!");` with some carefully chosen JavaScript. What should her JavaScript do? (Describe using at most 1–2 sentences.)

Solution: Load a payment form, extract the CSRF token, and then submit a transfer request with that CSRF token.

Or: Load a payment form, extract the CSRF token, and send it to Mallory.

(c) If Patsy-Bank added frame-busting code to the welcome page, would that stop this attack? Circle yes or no.

YES [NO]

(d) Mallory wants to attack Bob, a customer of Patsy-Bank. Name one way that Mallory could try to get Bob to click on a link she constructed.

Solution: Send him an email with this link (making it look like a link to somewhere interesting). Post the link on a forum he visits. Set up a website that Bob will visit, and have the website open that link in an iframe. Send Bob a text message or a message in Facebook with the link.

(There were many possible answers. You only needed to list one.)

Problem 4 ***Evaluating defenses*** **(16 points)**

Michael is considering several ideas for defending against some of the threats we've seen in this class. For each of the following scenarios, decide whether Michael's proposed defense is secure or not (i.e., whether it is effective at defending against the named threat). Circle "Secure" or "Insecure"; if you circle "Insecure", also describe in a sentence or less how an attacker could defeat Michael's defense.

(a) Michael notices that a bank is vulnerable to CSRF attacks. To defend against CSRF, Michael proposes that the bank add a X-Frame-Options header to every page.

SECURE [INSECURE]

Solution: Attack: It does nothing to stop it.

(b) To prevent buffer overruns, Michael proposes to allocate all buffers on the heap. In other words, even if a buffer is declared as a local variable, Michael proposes that the compiler should insert a call to `malloc()` to allocate space for the buffer, and free the buffer when the function returns.

SECURE [INSECURE]

Solution: Attack: Overwrite stuff in the heap to overwrite a function pointer. (Many other attacks are possible as well.)

(c) To prevent phishers from using homeographic attacks using internationalized characters (e.g., `paypal.com`, where the first `p` is in Cyrillic), Michael proposes that browsers should allow only the characters `A-Za-z0-9_-.` to appear in the domain name of a URL; if the domain name contains any other characters, the browser should refuse to load the URL.

SECURE INSECURE

Solution: We also accept INSECURE with the attack: use a domain name that exploits the similarities between vv/w or l/i/1 or O/0.

(d) To prevent session fixation attacks, Michael proposes to use 256-bit session IDs, where the first 128 bits are chosen randomly for each session, and the last 128 bits are a secret value that is specific to the server but the same for all sessions with that server.

(As a reminder, session fixation attacks apply to sites that accept a session ID in the URL and set a session cookie with the same value.)

SECURE INSECURE

Solution: Attack: Mallory can still take her own session ID, put it in a URL, and get Victoria the victim to fetch that URL.

Problem 5 ***Reasoning about memory safety*** **(20 points)**

Consider the following C code.

```
/* Requires: ??? */
void shuffle(int a[], int b[], size_t m, size_t n) {
    for (size_t i = 0; i < m; i++) {
        int tmp = a[i];
        a[i] = b[n-i];
        b[n-i] = tmp;
    }
}
```

For each of the following candidate preconditions in parts (a)–(d), answer whether that precondition is sufficient to ensure that `shuffle()` will be memory-safe. If it is not sufficient, also specify an example of an input that would satisfy the precondition but could cause memory-unsafe behavior.

(a) `a != NULL && b != NULL`

SUFFICIENT NOT SUFFICIENT

Solution: `a = {0}, b = {0}, m = 2, n = 1`

(b) `a != NULL && b != NULL && m == 0 && n == 0`

SUFFICIENT NOT SUFFICIENT

(c) `a != NULL && b != NULL && m == n`

SUFFICIENT NOT SUFFICIENT

Solution: `a = {0}, b = {0}, m = 2, n = 2`

(d) `a != NULL && b != NULL && m < size(a) && n < size(b)`

SUFFICIENT NOT SUFFICIENT

Solution: `a = {0,1,2}, b = {0}, m = 2, n = 0`

(e) Suggest a better precondition. Your precondition should be sufficient to ensure that `shuffle()` is memory-safe, and be as general as possible. Don't worry about what `shuffle()` is trying to accomplish; it just needs to be memory-safe.

Solution: `a != NULL && b != NULL && m <= size(a) && n < size(b) && m <= n+1`

Problem 6 ***Secure design and implementation*** **(14 points)**

(a) Blackbeard decides to set up his own course scheduling site, PirateCourses.com. To manage this machine remotely, he installs a SSH server on it. He discovers that the version of SSH he is running is vulnerable to a buffer overflow exploit, but he's too lazy to upgrade. Instead, he configures the SSH daemon to run on a custom port, 5467. However, when he logs in a week later, he finds all his data gone. What security principle did he ignore?

Solution: Don't rely on security through obscurity.

(b) Blackbeard fixes this, and his website is now flourishing. He hires Alice to implement tagging, so users can add tags to courses and search for courses by tag. For instance, when a user searches for the tag `easy_exams`, the following page is loaded:

`http://www.piratecourses.com/?query=easy_exams`

That page contains a list of all courses with the `easy_exams` tag.

Unfortunately, Alice doesn't know how to write secure code. Crabby the crab searches for the tag `courses that are easy to'; drop table users --` and suddenly none of the PirateCourses users can log in any more. What kind of vulnerability was this?

Solution: SQL injection

(c) Name two different reasonable techniques that Alice could have used that each would have prevented the vulnerability in part (b).

Solution: Prepared statements. Escaping. Input validation (whitelisting). Pick any two of those three.

Problem 7 ***Snooping on your friends*** **(11 points)**

Alice sets up a private wiki page for her friends, running on scripts.berkeley.edu, at

https://scripts.berkeley.edu/∼alice/wiki

Alice has a session cookie in her browser for the wiki page, which is still valid for two weeks from now. When she visits the wiki site, she is immediately logged in and can read the content.

Eve doesn't have an account on Alice's wiki, but is dying to read what Alice and her friends are saying on that wiki page. Eve has her own web site running on scripts.berkeley.edu, at

https://scripts.berkeley.edu/∼eve/

Consider that Eve manages to get Alice to visit Eve's page. How can Eve get a copy of Alice's wiki page?

Note that Alice is security savvy and she employs the following security techniques:

1. Her wiki employs CSRF protection correctly so a CSRF attack won't work.
2. The session cookie is httpOnly so JavaScript cannot access it.
3. Alice is not susceptible to phishing attacks. She checks the URL carefully wherever she goes (including the unicode!). When she visits a site that is not hers, including Eve's site, she does not provide her password or other sensitive information.
4. Alice will not click on anything in Eve's page other than visiting the page.
5. She scopes her cookies with domain=scripts.berkeley.edu and a path corresponding to ∼alice/wiki, so that other users' servers on scripts.berkeley.edu won't receive her cookies.

Describe Eve's attack here:

Solution: Eve's page will load `https://scripts.berkeley.edu/~alice/wiki` in an iframe. It will also include JavaScript that uses the JavaScript DOM API's to read the contents of the iframe (this is allowed by the same-origin policy, since it's the same origin) and then send that to a website controlled by Eve.