CS 361S - Network Security and Privacy
Spring 2014

MIDTERM

March 4, 2014

DO NOT OPEN UNTIL INSTRUCTED

YOUR NAME: ______________________________

Collaboration policy

No collaboration is permitted on this midterm. Any cheating (*e.g.*, submitting another person's work as your own, or permitting your work to be copied) will automatically result in a failing grade. The UTCS code of conduct can be found at `http://www.cs.utexas.edu/undergraduate-program/code-conduct`

Midterm (100 points)

Problem 1 (20 points)

Circle only one of the choices **(4 points each)**.

1. **TRUE** **FALSE** SHA-256 (with 256-bit output) is more resistant to attacks based on the birthday paradox than SHA-1 (with 160-bit output).
2. **TRUE** **FALSE** The same origin policy in modern Web browsers says that a script can access DOM objects only from the same domain, protocol, and port as the script itself.
3. **TRUE** **FALSE** The browser's same origin policy prevents a network attacker from injecting scripts into content received over HTTP from trusted websites.
4. **TRUE** **FALSE** If the key is truly random, as long as the plaintext, and never re-used, the one-time pad provides perfect message integrity.
5. **TRUE** **FALSE** If the encryption scheme is secure against chosen-plaintext attacks, an eavesdropper cannot learn anything about the plaintext by looking at the ciphertext, but can still tell if two ciphertexts encrypt the same message by comparing them for equality.

Problem 2 (6 points)

Does storing Hash(username || password)—that is, hash of the username concatenated with the user's password—on the server better defend against an attacker who breaks into the server and tries to crack passwords than just storing Hash(password)?

Problem 3

`BlogMolvania.com` is a popular Molvanîan blogging service. It lets users create new blog posts and read, edit, or delete other users' posts (if properly authorized). BlogMolvania wants to ensure that access to any of the three operations (read, edit, delete) on an existing blog post is allowed only if the user possesses the right URL.

Anyone can create a new post. Each post is assigned an identifier `post-id`. Identifiers are generated sequentially using a global counter. After creating a post, the author gets three URLs—for reading, editing, and deleting, respectively. These URLs look like this:

```
https://BlogMolvania.com/read/post-id
https://BlogMolvania.com/edit/post-id
https://BlogMolvania.com/delete/post-id
```

The author of the post is free to share these URLs with other users. Sharing the URL identifies the post and simultaneously allows the recipient to perform the indicated operation on this post.

Problem 3a (4 points)

Explain how a malicious user can gain unauthorized access to another user's blog post.

Problem 3b (4 points)

Explain how a malicious user can perform actions that he is not authorized to do. For example, after receiving a URL intended only for reading someone's post, how can he edit or delete this post?

Problem 3c (6 points)

How should BlogMolvania generate blog-post identifiers and URLs to prevent these attacks?

Problem 4 (6 points)

Can defenses against cross-site request forgery help foil clickjacking attacks (eg, where the user is tricked into submitting a form to a website)? Explain why or why not.

Problem 5

Problem 5a (4 points)

What are httpOnly cookies?

Problem 5b (6 points)

What attack are httpOnly cookies intended to prevent? Give an example attack that works if the site uses normal cookies but does not work with httpOnly cookies.

Problem 5c (6 points)

Give an example attack of the same broad category as in Problem 5b which is not prevented by httpOnly cookies.

Problem 6

Problem 6a (4 points)

What is a third-party cookie?

Problem 6b (6 points)

How do advertising networks use third-party cookies to learn which webpages you visited? Why does this not violate the same origin policy?

Problem 7

Consider the following PHP script for logging into a website:

```
$username = $_GET[user];
$password = $_GET[pwd];
$sql = "SELECT * FROM usertable
        WHERE username= '$username' AND password = '$password' ";
$result = $db->query($sql);
if ($result->num_rows > 0) { /* successful login */ }
else                       { /* login failed */ }
```

Problem 7a (5 points)

Give an example of a username that will successfully subvert the above authentication code.

Problem 7b (5 points)

The PHP function `addslashes` adds a slash before every quote. For example, `addslashes(x'y)` outputs the string `x\'y`.

Suppose user's input is sanitized as follows:

```
$username = addslashes($_GET[user]);
$password = addslashes($_GET[pwd]);
```

In the Chinese, Korean, and Japanese unicode character sets, some characters are encoded as single bytes, while others are double bytes. For example, the database interprets `0x5C` as \, `0x27` as `'`, `0x5C27` as `\'`, but `0xBF5C` is interpreted as a single Chinese character.

Give an example of a username that will successfully subvert the above authentication code even if the input is sanitized using `addslashes`.

Problem 7c (5 points)

How should `addslashes` be implemented to prevent SQL injection attacks?

Problem 8 (6 points)

Intuitively it seems like an encryption scheme is secure if an adversary can't decrypt the ciphertext without knowledge of the key. Why do we bother with the "encryption game" instead of simply defining security in terms of this property?

Problem 9 (7 points)

Consider a variant of Kerberos in which the key distribution center (KDC) is combined with the Ticket-Granting Service (TGS). Instead of requesting the ticket-granting ticket (TGT) from KDC and sending it to the TGS, in this variant the TGT is generated by the client's workstation and encrypted under the client's master key instead of the Ticket-Granting Service's (TGS) key. As in standard Kerberos, assume that the joint KDC/TGS knows every client's master key.

Let K_A be Alice's master key derived from her password and let S be a long, random session key freshly generated by Alice's workstation. Alice's TGT looks like this:

$$TGT_A = enc_{K_A}(S, \text{“Alice”}, IPaddr_A, time_A, lifetime)$$

When Alice requests a ticket for service V from the TGS, she sends the following message:

$$\begin{array}{lcll} Alice & \rightarrow & TGS & \text{“I am Alice, and I want to talk to service } V\text{”}, \\ & & & TGT_A, enc_S(\text{“Alice”}, IPaddr_A, time'_A) \end{array}$$

Is the resulting scheme less secure than the original Kerberos? Explain.