Homework 3: Web Security and Passwords

Due: Thursday, February 27 @ 11:59 pm

Web Security

Problem 1: CSRF Tokens

Recall the Cross-Site Request Forgery (CSRF) attack discussed in lecture. One possible mitigation for CSRF is a family of techniques known as CSRF tokens.

Question a) One might implement such a technique via cookies. In addition to a session ID cookie, every user gets a cookie containing a random CSRF token when they log in. Then, whenever a sensitive request is made, JavaScript executing in the browser reads this cookie, and copies it into the body of the request. The website then only accepts the request as valid if the CSRF token cookie matches the CSRF token in the body of the request.

Does this CSRF prevention technique work? If so, explain why and reference the web / browser security features that are relied upon in this technique; if not, outline a CSRF attack that works while the user is visiting a malicious website.

Question b) The Hotel CIT’s reservation web interface implements CSRF tokens in a different way from part a). When the website displays a sensitive <form> to the user, it generates a CSRF token, associates the value of this token with the user’s session on the server, then loads the CSRF token into a hidden field on the form before sending it to the user. On form submit, the server checks that the submitted token from the form is the same as the token stored in the session.

Explain why this CSRF prevention technique works; that is, it prevents any other domains from making authenticated requests to the target domain. (Hint: It involves the SOP.)

Question c) It’s clear what to do if a given CSRF token was valid—simply process the form action as intended. It’s less clear what to do when the token is invalid, but here’s what The Hotel CIT’s website does: it takes the fields in the form submission (except the CSRF token) and uses them to populate a new form (which points to the same endpoint as the form submission that was just rejected) along with the correct CSRF token. That way, if the token had expired, the user won’t need to reenter all the information in the form, they can just check it for correctness and resubmit.

Why would this strategy for handling incorrect CSRF tokens be a bad idea? (Assume that there are no XSS or SQLI vulnerabilities in the website and the website has no defenses against UI Redress attacks and does not set security headers such as Content-Security-Policy.)

Problem 2: Redirect Parameters

Many sites use redirect scripts that take a URL as a parameter and send the browser to that URL. Sites do this instead of simply linking directly to a particular site because it allows them to collect analytics on which
external links users of their site are clicking on. That is, instead of linking directly to external websites, they provide links on the same domain where the external link is exposed via a “redirect parameter”.

The CS166 course staff is developing a new startup, http://pizza.brown, a platform where students ask questions about course assignments. Staff members can link to external websites on pizza.brown (i.e., Zachary’s pizza.brown page links to http://cs.brown.edu/~zespirit/), and when they do, pizza.brown uses the /redirect endpoint to redirect users to the external website defined in the url parameter:


Question a) Explain why this kind of “open redirect” poses a significant security risk to pizza.brown users. (Hint: Consider phishing attacks.)

Question b) One possible mitigation to the attack in part (a) is for the pizza.brown developers to maintain a safelist of all allowed redirect targets. Under this system, if the url is not on the safelist, the redirect endpoint returns an error. Why might this be an unreasonable form of security?

Question c) Design a new redirect system that prevents the attack from part (a), and explain why it works. Your protocol should satisfy the following properties (and you should justify why it does):

P1. It must be possible for the website to authorize new redirect links. As discussed in part (b), your protocol should not rely on maintaining a set of allowed links (e.g. in a database), but the website must be able to tell whether or not a given redirect URL has been authorized by the website.

P2. Users cannot craft redirect links to non-authorized domains.

P3. There must be some way for the website to “revoke” access to a particular redirect URL, though the permission revocation does not have to be instantaneous.

Problem 3: Browser Fingerprinting

While many sites use cookies to track users and store information about their browsing habits, there exist more sophisticated techniques that are not as easy to detect and counter by the user. One technique, browser fingerprinting, allows sites to target ads and track users without ever storing any cookies on the client.

Question a) By visiting a website, what are some possible ways your browser could reveal information about you that could be used to uniquely fingerprint your browser? Name at least six distinct features. You might find https://amiunique.org/ helpful for this question.

Question b) A common tracking technique is called a tracking pixel, where a 1 × 1 pixel image is loaded onto your webpage from a different server. How could this image alone be used to track users across websites that load a tracking pixel from a particular server?

Question c) Consider the tracking pixel system described in part (b), and suppose this tracking pixel is embedded on bank.com. bank.com uses GET requests for every web endpoint (balance checks, money withdrawals, money transfers, etc.).

(i) How might the inclusion of a tracking pixel on bank.com allow the server hosting the tracking pixel to discover money transfer transactions, bank account identifiers, etc.?

(ii) Does your answer change in any way if bank.com uses POST requests instead? Explain.

(iii) Does your answer change if bank.com uses https connections while the server hosting the tracking pixel uses http connections? Explain.

Question d) Servers can use ETags to uniquely identify a particular version of a resource for the purposes of caching. When a resource is cached by the browser, any associated ETag is cached as well. When a future request is made for the cached resource, the browser will include the ETag, which allows the server to identify the version of the resource in the browser’s cache. It can then respond with the current version or, if the cached version is up-to-date, simply respond with response code 304 (“Not Modified”), indicating to the browser that it should simply use
the version in its cache. Explain how a server could use ETags to tell that two HTTP requests came from the same user, even if that user had deleted all their cookies.

Disclosure Ethics

Problem 4: Vulnerability Disclosure

When vulnerabilities are discovered in real-world systems, whose responsibility is it to disclose the vulnerability, and within what time frame? [Vulnerability_Discovery_Cheat_Sheet](https://owasp.org/www-project-cheat-sheets/cheatsheets/Vulnerability_Disclosure_Cheat_Sheet) serves as a good introduction to vulnerability disclosure in practice, including timelines, policies, and typical processes. As a case study, consider the 2019 vulnerability in Zoom, the video conferencing application, discovered by security researcher Jonathan Leitschuh:

- Leitschuh’s account of disclosing the vulnerability: [https://medium.com/p/ac75c83f4ef5/](https://medium.com/p/ac75c83f4ef5/)

The following questions are open-ended and will be graded for thought and justification. Please reference and/or quote specific arguments from the readings in your answers.

**Question a)** What are the differences between responsible disclosure, full disclosure, and private disclosure? Describe one advantage and disadvantage of each.

**Question b)** Suppose you were a security researcher who has just discovered a CSRF attack on the Airbnb iOS app. What are the steps for reporting this vulnerability? What information should you include in your report, and what is the average bounty you could expect?

**Question c)** Early in the non-disclosure period, Zoom offered Lietschuh a financial reward for reporting the vulnerability, on the condition that he would not be able to publicly disclose the vulnerability even after it was patched. Lietschuh declined this offer. If you were in Leitschuh’s shoes, would you have taken the offer? Should such offers be allowed by industry vulnerability disclosure policies (VDPs)?

**Question d)** Is 90 days an appropriate amount of time to allow vendors to patch vulnerabilities before full disclosure? Should this be shortened or lengthened, and if so, why? Should small startups have a different deadline than large tech companies?

**Question e)** Security researchers have received legal threats from industry vendors asking them to stop publicly disclosing vulnerabilities and proof of concept code that exploits them. In your opinion, should there be legal protections for security researchers who choose to publicly disclose vulnerabilities? Why or why not?

Authentication

Problem 5: Salting with Usernames

In the Authentication lecture, we walked through several attempts to securely store passwords in databases using various combinations of hashing and salting and concluded that a “per-user salting” strategy was the most secure option. One key advantage of per-user salting is that, since every user has a different salt, identical passwords will not have identical hashes in the database.

Some websites achieve per-user salting by using usernames as salts. In theory, this achieves the same thing as using randomly generated salts because every user on the website will have a different salt, and therefore if two users on the website share the same password, they will not have identical hashes. Furthermore, if we randomly generate salts, there’s a small chance that two users on the website will end up having the same salt for their password—since usernames are unique we’ll never have two users with the same salt under the username-as-salt scheme. However, in practice, using usernames as the salt is a bad idea. Why?
Problem 6: Better Passwords via a Browser Extension

PwdHash ([https://crypto.stanford.edu/PwdHash/](https://crypto.stanford.edu/PwdHash/)) is a browser extension that transparently converts a user’s password into domain-specific values. For example, if a user visits bank.com and types in the plaintext password iampassword, when the user submits the login form, PwdHash instead causes the browser to send hash(iampassword ++ bank.com) as the password, where ++ is the string concatenation operator and hash is some cryptographic hash function. (PwdHash knows the domain that the user is visiting on because, as a browser extension, it has direct access to the URLs a user visits.)

**Question a)** Julia, a user with the PwdHash extension installed in her browser, likes using the same password “running” for every single website. Does PwdHash protect her password from being broken by a dictionary attack? Explain.

**Question b)** Julia uses bank.com for her online banking operations. Suppose bank.com’s database is stolen. Does PwdHash protect Julia’s password from being cracked by a brute-force attack? Explain.

**Question c)** Suppose Julia visits a fake website set up by Max, a web attacker that controls a website that looks identical to bank.com but is actually hosted at bank.co. However, Julia doesn’t notice the phishing scam and types in her real password for bank.com into her browser. Does PwdHash protect Julia’s password from being stolen and used by Max? Explain.

**Question d)** Suppose Julia access bank.com over HTTP, and consider an attacker, Casey, who can eavesdrop on Julia’s internet connection. Does PwdHash protect Julia’s password from being stolen and used by Casey? Explain.

**Question e)** Does PwdHash increase or decrease the overall password complexity of its users’ passwords? Explain. (Make sure to explicitly state any assumptions you need to make in this question.)

**Question f)** Are there any significant usability concerns for any user of PwdHash? Explain.