**Question 1 - multiple choice, shuffle**

What is one difference between an HTTP GET and an HTTP POST request?

A: Only POST requests can encode parameters in the URL

Feedback: Both POST and GET requests can have parameters in the URL

\*B: Only POST requests may include parameter data in the request body

Feedback: POST requests are often issued for web forms whose content is included in the request data

C: Only GET requests are subject to the same-origin policy

Feedback: The same-origin policy applies to Javascript programs, not HTTP requests

D: Only GET requests use the REFERER header

Feedback: All requests can use the REFERER header

**Question 2 - multiple choice, shuffle**

Which of the following is true about static and dynamic web content?

A: Static content may be re-generated with each request

Feedback: Static content is unchanging

B: Static pages may include PHP programs, which execute at the browser

Feedback: PHP programs execute at the server to produce dynamic content; they are not part of static pages

C: Javascript programs embedded in HTML pages are run server-side to produce dynamic content

Feedback: Javascript programs embedded in HTML are run at the browser, at the client, not server

\*D: The server often produces dynamic content based on the contents of the database

Feedback: Dynamic content is regenerated with each request, often including database-resident content

**Question 3 - multiple choice, shuffle**

SQL injection exploits a bug in what interaction of a web application?

A: Client to server

Feedback: The client inputs data that the server passes on to the database, but SQL injection is not exploiting a bug involved in the client-server interaction

B: Server to client

Feedback: The bug is not in the what the server presents to the client, but with how the server acquired that content from the database

\*C: Server to database

Feedback: The bug is in how SQL queries are constructed at the server to be sent on to the database

D: Network to server

Feedback: The exploit often arrives via network request to the server, but the network request itself is not what's buggy

**Question 4 - multiple choice, shuffle**

SQL injection often allows an attacker to do which of the following?

\*A: Access information he shouldn't

Feedback: SQL injection often bypasses WHERE clauses, which refine SQL queries, and therefore the injection reveals more information than it normally would

B: Overrun a buffer to smash the stack

Feedback: SQL injection involves the way an SQL command is constructed, not an overrun of a buffer

C: Cause memory to be used after it's freed

Feedback: SQL injection involves the way an SQL command is constructed, not a memory management error

D: All of the above

Feedback: Stack smashing and use-after-free bugs are not related to SQL injection, which is a bug in how a SQL command is constructed

**Question 5 - multiple choice, no shuffle**

If you had to summarize the *key* (most specific) programming failure with SQL injection, it would be:

\*A: Confusing data with code

Feedback: Data entered by an untrusted user is formatted so as to be interpreted as SQL code, which can be used to work around the application's intention to treat it as data

B: Bypassing authentication

Feedback: Bypassing authentication is sometimes a consequence of SQL injection, but it is not the defining aspect of it

C: Trusting without verifying

Feedback: Trusting user data without verifying it is a general problem, not specific to SQL injection and thus not its key issue (but worth partial credit)

D: Circumventing the same origin policy

Feedback: Same-origin relates to the privileges given to Javascript code

**Question 6 - multiple choice, no shuffle**

What is *escaping* an example of?

A: Blacklisting

Feedback: Escaping is a transformation of text, while blacklisting rejects text based on whether it includes blacklisted elements

\*B: Sanitization

Feedback: Sanitization is a transformation of text that removes potentially harmful elements, and escaping does this when content could contain HTML markup

C: Checking

Feedback: Checking that an input is acceptable or not is different from escaping, which actually changes the input to become acceptable.

D: Whitelisting

Feedback: Whilelisting allows input if it conforms to the whitelisted specification, but does not transform it, which escaping does

**Question 7 - multiple choice, shuffle**

Suppose a web application implements authentication by constructing an SQL query from HTML from data using PHP's *prepared statements*. What would happen if an attacker entered $$\color{red}{\verb|FRANK' OR 1=1; -- |}$$ in the web form's user field?

\*A: The application will try to authenticate a user whose name is $$\color{red}{\verb|FRANK' OR 1=1; -- |}$$

Feedback: The text that is entered will be treated as data, and not confused as code

B: The text will modify the structure of the SQL query and possibly bypass authentication

Feedback: Prepared statements prevent the treatment of data as code, and thus will not affect the SQL query's structure

C: The text will be confused as the password and authentication will probably fail

Feedback: The content of the data will not affect how it is treated---if the form is normally used for a username, then this data will be too

D: The text will corrupt the query structure and the database will view it as a syntax error

Feedback: Prepared statements prevent the treatment of data as code, and thus will not affect the SQL query's structure

**Question 8 - multiple choice, shuffle**

Why is it undesirable to implement session identifiers using (only) hidden form fields?

A: These fields are easily modified by the user

Feedback: While true, such modification is not inherently problematic; e.g., cookies could be modified as well

\*B: The session ID is forgotten when the browser window is closed

Feedback: This adds inconvenience to the user, since closing the window necessitates logging in again, and complicates the construction of the site to always pass around the hidden field

C: Such fields cannot contain binary data

Feedback: Textual (possibly encoded, e.g., with base64) data is fine for session identifiers

D: Such fields cannot include timeout information

Feedback: The time can be encoded in the session identifier and timeouts enforced by the server

**Question 9 - multiple choice, shuffle**

Suppose a browser submits a GET request to URL $$\color{red}{\verb|http://www.mybank.com/accountinfo|}$$ on 20 February 2015. Which of the following cookies, if already stored at the browser, would be sent with the request?

A: $$\color{red}{\verb|edition=us; expires=Thu, 19-Feb-2015; path=/accountinfo/prefs; domain=.mybank.com|}$$

Feedback: This cookie is not sent because it has expired

B: $$\color{red}{\verb|prefs=small:blue:refresh; expires=Sat, 1-Aug-2015; path=/specialoffers/; domain=.mybank.com|}$$

Feedback: This cookie is not sent because the path is not a prefix of the path given in the URL

C: $$\color{red}{\verb|lang=us-english; expires=Sat, 1-Aug-2015; path=/accountinfo/; domain=.fidelity.com|}$$

Feedback: This cookie is not sent because the domain name does not match

\*D: $$\color{red}{\verb|sessid=14FEB15; expires=Sat, 28-Feb-2015; path=/; domain=.mybank.com|}$$

Feedback: This cookie has not timed out, has a path that is a prefix of the given path, and references the proper domain suffix

**Question 9 - multiple choice, variation 1, shuffle**

Suppose a browser submits a GET request to URL $$\color{red}{\verb|http://www.mybank.com/accountinfo|}$$ on 20 February 2015. Which of the following cookies, if already stored at the browser, would be sent with the request?

A: $$\color{red}{\verb|lang=us-english; expires=Sat, 1-Aug-2015; path=/accountinfo/; domain=.fidelity.com|}$$

Feedback: This cookie is not sent because the domain name does not match

B: $$\color{red}{\verb|edition=us; expires=Thu, 19-Feb-2015; path=/accountinfo/prefs; domain=.mybank.com|}$$

Feedback: This cookie is not sent because it has expired

C: $$\color{red}{\verb|edition=us; expires=Wed, 18-Feb-2015; path=/; domain=.mybank.com|}$$

Feedback: This cookie is not sent because it has expired

\*D: $$\color{red}{\verb|sessid=ABCDEFG; expires=Sat, 21-Feb-2015; path=/; domain=.mybank.com|}$$

Feedback: This cookie has not timed out, has a path that is a prefix of the given path, and references the proper domain suffix

**Question 10 - checkbox, shuffle, partial credit**

(3 pts) Which of the following are ways that session cookies could be stolen or forged?

\*A: Reading a cookie from an unencrypted web request

Feedback: An adversary that can see web requests (e.g., in an Internet cafe) can steal cookies from those requests

\*B: Compromising the browser or server

Feedback: Injected code could exfiltrate cookies used for all users/sites

\*C: Predicting the cookie's structure and reconstructing it

Feedback: Knowing how cookies are constructed, and knowing features of the user, site, etc. permits creation of the cookie

**Question 10 - checkbox, variation 1, shuffle, partial credit**

Which of the following are ways that session cookies could be stolen or forged?

\*A: Compromising the browser or server

Feedback: Injected code could exfiltrate cookies used for all users/sites

\*B: Predicting the cookie's structure and reconstructing it

Feedback: Knowing how cookies are constructed, and knowing features of the user, site, etc. permits creation of the cookie

C: Copying a cookie by keylogging

Feedback: Cookies are not entered by the user (via the keyboard), but determined by the software of the site itself

**Question 10 - checkbox, variation 2, shuffle, partial credit**

Which of the following are ways that session cookies could be stolen or forged?

\*A: Predicting the cookie's structure and reconstructing it

Feedback: Knowing how cookies are constructed, and knowing features of the user, site, etc. permits creation of the cookie

B: Stealing it from the password database

Feedback: Cookies are used for authentication, but are not passwords, and so are not stored in the password DB.

\*C: Reading a cookie from an unencrypted web request

Feedback: An adversary that can see web requests (e.g., in an Internet cafe) can steal cookies from those requests

**Question 11 - checkbox, shuffle, partial credit**

Which of the following are ways to reduce the impact of a stolen cookies?

\*A: Giving each cookie a timeout

Feedback: Timing out a cookie means it can only be misused for a limited period

B: Associate the cookie with the server's IP address

Feedback: Associating a cookie with the *client*'s IP address can help, despite false positives and false negatives, but associating it with the server's address would not

\*C: Associate the cookie with the client's IP address

Feedback: Associating a cookie with the client's IP address can prevent users not at that address from using the cookie, but note that this technique has usability problems and is not perfect (due to spoofing)

**Question 11 - checkbox, variation 1, shuffle, partial credit**

Which of the following are ways to reduce the impact of a stolen cookies?

A: Associate the cookie with the server's IP address

Feedback: Associating a cookie with the *client*'s IP address can help, despite false positives and false negatives, but associating it with the server's address would not

\*B: Changing a user's cookie from session to session

Feedback: Per-session cookies, if stolen, cannot affect future sessions

C: Prevent cookies from entering the DNS cache

Feedback: Cookies don't go in the DNS cache, DNS addresses do

**Question 11 - checkbox, variation 2, shuffle, partial credit**

Which of the following are ways to reduce the impact of a stolen cookies?

A: Prevent cookies from entering the DNS cache

Feedback: Cookies don't go in the DNS cache, DNS addresses do

\*B: Giving each cookie a timeout

Feedback: Timing out a cookie means it can only be misused for a limited period

\*C: Changing a user's cookie from session to session

Feedback: Per-session cookies, if stolen, cannot affect future sessions

**Question 12 - multiple choice, shuffle**

How can the REFERER field be used to defend against CSRF attacks?

\*A: It can be used to ensure that sensitive requests are (only) initiated by interaction with a site's own pages

Feedback: The browser fills in the referer field to be the page from which an HTTP request was generated, and thus confirms the request comes from a trusted page

B: It ensures that requests only come from authenticated users

Feedback: HTTP has nothing to do with user-level authentication (HTTPS authenticates a client with a server, but that has no impact on CSRF)

C: It can be used to check that a Javascript program is from the proper origin

Feedback: HTTP has nothing to do with locating Javascript programs

D: It can't be used reliably because it only works for dynamic content

Feedback: REFERER works for both static and dynamic content

**Question 13 - multiple choice, shuffle**

$$\color{red}{\verb|<script></script>|}$$tags in HTML pages most often identify programs written in what language?

\*A: Javascript

B: PHP

Feedback: PHP is a server-side language, not a browser-side one

C: Java

Feedback: Java code is typically identified by $$\color{red}{\verb|<applet></applet>|}$$ tags

D: C

Feedback: C code is not normally used for client-side scripting

**Question 14 - multiple choice, shuffle**

The browser implements security for Javascript programs for what reason?

\*A: Such programs may access browser-controlled resources, which include potentially sensitive data in HTML documents and cookies

B: It doesn't -- these programs are only used to render dynamic content but are otherwise not security-relevant

Feedback: Javascript programs may access sensitive cookies and page content, e.g., to display it, and thus are indeed security-relevant

C: Such programs could deny service by running forever

Feedback: While Javascript programs could run forever, browsers do not specifically defend against this

D: It doesn't -- Javascript programs run at the server so the browser can ignore them

Feedback: Javascript programs can run at the client, in the browser, e.g., when they are identified by $$\color{red}{\verb|script|}$$ tags

**Question 15 - multiple choice, shuffle**

XSS subverts what policy?

\*A: Same Origin

Feedback: XSS uploads a script from host A to site B, which serves the script and thus bestows in it the site's privileges, circumventing the same origin policy

B: Availability

Feedback: While compromising availability is one possible *consequence* of an XSS attack, the same origin policy is what is subverted

C: Whitelisting

Feedback: Whitelisting is a defensive technique, not a policy

D: Secure defaults

Feedback: There is no decision for which to establish a default choice, related to the weakness exploited by XSS

**Question 16 - multiple choice, shuffle**

What is the difference between *stored* (or persistent) XSS and *reflected* XSS?

A: Stored XSS embeds Javascript in an a URL, while reflected XSS embeds it in a mirrored site

Feedback: It's the reverse: Stored XSS works by injecting code in a site's served content, while reflected XSS injects code in a URL

B: Stored XSS works on database queries while reflected XSS works on cookies, which are received from and reflected back to the server

Feedback: XSS affects the privileges granted to executing Javascript programs, not cookies or database queries

\*C: Stored XSS works by injecting code in a site's served content, while reflected XSS injects code in a URL

D: Stored XSS is amenable to blacklisting but reflected XSS is not

Feedback: Blacklisting could be applied to either form of attack