This page contains the answers to the questions posed at the end of each chapter of the second edition.

Chapter 2 – Core Defense Mechanisms

1. In a typical application, access is handled using a trio of mechanisms relating to authentication, session management, and access control. These components are highly interdependent, and a weakness in any one of them will undermine the effectiveness of the overall access handling mechanism. For example, a defective authentication mechanism may enable an attacker to login as any user and so gain unauthorized access. If session tokens can be predicted, an attacker may be able to masquerade as any logged in user and gain access to their data. If access controls are broken, then any user may be able to directly use functionality that is supposed to be protected.

2. A session is a set of data structures held on the server, which are used to track the state of the user's interaction with the application. A session token is a unique string that the application maps to the session, and is submitted by the user to reidentify themselves across successive requests.

3. There are many situations where an application may be forced to accept data for processing that does not match a list or pattern of input that is known to be "good". For example, many people's names contain characters that can be used in various attacks. If an application wishes to allow people to register under their real names, it needs to accept input that may be malicious, and ensure that this is handled and processed in a safe manner nevertheless.

4. Defects in the any of the core mechanisms for handling access may enable you to gain unauthorized access to the administrative functionality. Further, data that you submit as a low privileged user may ultimately be displayed to administrative users, enabling you to attack them by submitting malicious data designed to compromise their session when it is viewed.

5. Yes. If it were not for Step 4, this mechanism would be robust in terms of filtering the specific items it is designed to block. However, because your input is decoded *after* the filtering steps have been performed, you can simply URL-encode selected characters in your payload to evade the filter:

   ```
   %22>%3cscript>alert(%22foo%22)</script>
   ```

   If Step 4 were performed first (or even not at all) then this bypass would not be possible.

Site Contents

- Second edition:
  - Book contents
  - Answers
  - Source code
- First edition:
  - Book contents
  - Answers
  - Source code
- Tools
- Task checklist
- Buy

Chapter 3 – Web Application Technologies

1. The `OPTIONS` method asks the server to report the HTTP methods that are available for a particular resource.

2. The `If-Modified-Since` header is used to specify the time at which the browser last received the requested resource. The `If-None-Match` header is used to specify the entity tag that the server issued with the requested resource when it was last received.

   In the two ways described, these headers are used to support caching of content within the browser, and they enable the server to instruct the browser to use a cached copy of a resource, rather than responding with the full contents of the resource if this is not necessary.

   When you are attacking an application, your browser may already have cached copies of resources that you are interested in, such as JavaScript files. By removing these two headers, you can override the browser's caching information and ensure that the server responds with a fresh copy of the resource you wish to view.

3. The `secure` flag is used to instruct the browser that the cookie should only ever be resubmitted over HTTPS connections, and never unencrypted HTTP.

4. The 301 status code tells the browser that the requested resource has moved permanently to a different location. For duration of the current browser session, if your browser needs to access the originally requested resource, it will use the location specified in the 301 response instead.

   The 302 status code tells the browser that the requested resource has moved temporarily to a different location. On the next occasion that the browser needs to access the originally requested resource, it will request this from the originally requested location.

5. The browser sends a `CONNECT` request to the proxy, specifying the destination hostname and port number as the URL within this request. If the proxy allows the request, it returns an HTTP response with a 200 status, keeps the TCP connection open, and from that point onwards acts as a pure TCP-level relay to the specified destination.

Chapter 4 – Mapping the Application

1. The filename `CookieAuth.dll` indicates that Microsoft ISA server is being used. This is the URL for the login function, and after a successful login the application will redirect to the URL `/default.aspx`.

2. The URL is a common fingerprint for the phpBB web forum software. Information about this software is readily available on the Internet, and you can perform your own installation to experiment on. A listing of members can be found at the following URL:

   ```
   http://wahh-app.com/forums/memberlist.php
   ```

Individual user profiles can be found via URLs like the following:

```
http://wahh-app.com/forums/profile.php?mode=viewprofile&u=2
```

Various vulnerabilities have been found in the phpBB software so you should confirm the version in use and research any associated problems.

3. The `.asp` file extension indicates that Microsoft's Active Server Pages are in use. The use of a `/public` path indicates that other interesting paths might exist, such as `/private`. The `action=view` parameter suggests that other actions may exist, such as `edit`, `add` or `delete`. The function of the `location=default` parameter should be investigated – this may contain a filename, and you should probe the application for path traversal vulnerabilities.
4. If the header is accurate, it indicates that the server is running Apache Tomcat. Tomcat is a Java Servlet Container, so the application probably uses Java and JSP technologies.
5. The first response uses the HTTP status code 200, which normally indicates that the request was successful. However, the Content-Location header indicates the location from which the response was retrieved. This appears to be a dynamically generated error page, and includes the value 404 in its query string, indicating that the response contains a customized "file not found" message.

   The second response uses the HTTP status code 401, which suggests that the requested resource is present but that users must supply HTTP authentication credentials in order to access it.

   In each case, you could substantiate your conclusion by requesting a clearly non-existent item in the same directory with the same extension (for example, `/iuwehuiwefuwedw.cpf`) and comparing the responses. In the first application, you would expect to see a response very similar to the original. In the second application, you would expect to see a different response containing a "file not found" message.

Chapter 5 – Bypassing Client-Side Controls

1. The data can be encrypted or hashed using a key stored on the server, as is optionally done for the ASP.NET ViewState. Unless an attacker somehow captures the key, they will be unable to encrypt arbitrary data or compute a valid hash for arbitrary data. However, the attacker may still be able to take data from one context and replay it in another – for example, the encrypted price for a cheap item could be submitted in place of the encrypted price for an expensive item. To prevent this attack, the application should include sufficient context within the protected data to be able to confirm that it originated in the same context as it is being employed – for example, the product code and price could be combined in a single encrypted blob.
2. The defense is trivial to bypass. An attacker does not need to submit the cookie that tracks the number of failed login attempts. They can either disable cookies in their browser, or use an automated script that submits requests without the relevant cookie.

   An alternative defense would be to use CAPTCHA controls to slow down an attacker, or to block the source IP address after five failed logins, although this may have an adverse impact where multiple clients are located behind a proxy or a NAT-ting firewall.
3. (a) The Referer header can be set to an arbitrary value by an attacker, and so is not a safe means of performing any access control checks.

   (b) This method will only be effective if the web server containing the diagnostic functions is in a parent or child domain of the originating web server, and the session cookie is appropriately scoped, otherwise the cookie will not be submitted to the diagnostic server. A back-end mechanism will need to be implemented for the diagnostic server to validate the submitted tokens with the originating server.

   (c) This method will be effective regardless of the domain name of the diagnostic server. It may be regarded as safe provided that the authentication tokens are not predictable and are transmitted in a secure manner (see Chapter 7). Again, a back-end mechanism for validating tokens will need to be implemented.
4. There are two basic methods:

   (a) You can intercept the request containing the form submission, and add the disabled parameter.

   (b) You can intercept the response containing the form, and remove the `disabled=true` attribute.
5. There is no means by which an application can ensure that a piece of logic has been run on the client. Everything that occurs on the client is within the control of the user.

Chapter 6 – Attacking Authentication

1. (a) The credentials are transmitted within the query string of the URL. These are at risk of unauthorized disclosure via the browser history, the logs of the web server and IDS, or simply by appearing on-screen.

   (b) The credentials are transmitted via an unencrypted HTTP connection, making them vulnerable to interception by an attacker who is suitably positioned on the network.

   (c) The password is an English word consisting of four lower case alphabetical characters. The application is not enforcing any effective password quality rules.
2. Self-registration functions are very often vulnerable to username enumeration because users can choose their own username and the application prevents them from registering an existing username.

   Applications can avoid self-registration functionality being misused in this way through two methods:

   (a) The application can generate its own usernames, assigning a non-predictable username to each new user when they have supplied the required personal information.

(b) The first step of the self-registration process can require users to enter their email address. The application then sends the user an email containing a one-time URL that they can use to continue the registration process. If the supplied email address is already registered, the user is notified of this in the email.

3. The rationale for requesting two randomly chosen letters from the user’s memorable word, rather than the entire word, is that even if an attacker captures all of the credentials supplied by a user in a single login, it is unlikely that the attacker will be able to repeat the login using those credentials, because a different pair of letters will be requested.

   If the application requests all of the required information in a single step, then it must select the randomly chosen letters in advance, without knowing the claimed identity of the authenticating user. This means that an attacker who knows only two letters from a user’s memorable word can simply reload the login form repeatedly until those two letters are requested, enabling them to log in using the captured credentials.

   To avoid this defect, the application must choose a new pair of letters following each successful login, and store these in the user’s profile until such time as the user successfully logs in again. When the user has identified themselves at stage one of the login, the pair of letters is retrieved from their profile, and requested from the user. In this way, an attacker who has captured the credentials in a single login will typically need to wait a very long period until the them items are re-requested by the application.

4. An attacker attempting to guess valid credentials can easily determine whether an individual item is valid or invalid. The application’s behavior effectively enables an attacker to break down the brute-force problem into a series of individual challenges.

   The vulnerability can be corrected by continuing through all steps of the login process even if an invalid item is submitted, returning a generic “login failed” message after the final stage, regardless of which item caused the failure. This massively increases the number of requests required to guess a user’s credentials using brute force.

5. The presence of the anti-phishing mechanism enables an attacker to break the problem of guessing valid credentials into two stages. An attacker can verify whether or not a particular username and date of birth are valid by completing step (a) twice with these values. If the same anti-phishing image is returned on each occasion, then the guessed credentials are almost certainly valid; otherwise, they are not. A scripted attack could quickly iterate through a wide range of dates of birth for a targeted username, in order to guess the correct value.

   Worse still, the mechanism devised is not effective in preventing phishing attacks. A cloned web application will receive the username and date of birth supplied by the user in step (a), and can submit these directly to the original application to retrieve the correct image to present to the user in step (b). If users have been told to trust the image to provide assurance of the application’s identity, then the mechanism may actually be counter-productive and may cause users to log in to a phishing site that they would not otherwise trust.

Chapter 7 – Attacking Session Management

1. The `sessid` cookie contains a Base64-encoded string. Decoding the two values you received reveals the following values:

   ```
   jim23:1241:1194870863
   ```

   ```
   jim23:1241:1194875132
   ```

   The decoded cookie contains three items of data, separated by semicolons. On first inspection, the three values may contain a username, numeric user identifier, and a changing numeric value. The latter contains 10 digits, and looks like a Unix time value. Translating these two values reveals the following:

   ```
   Mon, 12 Nov 2007 12:34:23 UTC
   ```

   ```
   Mon, 12 Nov 2007 13:45:32 UTC
   ```

   These represent the times at which each of your sessions was created.

   Hence, it appears that the session tokens are comprised of meaningful user-specific data and a predictable item. In principle, you could mount a brute force attack to guess the tokens issued to other application users.

2. A 6-character session token allows for a considerably larger range of possible values than a 5-character password. It may therefore appear that the shorter passwords present the most worthwhile target for an attack.

   However, there are important differences between brute force attacks targeting passwords and those targeting session tokens. When attempting to guess passwords, it is necessary to supply a username and password together, thus targeting at most one account with each request, and maybe none at all. You may already know some usernames, or be able to enumerate them, or you may need to guess usernames and passwords simultaneously. The login mechanism may contain multiple stages, or be slow to respond. The login mechanism may also enforce account lockout, considerably slowing down your attack.

   When attempting to guess session tokens, on the other hand, you can often target multiple users simultaneously. There may be 20 users logged in, or 2000, or zero. If a user is not presently logged in, then you cannot target them in this way. There is no easy way for an application to enforce any kind of “lockout” when a large number of invalid tokens are received. Token guessing attacks normally run very quickly – requests containing an invalid token usually receive a fast response containing an error message or redirection.

   In short, there is no definitive answer to the question, and the most worthwhile target will depend upon your purposes and other aspects of the application. If many users are logged in and you simply need to compromise any user, then targeting sessions may be best. If you wish to compromise the single administrative account, which rarely logs in, then a password guessing attack will be more effective.

3. (a) Yes. The domain and path both match the scope of the cookie.

   (b) No. The domain is not the same or a subdomain of the domain scope of the cookie.

   (c) Yes. The domain is a subdomain of the domain specified in the scope, and the path matches the scope.

(d) Yes. The domain and path both match the scope of the cookie. Although the protocol is HTTP, the `secure` flag was not specified, so the cookie is still transmitted.

(e) Yes. The domain matches the scope of the cookie. Because the path scope did not include a trailing slash after `/login`, the scope includes not only the path `/login/` but also any other patch matching the `/login` prefix.

(f) No. The path does not match the scope of the cookie.

(g) No. The domain is the parent of the domain specified in the scope, and so is not included.

(h) No. The domain is not the same or a subdomain of the domain scope of the cookie.

Note that the `HttpOnly` flag affects whether cookies are accessible via client-side JavaScript. It does not determine whether they are transmitted via HTTP or HTTPS connections.

4. Session hijacking is still possible. If an attacker obtains the tokens issued to a user, the attacker can immediately make requests using those tokens, and the server will accept the requests. However, if the user issues a single further request to the application, then the per-page token submitted by the user will be out of sequence, and the entire session will be invalidated. Hence, if the user is still interacting with the application, the window for exploitation may be very small. If an attacker simply wishes to perform a specific action with the user's privileges, it is likely that they can script an attack to perform the desired action within the available window.

5. The logout function is broken.

   The script invalidates the session token currently held in the browser, meaning that its previous value will not be submitted in any subsequent requests. It then initiates a redirection to the application start page. Any attempt to access protected functionality will be denied because the request will not be made as part of an authenticated session.

   However, the client-side application does not communicate to the server that a logout action has been performed. The user's session on the server will remain active, and the token previously issued will continue to be accepted if issued to the server. This will remain the case indefinitely until the session is timed out or otherwise cleaned up. During that interval, an attacker who has captured or guessed the token's value through some means can continue to use it to hijack the user's session.

Chapter 8 – Attacking Access Controls

1. Choose a range of important application functions that you are authorized to access. Walk through each function submitting a modified or absent `Referer` header. If the application rejects your requests, it may well be vulnerable. Try making the same requests in a user context where these are unauthorized, but restore the original `Referer` header each time. If the requests are now accepted, then the application is certainly vulnerable.

2. You should attempt the following tests, in order of effectiveness:

   (a) Modify the `uid` value to a different value with the same syntactic form. If your own account details are still returned, then the application is probably not vulnerable.

   (b) If you are able to register or otherwise access a different user account, log in using that account to obtain the other user's `uid` value. Using your original user context, substitute this new `uid` in place of your own. If sensitive data about the other user is displayed, then the application is vulnerable.

   (c) Use a script to iterate up and down for a few thousand values from your `uid`, and determine whether any other users' details are returned.

   (d) Use a script to request random `uid` values between 0 and 9999999999 (in the present example) and determine whether any other users' details are returned.

3. It is possible to spoof another user's IP address, although in practice this may be extremely difficult. More significantly, different end users on the Internet may share the same IP address if they are behind the same web proxy or NAT-ting firewall.

   One way in which IP-based access controls can be effective in this situation is as a defense-in-depth measure to ensure that users attempting to access administrative functions are located on the organization's internal network. Those functions should also, of course, be protected by robust authentication and session handling mechanisms.

4. There is no horizontal or vertical segregation of access within the application, so there is no need for any access controls that discriminate between different individual users.

   Even though all users are in the same category, the application still needs to restrict the actions that any user can perform. A robust solution will use the principle of least privilege to ensure that all user roles within the application's architecture have the minimum permissions necessary for the application to function. For example, because users only need read access to data, the application should access the database using a low privileged account with read-only permissions to only the relevant tables.

5. The files are Excel spreadsheets. These are static resources which cannot enforce any access controls over themselves, in the way that dynamic scripts can. It is possible that the application is using other methods, such as at the web server layer, to protect access to the resources, but this is not typically the case. You should quickly check whether the resources can be accessed without authentication.

Chapter 9 – Attacking Data Stores

1. You can determine the number of columns in two easy ways. First, you can `SELECT` the type-neutral value `NULL` from each column, increasing the number of columns until the application returns data, indicating that the correct number of columns were specified, for example:

   ```
   ' UNION SELECT NULL--
   ```

`' UNION SELECT NULL, NULL--`

`' UNION SELECT NULL, NULL, NULL--`

Note that on Oracle you will need to add `FROM DUAL` after the final `NULL` in each case.

Second, you can inject `ORDER BY` clauses and increment the specified column until an error occurs, indicating that an invalid column was requested:

`' ORDER BY 1--`

`' ORDER BY 2--`

`' ORDER BY 3--`

2. An easy way to confirm the database type is to use database-specific string concatenation syntax to construct some benign input within the query you control. For example, if the original value of the parameter is `London` you can submit the following items in turn:

   `'||'London`

   `'+'London`

   If the first results in the same behavior as the original, the database is probably Oracle. If the second results in the same behavior, the database is probably MS-SQL.

3. While it may seem counterintuitive, the user registration function is probably the safest. Registration functions normally use `INSERT` statements, which are unlikely to affect other records if you modify them. A function to update personal records is probably using conditional `UPDATE` statements. If you inject a payload like `' or 1=1--` you may cause all records in the table to be modified. Similarly, the function to unsubscribe is probably using conditional `DELETE` statements, and could impact on other users if you are not careful.

   That said, it is impossible to be completely certain in advance which statements are being carried out by any kind of functionality, and you should advise the application owner of the risks before you perform the test.

4. An easy way to achieve the same effect without using comment characters is with the input `' or 'a'='a`.

5. You can SQL comment characters to separate keywords and other items in your injected payloads, for example:

   `'/**/UNION/**/SELECT/**/username,password/**/FROM/**/users--`

6. You can use the CHAR command to return a string value from a numeric ASCII character code. For example, on Oracle the string `FOO` can be represented as:

   `CHAR(70)||CHAR(79)||CHAR(79)`

7. This situation arises where user-supplied input is being placed into other elements of a query, such as table and column names, rather than the query's parameters. A parameterized query cannot be precompiled with placeholders for these items, so a different solution needs to be used, probably based on very stringent input validation.

8. Because you already have administrative access, it is likely that you can retrieve any data you desire using the application itself, meaning that a SQL injection attack to retrieve the application's own data may be redundant. However, you can still leverage the attack to access any data relating to other applications that is held within the same database, or to escalate privileges within the database or the underlying operating system, to compromise the database server and extend your attack into the wider internal network.

9. XPath injection can only be used to retrieve data from the targeted XML file. Hence, if the application contains no sensitive data this is likely to be a low impact issue. Similarly, SQL injection flaws may not enable you to extract any sensitive data from the database. However, they can sometimes be leveraged to escalate privileges within the database and develop your attack in other ways. Depending on the situation, SQL injection may be a more significant vulnerability. OS command injection, on the other hand, is almost always a high impact vulnerability, because it usually enables you to directly compromise the underlying server and use it as the launch point for further attacks against internal systems.

10. If the function is accessing a database, then submitting the SQL wildcard `%` as the search query is likely to return a large number of records. Similarly, if the function is accessing an Active Directory, then submitting the wildcard `*` is likely to return a large number of records. Neither wildcard should have the same effect on the other system.

Chapter 10 – Attacking Back-End Components

1. Applications for configuring networking devices often contain functionality that cannot be easily implemented using normal web scripting APIs, such the ability to reboot the device, cycle log files, or reconfigure SNMP. These tasks can often be performed easily using one-line operating system commands. Therefore, application developers frequently implement the functionality by incorporating the relevant user input directly into a shell command string.

2. It appears that the user-supplied input is being incorporated into a file path that is used in a filesystem operation. It may be possible to provide crafted input to access arbitrary files on the server. You should try using `../` traversal sequences to access different directories. Since `.log` is being appended to your input, you should try using a NULL byte to terminate the filename. Note that the `home` directory that appears within the error message might be the same directory that appears within the URL, giving you a clue about the location of items within the web root.

3. The application may be vulnerable to XML external entity (XXE) injection. The prerequisites to retrieve the contents of arbitrary files would normally be as follows:

   (a) The XML interpreter used by the application must support external entities.

(b) The application must echo in its response the contents of an XML element that appears within the request

4. The variable `param` has the value `urlparam1,urlparam2,bodyparam,cookieparam`.

5. Neither attack is strictly a prerequisite for the other.

   Although HPI attacks often involve HPP, they need not do so. For example, an HPI attack may inject an entirely new parameter into a back-end request, to interfere with the application's processing. This type of attack does not depend upon any particular behavior in the application's handling of multiple parameters with the same name.

   HPP attacks can often be used in situations that do not involve HPI, particularly where several layers of processing is performed on user input. For example, some exploits against the Internet Explorer XSS filter use HPP techniques, but do not inject any parameter into back-end requests.

6. There are many alternative representations of the server's loopback IP address that could be used to bypass the application's filter. For example:

   - 127.1
   - 127.000.0.1
   - Any other address in the 127.0.0.0 Class A subnet
   - Any variations on these in binary or octal representations such as 017700000001.

7. Mail injection attacks against this application function would not require the mail server to support mail relaying. Hardcoding the `RCPT TO` field would not prevent mail injection if other mail headers contain user-controller input, since an attacker could inject a second recipient using a second `RCPT TO` line. The most effective defense in this situation is to stricly validate all user-supplied inputs to ensure they do not contain any newlines or other SMTP metacharacters.

Chapter 11 – Attacking Application Logic

1. Forced browsing involves circumventing any constraints imposed by in-browser navigation on the sequence with which application functions may be accessed. You should use forced browsing to test for faulty assumptions in multi-stage processes and other areas. These assumptions often lead to access control weaknesses which you can exploit using forced browsing.

2. If quotation marks are doubled up before the length limit is enforced, then you can introduce an odd number of single quotes into your input by causing the input to be truncated in between two doubled up quotes (see Chapter 9).

   If the length limit is applied before the doubling up, then you may still be able to exploit any buffer overflow conditions by placing a large number of single quotes at the start of your payload, causing this to extend sufficiently far to overflow the buffer with crafted data positioned towards the end of your payload.

3. Using valid credentials for an account you control, you should repeat the login process numerous times, modifying your requests in specific ways:

   - For each parameter submitted, try submitting an empty value, omitting the name/value pair altogether, and submitting the same item multiple times with different values.

   - If the process involves multiple stages, try performing these stages in a different sequence, skipping individual stages altogether, proceeding directly to arbitrary stages, and submitting parameters at stages where they are not expected.

   - If the same item of data is submitted more than once, probe to determine how each value is processed, and whether data that is validated at one stage is trusted later on.

4. The application may well be performing these two checks independently, validating the password against one username and the token’s value against a different username, and then creating an authenticated session in the context of *one* of the validated usernames.

   If an application user who possesses their own physical token manages to obtain the password of another user, they may be able to login as that user. Conversely, depending on how the logic functions, a user who can read the value from another user’s token may be able to login as that user without knowing their password. The overall security posture of the solution is thus significantly diminished.

5. This behavior indicates that the error message functionality is not thread safe, and is returning the details of the last error generated by any user. You should probe further using two different sessions simultaneously to confirm whether this is the case. If so, you should use a script to constantly trigger an informative message and log any deviations in its contents for interesting information relating to other application users.

Chapter 12 – Attacking Users: Cross-Site Scripting

1. User-supplied input is returned unmodified within the application’s response to that input.

2. In most cases, XSS flaws within unauthenticated functionality work just as effectively against authenticated users – the functionality behaves in the same way, resulting in arbitrary JavaScript execution within the context of the authenticated user’s session.

   Even if a target user is not logged in at the time of the attack, they may still be compromised. If the application is vulnerable to session fixation, then an attacker can capture their token and wait for them to log in. An attacker can inject code into the login page to capture keystrokes, or even present a Trojan login form which sends their credentials elsewhere.

3. The answer to the first question is "yes". The behavior of course enables arbitrary JavaScript to be injected via a crafted request. The answer to the second question is "maybe". Historically, various ways have existed of injecting arbitrary HTTP headers into cross-domain requests, to inject a malicious cookie. Older versions of Flash and `XMLHttpRequest` have been vulnerable in this way. Further, many applications designed to use a cookie will in fact accept the same named parameter in other locations, such as the query string or message body.

4. In isolation, it appears that this behavior could only ever be used by a user to attack themselves. However, in conjunction with another suitable vulnerability, such as an access control flaw, or cross-site request forgery vulnerability, it could be highly significant, and could enable an attacker to inject stored JavaScript into the pages displayed to other application users.
5. If the application displays HTML or text files without any sanitization, then JavaScript contained within these will execute within the browser of any user who views the attachment. Further, if a JPEG file contains HTML, then this will be automatically processed as HTML within some browsers. Many web mail applications do not adequately defend against XSS in message attachments.
6. Because `XMLHttpRequest` can be used to retrieve the full response from an HTTP request, it can only normally be used to make requests to the same domain as the one that is invoking it. However, HTML5 has introduced the facility for `XMLHttpRequest` to make cross-domain requests, and retrieve responses, with the permission of the requested domain (see Chapter 13).
7. There are countless different attack payloads for XSS exploits. Some of the more commonly discussed payloads are:
   - stealing the session cookie;
   - inducing user actions;
   - injecting Trojan functionality;
   - stealing cached autocomplete data; and
   - loglogging keystrokes.
8. You can “convert” the reflected XSS flaw into a DOM-based one. For example, if the vulnerable parameter is called `vuln`, you can use the following URL to execute an arbitrarily long script:

   ```
   /script.asp?vuln=<script>eval(location.hash.substr(1))</script>#alert('long script here ......')
   ```
9. If the POST method is mandatory, you cannot simply construct a crafted URL within the application that will execute an attack when a user visits it. However, you can create a third-party web page that submits to the vulnerable application a form using the POST method and the relevant parameters in hidden fields. You can use JavaScript to automatically submit the form when a user views your page.

Chapter 13 – Attacking Users: Other Techniques

1. (a) Arbitrary redirection to lend credibility to a phishing attack.

   (b) Injection of a cookie header to exploit a session fixation flaw.

   (c) A response splitting attack to poison the cache of a proxy server.
2. An attacker must be able to determine all of the relevant parameters to the function in advance – that is, they must not contain any secret or unpredictable values that an attacker cannot set without already having hijacked a victim’s session.
3. (a) The standard anti-CSRF defense of including an unpredictable parameter within requests for JavaScript objects containing sensitive data.

   (b) The insertion of invalid or problematic JavaScript at the start of a JavaScript response.

   (c) The mandatory use of the POST method for retrieval of JavaScript objects.
4. (a) Flash will request the file `/crossdomain.xml` when a Flash object attempts to make a cross-domain request and retrieve the response. Even if the object specifies an alternative location from which the cross-domain policy should be loaded, Flash will still request `/crossdomain.xml` to confirm whether this is permitted by the master policy.

   (b) Java will not request `/crossdomain.xml` to check for a cross-domain policy.

   (c)HTML5 will not request `/crossdomain.xml` to check for a cross-domain policy.

   (d) Silverlight will request the file `/crossdomain.xml` when a Silverlight object attempts to make a cross-domain request and retrieve the response, provided that the `/clientaccesspolicy.xml` file, which it requests first, does not exist.
5. Clickjacking attacks involve the attacker's website creating a frame containing the vulnerable website. They have nothing to do with whether the targeted site itself employs frames.
6. The vulnerability could be exploited using the following steps:

   (a) The attacker creates his own account on the application, and places a malicious payload into his own display name.

   (b) The attacker creates his own site that causes visitors to log in to the vulnerable application using the attacker's credentials (via a CSRF attack against the login function), and then request the page containing the malicious display name.

   (c) When a victim is induced to visit the attacker's website, she is logged in to the vulnerable application, and the attacker's JavaScript executes. This script persists itself within the victim's browser, logs out of the application, and presents some content that induces the victim to log in using her own credentials. If she does this, then the attacker's script can compromise both the victim's credentials and her resulting session. From the victim's persepective the attack is seamless and appears to involve simply following a link, and then being presented with the vulnerable application's own login function.

7. You can add the header

   ```
   Origin: attacker.com
   ```

   to each request, and see whether the application responds with the header

   ```
   Access-Control-Allow-Origin
   ```

   If so, you can determine from this header which external domains (if any) the application permits two-way interaction from.

8. (a) Some applications contain functionality that takes an arbitrary name/value in parameters and sets these within a cookie in the response.

   (b) Any HTTP header injection or XSS vulnerabilities can be used to set arbitrary cookies for the affected domain.

   (c) A man-in-the-middle attacker can set cookies for arbitrary domains.

Chapter 14 – Automating Customized Attacks

1. (a) HTTP status code

   (b) Response length

   (c) Contents of response body

   (d) Contents of Location header

   (e) Setting of any cookies

   (f) Occurrence of any time delays

2. There are no definitive answers to this question. The following are examples of fuzz strings that are suitable for testing each of the categories of vulnerability. There are many other strings that would be equally suitable.

   (a) `'`

   ```
   '; waitfor delay '0:0:30'--
   ```

   (b) `||ping -i 30 127.0.0.1;x||ping -n 30 127.0.0.1 &`

   (c) `../../../../../../../../../../etc/passwd`

   ```
   ..\..\..\..\..\..\..\..\..\..\boot.ini
   ```

   (d) `http://<yourservername>/`

3. In many situations, modifying a parameter's value in some way will result in an error, causing the application to stop the rest of its processing on that request. The application will not therefore execute various code paths in which the other parameters may be processed in unsafe ways.

   One effective way to ensure that you achieve a decent level of code coverage with your automated fuzzing is to use a benign request as your template, and to modify each parameter in turn, leaving the other parameter with their initial values. You can then go on to perform manual testing of multiple parameters simultaneously, based on the results of the fuzz testing and your understanding of the role of each parameter. If time permits, you can also go on to perform more elaborate fuzzing, changing multiple parameters simultaneously using different permutations of payloads.

4. Often, in addition to the redirection, the application will set a new cookie when you submit to valid credentials, assigning you an authenticated session that will result in different content when you follow the redirection. If this is the case, then you can use the presence of a `Set-Cookie` header as a reliable indicator of a hit.

   If this is not the case, and the application simply upgrades your existing session when you submit valid credentials, then your script will probably need to follow the target of the redirection and inspect the contents of the resulting page to determine whether you have successfully logged in.

5. If you are lucky, you will be able to devise a regular expression that uniquely matches the data preceding the information you need to capture, or the data item itself. Otherwise, you will probably need to create a completely custom script to parse each application response and identify the interesting item.

Chapter 15 – Exploiting Information Disclosure

1. The application is inserting your input directly into a dynamically constructed query. However, it appears to be stripping any whitespace characters from your input, as can be seen from the expression `having1` that appears in the error message.

   The condition is certainly exploitable. You can use SQL comments instead of whitespace to separate items of syntax in your query, for example:

   ```
   https://wahh-app.com/list.aspx?artist=foo'/**/having/**/1%3d1--
   ```

   which returns the different error message

   ```
   Server: Msg 8118, Level 16, State 1, Line 1
   ```

```
Column 'users.ID' is invalid in the select list because it is not contained in an aggregate function and there is
no GROUP BY clause.
```

This confirms that the condition can be exploited and completes the first step in enumerating the structure of the query being performed.

2. The error message indicates the usernames that the application is using to access the database, the mode of connecting, the absolute file paths of the application's web content, and the line numbers in the scripts where the errors were generated. In isolation, each item of information may appear inconsequential. However, in conjunction with other vulnerabilities this information may assist you in developing a focused attack against the application.
3. This is a system-generated error message produced by `cgiwrap`. It indicates that the script you requested cannot be executed on the server because it does not have suitable file permissions. This script is therefore probably of little interest to you.

   The error message contains some information that may be of use, including an email address. More significantly, however, it contains various details that have been copied from the client request. You should probe the server's handling of crafted input in the relevant request headers to see if the error message is vulnerable to XSS. Note that a user can be induced to make a request containing arbitrary request headers via a Flash object.
4. You supplied the value `admin` in the `name` parameter. The error message indicates that the application attempted (and failed) to connect to a database on a host named `admin`. It appears that the application is letting you control the database that it will use to fulfill the request.

   You should try submitting the IP address or hostname for a server that you control, and see if the application connects to you. You should also try to guess a range of IP addresses within the internal network, to see if you can probe for other databases reachable from the application server.

   Given that the supplied hostname has been copied into the error message, you should also investigate whether the application is vulnerable to XSS. Peripheral content such as error messages is often subject to less rigorous input validation and other controls than the primary functionality within the application.
5. The error message is generated by a script that is attempting to assign your string-based input to a numeric variable. It appears that you will hit this error any time that you supply data in this parameter that is not numeric. There is no indication that your input has caused a database error, or even been processed by a database. This parameter is almost certainly not vulnerable to SQL injection.

Chapter 16 – Attacking Native Compiled Applications

1. With stack-based overflows, you can usually get immediate control of the saved return address on the stack, and therefore the instruction pointer when the current function returns. You can point the instruction pointer at an arbitrary address containing your shellcode (usually within the same buffer that triggers the overflow).

   With heap-based overflows, you can usually set an arbitrary pointer in memory to an arbitrary value. Leveraging this modification to take control of the flow of execution will normally involve a further step. Further, once a heap buffer has been overflowed, your attack may not execute immediately but may depend upon unpredictable events that impinge upon the allocation of heap memory.
2. There is no separate record of the length of a standard C/C++ string. The string is considered to continue until the first null byte after the start of the string.
3. Although it may be relatively easy to probe for and detect a buffer overflow vulnerability in a remote web application, developing a working exploit for the bug will in general be extremely difficult (though not absolutely impossible).

   With local access to a vulnerable network device, on the other hand, it is possible to attach debugging equipment and fully investigate the nature of the vulnerability, and thereby develop a finely crafted attack to exploit it reliably.
4. The %n format specifier has been disabled by default in the latest implementations of the `printf` family of functions. Hence, you should always supply a large number of %s specifiers, which will always be supported and are very likely to trigger an exception if your input is handled in an unsafe way.

   Further, using only the `printf` family of specifiers will not detect vulnerable calls to other formatting functions, such as `FormatMessage`.
5. You have probably identified a heap overflow vulnerability. Every overlong request you are submitting is probably causing corruption of the heap control structures. However, heap corruption normally only results in an exception when a relevant heap operation takes place, and the timing of this operation may depend on other events that are unrelated to the requests you are submitting.

   Note that in some rare situations, the same behavior may occur for different reasons – for example, because of load balancing or deferred processing of your input.

Chapter 17 – Attacking Application Architecture

1. You can almost certainly exploit the vulnerability to retrieve application data held within the database. The application itself must possess the necessary credentials and privileges required to access its own data. You can examine the server-side application's scripts and configuration files to discover how it access the application. An obvious way to exploit the vulnerability you have found would be to create some new scripts within the web root that enable you to perform arbitrary queries and retrieve the results using your browser.
2. Even if you fully compromise the entire database server, this may not necessarily provide a means of compromising the application server. In a typical case, the application server accesses the database server solely as a database client, and the database server may not be trusted in any other way by the application server.

   Nevertheless, it is probably possible to modify the content returned to users, since some of this will be generated using data contained within the database. For example, even if the application contains no stored XSS vulnerabilities that can be triggered within the application itself, you may be able to inject arbitrary scripts into the application's responses by modifying data directly within the database. If this enables you to attack an administrative user then you may quickly be able to compromise the entire application.
3. The PHP language contains a number of powerful functions, which can launch operating system commands and access the file system. If you are able to modify the files used by another application, then you can probably compromise that application. However, the possibility of achieving this depends

greatly on the configuration of the PHP environment, and the existence of any controls over your actions implemented lower down the technology stack.

4. Locating all application components on the same server normally prevents effective segregation between those components, meaning that a compromise of one part of the application's architecture can quickly lead to the compromise of others. For example, a file disclosure vulnerability within the web application may enable you to retrieve files containing sensitive data from the database. Similarly, a SQL injection vulnerability may enable you use database functions to write arbitrary files on the server file system, creating scripts within the web root that you can access from your browser, and thereby directly compromise the application tier.
5. You can perform Internet searches on key URLs and parameter names to locate other applications employing items with the same names. If these applications appear to contain a large overlap of shared functionality, you can investigate where they are physically hosted to gain additional evidence.

Chapter 18 – Attacking the Application Server

1. A web server will display a directory listing if you request a URL for a directory and:

   (a) the web server cannot find a default document such as `index.html`;

   (b) directory listings are enabled;

   (c) you have the required permissions to access the directory.

2. WebDAV methods allow web-based authoring of web content.

   These methods may be dangerous if they are not subjected to strict access control. Further, because of the complex functionality involved, they have historically been a source of vulnerabilities within web servers – for example, as an attack vector for exploiting operating system vulnerabilities via the IIS server.

3. If the proxy allows connection back out to the Internet, you could use it to attack third party web applications on the Internet, with your requests appearing to originate from the misconfigured web server.

   Even if requests to the Internet are blocked, you may be able to leverage the proxy to access web servers within the organization that are not directly accessible, or to reach other web-based services on the server itself.

4. The PL/SQL Exclusion List is a pattern-matching blacklist designed to prevent the PL/SQL gateway from being used to access certain powerful database packages.

   Various bypasses have been discovered to the PL/SQL Exclusion List filter. These essentially arise because the filter employs very simple expressions, while the back-end database follows much more complex rules to interpret the significance of the input. Numerous ways have been discovered of crafting input that does not match the blacklist patterns but nevertheless succeeds in executing the powerful packages within the database.

5. It is possible that using HTTPS for communication may conceal your attacks from some network-layer intrusion detection systems. Using HTTP, however, will typically enable your automated attacks to execute much faster. The application may contain different content or behave differently when accessed via the different protocols, so in general you should be prepared to test using both.

Chapter 19 – Finding Vulnerabilities in Source Code

1. (a) Cross-site scripting

   (b) SQL injection

   (c) Path traversal

   (d) Arbitrary redirection

   (e) OS command injection

   (f) Backdoor passwords

   (g) Some native software bugs

2. PHP uses a range of different built-in arrays to store user-submitted data. If `register_globals` is enabled, then PHP creates a global variable for every request parameter, and applications may access a parameter simply by referencing a variable with the same name – there is no syntactic indication that this variable represents user input as opposed to any other variable defined elsewhere.

3. Method 1 is more secure.

   Although method 1 constructs a SQL query dynamically from user input, it doubles up all single quotation marks that appear within that input, and all user-supplied parameters are treated as string data. Although this is not the best practice way to handle SQL queries safely, there does not appear to be any opportunity for SQL injection in the present case.

   Method 2 uses a parameterized query, which is the preferred way to incorporate user-supplied data into SQL statements in a safe way. However, only two of the three items of user input are properly parameterized. One of the items is erroneously placed directly into the string that specifies the query structure, and so the application is fully vulnerable to SQL injection.

4. The application may be vulnerable to reflected XSS, because it appears that an on-screen welcome message is being constructed directly from a request parameter.

   It is not possible to confirm conclusively whether the application is vulnerable from this code snippet alone. To do this, you would need to investigate:

(a) whether any input validation is performed on the `name` parameter elsewhere; and

(b) whether any output validation is performed on `m_welcomeMessage` before it is copied into the application’s response.

5. Yes. With knowledge of the token creation algorithm used, it is possible to extrapolate forwards and backwards to identify all tokens created by the application, on the basis of a single sampled token.

   The Java API `java.util.Random` implements a linear congruential generator that generates pseudo-random numbers according to a fully predictable algorithm. Knowing the state of the generator at any iteration, it is possible derive the sequence of numbers that it will generate next, and (with a little number theory) derive the sequence that it generated previously.

   However, the `java.util.Random` generator maintains 48 bits of state, and the `nextInt` method only returns 32 bits of that state. Capturing a single output from the `nextInt` method is not sufficient to determine the state of the generator, or to predict its sequence of outputs.

   In the present case, this difficulty can be easily circumvented because the algorithm used by the application makes two successive calls to `nextInt`. Each session token created contains 32 bits of state from one iteration of the generator, and 32 bits of state from the next iteration. Given this information, it is straightforward to perform a local brute force exercise to discover the missing 16 bits of state at the first iteration (by trying each possible permutation of the missing 16 bits, and testing whether the generator outputs the 32 bits captured from the second iteration). Once the missing 16 bits have been confirmed, the full state of the generator is known, and you can proceed to derive subsequent and earlier outputs in the standard way.

   Ironically, the developers’ decision to make two calls to `nextInt` and combine the results renders the token creation algorithm more vulnerable than it would otherwise be.

   See the following paper by Chris Anley for more details of this type of attack:

   http://www.ngssoftware.com/research/papers/Randomness.pdf