CHAPTER

15

Exploiting Information Disclosure

Chapter 4 described various techniques you can use to map a target application and gain an initial understanding of how it works. That methodology involved interacting with the application in largely benign ways to catalog its content and functionality, determine the technologies in use, and identify the key attack surface.

This chapter describes ways in which you can extract further information from an application during an actual attack. This mainly involves interacting with the application in unexpected and malicious ways and exploiting anomalies in the application's behavior to extract information that is of value to you. If successful, such an attack may enable you to retrieve sensitive data such as user credentials, gain a deeper understanding of an error condition to fine-tune your attack, discover more details about the technologies in use, and map the application's internal structure and functionality.

Exploiting Error Messages

Many web applications return informative error messages when unexpected events occur. These may range from simple built-in messages that disclose only the category of the error to full-blown debugging information that gives away a lot of details about the application's state.

Most applications are subject to various kinds of usability testing prior to deployment. This testing typically identifies most error conditions that may arise when the application is being used in the normal way. Therefore, these conditions usually are handled in a graceful manner that does not involve any technical messages being returned to the user. However, when an application is under active attack, it is likely that a much wider range of error conditions will arise, which may result in more detailed information being returned to the user. Even the most security-critical applications, such as those used by online banks, have been found to return highly verbose debugging output when a sufficiently unusual error condition is generated.

Script Error Messages

When an error arises in an interpreted web scripting language, such as VBScript, the application typically returns a simple message disclosing the nature of the error, and possibly the line number of the file where the error occurred. For example:

```
Microsoft VBScript runtime error 800a0009
Subscript out of range: [number -1]
/register.asp, line 821
```

This kind of message typically does not contain any sensitive information about the state of the application or the data being processed. However, it may help you narrow down the focus of your attack. For example, when you are inserting different attack strings into a specific parameter to probe for common vulnerabilities, you may encounter the following message:

```
Microsoft VBScript runtime error '800a000d'
Type mismatch: ' [string: "'"]'
/scripts/confirmOrder.asp, line 715
```

This message indicates that the value you have modified is probably being assigned to a numeric variable, and you have supplied input that cannot be so assigned because it contains nonnumeric characters. In this situation, it is highly likely that nothing can be gained by submitting nonnumeric attack strings as this parameter. So for many categories of bugs, you are better off targeting other parameters.

A different way in which this type of error message may assist you is in giving you a better understanding of the logic that is implemented within the server-side application. Because the message discloses the line number where the error occurred, you may be able to confirm whether two different malformed requests are triggering the same error or different errors. You may also be able to determine the sequence in which different parameters are processed by submitting bad input within multiple parameters and identifying the location at which an error occurs. By systematically manipulating different parameters, you may be able to map the different code paths being executed on the server.

Stack Traces

Most web applications are written in languages that are more complex than simple scripts but that still run in a managed execution environment, such as Java, C#, or Visual Basic .NET. When an unhandled error occurs in these languages, it is common to see full stack traces being returned to the browser.

A stack trace is a structured error message that begins with a description of the actual error. This is followed by a series of lines describing the state of the execution call stack when the error occurred. The top line of the call stack shows the function that generated the error, the next line shows the function that invoked the previous function, and so on down the call stack until the hierarchy of function calls is exhausted.

The following is an example of a stack trace generated by an ASP.NET application:

```
[HttpException (0x80004005): Cannot use a leading .. to exit above the
top directory.]
   System.Web.Util.UrlPath.Reduce(String path) +701
   System.Web.Util.UrlPath.Combine(String basepath, String relative)+304
   System.Web.UI.Control.ResolveUrl(String relativeUrl) +143
   PBSApp.StatFunc.Web.MemberAwarePage.Redirect(String url) +130
   PBSApp.StatFunc.Web.MemberAwarePage.Process() +201
   PBSApp.StatFunc.Web.MemberAwarePage.OnLoad(EventArgs e)
   System.Web.UI.Control.LoadRecursive() +35
   System.Web.UI.Page.ProcessRequestMain() +750

Version Information: Microsoft .NET Framework Version:1.1.4322.2300;
ASP.NET Version:1.1.4322.2300
```

This kind of error message provides a large amount of useful information that may assist you in fine-tuning your attack against the application:

- It often describes the precise reason why an error occurred. This may enable you to adjust your input to circumvent the error condition and advance your attack.
- The call stack typically makes reference to a number of library and third-party code components that are being used within the application. You can review the documentation for these components to understand their intended behavior and assumptions. You can also create your own local

implementation and test this to understand the ways in which it handles unexpected input and potentially identify vulnerabilities.

- The call stack includes the names of the proprietary code components being used to process the request. The naming scheme for these and the interrelationships between them may allow you to infer details about the application's internal structure and functionality.
- The stack trace often includes line numbers. As with the simple script error messages described previously, these may enable you to probe and understand the internal logic of individual application components.
- The error message often includes additional information about the application and the environment in which it is running. In the preceding example, you can determine the exact version of the ASP.NET platform being used. This enables you to investigate the platform for known or new vulnerabilities, anomalous behavior, common configuration errors, and so on.

Informative Debug Messages

Some applications generate custom error messages that contain a large amount of debug information. These are usually implemented to facilitate debugging during development and testing and often contain rich detail about the application's runtime state. For example:

```
-------------------------------------------
* * * S E S S I O N * * *
-------------------------------------------
i5agor2n2pw3gp551pszsb55
SessionUser.Sessions App.FEStructure.Sessions
SessionUser.Auth 1
SessionUser.BranchID 103
SessionUser.CompanyID 76
SessionUser.BrokerRef RRadv0
SessionUser.UserID 229
SessionUser.Training 0
SessionUser.NetworkID 11
SessionUser.BrandingPath FE
LoginURL /Default/fedefault.aspx
ReturnURL ../default/fedefault.aspx
SessionUser.Key f7e50aef8fadd30f31f3aea104cef26ed2ce2be50073c
SessionClient.ID 306
SessionClient.ReviewID 245
UPriv.2100
```

```
SessionUser.NetworkLevelUser 0
UPriv.2200
SessionUser.BranchLevelUser 0
SessionDatabase fd219.prod.wahh-bank.com
```

The following items are commonly included in verbose debug messages:

- Values of key session variables that can be manipulated via user input
- Hostnames and credentials for back-end components such as databases
- File and directory names on the server
- Information embedded within meaningful session tokens (see Chapter 7)
- Encryption keys used to protect data transmitted via the client (see Chapter 5)
- Debug information for exceptions arising in native code components, including the values of CPU registers, contents of the stack, and a list of the loaded DLLs and their base addresses (see Chapter 16)

When this kind of error reporting functionality is present in live production code, it may signify a critical weakness in the application's security. You should review it closely to identify any items that can be used to further advance your attack, and any ways in which you can supply crafted input to manipulate the application's state and control the information retrieved.

Server and Database Messages

Informative error messages are often returned not by the application itself but by some back-end component such as a database, mail server, or SOAP server. If a completely unhandled error occurs, the application typically responds with an HTTP 500 status code, and the response body may contain further information about the error. In other cases, the application may handle the error gracefully and return a customized message to the user, sometimes including error information generated by the back-end component. In some situations, information disclosure can itself be used as a conduit for an attack. The information disclosed by an application in a debug message or exception is often unintentional and as a result the organization's security procedures may entirely overlook the existence of the disclosure.

The error returned may enable a range of further attacks, as described in the following sections.

Using Information Disclosure to Advance an Attack

When a specific attack is launched against a server back-end component, it is common for that component to give direct feedback on any errors encountered. This can help you fine-tune the attack. Database error messages often contain useful information. For example, they often disclose the query that generated the error, enabling you to fine-tune a SQL injection attack:

```
Failed to retrieve row with statement - SELECT object_data FROM
deftr.tblobject WHERE object_id = 'FDJE00012' AND project_id = 'FOO'
and 1=2--'
```

See Chapter 9 for a detailed methodology describing how to develop database attacks and extract information based on error messages.

Cross-Site Scripting Attacks Within Error Messages

As described in Chapter 12, securing against cross-site scripting is an arduous task, requiring identification of each output location of user-supplied data. Although most frameworks automatically HTML-encode data when reporting errors, this is by no means universal. Error messages can appear in multiple, often unusual places within an HTTP response. In the `HttpServletResponse.sendError()` call used by Tomcat, the error data is also part of the response header:

```
HTTP/1.1 500 General Error Accessing Doc10083011
Server: Apache-Coyote/1.1
Content-Type: text/html;charset=ISO-8859-1
Content-Length: 1105
Date: Sat, 23 Apr 2011 08:52:15 GMT
Connection: close
```

An attacker who has control over the input string `Doc10083011` could supply carriage return characters and conduct an HTTP header injection attack, or a cross-site scripting attack within the HTTP response. More details can be found here:

```
http://www.securityfocus.com/archive/1/495021/100/0/threaded
```

Frequently customized error messages are intended for a non-HTML destination, such as a console, yet they are erroneously reported to the user in an HTTP response. In these situations, cross-site scripting is often easily exploitable.

Decryption Oracles in Information Disclosure

Chapter 11 gave an example of how an unintentional "encryption oracle" could be harnessed to decrypt strings presented to the user in encrypted format. The same issue can apply to information disclosure. Chapter 7 gave an example of an application that provided an encrypted download link for file access. If a file had since been moved or deleted, the application reported that the file could not be downloaded. Of course, the error message contained the file's decrypted value, so any encrypted "filename" could be provided to the download link, resulting in an error.

In these cases, the information disclosure resulted from abuse of deliberate feedback. It is also possible for information disclosure to be more accidental if parameters are decrypted and then used in various functions, any of which may log data or generate error messages. An example encountered by the authors was a complex work flow application that made use of encrypted parameters transmitted via the client. Swapping the default values used for `dbid` and `grouphome`, the application responded with an error:

```
java.sql.SQLException: Listener refused the connection with the
following error: ORA-12505, TNS:listener does not currently know
of SID given in connect descriptor The Connection descriptor used
by the client was: 172.16.214.154:1521:docs/londonoffice/2010/general
```

This provided considerable insight. Specifically, `dbid` was actually an encrypted SID for a connection to an Oracle database (the connection descriptor takes the form *Server*:*Port*:*SID*), and `grouphome` was an encrypted file path.

In an attack analogous to many other information disclosure attacks, knowledge of the file path provided the necessary information to conduct a file path manipulation attack. Supplying exactly three path traversal characters in a filename, and navigating up a similar directory structure, it was possible to upload files containing malicious script directly into another group's work space:

```
POST /dashboard/utils/fileupload HTTP/1.1
Accept: text/html, application/xhtml+xml, */*
Referer: http://wahh/dashboard/common/newnote
Accept-Language: en-GB
Content-Type: multipart/form-data; boundary=------7db3d439b04c0
Accept-Encoding: gzip, deflate
Host: wahh
Content-Length: 8088
Proxy-Connection: Keep-Alive

--------7db3d439b04c0
Content-Disposition: form-data; name="MAX_FILE_SIZE"

100000
--------7db3d439b04c0
Content-Disposition: form-data; name="uploadedfile"; filename="../../../
newportoffice/2010/general/xss.html"
Content-Type: text/html
<html><body><script>...
...
```

HACK STEPS

1. When you are probing the application for common vulnerabilities by submitting crafted attack strings in different parameters, always monitor the application's responses to identify any error messages that may contain useful information.

   Attempt to force an error response from the application by supplying encrypted data strings in the wrong context, or by performing actions on resources that are not in the correct state to handle the action.

2. Be aware that error information that is returned within the server's response may not be rendered on-screen within the browser. An efficient way to identify many error conditions is to search each raw response for keywords that are often contained in error messages. For example:

   - error
   - exception
   - illegal
   - invalid
   - fail
   - stack
   - access
   - directory
   - file
   - not found
   - varchar
   - ODBC
   - SQL
   - SELECT

3. When you send a series of requests modifying parameters within a base request, check whether the original response already contains any of the keywords you are looking for to avoid false positives.

4. You can use the Grep function of Burp Intruder to quickly identify any occurrences of interesting keywords in any of the responses generated by a given attack (see Chapter 14). Where matches are found, review the relevant responses manually to determine whether any useful error information has been returned.

TIP If you are viewing the server's responses in-browser, be aware that Internet Explorer by default hides many error messages and replaces them with a generic page. You can disable this behavior by choosing Tools ➢ Internet Options and then choosing the Advanced tab.

Using Public Information

Because of the huge variety of web application technologies and components in common use, you should frequently expect to encounter unusual messages that you have not seen before and that may not immediately indicate the nature of the error that the application experienced. In this situation, you can often obtain further information about the message's meaning from various public sources.

Often, an unusual error message is the result of a failure in a specific API. Searching for the text of the message may lead you to the documentation for this API or to developer forums and other locations where the same problem is discussed.

Many applications employ third-party components to perform specific common tasks, such as searches, shopping carts, and site feedback functions. Any error messages that are generated by these components are likely to have arisen in other applications and probably have been discussed elsewhere.

Some applications incorporate source code that is publicly available. By searching for specific expressions that appear in unusual error messages, you may discover the source code that implements the relevant function. You can then review this to understand exactly what processing is being performed on your input and how you may be able to manipulate the application to exploit a vulnerability.

HACK STEPS

1. **Search for the text of any unusual error messages using standard search engines. You can use various advanced search features to narrow down your results. For example:**

   ```
   "unable to retrieve" filetype:php
   ```

2. **Review the search results, looking both for any discussion about the error message and for any other websites in which the same message has appeared. Other applications may produce the same message in a more verbose context, enabling you to better understand what kind of conditions give rise to the error. Use the search engine cache to retrieve examples of error messages that no longer appear within the live application.**

3. **Use Google code search to locate any publicly available code that may be responsible for a particular error message. Search for snippets of error messages that may be hard-coded into the application's source code. You can also use various advanced search features to specify the code language and other details if these are known. For example:**

   ```
   unable\ to\ retrieve lang:php package:mail
   ```

4. **If you have obtained stack traces containing the names of library and third-party code components, search for these names on both types of search engines.**

Engineering Informative Error Messages

In some situations, it may be possible to systematically engineer error conditions in such a way as to retrieve sensitive information within the error message itself.

One common situation in which this possibility arises is where you can cause the application to attempt some invalid action on a specific item of data. If the resulting error message discloses the value of that data, and you can cause interesting items of information to be processed in this way, you may be able to exploit this behavior to extract arbitrary data from the application.

Verbose open database connectivity (ODBC) error messages can be leveraged in a SQL injection attack to retrieve the results of arbitrary database queries. For example, the following SQL, if injected into a `WHERE` clause, would cause the database to cast the password for the first user in the users table to an integer to perform the evaluation:

```
' and 1=(select password from users where uid=1)--
```

This results in the following informative error message:

```
Error: Conversion failed when converting the varchar value
'37CE1CCA75308590E4D6A35F288B58FACDBB0841' to data type int.
```

TRY IT

http://mdsec.net/addressbook/32

A different way in which this kind of technique can be used is where an application error generates a stack trace containing a description of the error, and you can engineer a situation where interesting information is incorporated into the error description.

Some databases provide a facility to create user-defined functions written in Java. By exploiting a SQL injection flaw, you may be able to create your own function to perform arbitrary tasks. If the application returns error messages to the browser, from within your function you can throw a Java exception containing arbitrary data that you need to retrieve. For example, the following code executes the operating system command `ls` and then generates an exception that contains the output from the command. This returns a stack trace to the browser, the first line of which contains a directory listing:

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
try
{
    Process p = Runtime.getRuntime().exec("ls");
    InputStream is = p.getInputStream();
    int c;
    while (-1 != (c = is.read()))
        baos.write((byte) c);
}
```

```
catch (Exception e)
{
}
throw new RuntimeException(new String(baos.toByteArray()));
```

Gathering Published Information

Aside from the disclosure of useful information within error messages, the other primary way in which web applications give away sensitive data is by actually publishing it directly. There are various reasons why an application may publish information that an attacker can use:

- By design, as part of the application's core functionality
- As an unintended side effect of another function
- Through debugging functionality that remains present in the live application
- Because of some vulnerability, such as broken access controls

Here are some examples of potentially sensitive information that applications often publish to users:

- Lists of valid usernames, account numbers, and document IDs
- User profile details, including user roles and privileges, date of last login, and account status
- The current user's password (this is usually masked on-screen but is present in the page source)
- Log files containing information such as usernames, URLs, actions performed, session tokens, and database queries
- Application details in client-side HTML source, such as commented-out links or form fields, and comments about bugs

HACK STEPS

1. **Review the results of your application mapping exercises (see Chapter 4) to identify all server-side functionality and client-side data that may be used to obtain useful information.**
2. **Identify any locations within the application where sensitive data such as passwords or credit card details are transmitted from the server to the browser. Even if these are masked on-screen, they are still viewable within the server's response. If you have found another suitable vulnerability, such as within access controls or session handling, this behavior can be used to obtain the information belonging to other application users.**
3. **If you identify any means of extracting sensitive information, use the techniques described in Chapter 14 to automate the process.**

Using Inference

In some situations, an application may not divulge any data to you directly, but it may behave in ways that enable you to reliably infer useful information.

We have already encountered many instances of this phenomenon in the course of examining other categories of common vulnerability. For example:

- A registration function that enables you to enumerate registered usernames on the basis of an error message when an existing username is chosen (see Chapter 6).
- A search engine that allows you to infer the contents of indexed documents that you are not authorized to view directly (see Chapter 11).
- A blind SQL injection vulnerability in which you can alter the application's behavior by adding a binary condition to an existing query, enabling you to extract information one bit at a time (see Chapter 9).
- The "padding oracle" attack in .NET, where an attacker can decrypt any string by sending a series of requests to the server and observing which ones result in an error during decryption (see Chapter 18).

Another way in which subtle differences in an application's behavior may disclose information occurs when different operations take different lengths of time to perform, contingent upon some fact that is of interest to an attacker. This divergence can arise for various reasons:

- Many large and complex applications retrieve data from numerous back-end systems, such as databases, message queues, and mainframes. To improve performance, some applications cache information that is used frequently. Similarly, some applications employ a *lazy load* approach, in which objects and data are loaded only when needed. In this situation, data that has been recently accessed is retrieved quickly from the server's local cached copy, while other data is retrieved more slowly from the relevant back-end source.

  This behavior has been observed in online banking applications. A request to access an account takes longer if the account is dormant than if it is active, enabling a skilled attacker to enumerate accounts that have been accessed recently by other users.
- In some situations, the amount of processing that an application performs on a particular request may depend on whether a submitted item of data is valid. For example, when a valid username is supplied to a login mechanism, the application may perform various database queries to retrieve account information and update the audit log. It also may perform

computationally intensive operations to validate the supplied password against a stored hash. If an attacker can detect this timing difference, he may be able to exploit it to enumerate valid usernames.

- Some application functions may perform an action on the basis of user input that times out if an item of submitted data is invalid. For example, an application may use a cookie to store the address of a host located behind a front-end load balancer. An attacker may be able to manipulate this address to scan for web servers inside the organization's internal network. If the address of an actual server that is not part of the application infrastructure is supplied, the application may immediately return an error. If a nonexistent address is supplied, the application may time out attempting to contact this address before returning the same generic error. You can use the response timers within Burp Intruder's results table to facilitate this testing. Note that these columns are hidden by default, but can be shown via the Columns menu.

HACK STEPS

1. **Differences in the timing of application responses may be subtle and difficult to detect. In a typical situation, it is worth probing the application for this behavior only in selected key areas where a crucial item of interesting data is submitted and where the kind of processing being performed is likely to result in time differences.**
2. **To test a particular function, compile one list containing several items that are known to be valid (or that have been accessed recently) and a second list containing items that are known to be invalid (or dormant). Make requests containing each item on these lists in a controlled way, issuing only one request at a time, and monitoring the time taken for the application to respond to each request. Determine whether there is any correlation between the item's status and the time taken to respond.**
3. **You can use Burp Intruder to automate this task. For every request it generates, Intruder automatically records the time taken before the application responds and the time taken to complete the response. You can sort a table of results by either of these attributes to quickly identify any obvious correlations.**

Preventing Information Leakage

Although it may not be feasible or desirable to prevent the disclosure of absolutely any information that an attacker may find useful, various relatively straightforward measures can be taken to reduce information leakage to a minimum and to withhold the most sensitive data that can critically undermine an application's security if disclosed to an attacker.

Use Generic Error Messages

The application should never return verbose error messages or debug information to the user's browser. When an unexpected event occurs (such as an error in a database query, a failure to read a file from disk, or an exception in an external API call), the application should return the same generic message informing the user that an error occurred. If it is necessary to record debug information for support or diagnostic purposes, this should be held in a server-side log that is not publicly accessible. An index number to the relevant log entry may be returned to the user, enabling him or her to report this when contacting the help desk, if required.

Most application platforms and web servers can be configured to mask error information from being returned to the browser:

- In ASP.NET, you can suppress verbose error messages using the `customErrors` element of the `Web.config` file by setting the mode attribute to `On` or `RemoteOnly` and specifying a custom error page in the `defaultRedirect` node.
- In the Java Platform, you can configure customized error messages using the error-page element of the `web.xml` file. You can use the `exception-type` node to specify a Java exception type, or you can use the `error-code` node to specify an HTTP status code. You can use the `location` node to set the custom page to be displayed in the event of the specified error.
- In Microsoft IIS, you can specify custom error pages for different HTTP status codes using the Custom Errors tab on a website's Properties tab. A different custom page can be set for each status code, and on a per-directory basis if required.
- In Apache, custom error pages can be configured using the `ErrorDocument` directive in `httpd.conf`:

  ```
  ErrorDocument 500 /generalerror.html
  ```

Protect Sensitive Information

Wherever possible, the application should not publish information that may be of use to an attacker, including usernames, log entries, and user profile details. If certain users need access to this information, it should be protected by effective access controls and made available only where strictly necessary.

In cases where sensitive information must be disclosed to an authorized user (for example, where users can update their own account information), existing data should not be disclosed where it is not necessary. For example, stored credit card numbers should be displayed in truncated form, and password fields should never be prefilled, even if masked on-screen. These defensive measures help mitigate the impact of any serious vulnerabilities that may exist within the application's core security mechanisms of authentication, session management, and access control.

Minimize Client-Side Information Leakage

Where possible, service banners should be removed or modified to minimize the disclosure of specific software versions and so on. The steps needed to implement this measure depend on the technologies in use. For example, in Microsoft IIS, the `Server` header can be removed using URLScan in the IISLockDown tool. In later versions of Apache, this can be achieved using the `mod_headers` module. Because this information is subject to change, it is recommended that you consult your server documentation before carrying out any modifications.

All comments should be removed from client-side code that is deployed to the live production environment, including all HTML and JavaScript.

You should pay particular attention to any browser extension components such as Java applets and ActiveX controls. No sensitive information should be hidden within these components. A skilled attacker can decompile or reverse-engineer these components to effectively recover their source code (see Chapter 5).

Summary

Leakage of unnecessary information frequently does not present any kind of significant defect in an application's security. Even highly verbose stack traces and other debugging messages may sometimes provide you with little leverage in seeking to attack the application.

In other cases, however, you may discover sources of information that are of great value in developing your attack. For example, you may find lists of usernames, the precise versions of software components, or the internal structure and functionality of the server-side application logic.

Because of this possibility, any serious assault on an application should include a forensic examination of both the application itself and publicly available resources so that you can gather any information that may be of use in formulating your attacks against it. On some occasions, information gathered in this way can provide the foundation for a complete compromise of the application that disclosed it.

Questions

Answers can be found at `http://mdsec.net/wahh`.

1. While probing for SQL injection vulnerabilities, you request the following URL:

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

   You receive the following error message:

   ```
   Server: Msg 170, Level 15, State 1, Line 1
   Line 1: Incorrect syntax near 'having1'.
   ```

   What can you infer from this? Does the application contain any exploitable condition?

2. While you are performing fuzz testing of various parameters, an application returns the following error message:

   ```
   Warning: mysql_connect() [function.mysql-connect]: Access denied for
   user 'premiumdde'@'localhost' (using password: YES) in
   /home/doau/public_html/premiumdde/directory on line 15
   Warning: mysql_select_db() [function.mysql-select-db]: Access denied
   for user 'nobody'@'localhost' (using password: NO) in
   /home/doau/public_html/premiumdde/directory on line 16
   Warning: mysql_select_db() [function.mysql-select-db]: A link to
   the server could not be established in
   /home/doau/public_html/premiumdde/directory on line 16
   Warning: mysql_query() [function.mysql-query]: Access denied for
   user 'nobody'@'localhost' (using password: NO) in
   /home/doau/public_html/premiumdde/directory on line 448
   ```

   What useful items of information can you extract from this?

3. While mapping an application, you discover a hidden directory on the server that has directory listing enabled and appears to contain a number of old scripts. Requesting one of these scripts returns the following error message:

   ```
   CGIWrap Error: Execution of this script not permitted
   Execution of (contact.pl) is not permitted for the following reason:
   Script is not executable. Issue 'chmod 755 filename'

   Local Information and Documentation:
   CGIWrap Docs: http://wahh-app.com/cgiwrap-docs/
   Contact EMail: helpdesk@wahh-app.com

   Server Data:
   Server Administrator/Contact: helpdesk@wahh-app.com
   Server Name: wahh-app.com
   Server Port: 80
   Server Protocol: HTTP/1.1
   ```

```
Request Data:
User Agent/Browser: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT
5.1; .NET CLR 2.0.50727; FDM; InfoPath.1; .NET CLR 1.1.4322)
Request Method: GET
Remote Address: 192.168.201.19
Remote Port: 57961
Referring Page: http://wahh-app.com/cgi-bin/cgiwrap/fodd
```

What caused this error, and what common web application vulnerability should you quickly check for?

4. You are probing the function of a request parameter in an attempt to determine its purpose within an application. You request the following URL:

```
https://wahh-app.com/agents/checkcfg.php?name=admin&id=13&log=1
```

The application returns the following error message:

```
Warning: mysql_connect() [function.mysql-connect]: Can't connect to
MySQL server on 'admin' (10013) in
/var/local/www/include/dbconfig.php on line 23
```

What caused this error message, and what vulnerabilities should you probe for as a result?

5. While fuzzing a request for various categories of vulnerabilities, you submit a single quotation mark within each request parameter in turn. One of the results contains an HTTP 500 status code, indicating potential SQL injection. You check the full contents of the message, which are as follows:

```
Microsoft VBScript runtime error '800a000d'
Type mismatch: ' [string: "'"]'
/scripts/confirmOrder.asp, line 715
```

Is the application vulnerable?