Example of why a hidden input may not be safe
Let's run through an example.

The goal is to help you understand why using hidden inputs can be considered dangerous for sensitive data.

Setting the scene

You are desperate to play the latest games, and you want to purchase a decent gaming laptop.

You go onto a site called shoppingFoo and search for a laptop.

You find one.

The only problem is the price. The asking price is $1,500.

Using hidden inputs to store the price of an item

Because HTTP is stateless, hidden form inputs are one way that web apps can use to remember data from one page request to the next (called "persistence").

For example, to remember the price of a new Laptop accessed from a page URL previously requested, this form remembers the sale price, and will provide it again to the shopping cart application when the client submits the payment page.

Lets say the shopping cart page consists of the laptop you found, with the price being stored in a hidden input field:

```
<form method="POST" action="processPayment.js">
    <input type="hidden" name="price" value="1500">
    $1500 x Quantity: <input name="quantity" size=4>
```

```html
    <br/>
    </br>
    <input type="submit" value="Buy">
</form>
```

Since a user can’t “see” the above hidden input, the developers who built this form thought that it would be safe.

But, let's prove them wrong.

Hidden form inputs store the state information client-side, instead of server-side. And this my dear students, makes the data vulnerable.

What can happen?

We've seen in earlier sections of this course that forms often use the HTTP POST method to submit form data to a URL (such as /shoppingCartPage.js) that legitimate clients never see.

Unless there is code to prevent it, this means you can easily send altered hidden inputs to this POST URL by altering a local copy of the page, using a browser plug-in tool or in some cases simply typing different URL parameters into the browser's location bar.

Let's continue with our above example.

You want to alter the sale price so that you can buy your gaming laptop much more cheaply.

What if you inspected dev tools, or used a plugin, and changed the value in the hidden field to this (notice we've changed the value attribute of the hidden input):

```html
<form method="POST" action="processPayment.js">
    <input type="hidden" name="price" value="1">
    $1500 x Quantity: <input name="quantity" size=4>
    <br/>
    </br>
    <input type="submit" value="Buy">
</form>
```

When this form is submitted, you'll be able to order a Laptop at a price reduced from $1500 to $1.

What???

In fact, lets go balls to the wall and order 1000 laptops at this price.

A pretty good deal (prison aside) right?

The request may look like this:

```
POST /processPayment.js HTTP/1.1
Host: www.shoppingFoo.com
Referer: http://www.shoppingFoo.com/shoppingCartPage.js
Cookie: JSESSIONID=1234
Content-Type: application/x-www-form-urlencoded
POSTDATA quantity=1000&price=1
```

Unless the web application is smart enough to test for unauthorized prices, /processPayment.js accepts the request, processes the order, and returns a normal reply like this:

```
HTTP/1.1 302 Moved
Set-Cookie: JSESSIONID=1234;HttpOnly
Location: http://www.shoppingFoo.com/thankYou.js
Content-Length: 0
Connection: close
Content-Type: text/plain; charset=UTF-8
```

The client then loads the final “thank you” shopping cart page indicated in the reply’s Location: header.

Final comments

I hope you can begin to see how having an input type of hidden does not mean the information is secure.

I know I may have gone on a slight tangent with these past few lectures (this info is beyond the scope of this course about building forms), but I wanted you to at least be aware of the pitfalls of using the hidden input type.

Lets move on.