Web Application Security

Rajendra Kachhwaha
rajendra1983@gmail.com

August 12, 2015

Outline

Bypassing Client-Side Controls

1. A large proportion of web applications, rely on various measures implemented on the client side to control the data that they submit to the server.
2. In general, this represents a fundamental security flaw: the user has full control over the client and the data it submits and can bypass any controls that are implemented on the client side and are not replicated on the server.
3. An application may rely on client-side controls to restrict user input in two broad ways:
   First, an application may transmit data via the client component using a mechanism that it assumes will prevent the user from modifying that data when the application later reads it.

Bypassing Client-Side Controls

Second, an application may implement measures on the client side that control the users interaction with his or her own client, with the aim of restricting functionality and/or applying controls around user input before it is submitted.

This may be achieved using HTML form features, client-side scripts, or browser extension technologies.

You may wonder why, if the server knows and specifies a particular item of data, the application would ever need to transmit this value to the client and then read it back.

In fact, writing applications in this way is often easier for developers for various reasons:

Bypassing Client-Side Controls

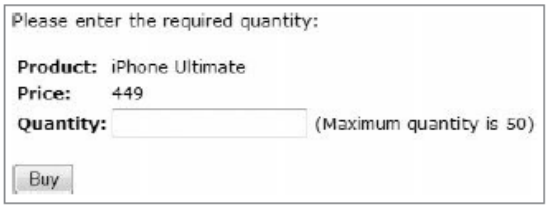
1. It removes the need to keep track of all kinds of data within the users session. Reducing the amount of per-session data being stored on the server can also improve the applications performance.
2. If the application is deployed on several distinct servers, with users potentially interacting with more than one server to perform a multi-step action, it may not be straightforward to share server-side data between the hosts that may handle the same users requests. Using the client to transmit data can be a tempting solution to the problem.
3. If the application employs any third-party components on the server, such as shopping carts, modifying these may be difficult or impossible, so transmitting data via the client may be the easiest way of integrating these.

Bypassing Client-Side Controls:Hidden Form Fields:

Hidden HTML form fields are a common mechanism for transmitting data via the client in a superficially unmodifiable way. If a field is flagged as hidden, it is not displayed on-screen. However, the fields name and value are stored within the form and are sent back to the application when the user submits the form. The classic example of this security flaw is a retailing application that stores the prices of products within hidden form fields. In the early days of web applications, this vulnerability was extremely widespread.

One way to achieve this is to edit the fields value using Inspect Element option, and click the Buy button. However, an easier and more elegant method is to use an intercepting proxy to modify the desired data on-the-fly.

Bypassing Client-Side Controls:Hidden Form Fields:



A typical HTML form

The code behind this form is as follows:

```
<form method="post" action="Shop.aspx?prod=1">
Product: iPhone 5 <br/>
Price: 449 <br/>
Quantity: <input type="text" name="quantity"> (Maximum quantity is 50)
<br/>
<input type="hidden" name="price" value="449">
<input type="submit" value="Buy">
</form>
```

Notice the form field called `price`, which is flagged as hidden. This field is sent to the server when the user submits the form:

```
POST /shop/28/Shop.aspx?prod=1 HTTP/1.1
Host: mdsec.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 20
quantity=1&price=449
```

Bypassing Client-Side Controls:HTTP Cookies:

Another common mechanism for transmitting data via the client is HTTP cookies. As with hidden form fields, normally these are not displayed on-screen, and the user cannot modify them directly. They can, of course, be modified using an intercepting proxy, by changing either the server response that sets them or subsequent client requests that issue them.
Consider the following variation on the previous example. After the customer has logged in to the application, he receives the following response:

```
HTTP/1.1 200 OK
Set-Cookie: DiscountAgreed=25
Content-Length: 1530
...
```

Bypassing Client-Side Controls:HTTP Cookies:

This DiscountAgreed cookie points to a classic case of relying on client-side controls to protect data transmitted via the client. If the application trusts the value of the DiscountAgreed cookie when it is submitted back to the server, customers can obtain arbitrary discounts by modifying its value. For example:

```
POST /shop/92/Shop.aspx?prod=3 HTTP/1.1
Host: mdsec.net
Cookie: DiscountAgreed=25 50
Content-Length: 10

quantity=1
```

Bypassing Client-Side Controls:URL Parameters:

Applications transmit data via the client using preset URL parameters. For example, when a user browses the product catalog, the application may provide him with hyperlinks to URLs like the following:

$http://mdsec.net/shop/?prod=3\&pricecode=32$

When a URL containing parameters is displayed in the browsers location bar, any parameters can be modified easily by any user without the use of tools.

Bypassing Client-Side Controls:The Referer Header:

Browsers include the Referer header within most HTTP requests. It is used to indicate the URL of the page from which the current request originated either because the user clicked a hyperlink or submitted a form, or because the page referenced other resources such as images. Hence, it can be leveraged as a mechanism for transmitting data via the client.

Because the URLs processed by the application are within its control, developers may assume that the Referer header can be used to reliably determine which URL generated a particular request.

For example, consider a mechanism that enables users to reset their password if they have forgotten it. The application requires users to proceed through several steps in a defined sequence before they actually reset their passwords value with the following request:

Bypassing Client-Side Controls:The Referer Header:

GET /auth/472/CreateUser.ashxHTTP/1.1
Host : mdsec.net
Referer : https : //mdsec.net/auth/472/Admin.ashx

The application may use the Referer header to verify that this request originated from the correct stage (Admin.ashx). If it did, the user can access the requested functionality.

However, because the user controls every aspect of every request, including the HTTP headers, this control can be easily circumvented by proceeding directly to CreateUser.ashx and using an intercepting proxy to change the value of the Referer header to the value that the application requires.

Bypassing Client-Side Controls:Hack Steps:

1. Locate all instances within the application where hidden form fields, cookies, and URL parameters are apparently being used to transmit data via the client.
2. Attempt to determine or guess the role that the item plays in the applications logic, based on the context in which it appears and on clues such as the parameters name.
3. Modify the items value in ways that are relevant to its purpose in the application. Ascertain whether the application processes arbitrary values submitted in the parameter, and whether this exposes the application to any vulnerabilities.

Bypassing Client-Side Controls:ASP.NET ViewState:

One commonly encountered mechanism for transmitting opaque data via the client is the ASP.NET ViewState. This is a hidden field that is created by default in all ASP.NET web applications. It contains serialized information about the state of the current page. The ASP.NET platform employs the ViewState to enhance server performance. It enables the server to preserve elements within the user interface across successive requests without needing to maintain all the relevant state information on the server side.

Bypassing Client-Side Controls:ASP.NET ViewState:

Developers can use it to store arbitrary information across successive requests. For example, instead of saving the products price in a hidden form field, an application may save it in the ViewState as follows:

stringprice = getPrice(prodno);

ViewState.Add("price", price);

The form returned to the user now looks something like this:

```
<form method="post" action="Shop.aspx?prod=3">
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNlBQMzOTlkZA==" />
Product: HTC Avalanche <br/>
Price: 399 <br/>
Quantity: <input type="text" name="quantity"> (Maximum quantity is 50)
<br/>
<input type="submit" value="Buy">
</form>
```

Bypassing Client-Side Controls:ASP.NET ViewState:

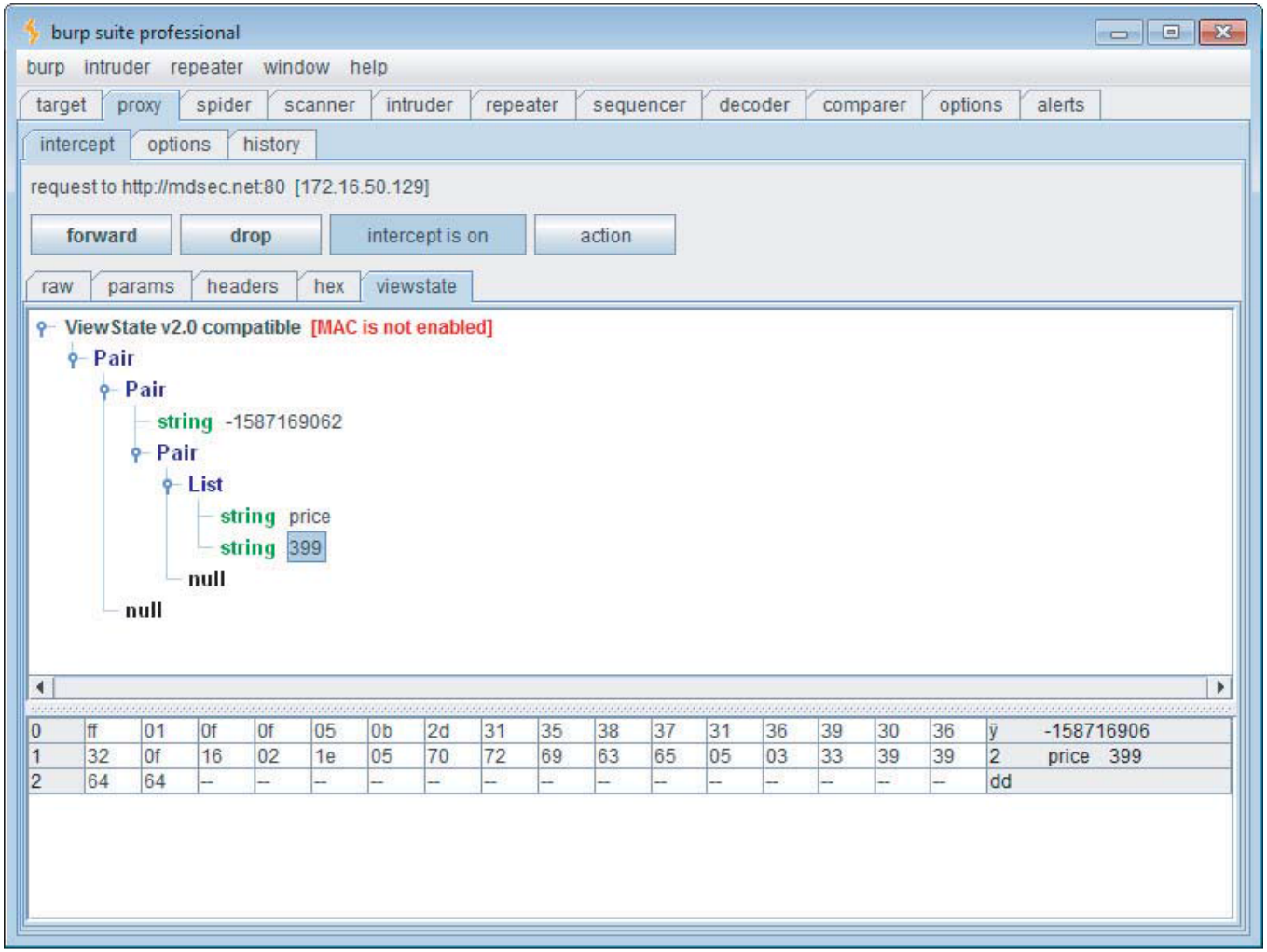
When the user submits the form, his browser sends the following:

```
POST /shop/76/Shop.aspx?prod=3 HTTP/1.1
Host: mdsec.net
Content-Type: application/x-www-form-urlencoded
Content-Length: 77

__VIEWSTATE=%2FwEPDwULLTE1ODcxNjkwNjIPFgIeBXByaWNlBQMzOTlkZA%3D%3D&
quantity=1
```

The request apparently does not contain the product price only the quantity ordered and the opaque ViewState parameter. Changing that parameter at random results in an error message, and the purchase is not processed.
The ViewState parameter is actually a Base64-encoded string that can be easily decoded to see the price parameter that has been placed there.

Bypassing Client-Side Controls:ASP.NET ViewState:



Bypassing Client-Side Controls:Hack Steps:

1. If you are attacking an ASP.NET application, verify whether MAC protection is enabled for the ViewState. This is indicated by the presence of a 20-byte hash at the end of the ViewState structure, and you can use the ViewState parser in Burp Suite to confirm whether this is present.
2. Even if the ViewState is protected, use Burp to decode the ViewState on various application pages to discover whether the application is using the ViewState to transmit any sensitive data via the client.
3. Try to modify the value of a specific parameter within the ViewState without interfering with its structure, and see whether an error message results.