Bypassing Client-Side Controls

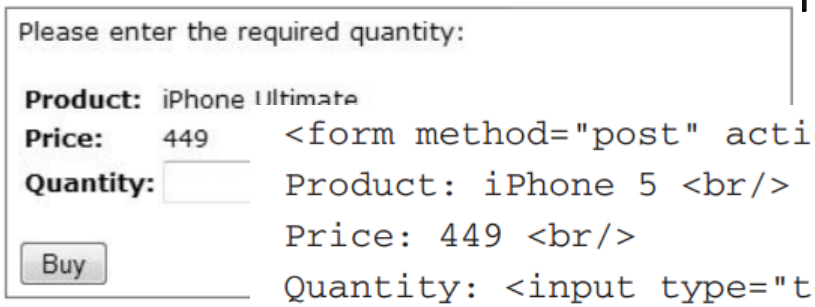
Transmitting Data Via the Client

- The benefits:
  - Reducing the amount of per-session data being stored on the server can also improve the application's performance.
  - If the application is deployed on several distinct servers, with users potentially interacting with more than one server to perform a multistep action, it may not be straightforward to share server-side data between the hosts that may handle the same user's requests. Using the client to transmit data can be a tempting solution to the problem.
  - 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.
- The limitations:
  - transmitting sensitive data in this way is usually unsafe and has been the cause of countless vulnerabilities in applications.

Hidden Form Fields

OWASP- SM-004

- 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-




```
<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>
```

HTTP Cookies

(OWASP-SM-002)

- 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
- odified using an nging either the server response that
requests that i

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

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

quantity=1
```

URL Parameters

OWASP-WS-0 05

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

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

- Upon the reception of an HTTP request, the code should do the following:
- Check:
  - 1. maximum length and minimum length
  - 2. Validate payload
  - 3. If possible, implement the following data validation strategies; "exact match", "known good," and "known bad" in that order.
  - 4. Validate parameter names and existence.

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 application's logic, based on the context in which it appears and on clues such as the parameter's name.
3. Modify the item's 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.

Script-Based Validation

- The input validation mechanisms built into HTML forms themselves are extremely simple and are insufficiently fine-grained to perform relevant validation of many kinds of input.
- For example, a user registration form might contain fields for name, e-mail address, telephone number, and zip code, all of which expect different types of input.
- Therefore, it is common to see customized client-side input validation implemented within scripts.

Script-Based Validation (2)

```
<form method=”post” action=”Shop.aspx?prod=2”
onsubmit=”return
validateForm(this)”>
Product: Samsung Multiverse <br/>
Price: 399 <br/>
Quantity: <input type=”text” name=”quantity”> (Maximum
quantity is 50)
<br/>
<input type=”submit” value=”Buy”>
</form>
<script>function validateForm(theForm)
{
var isInteger = /^\d+$/;
var valid = isInteger.test(quantity) &&
quantity > 0 && quantity <= 50;
if (!valid)
alert(’Please enter a valid quantity’);
return valid;
}
</script>
```

Script-Based Validation (3)

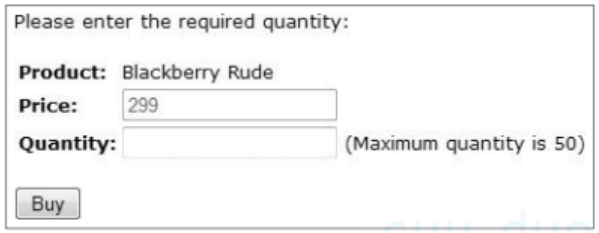
HACK STEPS

1. **Identify any cases where client-side JavaScript is used to perform input validation prior to form submission.**
2. **Submit data to the server that the validation ordinarily would have blocked, either by modifying the submission request to inject invalid data or by modifying the form validation code to neutralize it.**
3. **As with length restrictions, determine whether the client-side controls are replicated on the server and, if not, whether this can be exploited for any malicious purpose.**
4. **Note that if multiple input fields are subjected to client-side validation prior to form submission, you need to test each field individually with invalid data while leaving valid values in all the other fields. If you submit invalid data in multiple fields simultaneously, the server might stop processing the form when it identifies the first invalid field. Therefore, your testing won't reach all possible code paths within the application.**

Script-Based Validation (2)

Disabled Elements

- If an element on an HTML form is flagged as disabled, it appears on-screen but is usually grayed out and cannot be edited or used in the way an ordinary control can be.
- Also, it is not sent to the server when the form is




```
<form method="post" action="Shop.aspx?prod=5">
Product: Blackberry Rude <br/>
Price: <input type="text" disabled="true" name="price"
value="299">
<br/>
Quantity: <input type="text" name="quantity"> (Maximum
quantity is 50)
<br/>
<input type="submit" value="Buy">
</form>
```

Disabled Elements (2)

HACK STEPS

1. Look for disabled elements within each form of the application. Whenever you find one, try submitting it to the server along with the form's other parameters to determine whether it has any effect.
2. Often, submit elements are flagged as disabled so that buttons appear as grayed out in contexts when the relevant action is unavailable. You should always try to submit the names of these elements to determine whether the application performs a server-side check before attempting to carry out the requested action.
3. Note that browsers do not include disabled form elements when forms are submitted. Therefore, you will not identify these if you simply walk through the application's functionality, monitoring the requests issued by the browser. To identify disabled elements, you need to monitor the server's responses or view the page source in your browser.
4. You can use the HTML modification feature in Burp Proxy to automatically re-enable any disabled fields used within the application.

Opaque Data

- Data transmitted via the client is not transparently intelligible because it has been encrypted or obfuscated in some way.
- For example, instead of seeing a product's price stored in a hidden field, you may see a cryptic value being transmitted

Opaque Data

```
<form method="post" action="Shop.aspx?prod=4">
Product: Nokia Infinity <br/>
Price: 699 <br/>
Quantity: <input type="text" name="quantity"> (Maximum quantity is 50)
<br/>
<input type="hidden" name="price" value="699">
<input type="hidden" name="pricing_token"
value="E76D213D291B8F216D694A34383150265C989229">
<input type="submit" value="Buy">
</form>
```

Length Limits

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

Length Limits

HACK STEPS

1. Look for form elements containing a `maxlength` attribute. Submit data that is longer than this length but that is formatted correctly in other respects (for example, it is numeric if the application expects a number).
2. If the application accepts the overlong data, you may infer that the client-side validation is not replicated on the server.
3. Depending on the subsequent processing that the application performs on the parameter, you may be able to leverage the defects in validation to exploit other vulnerabilities, such as SQL injection, cross-site scripting, or buffer overflows.

How to defense

- The only secure way to validate client-generated data is on the server side of the application.
- Every item of data received from the client should be regarded as tainted and potentially malicious.

Attacking Authentication

cuu duong than cong . com