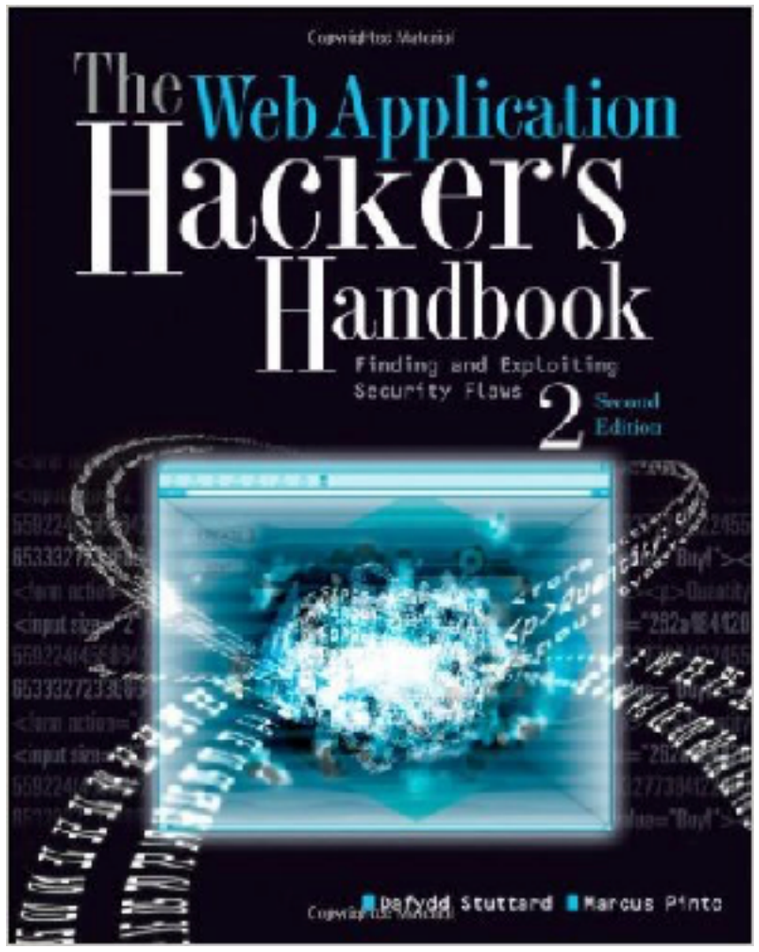
# CNIT 129S: Securing Web Applications

## Ch 8: Attacking Access Controls



# Access Control

- **Authentication and session management**
  - **Ensure that you know who is using the application**
- **Access Controls**
  - **Limit what actions are possible for authenticated users**
  - **Must be tested for every request and operation**

# Common Vulnerabilities

- **Vertical**
- **Horizontal**
- **Context-dependent**

# Vertical Privilege Escalation

- **Allow user to access a different part of the application's functionality**
  - **Such as escalating from User to Administrator**

# Horizontal Privilege Escalation

- **Allow a user to access a wider range of resources of the same type**

- **Such as reading another user's email**

# Context-Dependent: Business Logic Exploitation

- **User can do things that should not be possible in context**
- **Such as performing tasks out of order**
- **Or bypassing the payment step when checking out**

# Completely Unprotected Functionality

- **Functionality can be accessed by anyone who knows the URL**
- **OWASP calls this "Unsecured Direct Object Access"**
- **Also called "security through obscurity"**

```
https://wahh-app.com/admin/

https://wahh-app.com/menus/secure/ff457/DoAdminMenu2.jsp
```

# Finding Privileged URLs

- **App may not contain any link to the URLs**
- **Inspecting client-side code may reveal them**

```
var isAdmin = false;
...
if (isAdmin)
{
    adminMenu.addItem("/menus/secure/ff457/addNewPorta-
lUser2.jsp",
        "create a new user");
}
```

# Identifier-Based Functions

`https://wahh-app.com/ViewDocument.php?docid=1280149120`

- **Identifier for a document passed to server in a parameter**

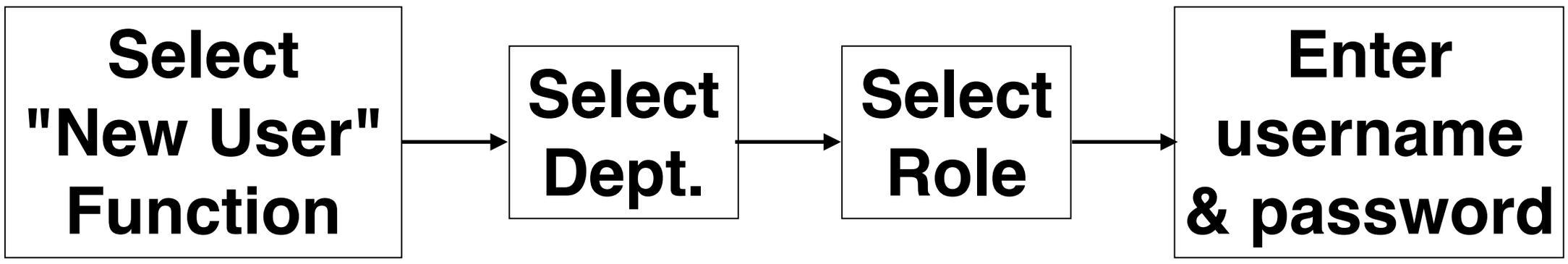
- **Access controls may be broken so everyone with the URL can view the document**

- **This happens when developers find it too difficult to share a session-based security model between servers using different technologies**

# Identifier-Based Functions

- **Document identifiers might be predictable**
  - **Such as simple incrementing numbers**
- **URLs are not treated as secrets**
  - **Often visible to attackers in logs, or elsewhere within an app**

# Multistage Functions




- **App may enforce access control for an early step, but not test it again at a later step**
- **So attacker can skip steps and escalate privileges**

# Static Files

- **Customers pay for a book, and then are sent to a download link like this**

```
https://wahh-books.com/download/9780636628104.pdf
```

- **It's a static resource, downloaded directly from the Web server**
- **No application-level code is executed**
- **Anyone with the URL can download the book**

# Platform Misconfiguration

- **Access to specified URL paths are restricted**
- **Ex: only Administrators can access the /admin path**

The platform-level configuration normally takes the form of rules that are akin to firewall policy rules, which allow or deny access based on the following:

- HTTP request method
- URL path
- User role

# Verb Tampering

- **Access control rule may only apply to the POST method**
- **Using GET may allow a non-administrator to perform admin-level tasks**
- **Also, the HEAD method is often implemented internally on the Web server with the GET method**
  - **And then just returning only the headers**
  - **So an admin function is still performed**
- **Unrecognized HTTP methods may default to GET**

# Insecure Access Control Methods

- **Parameter-based access control**
- **Referer-based access control**
- **Location-based access control**

# Parameter-Based Access Control

- **Privilege level in a hidden form field, cookie, or query string parameter**

```
https://wahh-app.com/login/home.jsp?admin=true
```

- **Attacker can just add the parameter to gain privileges**

# Referer-Based Access Control

- **HTTP Referer header grants access**
- **User can modify that field to gain privileges**

# Location-Based Access Control

- **Location restrictions, as in sports events**
- **Often uses geolocation of user's IP address**
- **Common methods of bypass:**

  - Using a web proxy that is based in the required location
  - Using a VPN that terminates in the required location
  - Using a mobile device that supports data roaming
  - Direct manipulation of client-side mechanisms for geolocation

# Testing with Different User Accounts

- **Burp can map the contents of an application using two user accounts and compare them**

**Figure 8.1** A site map comparison showing the differences between content that was accessed in different user contexts

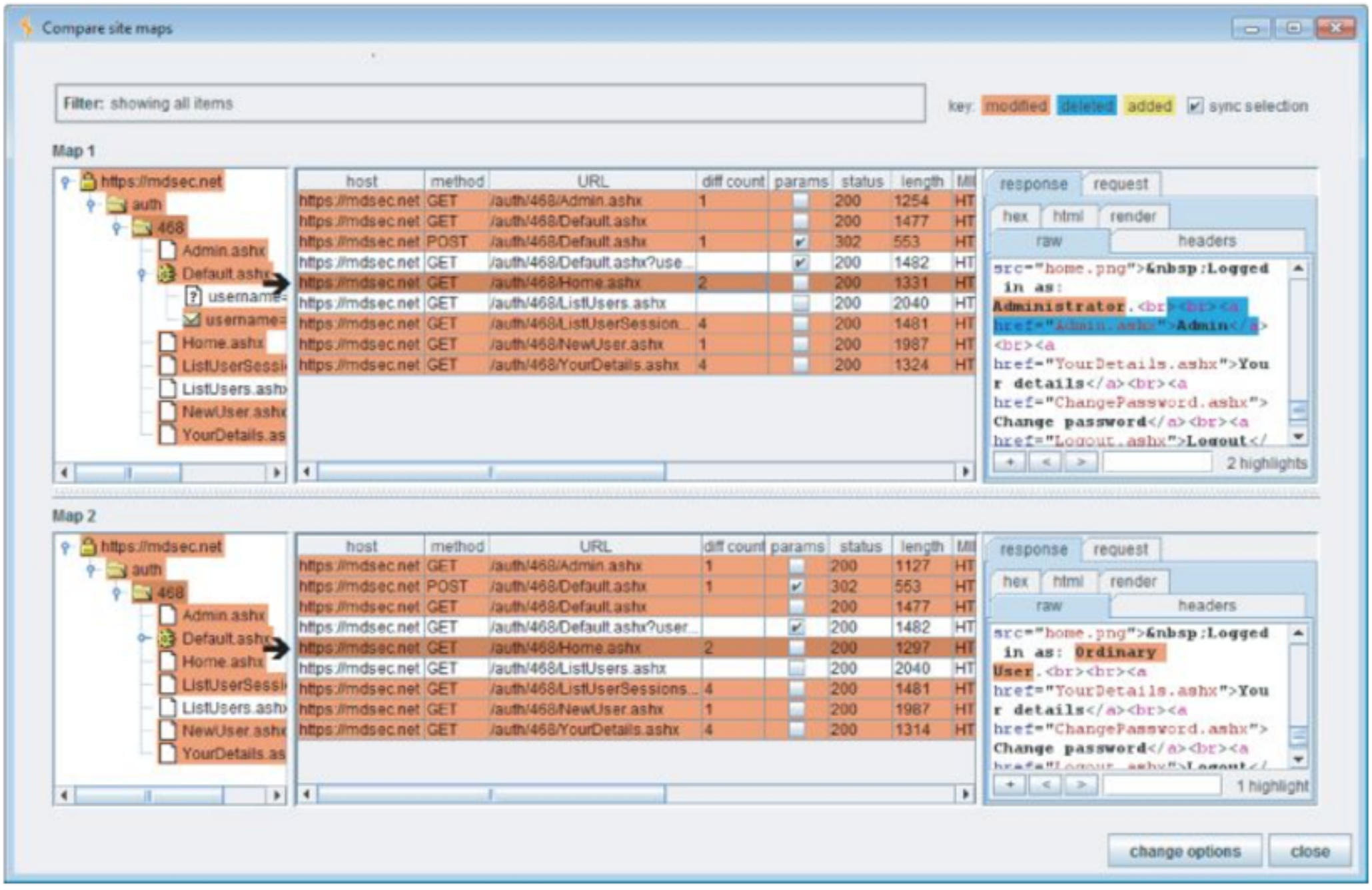


**Figure 8.2** The low-privileged user is denied access to the top-level admin page

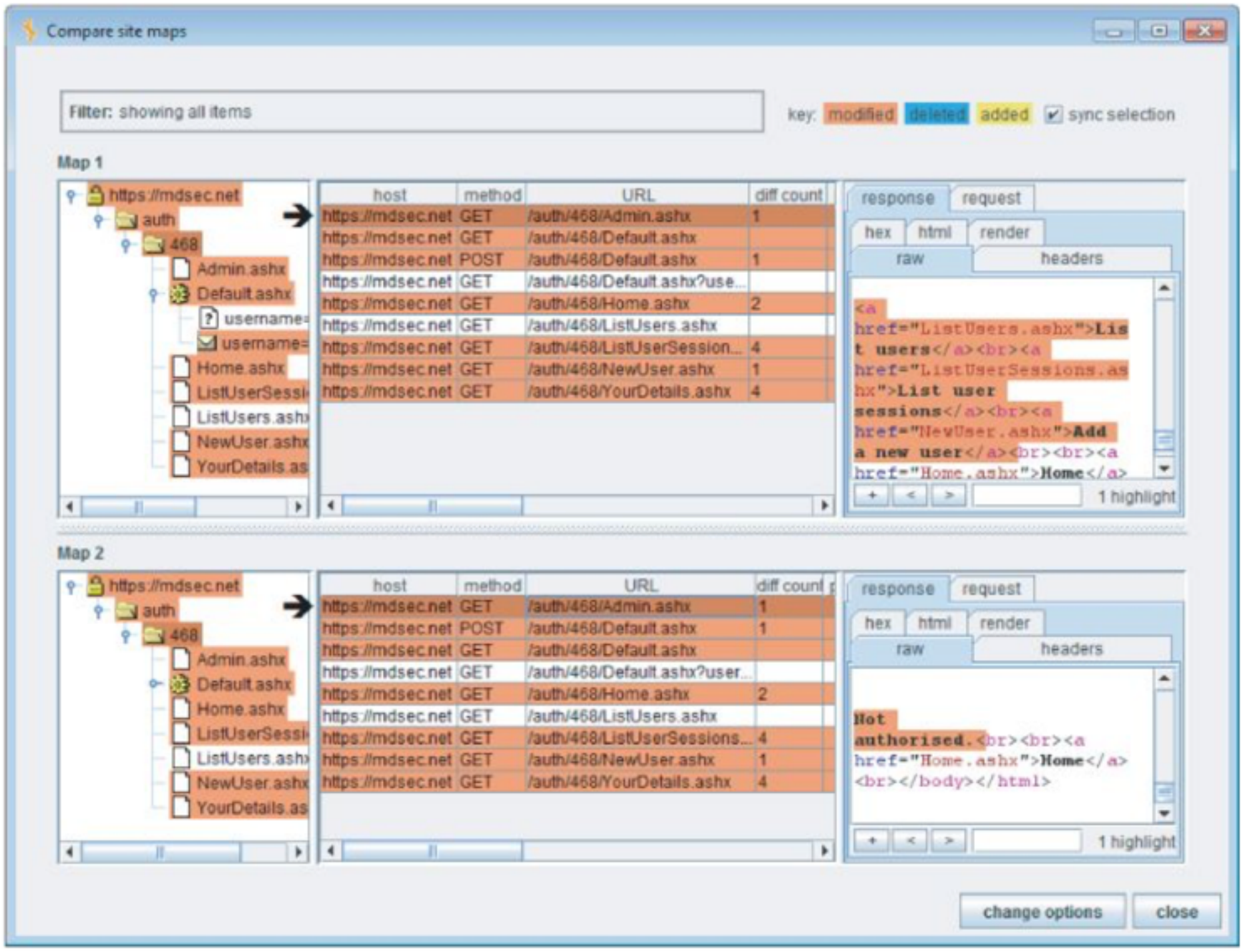
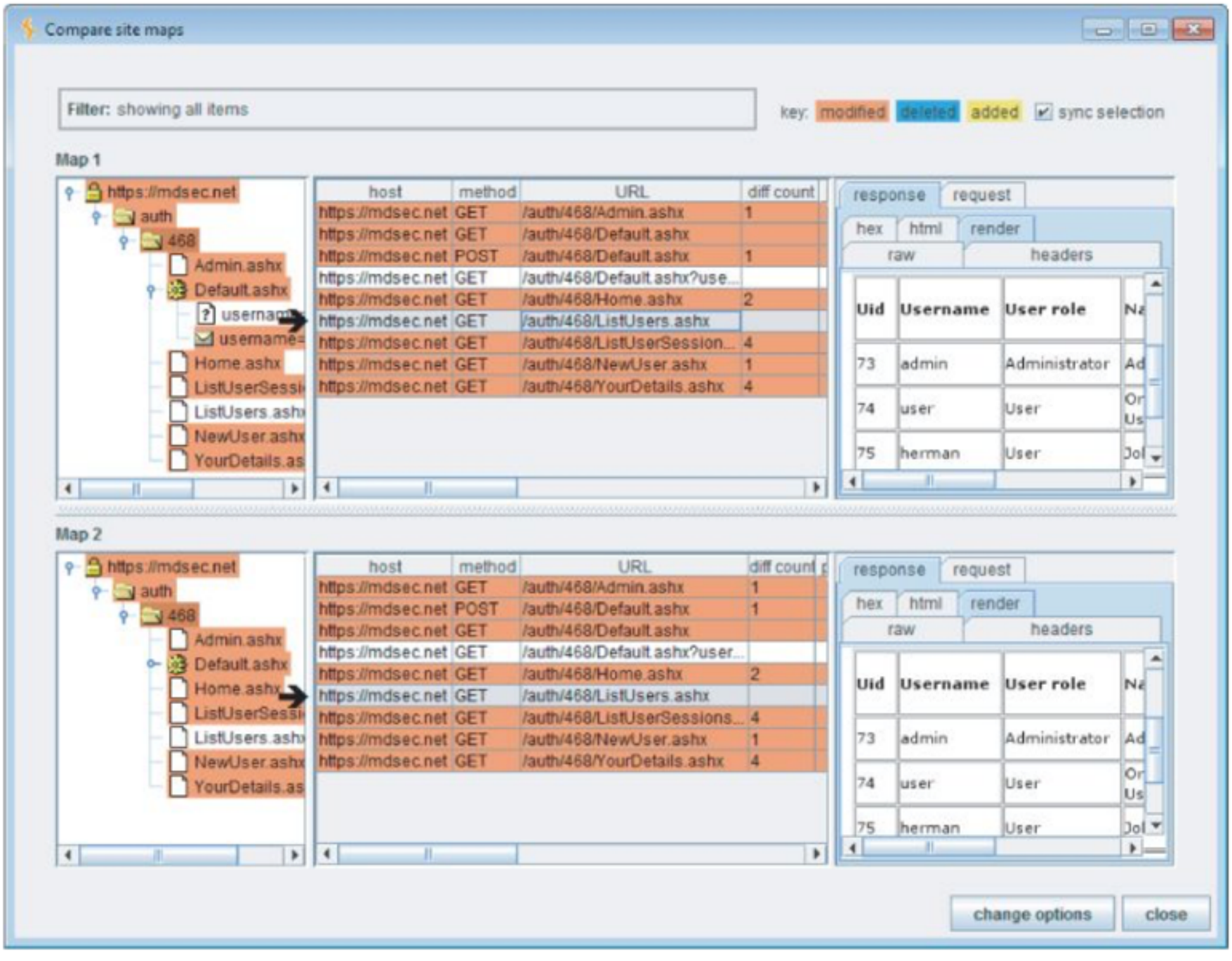


**Figure 8.3** The low-privileged user can access the administrative function to list application users



# Testing Direct Access to Methods

- **You may be able to guess other methods from the ones you see**
- **Test them to see if access is properly limited**
- **This request indicates use of the IBM HTTP Server (link Ch 8a)**

```
POST /svc HTTP/1.1
Accept-Encoding: gzip, deflate
Host: wahh-app
Content-Length: 37

servlet=com.ibm.ws.webcontainer.httpsession.IBMTrackerDebug
```

# Testing Controls Over Static Resources

- **Walk through the app while logged in as Administrator**
- **Note the URLs of high-privilege resources**
- **Log in as a low-privilege user**
- **Return to those URLs to see if you can still access them**
- **If you can, try to guess other sensitive URLS from the pattern of the ones you have found**

# Testing Restrictions on HTTP Methods

- **Log in as Administrator**
- **Find sensitive requests**
- **Try other methods: POST, GET, HEAD, invalid**
- **If other methods are honored, test them with a low-privilege account**

# Securing Access Controls

- **Don't rely on user's ignorance of URLS or identifiers like document ID2**
- **Don't trust any user-supplied parameters like admin=true**
- **Don't assume users will access pages in the intended sequence**
- **Don't trust the user not to tamper with data; revalidate it before using it**

# Best Practices

- **Explicitly evaluate and document access control requirements for every unit of application functionality**

- **Drive all access control decisions from the user's session**

- **Use a central application component to check access controls; use it for every client request; mandate that every page must include an interface to this component**

# Best Practices

- **For particularly sensitive functionality, further restrict access by IP address**
- **Protect static content by**
  - **Passing filename to a server-side page that implements access control logic, or**
  - **Use HTTP authentication or other features of the application server to restrict access**

# Best Practices

- **Don't trust resource identifiers from the client--validate them on the server**
- **Consider requiring re-authentication or two-factor authentication for security-critical application functions, such as creating a new payee**
- **Log events using sensitive data or actions**

# Advantages of Central Access Control

- It increases the clarity of access controls within the application, enabling different developers to quickly understand the controls implemented by others.
- It makes maintainability more efficient and reliable. Most changes need to be applied only once, to a single shared component, and do not need to be cut and pasted to multiple locations.
- It improves adaptability. Where new access control requirements arise, they can be easily reflected within an existing API implemented by each application page.
- It results in fewer mistakes and omissions than if access control code is implemented piecemeal throughout the application.

# Multilayered Privileges

**Figure 8.6** A privilege matrix for a complex application

| | | | Application Server | Application Roles | Database Privileges | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **User type** | **URL path** | **User role** | Search | Create Application | Edit Application | Purge Application | View Applications | Policy Updates | Rate Adjustment | View User Accounts | Create Users | View Company Ac | Edit Company Ac | Create Company | View Audit Log | Delegate privilege |
| Administrator | /* | Site Administrator | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| | | Support | ✓ | | ✓ | | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | | |
| Site Supervisor | /admin/*<br>/myQuotes/*<br>/help/* | Back Office – New business | | ✓ | | | ✓ | | | | | | | | | |
| | | Back Office – Referrals | | ✓ | ✓ | ✓ | | ✓ | ✓ | | | | | | | |
| | | Back Office – Helpdesk | ✓ | | | | ✓ | | | ✓ | | ✓ | | | ✓ | ✓ |
| Company Administrator | /myQuotes/*<br>/help/* | Customer – Administrator | | ✓ | ✓ | ✓ | ✓ | | | | ✓ | ✓ | ✓ | | | ✓ |
| | | Customer – New Business | | ✓ | | ✓ | ✓ | | | | | | | | | |
| | | Customer – Support | ✓ | | | | ✓ | | | ✓ | | | | | | |
| Normal User | /myQuotes/dash.jsp<br>/myQuotes/apply.jsp<br>/myQuotes/search.jsp<br>/help/* | User – Applications | ✓ | ✓ | | | ✓ | | | | | | | | | |
| | | User – Referrals | | | | | | | | | | | | | | |
| | | User – Helpdesk | | | | | | | | | | | | | | |
| | | Unregistered (Read Only) | ✓ | | | | ✓ | | | | | | | | | |
| Audit | (none) | Syslog Server Account | | | | | | | | | | | | | ✓ | |

https://attack.samsclass.info/phpfail.htm

# Sam Bowne

# Vulnerable PHP Examples

## 1. Weak Typing

**Log In:**

Username: samtest Password: •••••••

Log In

***Goal: log in as root***

The PHP uses this comparison:

```
md5($p) == '0e199122341212509014562288726851'
```

You can log in with a password of 240610708 even though it has a hash of **0e462097431906509019562988736854** because PHP interprets the hashes as numbers equal to zero.