

## Auditing Web Site Authentication, Part Two

*Mark Burnett* 2003-05-05

Inadequate user security is a problem that Web developers must address. Perhaps it is lack of standards. Perhaps it is a lack of auditing. This is the second part of an article addressing both of those issues by establishing a standard audit procedure by which to measure your own security. Test this list of questions against your own Web site's authentication scheme and see how it stands. The [first article](#) focused on issues surrounding usernames and passwords. This article will explore issues surrounding user privacy, session authentication, user security, and cookies.

### User Privacy

#### **Does the system transmit user credentials over a secure channel?**

Whenever users log in, set personal information, answer secret questions, etc., always be sure to use SSL for the connection. Further, make sure that you specify a unique realm for all SSL pages to be sure that the information is not revealed when browsing to non-encrypted pages.

Many Web sites such as Hotmail and Yahoo! do provide secure login pages, but this is not the default option; users must click on another link to log in securely.

#### **Are passwords stored using a strong hashing algorithm?**

You should never store a user's password in plain text. At a minimum you should use as strong encryption algorithm. The best thing to do is not store the password at all, but create a hash of the password using a message digest or one-way hashing algorithm. These algorithms derive a hash that cannot be used to determine the original password. The occurrence of any two passwords creating the same hash is extremely rare, so this method is very secure. To verify a password, you hash the user's input and compare it with the hash stored in the database.

In most cases, a Web site does not need to know a user's password; they just need to know that the user knows it. Hashing algorithms accomplishes that.

### Session Authentication

#### **Can anyone bypass authentication by accessing a module directly?**

To avoid backdoors, make sure that every protected resource authenticates the user's session. Never assume that a user will follow the correct path to a protected resource and never assume that an obscure filename will protect from unauthorized access.

### **Does the system allow authentication to the Web server's operating system or network?**

Many Web authentication systems are tied to operating system accounts. While this may be convenient for some purposes, you must carefully consider the consequences of doing this. For example, an attacker may be able to use your authentication system to guess passwords on administrator or other privileged accounts. They may also be able to lock out or otherwise impair important system accounts.

If you do use operating system accounts, limit the scope of logins through carefully planned user groups, organizational units, or even using separate domains.

### **Do users have the option to log out of their session?**

Always provide a conspicuous log-out link that abandons the session and destroys any session tokens. Make sure that users know that the safest way to end a session is to click on the log-out link.

### **Does the system destroy session tokens upon logging out or timing out?**

I have seen systems that delete any existing session tokens for users after they are authenticated. But you should also delete session stale tokens. Even better, add a timestamp to session tokens so that they expire after a period of time, even if the session has not timed out from inactivity. Every Web page should check the validity of a session token and require re-authorization if necessary.

### **Does the system prevent account hopping?**

A common programming flaw is to check a user's log in, then set a flag that indicates that the user is authenticated. Instead, you should generate a session token that is only valid for that user and check that token on each page. Otherwise, users may be able to switch to another

user's account through URL or cookie tampering. For example, consider this URL:

```
http://www.example.com/users/accountinfo.asp?userid=1254
```

My first thought when seeing a URL like that is to see what happens if I change the userid value to something like userid=1253, or even better, userid=1. If this system only looks at an authentication flag, it might let me view the details of someone else's account.

## User Security

### **Can a third party trick users into authenticating to fake login pages?**

Sites such as eBay, Hotmail, and PayPal have a big problem with users getting their login information hijacked because someone else tricks the user into logging in to a fake login screen that collects usernames and passwords. While this is in part owing to browser flaws, poor client configuration, and naive users, your site design and security policies can help minimize this. For example, your site should have a single, dedicated, and simple URL for user logins. For instance, consider some of these login URLs:

```
http://www.citibank.com/domain/redirect/signon/myciti.htm?  
BVE=http://Web.da-us.citibank.com&BVP=/cgi-  
bin/citifi/scripts/&BV_UseBVCookie=no&US&_u=visitor
```

```
https://www.aa.com/apps/utility/logInOut/LoginMember.jhtml;jsessionId=  
0CC3OQ2ND1JEAJNDSNAEQBFFUGLTT?  
anchorEvent=false&_requestid=316005&_requestid=316005
```

```
http://www.amazon.com/exec/obidos/flex-sign-in/ref=ya_hp_pi_1/103-  
5972766-0490219?opt=a&page=help/ya-sign-in-  
secure.html&response=account-name-change&method=GET&return-  
url=account-name-change
```

What I would prefer is to see URL's such as these:

<http://login.citibank.com/>

<http://login.aa.com/>

<http://login.amazon.com/>

None of these latter URLs actually exist; but which do you think would be easier to spoof? If Web sites would use simple login URLs, only allow users to log in from those URLs, and educate users of this fact, account hijacking could be reduced.

Another thing you can do to help the problem is take preventative measures on your end to monitor referrer strings on your login pages. Watch for common red flags such as concurrent logins from multiple IP addresses, and educate users.

### **Do you educate users on how to protect their account information?**

Every Web site should have a security page to teach users how to protect their account information and to understand your security policy. For example, your security page could tell users that you will never ask them to log in from an e-mail and that no one from your company will ever ask them for a password. Explain that the address bar should always show a simple login URL as described above. Users should also receive tips on how to select strong passwords and how to avoid common Internet scams.

### **Do you let users see their account history?**

An excellent way to catch intruders is to get the users involved. For instance, upon logging in you could show users the last few login times, complete with IP addresses. Users should also have the option to view a detailed account history showing detailed login information, account modifications, password changes, etc.

### **Can users easily report security incidents?**

Users often notice suspicious behavior with their accounts but do not know what they should do about it. Every Web site should have a clear method for reporting suspicious activity and you should encourage users to actively report security incidents.

### **Can users customize their security options?**

If you can't find the right balance between user security and user friendliness, let users decide for themselves. For example, users could set their own session timeouts, IP restrictions, failed logins allowance, etc. Yahoo! has an option that allows users to select how often they are prompted for a password and Hotmail.com let's users select their session timeout. This allows your users to be more secure if they want more security without affecting beginner users.

### **Can users revoke or delete accounts?**

If you allow users to create accounts, you should also allow them to easily close those accounts.

### **Cookies**

#### **Will a hijacked cookie allow a user logon?**

Writing a persistent cookie to a user's system may be convenient but it is also a great security risk. If an attacker is somehow able to hijack a user's cookie, the user's account will be compromised. Other alternatives you may consider are to only provide limited account access without re-authentication or to tie the cookie to other identifying information such as IP address, User Agent string, etc. Nonetheless, all cookies should have a reasonable fixed expiration date that requires re-authentication. Ultimately, the best solution is to require a login every session.

Account hijacking can be a serious issue for users but can affect you as well. Say, for example, that a user places a large order but later says it was fraudulent; that someone else hijacked his or her account. If you require a login each session, it becomes more difficult for users to repudiate transactions on their account.

Some of the world's largest Web sites have an option to automatically log in from a cookie. If you do this, keep in mind that the person logging in could either be the user, anyone who knows the user's password, anyone who has physical access to the user's computer, anyone with administrative network access to the user's computer, anyone who has broken in to the user's computer, or anyone who can see the network to sniff traffic.

If you want to ignore this advice and allow automatic log in using cookies, at least give users the option to not save or erase cookies. You can erase a user's cookie by sending them a blank,

expired cookie.

### **Is sensitive information stored in cookies?**

Account login is not the only risk with cookies. Many sites store personal account information in cookies. Never store actual user information in cookies; rather, store a token that points to user information on the server's database.

### **Are cookies marked as secure?**

According to [RFC 2109](#), cookies can be marked secure, preventing their transmittal to non-SSL Web pages. But many Web sites overlook marking cookies as secure. The problem with this is that if a user browses from a secure page to a non-secure page on the same site, the browser will continue to send the cookie. If a cookie is marked as secure, the browser will not send it to non-secure pages.

### **Do cookies specify a domain and path?**

Cookies also have Domain and Path properties to limit a cookie's scope. If you fail to set boundaries for cookies, it may be possible for an attacker to exploit a cross-site scripting flaw on another Web page or even another server to hijack a user's cookie.

### **General Principles**

Beyond these questions you should step back and ask yourself if your Web authentication system follows generally accepted best practices. For instance, does the system use multiple layers of defense? Does it properly limit the scope of trust? Is it simple and does it maintain a minimal attack surface? Does the system properly prevent spoofing, tampering, repudiation, information disclosure, denial of service, and privilege escalation attacks?

Despite the dramatic increase in security education and training over the last few years, passwords continue to be security's weakest link. Your Web application must do its part in keeping passwords secure. Some may argue that the principles in this article are a bit extreme, and perhaps it is just wishful thinking on my part for suggesting them, but they are a standard by which you can measure your own security. Obviously, you must take into consideration the unique risks and exposures of your own Web site. If you use user logins as a casual means to identify users, maybe all these security measures do not make sense. But if you are a financial,

medical, educational, or government institution, maybe you should take a good hard look at your authentication system.

## Relevant Links

[Auditing Web Site Authentication, Part One](#)

*Mark Burnett*

[Privacy Statement](#)

Copyright 2006, SecurityFocus