Avoid Session Management Security Pitfalls

Here's where some current methods fall short, and what you can do to ensure secure session management.

By Bill Jaeger, Contributing Writer

Can you improve your session management security?

Yes!

- Avoid basic authentication
- You can enhance existing methods
- There are proven cryptographic techniques for generating session identifiers

But...

- Secure session management is tricky
- Development teams may lack awareness
- There are risks inherent in the typical methods used

Do you remember the old Atari video game "Pitfall"? Even if you don't, I'm sure you'll commiserate with me when I say I've felt like Pitfall Harry as I swing through the e-business jungle, trying to help customers avoid the known security dangers and new surprises that have a way of popping up on the Internet.

Secure session management—ensuring only properly authenticated users can access your e-business—is one such known danger that's recently surprised a lot of people.

Why session management?

Before I get into the security aspects of session management, let's talk a bit about how the Web works, and why session management is even necessary.

As you know, e-business applications are quite different from traditional client-server applications in many ways. One of the most significant differences is that the Web is inherently stateless, with each and every page, graphic, or other request treated as a new, unique connection with no relationship to previous or future connections. One user browsing one Web page 100 times looks no different than 100 unique users browsing that same page. Compare that to, say, a telnet session in which a connection is established and remains established until explicitly terminated.

The stateless Web is of little use to many e-businesses, since features such as logins, customization, and shopping carts all require state. That is, one request has some dependence upon another.

EXAMPLES: Checking out at an online store first requires that something be placed into a shopping cart. Or displaying a user's customized portal interface requires the portal to uniquely identify the user and recall previously set profile information.

To eliminate problems, companies apply the client-server concept of a session to e-business, using several means to simulate a stateful session atop a standard, non-stateful Web request. Unfortunately, those simulation techniques often aren't designed with an eye toward security. They may leave your e-business exposed and users open to fraudulent use of their accounts.

REALITY CHECK: Interestingly, this problem isn't the exclusive domain of small or non-security conscious organizations. In late September 2000, for example, it was discovered that E*Trade's session management implementation was insecure, and that customer accounts could be compromised. Although E*Trade claimed there was no security risk, just days after the vulnerability was announced, they changed the way they generate sessions.

State mechanisms

There are a number of different ways to simulate state in e-business applications. The three most common methods are basic authentication, URL embedding, and cookies.

Continued

Contributing Writer Bill Jaeger is director of applied research at METASeS, a full service provider of end-to-end e-business security solutions and services, including the SecuritE-STAFF.com security knowledge management portal. http://www.metases.com.

Each of those methods basically does the same thing—they let the Web browser client transmit additional, custom information to the e-business server with each and every page, graphic, form, or other request.

For basic authentication, this additional information is the user's credentials (such as user ID and password); for URL embedding and cookies, this additional information is typically known as a session ID. Tracking this additional information on the server side, and associating it with user actions, lets e-businesses simulate state.

Security is an issue if another user, whether malicious or innocent, can guess (or intercept) the credential or session ID information. Effectively providing that information to the server lets one user access another user's session.

Let's take a look at each of these state mechanisms in more detail, and consider their security ramifications.

Basic authentication

Virtually every Web browser and server builds in support for basic authentication. When accessing a site that uses basic authentication, the user is prompted to enter a user ID and password that's valid for a specific security realm. Once entered, these credentials are remembered by the Web browser and provided to any Web server that exists within the same security realm for each and every request.

CAUTION: Although extremely simple to implement, basic authentication isn't secure. In particular, basic authentication suffers from these major problems:

- The complete user ID and password is transmitted in base64-encoded plain text with each and every request to a given security realm as part of the HTTP header. Because these credentials are in plain text, it's trivial for a malicious user to intercept the credentials when used over non-SSL encrypted connections.
- Web servers and interim proxy servers can be configured to log user credentials and other information provided as part of the HTTP header. This means a user's full credentials may be unknowingly sitting in a log file on an untrusted proxy or Web server and subject to being revealed through server misconfiguration, lack of discretion on the part of the server owner, or through hacking. If intercepted, the user's credentials can potentially be used to provide unfettered access to the e-business.

BOTTOM LINE: Basic authentication provides no mechanism for the server to expire a user's session, placing the onus on the user to shut down his Web browser or reboot his computer to clear credentials from the Web browser.

To illustrate the potential dangers of basic authentication and the damage that can be caused if a user's credentials are intercepted, consider Morgan Stanley Dean Witter's RetireView 401(k) management Web site, which uses basic authentication to authenticate users based upon social security and personal identification numbers. Other uses of social security information aside, those same credentials are also used for telephone account access.

TIP: It's best to avoid basic authentication as an authentication or session management mechanism due to its inherent security risks.

Security is an issue if another user, whether malicious or innocent, can guess (or intercept) the credential or session ID information.

Session IDs

Session IDs are an application-level construct whereby you construct some sort of unique identifier and assign it to each user to differentiate one from another. For sites that require users to authenticate themselves using a mechanism other than basic authentication (such as form-based login), the session ID effectively becomes a surrogate for the user's credentials for as long as the session ID is valid.

CAUTION: Generally speaking, session IDs have the potential to provide a fairly secure means of enabling state. Unfortunately, though, many session ID methods suffer from insecurities, including:

- Easily-guessed session IDs, such as sequentially increasing integers. Malicious users can easily modify such session identifiers in an attempt to find valid sessions. It may also be trivial to use widely available or easily written programs to conduct an exhaustive search to find valid session IDs.
- The small size of largely invariable session IDs that can be easily guessed or brute forced by an exhaustive search. Recent reports indicate that Charles Schwab's online investment service suffers from this problem.
- Trivially encoding user credentials within the session identifier. This is no better than basic authentication.
- Not having the server expire and no longer honoring the session ID after some time period, such as no more than six hours after login or 15 minutes of inactivity.
- Not having the server generate a new session ID for every login, and instead permanently associating a session ID with a user.

Session identifiers pose other security risks depending upon whether they're embedded within URLs or implemented as cookies.

URL embedding

URL embedding refers to placing a session ID within a URL. Companies often do this to eliminate the need for cookies, given public outcry and misunderstanding surrounding their use.

CAUTION: URL embedding, however, poses the following security risks:

- When following links on a Web site, the URL of the referring location (for example, the URL containing the link that was clicked on) is stored in the log files of Web and interim proxy servers. If the URL contains the session identifier, it will be included in the log files of those servers.
- Users frequently copy-and-paste URLs of interesting Web pages and bookmark them, send them to friends, or post them in online forums. In cases where users provide these session ID-containing URLs to others, users may be

unknowingly giving full access to their accounts to strangers or malicious users. I've seen this occur many times in online forums where users have unwittingly exposed their true identities, addresses, and credit card information.

Cookies

Storing session ID information in cookies also poses certain security risks, largely due to implementation flaws in application and Web server software. Risks include:

- Cookie contents are susceptible to being revealed to a malicious user through well-known cross-site scripting JavaScript vulnerabilities. This allows cookie contents to potentially be retrieved under certain conditions through HTML-enabled e-mail packages such as Outlook Express, or through Web sites that allow users to submit and subsequently display HTML. Charles Schwab's online investment service, IBM, public message forums, several Web mail providers, and many other Web sites are susceptible to this problem.
- Permanent, or disk-based, cookies can be easily retrieved from public terminals, through poor system security, or through operating system defects.

TIP: Memory-based cookies are arguably the most secure means of storing session identifier information.

A better way

Often, development teams use poor session-management techniques simply because they aren't aware of (or don't understand) the security issues.

Follow this checklist of session management best practices:

1. Don't use long-term secrets such as user IDs and passwords as session identifiers, even if encrypted. Instead, use strong, random tokens that offer no value to other users or Web sites.
2. Use proven cryptographic techniques for generating session identifiers to ensure that they can't be easily guessed or modified.
   EXAMPLE: A recent posting by Alan DeKok to the BUGTRAQ mailing list suggested this formula for generating session identifiers:

   ```
   cookie = md5(secret + md5(secret + expiration + user-IP
   + user-ID)) + expiry + user-ID
   ```

   - md5 is the message digest 5 cryptographic hash algorithm
   - + denotes string concatenation
   - secret is a frequently changed value known only to the Web server(s)
   - expiration is the timestamp when the session expires
   - user-IP is the user's IP address
   - user-ID is a unique user identifier.

   See the sidebar for additional information on md5(), cryptographic hashes, and the BUGTRAQ mailing list.
   This formula effectively binds the cookie to one—and only one—login session so the cookie is of no value to anyone but the legitimate user. There are other, similar approaches you can take to generate strong cookies, but this example serves as a good illustration to show you what's possible and what you should consider when generating secure session identifiers.

To learn more

For information on cryptographic techniques, see:
RSA Laboratories' Frequently Asked Questions About Today's Cryptography
http://www.rsasecurity.com/rsalabs/faq

Applied Cryptography,
by Bruce Schneier (John Wiley & Sons, 1995)

For information on cross-site scripting vulnerabilities, see:
CERT Advisory CA-2000-02
"Malicious HTML Tags Embedded in Client Web Requests"
http://www.cert.org/advisories/CA-2000-02.html

Apache Cross-Site Scripting Info
http://www.apache.org/info/css-security/index.html

Microsoft Information on Cross-Site Scripting Security Vulnerability
http://www.microsoft.com/technet/security/crssite.asp

For information on the BUGTRAQ mailing list, see:
SecurityFocus
http://www.securityfocus.com

3. Don't "invent" cryptographic or other obfuscation techniques for hiding cookie information. It will be discovered.
4. Expire all session identifiers after both a) an absolute time (such as 6 hours); and b) after a period of inactivity (such as 15 minutes). This greatly reduces the period of exposure should someone compromise a user's session ID.
5. Provide a logout function so users can voluntarily invalidate their session identifier when they're through with their session.
6. Don't rely upon Web browser clients to expire or invalidate cookie information; be sure to do this on the server side.
7. Validate session identifier information returned by the client to ensure that it's still valid (for example, it hasn't expired, it passes cryptographic checks, etc.). If the session identifier is invalid, require the user to re-authenticate.
8. Use memory-based, not disk-based, cookies for all security-sensitive session information.

Crossing to safety

Secure session management can be tricky, with many traps and pitfalls around every corner. A lack of security knowledge and understanding of how the complex Web of technologies can be exploited can easily leave your e-business exposed to security compromise if appropriate safeguards aren't implemented.

Despite the numerous security risks surrounding session management, there are simple methods you can use to greatly enhance session management security.

ASK YOURSELF: What dangers are lurking in your e-business session management practices? ADVISOR