Session Management in Web Applications

Author:
EUROSEC GmbH Chiffriertechnik & Sicherheit
Tel: 06173 / 60850, www.eurosec.com

What is Web-based Session Management

- user authentication management
- user preferences management
- site history of previously seen contents
- authorization and status information (e.g. “valid customer”, “shopping basket items”, etc.)
- trust delegation within (distributed) web applications

Usual Technical Solutions (Overview)

- Cookies
- URL-encoding
- Forms with Hidden Field
- HTTP Referrer Fields
- HTTP Basic/Digest Authentication
- Client Certificate

Neglected Security Requirements

- Session time-out after a certain amount of time
- prevention/detection of brute force attacks and/or password guessing
- prevention of unauthorized manipulation of session data
- prevention of theft of sensitive session tokens and user credentials
- servers should be able to terminate sessions in case of necessity (independent from client session token validity)

translates to:

- session logout mechanisms that delete (overwrite) the browser`s session cookie
- in case of necessity: use of strong (pseudo-) random numbers
- sufficient session ID randomness
- sufficient session ID length
- use different session identifiers when shifting between secure and insecure contents (i.e. authenticated vs. non-authenticated, http vs. https, sensitive application areas vs. nonsensitive, etc.)
- limited trust delegation
- reauthentication prior to sensitive actions

Popular Problems

- secret user credentials are transmitted in the clear
- server-generated authentication data can be forged (weak crypto, exhaustive search, simple-to-guess numbering or mechanism)
- stealing of secret user credentials
- no account logout
- .....

Active and Passive Attack Scenarios

- Session Hijacking
- Eavesdropping
- Session Fixation
- Cross-Site Scripting
- Exhaustive Search
- Intelligent Manipulation of Session Data
- Network Manipulation (e.g. change of DNS entries)
- etc.

URL-based Solutions

- The session token is part of the URL and will be transmitted to the web server through HTTP GET requests
- Example: http://www.blabla.com/buy.asp?article=27781;sessionid=IE5579901578
- This mechanism works even in case of user client security restrictions (if, for example, no cookies are allowed). Thus, it can serve as a fallback mechanism if the server detects problems with other mechanisms like cookies or JavaScript

Problems:

- All information contained in the URL might be stored in firewall or proxy log files
- Also, it could simply be printed out with the web page on a shared printer
- Moreover, the URL, including potentially sensitive data, can be sent in the HTTP referrer field to other web servers. This implies a high risk of sensitive session data being disclosed to unauthorized third parties

Form-based Solutions with Hidden Fields

- Session token and information can be included in so-called hidden fields in a form
- The form will be sent by the user client to the server with an HTTP POST command. This looks like the following example:

```
<FORM METHOD=POST ACTION=”/cgi-bin/order_goods.pl”>
<INPUT TYPE=”hidden” NAME=”customerid” VALUE=”0815”>
<INPUT TYPE=”hidden” NAME=”valid_order” VALUE=”yes”>
<INPUT TYPE=”hidden” NAME=”productid” VALUE=”4711”>
```

- As described in the case of URL-encoded session tokens, this form-based mechanism works even in case of user client security restrictions (if, for example, no cookies are allowed). Thus, it can serve as a fallback mechanism if the server detects problems with other mechanisms like cookies or JavaScript
- The form-based solution does not provide pre-defined protection mechanisms as those provided by cookies, for example
- Moreover, the form-based approach requires repeated sending of HTTP forms via POST commands, which implies a certain amount of overhead that might reduce performance. Also, hidden fields cannot last beyond a certain interactive session

HTTP Basic and/or Digest Authentication-based Solutions

- In HTTP Basic authentication, the client sends his username and password in cleartext as part of the HTTP request
- In all subsequent HTTP requests for content from subdirectories of the original request, these credentials will be automatically resent
- In HTTP Digest authentication, no passwords are sent in the clear
- Instead, a cryptographic hash value containing the username, password, and additional security-relevant data, will be transmitted from the client to the server (more details can be found in *RFC2617 - HTTP Authentication: Basic and Digest Access Authentication*, June 1999)

Problems:

- HTTP Basic authentication is vulnerable to passive eavesdropping. Moreover, it provides no mechanism for explicit session expiration (i.e. logout)
- HTTP Digest authentication cannot guarantee sufficient support on all client platforms
- Both mechanisms do not provide session tracking, but only authentication

Cookie-based Solutions

- Cookies are a simple session management mechanism
- Cookies have been designed to track status information about user visits on web servers (who requested which pages at what time)
- Originally, nobody intended to use cookies for authentication purposes
- Nevertheless, a large quantity of web applications today uses cookies for user authentication and authorization
- Cookies are normally sent from the web server to the client using two different methods: either within HTTP headers or by JavaScript

Structure of a Cookie

The syntax for the Set-Cookie2 response header, according to RFC2965 syntax notation, is as follows:

```
set-cookie     =   "Set-Cookie2:" cookies
cookies        =   1#cookie
cookie         =   NAME "=" VALUE *(";" set-cookie-av)
NAME           =   attr
VALUE          =   value
set-cookie-av  =   "Comment" "=" value
               |   "CommentURL" "=" <"> http_URL <">
               |   "Discard"
               |   "Domain" "=" value
               |   "Max-Age" "=" value
               |   "Path" "=" value
               |   "Port" [ "=" <"> portlist <"> ]
               |   "Secure"
               |   "Version" "=" 1*DIGIT
portlist       =   1#portnum
portnum        =   1*DIGIT
```

Cookies: the “Secure” Option

- The “Secure” option tells the user client that the cookie in question should be sent over an SSL-protected channel
- At least this is what common web browsers do, even though the RFC2965 makes a slightly different statement: “The Secure attribute (with no value) directs the user agent to use only (unspecified) secure means to contact the origin server whenever it sends back this cookie, to protect the confidentiality and authenticity of the information in the cookie. The user agent (possibly with user interaction) MAY determine what level of security it considers appropriate for "secure" cookies. The Secure attribute should be considered security advice from the server to the user agent, indicating that it is in the session's interest to protect the cookie contents. When it sends a "secure" cookie back to a server, the user agent SHOULD use no less than the same level of security as was used when it received the cookie from the server.”

Cookies: Further Options

- The **“Discard”** option tells the user client to discard the cookie as soon as the user client terminates
- The **“Domain”** and **“Path”** values specify the domain and corresponding folder subtree to which the cookie applies
- The **“Max-Age”** option allows to set an explicit cookie life time. If this life time is over or if “Max-Age” is set to zero, then the cookie should be discarded by the user client. Thus, the origin server can effectively terminate a session by sending the client a corresponding Set-Cookie2 header with Max-Age=0
- If no “Max-Age” option is present, then the cookie is a so-called session cookie, i.e. the user client is supposed to store the cookie temporarily in its memory space and discard the cookie as soon as the user client terminates (i.e. the browser is closed)
- The **“Port”** option allows to define valid port numbers to which a given cookie can be returned. The security relevance of this feature is of minor relevance, nevertheless it could be useful in some specific scenarios

Persistent Cookies

- Persistent cookies will be stored (in case of Microsoft Windows2000/XP, for example) in C:\Documents and Settings\<username>\Cookies
- Each such cookie is stored in a .txt file starting with the username, followed by an "@" and some information about the issuer domain
- Persistent cookies have no protection from the operation system against user manipulations

Microsoft`s HttpOnly Option

- Microsoft implemented protection mechanisms into Internet Explorer 6, the so-called HTTP-only option
- HttpOnly is not part of an RFC or similar standard
- „; HttpOnly“ gets appended to a cookie; if IE6 receives such a cookie, it does not allow scripted (e.g. via JavaScript) access to this cookie
- other browsers may either ignore this option or even reject the whole cookie
- Http-only mechanism is good, but not absolutely secure against all sorts of scripting attacks

How to change Cookies or Hidden Fields

Changing cookies, URLs or hidden fields is quite simple; all you need is (one of) the following:

- local proxy (e.g. Paros)
- text editor, for persistent cookies stored in text files
- telnet or other simple Unix tools
- in case of URLs you need no editor at all...
- sometimes: Base64 decoder

Example: Apache

- Apache provides a so-called user tracking mechanism with mod_usertrack.c (in previous versions mod_cookies.c)
- Even though the source code documentation in Apache version 2 explicitly warns **not** to use this mechanism for security purposes like authentication, many application programmers still do (who cares about documentation?!…)
- The generated cookies include the client`s IP address, the web server`s process ID, as well as the time of the web request. An example cookie looks like this: Apache=”64.3.40.151.27273996348638848”

Ad Hoc Measures / Recommendations

- use session identifier strings that cannot be efficiently guessed or otherwise computed
- use long (pseudo) random numbers within session identifiers
- do not store session status information in cookies and/or URL strings or hidden fields
- use only a session identifier on the client side that refers to all relevant information that is stored on the server side
- in case that session information must be stored on the client side, then it should be encrypted and/or cryptographically hashed
- change session identifiers whenever the user switches between authenticated/unauthenticated, secure/insecure areas, etc.
- make use of cookie security options (as described above)
- use URL-/Cookie-/Hidden Field-based session identifiers in parallel (each using a different ID string)
- make use of the HTTP referrer field as an *additional* security feature: deny requests with wrong referrer data; depending on the application scenario, however, it could be necessary to allow requests with missing referrer data
- divide the application into two parts on separate servers: one for "insecure" HTTP requests, the other for "secure" HTTPS requests; include corresponding path restrictions in the cookie path option

Dangerous Confusion among old and new RFC (?)

Example for one (among others) problem in "HTTP State Management Mechanism" RFCs:

- Interpretation of the Path attribute according to RFC2109 from 1997:
- "Defaults to the path of the requesting URL that generated the Set-Cookie response, up to, but **not** including, the right-most /."
- Interpretation of the Path attribute according to the successor of RFC2109, i.e. RFC2965 from 2000:
- "Defaults to the path of the requesting URL that generated the Set-Cookie2 response, up to **and** including the right-most /."
- This is a quite important difference, as can be seen from the following examples:
- www.shared-host.example.com/thilo/
- www.shared-host.example.com/thilo-the-victim/
- Setting a path attribute "/thilo" would, according to the new standard, not allow the area "/thilo-the-victim" to access cookie information from "/thilo". If the old standard applies, however, this would be allowed.

Stronger Improvements: Dynamic Links

- Idea: generate dynamic links that include session identifier information (e.g. user ID and previous page)
- different implementation variants are possible
- some use JavaScript on the client side to generate those links; this allows unauthorized user manipulations
- problems: page reload and browser „back button“ can imply difficulties

Conclusion

- many web applications still suffer from weak session management
- for „medium security“, a good cookie-based standard solution can be sufficient
- for higher security needs, a Dynamic Link-based approach is recommended
- each web application should be checked against all problem categories mentioned above
- never trust user input

Appendix

Why this presentation from us? - our professional background:

- several project years‘ in research projects on secure software development, with partners like SAP, Deutsche Bank, Commerzbank, Universities, etc.
- large amount of application vulnerability checks for software companies, including reviews and coaching for developers
- Security requirements- and design specifications for large development projects
- Writing of company guidelines and checklists for secure development, mostly in banking and finance sector
- Reverse engineering and reviews of security mechanisms and crypto algorithms
- Implementation of security functions for customers

Final Remark

- the present work has been conducted within the so-called **secologic** research project (term: 2005+2006)
- for more details see www.secologic.org
- we are grateful to the German Ministry *Bundesministerium für Wirtschaft* for supporting this project
- we appreciate all suggestions and feedback with respect to our presentation slides and white papers

Copyright Remark