PHP Session Security

Przemek Sobstel, 2007
http://sobstel.org

Table of contents

Introduction

Sessions are the way of saving the state and user specific variables across subsequent page requests. This is achieved by handing a unique and difficult-to-guess identity value (session id) to the browser (either in a cookie or the URL) which the browser submits with every new request. The session is alive as long as the browser keeps sending the id with every new request.

The ability to restrict and maintain user actions within unique sessions is critical to web security. Unfortunately security of PHP's native session handling mechanism leaves a lot to be desired and additional measures need to be taken to ensure sufficient level of session security.

Threats

Session Fixation

Session Fixation makes the user using an explicit session id provided by the attacker. The attack ensues before the user enters and logs into the website.

Esser S.: *PHP and Session Fixation*, http://www.hardened-php.net/php_and_session_fixation.48.html

Kolsek M.: *Session Fixation Vulnerability in Web-based Appliactions*, ACROS Security 2002. http://www.acros.si/papers/session_fixation.pdf

Shiflett C.: *Security Corner: Session Fixation*, http://shiflett.org/articles/session-fixation

Session Hijacking

This term generally refers to all attacks that attempt to gain access to another user's existing session by obtaining session id. The attack ensues after the user enters the website and the session is started.

The attacker can get user's session id for example by guessing it (Session Prediction), sniffing transmission (Session Sniffing), browsing session storage (Session Exposure), using Cross-Site Scripting (XSS) or reading it from HTTP_REFERER header.

Session Sniffing

Capturing session id in transit through network interception, e.g. “man-in-the-middle” method.

Session Prediction

Session Prediction means guessing a valid session id using various tools and methods (e.g. brute-force technique). The attack is possible when session id is weakly encrypted, too short or assigned sequentially.

Session Exposure

The vulnerability relates to shared hosts. PHP uses the same directory to store all session data by default (/tmp on Unix and c:\windows\temp on Windows). Session variables are saved to files which names are based on the session ids and each file contain the variables for that session in clear text. It may lead to disclosure of session data as any user can browse the filesystem and read these data. Changing directory location, although raises the bar for potential attacker, might not be sufficient as other users can still read it if they find it. Even when session data are stored in database, the leak is still possible if permissions are not handled properly, e.g. one user has access to all tables.

Session Poisoning

Session Poisoning is strictly connected with Session Exposure with the only difference that while the latter refers only to reading session data, Session Poisoning refers to adding, modifing or deleting (writing in general) the data. It can be also used to create new session, e.g. in order to carry out Session Fixation attack.

Session Injection

The threat is a result of insufficient data validation and concerns situations when session variables are created using external (user submitted) data.

Session Poisoning can be used to inject malicious code here, too. Also, the attacker can modify session id in order to execute SQL Injection or Code Injection attacks.

Insufficient Session Expiration

Sessions that do not expire on the HTTP server can allow an attacker unlimited time to guess or brute-force a valid authenticated session id and eventually gain access to that user's web accounts. Additionally, session id can be potentially logged and cached in proxy servers that, if broken into by an attacker, could be exploited if the particular session has not been expired on the HTTP server.

Best practices

Session lifetime control

- Expire session after a short period of inactivity - set the idle timeout to 5 minutes for highly protected applications through to no more than 30 minutes for low risk applications.
- Enable logout option - explicitly expire and destroy the session on logout.
- Avoid persistent logins ("remember me" option) - optionally you can constrain the information retained and revealed by the application, i.e. force the user to re-log in before performing any critical operations.

- Expire session on security error - any security error in the application should result in termination of the session.
- Expire long lasting session – force re-authentication for session, which despite being active has reached the maximal allowed time, e.g. a few hours.
- Remove session cookie when destroying a session.

Session identifier

- Use only cookies to propagate session id, because when transmitted via a URL parameter, GET requests can potentially be stored in browser history, cache and bookmarks. It can be also easily viewable then.
- Rotate session id - regenerate (replace with new one) session id, at least whenever the user's privilege level changes. Generally it can be regenerated prior to any significant transaction, after a certain number of requests or after a period of time.
- Check whether session id is valid (of expected size and type) and has been generated by the application and not provided by the user.
- Session id should be adequately long, unpredictable, hard to reproduce and created from high quality random sources.

Session cookie

- Set the domain of the cookie to something more specific than the top level domain.
- Don't store in cookie anything (at least any sensitive data like username or password) but session id.
- Set http-only flag to disable capturing session id via XSS.
- When possible, use strong encryption (SSL) as well as "secure" attribute to allow transmiting cookies only over https.

Session data storage

- Determine where the application framework stores session data and if it is filesystem or database, determine who else might have access to these data.
- When you store session data in files, ensure the application is configured to use separate private directory (e.g. session.save_path directive). Use filesystem permissions to protect these files from observation or modification by users other than the accounts required to operate the web and application servers. If this is not possible, the session data needs to be encrypted or contain only nonsensitive data. Note that PHP uses public system temporary directory by default
- All session variables must be validated to ensure that is of right form and contains no unexpected characters.

Page and form tokens

A page token (so-called “nonce”) is a unique token given when a page is downloaded and is presented by the user when accessing the next page. The server expects a particular value for the user to access the next page. Only if the token submitted matches what the server is expecting, the next page is served. An application can use this to ensure that a user accesses pages only in the sequence determined by the application. The user cannot paste a deep URL in the browser and skip pages just because he has a session id, as the page token would not be authorized to access the deeper URL directly.

Incorporate a hidden field with a cryptographically secure page or form nonce. The nonce should be removed from the active list as soon as it is submitted to prevent page or form re-submission.

Caching

Prevent client-side page caching on pages that display sensitive information.

Further reading

Alshanetsky I.: *PHP Architect's Guide to PHP Security*, Marco Tabini & Associates, Inc. 2005.

Bar-Gad I., Klein A.: *Developing Secure Web Applications*, Sanctum Inc. 2002. http://www.cgisecurity.com/lib/WhitePaper_DevelopingSecureWebApps.pdf

Murphy C.: *Security for websites – breaking sessions to hack into a machine*, (IN)SECURE Magazine 3/2006, p.18-20.

Ollmann G.: *Web Based Session Management, Best Practices in Managing HTTP Based Client Sessions,* http://www.technicalinfo.net/papers/WebBasedSessionManagement.html

PHP Manual – Security. http://pl2.php.net/manual/en/security.php

A Guide to Building Secure Web Applications, The Open Web Application Security Projects, http://www.owasp.org/index.php/Category:OWASP_Guide_Project.

The Ten Most Critical Web Application Security Vulnerabilities, The Open Web Application Security Projects 2004. http://www.owasp.org/documentation/topten.html..

License

Some parts of the document has been partly based on The Open Web Application Security Project's wiki content – www.owasp.org.