Session Security

Aleksandar Kotevski[1] and Gjorgi Mikarovski[2]

[1]Faculty of law Science – 7000 Bitola Macedonia
aleksandar.kotevski@uklo.edu.mk
[2]Faculty of Technical Science – 7000 Bitola Macedonia
gjorgi.mikarovski@uklo.edu.mk

Abstract. Sessions are one way of saving the state and some useful user information across subsequent page requests. Sessions usually are used for storing information about registered user, selected language (in situation of Multilanguage portals), item in shopping card (in situation in e-commerce) etc.

Keywords: Session,cookie,fixation, hijacking,poisoning, -site scripting,secure,http,ssl,

1 Introduction

If you working with sessions, you must know that session module cannot guarantee that the information which are store in a session is only viewed by the user who created the session, anyone with access to the server can access the PHP session files. Because of security aspect it is not clever to store critical information in session variables. Therefore, you must take additional measures to actively protect the integrity of the session, depending on the value associated with it.

2 Manipulating with session

Specific issues that may manipulate with sessions are:

- Creation and identification
- Session termination and timeout – what triggers a session termination? How are the resources of the terminated session recycled?
- Concurrency issues

From security aspect, the following considerations must be made:

- It should never be possible for one client to be able to predict the token another client received, or is in the process of receiving, or will receive.
- Furthermore, it is desirable that a client will not be able to predict the next token when he/she will get access to the site. This is useful in minimizing the damage of

stealing the token while it travels (in the clear) to and from, and while it is stored on disk from the client.
- Any token should have a reasonable expiration period – again, to minimize the damage in case of being stolen.

3 What is session variable

Session variables have the following features:

- Unless specified otherwise, session variables expire 20 minutes after a visitor leaves the site.
- Session variables will expire if no activity is detected on the site for 20 minutes by that specific site visitor, or if the visitor quits out of their web browser.
- In order for session variables to work, the visitor's browser must be set to accept cookies.
- The pages for the site all must be located within a single directory on the web server.
- Information stored in session variables is site visitor specific. Different site visitors cannot access each other's session variable information.

4 Types of session attack

There are few types of attacks that you need to be wary about when you are working with session variables:

- Session Fixation
- Session Hijacking
- Session Poisoning (injection)
- Session cross-site scripting

4.1 Session Fixation

Web session security is mainly focused on preventing the attacker from obtaining – intercepting, predicting or brute-forcing - a session ID issued by the web server to the user's browser.
The attack itself is very basic. The hacker forms a link or redirects which sends the user to your site with the session ID present:

```
<a href=http://hostname/index.php?PHPSESSID=1234> Click
</a>
```

When users click on that link or are redirected there, they connect to your site with a session ID that has been set by the attacker. The attacker can now wait for the users to log in and access your site using their credentials.

First, the attacker – who in this case is also a legitimate user of the mail system – logs in to the server and issues a session ID 1234.

Sends a hyperlink `domainane.com/login.php?sessionid=1234` to the user, trying to lure him into clicking on it. The user clicks on the link, which opens the server's login page in his browser. Note that upon receipt of the request for login.php?sessionid=1234, the web application has established that a session already exists for this user and a new one need not to be created. Finally, the user provides his credentials to the login script and the server grants him access to his email account.

Knowing the session ID, the attacker can also access the user's account via `email.php?sessionid=1234.` Since the session has already been fixed before the user logged in, we say that the user logged into the attacker's session.
To prevent this, PHP has to define a `session_regenerate_id()`. This function generates a new session file for the user, gets rid of the old one, and issues a new session cookie if your site utilizes them. Another good practice is setting a session time-out in the php.ini file.
But, unfortunately these methods do not guarantee that an attacker can't get your users' session Ids, so, you could use SSL/TSL. SSL (Secure Sockets Layer) provides one of the most commonly available security mechanisms on the Internet and it's used extensively by web browsers to provide secure connections for transferring sensitive data. An SSL-protected HTTP transfer uses port 443 and is identified with a special URL method - https.

4.2 Session Hijacking

There are three common methods for session defense:

- User agent verification
- IP address verification
- Secondary token

There are two major drawbacks to this method:

- A lot of locations are behind a NAT proxy, so it is possible that the attacker and the user both have the same IP address
- Large ISPs like AOL - a number of them, and AOL specifically, have massive proxy setups that send the user out via a different IP address with every page request.

4.3 Session poisoning

Session poisoning or session data pollution, modification, injection is to exploit insufficient input validation in server applications which copies user input into session variables.

4.4 Session cross-site scripting

Cross-Site Scripting attacks are type of injection problem, in which malicious scripts are injected into the otherwise benign and trusted web sites. Cross-site scripting (XSS) attacks occur when an attacker uses a web application to send malicious code, generally in the form of a browser side script, to a different end user. XSS is just a special case of code injection. In this type of attack, the malicious user embeds HTML or other client-side script into your Web site. The attack looks like it is coming from your Web site, which the user trusts. This enables the attacker to bypass a lot of the client's security, gain sensitive information from the user, or deliver a malicious application. There are two types of XSS attacks: reflected or non-persistent and stored or persistent.

5 Secure practice

At least, each developer needs to follow this secure practice to make the application secure:

- Set a new path for your session data storage
- Do not pass your session identifier in
- If skeptical on the entire HTTP Headers issue, use a security token at all time
- If nothing in the world could determine you to filter every incoming message than the least you can do is to use a security token at all time. A light example of such a token could be: $token = md5(uniqid(rand(), true));

6 Conclusion

- To use SSL when authenticating users or performing sensitive operations.
- Regenerate the session id whenever the security level changes
- Have sessions timeout (time for which sessions expire)
- Store authentication details on the server, not to cookie
- Lock down access to the sessions on the file system, use custom session handling
- For sensitive operations consider requiring the user to provide their authenication details

7 References

1. Ballad, W.: Securing PHP Web Applications
2. Alshanetsky, I.: PHP Architect's Guide to PHP Security, Marco Tabini & associates
3. Murphy, C.: Security for Websites – Breaking Sessions to Hack into a Machine