Preventing Http Session Fixation Attacks

Armando Romeo*

©HackersCenter

http://www.hackerscenter.com

18/12/05

* zinho-nospam@hackerscenter.com

1 Introduction

In this paper I will try to explain, from a web developer point of view, the best practices to take to prevent one of the most powerful but yet least known attacks on http sessions.

2 SESSION FIXATION

2.1 Session fixation overview

Before demonstrating the techniques to use to be secure from such attacks, let's understand how session fixation attacks are held and what a malicious user needs to mount them.

The real problem with both session hijacking and session fixation attacks are the way http protocol handles connection state: to be more specific we must say that HTTP is a steateless protocol, so it doesn't take care of the state between subsequent requests from an user and it doesn't care about "user session" at all.

User sessions is a concept introduced with modern web site applications/web servers due to the need of authenticating and identifying an user keeping his information and (sensible) data from page to page under the same website.

While session hijacking is mounted by stealing a valid open session id, session fixation attacks are mounted by forging a new valid "trap" session to be used by the victim. Victim will then login on the target server using this trap session, known by the attacker, so that the attacker won't have to guess/bruteforce/steal the session id.

2.2 Session id through URL or Session Cookies

Keeping up the connection state with the web server means exchanging session id between client and server at each request: at first, session id is generated from the web server. Client, in order to be recognized, will have to send the same session id at each subsequent request. There are two ways to exchange session id's between client and server: session cookies and url.

Although cookie offer a greater security they are just a step away from being used to mount a session fixation attack. In case the web application developer irresponsibly chose to retrieve/provide session id's by URL (very common nowadays) attacker will have to trick victim into just clicking a forged (maybe hidden) url.

In case the developer chose the cookie way, attacker will have to successfully install a valid session cookie on victim's machine with his trap session. Even if this is a mitigating factor, it doesn't solve the problem at all.

How to install cookies through XSS or server side scripts is not the purpose of this paper, but let's assume that installing this cookie won't represent a problem for the able and (relatively) skilled attacker.

2.3 Permissive and strict environments

According to the environment your web application is running on, there are different policies with which sessions are handled.

As I will demonstrate later on, the difference between them is little and they're meaningless under a security point of view.

Permissive systems like JRUN accept any session id issued by the client without checking if the session id is a valid one or it has been previously generated by the web server. This means that attacker can use a completely random id.

For example:
"target.com?JSESSIONID=66666666" is a valid id, even if it was not generated by the server.

This simplifies attacker efforts of grabbing a valid id to be used as trap session.

Strict systems like ASP are again just one

security step far from our goal: they accept only server generated id's. This means that attacker must first retrieve a valid id from the target site.

Many websites generate session id's before user authentication. This behaviour is easy to recognize because, urls or links are already fed with a session id or session cookies are set upon site entrance. Although this may not seem a bad practice, in this case will shorten and lighten the attackers efforts to forge a trap session. In my experience this is not uncommon, and it is used by many big e-Commerce websites as well.

If this is not the case, our malicious user will just have to register an account on the target web site in order to have a fully valid and open session id to be used as trap for the victim.

As one can easily understand there's not big difference between a permissive or a strict server: while in the first case forging a session id is completely stealth, the second one probably requires the attaker to have some skills into covering his tracks while registering on the target machine and logging in to retrieve a valid id.

Due to the stealthness of the whole attack (that will result for the attacker to gain control over victim's account), target server has no clue of what's going on, so it won't even discriminate the attacker from the victim.

Once the attacker has gained a valid session id, he will have to force the victim to log into it.

The attacker will have to eventually keep the trap session alive while waiting for the victim to log into. This is where a low absolute session timeout value would restrict the chances for the malicious user to succeed in his attack.

3 Common practices to prevent session fixation

Secure programming practices that generally decrease the chances of a successfull attack are almost all about how tight the login policy is. Most of the session id's are generated at login time and regenerating a session id immediately after a successful login could be a good idea.

3.1 Regenerating session in PHP

Php offers a really comfortable functions to generate a new session id:

```
session_regenerate_id();
```

Web developers using php<4.3.3 must set the session cookie manually to the new session id: any other browser instance of the same site will result into using the old session id!

If the old session is not destroyed the victim will still have the old session id open.

For the versions 4.3.2-3 this snippet from Fou-Lu from codingforums.com will solve the problem:

```
<?php
error_reporting(E_ALL);
ini_set('session.use_cookies', '1');
ini_set('session.use_only_cookies', '1');session_start();
$oldsession = session_id();
session_regenerate_id();
$newsession = session_id();
session_id($oldsession);
session_destroy();
$old_session = $_SESSION;
session_id($newsession);
session_start();
$_SESSION = $old_session;
setcookie(session_name(), session_id(), NULL, '/');
?>
```

This poses to our attention another no less

dangerous problem: open session id's into http_referer urls.

Any web site's log file keeps information about visitors behaviour including where they are coming from (http_referer).

If a user logs into the target server getting a new session id generated for him and his old session is not closed/cookie destroyed, when leaving the target site, he will leave the open session in the new site log file!

This is why users should always use the logout feature and web developers should always keep one and only one session id for each user and keep it alive for as little time as posible.

3.2 ASP

ASP language doesn't provide a "ready-to-go" function to regenerate a new session id. It offers only cookie based sessions so it makes the job of the attacker a bit harder: he/she will have to install a cookie instead of just forcing the victim to click on a link.

ASP web server generates a new key upon each restart with which will encrypt the session id and send it through a session cookie to the requesting client. Consecutive requests result into consecutive session id's values (but encrypted into the cookie). While this prevents from capturing valid cookies, it doesn't prevent from session fixation.

A valid session cookie with a valid open fixed encrypted trap session can still be injected on the victim machine.

ASP server sends a different session id for each different asp file request. When a value is first stored in a session variable the session becomes "fixed" for that user and will be kept alive and equal for any further request. This means that an attacker can recognize a fixed request by giving a look at how the session id changes into his session cookie: the session is fixed and ready to be used as a trap session as soon as it remains the same for two consecutive requests. This demonstrates that the attacker is not always forced to login into the target server in order to have a fixed session!

To prevent such behaviour you can turn on ASP buffering to prevent unnecessary session cookies to be sent to the user and avoid session to be used before user authentication.

An ASP based web application should use one of the techniques we are going to see in the next paragraphs.

4. General Techniques

4.1 "Mutex" - without overhead - (prevents Session fixation and hijacking under certain circumstances)

The idea behind this kind of technique is to create a mutual exclusion access on the same session id between the attacker and the victim. The exclusion is based upon IP acknowledgement. Let's see how it is implemented.

Upon user authentication both a session cookie and a session var are created. The session var will store the logged in user's ip address. So, this is what happen when an attack is issued:

1. Attacker gains a trap session by loggin into the target server.
2. A session variable is set to the ip of the attacker.
3. Attacker makes the victim log in with this trap session.
4. The session var is updated to the ip value of the victim.

For each page that need some privilege level a check between HTTP_REMOTE_ADDR and the session var is done. If they match then access is grant. This is a simple binding between session id's and ip address.

It should be considered that denying access to the login page wether the two ip are different could lead to a denial of service: attacker could keep the connection alive and the victim wouldn't be able to authenticate himself as the login page redirects him to an error page. Thus, the best action to take here is always to allow for new logins and making the ip check on the other pages.

IP address is the only thing that makes us recognize users uniquely. Unfortunately this is not true in the case users are behind the same public proxy. Mutex technique is simple to implement, doesn't use any database stored value for authentication but still doesn't ensure an acceptable level of security.

Environments allowing random trap session (not validated nor generated by the server) like PHP, get more benefits from this technique than any other: not only it ensures a basic protection based on a bind between ip and session id but it also prevents from arbitrary chosen session id's. Checking for the existence of the session var with the ip address forces an attacker to log-in, in order to have a valid session id.

As long as the ip between attacker and victim are not the same Mutex method works fine also against a Session Hijacking attack.

4.2 "Mutex with token" - without overhead - (prevents Session fixation and Session hijacking)

Keeping the benefits of the Mutex method we can develop a much more secure way to handle sessions. This method requires users to have both persistent and session cookies enabled to work.

The idea behind it is to bind ip with something that only victim knows: password.

Furthermore, this method is meant to avoid database access for each page that would represent an important issue into a large scale web application.

Once again, session starts at login time:

1. Attacker registers on target site and logs-in to have a valid trap session.
2. A cookie value is set to md5 (password+ip). It is the so called token.
3. A session var is initialized with the password value.
4. Victim logs in into the trap session.
5. A new (different) token is generated and sent to victim through cookie. Here is the focal point: password used to generate the token is not taken from that in the session var(attacker's password) but from the victim login input.
6. Victim is now the master of the session id. Attacker is no more able to navigate as a logged in user on target site.

Each page will compare the token in the cookie and the token created with the password saved in the session var and the remote address.

In order to succeed, the attacker must:

1. Use the same ip of the victim.
2. Make the victim log into the trap session.
3. Steal the cookie to the victim taking advantage of some xss hole in the target site

It is easy to understand that with the use of the token the probability of the attacker to mount a successful session fixation attack decreases drastically.

If the attacker is capable of stealing victim's cookie he wouldn't need to mount a session fixation attack to break into victim account: he/she could just steal an open session cookie. So, making the assumption that attacker can't steal victim cookie, can be the best lightweight solution to session fixation.

The use of the password as the seed for the token can be replaced with the use of a

random value. This should be then crypted. it has been demonstrated that in a number of shopping carts scripts, the use of numeric tokens can be easily guessed or bruteforced. This is why I preferred to use a password+ip to generate a token: in order to be guessed, the attacker would need user ip and should then bruteforce the password value that (if the password is reasonably long, i.e. Longer than 6 chars) in 99% of the cases can take longer time than the session timeout length.

The use of the password can also be employed into other methods to avoid hijacking, matching user password with the token.

To make the token even harder to be guessed/cracked the use of a random string can turn useful: Upon user login a random case-sensitive alphanumeric value (rand_string) is generated and stored into a session var.

The token will become now something like this: md5 (password + rand_string + ip). Even if the attacker knows victim's ip, it will be much more time consuming to perform a bruteforce and in any case nearly impossible to reconstruct victim's password.

5. Conclusions

The responsibility of preventing problems related to http session management is left to web application developers.

Some systems can be more secure than others providing with easier and smarter approaches to session id management. However there is not a fully secure environment where a http session is protected without a careful analysis of the risks with the consequent development policies taken at designing time.

Session fixation attacks are not common as XSS or SQL injections are. As education to security is more and more spread among web developers, hackers will try to find new less-known attacking techniques. Although Http Session fixation is not new it is almost unknown among developers and is getting some success among hackers. This is why Http Session fixation attacks should not be underestimated.

6. Bibliography

I strongly recommend http://www.acros.si/papers/session_fixation.pdf to have a complete overview of how to mount a session fixation attack.