What is the best way to prevent session hijacking?

Asked 16 years, 4 months ago Modified 3 years, 9 months ago

Viewed 161k times

138

Specifically this is regarding when using a client session cookie to identify a session on the server.

Is the best answer to use SSL/HTTPS encryption for the entire web site, and you have the best guarantee that no man in the middle attacks will be able to sniff an existing client session cookie?

And perhaps second best to use some sort of encryption on the session value itself that is stored in your session cookie?

If a malicious user has physical access to a machine, they can still look at the filesystem to retrieve a valid session cookie and use that to hijack a session?

security session cookies

Share

Improve this question

Follow

edited Aug 23, 2008 at 15:33

Chris

6,842 ● 6 ● 53 ● 67

asked Aug 22, 2008 at 17:02

Adam
28.8k ● 15 ● 61 ● 73

14 Answers

Sorted by: Highest score (default)

147

Encrypting the session value will have zero effect. The session cookie is already an arbitrary value, encrypting it will just generate another arbitrary value that can be sniffed.

The only real solution is HTTPS. If you don't want to do SSL on your whole site (maybe you have performance concerns), you might be able to get away with only SSL protecting the sensitive areas. To do that, first make sure your login page is HTTPS. When a user logs in, set a secure cookie (meaning the browser will only transmit it over an SSL link) in addition to the regular session cookie. Then, when a user visits one of your "sensitive" areas, redirect them to HTTPS, and check for the presence of that secure cookie. A real user will have it, a session hijacker will not.

EDIT: This answer was originally written in 2008. It's 2016 now, and there's no reason not to have SSL across your entire site. No more plaintext HTTP!

Share Improve this answer Follow

edited Aug 31, 2016 at 20:51

answered Aug 22, 2008 at 17:11

Josh Hinman
6,795 ● 7 ● 39 ● 47

31 HTTPS will prevent the sniffing only. But if you have a XSS, or the session IDs can be guessed easily, or you are vulnerable to session fixation, or your session ID storage is weak (SQL injection?), SSL will be no improvement at all. – Calimo Jan 22, 2012 at 18:15

5 @Josh If a malicious user has physical access to a machine, they can still look at the filesystem to retrieve a valid session cookie and use that to hijack a session? – Pacerier Jun 14, 2012 at 1:57

77 If a malicious user has physical access to a filesystem, they don't need to hijack a session. – Josh Hinman Jun 15, 2012 at 6:28

25 @Josh. Not true, sometimes a user has limited time physical access to a filesystem. Imagine a laptop left unlocked by a colleague rushing to the toilet, now all I need to do is to go to that laptop, install EditThisCookie plugin, grab his cookies at plus.google.com using EditThisCookie export feature **and now I have his account**. Time taken: 18 seconds. – Pacerier Oct 13, 2012 at 20:47

2 I'm pretty sure google will have some kind of security feature build-in as this is obviously the first thing you would think of when talking about session hijacking – xorinzor Dec 4, 2012 at 17:10

44

The SSL only helps with sniffing attacks. If an attacker has access to your machine I will assume they can copy your secure cookie too.

At the very least, make sure old cookies lose their value after a while. Even a successful hijaking attack will be thwarted when the cookie stops working. If the user has a cookie from a session that logged in more than a month ago, make them reenter their password. Make sure that whenever a user clicks on your site's "log out" link, that the old session UUID can never be used again.

I'm not sure if this idea will work but here goes: Add a serial number into your session cookie, maybe a string like this:

SessionUUID, Serial Num, Current Date/Time

Encrypt this string and use it as your session cookie. Regularly change the serial num - maybe when the cookie is 5 minutes old and then reissue the cookie. You could even reissue it on every page view if you wanted to. On the server side, keep a record of the last serial num you've issued for that session. If someone ever sends a cookie with the wrong serial number it means that an attacker may be using a cookie they intercepted earlier so invalidate the session UUID and ask the user to reenter their password and then reissue a new cookie.

Remember that your user may have more than one computer so they may have more than one active session. Don't do something that forces them to log in again every time they switch between computers.

answered Aug 23, 2008 at 18:07

Share Improve this answer Follow

bsmcat

I know this is an old post, but just wanted to include that if an attacker were to hijack the session within the window allotted by the "serial number" then this wouldn't affect him. – crush Feb 15, 2013 at 16:11

@crush, but then the attacker would be locked out after the allotted window, right? So at the very least the attack window is small(er). – Johanneke Jul 11, 2013 at 16:46

3 Only problem is if the user leaves your website for 5 minutes, they'll have to login again – elipoultorak Nov 16, 2015 at 11:26

2 I reissued the cookie on every non-GET request and it caused troubles in cases I needed to send multiple XHR requests at once. The first request worked, but the rest used the now-obsolete original cookie. I didn't want to chain those requests (to constantly update the cookie) or use some "do_not_update_coookie" flag, so I abandoned this concept altogether. – mikiqex Oct 7, 2020 at 5:34

21

```
// Collect this information on every request
$aip = $_SERVER['REMOTE_ADDR'];
$bip = $_SERVER['HTTP_X_FORWARDED_FOR'];
$agent = $_SERVER['HTTP_USER_AGENT'];
session_start();

// Do this each time the user successfully logs
in.
$_SESSION['ident'] = hash("sha256", $aip . $bip .
$agent);

// Do this every time the client makes a request
to the server, after authenticating
$ident = hash("sha256", $aip . $bip . $agent);
```

```
if ($ident != $_SESSION['ident'])
{
    end_session();
    header("Location: login.php");
    // add some fancy pants GET/POST var headers
for login.php, that lets you
    // know in the login page to notify the user
of why they're being challenged
    // for login again, etc.
}
```

What this does is capture 'contextual' information about the user's session, pieces of information which should not change during the life of a single session. A user isn't going to be at a computer in the US and in China at the same time, right? So if the IP address changes suddenly within the same session that strongly implies a session hijacking attempt, so you secure the session by ending the session and forcing the user to re-authenticate. This thwarts the hack attempt, the attacker is also forced to login instead of gaining access to the session. Notify the user of the attempt (ajax it up a bit), and vola, Slightly annoyed+informed user and their session/information is protected.

We throw in User Agent and X-FORWARDED-FOR to do our best to capture uniqueness of a session for systems behind proxies/networks. You may be able to use more information then that, feel free to be creative.

It's not 100%, but it's pretty damn effective.

There's more you can do to protect sessions, expire them, when a user leaves a website and comes back force them to login again maybe. You can detect a user leaving and coming back by capturing a blank HTTP_REFERER (domain was typed in the URL bar), or check if the value in the HTTP_REFERER equals your domain or not (the user clicked an external/crafted link to get to your site).

Expire sessions, don't let them remain valid indefinitely.

Don't rely on cookies, they can be stolen, it's one of the vectors of attack for session hijacking.

Share Improve this answer Follow

edited Dec 3, 2013 at 21:56

answered Dec 3, 2013 at 21:24

theironyis
211 ● 2 ● 3

---

3 I've run into a situation before where a certain (fairly large, but technically backward) ISP would change the IP of the user's browser from time-to-time based on re-routing their users' connections. This made the REMOTE_ADDR check return false negatives for us. – goofballLogic Aug 20, 2015 at 15:36

Why bother creating a hash? `$_SESSION['ident'] = $aip . $bip . $agent;` would be just as secure. – Dan Bray May 9, 2017 at 19:53

4 If you have Mobile users using your website on the go and moving from one access point to the other, their IPs will keep changing and they will keep getting signed out of their account. – Kareem Oct 29, 2017 at 18:09

1 What about proxies and VPNs? Anyone using a TOR proxy will be constantly signed out of their accounts.. every few minutes in fact. – Bradley Dec 15, 2017 at 3:18

1 Even if you overlook that, IP addresses can be manipulated by the client, so this wouldn't do much to dissuade an attacker anyway. – Bradley Dec 15, 2017 at 3:19

21

Have you considered reading a book on PHP security? Highly recommended.

I have had much success with the following method for non SSL certified sites.

1. Dis-allow multiple sessions under the same account, making sure you aren't checking this solely by IP address. Rather check by token generated upon login which is stored with the users session in the database, as well as IP address, HTTP_USER_AGENT and so forth
2. Using Relation based hyperlinks Generates a link (eg. http://example.com/secure.php?token=2349df98sdf98a9asdf8fas98df8) The link is appended with a x-BYTE (preferred size) random salted MD5 string, upon page redirection the randomly generated token corresponds to a requested page.
   - Upon reload, several checks are done.
   - Originating IP Address
   - HTTP_USER_AGENT

    - Session Token
    - you get the point.

3. Short Life-span session authentication cookie. as posted above, a cookie containing a secure string, which is one of the direct references to the sessions validity is a good idea. Make it expire every x Minutes, reissuing that token, and re-syncing the session with the new Data. If any mis-matches in the data, either log the user out, or having them re-authenticate their session.

I am in no means an expert on the subject, I'v had a bit of experience in this particular topic, hope some of this helps anyone out there.

Share Improve this answer Follow

edited Dec 29, 2016 at 18:09
Cœur
38.6k ● 26 ● 202 ● 276

answered Jul 4, 2011 at 0:11
Nathan
219 ● 2 ● 2

4 Are there any books you would recommend? – Rikki Jun 10, 2012 at 22:49

1 Especially given that the OP didn't state anything about PHP specifically, I would say that it's better to look at just a general security book (especially as the security between different languages differs only in implementation details, but the concepts remain the same). – matts1 Mar 17, 2014 at 9:07

1 in 2017, even facebook/gmail etc. allows multiple sessions under the same account (especially with mobile devices) unless you hate your users, #1 is a bit outdated. – cowbert Jan 24, 2018 at 4:32

12

There is no way to prevent session hijaking 100%, but with some approach can we reduce the time for an attacker to hijaking the session.

Method to prevent session hijaking:

1 - always use session with ssl certificate;

2 - send session cookie only with httponly set to true(prevent javascript to access session cookie)

2 - use session regenerate id at login and logout(note: do not use session regenerate at each request because if you have consecutive ajax request then you have a chance to create multiple session.)

3 - set a session timeout

4 - store browser user agent in a $_SESSION variable an compare with $_SERVER['HTTP_USER_AGENT'] at each request

5 - set a token cookie ,and set expiration time of that cookie to 0(until the browser is closed). Regenerate the cookie value for each request.(For ajax request do not regenerate token cookie). EX:

```
    //set a token cookie if one not exist
    if(!isset($_COOKIE['user_token'])){
                    //generate a random string
for cookie value
        $cookie_token =
bin2hex(mcrypt_create_iv('16' ,
MCRYPT_DEV_URANDOM));

        //set a session variable with that
random string
        $_SESSION['user_token'] =
$cookie_token;
        //set cookie with rand value
        setcookie('user_token', $cookie_token ,
0 , '/' , 'donategame.com' , true , true);
    }

    //set a sesison variable with request of
www.example.com
    if(!isset($_SESSION['request'])){
        $_SESSION['request'] = -1;
    }
    //increment $_SESSION['request'] with 1 for
each request at www.example.com
    $_SESSION['request']++;

    //verify if $_SESSION['user_token'] it's
equal with $_COOKIE['user_token'] only for
$_SESSION['request'] > 0
    if($_SESSION['request'] > 0){

        // if it's equal then regenerete value
of token cookie if not then destroy_session
        if($_SESSION['user_token'] ===
$_COOKIE['user_token']){
            $cookie_token =
bin2hex(mcrypt_create_iv('16' ,
MCRYPT_DEV_URANDOM));
```

note: do not regenerate token cookie with ajax request
note: the code above is an example. note: if users logout

then the cookie token must be destroyed as well as the session

6 - it's not a good aproach to use user ip for preventing session hijaking because some users ip change with each request. THAT AFFECT VALID USERS

7 - personally I store session data in database , it's up to you what method you adopt

If you find mistake in my approach please correct me. If you have more ways to prevent session hyjaking please tell me.

Share Improve this answer Follow

edited Dec 23, 2017 at 22:52
ROMANIA_engineer
56.5k ● 30 ● 208 ● 205

answered Nov 1, 2013 at 12:16
Alexandru
303 ● 4 ● 9

hi, i m trying to prevent session hijacking (in ASP.NET)and considered all above steps u suggested. It is approximate working but when I use InPrivateBrowsing/incognito mode of browser the session is Hijacked. Can u please suggest any additional thing to add in sessionId string?
– Vishwanath Mishra Dec 16, 2014 at 12:33

9

Try Secure Cookie protocol described in this paper by Liu, Kovacs, Huang, and Gouda:

As stated in document:

> A secure cookie protocol that runs between a client and a server needs to provide the following four services: authentication, confidentiality, integrity and anti-replay.

As for ease of deployment:

> In terms of efficiency, our protocol does not involve any database lookup or public key cryptography. In terms of deployability, our protocol can be easily deployed on an existing web server, and it does not require any change to the Internet cookie specication.

In short: it is secure, lightweight, works for me just great.

Share Improve this answer Follow

edited Sep 13, 2013 at 19:06
Crowie
3,272 ● 7 ● 30 ● 50

answered Nov 24, 2008 at 10:54
Hubert
2,313 ● 1 ● 19 ● 14

---

9 Your link is a spec of the protocol - do you have a link to an implementation? That would be great - thanks. – Adam Aug 7, 2010 at 17:17

Not sure this protocol helps much, see security.stackexchange.com/a/20405/8618 – akostadinov Apr

28, 2022 at 17:57

5

There are many ways to create protection against session hijack, however all of them are either reducing user satisfaction or are not secure.

- IP and/or X-FORWARDED-FOR checks. These work, and are pretty secure... but imagine the pain of users. They come to an office with WiFi, they get new IP address and lose the session. Got to log-in again.
- User Agent checks. Same as above, new version of browser is out, and you lose a session. Additionally, these are really easy to "hack". It's trivial for hackers to send fake UA strings.
- localStorage token. On log-on generate a token, store it in browser storage and store it to encrypted cookie (encrypted on server-side). This has no side-effects for user (localStorage persists through browser upgrades). It's not as secure - as it's just security through obscurity. Additionally you could add some logic (encryption/decryption) to JS to further obscure it.
- Cookie reissuing. This is probably the right way to do it. The trick is to only allow one client to use a cookie at a time. So, active user will have cookie re-issued every hour or less. Old cookie is invalidated if new one is issued. Hacks are still possible, but much

harder to do - either hacker or valid user will get access rejected.

Share Improve this answer Follow

answered Aug 17, 2015 at 9:57
Hatch
696 ● 8 ● 8

4

Ensure you don't use incremting integers for session IDs. Much better to use a GUID, or some other long randomly generated character string.

Share Improve this answer Follow

answered Aug 23, 2008 at 18:10
Kibbee
66.1k ● 28 ● 144 ● 184

3

AFAIK the session object is not accessible at the client, as it is stored at the web server. However, the session id is stored as a Cookie and it lets the web server track the user's session.

To prevent session hijacking using the session id, you can store a hashed string inside the session object, made using a combination of two attributes, remote addr and remote port, that can be accessed at the web server inside the request object. These attributes tie the user session to the browser where the user logged in.

If the user logs in from another browser or an incognito mode on the same system, the IP addr would remain the same, but the port will be different. Therefore, when the application is accessed, the user would be assigned a different session id by the web server.

Below is the code I have implemented and tested by copying the session id from one session into another. It works quite well. If there is a loophole, let me know how you simulated it.

```
@Override
protected void doGet(HttpServletRequest request,
HttpServletResponse response)
        throws ServletException, IOException {
    HttpSession session = request.getSession();
    String sessionKey = (String)
session.getAttribute("sessionkey");
    String remoteAddr = request.getRemoteAddr();
    int remotePort = request.getRemotePort();
    String sha256Hex =
DigestUtils.sha256Hex(remoteAddr + remotePort);
    if (sessionKey == null ||
sessionKey.isEmpty()) {
        session.setAttribute("sessionkey",
sha256Hex);
        // save mapping to memory to track which
user attempted
        Application.userSessionMap.put(sha256Hex,
remoteAddr + remotePort);
    } else if (!sha256Hex.equals(sessionKey)) {
        session.invalidate();

response.getWriter().append(Application.userSessionM
        response.getWriter().append(" attempted to
hijack session id
").append(request.getRequestedSessionId());
        response.getWriter().append("of user
").append(Application.userSessionMap.get(sha256Hex))
        return;
    }
    response.getWriter().append("Valid
Session\n");
}
```

I used the SHA-2 algorithm to hash the value using the example given at SHA-256 Hashing at baeldung

Looking forward to your comments.

Share Improve this answer Follow

edited Feb 11, 2018 at 15:35

answered Feb 10, 2018 at 18:13

Jzf
90 ● 2 ● 9

@zaph I'm sorry, but I didn't add the answer to gain rep. The point is not SHA-2 encryption either. But the fact that I was able to detect usage of another user's session id Cookie into another user's session, i.e., session hijacking as in the original question. I tested my solution and found it works. But I am not the expert on this, so I would like the community to check if my answer is good enough. Maybe you can see what my code snippet does and decide whether it answers the question. Thanks! – Jzf Feb 11, 2018 at 13:26

@zaph Thank you for pointing out the mistake. I have updated my answer as per your suggestion. Please remove your down vote if you think the answer is helpful. – Jzf Feb 11, 2018 at 15:37

Let us consider that during the login phase the client and server can agree on a secret salt value. Thereafter the

1

server provides a count value with each update and expects the client to respond with the hash of the (secret salt + count). The potential hijacker does not have any way to obtain this secret salt value and thus cannot generate the next hash.

Share Improve this answer Follow

edited Feb 14, 2014 at 19:37

answered Feb 14, 2014 at 19:31

davej
165 ● 1 ● 6

3 But how do you want to store this salt on client side that no one could steal it? – Dcortez Jun 24, 2016 at 5:40

0

To reduce the risk you can also associate the originating IP with the session. That way an attacker has to be within the same private network to be able to use the session.

Checking referer headers can also be an option but those are more easily spoofed.

Share Improve this answer Follow

answered Aug 22, 2008 at 17:04

Julio César
13.2k ● 10 ● 40 ● 46

7 no, you can't use the originating IP, as it may change - either through dynamic IPs, changed when a user is temporarily disconnected, or through the (implicit or explicit) usage of a proxy farm. Also, session hijackers coming from the same

ISP may use the same proxy & IP as a legitimate user... – Olaf Kock Oct 3, 2008 at 15:35

1 PHP-Nuke has a good page about their session approach, and they talk in detail about how hooking it to the IP doesn't work with all ISPs phpnuke.org/... – Sembiance Jun 29, 2010 at 19:02

4 Changing Ip's are very common with mobile internet providers. With Vodafone, my IP changes with EVERY request. – Blaise Mar 27, 2011 at 18:02

1 Somewhat of an old post but to further this. IP's are as mentioned a bad idea. As mentioned mobile users tend to roam IP's. Some ISP's in the past also had roaming IP's (such as AOL) however this approach is becoming more popular now with the shortage of IPv4 IP's. Such ISP's include Plus net and BT – Peter Mar 26, 2013 at 20:33

0

Use SSL only and instead of encrypting the HTTP_USER_AGENT in the session id and verifying it on every request, just store the HTTP_USER_AGENT string in your session db as well.

Now you only have a simple server based string compare with the ENV'HTTP_USER_AGENT'.

Or you can add a certain variation in your string compare to be more robust against browser version updates. And you could reject certain HTTP_USER_AGENT id's. (empty ones i.e.) Does not resolve the problem completley, but it adds at least a bit more complexity.

Another method could be using more sophisticated browser fingerprinting techniques and combine theyse values with the HTTP_USER_AGENT and send these values from time to time in a separate header values. But than you should encrypt the data in the session id itself.

But that makes it far more complex and raises the CPU usage for decryption on every request.

Share Improve this answer Follow

answered Feb 10, 2021 at 21:33

NetDiver

29 ● 3

0

If ISP hijack the certificate-verification, ISP will possibly initiate a Man-in-the-middle attack. Especially with a compromised certificate authorities.

So I believe you can not prevent session hijack from ISP. Especially when legal forces come with a fake certificate got from CA under law enforce.

You will need something outside the network to protect your session, for example one time pad. This is why one time pad so sensitive and can only be sold by few companies.

Be careful, one time pad may be exploited. Choose your one time pad with profession.

Share Improve this answer Follow

edited Mar 11, 2021 at 6:30

answered Mar 11, 2021 at 4:28

Danger Saints
537 ● 1 ● 4 ● 10

about the OTP security, please google 'crypto ag swiss' – Danger Saints Mar 11, 2021 at 5:19

-15

Protect by:

```
$ip=$_SERVER['REMOTE_ADDER'];
$_SESSEION['ip']=$ip;
```

Share Improve this answer Follow

edited May 31, 2013 at 10:14

Apurv
3,743 ● 3 ● 34 ● 53

answered Jun 30, 2011 at 9:36

Nima
3 ● 1

13 I fail to see what you are doing here. – samayo Jul 28, 2013 at 12:50