正确理解 Session 的安全性

96 虞大胆 (/u/4fd01692f6d9) + 关注
(/u/4fd01692f6d9) 2018.02.12 22:17* 字数 2010 阅读 728 评论 0 喜欢 8

Session 是 Web 开发中必须涉及的一个话题，面试的时候很多人理解 Session 和 Cookie 的时候总是就一句“一个存储在服务器端，一个存储客户端”，被面试的人回答的时候可能自己也觉得很空洞，而面试官肯定也会很不满意，其实完全可以换个话题来考察对于 Session 的理解，这就是这篇博文的题目“正确理解 Session 的安全性”。

在 PHP 语言中，Session 安全性这个锅不能全部让 PHP 背。Session 不安全的根本原因其实就是二点，第一个就是会话传递机制导致的（Cookie 和 URL参数）劫持问题（只要有网络就会有劫持，所以劫持不是这篇博文的重点），另外个就是会话固定的问题。

假如你仅仅使用 PHP 原生的函数去管理你的 Session，可能会有安全性的问题，所以提倡使用第三方的 Session 类库（它考虑很多安全性问题）。
另外 PHP 本身也提供了很多指令和方法来提高安全性（很多文章描述 Session 有多么不安全，并且弄出了很多解决方案，但其实高版本的 PHP 提供了很多内置的解决方案，这就是看手册的重要性）。

简单描述下 Session ，Session 的数据是存储在服务器端的，那么服务器如何保持会话呢（或者说如何找到特定的用户呢？），可以通过二种方式（Cookie 和 URL 参数）来传递。一涉及到传递就比如会有危险，并且在 PHP 中这个危险被放大了。

仅仅使用 Cookie 来存放 Session ID

Session ID 可以使用 Cookie 和 URL 参数来传递，相对来说，Cookie 更安全（ URL 暴露的频率更高），假如你不考虑 Cookie 被禁用的问题，你应该只使用 Cookie 来传递。PHP.ini 提供了 session.use_only_cookies 指令来加强安全性。

保护你的 Cookie

Cookie 本身也是不安全的，所以提升 Cookie 安全性的方法也同样适用与 Session。
关键的就是 HttpOnly(不允许 Javascript 脚本读取)，另外假如你的网站是支持 Https 协议的，那么配置 Cookie 只能通过 Https 协议传递。

Cookie 和 Session 的过期时间保持一致。

很多开发者非常注意 Session 的 GC 过期时间，GC 代表到了特定日期，PHP 会自动清除服务器端的 Session 文件，但是还很多人理解这个概念有误区：

- Session 的存活时间是“变化的”，只要会话一初始化，Session 文件的修改时间就会更新，换句话说，设置该 GC 时间为 2 小时过期，但是只要用户间隔 2 小时一直保持会话更新（比如刷新页面），则 Session 一直会存在。
- GC 到期后，由 PHP 来控制删除 Session 文件，但是并不是每次就会删除过期文件，所以不能依赖它（但是可以通过自定义 Session 管理器来实现这个机制）。

考虑到 Session GC 不可依赖，所以间接的可以设置 Cookie 对应的过期时间（session.cookie_lifetime 指令）等于 GC 时间，这样一到过期时间，由于 Cookie 失效了，等同于 Session 也失效了（虽然 Session 对应的文件并没有删除，假如攻击者知道 Session ID 还是会带来安全问题）。

Session Time-Outs

由于 Session 的 GC 不能依赖，所以很多类库通过一些方案来解决该问题，考虑这样一个场景“假如你登陆了一个网站，并且很长时间没有操作了”，那么这个会话有效吗？理论上来说应该不算有效，会话的有效时间应该是“变化的”，当用户触发了会话，则更新一次（等于将过期时间延长）。

```
session_start();
$activeTime = 3600;
$t = time();
$lastactivetime = $_SESSION["lastactivetime"] ;
if (isset($lastactivetime)) {
    if (($lastactivetime + $activeTime) > $t) {
        return;
    }
}
$_SESSION["lastactivetime"] = $t;
```

在 PHP CodeIgniter 框架中，这个激活有效时间称为 sess_time_to_update。
当然这个方法也并不是特别安全，因为一般来说攻击者会不断的维持会话处于“激活状态”。

Regenerate the Session ID

假如攻击者劫持了你的 Session ID，但是你又不知道，为了安全建议经常性的重置 Session ID（比如不定期的使用session_regenerate_id()函数），这样攻击者等同于拿到的是旧的 Session ID（假设新的 Session ID 攻击者获取不到），这样就不能获取 Session 数据了。
但这也不是完全有效的（因为攻击者有方法能劫持你的 Session ID）。

更有效的方式其实应该是在用户操作更高权限功能的时候（比如电商结账的时候），让用户重新输入密码去验证（获取用户密码是另外一种攻击方式 ）。这是从应用层角度考虑最有效的保护方式。

会话固定

会话固定我理解为二层意思，只要有正确的 Session ID ，不管你从什么设备上发起请求，服务器程序校验出有对应的 Session 信息就认为是正确的，第二层意思就是假如客户端伪造一个 Session ID（伪造包含二部分，第一本身服务器上没有这个 Session ID，第二 Session ID的值格式也可以是错误的），PHP 处理的时候看到这个 Session ID不存在，就以这个Session ID 去初始化，这样的机制可能带来攻击。

举个例子，一个攻击者发送你一个连接，比如 `https://www.jd.com/?SID=1234` ，你打开后，看到的是京东的网站，然后你登录了（这时候有了正确的身份验证信息），攻击者这时候在自己的电脑就能以你的身份购买东西了（因为攻击者和用户是同一个 Session ID）。

那么如何解决呢？除了上面提到的一系列方法，还可以提供二个方法。

第一种是应用层的解决方案，上面说了 Session ID 和客户端环境没有关系，那么可以加上这个关系，比如初始化 Session ID 的时候，可以将用户浏览器的 UA 头保存到 Session 信息中，假如攻击者想通过这个 Session ID 获取权限的时候，服务器端代码一看这个 Session ID 中的 UA 信息和访问者不一样，就认为是非法请求。

第二种解决方案是 PHP 提供的，session.use_strict_mode 指令假如开启，未初始化的 Session ID PHP 会重新生成一个，很大程度上解决了安全问题。

在这个指令没出现的情况下，一般通过如下方式去解决：

```
//使用 session_id 作为校验 ID
session_destory();
session_regenerate_id();
$_SESSION['valid_id'] = session_id();

//校验 session_id 是否初始化
if ($_SESSION['valid_id'] !== session_id()) {
  session_regenerate_id();
  $_SESSION['valid_id'] = session_id();
}
```

最后，安全问题是相对的，没有绝对的安全，尽量让其更安全，并通过应用解决方案去提升安全。

并且这里也没有提到如何进行 Session 劫持的问题，这已经不属于 PHP 解决的范畴了。

正确的关闭会话方式

很多开发者在用户退出的时候，不会主动或者正确的关闭会话，关闭会话包含三个方面，第一就是传递 Session ID 的 Cookie 应该删除，第二就是 Session 文件应该删除，第三在 PHP 进程中的 Session 全局变量也应该清除，用代码来说明下：

```
$_SESSION = array();
session_unset();
$name = session_name();
if (isset($_COOKIE[$name])) {
    $r = session_get_cookie_params();
    setcookie($name, '', time() - 3600, $r['path'], $r['domain'], $r['
    secure'], $r['httponly']);
}
session_destroy();
```

推荐阅读 : http://www.acros.si/papers/session_fixation.pdf (https://link.jianshu.com?t=http://www.acros.si/papers/session_fixation.pdf)