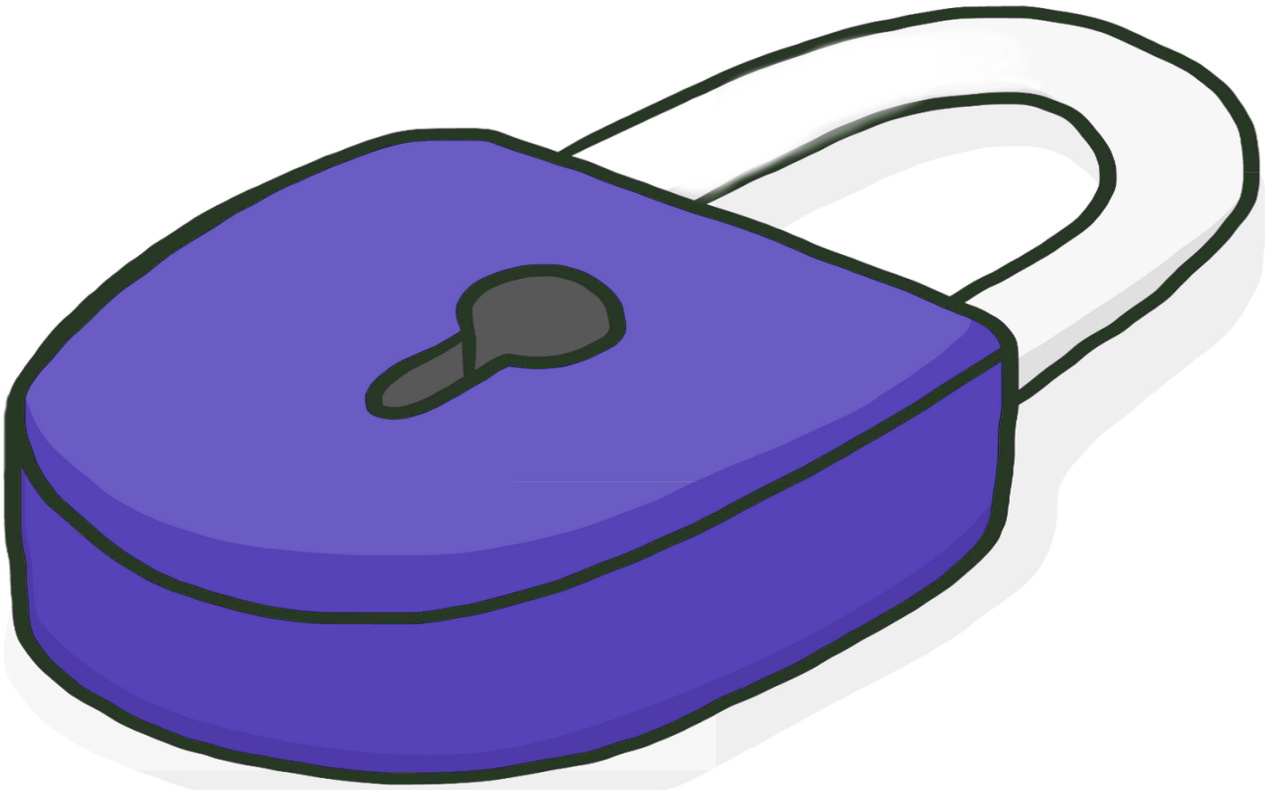
构建 安全的 PHP 应用



一本实用手册

Ben Edmunds

译者：张庆龙

构建安全的 PHP 应用

因为不确定你的 PHP 应用是否足够安全而无法安然入睡？读完本书，让我们确保每天能够按时下班回家，并能高枕无忧的做闭目佳人！

Ben.edmunds@gmail.com and 张庆龙

This book is for sale at http://leanpub.com/buildingsecurephpapps-ch

This version was published on 2015-09-06

让我们一起为中国的软件工程师做些事情

帮我们在 ***weibo, twitter,*** 以及你的个人博客或者任何地方推荐这本有用的书

微博话题请点击 ***http://weibo.com/p/1008088d842076a09287c8bee8bf694ae50e6f***

twitter 标签请点击 ***https://twitter.com/hashtag/***构建安全的 ***PHP*** 应用***?src=hash***

图书托管及购买地址***https://leanpub.com/buildingsecurephpapps-ch***

Contents

译者序

2015 年 7 月的一天，我正在 Feedly 上悠然的阅读，突然出现一篇博客有介绍这本书，因为说的是我当时正在关注的 PHP 安全，所以认真看了那篇博客，顺着博客里的链接找到了这本书的官网，看起来很靠谱，而恰好我也需要，网站上有购买链接，一直到 leanpub 上我都在寻找是否有中文版可以购买，但没有找到，而后在 Twitter 上联系了作者 Ben，问他关于中文版的事，遗憾的是还没人翻译过，当时就想这本书也不长，为什么我不来翻译一下，为中国的开发者做点贡献呢，当然也会收获小小的成就感，随后我们用邮件沟通了翻译的详细事项，只用了来回两封邮件，不过几百个单词，交流真简单明了，我爱上了这种高效简单的沟通方式！当然以后如果还有这样的机会，我想我还会继续行好事，做想做该做的，不问前程！如果有翻译不好的地方，可以随时邮件我，我会非常积极的修改，这本书在相当长的一段时间都会以电子版的形式存在，所以当我修改了你提出的问题后，一定会最快的同步线上，这样购买的所有人都会收到更新了，一起行好事吧！

写在前面

几年前我用 PHP 的 CodeIgniter 框架写了一个网页程序，但是这个框架并没有内置任何类型的身份验证系统。当然，这并不会难倒像我这样的一个好 (懒惰) 的开发者，我到处寻找一个靠谱的库来让我的应用拥有健壮的身份验证能力。然而令人失望的是我发现在 CodeIgniter 上并没有一个简洁、可靠并能满足身份验证需求的库。这让我走向了开发 Ion Auth(可以从 Github 找到) 之路，它是为 CodeIgniter 开发的一款轻量级的身份验证库，并在为网页应用的安全上做了一个长时间的改革迭代，同时也帮助其他开发人员这样做。

多年之后，我们都已经换了很多的框架和语言，但是我仍然对被忽视的基础安全方面保持持续关注。让我们一起改变这个现状吧。我希望能够帮助大家再也不用生活在密码泄露的恐惧中，再不会为恶心的 SQL 注入而担心，能够轻松的避免那些“黑客”的临幸。让我们都能确保可以每天按时下班回家，并能高枕无忧的做闭目佳人！

这本书将会是一本可以在具体项目中进行参考的快速阅读手册。意思是你可以在数个小时内快速看完并在你需要的时候随时查阅。我也会尽量使阅读本书变得更加有趣。

格式说明

若无特殊说明，在本书中缩进内的示例代码均为 PHP

以 `$` 符号开头的行

```
$ ls -al
```

是普通用户下在命令行下的命令示例

以 `#` 符号开头的行

```
# ls -al
```

是 root 用户命令行下的命令示例

服务器命令行示例会呈现 *nix(centos, redhat, ubuntu, osx, etc) 操作系统风格的样式

我会努力让示例代码有合适的换行，这样方法参数会在不同的行。这本书的代码风格虽然看起来比较奇怪但是比那些封装的代码更容易阅读。

勘误表

如果您发现任何问题都可以毫不犹豫的联系我的邮箱[1]
如果您有翻译问题也可以联系译者邮箱[2]

示例代码

若无特殊说明，例子中的代码均为 PHP。我会尽可能使用原生的 PHP，除非它造成了太多的冗余。当使用原生 PHP 来阐述问题太啰嗦时，我也会使用 Laravel 框架更优雅的风格让大家更容易理解。

一些代码过时或者不清楚，你可以在Github repository[3]找到更全的代码。

来吧！

[1] http://xkcd.com/327/
[2] 对于大多数精确的细节都没有公开，所以我们不能肯定这些都会归结于 SQL 注入攻击。大多数都是这样认为。
[3] https://github.com/benedmunds/Building-Secure-PHP-Apps-Examples

关于作者

Ben Edmunds[4] 带领开发团队做最前沿的网页和手机应用。他是一个活跃积极的领导者、开发者，也是在多个开发社区的演讲者。他已经在软件研发领域专业研究十余载并做过从机器人技术到政府项目的各种工作。

PHP Town Hall 播客的联合主持。波特兰 PHP Usergroup 联合组织者。开源项目倡导者。

关于译者

张庆龙[5] 技术上致力于 web 开发，主要使用 PHP 和 Ruby on Rails，热衷使用 Mac 进行开发和日常工作。热爱生活，喜钓鱼，爱表演，好搞小幽默。主要活动在京津冀地区。

[4] http://benedmunds.com
[5] http://akmumu.com

第一章 - 不相信任何用户，格式化所有的输入！

我们从一个故事说起，麦克是俄克拉何马州一个私立学校的系统管理员。他的主要工作是保持学校的网络畅通和服务器稳定。最近他开始为学校开发一个可以自动完成各种任务的网页应用以供内部使用。他没有参加过正规的培训而且是刚从一年前开始编程，但是他对自己的工作自我感觉良好。他了解一些 PHP 基础而且已经为学校开发了一个有够稳定的客户关系管理系统。还有大量的功能等着添加，但是基本功能已经完备。麦克甚至由于精简业务和节省学校开支获得了来自院长的称赞。

一切都很好，但是有一天一个特别的学生带来了噩梦。这个学生的名字是 Little Bobby Tables[6]。这天，大乔在管理员办公室打电话给麦克问为什么系统崩溃了。经过排查，麦克发现那个存有所有学生信息的数据库表完全消失了。好家伙，Little Bobby 的全名竟然是”Robert’); DROP TABLE students;–“，太尴尬了。这时候数据库还没有备份，这是麦克即将要做的一件事，但是还没开始。这回麦克摊上大事儿了。

[6] http://xkcd.com/327/

SQL 注入

真实的故事

虽然在真实的世界里名字中包括危险 SQL 语句的可能非常小，但是像这种 SQL 注入漏洞却每天都在发生：[7]

- 2012 年，LinkedIn 由于一个隐藏的 SQL 注入漏洞泄露了 600 万用户数据
- 2012 年，雅虎！泄露了 450,000 的用户密码
- 2012 年，Nvidia 的 400,000 个密码被盗
- 2012 年，Adobe 的 150,000 个密码被盗
- 2013 年，eHarmony 泄露了大概 150 万的用户密码

SQL 注入的原理

如果你可以不加修饰的接受用户直接输入的内容，那么一个歹毒的用户就可以传入一个奇葩的数据，直接改变你的 SQL 语句。

假设你的代码像下面这样:[8]

```
1 mysql_query('UPDATE users
2   SET first_name="' . $_POST['first_name'] . '"
3   WHERE id=1001');
```

你可能希望生成的 SQL 语句为:

```
UPDATE users set first_name="Liz" WHERE id=1001;
```

但是如果有个歹毒的用户把他的名字写成:

```
Liz", last_name="Lemon"; --
```

那生成的 SQL 语句就变成了下面这样:

[7]对于大多数精确的细节都没有公开，所以我们不能肯定这些都会归结于 SQL 注入攻击。大多数都是这样认为。

[8]那些 `mysql_*` 扩展和它的方法已经被正式弃用。不要再使用它们。

```
UPDATE users
SET first_name="Liz", last_name="Lemon"; --"
WHERE id=1001;
```

现在你所有的用户的名字都被改成了 Liz Lemon, 这岂不是很悲催。

如何防止上面的事发生

一个有效预防 SQL 注入的方法是格式化输入（也被称为转义）。你可以为每个特别的输入转义。或者更好的方法被称作参数约束，这是我强烈推荐的方法，它拥有更高的安全性。使用 PHP 的 PDO 类[9]，你的代码现在变成下面这样：

```
$db = new PDO(...);
$query = $db->prepare('UPDATE users
  SET first_name = :first_name
  WHERE id = :id');

$query->execute([
  ':id'         => 1001,
  ':first_name' => $_POST['first_name']
]);
```

使用约束的参数就意味着每个值都能被恰当的引用，转义，且只匹配一个值。需要牢记的是，参数约束虽然可以保护你的 SQL 语句，但是却不会在插入数据库之后保护输入的数据。记住，任何数据都有可能是具有破坏性的。你仍然需要剔除 and/or 这些容易被遗忘且稍后会展示在用户页面上的内容。你可以在存入数据库的时候就进行处理，或者在展示的时候处理，但是不要省略这重要的一部。我们会在接下来的“Sanitizing Output”详细探讨。

现在你的代码稍微多了一点，但是更安全了。你不用担心再被另一个 Little Bobby Tables 弄糟你的日子。参数约束已经很帅了是吧，知道还有什么很帅么？是我！哈哈。

最佳做法和其他解决方案

存储过程是另一个防范 SQL 注入的方法，它是在数据库上编译的。用上存储过程以后就意味着你不太可能被 SQL 注入，因为你的数据一开始就没有通过 SQL 语句的形式传输。一般来说，存储过程是让人苦恼的，有以下几个主要原因：

1. 难以测试
2. 把业务逻辑放到了应用外（即数据库层）
3. 不易版本控制，因为他没在你的代码而是在数据库

[9] http://us1.php.net/manual/en/intro.pdo.php

4. 在需要修改这部分逻辑的时候直接限制了能胜任之人的数量（为什么存储过程在我们日常开发中不多见?）

客户端 **Javascript** 并不是验证数据的好办法。它很容易被篡改或者被一个只具备初级互联网知识水平的歹毒用户回避。跟我重复：我永远不会只依靠 Javascript 验证；我永远不会只依靠 Javascript 验证。你当然可以使用 Javascript 来提供实时交互并做更好的用户体验，但是为了表达对神的真爱，还是需要在后端增加处理逻辑来保证一切都是合法的。

Mass Assignment(批量赋值)

Mass assignment 是一个非常能提高开发速度的实用工具，但若使用不当也会带来严重问题。

假如你有一个 `User` 的模型，你需要对它进行一些更改。可以依次更新每个字段，或者可以把所有需要修改的值一次性通过表单传过去。

比如下面是你的表单：

```
<form action="...">
  <input name="first_name" />
  <input name="last_name" />
  <input name="email" />
</form>
```

然后使用后端的 PHP 代码处理提交的表单。如果使用 Laravel 框架，代码是下面这样：

```
$user = User::find(1);
$user->update(Input::all());
```

真是又快又好是吧？但是如果有个歹毒的用户修改了表单，给了她自己管理员权限呢？

```
<form action="...">
  <input type="text" name="first_name" />
  <input type="text" name="last_name" />
  <input type="text" name="email" />
  <input type="hidden" name="permissions" value="{'admin':'true'}" />
</form>
```

我们的后端代码就会错误的改变了用户的权限。

听起来又有一个愚蠢的问题需要解决了，但这却是目前大多数开发者和网站可能深受其害的。最近，大家应该都听说了一个开发者扬言 Ruby on Rails 很容易被这个的漏洞利用。Egor Homakov 最初向 Rails 团队提交问题说 Rails 在刚被初始化之后是不够安全的，他这个 bug 很快就被关掉了。核心团队认为要避免这个问题的方法对新手来说都是常识 (attr_accessible)，这个问题属于开发者而不是 Rails 应该做的 (大家一般都会这样想)。Homakov 觉得好心被当成了驴的某些器官，非常生气，就黑了 Github 上的 Rails 账号 (Github 是用 Rails 搭建的)，给了自己他们代码仓库的管理权限。插入了一些提交信息，并重新打开了他那个被关掉的 issue。当然了，这证明了他的观点，现在 Rails 团队也已经默认保护了此类攻击。

你是如何应对类似的攻击呢？具体实现方法还要看你使用的框架和语言，但是你有几个通用的选择：

- 彻底关掉 mass assignment
- 给可以安全被批量赋值的字段加白名单
- 给那些危险的加黑名单

根据你的实现方法，这些也可以被同时使用。

在 Laravel 框架你可以在 models 中添加一个 `$fillable` 变量做可被批量赋值的白名单字段:

```
1  class User extends Eloquent {
2
3    protected $table = 'users';
4
5    protected $fillable = ['first_name', 'last_name', 'email'];
```

这样就会在批量赋值的时候把”permissions” 字段过滤。另一种方法就是增加一个 `$guarded` 变量做黑名单:

```
1  class User extends Eloquent {
2
3    protected $table = 'users';
4
5    protected $guarded = ['permissions'];
```

哪种对你的程序方便就用哪种。

如果你不是用 Laravel，你的框架可能已经集成了类似黑/白名单的方法。如果你使用的定制框架，干嘛不自己实现他呢！

类型转换

我一般还喜欢做另外的一步，不光为安全还考虑到数据完整性，那就是对已知格式进行类型转换。由于 PHP 是一门动态类型的语言[10], 一个值有可能是很多的类型：字符型，整形，浮点型等。通过类型转换，我们可以确认数据都会匹配我们的所需。上面的例子中，如果我们知道变量 ID 一定会是整形的数字，那么它被类型转换将变得很有意义：

```
$id = (int) 1001;

$db    = new PDO(...);
$query = $db->prepare('UPDATE users
  SET first_name = :first_name
  WHERE id = :id');

$query->execute([
  ':id'         => $id, //我们知道它是个整形
  ':first_name' => $_POST['first_name']
]);
```

上面这样并没什么意义，因为 ID 是我们自己定义的，我们本来就知道他是整形，但是如果 ID 是在别的表单传递过来的，它就有了让我们心如止水的意义。

PHP 提供了一系列可以转换的类型：

```
$var = (array) $var;
$var = (binary) $var;
$var = (bool) $var;
$var = (boolean) $var;
$var = (double) $var;
$var = (float) $var;
$var = (int) $var;
$var = (integer) $var;
$var = (object) $var;
$var = (real) $var;
$var = (string) $var;
```

这不仅在处理数据库方面很有帮助，而且在你整个应用程序上都有用。因为即使 PHP 是动态类型但并不意味着你不能在某些特别的地方尝试强制类型。科学开发！

[10] http://stackoverflow.com/questions/7394711/what-is-dynamic-typing

净化输出

输出到浏览器

你不仅要关注数据的插入，还应该净化/转义任何由用户生成最终会被输出到浏览器的内容。

你可以在存入数据库之前修改或转义数据内容，或者在需要展示到浏览器的时候做这件事。这通常要看你的数据是如何编辑以及它的用处。比如，如果用户可能再次编辑，原样存储，净化输出将更有意义。

什么时候转义会输出的用户内容有大的安全意义呢？假如用户提交了下面这样的 Javascript 片段到你的应用，等下还可能被显示到浏览器:

```
<script>alert('I am not sanitized!');</script>
```

如果在输出的时候你什么都不做，这个恶毒的 Javascript 通常都会像你自己写的一样被执行。这只是一个无所谓的 `alert()`, 但是黑客一般都不是这么友善。任何可能被显示且从外面插入到你应用的数据，你都要对其处理。

还有一个这方面的应用是图片的 XIFF 数据。如果你的应用会展示一个用户上传的图片的 XIFF 数据，这也应该被格式化。

如果你正使用模板引擎或者你使用的框架提供了模板机制，它可能已经自动完成了转义，或者它会提供一个相关的方法给你。详细的使用和实现你可以查看相关的文档。

这完全取决于你，PHP 内置了很多函数，当你要在浏览器显示数据时都会成为你最好的朋友: `htmlentities()`[11] 和 `htmlspecialchars()`[12]. 他们都可以转义使得在展示之前就变得更加安全。

`htmlspecialchars()` 可以完成 90% 你想要的结果。他会自动转义 (如 <, >, &) 这些特殊的字符至 HTML 实体。

`htmlentities()` 和 `htmlspecialchars()` 性质相差无几. 如果可能他会转义所有的字符到 HTML 实体。这在很多时候就不太实用。确认这些方法使用之后的结果，之后再评估哪种是你想要的。

命令行的响应

别忘了对命令行脚本的运行结果进行格式化。相关的函数有 `escapeshellcmd()`[13] 和 `escapeshellarg()`[14].

[11] http://us1.php.net/htmlentities
[12] http://us1.php.net/htmlspecialchars
[13] http://us1.php.net/escapeshellcmd
[14] http://us1.php.net/escapeshellarg

顾名思义。使用 `escapeshellcmd()` 转义所有命令。可以防止所有命令被执行。`escapeshellarg()` 用来搞封装的参数来确保他们被正确的转义，确保不会用你的应用程序来处理命令一样处理参数。