Django中的OWASP防御手段总结--第一部分

Pinging / 2019-04-26 09:39:00 / 浏览数 3985 安全技术 WEB安全 顶(0) 踩(0)

■■■■Django■■■OWASP■■

本系列的第1部分将重点介绍Django针对OWASP Top
10中列出的一些最常见风险的攻击、防御措施，而在第2部分中，我们将重点介绍由于错误配置和不安全的编码而导致的攻击。
对于那些不了解此类攻击的人员，OWASP Top 10列出了在实际应用程序和API中发现的最常见的Web应用程序安全漏洞。 风险按A1 -
A10的顺序列出，A1是最普遍的风险。 2017版本的OWASP Top 10和Django 2.2伪代码同样在本博文中包含。

A1 - 注入部分

十多年来，注入一直是十大风险中列出的最高风险之一，并被列为最新版本的A1风险。 当不受信任的数据用于执行非预期的命令时，注入便会发生。
一个常见的例子是SQL注入。 Django提供了一个内置的对象关系映射层，可以通过参数化来防止注入攻击。
而我们如果想防止注入的发生，我们需要提前进行设置并使用ORM模型。

```
class Person(models.Model):
       first_name = models.CharField(max_length=30)
       last_name = models.CharField(max_length=30)
```

模型用于定义数据库中表的字段和行为。

一旦定义了此模型，就为CRUD交互提供了方法。

```
newPerson = Person.objects.create(first_name=request.user.first_name, last_name=request.user.last_name)
newPerson.save()
```

create()方法中使用的request.user.first_name和request.user.last_name参数将被转义以防止SQL注入攻击。

除了ORM方法之外■Django还允许开发人员使用raw()和execute()方法输入原始SQL。 如果正确完成，这些方法也可以使用参数化。

```
from django.db import connection
...
def custom_sql(self):
 with connection.cursor() as cursor:
   cursor.execute("INSERT INTO Person (first_name, last_name) \
                   VALUES (%s, %s)", [param1, param2])
...
```

使用execute()方法时，请务必记住格式说明符（本例中为%s）应该在没有单引号'%s'的情况下使用。
虽然这可能看起来像字符串，但这并不是对参数执行了转义操作。

A2 - 认证失效

破坏的身份验证和会话管理是另一个OWASP Top
10中常见的漏洞。Django有一个基本的身份验证系统，提供身份验证，授权和会话管理，但是却缺乏其他的验证过程。
该框架不提供密码强度检查、登录尝试的限制或双因素身份验证。 这些功能只能在第三方软件包中实现，但它们不是开箱即用的。

要启用Django的身份验证框架，用户需要在INSTALLED_APPS■MIDDLEWARE部分下的settings.py文件中添加一些内容：

```
INSTALLED_APPS = [
 ...
 'django.contrib.auth',
 'django.contrib.contettypes',
 ...
 ]

MIDDLEWARE = [
 ...
 'django.contrib.sessions.middleware.SessionMiddleware',
 'django.contrib.auth.middleware.AuthenticationMiddleware',
 ...
 ]
```

由于该框架缺乏某些保护和功能，因此第2部分将进一步讨论此风险。

A3 - 敏感数据泄露

如果应用程序处理或存储敏感数据，则此漏洞便有可能产生。
Django提供了一些设置，可以在传输和静止时保持敏感数据的安全，但默认情况下并非全部启用它们。

如果应用程序正在存储密码，则可以将PASSWORD_HASHERS列表添加到setting.py文件中以指定散列算法。
默认情况下，Django使用PBKDF2算法，但可以将其他算法添加到列表中以检查现有旧密码。

在settings.py文件中还有一些其他要设置的设置：

```
SECURE_SSL_REDIRECT = True这会将所有HTTP流量重定向到`HTTPS

SESSION_COOKIE_SECURE = True这将确保会话cookie仅通过HTTPS发送

CSRF_COOKIE_SECURE = True这将确保CSRF令牌仅通过HTTPS发送
```

另一种可用于传输中数据的方法是@sensitive_variables()修饰器。 可以在函数之前添加修饰器，以防止敏感变量的值显示在错误报告中。
敏感值的变量名称作为参数传递给修饰器。

```
@sensitive_variables('first_name', 'last_name', 'credit_card', 'ssn', 'blood_type', 'etc')
def save_user_info(user)
   first_name = user.first_name
   last_name = user.last_name
   credit_card = user.ccn
   ssn = user.ssn
   ...
```

如果方法中的所有变量都是敏感的，则可以在不带参数的函数之前添加sensitive_variables修饰器。

A4 - XML外部实体

XML外部实体（XXE）攻击可导致远程命令执行、拒绝服务和数据泄露。 Django
2.2版本不易受到XXE攻击，因为XML解串器不允许DTD，获取外部实体或执行实体扩展的能力。
但是对于Django应用程序接受XML的部分，则值得我们研究，因为它可能正在使用第三方库进行XML解析。
默认情况下，许多用于XML解析的第三方库未受到针对XXE攻击的保护。

A5 - 访问控制中断

断开访问控制是缺少功能级访问控制和不安全对象引用（IDOR）的组合。Django
Framework有一个简单的权限系统，允许具有特定权限的角色、组的用户访问受保护的文件。

如果django.contrib.auth位于settings.py文件中的INSTALLED_APPS列表中，则Django将自动为应用程序中的每个Model创建，添加，更改，删除和查看权限。
例如app_label = appa的应用程序中有一个名为BlogPost的模型。 创建用户时，可以为这些用户分配这些权限。

```
...
def add_new_blogger(request):
   user = Users.objects.create_user(request.first_name, request.email, request.password)
   user.user_permissions.add('appa.create_blogpost')
   user.save()
...
```

在该示例中，使用view_blogpost权限创建了用户。
为了防止用户删除博客帖子，可以在delete_blog()函数之前添加@permission_required()修饰器以检查用户的权限。

```
@permission_required('appa.delete_blogpost')
def delete_blog(request, blog_id):
   ...
```

其他修饰器（如@login_required）可用于确保未经身份验证的用户无法查看或使用需要身份验证的应用程序部分。
诸如django-guardian之类的第三方软件包以及可用于扩展Django权限系统的功能的软件包。

A6 - 安全配置错误

Django应用程序中的安全性错误配置可能导致敏感数据暴露、破坏访问控制、XSS。
因为这将是本博客文章第2部分的大部分内容，所以在那里我将列出一个例子以便学习。在设置中请确保关闭调试：

```
DEBUG = False
```

A7 - XSS

随着越来越多的框架提供针内置保护，跨站点脚本（XSS）变得越来越少。 Django通过在以HTML格式显示之前转义特殊字符来提供防止XSS的模板。

可以使用Django render()函数将Python变量从函数传递给模板。

```
from django.shortcuts import render

def say_hi(request):
   first_name = request.user.first_name
   last_name = request.user.last_name
   context = {
       'first_name': first_name,
       'last_name': last_name,
   }
   return render(request, 'Profile/welcome.html', context)
```

上面的示例代码是将上下文中的变量发送到`welcome.html`。 当在模板中使用双重打开和关闭花括号时，Django会转义变量的值。

```
...
<h1> Hi , !</h1>
...
```

A8 - 不安全的反序列化

当不受信任的数据被反序列化为可以操作逻辑或可以执行远程代码的对象时，会发生不安全的反序列化攻击。
Django包含一个基本的序列化框架，可用于将模型序列化为其他格式。 反序列化时，框架将检查序列化数据中的字段是否存在于模型上。
如果字段不匹配，则会引发错误。 这是Django针对不安全的反序列化攻击提供的唯一保护。 防止这些攻击的最佳方法是不接受来自不受信任来源的序列化数据。

因为Django开发人员经常使用其他库进行序列化，例如Python的pickle或第三方库，所以我们将在第2部分中进一步讨论这个主题。

A9 - 使用具有已知漏洞的组件进行攻击

使用具有已知漏洞的组件是Top 10中最常见的风险之一。在撰写本文时，Django
2.2版本不包含已知漏洞，但大多数应用程序使用第三方库，因此他们不必重新发明轮子。
应使用依赖项检查等扫描程序查找具有已知漏洞的第三方软件包并更新这些库。 我们将在第2部分深入探讨这个主题。

A10 - 记录和监控

当您的应用程序没有足够的日志记录和监视时，攻击和可疑活动可能会被忽视。 默认情况下，Django使用Python本机日志记录模块进行系统日志记录。
当在`settings.py`中禁用`DEBUG`时，Django会将所有`ERROR`■`CRITICAL`日志消息通过电子邮件发送给站点管理员。
在第2部分中，我们将进一步配置Django的日志记录并启用`django.security.*`日志消息。

■■■■■■■■■■■[https://nvisium.com/blog/2019/04/18/django-vs-the-owasp-top-10-part-1.html](https://nvisium.com/blog/2019/04/18