第 3 章

认证与授权的攻击与防护

【本章重点】 掌握认证与授权的攻击定义以及特点。

【本章难点】 熟悉认证与授权的攻击原理，掌握对攻击的防护方法。

3.1 认证与授权攻击背景与相关技术分析

3.1.1 认证与授权的攻击定义

认证(Authentication)：是指验证你是谁，一般需要用到用户名和密码进行身份验证。

授权(Authorization)：是指你可以做什么，而且这发生在验证通过后，能够做什么操作。例如，对一些文档的访问权限、更改权限、删除权限，需要授权。

通过认证系统确认用户的身份。通过授权系统确认用户具体可以查看哪些数据，执行哪些操作。

3.1.2 认证与授权的特点

1. 认证

(1) 密码维度：单因素认证、双因素认证、多因素认证(密码、手机动态口令、数字证书、指纹等各种凭证)。

(2) 密码强度：

- 长度：普通应用 6 位以上；重要应用 8 位以上，考虑双因素。
- 复杂度：密码区分大小写；密码为大写字母、小写字母、数字、特殊符号中两种以上的组合；不要有(语义上)连续性的字符，如 123、abc；避免出现重复字符，如 111；不要使用用户的公开数据，或与个人隐私相关的数据作为密码，如 QQ 号、身份证号、手机号、生日、昵称、英文名、公司名等。

(3) 密码保存：密码必须以不可逆的加密算法，或单项散列函数算法加密后存储在数据库中。目前业界比较普遍的做法：在用户注册时就已经将密码哈希(MD5、SHA-1、SHA-2)后保存在数据库中，登录时验证密码仅是验证用户提交的密码哈希值是否与保存在数据库中的密码哈希值一致，即服务器不会保存密码原文。

破解 MD5 加密后密码的方法是彩虹表：收集尽可能多的明文密码和对应 MD5 值，然后通过 MD5 值反查到该 MD5 值对应的密码原文。防御方法：加盐，即 MD5

(Username+Password+Salt),其中 Salt 为一个固定的随机字符串,应该保存在服务器端的配置文件中,妥善保管。目前,MD5 与 SHA-1 都不安全,SHA-2 相对安全,不易被彩虹表碰撞出原始密码。

(4) SessionID(会话编号):当登录认证成功后,服务器分配一个唯一凭证 SessionID 作为后续访问的认证媒介。会话(Session)中会保存用户的状态和其他相关信息,当用户的浏览器发起一次访问请求时,将该用户持有的 SessionID 上传给服务器。常见的做法是将 SessionID 加密后保存在 Cookie 中,因为 Cookie 会随着 HTTP 请求头发送,且受到浏览器同源策略的保护。生成的 SessionID 要足够随机,比较成熟的开发框架都会提供 Cookie 管理、Session 管理的函数,要善用这些函数和功能。

SessionID 可能带来的风险:一旦有效的 SessionID 泄露,则在其有效期内攻击者能够以对应用户的身份访问站点。泄露方式:Cookie 泄露(SessionID 保存在 Cookie 中的情况),Referer 的 URL 中泄露(URL 携带 SessionID 的情况)。

Session Fixation 攻击:SessionID 没有及时更替、销毁,导致旧的 SessionID 仍然有效,一旦激活,便可以正常使用。

防御方法:在用户登录完成后要重新设置 SessionID;用户更换访问设备的时候要求重新登录;使用 Cookie 代替 SessionID 的作用。

Session 保持攻击:Session 是有生命周期的,当用户长时间未活动,或用户单击退出后,服务器将销毁 Session。

① 如果只依赖 Session 的生命周期控制认证的有效期,Session 保持攻击可以通过脚本持续刷新页面(发送访问请求)保持 Session 的活性。

② 如果 SessionID 保存在 Cookie 中,依赖 Cookie 的定时失效机制控制认证的有效期,那么 Session 保存攻击可以手动修改 Cookie 的失效时间,甚至将 Cookie 设置为永久有效的 Third-party Cookie,以此延长 Session 的活性。

防御方法:用户登录一定时间后强制销毁 Session;当用户的客户端(IP、UserAgent、Device)发生变化的时候要求重新登录;每个用户只允许拥有一个 Session。

(5) 单点登录(Single Sign On,SSO):统一认证,缺点是风险集中,单点一旦被攻破,影响范围涉及所有用单点登录的系统,降低这种风险的办法是在一些敏感的系统里再单独实现一些额外的认证机制(如网上支付平台,在付款前要求用户输入支付密码,或通过手机短信验证用户身份)。

目前业内最开放和流行的单点登录系统是 OpenID(一个开放的单点登录框架),其使用 URI 作为用户在互联网上的身份标识。每个用户将拥有一个唯一的 URI。用户登录网站时只提交他的 OpenID(URI)以及 OpenID 的提供者,网站就会将用户重定向到 OpenID 的提供者进行认证,认证完成后重定向回网站。

例如,用户 Ricky 在 OpenID 服务的提供者 openidprovider.com 拥有一个 URI,即 ricky.openidprovider. com;此时他准备访问某网站的一个页面(www. xxx. com/yyy.html),在 xxx 网站的登录界面会提示请用 OpenID 登录;于是,Ricky 输入他的 OpenID(URI),即 ricky.openidprovider.com,然后登录;页面将跳转到 openidprovider.com 站点,进入认证阶段;认证成功后,页面自动跳转到 www.xxx.com/yyy.html,如果认

证失败，则不会跳转。

2. 授权

Web应用中，根据访问客体的不同，常见的访问控制可分为基于URL的访问控制（常用）、基于方法的访问控制和基于数据的访问控制。

（1）垂直权限管理：定义角色，建立角色与权限的对应关系——基于角色的访问控制（Role-Based Access Control，RBAC）。用户→角色→权限。例如，Spring Security中的权限管理（功能强大，但缺乏一个管理界面可供用户灵活配置，每次调整权限都要重新修改配置文件或代码）；PHP框架Zend Framework中使用Zend ACL做权限管理。使用RBAC模型管理权限时应当使用“最小权限原则”“默认拒绝”。

（2）水平权限管理：水平权限问题出现在RBAC模型中的同一种角色的权限控制上，系统只验证了能访问数据的角色，但没有反过来再对用户与数据的权限关系进行细分，此时需要基于数据的访问控制。

（3）OAuth：OAuth是一个在不提供用户名和密码的情况下，授权第三方应用访问Web资源的安全协议。三个角色：用户（User）、服务提供方（Server）、资源持有者（Resource Owner）。现在的版本是OAuth 2.0。

没必要自己完全实现一个OAuth协议，使用第三方实现的OAuth库是一个不错的选择。

3.1.3　认证与授权攻击产生的原理

1. 权限

很多系统（如CRM、ERP、OA）中都有权限管理，其中的目的一个是为了管理公司内部人员的权限，另外一个是避免人人都有权限，一旦账号泄露，对公司带来负面影响。

权限一般分为两种：访问权限和操作权限。访问权限即某个页面的权限，对于特定的一些页面，只有特定的人员才能访问。而操作权限指的是页面中具体到某个行为，肉眼能看到的可能就是一个审核按钮或提交按钮。

权限的处理方式可以分为两种：用户权限和组权限。设置多个组，不同的组设置不同的权限，而用户设置到不同的组中，那就继承了组的权限，这种方式就是组权限管理，一般使用这种方式管理。用户权限管理则比较简单，对每个用户设置权限，而不是拉入某个组里面，这种方式灵活性不强，用户多的时候比较费劲，每次都要设置很久的权限，而一部分用户权限是有共性的，所以组权限是目前常用的处理方式。

在权限方面，还包括数据库的权限、网站管理的权限以及API/Service的权限。

DBA需要控制好IDC的数据库权限，而不是将用户权限设置为*.*，需要建立专门供应用程序使用的账号，并且需要对不同的数据库和不同的表赋予权限，专门供应用程序使用的账号就不能进行更改表、更改数据库及删除操作，否则如果有SQL注入漏洞或程序中有Bug，黑客就能轻易连接到数据库获取更多的信息。因为DBA账号除了可以更改数据库结构、数据及配置外，还可以通过LOAD DATA INFILE读取某个文件，相当于整台服务器都被控制了。

API可以分为Private API和Open API。Private API一般是不希望外网访问的，如

果架设在内网，最好使用内网IP访问，如果是公网，最好设置一定的权限管理。Open API的权限就相对复杂很多，除了校验参数正确性，还须校验用户是否在白名单中，若在白名单里，还须校验用户对应的权限，有些时候还需要考虑是否要加密传输等。

2. 密码猜测

以下哪种错误提示更合适呢？

- 输入的用户名不正确。
- 输入的密码不正确。
- 输入的用户名或密码不正确。

前面两种提示信息其实是在暗示用户正确输入了什么，哪个不正确。第三种提示信息比较模糊，可能是用户名错误，也可能是密码错误。如果非要说前两种提示信息更准确，更适于普通用户，就会给黑客们带来可乘之机，实在不知道到底是哪个错误，难度增加不少。若使用工具或批处理脚本强制枚举破解，则需要的时间更多。

2011年11月22日，360安全中心发布了中国网民最常用的25个“弱密码”：000000、111111、11111111、112233、123123、123321、123456、12345678、654321、666666、888888、abcdef、abcabc、abc123、a1b2c3、aaa111、123qwe、qwerty、qweasd、admin、password、p@ssword、passwd、iloveyou、5201314。

如何应对密码猜测攻击呢？一般有以下3种方案：

- 超过错误次数账户锁定。
- 使用RSA/验证码。
- 使用安全性高的密码策略。

很多网站将三种方案结合起来使用。另外，在保存密码到数据库时，也一定要检查是否经过严格的加密处理，不要再出现某天网站被暴库了结果却保存的是明文密码。

3. 找回密码的安全性

最不安全的做法是在邮件内容中发送明文新密码，一旦邮箱被盗，对应网站的账号也会被盗；一般做法是在邮件中发送修改密码链接，测试时就需要特别注意用户信息标识是否加密、加密方法以及是否易破解；还有一种做法是修改密码时回答问题，问题回答正确才能进行修改。

4. 注册攻击

常见的是恶意注册，以避免注册后被恶意搜索引擎爬取，在线机器人投票，注册垃圾邮箱等。缓解注册攻击的方法：使用RSA/验证码。

5. Cookie安全

Cookie中记录着用户的个人信息、登录状态等。使用Cookie欺骗可以伪装成其他用户获取隐私信息等。

常见的Cookie欺骗有以下4种方法：

- 设置Cookie的有效期。
- 通过分析多账户的Cookie值的编码规律，使用破解编码技术任意修改Cookie的值达到欺骗目的，这种方法较难实施。
- 结合XSS攻击上传代码获取访问页面用户Cookie的代码，获得其他用户的

Cookie。

- 通过浏览器漏洞获取用户的 Cookie，这种方法需要非常熟悉浏览器。

如何防范？

- 不要在 Cookie 中保存敏感信息。
- 不要在 Cookie 中保存没有经过加密的或者容易被解密的敏感信息。
- 对从客户端取得的 Cookie 信息进行严格校验，如登录时提交的用户名、密码的正确性。
- 记录非法的 Cookie 信息进行分析，并根据这些信息对系统进行改进。
- 使用 SSL 传递 Cookie 信息。
- 结合 Session 验证对用户访问授权。
- 及时更新浏览器漏洞。
- 设置 HttpOnly 增强安全性。
- 实施系统安全性解决方案，避免 XSS 攻击。

6. Session 安全

服务端和客户端之间通过 Session 连接沟通。当客户端的浏览器连接到服务器后，服务器就会建立一个该用户的 Session。每个用户的 Session 都是独立的，并且由服务器维护。每个用户的 Session 由一个独特的字符串识别，成为 SessionID。用户发出请求时，所发送的 http 表头内包含 SessionID 的值。服务器使用 http 表头内的 SessionID 识别是哪个用户提交的请求。一般 SessionID 传递方式：URL 中指定 Session 或存储在 Cookie 中，后者广泛使用。

会话劫持是指攻击者利用各种手段获取目标用户的 SessionID。一旦获取到 SessionID，攻击者可以利用目标用户的身份登录网站，获取目标用户的操作权限。

攻击者获取目标用户 SessionID 的方法：

- 暴力破解：尝试各种 SessionID，直到破解为止。
- 计算：如果 SessionID 使用非随机的方式产生，那么就有可能计算出来。
- 窃取：使用网络截获、XSS、CSRF 攻击等方法获得。

如何防范？

- 定期更改 SessionID，这样，每次重新加载都会产生一个新的 SessionID。
- 只从 Cookie 中传送 SessionID 结合 Cookie 验证。
- 只接受 Server 产生的 SessionID。
- 只在用户登录授权后生成 Session 或只在用户登录授权后变更 Session。
- 为 SessionID 设置 Time-Out 时间。
- 验证来源，如果 Refer 的来源是可疑的，就删除 SessionID。
- 如果用户代理 user-agent 变更，就重新生成 SessionID。
- 使用 SSL 连接。
- 防止 XSS、CSRF 漏洞。

3.2 认证与授权攻击经典案例重现

3.2.1 试验 1: Zero 网站能获得管理员身份数据

缺陷标题：网站 http://zero.webappsecurity.com/在地址栏追加 admin 可进入管理员页面

测试平台与浏览器：Windows 10 + IE11 或 Chrome 45.0 浏览器

测试步骤：

(1) 打开网站 http://zero.webappsecurity.com/。

(2) 在地址栏后追加 admin，按 Enter 键。

期望结果：浏览器提示无法找到网页，或者出现管理员登录页面。

实际结果：跳转到管理员页面，单击 Users 链接能看到系统中所有的用户名与密码，结果如图 3-1 和图 3-2 所示。

图 3-1 进入管理员页面

【攻击分析】

这是典型的身份认证与会话管理方面的安全问题，2017 年，失效的身份认证排在全球 Web 安全第二位。身份认证，最常见的是登录功能，往往是提交用户名和密码，在安全性要求更高的情况下，有防止密码暴力破解的验证码、基于客户端的证书、物理口令卡等。

会话管理，HTTP 本身是无状态的，利用会话管理机制实现连接识别。身份认证的结果往往是获得一个令牌，通常放在 Cookie 中，之后对用户身份根据这个授权的令牌进行识别，而不需要每次都登录。

用户身份认证和会话管理是一个应用程序中最关键的过程，有缺陷的设计会严重破坏这个过程。在开发 Web 应用程序时，开发人员往往只关注 Web 应用程序所需的功能。由于这个原因，开发人员通常会建立自定义的认证和会话管理方案。但要正确实现这些方案却很难，结果这些自定义的方案往往在如下方面存在漏洞：退出、密码管理、超时、记

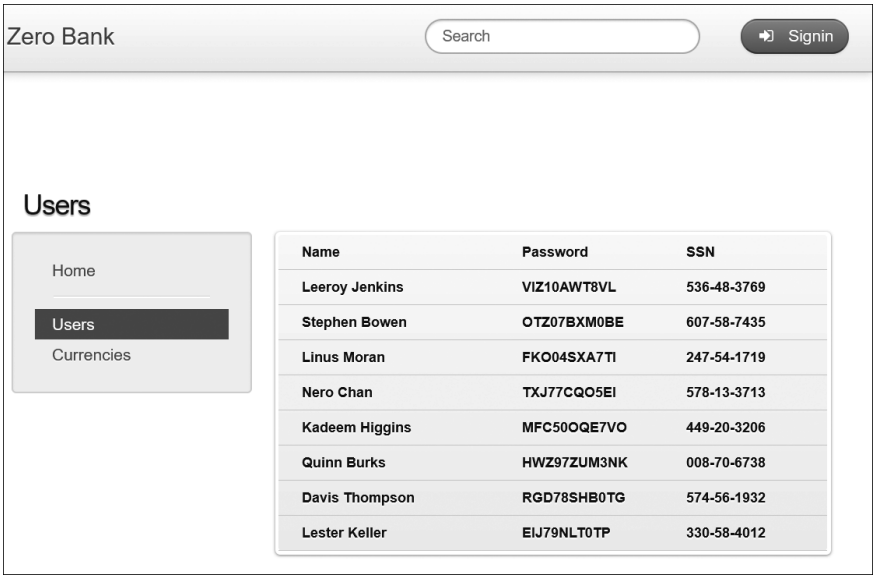

图 3-2　查看到系统中所有的用户名与密码

住我、账户更新等。因为每个系统实现都不同，业务定义也不同，要找出这些漏洞，有时会很困难。

如何验证程序是否存在失效的认证和会话管理？

最需要保护的数据是认证凭证(Credentials)和会话 ID。

(1) 当存储认证凭证时，是否总是使用哈希或加密保护？

(2) 认证凭证是否可猜测，或者能够通过薄弱的账户管理功能(例如账户创建、密码修改、密码恢复、弱会话 ID)重写？

(3) 会话 ID 是否暴露在 URL 里(例如，URL 重写)？

(4) 会话 ID 是否容易受到会话固定(Session Fixation)的攻击？

(5) 会话 ID 会超时吗？用户能退出吗？

(6) 成功注册后，会话 ID 会轮转吗？

(7) 密码、会话 ID 和其他认证凭据是否只通过 TLS(传输层安全)连接传输？

3.2.2　试验 2: CTFPostbook 用户 A 能修改用户 B 的数据

缺陷标题：CTFPostbook 网站＞用户 A 登录后，可以修改其他用户的数据

测试平台与浏览器：Windows 10 ＋ IE11 或 Chrome 浏览器

测试步骤：

(1) 打开国外安全夺旗比赛网站主页 https://ctf.hacker101.com/ctf，如果已有账户，则直接登录；如果没有账户，请注册一个账户并登录。

(2) 登录成功后，请进入 Postbook 网站 https://ctf.hacker101.com/ctf/launch/7，如图 3-3 所示。

(3) 单击 Sign up 链接注册两个账户，如 admin/admin，abcd/bacd。

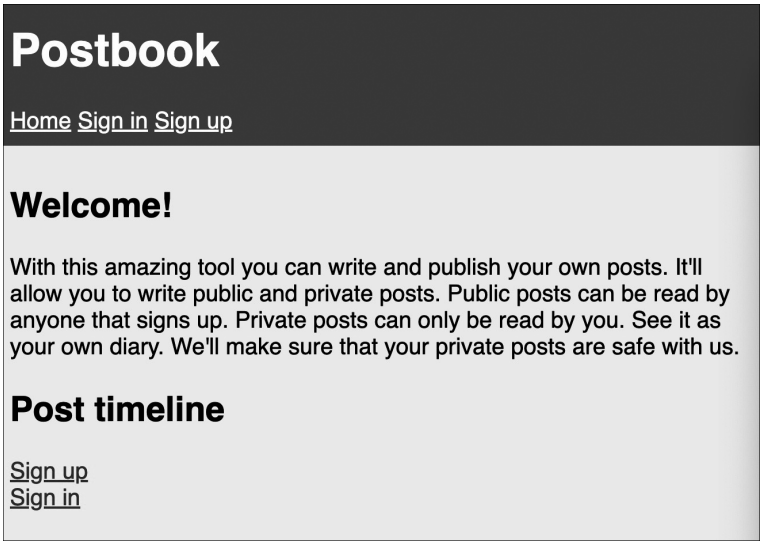

图 3-3　进入 Postbook 网站

(4) 用 admin/admin 登录,然后创建两个帖子,再用 abcd/abcd 登录创建两个帖子。

(5) 观察 abcd 用户修改帖子的链接：XXX/index.php? page=edit.php&id=5。

(6) 篡改步骤(5)URL 中的 id 为 1,2 等,以 abcd 身份修改 admin 或其他用户的帖子,如图 3-4 所示。

图 3-4　用户 abcd 篡改 URL,修改其他用户的帖子

期望结果：因身份权限不对，拒绝访问。

实际结果：用户 abcd 能不经其他用户许可，任意修改其他用户的数据，成功捕获 Flag，如图 3-5 所示。

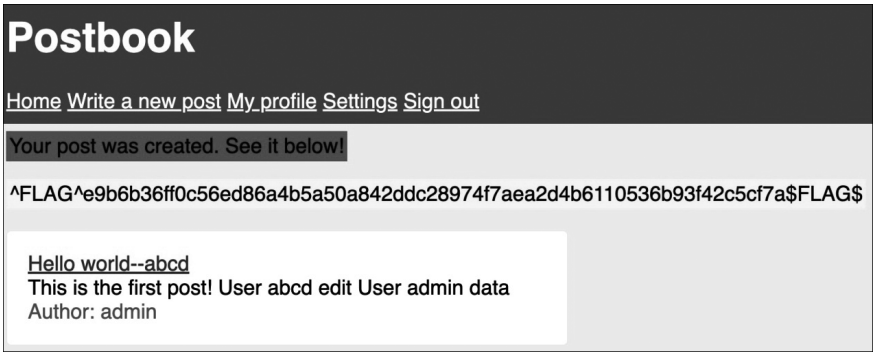

图 3-5　用户 abcd 成功修改用户 admin 的帖子，成功捕获 Flag

【攻击分析】

在 Web 安全测试中，权限控制出错的例子非常多，例如：

(1) 用户 A，在电子书籍网站购买了三本电子书，然后用户 A 单击书名就能阅读这些电子书，每本电子书都有 bookid，用户 A 通过篡改 URL，把 bookid 换成其他 id，就有可能可以免费看别人购买的电子书籍。

(2) 普通用户 A，拿到了管理员的 URL，试图去运行，结果发现自己也能操作管理员的界面。

(3) 普通用户 A，找到修改/删除自己帖子的 URL，通过篡改 URL 把帖子 id 改成其他人的 id，就可以修改/删除别人的帖子。

软件工程师在实现基本功能后，需要考虑不具有权限的人是否能直接进行这些非法操作。

3.2.3　试验 3: CTF Postbook 用户 A 能用他人身份创建数据

缺陷标题：CTFPostbook 网站＞用户 A 登录后，可以用他人身份创建数据

测试平台与浏览器：Windows 10 ＋ IE11 或 Chrome 浏览器

测试步骤：

(1) 打开国外安全夺旗比赛网站主页 https://ctf.hacker101.com/ctf，如果已有账户，则直接登录；如果没有账户，请注册一个账户并登录。

(2) 登录成功后，请进入 Postbook 网站 https://ctf.hacker101.com/ctf/launch/7。

(3) 单击 Sign up 链接注册两个账户，如 admin/admin，abcd/bacd，如果已有账户，请忽略此步。

(4) 用 abcd/abcd 登录，单击 Write a new post 链接，在这个页面右击，选择“检查(Inspect)”，出现图 3-6。

(5) 观察右端源代码，发现 Title(title)字段是必填项 required，Post(body)字段也是

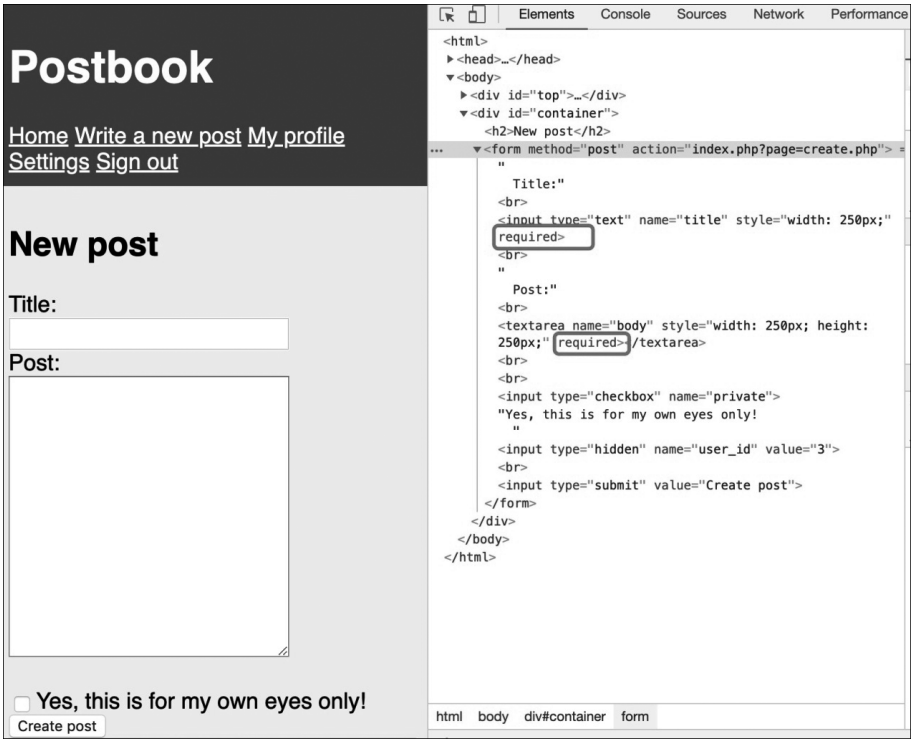

图 3-6　创建一个新帖 Write a new post

必填项 required，当前的帖子是登录用户 user_id 为 3。

(6) 篡改步骤(5)中的源代码，将 Title 后面的 required 删除，将 Post 后的 required 删除，将 user_id 改为 1。

期望结果：因必填字段未填，并且因身份权限不对，故拒绝访问。

实际结果：用户 abcd 能绕过客户端必填字段检查，同时以系统第一个用户 admin 身份任意创建数据，成功捕获 Flag，如图 3-7 所示。

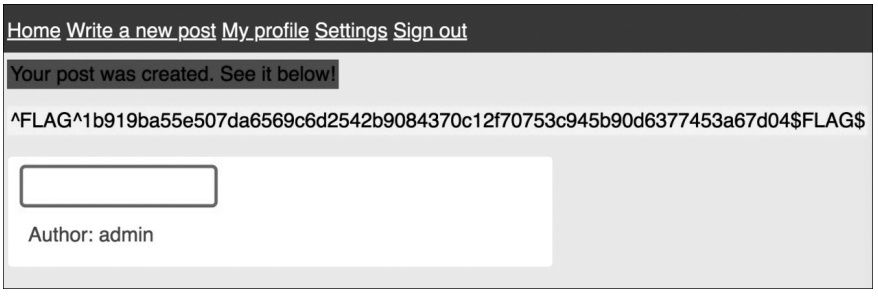

图 3-7　用户 abcd 成功以 admin 身份创建一个空白帖子，成功捕获 Flag

【攻击分析】

本例实际至少用到 3 种攻击方法，所以有时候系统的漏洞可以用多种手段进行攻击。

(1) 客户端绕行：本书第 15 章有详细讲解，就是常见的限制只有客户端的防护，没有

服务器端的防护，这样就导致攻击者能通过各种工具或手段轻松绕过客户端的防护，直接把非法数据提交到后台数据库。本例中，如果没有删掉 title 后面的 required 字段，那么空白标题是提交不成功的。

(2) HTTP 参数篡改：本书第 8 章有详细讲解，就是用户通过各自的工具或手段将原先要提交到服务器端的参数值进行篡改，后台没有做相应的防护，导致数据直接提交到后台数据库。本例中，登录的那个人 user_id 是 3，默认情况下创建/修改都是自己的帖子，但是攻击者通过各种工具或方法将 user_id 篡改成 1，一般系统的第一个用户都是 admin，所以如果后台没有做身份权限校验，就以 admin 身份创建数据了。

(3) 认证与授权错：本例是普通用户，通过篡改自己的 user_id 为其他人的 user_id，可以给系统中任何存在的一个人创建帖子，即使是管理员账户数据，也能被普通用户创建。

3.3　认证与授权攻击的正确防护方法

3.3.1　认证与授权总体防护思想

每个资源在访问前，首先要确认谁可以访问，能做什么操作。

基本授权条款见表 3-1。

表 3-1　基本授权条款

User(用户)	对象尝试执行任务或访问数据
Group(组)	需要访问资源的对象集合
Role(角色)	执行功能所需的任务集合
Rule(规则)	检查对象是否应该能够访问资源的逻辑
Permissions(权限)	用于确定对象是否应该有权访问资源的属性

常见的授权类型：

1. 自主访问控制(Discretionary Access Control，DAC)

资源所有者设置的权限，可分配授权(Assignable authorization)，由客体的属主对自己的客体进行管理，由属主自己决定是否将自己的客体访问权或部分访问权授予其他主体，这种控制方式是自主的。也就是说，在自主访问控制下，用户可以按自己的意愿，有选择地与其他用户共享他的文件。

2. 基于角色的访问控制(Role-Based Access Control，RBAC)

用户通过角色与权限进行关联。简单地说，一个用户拥有若干角色，每个角色拥有若干权限，这样就构成“用户-角色-权限”的授权模型。在这种模型中，用户与角色之间，角色与权限之间一般都是多对多的关系。

其基本思想是，对系统操作的各种权限不是直接授予具体的用户，而是在用户集合与权限集合之间建立一个角色集合。每种角色对应一组相应的权限。一旦用户被分配了适

当的角色，该用户就拥有此角色的所有操作权限。这样做的好处是，不必在每次创建用户时都进行分配权限的操作，只要分配用户相应的角色即可，而且角色的权限变更比用户的权限变更要少得多，这样将简化用户的权限管理，减少系统的开销。

3. 基于规则的访问控制（Rule-Based Access Control）

基于规则的安全策略系统中，所有数据和资源都标注了安全标记，用户的活动进程与其原发者具有相同的安全标记。系统通过比较用户的安全级别和客体资源的安全级别，判断是否允许用户进行访问。这种安全策略一般具有依赖性与敏感性。

4. 数字版权管理（Digital Rights Management，DRM）

版权保护机制，用于保护内容创建者和未授权的分发。

5. 基于时间的授权（Time Based Authorization，TBA）

根据时间对象请求，确定访问资源。

3.3.2 能引起认证与授权的错误代码段

系统没有做任何访问与授权控制，任何人通过拼凑的 URL，不用登录就可访问只有管理员可以运行的 URL。

系统仅做了部分的访问控制，许多需要登录才能访问的 URL 没有配置到登录验证中，导致任何人都可以删除、修改他人的敏感信息。

系统只做了登录认证，没有做严格的授权控制，导致用户 A 登录后，可以通过篡改 URL 修改与删除他人的资料。

错误的情况错综复杂，这里不列具体的错误代码段，请参考 3.3.3 节的正确代码段。

3.3.3 能防护认证与授权的正确代码段

如果某操作只有登录用户可以执行，则可以将 logon-config.xml 文件修改为如图 3-8 所示的代码段。

```
<!--Defined in logon-config.xml example-->
<signon-config>
    <!--From Sign On Page-->
    <signon-form-login-page>/war.context.root.login@/login/login.do</signon-form-login-page>
    <!--Error page when Sign on Fails-->
    <sign-form-error-page></sign-form-error-page>

    <!--A protected resource-->
    <security-constraint>
      <web-resource-collection>
         <web-resource-name>uploadfile action</web-resource-name>
         <url-pattern>user/fileupload/uploadAction</url-pattern>
      </web-resource-collection>
      <web-resource-collection>
         <web-resource-name>.....</web-resource-name>
         <url-pattern>......</url-pattern>
      </web-resource-collection>
    </security-constraint>
</signon-config>
```

图 3-8 只有登录用户才能操作

从这个 XML 定义，可以看到 uploadfile 操作需要登录保护。运行 user/fileupload/

uploadAction 这个 URL 将直接检查用户是否已登录网站，如果没有登录，则显示登录页面。

在实际应用中，有些 URL 只有 admin 用户才能访问，普通用户无法做到。如果这个操作只有 admin 用户可以做，那么可以修改 admin-logon-config.xml 文件为如图 3-9 所示的代码段。

```
<!--Defined in admin-logon-config.xml example-->
<signon-config>
    <!--From Sign On Page-->
    <signon-form-login-page>/war.context.root.adminlogin@/login/adminlogin.do</signon-form-login-page>
    <!--Error page when Sign on Fails-->
    <sign-form-error-page></sign-form-error-page>

    <!--Admin User protected resource-->
    <security-constraint>
      <web-resource-collection>
         <web-resource-name>delete user action</web-resource-name>
         <url-pattern>user/operation/deleteUserAction</url-pattern>
      </web-resource-collection>
      <web-resource-collection>
         <web-resource-name>.....</web-resource-name>
         <url-pattern>......</url-pattern>
      </web-resource-collection>
    </security-constraint>
</signon-config>
```

图 3-9　只有 admin 用户才能操作

对于这种情况，只有 admin 用户可以执行删除用户操作，如果攻击尝试自己拼装 user/operation/deleteUserAction 之类的 URL，则直接检查用户是否已经登录并以 admin 用户身份登录，如果没有登录，则进入登录页面。如果用低权限用户登录，则阻止他的操作。

以上两个基于角色的访问控制实际上可以做很多 Web 安全保护，但远远不够。例如，用户 A 在 BBS 或博客中发布主题，他不希望用户 B 编辑或删除它。然而，在这种情况下，如果只考虑登录情况（身份认证情况），那么用户 B 可以删除用户 A 的内容。

如果此操作只有所有者自己可以做，或高级用户可以做，或管理用户可以做，然后在 auth.xml 文件中定义为如图 3-10 所示的代码段（授权操作）。

```
<!--Defined in auth.xml for Rule-Based Access Control-->
<auth-rule>
    <name>Check the ownership of blog in website</name>
    <url-pattern>blog/blogAction.do?AT=EditBlog&amp;blogID=PARAM_PRESENT</url-pattern>
    <url-pattern>blog/blogAction.do?AT=DeleteBlog&amp;blogID=PARAM_PRESENT</url-pattern>

    <!-- we define only 3 types of user can do this(Blog Owner, Moderator, Admin) -->
    <principal-ref>
      OWNER_OF_BLOG$[system.class.com_webapp_common_auth_BlogHelper_getUserFromBlogID]$[blogID]
    </principal-ref>
    <principal-ref>
      MODERATOR_OF_BLOG$[system.class.com_webapp_common_auth_BlogHelper_getUserFromBlogID]$[blogID]
    </principal-ref>
    <principal-ref>
      ADMIN_OF_BLOG$[system.class.com_webapp_common_auth_BlogHelper_getUserFromBlogID]$[blogID]
    </principal-ref>
</auth-rule>
```

图 3-10　复杂权限定义

定义一些功能级别访问控制，例如：

用户 A 在一个网站上发布博客，用户 A 希望每个登录用户都可以查看他的博客内容，但不希望其他用户编辑或删除自己的博客。此博客的 URL 可能如下：

(1) ../blog/blogAction? AT=ViewBlog&blogID=***

(2) ../blog/blogAction? AT=EditBlog&blogID=***

(3) ../blog/blogAction? AT=DeleteBlog&blogID=***

然后，对于 ViewBlog 操作，只在 logon-config.xml 中添加它就会满足要求，但是对于 EditBlog 和 DeleteBlog 操作，如果只在 logon-config.xml 中添加登录就能访问，将导致用户 B 登录站点可以编辑或删除用户 A 的博客，这是一个巨大的安全漏洞，因此需要在 auth.xml 中使用规则定义，只有满足所有已定义的主体，才可以执行相应的操作。

在 auth.xml 中定义了博客所有者(创建这个博客的人 OWNER)、版主(该主题博客的版主 MODERATOR)、管理员(博客或网站的管理员 ADMIN)可以编辑和删除博客，其他人想访问这个链接，将出现权限不够的错误提示。

如果系统严格遵循安全设计，那么这些功能级别访问控制需要清晰的定义和实现。

3.3.4 认证与授权最佳实践

(1) 确保请求发出者具有相应权限，以执行该请求要进行的操作；如果没有，则拒绝。

(2) 拥有产品的授权政策。

(3) 每个资源在访问前，首先要确认谁可以访问，能做什么操作。

(4) 始终实现最小权限管理。

- 不要将所有进程作为 root 或 Administrator 运行。
- 使用 root 绑定到端口，然后立即切换到非特权账户。
- sudo＜任务名称＞，而不是 sudosu-。
- 如果只需要定期修改，就不要一直留下文件“可写”(writeable)。

(5) 总是失败关闭，永远不会对失败打开；验证失败，就要禁止访问。

3.4 认证与授权攻击动手实践与扩展训练

3.4.1 Web 安全知识运用训练

请找出以下网站的认证与授权攻击安全缺陷：

(1) testfire 网站：http://demo.testfire.net

(2) testphp 网站：http://testphp.vulnweb.com

(3) testasp 网站：http://testasp.vulnweb.com

(4) testaspnet 网站：http://testaspnet.vulnweb.com

(5) zero 网站：http://zero.webappsecurity.com

(6) crackme 网站：http://crackme.cenzic.com

(7) webscantest 网站：http://www.webscantest.com

(8) nmap 网站：http://scanme.nmap.org

3.4.2　安全夺旗 CTF 训练

请从安全夺旗 CTF 提供的各个应用中找出认证与授权攻击安全缺陷：

(1) A little something to get you started 应用：https://ctf.hacker101.com/ctf/launch/1

(2) Micro-CMS v1 应用：https://ctf.hacker101.com/ctf/launch/2

(3) Micro-CMS v2 应用：https://ctf.hacker101.com/ctf/launch/3

(4) Pastebin 应用：https://ctf.hacker101.com/ctf/launch/4

(5) Photo Gallery 应用：https://ctf.hacker101.com/ctf/launch/5

(6) Cody's First Blog 应用：https://ctf.hacker101.com/ctf/launch/6

(7) Postbook 应用：https://ctf.hacker101.com/ctf/launch/7

(8) Ticketastic：Demo Instance 应用：https://ctf.hacker101.com/ctf/launch/8

(9) Ticketastic：Live Instance 应用：https://ctf.hacker101.com/ctf/launch/9

(10) Petshop Pro 应用：https://ctf.hacker101.com/ctf/launch/10

(11) Model E1337-Rolling Code Lock 应用：https://ctf.hacker101.com/ctf/launch/11

(12) TempImage 应用：https://ctf.hacker101.com/ctf/launch/12

(13) H1 Thermostat 应用：https://ctf.hacker101.com/ctf/launch/13

(14) Model E1337 v2-Hardened Rolling Code Lock 应用：https://ctf.hacker101.com/ctf/launch/14

(15) Intentional Exercise 应用：https://ctf.hacker101.com/ctf/launch/15

(16) Hello World! 应用：https://ctf.hacker101.com/ctf/launch/16

提醒 1：可以在 http://collegecontest.roqisoft.com/awardshow.html 中查阅历年全国高校大学生在这些网站中发现的更多安全相关的缺陷。

提醒 2：本章中讲解的安全技术，因为对系统的破坏性很大，为避免产生法律纠纷，请不要乱用。请在自己设计的网站上测试；或者你已得到授权允许做安全测试，才可以用各种安全测试技术或安全测试工具进行安全测试(本章动手实践与扩展训练中所举的样例网站，都是公开可以做各种安全测试的)。