关于OWASP常见Web漏洞的调研报告

武昱涵

SQL注入

基本原理

SQL程序未对用户输入做适当的验证和过滤，导致用户可以在查询中插入SQL代码，从而进行未授权的操作，例如获取数据库信息、提权、增删改等

注入类型

分类依据	类型
获取信息的方式	布尔盲注，时间盲注，报错注入 ，union查询注入，堆叠注入等
提交方式	GET、POST、COOKIE、HTTP 注入等
注入点类型	数字类型的注入、字符串类型的注入、搜索型注入等
其他注入	二次注入、User-Agent 注入、文件读写、宽字节注入 、万能密码等

利用条件

先通过以下几个方法判断是否存在注入：

- 加单引号、双引号、单括号、双括号，如果报错则可能存在漏洞
- 在URL后面加and 1=1，and 1=2，查看页面显示是否相同，如果不同则一定存在注入漏洞
- 可以通过语句执行时间判断，但不能太简单，一个例子是`sleep()`和`benchmark()`两个函数

最易出现注入漏洞的点就是登陆界面，以及其他需要与数据库交互的场景，例如user-agent和cookie

防御方法

这里举几个经典的例子：

- 预编译，先形成语法树，如此用户的输入不能被认为是指令
- 使用正则表达式过滤敏感字符串
- 严格限制用户权限
- 控制报错信息的显示，防止攻击者利用报错信息发现漏洞

绕过方法

这里举几个经典的例子：

- 宽字节注入：利用汉字编码的特点，注入一个字节，使得程序将其与前面的反斜线合并认为是一个汉字，从而绕过转义字符
- 绕过引号限制：使用hex编码或者`CHAR()`函数
- 绕过字符串黑名单：使用`concat()`拆散的字符连接起来

参考资料

https://www.freebuf.com/articles/web/404072.html

SSRF服务端请求伪造

基本原理

顾名思义，攻击者伪造了一个看起来是由服务端发起的请求，原因一般是服务端开启了从其他服务端应用获取数据的功能，但是没有对**目标地址**做限制和过滤，一般用于攻击从外网无法访问的内部系统。攻击者一般可以实现5种目的：

- 可以对外网、服务器所在内网、本地进行端口扫描，获取一些服务的 banner 信息
- 攻击运行在内网或本地的应用程序（比如溢出）
- 对内网 WEB 应用进行指纹识别，通过访问默认文件实现
- 攻击内外网的 web 应用，主要是使用 GET 参数就可以实现的攻击（比如 Struts2，sqli 等）
- 利用 `file` 协议读取本地文件等

利用条件

主要涉及的函数：

- `file_get_contents()`，该函数把整个文件读入一个字符串中
- `fsockopen()`，该函数会使用socket与服务端建立TCP连接，传输原始数据
- `curl_exec()`，该函数与目标URL发起一个会话，功能非常丰富，详见参考资料

进一步地，可以利用以下协议进行SSRF攻击：

- dict协议（字典协议,探测端口指纹信息,写入和反弹shell)
- file协议（读取文件,如果遇到特殊字符使用filter以base64读取)
- http协议（常用于file_get_contents函数)
- ftp协议（扫描端口极其好用)
- gopher协议

防御方法

这里举几个经典的例子：

- 控制错误信息，防止用户根据错误信息判断远程服务器的端口状态
- 限制请求的端口为HTTP常用端口，例如80，443，8080等
- 禁用不需要的协议，仅允许HTTP/HTTPS
- 设置URL白名单或者限制内网IP，如果对方读取环回地址则立即终止运行

绕过方法

这里举几个经典的例子：

- 更改IP地址写法，例如点分8进制，点分16进制，10进制整数，16进制整数等
- 对于URL解析，也有一些绕过方法：
    - `http://www.baidu.com@192.168.0.1/`与`http://192.168.0.1`指向相同的内容
    - 可以指向任意IP的域名`xip.io`

- 短地址
- 用句号代替点
- 利用带圈字符

参考资料

https://www.freebuf.com/vuls/365150.html

https://www.cnblogs.com/beili/p/9855895.html

XSS跨站脚本攻击

基本原理

类似于SQL注入，攻击者将恶意脚本插入网页中，当用户浏览时，脚本就会运行。原因是WEB服务器在读取**用户可控数据**输出到HTML页面的过程中没有进行安全处理，用户可控制数据主要包括url、参数、HTTP头部字段（cookie、referer、HOST等）、HTTP请求正文等。

XSS攻击大致可分为3种：

- 反射型XSS：将脚本注入到URL参数中，脚本没有被存储，只能在单次请求中生效
- 存储型XSS：也叫持久型XSS，将脚本存储在服务器上，一般通过POST请求表单实现（如评论区、登录框、留言板等），可以多次生效
- DOM-XSS：本质上是一种特殊的反射型XSS，源于前端JavaScript对用户输入数据的不当处理，而非来自服务器端响应。这种类型的XSS利用的是浏览器DOM环境，而不是HTML文档本身的漏洞。

简而言之，一般的反射型和持久型XSS的数据来源是后端，而DOM-XSS的数据来源是前端

利用条件

这里举几个经典的例子：

- cookie窃取：窃取有效cookie，而后冒充合法用户进行非授权操作，例如会话劫持
- 网站挂马：通过运行恶意脚本自动下载恶意软件或者重定向到钓鱼网站
- 隐私窃取：窃取服务器中的用户隐私数据

防御方法

这里举几个经典的例子：

- 输入过滤与检查：去除或转义特殊字符，限制输入内容长度
- 使用CSP（Content Security Policy，内容安全策略）
- 设置session cookie为HTTPOnly属性，防止通过JavaScript访问
- 使用DOMPutify等前端安全库

绕过方法

这里举几个经典的例子：

- 用`/`代替空格
- 大小写绕过，双写关键字，字符拼接
- 编码绕过：ASCII，Unicode，hex，base64
- 类似SSRF的绕过，更改IP地址的写法

参考资料

https://cloud.tencent.com/developer/article/2413289

https://blog.csdn.net/rendaa/article/details/113592738

https://www.cnblogs.com/sfsec/p/15178028.html

CSRF跨站请求伪造

基本原理

在已登录的web程序上劫持用户身份进行非法操作，也被称为one-click attack或者session riding，注意不要与XSS混淆。CSRF的产生有以下几个主要原因：

- http协议使用session在服务端保存用户的个人信息，客户端浏览器用cookie标识用户身份
- cookie的认证只能确保是某个用户发送的请求,但是不能保证这个请求是否是"用户自愿的行为"
- 综上所述，当用户登录了某个页面的web应用，如果点击了具有CSRF恶意代码的URL，就会触发

CSRF的简单分类：

分类依据	类别
请求类型	GET型，POST型
攻击方式	HTML CSRF、JSON HiJacking、Flash CSRF 等

CSRF和XSS的不同

（1）XSS主要是获取cookie，达到控制客户端的目的；CSRF主要是劫持用户身份，让客户端做一些“不愿做”的事情

（2）危害上来说，XSS较大一些

（3）应用难度上来说，CSRF需要用户登录，并访问恶意URL，条件比较苛刻，而XSS只需要一次点击或者存储到服务器即可（实际上，可以用XSS把CSRF的恶意URL注入到页面中）

利用条件

根据漏洞原理，应当满足以下几个条件：

- 用户登录了网站，且生成了cookie
- 网站中有植入了CSRF恶意代码的URL
- 用户点击了对应的URL

例如：抓包后修改referer再重新提交，如果能请求就存在CSRF漏洞

防御方法

这里举几个经典的例子：

- 登录等操作时使用验证码
- http头中使用并验证token（随机数）和referer（告知服务器用户访问当前页面之前的位置）字段
- 修改密码时必须填入原密码

绕过方法

这里举几个经典的例子：

- 使用了token和referer但是传输方式不安全（例如以GET提交，使用JS读取等）
- 伪造referer头
- 社会工程学：诱使用户泄露验证码

参考资料

https://www.freebuf.com/articles/web/247866.html

XXE（XML外部实体注入）

基本原理

XML注入类似于SQL注入，攻击者在输入内容中注入XML字段以达到目的（例如注册用户）。

而XXE相比于一般的XML注入，攻击面更广，危害更大。其基本原理是应用在解析XML输入时，没有禁止外部实体的加载，攻击者可以借此加载恶意程序，实现网络扫描、文件读取等。

利用条件

一个例子：在登录界面抓包，将账号密码部分替换成读取文件的代码，或者查看端口开放情况，查看是否有结果

如果代码不明显，可以使用burpsuit的爬虫工具等进行扫描

防御方法

这里举几个经典的例子：

- 过滤用户输入的数据，例如尖括号、一些关键字等
- 直接在代码层面禁用外部实体

绕过方法

这里举几个经典的例子：

- 如果关键词被过滤（SYSTEM，INTITY等），可以用编码绕过，如hex，base64等
- 如果http被过滤，可以采用data、file，php://filter等协议绕过

参考资料

https://www.cnblogs.com/l0nmar/p/13339015.html

https://www.cnblogs.com/chu-jian/p/17489142.html

逻辑漏洞

基本原理

逻辑漏洞是指由于程序逻辑不严谨导致一些逻辑分支处理错误造成的漏洞。

利用条件

在实际开发中，因为开发者水平不一没有安全意识，而且业务发展迅速内部测试没有及时到位，所以常常会出现类似的漏洞。总得来说有三个主要方面：

- **缺乏适当的权限控制**：系统未对用户的权限进行有效验证。
- **业务逻辑设计不当**：应用程序的业务流程未考虑到所有可能的攻击路径。
- **输入验证不足**：用户输入未经过严格验证，导致应用程序执行意外操作。

下面举一些经典例子：

- 购买
  - 修改购买数量为负数
  - 修改金额为负数
  - 重放成功的请求
- 注册
  - 覆盖注册
  - 尝试重复用户名
  - 注册遍历猜解已有账号
- 密码
  - 密码未使用哈希算法保存
  - 没有验证用户设置密码的强度
- 验证码
  - 验证码可重用
  - 验证码可预测
  - 验证码用于手机短信/邮箱轰炸
  - 验证码可OCR或使用机器学习识别
- 随机数
  - 使用不安全的随机数发生器
  - 使用时间等易猜解的因素作为随机数种子

防御方法

总得来说有以下几个方面：

- 实施严格的权限控制，遵守最小权限原则
- 完善业务逻辑审计流程
- 对用户输入进行严格的验证和清理
- 实施监控和日志记录

绕过方法

这里举几个经典的例子：

- 攻击者直接修改请求参数或者利用API接口，绕过权限检查
- 在竞态条件中，攻击者可快速发送多个请求，干扰系统状态（即并发请求）

- 社会工程学

参考资料

https://wiki.wgpsec.org/knowledge/web/logical.html

https://websec.readthedocs.io/zh/latest/vuln/logic.html