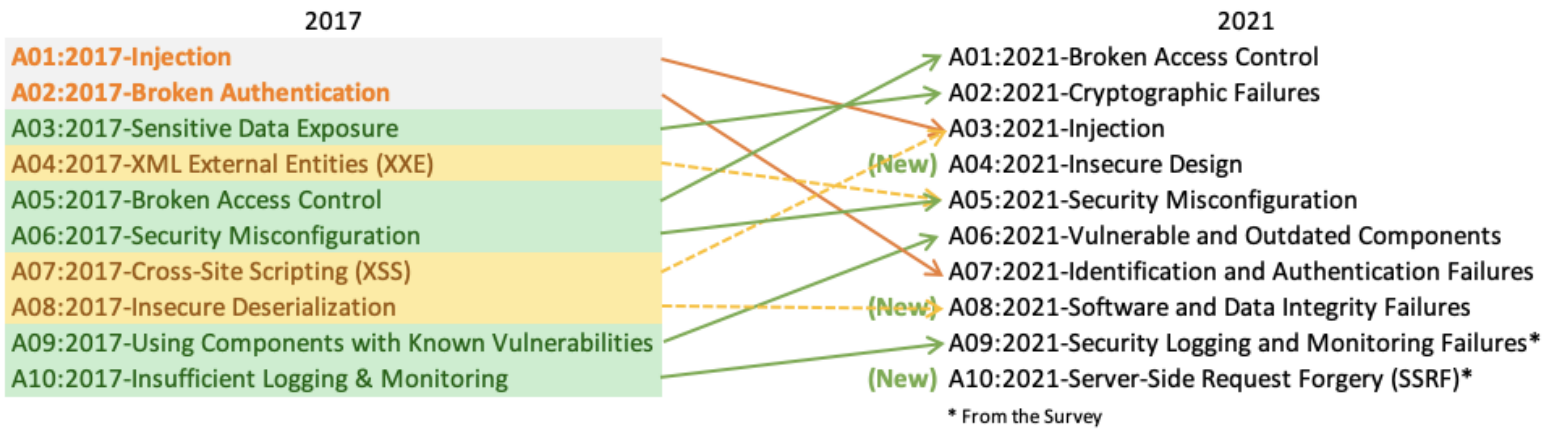
常见WEB漏洞攻击原理及入侵检测配置

1 常见WEB漏洞

OWASP TOP 10

Open Worldwide Application Security Project (OWASP) 是一个致力于提高软件安全性的非营利性基金会。它以“开放式社区”模式运作，这意味着任何人都可以参加与 OWASP 相关的在线聊天、项目等，并为其做出贡献。从在线工具和视频到论坛和活动，OWASP 确保其所有内容均为免费，而且可以在其网站上轻松访问。

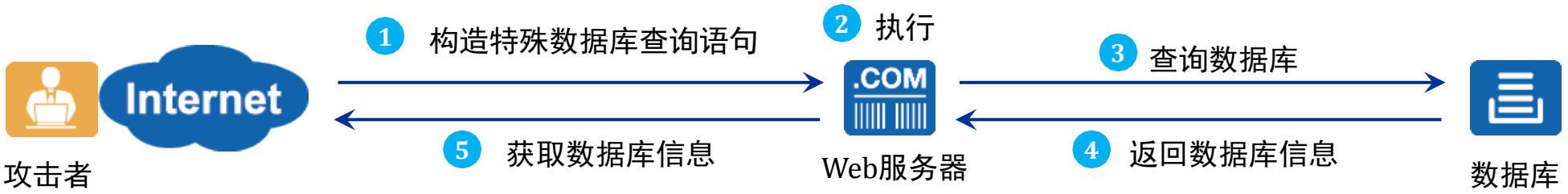
OWASP Top 10 提供了十大最关键的 Web 应用程序安全风险的排名和补救指南。基于 OWASP 开放式社区贡献者的广泛知识和经验，该报告采纳了来自世界各地的安全专家的共识。风险等级根据所发现的安全缺陷的频率、所发现漏洞的严重程度及其潜在的影响程度进行排序。该报告的目的是让开发人员和 Web 应用程序安全人员深入了解最常见的安全风险，以便他们能够将报告的结果和建议纳入自己的安全实践中，从而尽量减少其应用程序中已知的风险。



A01：2021-失效的访问控制 Broken Access Control	从第5位上升成为Web应用程序安全风险最严重的类别；提供的数据表明，平均3.81%的测试应用程序具有一个或多个CWE，且此类风险中CWE总发生漏洞应用数超过31.8万次。在应用程序中出现的34个匹配为“失效的访问控制”的CWE次数比任何其他类别都多。
A02：2021-加密机制失效 Cryptographic Failures	排名上升一位。其以前被称为“A3:2017-敏感信息泄漏（Sensitive Data Exposure）”。敏感信息泄漏是常见的症状，而非根本原因。更新后的名称侧重于与密码学相关的风险，即之前已经隐含的根本原因。此类风险通常会导致敏感数据泄露或系统被攻破。
A03：2021-注入 Injection	排名下滑两位。94%的应用程序进行了某种形式的注入风险测试，发生安全事件的最大率为19%，平均率为3.37%，匹配到此类别的33个CWE共发生27.4万次，是出现第二多的风险类别。原“A07:2017-跨站脚本（XSS）”在2021年版中被纳入此风险类别。
A04：2021-不安全设计 Insecure Design	2021年版的一个新类别，其重点关注与设计缺陷相关的风险。如果我们真的想让整个行业“安全左移”，我们需要更多的威胁建模、安全设计模式和原则，以及参考架构。不安全设计是无法通过完美的编码来修复的；因为根据定义，所需的安全控制从来没有被创建出来以抵御特定的安全攻击。
A05：2021-安全配置错误 Security Misconfiguration	排名上升一位。90%的应用程序都进行了某种形式的配置错误测试，平均发生率为4.5%，超过20.8万次的CWE匹配到此风险类别。随着可高度配置的软件越来越多，这一类别的风险也开始上升。原“A04:2017-XML External Entities(XXE) XML外部实体”在2021年版中被纳入此风险类别。
A06：2021-自带缺陷和过时的组件 Vulnerable and Outdated Components	排名上升三位。在社区调查中排名第2。同时，通过数据分析也有足够的数据进入前10名，是我们难以测试和评估风险的已知问题。它是唯一一个没有发生CVE漏洞的风险类别。因此，默认此类别的利用和影响权重值为5.0。原类别命名为“A09:2017-Using Components with Known Vulnerabilities 使用含有已知漏洞的组件”。
A07：2021-身份识别和身份验证错误 Identification and Authentication Failures	排名下滑五位。原标题“A02:2017-Broken Authentication失效的身份认证”。现在包括了更多与识别错误相关的CWE。这个类别仍然是Top 10的组成部分，但随着标准化框架使用的增加，此类风险有减少的趋势。
A08：2021-软件和数据完整性故障 Software and Data Integrity Failures	2021年版的一个新类别，其重点是：在没有验证完整性的情况下做出与软件更新、关键数据和CI/CD管道相关的假设。此类别共有10个匹配的CWE类别，并且拥有最高的平均加权影响值。原“A08:2017-Insecure Deserialization不安全的反序列化”现在是本大类的一部分。
A09：2021-安全日志和监控故障 Security Logging and Monitoring Failures	排名上升一位。来源于社区调查（排名第3）。原名为“A10:2017-Insufficient Logging & Monitoring 不足的日志记录和监控”。此类别现扩大范围，包括了更多类型的、难以测试的故障。此类别在CVE/CVSS 数据中没有得到很好的体现。但是，此类故障会直接影响可见性、事件告警和取证。
A10：2021-服务端请求伪造 Server-Side Request Forgery	2021年版的一个新类别，来源于社区调查（排名第1）。数据显示发生率相对较低，测试覆盖率高于平均水平，并且利用和影响潜力的评级高于平均水平。加入此类别风险是说明：即使目前通过数据没有体现，但是安全社区成员告诉我们，这也是一个很重要的风险。

SQL注入

- SQL注入指的是攻击者利用Web应用对用户输入数据过滤不够严格的弱点，构造特殊的字符串作为输入，欺骗数据库服务器执行未授权的恶意查询，最终导致数据信息泄露。
- SQL注入的攻击过程如下：



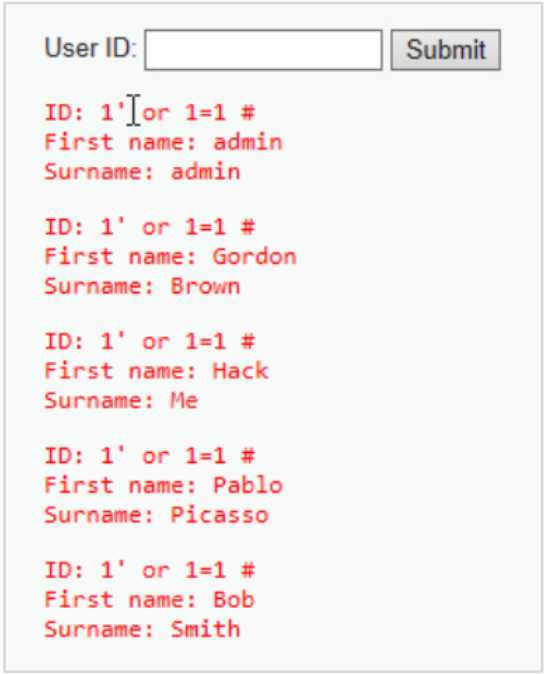
SQL注入

SQL注入获取Web应用管理员账号的示例如下：

- 攻击者在登录界面输入用户名1 ‘ or 1=1 #，在Web网站执行则会变成如下的SQL语句：

- 符号“#”会将随后的代码注释掉，where条件就变成title like ‘%1’ or 1=1，是一个永真条件，此时SQL语句返回所有的用户名。

```
select * from database.users where title like ‘%1’or 1=1 # %
```



SQL注入危害

01 数据库信息泄漏

02 网页篡改

03 数据库被恶意操作

04 服务器被远程控制

05 种植木马

SQL注入类型—根据注入点类型划分

数字型注入点

字符型注入点

搜索型注入点

SQL注入类型—根据数据提交方式划分

GET注入

提交数据的方式是 GET，注入点的位置在 GET 参数部分。

POST注入

使用 POST 方式提交数据，注入点位置在 POST 数据部分，常发生在表单中。

Cookie注入

HTTP 请求的时候会带上客户端的 Cookie, 注入点存在 Cookie 当中的某个字段。

HTTP头部注入

注入点在 HTTP 请求头部的某个字段中，比如存在 User-Agent 字段中。

SQL注入类型—按照执行效果划分

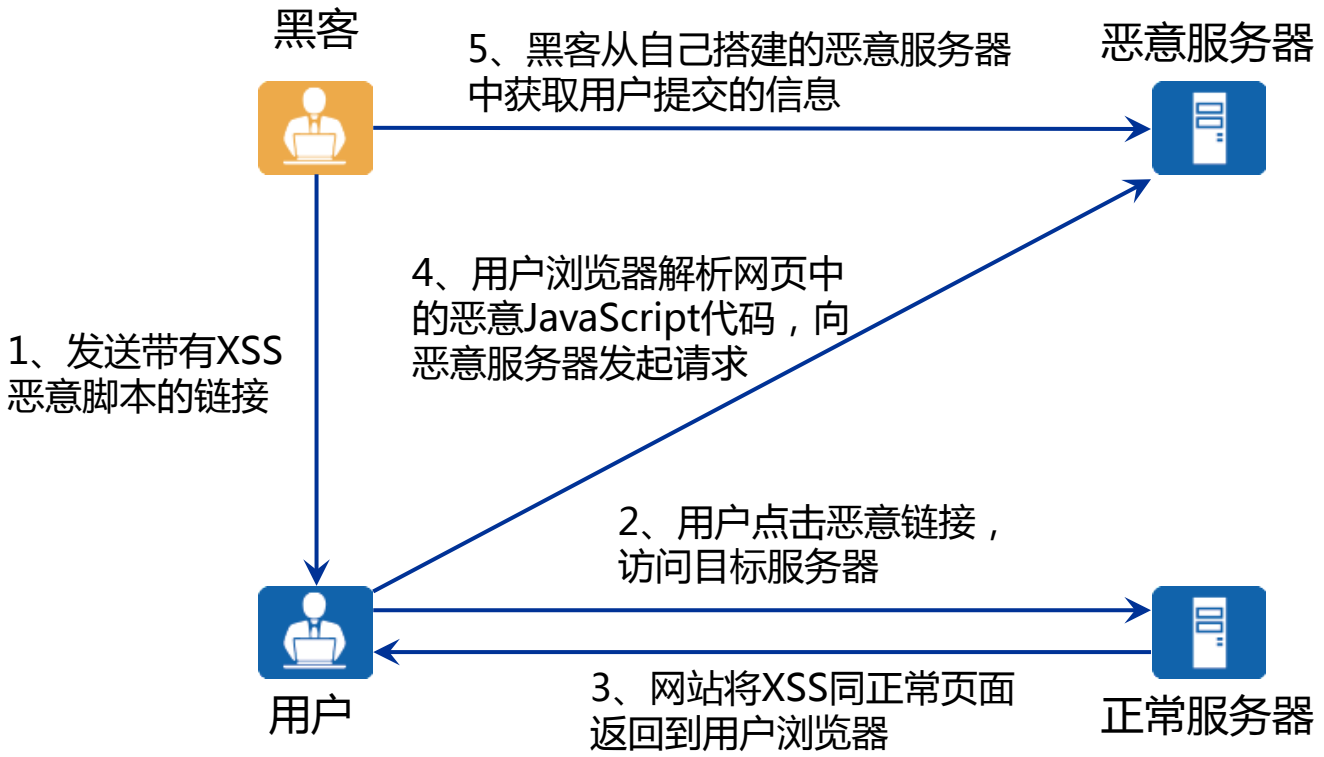
联合查询注入	可以使用union的情况下注入。
报错型注入	即页面会返回错误信息，或者把注入的语句结果直接返回在页面中。
基于时间的盲注	即不能根据页面返回内容判断任何信息，用条件语句查看时间延迟语句是否执行（即页面返回时间是否增加）进行判断。If() ,Sleep(5)
基于布尔的盲注	即可以根据返回页面判断条件真假的注入。
堆查询注入	可以同时执行多条SQL语句的注入。

反射型xss攻击流程

又称非持久型XSS，这种攻击方式往往具有一次性，只在用户单击时触发。跨站代码一般存在链接中，当受害者请求这样的链接时，跨站代码经过服务端反射回来，这类跨站的代码通常不存储服务端。

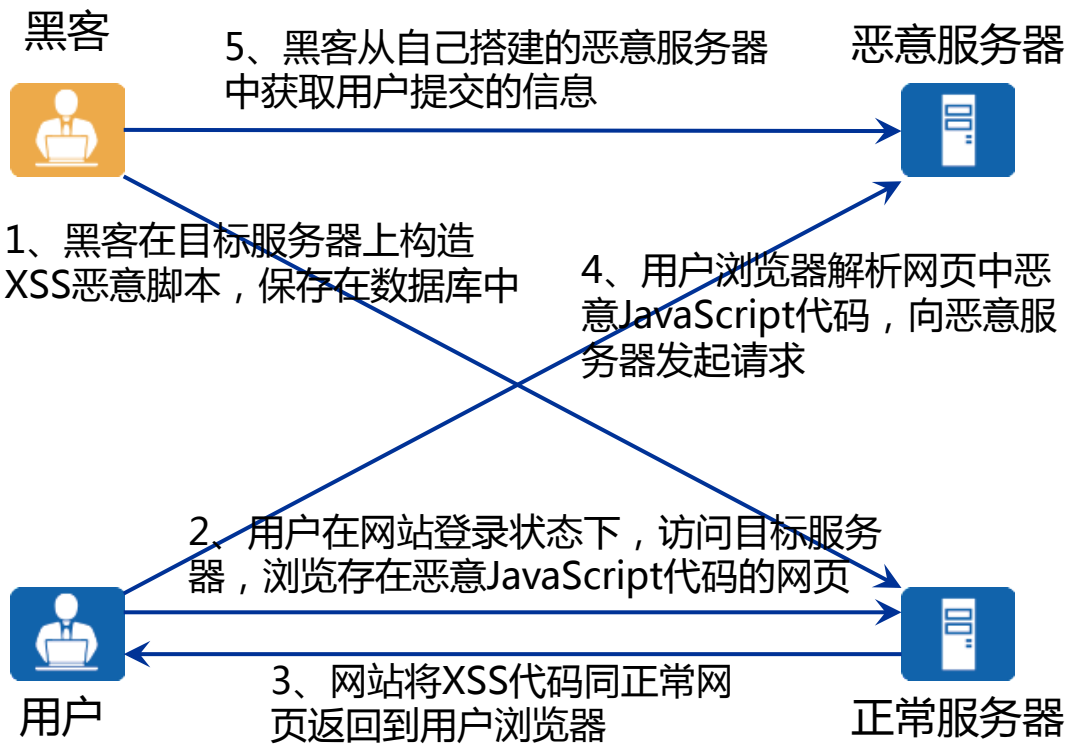
常见出现位置：

- 网站的搜索栏
- 用户登录入口
- 输入表单等地方



存储型XSS攻击原理

存储型XSS（ Stored xss Attacks ），也是持久型XSS，比反射型XSS更具有威胁性。攻击脚本将被永久的存放在目标服务器的数据库或文件中。这是利用起来最方便的跨站类型，跨站代码存储于服务端（比如数据库中）。



DOM型XSS

DOM是文档对象模型（ Document Object Model ）的缩写。它是HTML文档的对象表示，同时也是外部内容（例如 JavaScript ）与HTML元素之间的接口。解析树的根节点是 “ Document"对象。

DOM（Document object model），使用DOM能够使程序和脚本能够动态访问和更新文档的内容、结构和样式。它是基于DoM文档对象的一种漏洞，并且DOM型XSS是基于JS上的，并不需要与服务器进行交互。

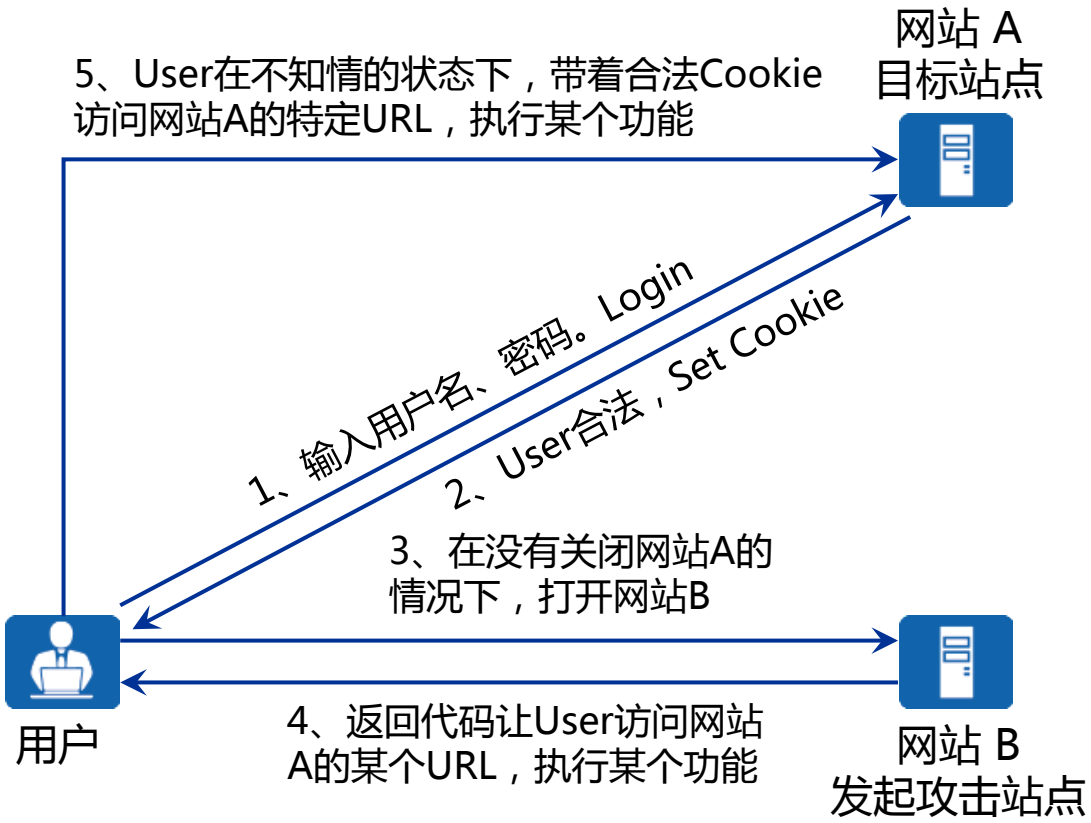
```
<script> function test(){
var str = document.getElementById("text").value;
            document.getElementById("t").innerHTML = "<a
href='"+str+"'>testLink</a>";
}
</script>
<div id="t"></div>
<input type="text" id="text" value="" />
<input type="button" id="s" value="write" onclick="test()" />
```

跨站请求伪造CSRF

CSRF(跨站请求伪造)，攻击者通过一些技术手段欺骗用户的浏览器去访问一个自己以前认证过的站点并运行一些操作（如发邮件，发消息，甚至财产操作（如转账和购买商品））。因为浏览器之前认证过，所以被访问的站点会认为这是真正的用户操作而去运行。

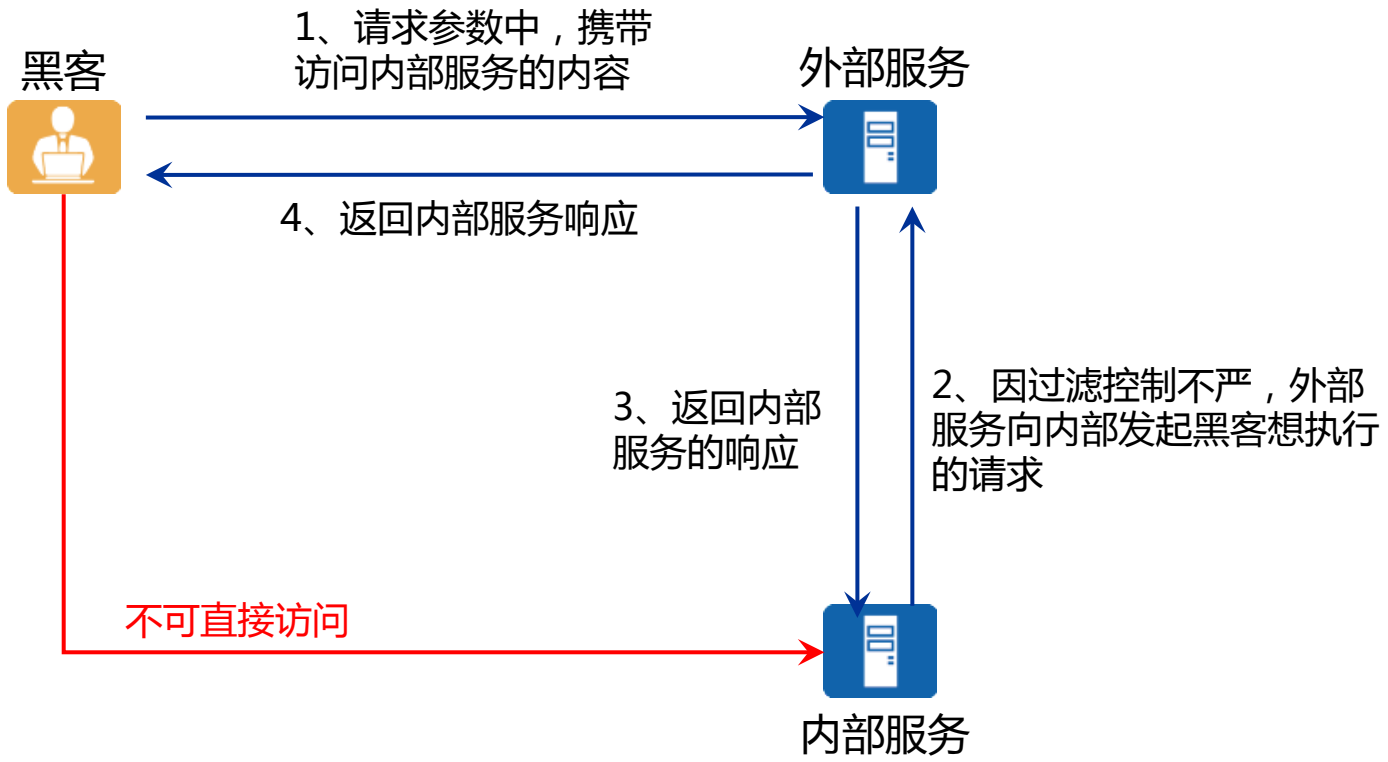
攻击条件：

- 用户已经登陆了网站，具有执行各个功能的权限。
- 攻击者知道功能的数据包。
- 用户访问了攻击者构造的恶意URL。



服务器端请求伪造SSRF

SSRF形成的原因大都是由于服务端提供了从其他服务器应用获取数据的功能,但又没有对目标地址做严格过滤与限制，导致攻击者可以传入任意的地址来让后端服务器对其发起请求，并返回对该目标地址请求的数据。

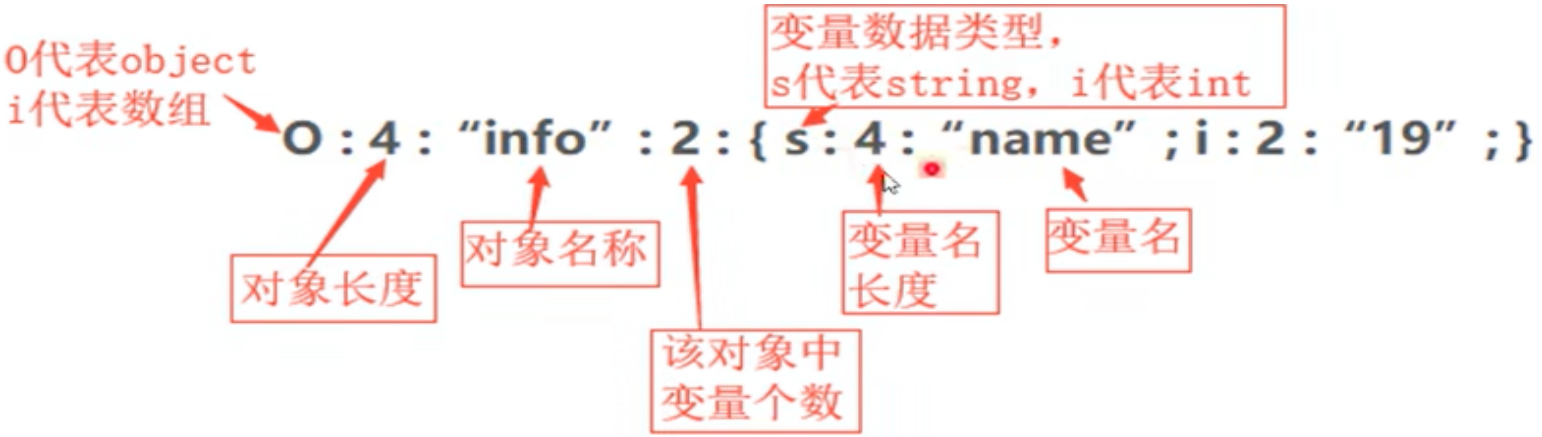


反序列化

反序列化的定义是，将对象的状态信息转换为可以存储或传输的形式的过程。在序列化期间，对象将其当前状态写入到临时或持久性存储区。以后，可以通过从存储区中读取或反序列化对象的状态，重新创建该对象。

简单的说，序列化就是把一个对象变成可以传输的字符串，可以以特定的格式在进程之间跨平台、安全的进行通信。


```php
<?php
$key='hello';
echo serialize($key);
?>
<?php
$key='s:5:"hello";';
echo unserialize($key);
?>
```



PHP反序列化

PHP反序列化漏洞也叫PHP对象注入，是一个非常常见的漏洞，这种类型的漏洞虽然有些难以利用，但一旦利用成功就会造成非常危险的后果。漏洞的形成的根本原因是程序没有对用户输入的反序列化字符串进行检测，导致反序列化过程可以被恶意控制，进而造成代码执行、getshell等一系列不可控的后果。反序列化漏洞并不是PHP特有，也存在于Java、Python等语言之中，但其原理基本相通。

```
serialize()          //将一个对象转换成一个字符串
unserialize()        //将字符串还原成一个对象
触发：unserialize 函数的变量可控，文件中存在可利用的类，类中有魔术方法：
__construct()        //创建对象时触发
__destruct()         //对象被销毁时触发
__call()             //在对象上下文中调用不可访问的方法时触发
__callStatic()       //在静态上下文中调用不可访问的方法时触发
__get()              //用于从不可访问的属性读取数据
__set()              //用于将数据写入不可访问的属性
__isset()            //在不可访问的属性上调用 isset()或 empty()触发
__unset()            //在不可访问的属性上使用 unset()时触发
__invoke()           //当脚本尝试将对象调用为函数时触发
```