常见漏洞介绍

> 大纲：
>
> 漏洞介绍
>
> 附录文档

漏洞介绍

> 常见漏洞修复方案：https://site.bytedance.net/docs/4820/9668/sast_fix/

SQL注入漏洞

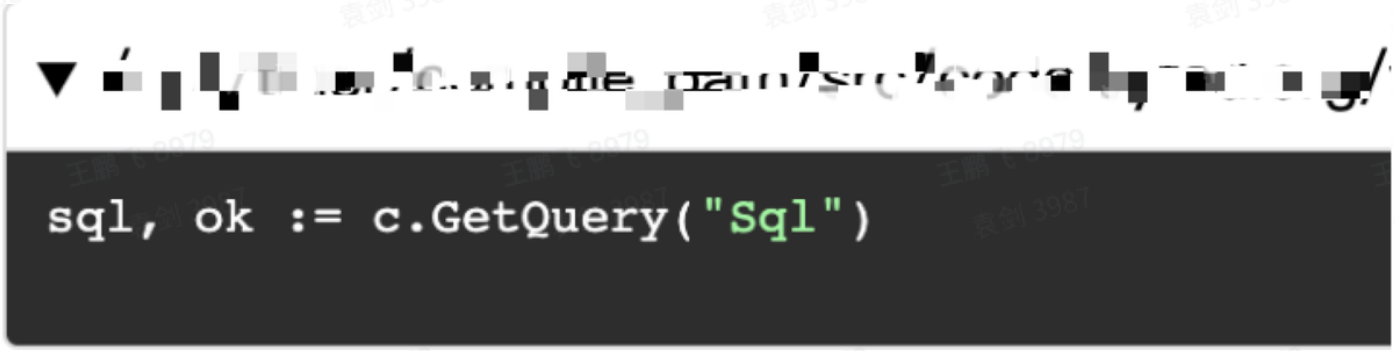
漏洞介绍

所谓 SQL 注入（SQL Injection），就是通过把 SQL 命令插入到 Web 表单提交或输入域名或页面请求的查询字符串，最终达到欺骗服务器执行恶意的 SQL 命令。

具体来说，它是利用现有应用程序，将（恶意）的 SQL 命令注入到后台数据库引擎执行的能力，它可以通过在 Web 表单中输入（恶意）SQL 语句得到一个存在安全漏洞的网站上的数据库，而不是按照设计者意图去执行 SQL 语句。

场景举例

1. 将用户的输入直接拼接到SQL语句中查询
    a. **攻击者可以构造恶意的SQL语句，对数据库进行增删改查操作**
2. 直接依赖用户传递SQL语句，进行查询



修复建议

1. Python
   a. 使用ORM框架执行SQL
   b. 对SQL语句进行参数绑定
   c. 在需要使用order by，group by时，对order by 、group by的参数进行白名单限制。

```
bad case:
execute("select * from user where username=%s"%username)

good case:
execute("select * from user where username=%s", username)
```

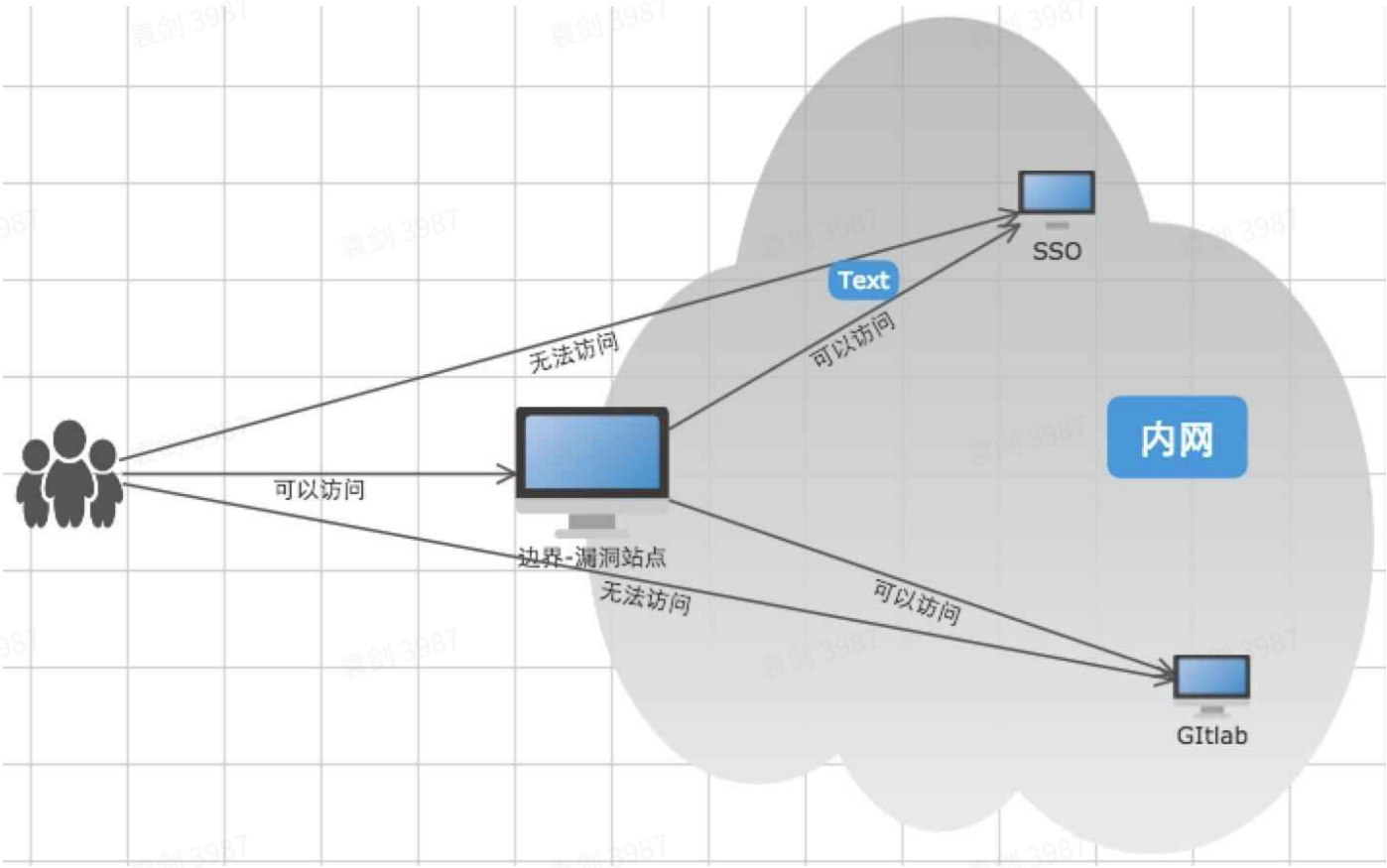
2. Go
   a. 使用ORM框架执行SQL
   b. 对SQL语句进行参数绑定
   c. 在需要使用order by，group by时，对order by 、group by的参数进行白名单限制。
3. 其他语言可参考上面的Go+Python

SSRF 服务器端请求伪造漏洞

漏洞介绍

SSRF（Server-side Request Forgery，服务器端请求伪造）是一种 **由攻击者构造形成 、 由服务端发起请求** 的一个安全漏洞。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统。

下图显示，漏洞站点可以由外部用户直接访问，也可以访问内网中的应用，处于网络边界；用户无法直接访问内网的SSO与Gitlab。如此，用户可以构造恶意请求通过漏洞站点请求SSO、GItlab与一些其他内网服务。



场景举例

- 从指定URL地址获取网页文本内容 (网站翻译)
- 对用户提供的远端图片添加水印 (在线编辑器、加水印功能)
- 下载用户提供的文件 (下载用户提供的图片保存到云端服务器)
- 判断网站是否存活，状态码等 (广告平台获取广告主网站信息)
- 获取网站的title等功能 (分享网站URL时，输出网站标题)

注意： 所有 **由攻击者构造形成 、 由服务端发起请求 的功能， 都可能存在这个问题。**

修复建议

1. **使用安全部门提供的** Polaris 安全模块使用指南
2. **如果确认平台是纯内部使用，不会有来自公网的数据，该漏洞无需修复**

CSRF 跨站请求伪造漏洞

漏洞介绍

CSRF（Cross-site Request Forgery）跨站请求伪造，由于目标站无 token/referer 限制，导致攻击者可以用户的身份完成操作达到各种目的。

场景举例

1. 点赞功能（一个POST接口请求）
   a. 攻击者恶意构造了一个页面，在页面中使用ajax构造了该POST请求（自动提交）
   b. 攻击者通过私聊发消息告知用户该页面链接
   c. 用户访问该链接，页面中的ajax自动发起点赞请求
   d. 由于是在用户本地浏览器发起的请求，浏览器默认携带用户的Cookie信息
   e. 结果：用户访问了一个恶意页面，自动发起了对默认文章/视频 点赞操作（用户完全无感知）
2. 用户修改密码功能（参考上方利用流程）
3. 用户删除视频功能 （参考上方利用流程）

修复建议

1. 开启框架的CSRF防御机制（例如：Django的*Csrf*ViewMiddleware中间件）
2. Token机制
   a. 服务端生成随机token
      i. 将token下发到前端页面中
      ii. 在cookie中存储一份一致的token
   b. 每次接口请求，都校验参数中的token和cookie中的token是否一致
3. Referer验证
   a. 对Referer来源开始白名单机制
      i. 白名单中不能存在referer为空的情况
      ii. 白名单需要匹配完整主域名，例如toutiao.com，需要加上点号的判断（.toutiao.com），避免ssstoutiao.com进行绕过

> 图中将csrftoken加在了header字段中，实际操作中，可放置于post data中

XSS 跨站脚本攻击漏洞

漏洞介绍

跨站脚本（Cross-site scripting，通常简称为：XSS）是一种网站应用程序的安全漏洞攻击，是代码注入的一种。

它允许恶意用户将代码注入到网页上，其他用户在访问网页时就会受到影响。这类攻击通常包含了 HTML 以及用户端脚本语言。

XSS 攻击通常指的是通过利用网页开发时留下的漏洞，通过巧妙的方法注入恶意指令代码到网页，使用户加载并执行攻击者恶意制造的网页程序。

这些恶意网页程序通常是 JavaScript，但实际上也可以包括Java，VBScript，ActiveX，Flash 或者甚至是普通的 HTML。

攻击成功后，攻击者可能得到更高的权限（如执行一些操作）、私密网页内容、会话和 Cookie 等各种内容。

场景举例

1. 前端将用户输入的内容，直接展示在页面标签中
    a. 如果用户输入内容为："/></script><img src=1 onerror=alert(1)><name aa="
    b. 前端渲染用户输入内容时，就会将恶意标签进行渲染
    c. 导致可以执行任意JS（**包括但不限于：构造恶意链接，盗取用户cookie，伪造用户身份去发起删除请求等等**）
2. 后端接口将用户的数据，以json的形式进行返回
    a. 由于未设置返回页面的conten-type，默认为text/html
    b. 用户数据中存在恶意标签内容，导致浏览器渲染执行了恶意标签
3. 前端将接口中的json数据直接赋值到script标签中的变量
    a. 该变量内容未恶意标签"/></script><img src=1 onerror=alert(1)><name aa="
    b. 浏览器优先闭合script标签，导致恶意标签内容被渲染执行
4. 服务端接口用户的富文本内容（富文本内容本身是包括标签的）
    a. 服务端利用正则表达式匹配恶意内容
    b. 攻击者大量尝试，绕过正则规则，构造恶意payload

修复建议

1. 非富文本场景
    a. 进行htmlencode编码操作
        i. 已经上线的业务，做编码操作，需要保证前端后端同学信息同步
2. 富文本场景

a. **使用安全部门提供的Polaris安全工具包进行dom解析，通过白名单标签过滤**（具体见文末附录）

越权操作漏洞

漏洞介绍

常见的越权分为两种：水平越权、垂直越权。一般来说系统会分为三类用户：特权用户、普通用户、非登陆用户。

水平越权，指普通用户A可以查看普通用户B的信息。例如，我发现某购物网站一个订单系统的水平越权漏洞，就可以越权查看别人的订单。

垂直越权，指普通用户可以使用特权用户的功能。

场景举例

1. 通过订单ID获取订单内容
   a. 通过遍历订单ID，越权获取其他人的订单信息
2. 删除用户个人资源
   a. 通过变更个人资源的ID号，越权删除其他用户的资源
3. 添加用户管理员
   a. 该接口对使用用户进行了认证，未进行鉴权
   b. 普通用户可以添加任意管理员

修复建议

1. 对操作用户所拥有的权限和要操作的资源做关联鉴权

爆破撞库

漏洞介绍

爆破撞库基本上都出现在登陆口，由于没有做限制导致可以批量尝试用户密码是否正确。

攻击者常用的手法是，准备大量的之前已经流传出去的账号密码数据库，对登陆口等位置进行大量尝试。

场景举例

1. 登陆场景， 输入用户名之后， 会进行提示用户名是否正确
2. 重置密码场景， 用户输入用户名后，系统提示你要找回的用户名是否存在。

如果没有限制，可以进行批量跑的话，就会存在扫号的问题。

修复建议

1. 增加图形验证码
2. 模糊提示性文案（例：用户名或密码不正确）
3. 对这类接口进行频率控制

代码执行和命令注入漏洞

漏洞介绍

远程代码执行（Remote Code Execution，简称 RCE）指攻击者可以通过程序漏洞在服务器上远程执行恶意代码。

而命令注入（Command Injection）指程序代码把用户提交的数据直接拼接到系统命令执行的功能内，造成用户可执行非法命令。

命令注入是针对系统命令，而远程代码执行是程序代码缺陷，命令注入一般认为是远程代码执行的一种。

场景举例

1. 将用户post的json数据做格式转化
   a. eval(request.body)
      i. 攻击者可以输入任意恶意代码，服务端都会执行
2. 根据用户输入执行系统漏洞

```
64    @app.route(    ■■■    ■', methods=['get'])
65    def execute_cmd():
66        shell = request.args.get('shell')
67        os.system(shell)
```

app.py

```
new_name = request.args.get('new_name')
pack_name = request.args.get('pack_name')
version = request.args.get('version')
AppId = getAppId(request)
form = {'case_name': case_name, 'pack_name': pack_name, 'version': version, 'AppId': AppId}
```

app.py

```
os.system('sudo cp -rf {} {}'.format(base_url + 'img/' + pack_name + '/' + version + '/' + case_name,
                                     base_url + 'img/' + pack_name + '/' + version + '/' + new_name))
```

修复建议

1. **使用安全部门提供的Polaris安全工具包进行安全的eval执行**（具体见文末附录）
2. 命令执行操作推荐
   a. Python：

i. `s=subprocess.Popen(["ls",";id"], shell=False, stderr=subprocess.PIPE, stdout=subprocess.PIPE)`

ii. 这样即使;id可控，也不能任意命令执行（前提：要执行的命令不能为bash/sh之类）

iii. 过滤特殊字符：`| || & && { } ( ) ; `` $ ' " % * < >`

iv. 使用官方推荐过滤函数：`shlex.quote`

b. Go：

i. 利用下面的代码调用第三方程序时，应确保prog并非从用户输入获取。如非必要，禁止将prog设置为`/bin/bash`、`/bin/sh`等shell程序或`python`。

ii. `exec.Command(prog, args...)`

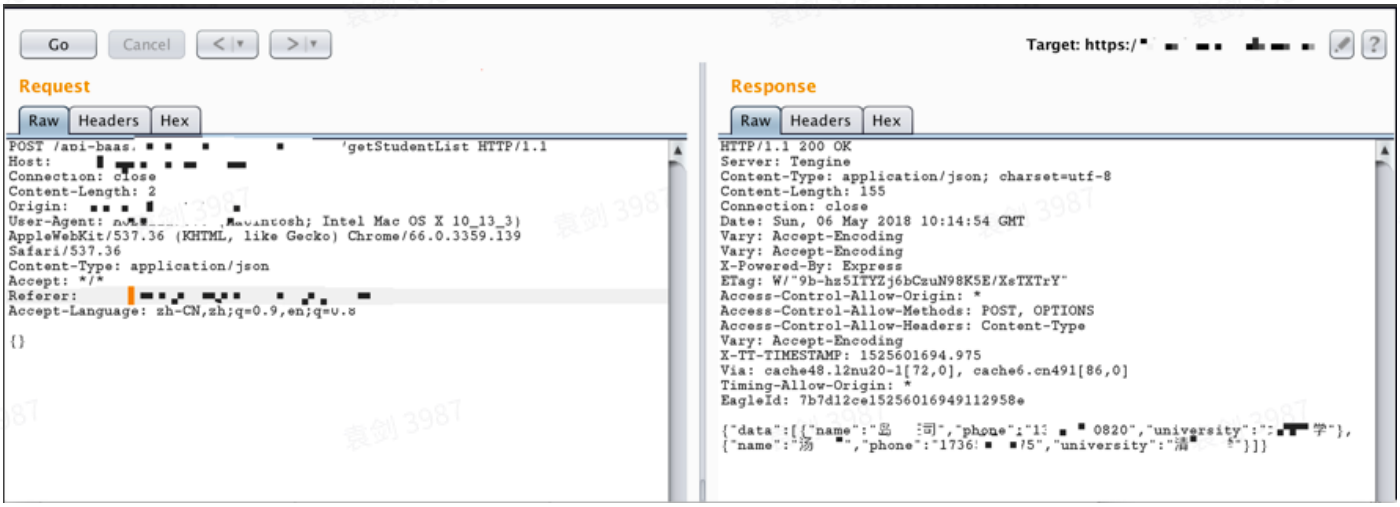
信息泄露

漏洞介绍

信息泄露主要是由于一些配置缺陷或是接口功能未屏蔽敏感信息所导致，可能泄露网站源码、数据库备份、网站绝对路径、用户敏感信息、sql语句、应用配置信息、敏感目录等等。

场景举例

1. 线上服务debug模式未关闭
2. 将网站源码打包上传到cdn域名下
3. 接口返回信息未脱敏



> 接口返回用户的真实姓名、手机号、大学等信息，无需登陆

修复建议

1. 关闭debug模式，防止调试信息泄露。或是当web应用出错时，统一返回一个错误页面或直接跳转至首页
2. 网站目录下禁止上传备份文件或是phpinfo页面
3. 针对接口数据，应结合业务场景判断是否可以直接返回用户的完整真实信息或是其他敏感信息
4. 删除在公开网络共享的敏感信息，包括身份认证信息、项目功能代码、内部资产等信息

任意文件上传/下载

漏洞介绍

文件上传和下载漏洞，基本上只出现在对应的文件上传下载功能上。由于没有做限制导致可以上传下载任意文件。

场景举例

1. 获取用户输入的文件名称，下载该文件
   a. **通过构造`../../../../../etc/passwd`，可以下载服务器任意文件文件**
2. 可以上传任意格式的文件
   a. 上传`.exe`文件，运营同学后台下载该文件点击时，实际执行了病毒木马
   b. 上传`.jsp/.jspx/.php/.php3/.asp/.aspx`文件，如果该文件类型可以被服务端解析，就能获取到服务器的webshell（可以执行任意命令）

修复建议

1. 文件上传漏洞：
   a. 大小做限制
   b. 后缀白名单限制(只允许：png、jpg等图片格式)
   c. 路径不可更改
   d. 保存文件名随机生成
   e. 上传文件保存的目录不解析可执行文件
   f. 上传文件到tos的文件需携带content-type上传，比如python的storage/pytos仓库的设置方式
2. 文件下载漏洞：
   a. 如非必要，不要使用动态下载功能
      i. http://download.xx.com/down?file=11111.doc
      ii. http://download.xx.com/down?file=../../../../../../root/.bash_history
   b. 直接提供静态URL。
      i. http://download.xx.com/down/1111.doc

JsonP劫持漏洞

漏洞介绍

jsonp是一个非官方的协议，利用script元素的开放策略，网页可以得到从其他来源动态产生的json数据，因此可以用来实现跨域。

场景举例

1. 通过jsonp获取其他域下返回的用户敏感信息
    a. 攻击者恶意构造了一个页面，在页面中使用ajax构造了该请求（自动提交）
    b. 攻击者通过私聊发消息告知用户该页面链接
    c. 用户访问该链接，页面中的ajax自动发起请求
    d. 由于是在用户本地浏览器发起的请求，浏览器默认携带用户的Cookie信息
    e. 结果：用户访问了一个恶意页面，该恶意页面将获取到的用户敏感信息加载在js中，发送到攻击者手中。（类似csrf的利用过程）

修复建议

1. 使用CORS进行跨域资源访问
    a. 针对主域名白名单时，需要考虑点号，例如`.toutiao.com`，避免ssstoutiao.com进行绕过

附录文档

Polaris 安全模块使用指南

常见漏洞测试手段

安全编码规范