Python代码审计连载之一：CSRF

笑然 / 2016-11-10 05:02:02 / 浏览数 4354 安全技术 漏洞分析 顶(0) 踩(0)

原文地址：https://www.cdxy.me/?p=736

防御机制

目前主流CSRF的防御思路可总结为：在用户携带的信息(Cookie)之外置入token并在服务端检验，该token要满足一次性、随机性

主流Python后端框架(Flask/Django/Tornado)使用Session/Cookie-Form的验证机制避免CSRF。

Flask官方文档给出的解决方案(Session-Form)：

服务端生成_csrf_token并置入用户Session和Form。用户提交Form之后，服务端对比Form中的_csrf_token字段和Session中是否相同，相同则通过，否则返回403。

```
@app.before_request
def csrf_protect():
   if request.method == "POST":
       token = session.pop('_csrf_token', None)
       if not token or token != request.form.get('_csrf_token'):
           abort(403)

def generate_csrf_token():
   if '_csrf_token' not in session:
       session['_csrf_token'] = some_random_string()
   return session['_csrf_token']

app.jinja_env.globals['csrf_token'] = generate_csrf_token
```

攻击面

我想到的点：

1. 策略没覆盖到. (GET, json)
2. 修改(覆盖)Cookie中存储的token. (XSS, CRLF)
3. 算法角度预测到用户token.
4. 让策略失效.(关闭或打断CSRF检验机制的运行)

修改用户Cookie造成CSRF

1 鸡肋XSS达不到httponly的Cookie时，可以试试做CSRF:

知乎某处XSS+刷粉超详细漏洞技术分析

2 利用Python-cookie解析漏洞篡改token:

(CVE-2016-7401) CSRF protection bypass on any Django powered site via Google Analytics

当添加的Cookie中存在]符号时，解析器会以该符号分隔带入新的字段。

```
>>> from http import cookies
>>> C = cookies.SimpleCookie()
>>> C.load('__utmz=blah]csrftoken=x')
>>> C
<SimpleCookie: csrftoken='x'>
```

这样我们就可以在添加Cookie时，伪造出一个`csrftoken`字段的值，并与Form中设为相同，这样后端对比通过，做成CSRF.

3 请求头注入，如`CRLF-injection`、`overflow`等。

(Twitter demo)HTTP Response Splitting with Header Overflow

策略之外

使用GET方法时的原则是——让GET只负责"读取"资源的操作

实际开发中问题往往出在一些简单按钮的功能，如"删除"、"回收"、"置顶"、"清除缓存"等功能，很容易通过GET完成。这直接导致了CSRF的隐患。

使用其他的HTTP方法同样存在该问题，一般来说对GET/HEAD/OPTIONS/TRACE无需检验，具体请参考各框架及插件文档。

此外，ajax-json也是也是容易被忽略的点。

针对这个点Flask和Django都给出了解决方案，或者放在请求头中也可：

Flask-WTF:

> If you need to send the token via AJAX, and there is no form:
>
> ```
> <meta name="csrf_token" content="{{ csrf_token() }}" />
> ```
>
> You can grab the csrf token with JavaScript, and send the token together.

Django Document

> While the above method can be used for AJAX POST requests, it has some inconveniences: you have to remember to pass the CSRF token in as POST data with every POST request. For this reason, there is an alternative method: on each XMLHttpRequest, set a custom X-CSRFToken header to the value of the CSRF token. This is often easier, because many JavaScript frameworks provide hooks that allow headers to be set on every request.

开发者的疏忽

开发者并未开启全局CSRF防御策略，而是进行单个view的控制(如@csrf_protect)，这样未免会有遗漏的情况。

是否可预测、可绕过？

Flask-WTF处理CSRF的关键逻辑在python2.7/site-packages/flask_wtf/csrf.py，这个文件相当于Session-Form防御思路的完整实现。

这里给出Flask-WTF token生成函数源码：

```
def generate_csrf(secret_key=None, time_limit=None, token_key='csrf_token', url_safe=False):
    """Generate csrf token code.

    :param secret_key: A secret key for mixing in the token,
                       default is Flask.secret_key.
    :param time_limit: Token valid in the time limit,
                       default is 3600s.
    """
    if not secret_key:
        secret_key = current_app.config.get(
            'WTF_CSRF_SECRET_KEY', current_app.secret_key
        )

    if not secret_key:
        raise Exception('Must provide secret_key to use csrf.')

    if time_limit is None:
        time_limit = current_app.config.get('WTF_CSRF_TIME_LIMIT', 3600)

    if token_key not in session:
        session[token_key] = hashlib.sha1(os.urandom(64)).hexdigest()

    if time_limit:
        expires = int(time.time() + time_limit)
        csrf_build = '%s%s' % (session[token_key], expires)
    else:
        expires = ''
        csrf_build = session[token_key]

    hmac_csrf = hmac.new(
        to_bytes(secret_key),
        to_bytes(csrf_build),
        digestmod=hashlib.sha1
    ).hexdigest()
    delimiter = '--' if url_safe else '##'
    return '%s%s%s' % (expires, delimiter, hmac_csrf)
```

采用了HMAC算法，hash算法是SHA1，本人水平有限，无法评估其缺陷。

至于这个算法的优点摘抄如下：

What does this "_time_independent_equals" mean?

> That function does not simply compare the strings, it tries to always take the same amount of time to execute.
>
> This is useful for security tasks like comparing passwords. If the function returned on the first mismatching byte, an attacker could try all possible first bytes and know that the one that takes longest is a match. Then they could try all possible second bytes and know that the one that takes longest is a match. This can be repeated until the entire string is deduced. (In reality you have to do a lot of averaging to overcome random delays in the network, but it works if you are patient.)

上一篇：GitLab 任意文件读取漏洞 (... 下一篇：Python代码审计连载之二：SSTI

1. 0 条回复
   - 动动手指，沙发就是你的了！