CSRF（Cross-Site Request Forgery，跨站点伪造请求）是一种网络攻击方式，该攻击可以在受害者毫不知情的情况下以受害者名义伪造请求发送给受攻击站点，从而在未授权的情况下执行在权限保护之下的操作，具有很大的危害性。具体来讲，可以这样理解CSRF攻击：攻击者盗用了你的身份，以你的名义发送恶意请求，对服务器来说这个请求是完全合法的，但是却完成了攻击者所期望的一个操作，比如以你的名义发送邮件、发消息，盗取你的账号，添加系统管理员，甚至于购买商品、虚拟货币转账等。

1、CSRF原理

案例：FileUploadAndDownLoad　lezijie-note

CSRF攻击原理比较简单，如图1所示。其中Web A为存在CSRF漏洞的网站，Web B为攻击者构建的恶意网站，User C为Web A网站的合法用户。

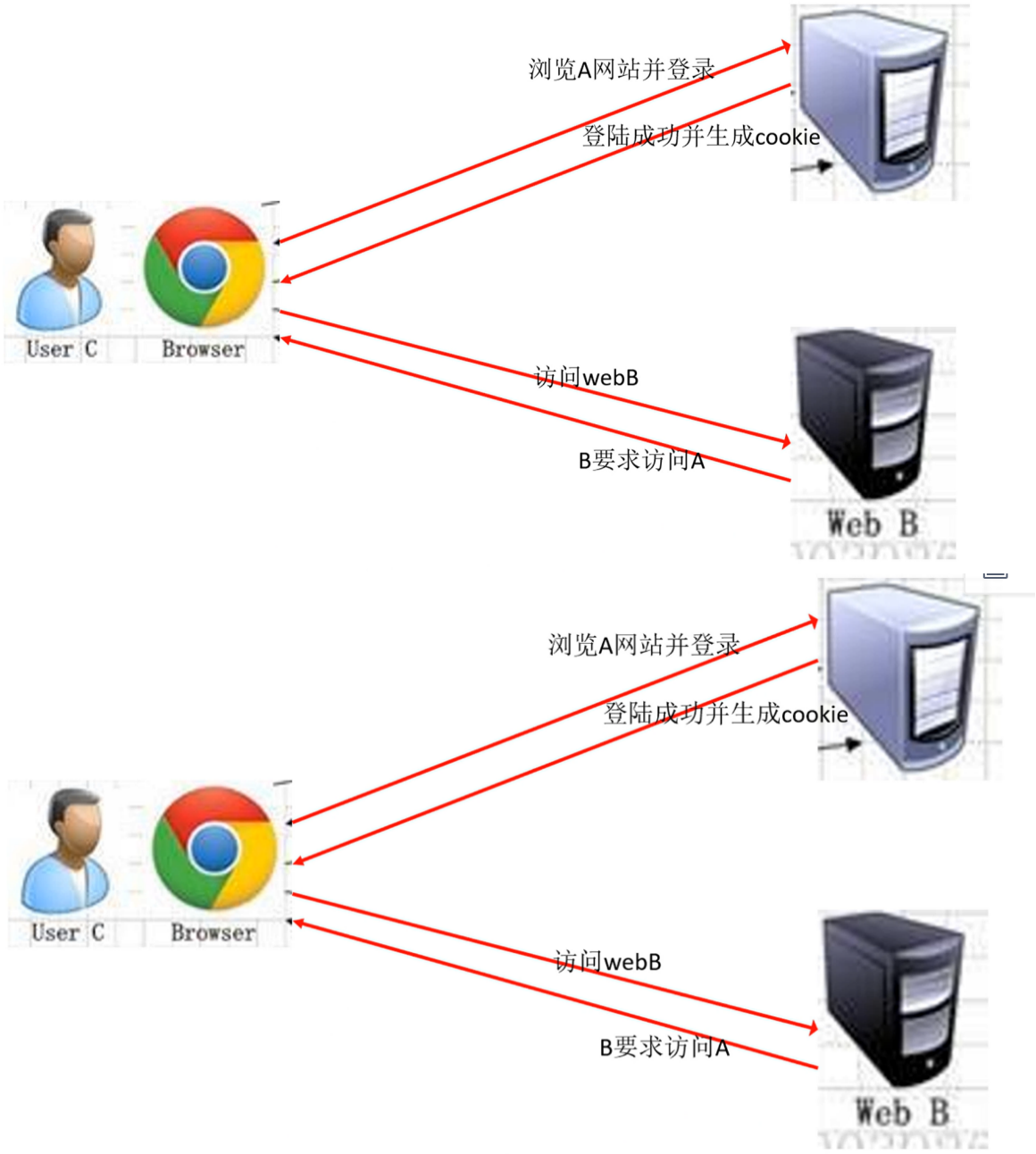
1. 用户C打开浏览器，访问受信任网站A，输入用户名和密码请求登录网站A；

2.在用户信息通过验证后，网站A产生Cookie信息并返回给浏览器，此时用户登录网站A成功，可以正常发送请求到网站A；

3. 用户未退出网站A之前，在同一浏览器中，打开一个TAB页访问网站B；
4. 网站B接收到用户请求后，返回一些攻击性代码，并发出一个请求要求访问第三方站点A；
5. 浏览器在接收到这些攻击性代码后，根据网站B的请求，在用户不知情的情况下携带Cookie信息，向网站A发出请求。网站A并不知道该请求其实是由B发起的，所以会根据用户C的Cookie信息以C的权限处理该请求，导致来自网站B的恶意代码被执行。

例如：你再使用网银给A转账，这个时候QQ发过来一个链接，你点了这个链接，访问了网站B，B给浏览器发送访问A的请求，由于浏览器是公用的，并且浏览器保存了你的用户信息，这个链接使用了你的身份信息给银行发送了一条请求，给B进行了转账。银行服务器，比较了用户信息，发现是本人操作。进行转账。

7 CSRF漏洞简介



将长链接转换成短链接，短url生成

1 验证HTTP Referer字段

2 在请求地址中添加token并验证

3 在HTTP头中自定义属性并验证

4 用户端的防御

CSRF漏洞防御

1、验证HTTP Referer字段

2、在请求地址中添加token并验证

3、在HTTP头中自定义属性并验证

4、用户端的防御

CSRF漏洞防御主要可以从三个层面进行，即服务端的防御、用户端的防御和安全设备的防御。

2.1　服务端的防御

2.1.1 验证HTTP Referer字段

根据HTTP协议，在HTTP头中有一个字段叫Referer，它记录了该HTTP请求的来源地址。在通常情况下，访问一个安全受限页面的请求必须来自于同一个网站。比如某银行的转账是通过用户访问http://bank.test/test?page=10&userID=101&money=10000页面完成，用户必须先登录bank.test，然后通过点击页面上的按钮来触发转账事件。当用户提交请求时，该转账请求的Referer值就会是转账按钮所在页面的URL（本例中，通常是以bank. test域名开头的地址）。而如果攻击者要对银行网站实施CSRF攻击，他只能在自己的网站构造请求，当用户通过攻击者的网站发送请求到银行时，该请求的Referer是指向攻击者的网站。因此，要防御CSRF攻击，银行网站只需要对于每一个转账请求验证其Referer值，如果是以bank. test开头的域名，则说明该请求是来自银行网站自己的请求，是合法的。如果Referer是其他网站的话，就有可能是CSRF攻击，则拒绝该请求。

2.1.2 在请求地址中添加token并验证

CSRF攻击之所以能够成功，是因为攻击者可以伪造用户的请求，该请求中所有的用户验证信息都存在于Cookie中，因此攻击者可以在不知道这些验证信息的情况下直接利用用户自己的Cookie来通过安全验证。由此可知，抵御CSRF攻击的关键在于：在请求中放入攻击者所不能伪造的信息，并且该信息不存在于Cookie之中。鉴于此，系统开发者可以在HTTP请求中以参数的形式加入一个随机产生的token，并在服务器端建立一个拦截器来验证这个token，如果请求中没有token或者token内容不正确，则认为可能是CSRF攻击而拒绝该请求。

2.1.3 在HTTP头中自定义属性并验证

自定义属性的方法也是使用token并进行验证，和前一种方法不同的是，这里并不是把token以参数的形式置于HTTP请求之中，而是把它放到HTTP头中自定义的属性里。通过XMLHttpRequest这个类，可以一次性给所有该类请求加上csrftoken这个HTTP头属性，并把token值放入其中。这样解决了前一种方法在请求中加入token的不便，同时，通过这个类请求的地址不会被记录到浏览器的地址栏，也不用担心token会通过Referer泄露到其他网站。

3 其他防御方法

1. CSRF攻击是有条件的，当用户访问恶意链接时，认证的cookie仍然有效，所以当用户关闭页面时要及时清除认证cookie，对支持TAB模式（新标签打开网页）的浏览器尤为重要。
2. 尽量少用或不要用request()类变量，获取参数指定request.form()还是request. querystring ()，这样有利于阻止CSRF漏洞攻击，此方法只不能完全防御CSRF攻击，只是一定程度上增加了攻击的难度。

Java 代码示例

下文将以 Java 为例，对上述三种方法分别用代码进行示例。无论使用何种方法，在服务器端的拦截器必不可少，它将负责检查到来的请求是否符合要求，然后视结果而决定是否继续请求或者丢弃。在 Java 中，拦截器是由 Filter 来实现的。我们可以编写一个 Filter，并在 web.xml 中对其进行配置，使其对于访问所有需要 CSRF 保护的资源的请求进行拦截。

在 filter 中对请求的 Referer 验证代码如下

清单 1. 在 Filter 中验证 Referer

```
public class RefererFilter implements Filter {
    private String excludedPage;
    private String[] excludedPages;
    /**
     * Default constructor.
     */
    public RefererFilter() {
        // TODO Auto-generated constructor stub
    }

    /**
     * @see Filter#destroy()
     */
    public void destroy() {
        // TODO Auto-generated method stub
    }

    /**
     * @see Filter#doFilter(ServletRequest, ServletResponse, FilterChain)
     */
    public void doFilter(ServletRequest request, ServletResponse response,
FilterChain chain) throws IOException, ServletException {
        // 从 HTTP 头中取得 Referer 值
        HttpServletRequest hrequest=(HttpServletRequest) request;
        // 定义表示变量 并验证用户请求URL 是否包含不过滤路径
        boolean flag = false;
        for (String page:excludedPages) {
            if (hrequest.getRequestURI().equals(page)){
                flag = true;
            }
        }
        if(flag){
            chain.doFilter(request,response);
```

```
            return;
        }

        String referer=hrequest.getHeader("Referer");
        // 判断 Referer 是否以 bank.example 开头
        if((referer!=null) &&
(referer.trim().startsWith("http://localhost:8081"))){
            chain.doFilter(request, response);
        }else{
        request.getRequestDispatcher("error.jsp").forward(request,response);

        }
    //  chain.doFilter(request, response);
    }

    /**
     * @see Filter#init(FilterConfig)
     */
    public void init(FilterConfig fConfig) throws ServletException {
         excludedPage = fConfig.getInitParameter("ignores");//此处的ignores就是在
web.xml定义的名称一样。
            if (excludedPage != null && excludedPage.length() > 0){
                excludedPages = excludedPage.split(",");
            }

    }

}
```

web.xml

```
<filter>
    <filter-name>RefererFilter</filter-name>
    <display-name>RefererFilter</display-name>
    <description></description>
    <filter-class>com.lezijie.note.filter.RefererFilter</filter-class>
          <init-param>
            <param-name>ignores</param-name>
            <param-value>/note/login.jsp,/note/</param-value>
        </init-param>
  </filter>
  <filter-mapping>
    <filter-name>RefererFilter</filter-name>
    <url-pattern>/*</url-pattern>
  </filter-mapping>
```

以上代码先取得 Referer 值，然后进行判断，当其非空并以 bank.example 开头时，则继续请求，否则的话可能是 CSRF 攻击，转到 error.jsp 页面。

如果要进一步验证请求中的 token 值，代码如下

2. 在 filter 中验证请求中的 token

Syntax error in graph
mermaid version 8.8.3

首先判断 session 中有没有 csrftoken，如果没有，则认为是第一次访问，session 是新建立的，这时生成一个新的 token，放于 session 之中，并继续执行请求。如果 session 中已经有 csrftoken，则说明用户已经与服务器之间建立了一个活跃的 session，这时要看这个请求中有没有同时附带这个 token，由于请求可能来自于常规的访问或是 XMLHttpRequest 异步访问，我们分别尝试从请求中获取 csrftoken 参数以及从 HTTP 头中获取 csrftoken 自定义属性并与 session 中的值进行比较，只要有一个地方带有有效 token，就判定请求合法，可以继续执行，否则就转到错误页面。生成 token 有很多种方法，任何的随机算法都可以使用，Java 的 UUID 类也是一个不错的选择。

除了在服务器端利用 filter 来验证 token 的值以外，我们还需要在客户端给每个请求附加上这个 token，这是利用 js 来给 html 中的链接和表单请求地址附加 csrftoken 代码，其中已定义 token 为全局变量，其值可以从 session 中得到。

清单

3. 在客户端对于请求附加 token

```
function appendToken(){

updateForms();

updateTags();

}

function updateForms() {

// 得到页面中所有的 form 元素

var forms = document.getElementsByTagName('form');

for(i=0; i<forms.length; i++) {

    var url = forms[i].action;

    // 如果这个 form 的 action 值为空，则不附加 csrftoken

    if(url == null || url == "" ) continue;

    // 动态生成 input 元素，加入到 form 之后

    var e = document.createElement("input");

    e.name = "csrftoken";

    e.value = token;

    e.type="hidden";

    forms[i].appendChild(e);
```

```
}

}

function updateTags() {

var all = document.getElementsByTagName('a');

var len = all.length;

// 遍历所有 a 元素

for(var i=0; i<len; i++) {

    var e = all[i];

    updateTag(e, 'href', token);

}

}

function updateTag(element, attr, token) {

var location = element.getAttribute(attr);

if(location != null && location != '' '' ) {

    var fragmentIndex = location.indexOf('#');

    var fragment = null;

    if(fragmentIndex != -1){

        //url 中含有只相当页的锚标记

        fragment = location.substring(fragmentIndex);

        location = location.substring(0,fragmentIndex);

    }


    var index = location.indexOf('?');

    if(index != -1) {

        //url 中已含有其他参数

        location = location + '&csrftoken=' + token;

    } else {

        //url 中没有其他参数

        location = location + '?csrftoken=' + token;
```

```
    }


    if(fragment != null){
        location += fragment;
    }
    element.setAttribute(attr, location);
}
}
```

在客户端 html 中，主要是有两个地方需要加上 token，一个是表单 form，另一个就是链接 a。这段代码首先遍历所有的 form，在 form 最后添加一隐藏字段，把 csrftoken 放入其中。然后，代码遍历所有的链接标记 a，在其 href 属性中加入 csrftoken 参数。注意对于 a.href 来说，可能该属性已经有参数，或者有锚标记。因此需要分情况讨论，以不同的格式把 csrftoken 加入其中。