link: null
title: 珠峰架构师成长计划
description: 诱导用户点击恶意链接来造成一次性攻击
keywords: null
author: null
date: null
publisher: 珠峰架构师成长计划
stats: paragraph=97 sentences=235, words=1815

1. XSS(Cross-Site Script)

诱导用户点击恶意链接来造成一次性攻击

- 反射型XSS攻击是一次性的，必须要通过用户点击链接才能发起
- 一些浏览器如Chrome其内置了一些XSS过滤器，可以防止大部分反射型XSS攻击
- 反射型XSS其实就是服务器没有对恶意的用户输入进行安全处理就直接反射响应内容，导致恶意代码在浏览器中执行的一种XSS漏洞

```
const express = require('express');
const fs = require('fs');
const path = require('path');
const app = express();
const bodyParser = require('body-parser');
app.use(bodyParser.urlencoded({ extended: true }));
app.use(bodyParser.json());
app.use(express.static(path.resolve(__dirname, 'public')));

app.get('/list', function (req, res) {
    let { category } = req.query;
    res.header('Content-Type', 'text/html;charset=utf-8');
    res.send(`你输入的分类是: ${category}`);
});
app.listen(3000, () => console.log('The server is starting at port 3000'));
```

黑客将代码存储到漏洞服务器中，用户浏览相关页面发起攻击

类型 反射型 存储型 持久性 非持久 持久化(存储在服务器) 触发时机 需要用户点击 不需要用户交互也可以触发 危害 危害较小 危害更大

public\comment-list.html

```
    评论列表

                        评论列表

                                        用户名

                                        内容

        function getCommentList() {
            $.get('/api/comments').then(comments => {
                let html = comments.map(item => (
                    `
                <li class="list-group-item">
                        <div class="media">
                            <div class="media-left">
                                <a href="#">
                                   <img style="border-radius:5px" class="media-object" src="${item.avatar}" >
                                </a>
                            </div>
                            <div class="media-body">
                                <h4 class="media-heading">用户名: ${item.username}</h4>
                                <p>内容: ${item.content}</p>
                                <p>时间: ${item.time}</p>
                            </div>
                        </div>
                </li>
                `
                )).join('');
                $('.comment-list').html(html);
            });
        }
        getCommentList();
        function addComment(event) {
            event.preventDefault();
            let username = $('#username').val();
            let content = $('#content').val();
            if (!content) return;
            $.post('/api/comments', { username, content }).then(data => {
                getCommentList();
                $('#content').val('');
            });
        }
```

server.js

```
let comments = [
    { avatar: 'http://cn.gravatar.com/avatar/01459f970ce17cd9e1e783160ecc951a', username: '张三', content: '今天天气不错', time: new Date().toLocaleString() },
    { avatar: 'http://cn.gravatar.com/avatar/01459f970ce17cd9e1e783160ecc951a', username: '李四', content: '是的', time: new Date().toLocaleString() }
];
app.get('/api/comments', function (req, res) {
    res.json(comments);
});
app.post('/api/comments', function (req, res) {
    let comment = req.body;
    comments.push({
        ...comment,
        avatar: 'http://cn.gravatar.com/avatar/01459f970ce17cd9e1e783160ecc951a',
        time: new Date().toLocaleString()
    });
    res.json(comments);
});
```

不需要服务器端支持，是由于DOM结构修改导致的，基于浏览器DOM解析的攻击

```
    <h1>输入链接地址，然后点击按钮h1>
    <div id="content">div>
    <input type="text" id="link">
    <button onclick="setup()">设置button>
    <script src="https://cdn.bootcss.com/jquery/3.3.1/jquery.js">script>
    <script>
        function setup() {

            let html = `${$('#link').val()}">点我`;
            $('#content').html(html);
        }
    script>
body>
```

实现XSS攻击的恶意脚本被称为 XSS payload

- 窃取用户的Cookie document.cookie
- 识别用户浏览器 navigator.userAgent
- 伪造请求 GET POST请求
- XSS钓鱼 通过XSS向网页上注入钓鱼链接，让用户访问假冒的网站
- 给cookie设置httpOnly属性 脚本无法读取该Cookie,自己也不能用，非根本解决XSS

login.html

```
    登录

                        用户名

                        密码

        function login() {
            let username = $('#username').val();
            let password = $('#password').val();
            $.post('/api/login', { username, password }).then(data => {
                if (data.code == 0) {
                    location.href = `/user.html?username=${username}`;
                }
                $('#username').val('');
                $('#password').val('');
            });
        }
```

user.html

```
        document.write(document.cookie);
```

server.js

```
let users = [{ username: 'a', password: '123456', avatar: 'http://cn.gravatar.com/avatar/01459f970ce17cd9e1e783160ecc951a' }, { username: 'b', password:
'123456', avatar: 'http://cn.gravatar.com/avatar/01459f970ce17cd9e1e783160ecc951a' }];
let userSessions = {};
app.post('/api/login', function (req, res) {
    let body = req.body;
    let user;
    for (let i = 0; i < users.length; i++) {
        if (body.username == users[i].username && body.password == users[i].password) {
            user = users[i];
            break;
        }
    }
    if (user) {
        const sessionId = 'user_' + Math.random() * 1000;
        res.cookie('username', user.username);
        res.cookie('sessionId', sessionId, { httpOnly: true });
        userSessions[sessionId] = {};
        res.json({ code: 0, user });
    } else {
        res.json({ code: 1, data: '没有该用户' });
    }
});
```

- 一般来说，URL只能使用英文字母（a-zA-Z）、数字（0-9）、-_.~4个特殊字符以及所有（;,/?:@&=+$#）保留字符。
- 如果使用了一些其他文字和特殊字符，则需要通过编码的方式来进行表示

```
var url1 = 'http://www.珠峰培训.com';
var url2 = 'http://www.a.com?名称=珠峰';
var url3 = 'http://a.com?name=?&';
```

- encodeURI encodeURI是用来编码URI的,最常见的就是编码一个 URL。encodeURI 会将需要编码的字符转换为 UTF-8 的格式。对于保留字符（;,/?:@&=+$#），以及非转义字符（字母数字以及 -_.!~*'()）不会进行转义。
- encodeURI 不转义&、?和= encodeURI(url3);//http://a.com?name=?&
- encodeURIComponent 是用来编码 URI 参数的,它会跳过非转义字符（字母数字以及-_.!~*'()）。但会转义 URL的 保留字符（;,/?:@&=+$#, encodeURIComponent(url3));//http%3A%2F%2Fa.com%3Fname%3D%3F%26
- 所有完整编码一个URL字符串需要encodeURI和encodeURIComponent联合使用 console.log(encodeURI('http://a.com?name=') + encodeURIComponent('?&')); http://a.com?name=%3F%26

在 HTML 中，某些字符是预留的，比如不能使用小于号（

- HTML 十六进制编码 &#xH;
- HTML 十进制编码 &#D;
- HTML 实体编码 < 等

在 HTML 进制编码中其中的数字则是对应字符的 unicode 字符编码。 比如单引号的 unicode 字符编码是27，则单引号可以被编码为'

```
function htmlEncode(str) {
  return String(str)
    .replace(/&/g, '&')
    .replace(/"/g, '"')
    .replace(/'/g, ''')
    .replace(/, '<')
    .replace(/>/g, '>');
}
```

avaScript 中有些字符有特殊用途，如果字符串中想使用这些字符原来的含义，需要使用反斜杠对这些特殊符号进行转义。我们称之为 Javascript编码

- 三个八进制数字，如果不够个数，前面补0，例如 "e" 编码为"\145"
- 两个十六进制数字，如果不够个数，前面补0，例如 "e" 编码为"\x65"
- 四个十六进制数字，如果不够个数，前面补0，例如 "e" 编码为"\u0065"
- 对于一些控制字符，使用特殊的C类型的转义风格（例如\n和\r）

```
var str = "zfpx"";
var str = "zfpx\"";
```

- 永远不要相信用户的输入
- 用户格式判断 白名单
- 过滤危险字符 去除
- 事件属性中

使用之前要做urlencode()

```
Document

    function htmlEncode(str) {
        return String(str)
            .replace(/&/g, '&amp;')
            .replace(/"/g, '&quot;')
            .replace(/'/g, '&#39;')
            .replace(/</g, '&lt;')
            .replace(/>/g, '&gt;');
    }

    let data = {
        desc: "<script>alert(1);<\/script>",
        clsName: '"><script>alert(2);<\/script>',
        url: '"><script>alert(3);<\/script>',
        id: '"><script>alert(4);<\/script>',
    }
    $('#intag').html(htmlEncode(data.desc));
    $('#tagAttr').html(`<a class = "${htmlEncode(data.clsName)}">标签属性中</a>`);

    $('#inEvent').html(`<a href="#" onclick = "go('${data.url}')">事件参数</a>`);
    $('#inLink').html(`<a href="http://localhost:3000/articles/${encodeURI(data.id)}">link</a>`);
    function go(url) {
        console.log(url);
    }

    //使用"\"对特殊字符进行转义，除数字字母之外，小于127使用16进制"\xHH"的方式进行编码，大于用unicode（非常严格模式）。
    var JavaScriptEncode = function (str) {
        var hex = new Array('0', '1', '2', '3', '4', '5', '6', '7', '8', '9', 'a', 'b', 'c', 'd', 'e', 'f');
        function changeTo16Hex(charCode) {
            return "\\x" + charCode.charCodeAt(0).toString(16);
        }

        function encodeCharx(original) {
            var found = true;
            var thecharchar = original.charAt(0);
            var thechar = original.charCodeAt(0);
            switch (thecharchar) {
                case '\n': return "\\n"; break; //newline
                case '\r': return "\\r"; break; //Carriage return
                case '\'': return "\\'"; break;
                case '"': return "\\\""; break;
                case '\&': return "\\&"; break;
                case '\\': return "\\\\"; break;
                case '\t': return "\\t"; break;
                case '\b': return "\\b"; break;
                case '\f': return "\\f"; break;
                case '/': return "\\x2F"; break;
                case '<': return "\\x3C"; break;
                case '>': return "\\x3E"; break;
                default:
                    found = false;
                    break;
            }
            if (!found) {
                if (thechar > 47 && thechar < 58) { //数字
                    return original;
                }

                if (thechar > 64 && thechar < 91) { //大写字母
                    return original;
                }

                if (thechar > 96 && thechar < 123) { //小写字母
                    return original;
                }

                if (thechar > 127) { //大于127用unicode
                    var c = thechar;
                    var a4 = c % 16;
                    c = Math.floor(c / 16);
                    var a3 = c % 16;
                    c = Math.floor(c / 16);
                    var a2 = c % 16;
                    c = Math.floor(c / 16);
                    var a1 = c % 16;
                    return "\\u" + hex[a1] + hex[a2] + hex[a3] + hex[a4] + "";
                }
                else {
                    return changeTo16Hex(original);
                }

            }
        }

        var preescape = str;
        var escaped = "";
        var i = 0;
        for (i = 0; i < preescape.length; i++) {
            escaped = escaped + encodeCharx(preescape.charAt(i));
        }
        return escaped;
    }
```

2. CSRF

Cross Site Request Forgery 跨站请求伪造

bank.html

```
    我的银行

                        用户名

                        余额

                                转账用户

                                金额

        $(function () {
            $.get('/api/user').then(data => {
                console.log(data);
                console.log(data.user.username);
                if (data.code == 0) {
                    $('#username').html(data.user.username);
                    $('#money').html(data.user.money);
                } else {
                    alert('用户未登录');
                    location.href = '/login.html';
                }
            });
        });
        function transfer(event) {
            event.preventDefault();
            let target = $('#target').val();
            let amount = $('#amount').val();
            $.post('/api/transfer', { target, amount }).then(data => {
                if (data.code == 0) {
                    alert('转账成功');
                    location.reload();
                } else {
                    alert('用户未登录');
                    location.href = '/login.html';
                }
            });
        }
```

```
app.get('/api/user', function (req, res) {
    let { username } = userSessions[req.cookies.sessionId];
    if (username) {
        let user;
        for (let i = 0; i < users.length; i++) {
            if (username == users[i].username) {
                user = users[i];
                break;
            }
        }
        res.json({ code: 0, user });
    } else {
        res.json({ code: 1, error: '用户没有登录' });
    }
});

app.post('/api/transfer', function (req, res) {
    let { target, amount } = req.body;
    amount = isNaN(amount) ? 0 : Number(amount);
    let { username } = userSessions[req.cookies.sessionId];
    if (username) {
        let user;
        for (let i = 0; i < users.length; i++) {
            if (username == users[i].username) {
                users[i].money -= amount;
            } else if (target == users[i].username) {
                users[i].money += amount;
            }
        }
        res.json({ code: 0 });
    } else {
        res.json({ code: 1, error: '用户没有登录' });
    }
})
```

- 用户不知情 验证码 影响用户体验
- 跨站请求 使用refer验证 不可靠
- 参数伪造 token 最主流的防御CSRF

server.js

```
var svgCaptcha = require('svg-captcha');
app.get('/api/captcha', function (req, res) {
    let session = userSessions[req.cookies.sessionId];
    if (session) {
        var codeConfig = {
            size: 5,
            ignoreChars: '0o1i',
            noise: 2,
            height: 44
        }
        var captcha = svgCaptcha.create(codeConfig);
        session.captcha = captcha.text.toLowerCase();
        res.send({ code: 0, captcha: captcha.data });
    } else {
        res.json({ code: 1, data: '没有该用户' });
    }
});
```

bank.html

```
class="form-group">
    <label for="captcha" id="captcha">label>
    <input id="captcha" class="form-control" placeholder="请输入验证码">
div>

 $.get('/api/captcha').then(data => {
    if (data.code == 0) {
        $('#captcha').html(data.captcha);
    } else {
        alert('用户未登录');
        location.href = '/login.html';
    }
});
```

```
 let referer = req.headers['referer'];
    if (/^https?:\/\/localhost:3000/.test(referer)) {

    } else {
        res.json({ code: 1, error: 'referer不正确' });
    }
```

bank.html

```
function getClientToken() {
            let result = document.cookie.match(/token=([^;]+)/);
            return result ? result[1] : '';
        }
function transfer(event) {
            event.preventDefault();
            let target = $('#target').val();
            let amount = $('#amount').val();
            let captcha = $('#captcha').val();
            $.post('/api/transfer', {
                target,
                amount,
                captcha,
                clientToken: getClientToken()
            }).then(data => {
                if (data.code == 0) {
                    alert('转账成功');
                    location.reload();
                } else {
                    alert('用户未登录');
                    location.href = '/login.html';
                }
            });
}
```

server.js

```
app.post('/api/transfer', function (req, res) {

    let { target, amount, clientToken, captcha } = req.body;
    amount = isNaN(amount) ? 0 : Number(amount);
    let { username, token } = userSessions[req.cookies.sessionId];
    if (username) {
        if (clientToken == token) {
            let user;
            for (let i = 0; i < users.length; i++) {
                if (username == users[i].username) {
                    users[i].money -= amount;
                } else if (target == users[i].username) {
                    users[i].money += amount;
                }
            }
            res.json({ code: 0 });
        } else {
            res.json({ code: 1, error: '违法操作' });
        }
    } else {
        res.json({ code: 1, error: '用户没有登录' });
    }

    res.json({ code: 1, error: 'referer不正确' });

})
```

不断传播的xss+csrf攻击 worm.js

```
const attack = '';
$.post('/api/comments', { content: 'haha' + attack });
```

3. DDOS攻击

分布式拒绝服务(Distribute Denial Of Service)

- 黑客控制大量的肉鸡向受害主机发送非正常请求，导致目标主机耗尽资源不能为合法用户提供服务
- 验证码是我们在互联网十分常见的技术之一。不得不说验证码是能够有效地防止多次重复请求的行为。
- 限制请求频率是我们最常见的针对 DDOS 攻击的防御措施。其原理为设置每个客户端的请求频率的限制
- 增加机器增加服务带宽。只要超过了攻击流量便可以避免服务瘫痪
- 设置自己的业务为分布式服务，防止单点失效
- 使用主流云系统和 CDN（云和 CDN 其自身有 DDOS 的防范作用）
- 优化资源使用提高 web server 的负载能力

4. HTTP劫持

- 在用户的客户端与其要访问的服务器经过网络协议协调后，二者之间建立了一条专用的数据通道，用户端程序在系统中开放指定网络端口用于接收数据报文，服务器端将全部数据按指定网络协议规则进行分解打包，形成连续数据报文。
- 用户端接收到全部报文后，按照协议标准来解包组合获得完整的网络数据。其中传输过程中的每一个数据包都有特定的标签，表示其来源、携带的数据属性以及要到何处，所有的数据包经过网络路径中ISP的路由器传输接力后，最终到达目的地，也就是客户端。
- HTTP劫持是在使用者与其目的网络服务所建立的专用数据通道中，监视特定数据信息，提示当满足设定的条件时，就会在正常的数据流中插入精心设计的网络数据报文，目的是让用户端程序解释"错误"的数据，并以弹出新窗口的形式在使用者界面展示宣传性广告或者直接显示某网站的内容。

参考