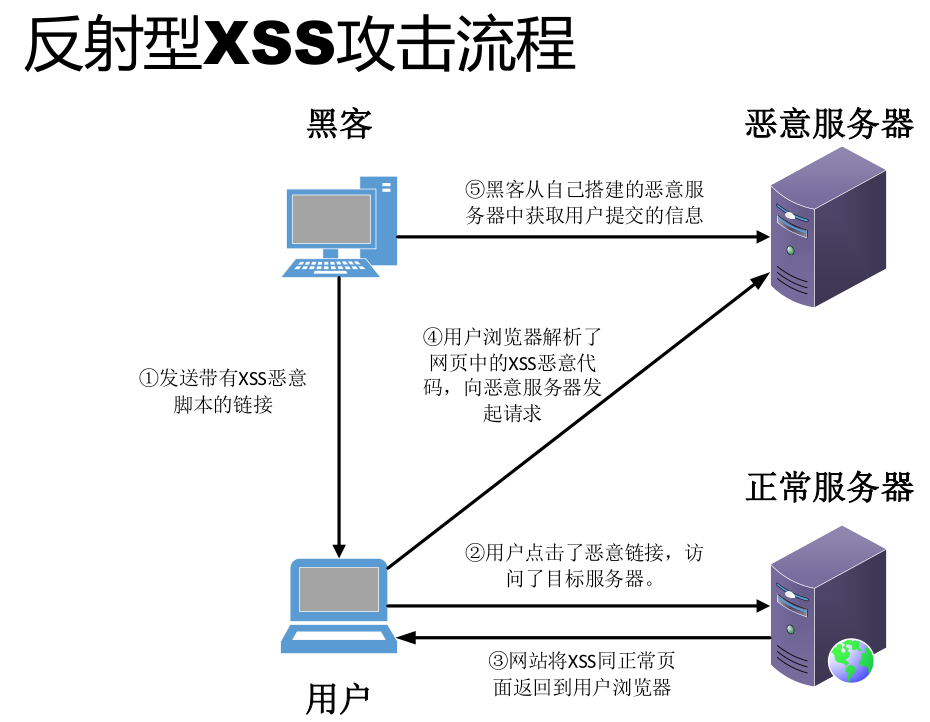
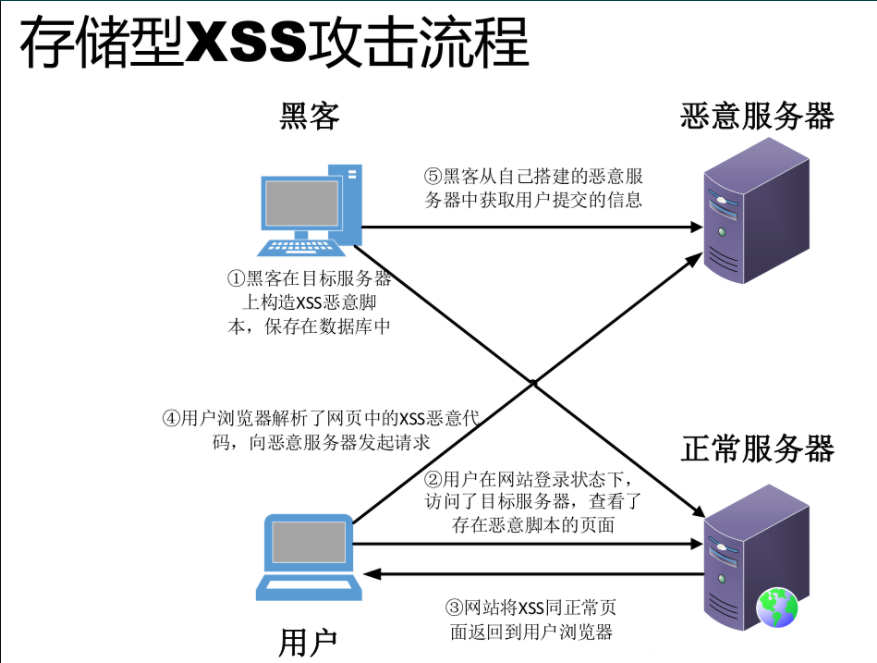
Web攻防-XSS跨站&浏览器UXSS&突变MXSS&Vue&React框架&JQuery库&写法和版本

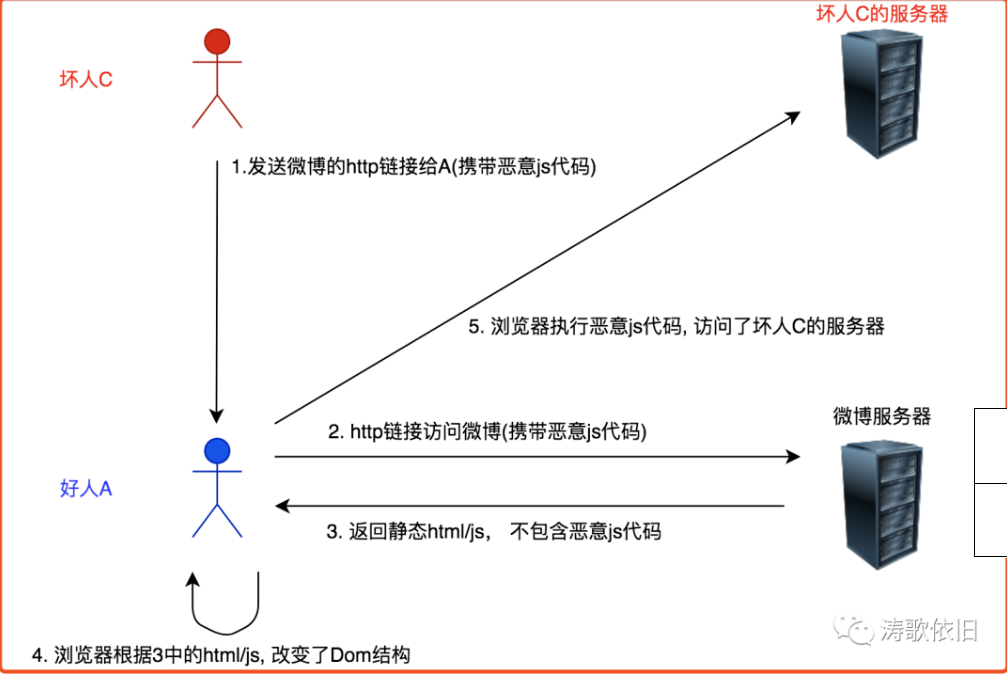


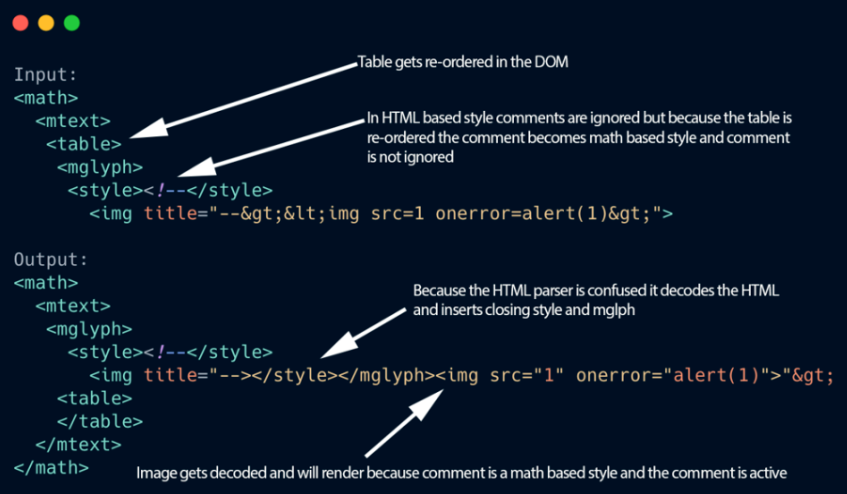
#知识点：  
1、Web攻防-SQL注入-产生原理&应用因素  
2、Web攻防-SQL注入-各类数据库类型利用  
----------------------------------------------  
1、Web攻防-SQL注入-参数类型\*参数格式  
2、Web攻防-SQL注入-XML&JSON&BASE64等  
3、Web攻防-SQL注入-数字字符搜索等符号绕过  
----------------------------------------------  
1、Web攻防-SQL注入-操作方法&增删改查  
2、Web攻防-SQL注入-HTTP头&UA&Cookie  
3、Web攻防-SQL注入-HTTP头&XFF&Referer  
----------------------------------------------  
1、Web攻防-SQL注入-操作方法&增删改查  
2、Web攻防-SQL注入-布尔&延时&报错&盲注  
----------------------------------------------  
1、Web攻防-SQL注入-高权限用户差异  
2、Web攻防-SQL注入-跨库&文件读写带外  
----------------------------------------------  
1、Web攻防-SQL注入-堆叠&二次注入  
2、Web攻防-SQL注入-SQLMAP进阶使用  
----------------------------------------------  
1、Web攻防-XSS跨站-原理&分类&利用点  
2、Web攻防-XSS跨站-反射&存储&DOM型  
----------------------------------------------  
1、Web攻防-XSS跨站-文件类型-html&pdf&swf&svg  
2、Web攻防-XSS跨站-功能逻辑-postMessage&localStorage  
----------------------------------------------  
1、Web攻防-XSS跨站-浏览器&转换-UXSS&MXSS  
2、Web攻防-XSS跨站-框架和库-VUE&React&Electron&JQuery  
   
#章节点：(待补充)  
OWTOP10，SQL注入，文件安全（上传,下载,读取,删除,包含等），目录遍历，  
CSRF，SSRF，CRLF，SSTI注入，XML&XXE安全，RCE执行（代码或命令等）  
反序列化，业务逻辑（验证码，接口枚举，支付购买，机制验证，越权，并发等）  
未授权访问，失效访问控制，弱口令安全，第三方组件安全，CORS，JSONP等

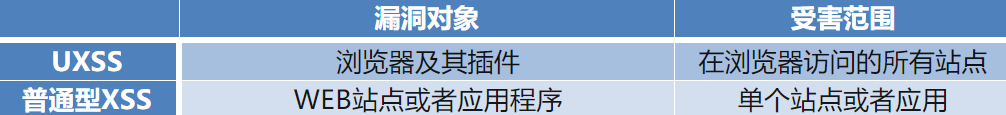












演示案例：

* WEB攻防-XSS跨站-原理&分类&利用点
* WEB攻防-XSS跨站-反射&存储&DOM型
* WEB攻防-XSS跨站-SWF&SVG&PDF&HTML等
* WEB攻防-XSS跨站-PostMessage&LocalStorage
* WEB攻防-XSS跨站-框架&库&浏览器&插件&突变
* WEB攻防-XSS跨站-MXSS&UXSS&框架版本&写法

#XSS跨站-输入输出-原理&分类&利用点  
漏洞原理：接受输入数据，输出显示数据后解析执行  
基础类型：反射(非持续)，存储(持续)，DOM-BASE  
拓展类型：jquery，mxss，uxss，pdfxss，flashxss，上传xss等  
业务类型：localStorage，postMessage，React，Electron，VUE等  
常用标签：<https://www.freebuf.com/articles/web/340080.html>  
攻击利用：盲打，COOKIE盗取，凭据窃取，页面劫持，网络钓鱼，权限维持等  
安全修复：字符过滤，实例化编码，http\_only，CSP防护，WAF拦截等  
测试流程：看输出想输入在哪里，更改输入代码看执行（标签，过滤决定）  
   
#XSS跨站-分类测试-反射&存储&DOM型  
-数据交互的地方  
 get、post、headers  
 反馈与浏览  
 富文本编辑器  
 各类标签插入和自定义  
-数据输出的地方  
 用户资料  
 数据输出  
 评论，留言等  
 关键词、标签、说明  
 文件上传  
   
1、反射非持续型：（某案例测试）  
常见情况是攻击者通过构造一个恶意链接的形式，诱导用户传播和打开，由于链接内所携带的参数会回显于页面中或作为页面的处理数据源，最终造成XSS攻击。  
   
2、存储持续型：（某案例测试）  
存储型XSS是持久化的XSS攻击方式，将恶意代码存储于服务器端，当其他用户再次访问页面时触发，造成XSS攻击。  
   
3、DOM-Base型：（某案例测试）  
通过修改原始的客户端代码，受害者浏览器的DOM环境改变，导致有效载荷的执行。页面本身没有变化，但由于DOM环境被恶意修改，代码被包含进导致执行。  
   
SRC挖洞复盘  
上海交大xss漏洞  
私信xss  
微博两个反射xss  
腾讯相册bypass-xss  
某鹅邮箱xss  
   
4、文件类型触发XSS  
#SVG-XSS  
SVG(Scalable Vector Graphics)是一种基于XML的二维矢量图格式，和我们平常用的jpg/png等图片格式所不同的是SVG图像在放大或改变尺寸的情况下其图形质量不会有所损失，并且我们可以使用任何的文本编辑器打开SVG图片并且编辑它，目前主流的浏览器都已经支持SVG图片的渲染。  
<svg xmlns="http://www.w3.org/2000/svg" version="1.1">  
 <circle cx="100" cy="50" r="40" stroke="black" stroke-width="2" fill="red" />  
 <script>alert(1)</script>  
</svg>  
   
#PDF-XSS  
1、创建PDF，加入动作JS  
2、通过文件上传获取直链  
3、直链地址访问后被触发  
项目：迅捷PDF编辑器试用版  
   
#SWF-XSS  
-制作swf-xss文件：  
1、新建swf文件  
2、F9进入代码区域  
3、属性发布设置解析  
//取m参数  
var m=\_root.m;  
//调用html中Javascript中的m参数值  
flash.external.ExternalInterface.call(m);  
触发：?m=alert(/xss/)  
项目：Adobe Flash Professional CS6  
   
-测试swf文件xss安全性：  
1、反编译swf文件  
2、查找触发危险函数  
3、找可控参数访问触发  
xss一是指执行恶意js，那么为什么说flash xss呢？是因为flash有可以调用js的函数，也就是可以和js通信，因此这些函数如果使用不当就会造成xss。常见的可触发xss的危险函数有：getURL，navigateToURL，ExternalInterface.call，htmlText，loadMovie等等  
项目：JPEXS Free Flash Decompiler  
   
1、上传swf文件可以做xss漏洞  
2、找到目标上存在的swf进行反编译后找xss漏洞  
#HTML-XSS  
单纯在HTML代码中写XSS代码即可  
   
#其他：XML格式等  
挖掘：从安全文件上传到XSS的转换（红队玩法还可以配合钓鱼）  
利用文件上传获取文件访问地址，访问触发（浏览器格式解析问题会导致失效）  
   
5、功能逻辑触发XSS：  
#PostMessage XSS  
一个用于在网页间安全地发送消息的浏览器API。它允许不同的窗口（例如，来自同一域名下的不同页面或者不同域名下的跨域页面）进行通信，而无需通过服务器。通常情况下，它用于实现跨文档消息传递（Cross-Document Messaging），这在一些复杂的网页应用和浏览器插件中非常有用。  
   
安全原因：当发送参数可控且接收方处理不当时，将导致XSS  
   
模拟漏洞挖掘场景：  
打开<http://192.168.1.4:82/60/xssreceive.html>  
分析源码：<script>  
 // 添加事件监控消息  
 window.addEventListener("message", (event) => {  
 location.href = `${event.data.url}`;  
 });  
 </script>  
黑盒挖掘：  
如何去挖这种隐藏比较深的漏洞，这里强推一个国外白帽写的谷歌浏览器插件：  
<https://github.com/fransr/postMessage-tracker>安装之后，只要是当前页面创建了message事件监听，这个插件就会定位到其代码。  
   
白盒挖掘：  
找代码中此操作类函数及关键字分析  
复盘：https://mp.weixin.qq.com/s/M5YIkJEoHZK6\_I7nK6aj5w  
   
#localStorage型xss（存储型Xss升级版）  
Web应用使用localStorage在用户浏览器中存储数据。localStorage允许网页在本地存储键值对，这些数据可以在浏览器关闭后仍然保留，并且在同一域名下的不同页面之间共享。当应用程序从localStorage中读取数据并将其显示在页面上时，如果没有对数据进行充分的验证和过滤，攻击者就有可能通过修改localStorage中的数据来注入恶意脚本实现XSS。  
   
安全原因：当localStorage可控且有页面有操作其数据，将导致XSS  
黑盒挖掘：寻找输入点（如表单、搜索框、评论区等），构造测试用例找页面调用  
白盒挖掘：查找与localStorage相关的代码段分析数据存储及数据读取  
挖掘：先看目标应用localStorage  
1、先找能控制的键名键值（怎么找）  
2、再找对键名键值做输出操作（怎么找）  
   
6、第三方或框架等因素触发XSS  
Jquery XSS  
Vue xss  
React xss  
Electron XSS  
mxss  
uxss  
.......  
   
分类：  
1、框架或三方库的XSS  
2、浏览器或插件的XSS  
3、客户端预览内核的XSS  
说明：使用框架开发的或第三方库引用操作的，默认安全写法会自带过滤，所以测试此类的应用需存在漏洞版本或不安全写法导致XSS；同样UXSS也需要存在漏洞的浏览器版本或插件导致；MXSS也要不同环境下的转变解析导致，需多测试。  
   
#Vue-XSS  
搭建：  
npm create vite@latest  
cd vue-xss-demo  
npm install  
修改：  
App.vue:  
<template>  
 <div>  
 <h1>XSS 漏洞演示</h1>  
 <input v-model="userInput" placeholder="输入你的内容" />  
 <button @click="showContent">显示内容</button>  
 <div v-html="displayContent"></div>  
 </div>  
</template>  
   
<script>  
export default {  
 data() {  
 return {  
 userInput: '', // 用户输入  
 displayContent: '' // 显示的内容  
 };  
 },  
 methods: {  
 showContent() {  
 // 直接将用户输入的内容渲染到页面  
 this.displayContent = this.userInput;  
 }  
 }  
};  
</script>  
   
<style>  
#app {  
 font-family: Avenir, Helvetica, Arial, sans-serif;  
 text-align: center;  
 margin-top: 60px;  
}  
</style>  
   
启动：npm run dev  
测试：<img src="x" onerror="alert('XSS')" />  
修复：使用文本插值（{{}}）代替 v-html  
   
#React-XSS  
搭建：  
npx create-react-app react-xss-example  
cd react-xss-example  
修改：  
App.js  
import React, { useState } from 'react';  
import ReactDOM from 'react-dom';  
   
function App() {  
 const [userInput, setUserInput] = useState('');  
 const [displayedInput, setDisplayedInput] = useState('');  
   
 const handleInputChange = (e) => {  
 setUserInput(e.target.value);  
 };  
   
 const displayInput = () => {  
 setDisplayedInput(userInput);  
 };  
   
 return (  
 <div>  
 <input type="text" value={userInput} onChange={handleInputChange} placeholder="输入内容" />  
 <button onClick={displayInput}>显示输入</button>  
 <div dangerouslySetInnerHTML={{\_\_html: displayedInput}}/>  
 {/\*<div>{displayedInput}</div>\*/}  
 </div>  
 );  
}  
export default App;  
启动：npm start  
测试：<img src="x" onerror="alert('XSS')" />  
修复：直接使用{displayedInput}来显示  
   
#Electron-XSS  
搭建：  
mkdir electron-xss-example  
cd electron-xss-example  
npm init -y  
安装修改：  
npm install electron --save-dev  
main.js 和 index.html 文件复制到项目根目录下  
// main.js  
const { app, BrowserWindow } = require('electron');  
const path = require('path');  
   
function createWindow() {  
 const win = new BrowserWindow({  
 width: 800,  
 height: 600,  
 webPreferences: {  
 nodeIntegration: true,  
 contextIsolation: false  
 }  
 });  
   
 win.loadFile('index.html');  
}  
   
app.whenReady().then(() => {  
 createWindow();  
   
 app.on('activate', function () {  
 if (BrowserWindow.getAllWindows().length === 0) createWindow();  
 });  
});  
   
app.on('window-all-closed', function () {  
 if (process.platform !== 'darwin') app.quit();  
});  
   
// index.html  
<!DOCTYPE html>  
<html>  
   
<head>  
 <meta charset="UTF-8">  
 <title>Electron XSS Example</title>  
</head>  
   
<body>  
 <input type="text" id="userInput" placeholder="输入内容">  
 <button onclick="displayInput()">显示输入</button>  
 <div id="displayArea"></div>  
 <script>  
 function displayInput() {  
 const input = document.getElementById('userInput').value;  
 const displayArea = document.getElementById('displayArea');  
 displayArea.innerHTML = input;  
 }  
 </script>  
</body>  
   
</html>  
配置package.json  
{  
 "name": "electron-xss-example",  
 "version": "1.0.0",  
 "description": "",  
 "main": "main.js",  
 "scripts": {  
 "start": "electron ."  
 },  
 "keywords": [],  
 "author": "",  
 "license": "ISC",  
 "devDependencies": {  
 "electron": "^23.2.1"  
 }  
}  
启动：npm start  
测试：<img src="x" onerror="alert('XSS')" />  
修复：使用 textContent 代替 innerHTML 来显示文本  
   
#JQuery XSS  
参考：  
水洞：https://mp.weixin.qq.com/s/FsFvQlVrb\_J4wsyE8gpprA  
介绍：https://mp.weixin.qq.com/s/EMsK1c901-bDYapvHxs-VQ  
漏扫：  
https://github.com/mahp/jQuery-with-XSS  
https://github.com/honeyb33z/cve-2020-11023-scanner  
   
#MXSS：  
参考：  
https://mp.weixin.qq.com/s/31zaBzZ1e6rNobYCrn7Qhg  
模拟：（见图）  
https://portswigger-labs.net/mxss/  
<math><mtext><table><mglyph><style><!--</style><img title="--&gt;&lt;img src=1 onerror=alert(1)&gt;">  
复盘：  
https://www.fooying.com/the-art-of-xss-1-introduction/  
   
#UXSS：Universal Cross-Site Scripting  
UXSS利用浏览器或者浏览器扩展漏洞来制造产生XSS并执行代码的攻击类型。  
复盘：  
MICROSOFT EDGE uXSS CVE-2021-34506  
Edge浏览器翻译功能导致JS语句被调用执行  
https://www.bilibili.com/video/BV1fX4y1c7rX  
https://mp.weixin.qq.com/s/rR2feGeuRt3hOFPkV3\_6Ow

涉及资源：[资源下载地址](https://docs.qq.com/doc/DQ3Z6RkNpaUtMcEFr)