COMPASS CTF Tutorial 2: Network Protocols and Web Vulnerabilities

COMPASS CTF 教程【2】：网络协议与 Web 漏洞

30016794 Zhao, Li (Research Assistant)

COMPuter And System Security Lab, Computer Science and Technology Department, College of Engineering (CE), SUSTech University.
南方科技大学 工学院 计算机科学与技术系 计算机与系统安全实验室

2023 年 8 月 6 日

什么是 HTML

HTML 是用来描述网页的一种语言。[1]

- HTML 指的是超文本标记语言 (Hyper Text Markup Language)
- HTML 不是一种编程语言，而是一种标记语言 (Markup language)
- 标记语言是一套标记标签 (Markup tag)
- HTML 使用标记标签来描述网页

什么是 HTML

总的来说，HTML 本身不具有编程逻辑，它是一种将格式与内容分离编排的语言。
用户在浏览器端解析的网页大都是由 HTML 语言组成。
由于是通过浏览器动态解析，因此可以使用普通文本编辑器来编写 HTML。

HTML 中的标签与元素

标签和元素共同构成了 HTML 多样的格式和丰富的功能。
HTML 元素以开始标签起始，以结束标签终止。元素处于开始标签与结束标签之间，标签之间可以嵌套，一个典型的 HTML 文档如下：

```
<html>
<!– html 文档声明标签 –>
<head>
<!–头部–>
<meta charset=”utf-8”> <!–网站编码–>
<title>标题</title>
</head>
```

HTML 中的标签与元素

<body>
<!– html 文档主体 –>
Hello World
<!– 注释 –>
</body>
</html>

信息隐藏

HTML 中的部分标签用于元信息展示、注释等功能，并不用于内容的显示。另一方面，一些属性具有修改浏览器显示样式的功能，在 CTF 中常被用来进行信息隐藏。

标签

<!–...–>，定义注释

<!DOCTYPE>，定义文档类型

<head>，定义关于文档的信息

<meta>，定义关于 HTML 文档的元信息

<iframe>，定义内联框架

属性

hidden，隐藏元素

XSS

关于 XSS 漏洞的详细介绍见 OWASP Top Ten Project 漏洞基础。导致 XSS 漏洞的原因是嵌入在 HTML 中的其它动态语言，但是 HTML 为恶意注入提供了输入口。常见与 XSS 相关的标签或属性如下：

<script>，定义客户端脚本
<img src=>，规定显示图像的 URL
<body background=>，规定文档背景图像 URL
<body onload=>，body 标签的事件属性
<input onfocus= autofocus>，form 表单的事件属性
<button onclick=>，击键的事件属性
<link href=>，定义外部资源链接
<object data=>，定义引用对象数据的 URL
<svg onload=>，定义 SVG 资源引用

HTML 编码

HTML 编码是一种用于表示问题字符已将其安全并入 HTML 文档的方案。HTML 定义了大量 HTML 实体来表示特殊的字符。

HTML 编码	特殊字符
&quot	"
&apos	'
&amp	&
&lt	<
&gt	>

HTML 编码

此外，任何字符都可以使用它的十进制或十六进制的 ASCII 码进行 HTML 编码，例如：

HTML 编码	特殊字符
&#34	"
&#39	'
&#x22	"
&#x27	'
&gt	>

HTML5 新特性

其实 HTML5 已经不新了，之所以还会在这里提到 HTML5，是因为更强大的功能会带来更多意想不到的问题。[2]
HTML5 的一些新特性：

- 新的语义元素标签
- 新的表单控件
- 强大的图像支持
- 强大的多媒体支持
- 强大的 API

什么是 HTTP

HTTP 是 Web 领域的核心通信协议。最初的 HTTP 支持基于文本的静态资源获取，随着协议版本的不断迭代，它已经支持如今常见的复杂分布式应用程序。
HTTP 使用一种基于消息的模型，建立于 TCP 层之上。由客户端发送一条请求消息，而后由服务器返回一条响应消息。

HTTP 请求与响应

一次完整的请求或响应由消息头、一个空白行和消息主体构成。以下是一个典型的 HTTP 请求：

```
GET / HTTP/1.1
Host: www.github.com
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64; rv:52.0) Gecko/20100101
Firefox/52.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: zh-CN,zh;q=0.8,en-US;q=0.5,en;q=0.3
Accept-Encoding: gzip, deflate
Upgrade-Insecure-Requests: 1
Cookie: logged_in=yes;
Connection: close
```

HTTP 请求与响应

第一行分别是请求方法，请求的资源路径和使用的 HTTP 协议版本，第二至九行为消息头键值对。

以下是对上面请求的回应（并不一定和真实访问相同，这里只是做为示例）：

```
HTTP/1.1 200 OK
Date: Tue, 26 Dec 2017 02:28:53 GMT
Content-Type: text/html; charset=utf-8
Connection: close
Server: GitHub.com
Status: 200 OK
<!DOCTYPE html>
.....
```

第一行为协议版本、状态号和对应状态的信息，第二至六为返回头键值对，紧接着为一个空行和返回的内容实体。

HTTP 方法

在提到 HTTP 方法之前，我们需要先讨论一下 HTTP 版本问题。HTTP 协议现在共有三个大版本，版本差异会导致一些潜在的漏洞利用方式。[3]

版本	简述
HTTP 0.9	该版本只允许 GET 方法，具有典型的无状态性， 无协议头和状态码，支持纯文本
HTTP 1.0	增加了 HEAD 和 POST 方法，支持长连接、缓存和身份认证
HTTP 1.1	增加了 Keep-alive 机制和 PipeLining 流水线， 新增了 OPTIONS、PUT、DELETE、TRACE、CONNECT 方法
HTTP 2.0	增加了多路复用、头部压缩、随时复位等功能

HTTP 方法

请求方法	描述
GET	请求获取 URL 资源
POST	执行操作，请求 URL 资源后附加新的数据
HEAD	只获取资源响应消息报头
PUT	请求服务器存储一个资源
DELETE	请求服务器删除资源
TRACE	请求服务器回送收到的信息
OPTIONS	查询服务器的支持选项

URL

URL 是统一资源定位符，它代表了 Web 资源的唯一标识，如同电脑上的盘符路径。最常见的 URL 格式如下所示：[4]

protocol://[user[:password]@]hostname[:post]/[path]/file[?param=value]

协议 分隔符 用户信息 域名 端口 路径 资源文件 参数键 参数值

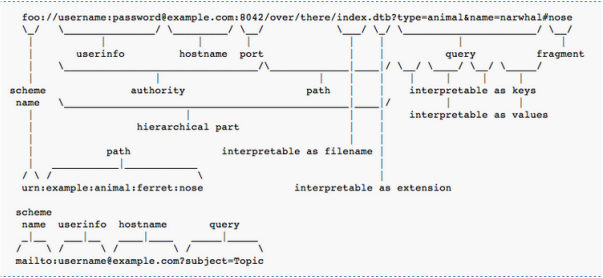

图: URL 示例

HTTP 消息头

HTTP 支持许多不同的消息头，一些有着特殊作用，而另一些则特定出现在请求或者响应中。

消息头	描述	备注
Connection	告知通信另一端，在完成 HTTP 传输后是关闭 TCP 连接，还是保持连接开放	
Content-Encoding	规定消息主体内容的编码形式	
Content-Length	规定消息主体的字节长度	
Content-Type	规定消息主体的内容类型	

HTTP 消息头

消息头	描述	备注
Accept	告知服务器客户端愿意接受的内容类型	请求
Accept-Encoding	告知服务器客户端愿意接受的内容编码	请求
Authorization	进行内置 HTTP 身份验证	请求
Cookie	用于向服务器提交 cookie	请求
Host	指定所请求的完整 URL 中的主机名称	请求
Origin	跨域请求中的请求域	请求
Referer	指定提出当前请求的原始 URL	请求
User-Agent	提供浏览器或者客户端软件的有关信息	请求

HTTP 消息头

消息头	描述	备注
Cache-Control	向浏览器发送缓存指令	响应
Location	重定向响应	响应
Server	提供所使用的服务器软件信息	响应
Set-Cookie	向浏览器发布 cookie	响应
WWW-Authenticate	提供服务器支持的验证信息	响应

Cookie

Cookie 是大多数 Web 应用程序所依赖的关键组成部分，它用来弥补 HTTP 的无状态记录的缺陷。服务器使用 Set-Cookie 发布 cookie，浏览器获取 cookie 后每次请求会在 Cookie 字段中包含 cookie 值。
Cookie 是一组键值对，另外还包括以下信息：

- expires，用于设定 cookie 的有效时间。
- domain，用于指定 cookie 的有效域。
- path，用于指定 cookie 的有效 URL 路径。
- secure，指定仅在 HTTPS 中提交 cookie。
- HttpOnly，指定无法通过客户端 JavaScript 直接访问 cookie。

状态码

状态码表明资源的请求结果状态，由三位十进制数组成，第一位代表基本的类别：

- 1xx，提供信息
- 2xx，请求成功提交
- 3xx，客户端重定向其他资源
- 4xx，请求包含错误
- 5xx，服务端执行遇到错误

状态码

常见的状态码及短语如下所示：

状态码	短语	描述
100	Continue	服务端已收到请求并要求客户端继续发送主体
200	Ok	已成功提交，且响应主体中包含请求结果
201	Created	PUT 请求方法的返回状态，请求成功提交
301	Moved Permanently	请求永久重定向
302	Found	暂时重定向
304	Not Modified	指示浏览器使用缓存中的资源副本

状态码

状态码	短语	描述
400	Bad Request	客户端提交请求无效
401	Unauthorized	服务端要求身份验证
403	Forbidden	禁止访问被请求资源
404	Not Found	所请求的资源不存在
405	Method Not Allowed	请求方法不支持
413	Request Entity Too Large	请求主体过长
414	Request URI Too Long	请求 URL 过长
500	Internal Server Error	服务器执行请求时遇到错误
503	Service Unavailable	Web 服务器正常，但请求无法被响应

状态码

401 状态支持的 HTTP 身份认证：

- Basic，以 Base64 编码的方式发送证书
- NTLM，一种质询-响应机制
- Digest，一种质询-响应机制，随同证书一起使用一个随机的 MD5 校验和

HTTPS

HTTPS 用来弥补 HTTP 明文传输的缺陷。通过使用安全套接字 SSL，在端与端之间传输加密后的消息，保护传输数据的隐密性和完整性，并且原始的 HTTP 协议依然按照之前同样的方式运作，不需要改变。

使用浏览器执行前端 JavaScript

大多数浏览器通过 F12 可以调出调试窗口，如图所示。在调试窗口中可以执行相关代码。JS 是一种解释性语言，由解释器对代码进行解析。

```
console.log("Hello World!")
```

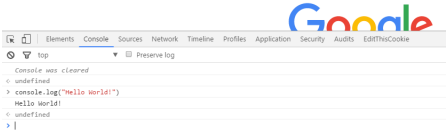

图: 在 chrome 浏览器中使用 F12

使用浏览器执行前端 JavaScript

在浏览器中，会集成 JS 的解析引擎，不同的浏览器拥有不同的解析引擎，这就使得 JS 的执行在不同浏览器上有不同的解释效果。

浏览器	引擎
IE/Edge	Chakra
Firefox	SpiderMonkey
Safari	SFX
Chrome	V8
Opera	Carakan

使用浏览器执行前端 JavaScript

嵌入在 HTML 中的 JS 代码通常有以下几种形式：

直接插入代码块

```
<script>console.log('Hello World!');</script>
```

加载外部 JS 文件

```
<script src="Hello.js"></script>
```

使用 HTML 标签中的事件属性

```
<a href="javascript:alert('Hello')"></a>
```

JavaScript 数据类型

作为弱类型的语言，JS 的变量声明不需要指定数据类型：

```
var pi=3.14;
var pi='ratio of the circumference of a circle to its diameter';
```

当然，可以通过 “ new ” 来声明变量类型：

```
var pi=new String;
var pi=new Number;
var pi=new Boolean;
var pi=new Array;
var pi=new Object;
```

JavaScript 数据类型

上一个示例也展示了 JS 的数据类型，分别是字符串、数字、布尔值、数组和对象。有两个特殊的类型是 Undefined 和 Null，形象一点区分，前者表示有坑在但坑中没有值，后者表示没有坑。另外，所有 JS 变量都是对象，**但是需要注意的是，对象声明的字符串和直接赋值的字符串并不严格相等**。

JavaScript 编程逻辑

基础

JS 语句使用分号分隔。

逻辑语句

if 条件语句:

if (condition) {

代码块

}

else {

代码块

}

JavaScript 编程逻辑

switch 条件语句:
switch(n) {
case 1:
代码块
break;
case 2:
代码块
break;
default:
代码块
}

JavaScript 编程逻辑

for/for in 循环语句:
for (代码 1; 代码 2; 代码 3) {
代码块
}
for (x in xs) {
代码块
}

JavaScript 编程逻辑

while/do while 循环语句:

```
while (条件) {
代码块
}
do {
代码块
}
while (条件);
```

JavaScript 打印数据

在浏览器中调试代码时，经常用到的手段是打印变量。

函数	作用
window.alert()	弹出警告框
document.write()	写入 HTML 文档
console.log()	写入浏览器控制台

JavaScript 打印数据

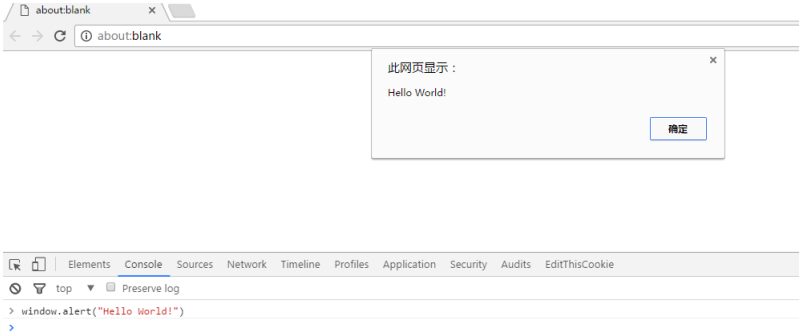

图: 浏览器弹出警示框

JavaScript 打印数据

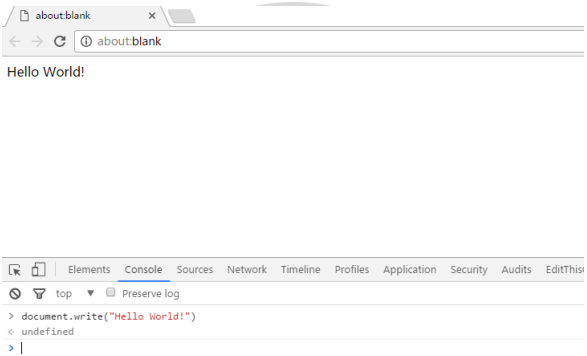

图: 写入 HTML 文档

JavaScript 框架

JS 同样有许多功能强大的框架。大多数的前端 JS 框架使用外部引用的方式将 JS 文件引入到正在编写的文档中。

jQuery

jQuery 封装了常用的 JS 功能，通过选择器的机制来操纵 DOM 节点，完成复杂的前端效果展示。

Angular

实现了前端的 MVC 架构，通过动态数据绑定来简化数据传递流程。

JavaScript 框架

React
利用组件来构建前端 UI 的框架。
Vue
MVVM 构架的前端库，理论上讲，将它定义为数据驱动、组件化的框架，但这些概念也可能适用于其他框架，所以可能只有去真正使用到所有框架才能领悟到它们之间的区别。
其他
还有许许多多针对不同功能的框架，比如针对图表可视化、网络信息传递或者移动端优化等等。

JavaScript 框架

双向数据绑定

传统基于 MVC 的架构的思想是数据单向的传送到 View 视图中进行显示，但是有时我们还需要将视图层的数据传输回模型层，这部分的功能就由前端 JS 来接手，因此许多近几年出现的新框架都使用数据双向绑定来完成 MVVM 的新构架，这就带给了用户更多的权限接触到程序的编程逻辑，进而产生一些安全问题，比较典型的就是许多框架曾经存在的模板注入问题。

JavaScript DOM 和 BOM

DOM	文档对象模型，JS 通过操纵 DOM 可以动态获取、修改 HTML 中的元素、属性、CSS 样式，这种修改有时会带来 XSS 攻击风险
BOM	浏览器对象模型，类比于 DOM，赋予 JS 对浏览器本身进行有限的操纵，获取 Cookie、地理位置、系统硬件或浏览器插件信息等

JavaScript 混淆

由于前端代码的可见性，出于知识产权或者其他目的，JS 代码通过混淆的方法使得自己既能被浏览器执行，又难以被人为解读。常见的混淆方法有重命名变量名和函数名、挤压代码、拼接字符、使用动态执行函数在函数与字符串之间进行替换等。下面对比代码混淆前后的差异。

混淆前：

```
console.log('Hello World!');
```

JavaScript 混淆

混淆后:

```
console["\x6c\x6f\x67"]('\x48\x65\x6c\x6c\x6f \x57\x6f\x72\x6c\x64\x21');
```

更加复杂的混淆后:

```
eval(function(p,a,c,k,e,d){e=function(c){return(c<a?"":e(parseInt(c/a)))+((c=c%a)>35?
String.fromCharCode(c+29):c.toString(36))};if(!''.replace(/^/,String)){while(c--)d[e(c)]=k[c]||e(c);k=[function(e){return
d[e]}];e=function(){return'\\w+'};c=1;};while(c--)if(k[c])p=p.replace(new RegExp('\\b'+e(c)+'\\b','g'),k[c]);return p;}
('1.0(\'3 2!\');',4,4,'log|console|World|Hello'.split('|'),0,{}))
```

由于之前提到的特性，无论混淆有多么复杂，最终它都能够在浏览器中被解释执行。

使用 Node.js 执行后端 JavaScript

在安装完成 Node.js 后，我们可以尝试编写第一个后端 JS 程序。

1. 打开文本编辑器，写入

```
console.log("Hello World");
```

并保存为 hello.js

2. 使用

```
node hello.js
```

来执行文件。[5]

使用 Node.js 执行后端 JavaScript

图: Node.js

世界各地的网站都使用各种编程语言进行编程。虽然每种编程语言都有开发人员应该注意的特定漏洞，但无论选择哪种语言或框架，都会出现一些互联网的基本问题。[6]

这些漏洞通常会在 CTF 中以网络安全挑战的形式出现，用户需要利用漏洞来获得某种更高级别的权限。

SQL 注入

SQL 注入是指应用程序在接收用户输入时，未验证用户输入是否包含附加 SQL 的漏洞。

```
<?php
$username = $_GET['username']; // kchung
$result = mysql_query("SELECT * FROM users WHERE username='$username'");
?>
```

SQL 注入

如果我们看一下 $username 变量，在正常操作下，我们可能会认为用户名参数是一个真实的用户名（如 kchung）。
但恶意用户可能会提交不同类型的数据。例如，如果输入的是 ′ ？
应用程序就会崩溃，因为生成的 SQL 查询不正确。
SELECT * FROM users WHERE username=”’

SQL 注入

有了单引号会导致应用程序出错的知识，我们就可以进一步扩展 SQL 注入。
如果我们的输入是 $'OR1=1$ ？
SELECT * FROM users WHERE username=" OR 1=1
在 SQL 中，1 确实等于 1。如果我们重新解释，SQL 语句实际上是说：
SELECT * FROM users WHERE username=" OR true

SQL 注入

这将返回表中的每一行，因为存在的每一行都必须为真。
我们还可以注入注释和终止字符，如 −− 或 /∗ 或 ; 。这样就可以在注入语句后终止 SQL 查询。例如，′ − − 是一种常见的 SQL 注入有效载荷。
SELECT * FROM users WHERE username=”− ’
该有效载荷将用户名参数设置为空字符串，以跳出查询，然后添加了一个注释 (−−)，有效地隐藏了第二个单引号。
利用这种在现有查询中添加 SQL 语句的技术，我们可以迫使数据库返回本不应该返回的数据。

命令注入

命令注入是一个允许攻击者向运行网站的计算机提交系统命令的漏洞。当应用程序未能对进入系统 shell 的用户输入进行编码时，就会发生这种情况。当开发人员在应用程序的编程语言中使用 system() 命令或等效命令时，这种漏洞非常常见。

```
import os
domain = user_input() # ctf101.org
os.system('ping ' + domain)
```

上述代码在正常情况下将会 ping 通 ctf101.org 域名。

但要考虑一下如果 user_input() 函数返回不同的数据会发生什么呢？

命令注入

```
import os
domain = user_input() # ; ls
os.system('ping ' + domain)
```

因为有额外的分号，os.system() 函数被指示运行两个命令。
程序看起来如下所示：

```
ping ; ls
```

命令注入

由于 ping 命令被终止并添加了 ls 命令，ls 命令将会除了空的 ping 命令之外被执行！这是命令注入的核心概念。ls 命令当然可以与另一个命令（如 wget 、curl 、bash 等）进行替换。

命令注入是 Web 应用程序和与系统命令进行接口的应用程序中非常常见的特权升级手段。许多类型的家用路由器接受用户输入并直接将其添加到系统命令中。因此，许多此类家用路由器型号容易受到命令注入的攻击。

目录遍历

目录遍历是一种漏洞，应用程序接受用户输入并将其用于目录路径。
任何由用户输入控制但未经适当净化或适当沙盒化的路径都可能容易受到目录遍历攻击。
例如，考虑一个允许用户从 GET 参数中选择要加载的页面的应用程序。

```
<?php
$page = $_GET['page']; // index.php
include("/var/www/html/" . $page);
?>
```

目录遍历

在正常操作下，页面应该是 index.php 。但是，如果有恶意用户输入了其他内容呢？

```
<?php
$page = $_GET['page']; // ../../../../../../../../etc/passwd
include("/var/www/html/" . $page);
?>
```

在这里，用户提交了 *../../../../../../../../etc/passwd* 。
这将导致 PHP 解释器离开其预设查找的目录（'/var/www/html'）而被强制上移到根目录。

```
include("/var/www/html/../../../../../../../../etc/passwd");
```

目录遍历

最终，这将变成了 /etc/passwd ，因为计算机不会跳出其顶级目录的上一级目录。因此，该应用程序将加载 /etc/passwd 文件并将其发送给用户，如下所示：

root:x:0:0:root:/root:/bin/bash
daemon:x:1:1:daemon:/usr/sbin:/usr/sbin/nologin
bin:x:2:2:bin:/bin:/usr/sbin/nologin
...

同样的概念可以应用于从用户那里接收某些输入，然后用于访问文件、路径或类似内容的应用程序。这种漏洞经常被用来泄露敏感数据或提取应用程序源代码以寻找其他漏洞。

CSRF

跨站请求伪造或 CSRF 攻击，发音为 see surf ，在已认证用户上进行的一种攻击，利用状态会话来执行状态更改攻击，如购买、资金转移或更改电子邮件地址。CSRF 的整个前提是基于会话劫持，通常是通过在网页中注入恶意元素，例如 img 标签或 iframe ，来引用未经验证的外部资源。

CSRF

GET 请求通常被网站用于获取用户输入。假设用户登录到一个银行网站，该网站为其浏览器分配一个 cookie 以保持登录状态。如果他们转账一些钱，发送到服务器的 URL 可能具有以下模式：

http://securibank.com/transfer.do?acct=[RECEPIENT]&amount=[DOLLARS]

了解这种格式，攻击者可以发送一封带有超链接的电子邮件，也可以包含一个大小为 0 乘 0 像素的图像标签，该标签将自动由浏览器请求，如下所示：

<img
src="http://securibank.com/transfer.do?acct=[RECEPIENT]&amount=[DOLLARS]"
width="0" height="0" border="0">

XSS

跨站脚本攻击或 XSS 是一种漏洞，其中一个应用程序的用户可以发送 JavaScript 代码，由同一应用程序的其他用户的浏览器执行。
这是一种漏洞，因为 JavaScript 对用户的 Web 浏览器有很高的控制权限。
例如，JavaScript 具有以下能力：

- 修改页面（称为 DOM）
- 发送更多的 HTTP 请求
- 访问 Cookies

XSS

通过结合所有这些能力，XSS 可以恶意使用 JavaScript 提取用户的 Cookies 并将其发送到被攻击者控制的服务器。XSS 还可以修改 DOM 以钓鱼用户的密码。这只是 XSS 可以用于的一小部分。
XSS 通常被分为三个类别：

- 反射型 XSS
- 存储型 XSS
- DOM XSS

XSS

反射型 XSS 是指通过 URL 参数提供 XSS 攻击的一种方式。
例如：
https://ctf101.org?data=<script>alert(1)</script>
可以在数据的 GET 参数中看到提供的 XSS 攻击。如果应用程序容易受到反射型 XSS 攻击，应用程序将会采用这个数据参数值，并将其注入到 DOM 中。

```
<html>
<body>
<script>alert(1)</script>
</body>
</html>
```

XSS

根据注入攻击的位置不同，它可能需要以不同的方式构建。
此外，攻击有效载荷可以根据攻击者的需求进行更改，无论是提取 Cookie 并将其提交给外部服务器，还是仅仅修改页面以进行篡改。
然而，反射型 XSS 的一个缺点是它需要受害者从攻击者所控制的资源中访问易受攻击的页面。请注意，如果未提供数据参数，攻击将无法生效。
在许多情况下，浏览器可以很容易地检测到 URL 中的恶意 XSS 有效载荷，因此反射型 XSS 经常被浏览器检测到。

XSS

存储型 XSS 与反射型 XSS 在一个关键的方面有所不同。在反射型 XSS 中，攻击通过 GET 参数提供。但在存储型 XSS 中，攻击是通过网站本身提供的。
想象一个允许用户发表评论的网站。如果一个用户可以将 XSS 有效载荷提交为评论，然后其他人查看这条恶意评论，那就是存储型 XSS 的一个例子。
原因在于网站本身向其他用户提供了 XSS 有效载荷。这使得从浏览器的角度很难检测，没有任何浏览器可以通用地防止存储型 XSS 对用户进行攻击。

XSS

DOM XSS 是由浏览器本身将 XSS 有效载荷注入到 DOM 中造成的 XSS 攻击。虽然服务器本身可能会正确地防止 XSS 攻击，但客户端脚本可能会意外地接收有效载荷并将其插入到 DOM 中，从而触发有效载荷。
服务器本身不应受责备，而是客户端的 JavaScript 文件导致了这个问题。

参考文献

[1] https://firmianay.gitbook.io/ctf-all-in-one/1_basic/1.4_web_basic/1.4.1_html_basic.

[2] https://html5sec.org/.

[3] https://www.cnblogs.com/andashu/p/6441271.html.

[4] Wikimedia Foundation. 2023. https://en.wikipedia.org/wiki/URL.

[5] 浅谈 node.js 安全 [EB/OL]. https://zhuanlan.zhihu.com/p/28105239.

[6] LLC O L C. Web exploitation[EB/OL]. https://ctf101.org/web-exploitation/overview/.