

Cookie之困

走 马

清华大学网络与信息安全实验室NISL



Cookies Lack Integrity: Real-World Implications

USENIX Security '15

*Xiaofeng Zheng, Jian Jiang, Jinjin Liang, Haixin Duan,
Shuo Chen, Tao Wan and Nicholas Weaver*

<https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/zheng>

谁忽视了Cookie安全



星巴克



离开星巴克时需要做什么？

Cookie基础

- 用于保持HTTP会话状态/缓存信息

- 由服务器/脚本写入

Server:

```
Set-Cookie: user=bob; domain=.bank.com; path=/;
```

JS:

```
document.cookie="user=bob; domain=.bank.com; path=/";
```

- 存储于浏览器/传输于HTTP头部

Cookie: user=bob; cart=books;

JS:

```
console.log(document.cookie);  
→ "user=bob; cart=books;"
```

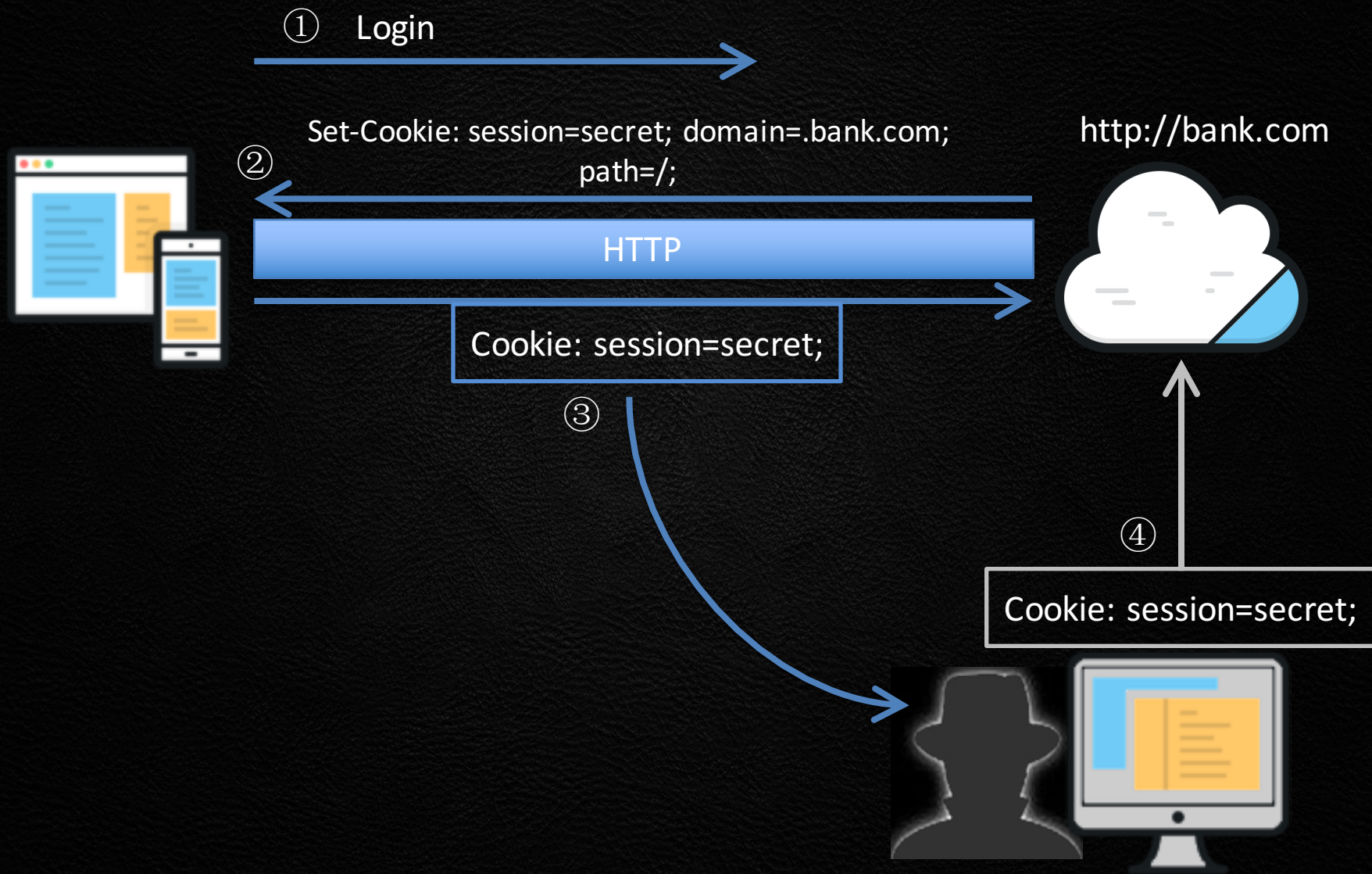
写时带属性，读时无属性

- 三元组

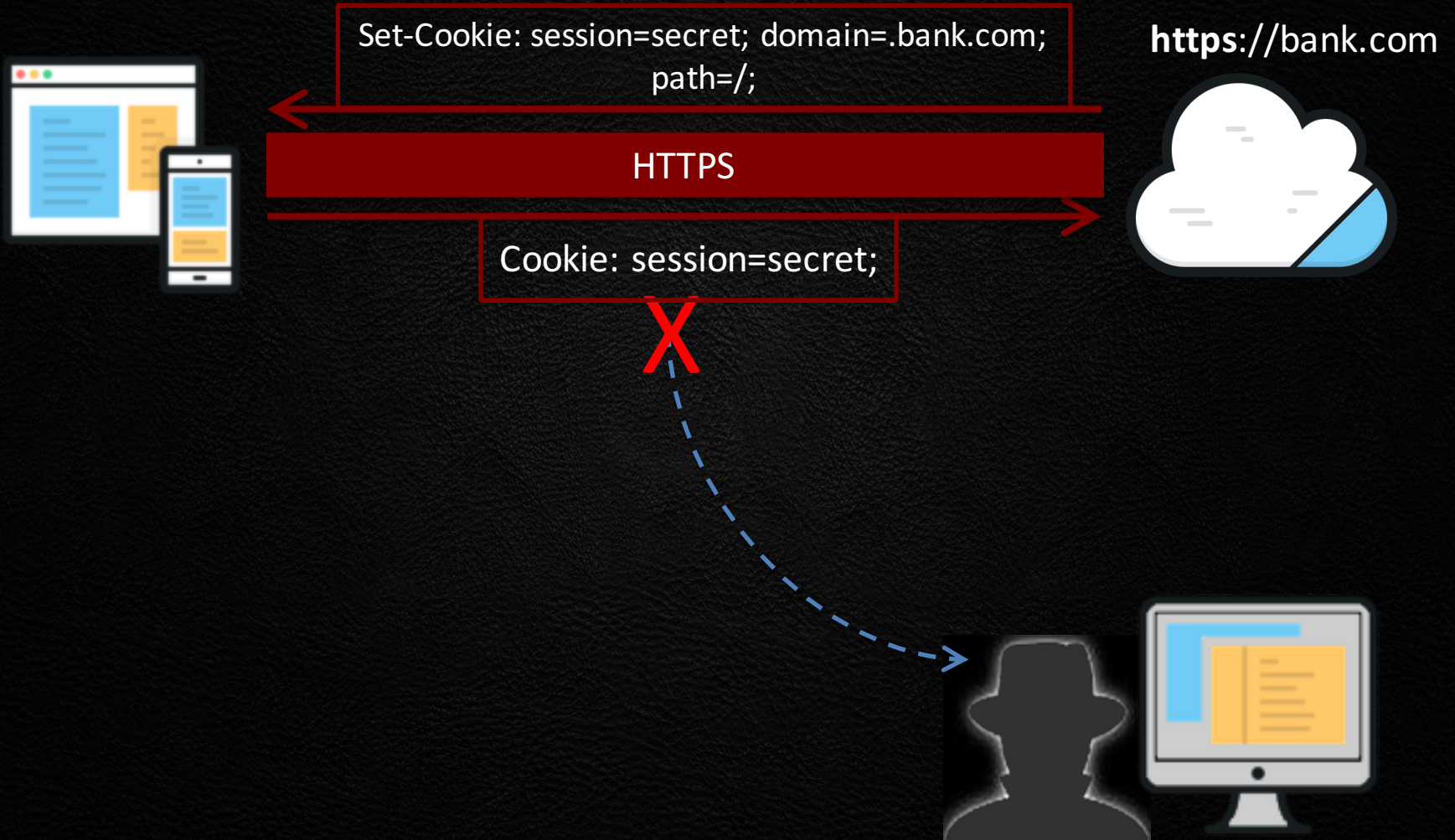
[name, domain, path]: 唯一确定Cookie

name, domain, path任一不同，则Cookie不同

泄露: HTTP



HTTPS



Cookie基础：同源策略 (SOP)

- Web SOP: [protocol, domain, port]

http://www.bank.com

http://www.bank.com:8080

https://www.bank.com

} 非同源 (受SOP隔离保护)

- Cookie SOP: [domain, path]

- 仅以domain/path作为同源限制
- 不区分端口
- 不区分**HTTP / HTTPS**

Cookie: session=secret; domain=.bank.com; path=/;

http://bank.com

https://bank.com

Cookie基础：Domain向上通配

- 在对Cookie读写时，以“通配”的方式判断Domain是否有效

写入：

当页面为 `http://www.bank.com` 时：

<code>Set-Cookie: user1=aaa; domain=.bank.com; path=/;</code>	接受
<code>Set-Cookie: user2=bbb; domain=www.bank.com; path=/;</code>	接受
<code>Set-Cookie: user3=ccc; domain=.www.bank.com; path=/;</code>	接受
<code>Set-Cookie: user4=ddd; domain=other.bank.com; path=/;</code>	拒绝

读取：

访问 `http://www.bank.com`

`Cookie: user1=aaa; user2=bbb; user3=ccc;`

访问 `http://user.bank.com`

`Cookie: user1=aaa;`

Cookie基础: Path向下通配

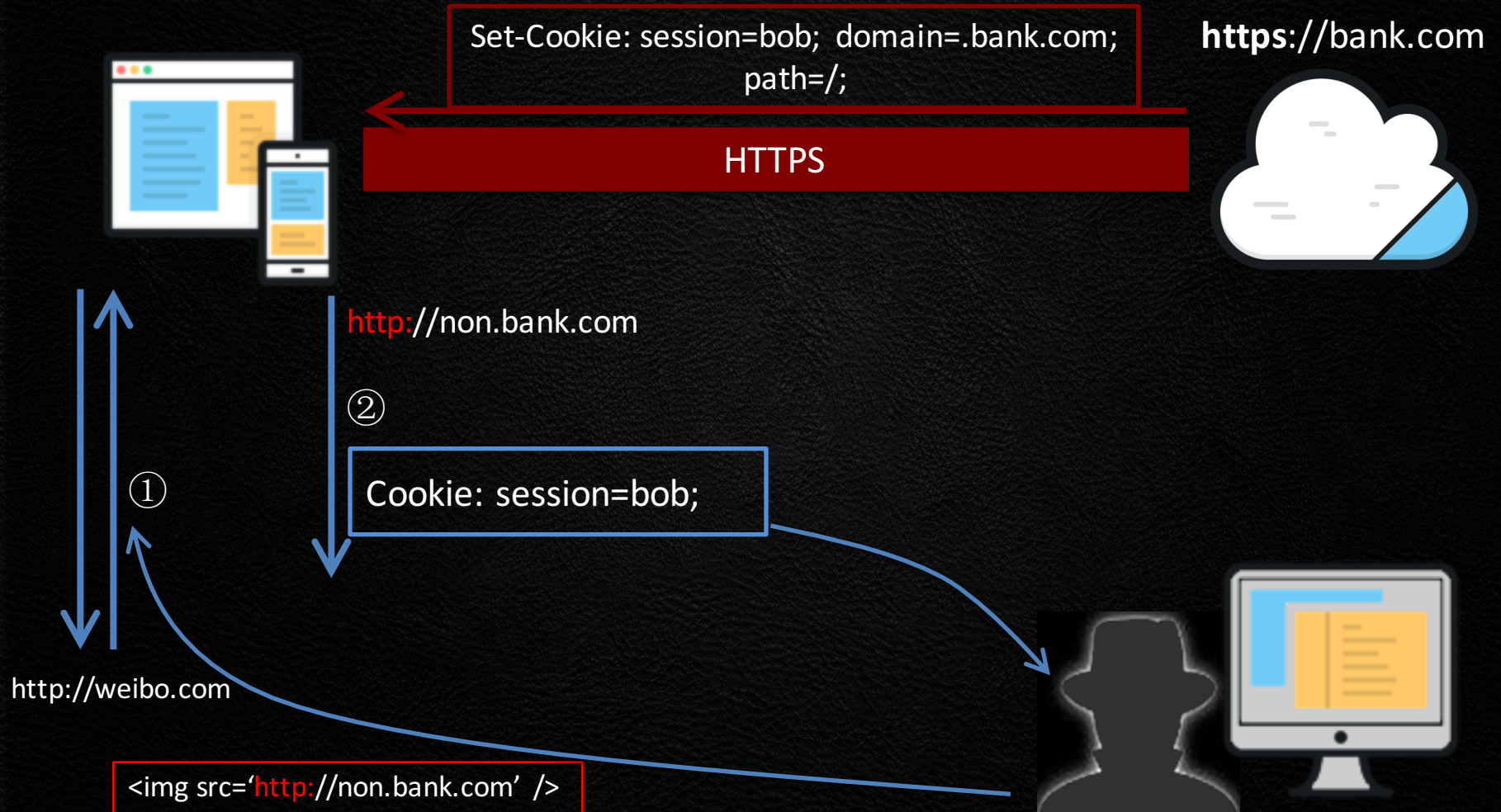
```
Set-Cookie: session=bob; domain=.bank.com; path=/;
```

```
Set-Cookie: cart=books; domain=.bank.com; path=/buy/;
```

```
http://bank.com/  
Cookie: session=bob;
```

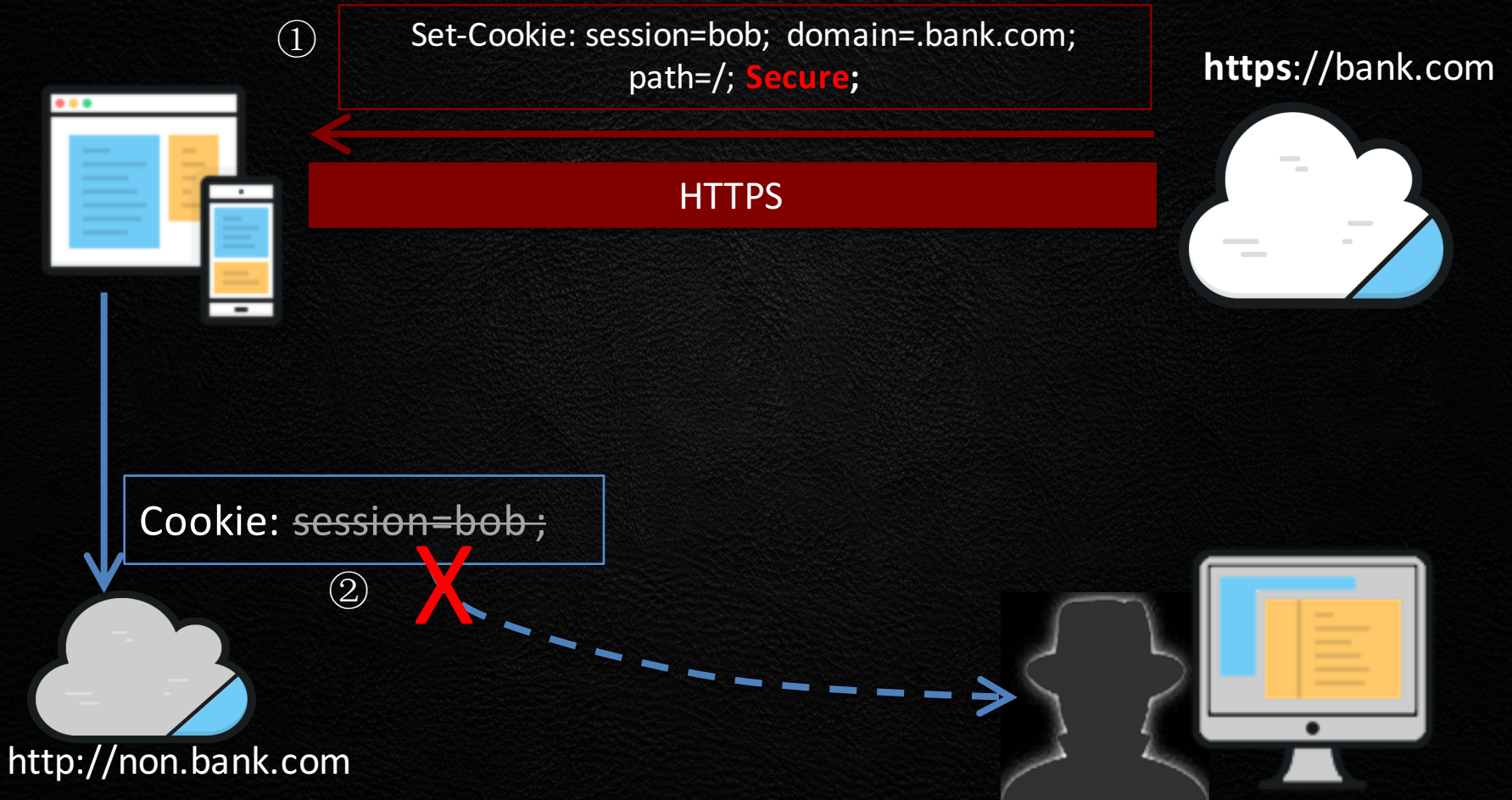
```
http://bank.com/buy/  
Cookie: session=bob; cart=books;
```

泄露: Cookie in HTTPS



Secure Flag

RFC: 带有Secure属性的Cookie仅能在HTTPS会话中传输



Secure Flag: 缺乏完整性保护

RFC 6265:

Although seemingly useful for protecting cookies from active network attackers, the Secure attribute protects only the cookie's confidentiality.

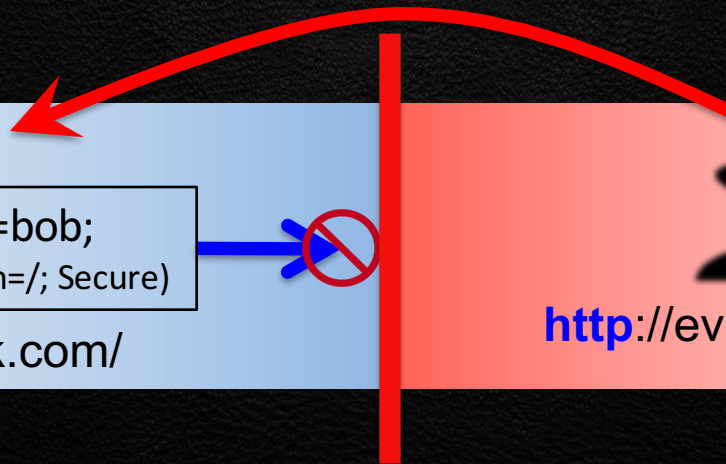
*An active network attacker can **overwrite** Secure cookies from an insecure channel, disrupting their integrity.*

```
Set-Cookie: session=attacker; domain=.bank.com; path=/;
```

```
Cookie: session=bob;  
(domain=.bank.com; path=/; Secure)
```

<https://good.bank.com/>

<http://evil.bank.com/>



Secure Cookie覆盖



Cookie注入: *Authenticated-as-Attacker*

The image shows a sequence of browser windows illustrating a cookie injection attack. The top window shows a search for 'kcon' on Google. The middle window shows the search results for 'kcon'. The bottom window shows the 'Google 网络与应用' history page, which lists search records for '蓝莲花' and 'kcon'. A red box highlights the '走马' (Cookie Injection) button in the browser's address bar. A red arrow points from a silhouette of a hacker to this button. The history page shows search records for '蓝莲花' and 'kcon'.

时间	内容	时间
<input type="checkbox"/>	今天	某些内容可能不会立即显示
<input type="checkbox"/>	搜索 蓝莲花	下午4:37
<input type="checkbox"/>	搜索 kcon	下午4:37

Cookie注入：反射

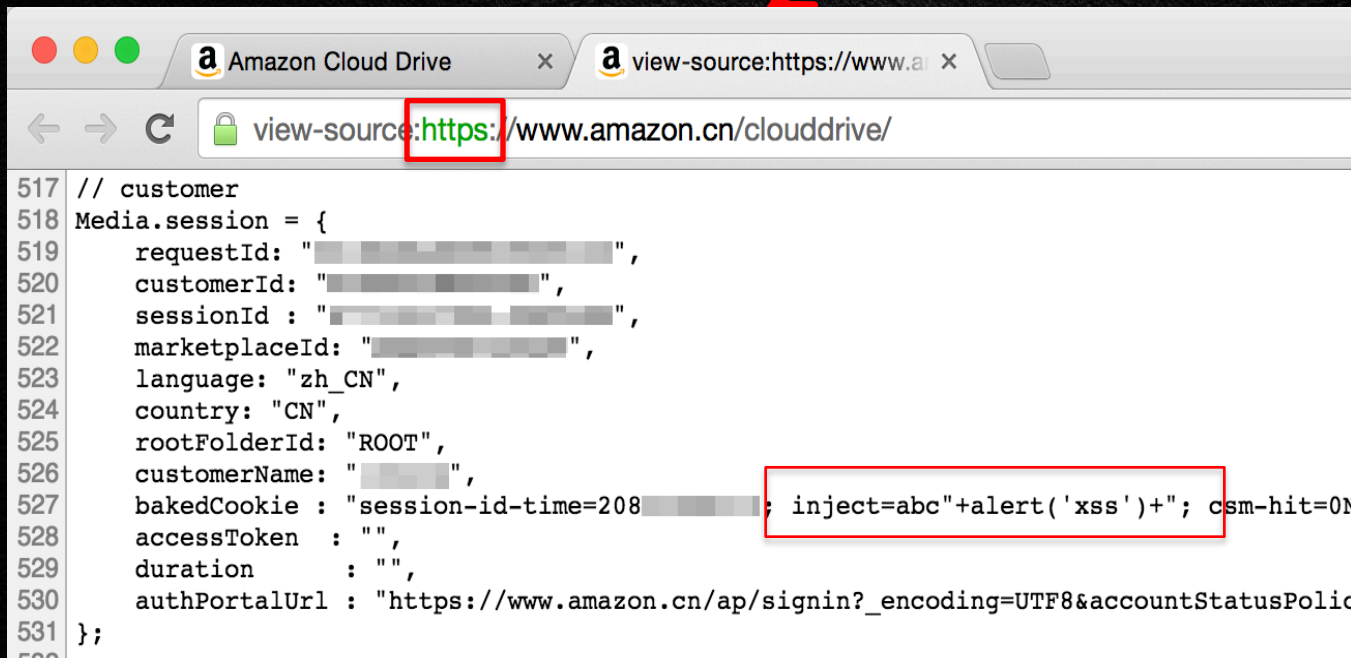
广义的反射：

- 服务端将Cookie拼接到HTML页面
- JS将Cookie渲染到DOM/参与运算

Cookie反射: XSS

```
Set-Cookie: inject=abc"+alert('xss')+";  
domain=.amazon.cn; path=/;
```

Amazon Cloud



```
517 // customer  
518 Media.session = {  
519   requestId: "████████████████████",  
520   customerId: "████████████████████",  
521   sessionId : "████████████████████",  
522   marketplaceId: "████████████████████",  
523   language: "zh_CN",  
524   country: "CN",  
525   rootFolderId: "ROOT",  
526   customerName: "████████████████████",  
527   bakedCookie : "session-id-time=208████████████████████; inject=abc"+alert('xss')+"; csm-hit=0N",  
528   accessToken : "",  
529   duration : "",  
530   authPortalUrl : "https://www.amazon.cn/ap/signin?_encoding=UTF8&accountStatusPolic",  
531 };  
532
```



个案?

Cookie反射（服务端）

	网站	反射		网站	反射
1	google.com	Y	16	weibo.com	Y
2	facebook.com		17	blogspot.com	
3	youtube.com		18	tmall.com	Y
4	yahoo.com	Y	19	sohu.com	
5	baidu.com	Y	20	yahoo.co.jp	
6	wikipedia.org		21	yandex.ru	Y
7	twitter.com		22	vk.com	
8	qq.com	Y	23	bing.com	Y
9	taobao.com	Y	24	wordpress.com	Y
10	amazon.com	Y	25	google.de	Y
11	linkedin.com	Y	26	pinterest.com	
12	live.com	Y	27	ebay.com	Y
13	google.co.in	Y	28	360.cn	Y
14	sina.com.cn	Y	29	google.co.uk	Y
15	hao123.com	Y	30	instagram.com	Y

非常普遍！

惯性思维1: “可信”的Cookie

- 谁输入的Cookie? 服务端 or 第三方
惯性: 服务器
- 检测自信、过滤简陋
BOA的实例:

反射: Cookie: BA_0021=OLB

```
boaMboxCreate("****", '****', '****', '****', 'profile.BA_0021=OLB', '****', '****', '****');
```

过滤: Cookie: BA_0021=OLB'xss

```
boaMboxCreate("****", '****', '****', '****', 'profile.BA_0021=OLB#xss', '****', '****', '****');
```

运算:

```
function boaMboxCreate() {  
    var argStr = process(arguments); //将参数处理为一个字符串  
    eval("mboxCreate(" + argStr + ")");  
}
```

绕过: Set-Cookie: BA_0021=OLB\x27+alert(1)+\x27; domain=.bankofamerica.com; path=/;

惯性思维2：“唯一”的Cookie

- 惯性：键值对

Cookie的键是什么？ name？

- Cookie由[name, domain, path]三元组唯一确定

[name, domain, path]才是Cookie的键

写时带属性，读时无属性

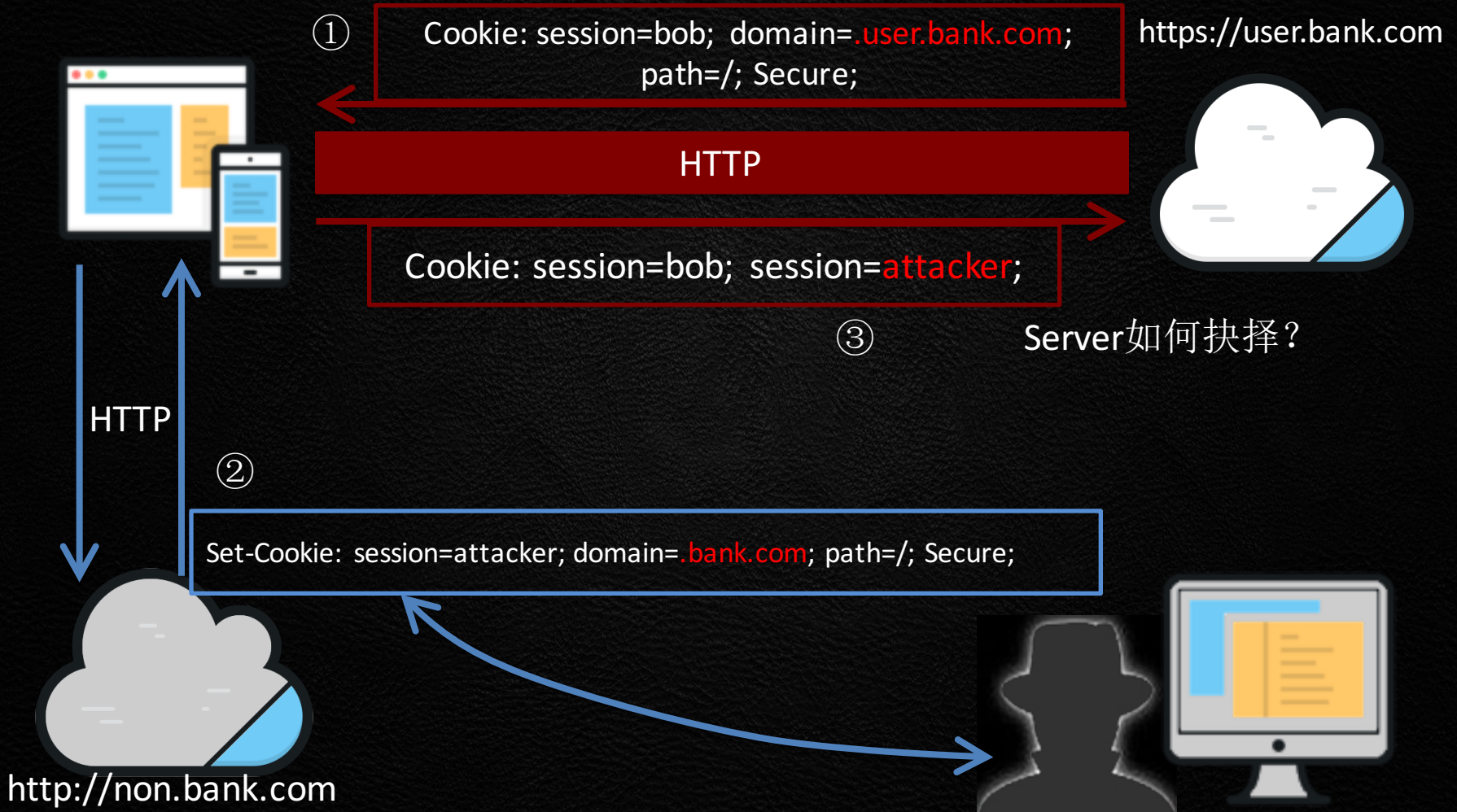
读取/发送时：

JS或Server只能看到name

domain、path由浏览器根据当前URL选择

允许重名

重名：不唯一的Cookie



重名：标准

重名Cookie如何处理？

RFC“标准”这样解释：

如果Cookie头中存在两个同名Cookie，服务器*不应该*根据它们出现的先后顺序来决定谁有效。

——RFC6265

即，实际没有标准！！！！

诡异的规范

如果Cookie头中存在两个同名Cookie，服务器不应该根据它们出现的先后顺序来决定谁有效。

——RFC6265

```
Cookie: session=bob; session=attacker;
```

写时带属性、读时无属性
除了先后顺序，没有其他区别

潜台词：

“我也没辙，那就约定俗成，按顺序来处理吧”

优先级顺序：Server & Browser

Server语言/框架	优先级
PHP	取前者
ASP/ASP.NET	
Java/Spring	
NodeJS	
GoLang	
Python	取后者

JS库/框架	优先级
JQuery	取前者
AngularJS	
ExtJS	
Dojo	
YUI	取后者 额外提供取前者的接口

重名：顺序/优先级

浏览器对Cookie String的排序原则

- 具有**更长Path**的Cookie更靠前；
- 如果Path长度相等，**更早创建**的Cookie更靠前；

——RFC6265

```
Cookie: session=bob; session=attacker;
```

提高优先级： 更长Path

目标页面： <https://user.bank.com/admin/index.php?type=1>

Cookie: session=**attacker**; session=bob;

Server → Set-Cookie: session=bob; domain=.user.bank.com; path=/
Attacker → Set-Cookie: session=attacker; domain=.bank.com; path=**/admin**;

Server → Set-Cookie: session=bob; domain=.user.bank.com; path=/admin;
Attacker → Set-Cookie: session=attacker; domain=.bank.com; path=**/admin**;

总能最长？ Cookie职责之一，负责多页面之间的状态传递

Server → Set-Cookie: session=bob; domain=.user.bank.com; path=/admin/;
Attacker → Set-Cookie: session=attacker; domain=.bank.com; path=**/admin/index.php**;

Attacker → Set-Cookie: session=attacker; domain=.bank.com; path=**/admin/index.php?type=1**; (Firefox)

提高优先级： 更早创建

目标页面： <https://user.bank.com/>

```
Server → Set-Cookie: session=bob; domain=.bank.com; path=/;
```

```
Attacker → Set-Cookie: session=none; domain=.bank.com; path=/; expires=Mon, 1 Jan 1970
```

```
Attacker → Set-Cookie: session=attacker; domain=user.bank.com; path=/;
```

...

```
Server → Set-Cookie: session=bob; domain=.bank.com; path=/;
```

```
Cookie: session=attacker; session=bob;
```

更早创建

```
Attacker → Set-Cookie: session=attacker; domain=user.bank.com; path=//; (Firefox)
```

精确控制

- 精确控制作用域
domain、 path
- 总能构造更高优先级
Path、 Creation-time

那么...

精确攻击：隐蔽的身份替换

对google注入：Set-Cookie: session=attacker; domain=**www.google.com**; path=**/search**;

Ajax: <https://www.google.com/search?pq=kcon>

身份：攻击者（信息泄露）

Cookie: **session=attacker**; session=bob;



<https://www.google.com/>

身份：受害者（无察觉）

Cookie: session=bob;



<https://history.google.com/history/>

身份：受害者（无察觉）

<https://mail.google.com/>
<https://drive.google.com/>

身份：受害者（无察觉）

惯性思维3：“一个”HTTPS页面

<https://www.bank.com>

欢迎, Bob

系统消息

好友列表

我的订单

浏览记录

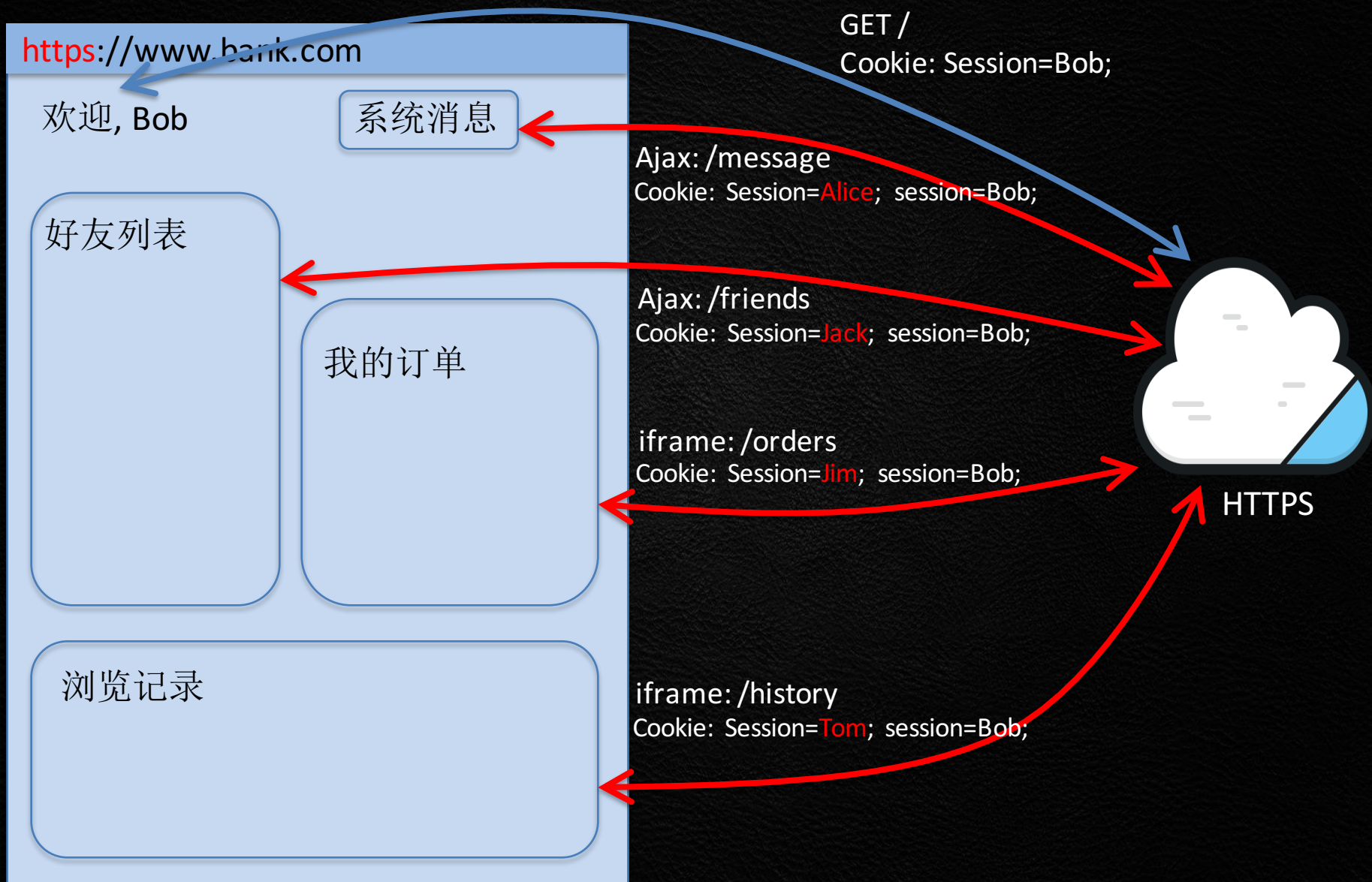
这是一个HTTPS页面；

HTTPS可以保证页面的完整性；

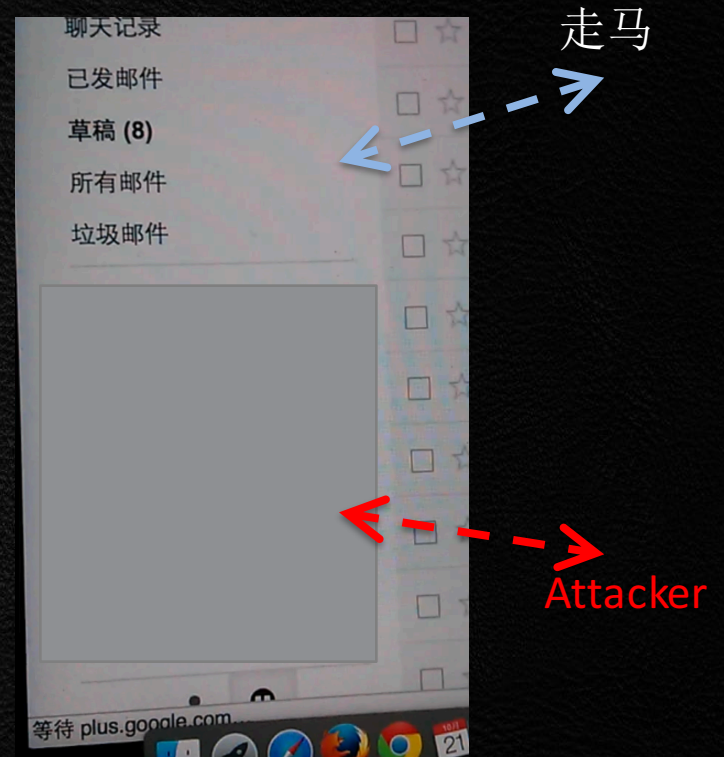
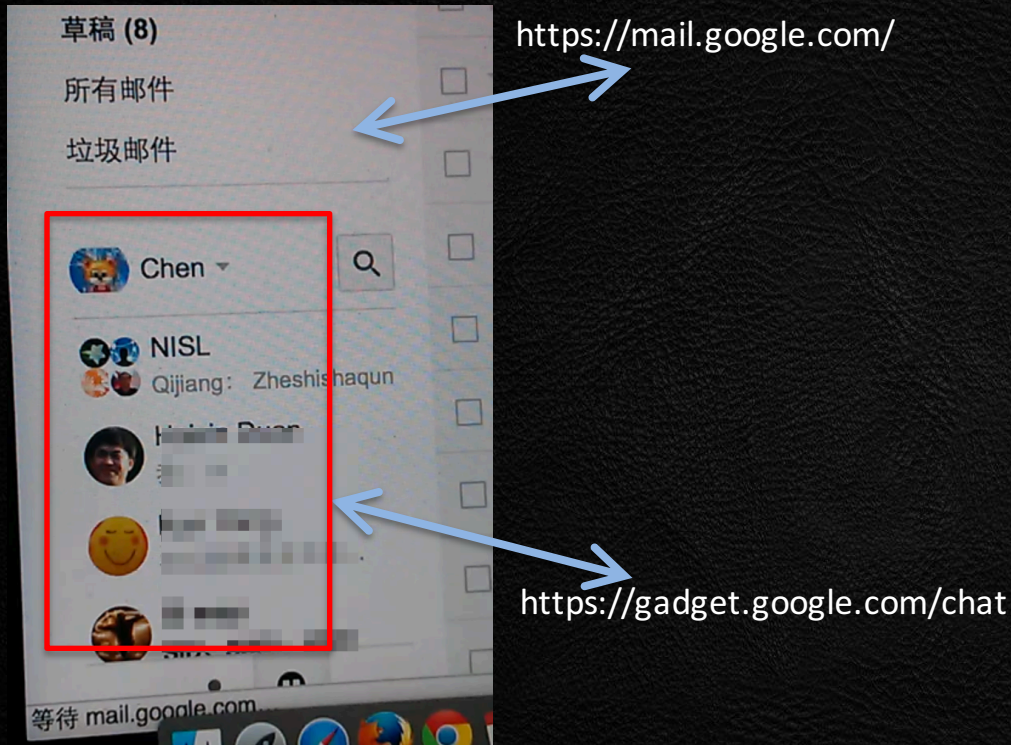
因此，这一个页面的信息是完整的。
(不可篡改的)

真是“一个”页面吗？

“一个”HTTPS页面



精确替换攻击: gmail



Set-Cookie: session=attacker; domain=gadget.google.com; path=/chat;

精确替换攻击：充值



走马

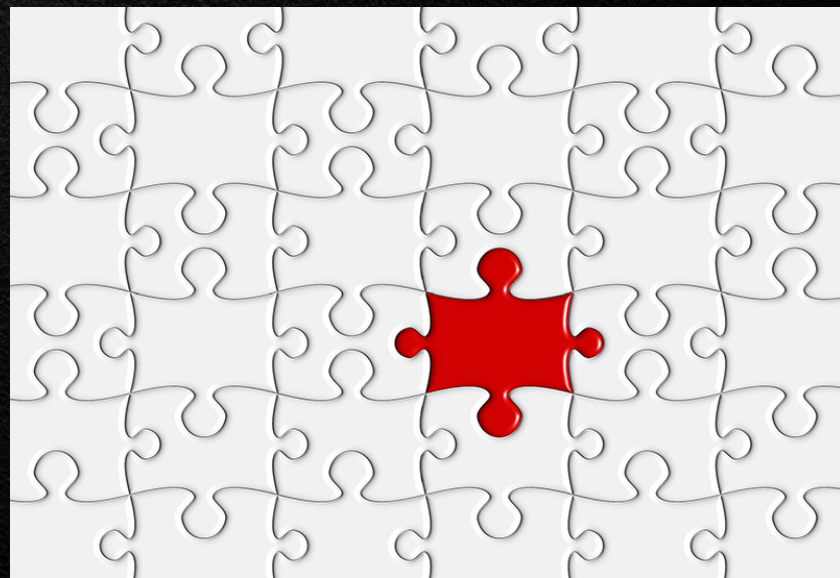
Attacker

```
passport.jd.com/new/helloService.ashx?m=ls&callback=jsonp1440239320356&_=1440239320428 (1 match)  
1 | jsonp1440239320356({"info":{"<a href="\http://\home.jd.com\" target=\"_blank\" class=\"link-user\">走马 </a>
```

```
Set-Cookie: session=attacker; domain=pay.jd.com; path=/payment/bankChoose_Common.action;
```

正确看待HTTPS页面

- HTTPS页面往往并非“一个”页面
- 往往由多个子页面以及多个Ajax拼凑而成
- 攻击者利用Cookie可对其进行篡改



惯性思维4：“一次”HTTPS操作



“一次”HTTPS操作

您已选择的支付方式:




中国建设银行
China Construction Bank

单笔限额	每日限额	
1000	1000	
5万	10万	
100000	100000	US

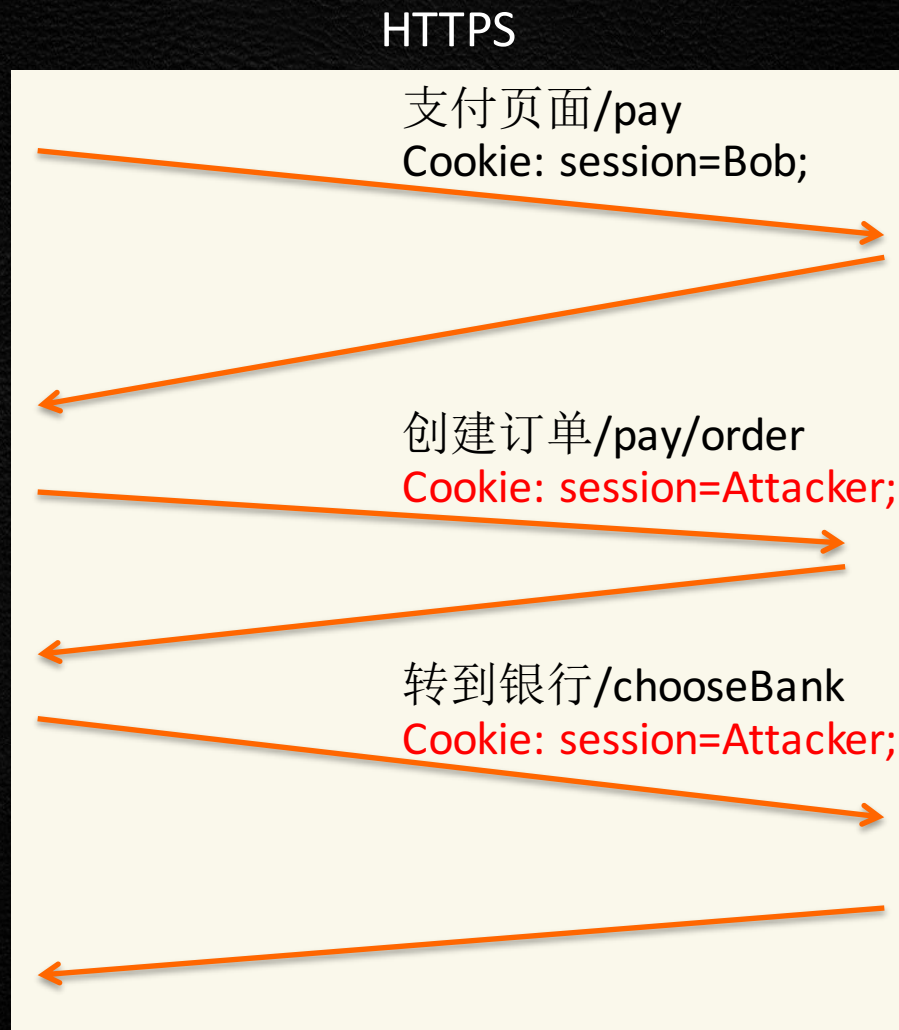
立即支付

自动跳转

自动跳转

 **充值成功**

充值单号: 32 [redacted]



“一次”HTTPS操作：识别链断裂

国内网银支付现状

The screenshot shows a web browser window with two tabs: '支付宝 - 网上支付 安全快速' and '中国建设银行 个人网上银行'. The address bar displays the URL: `https://ibsbjstar.ccb.com.cn/app/ccbMainAliPayPlatV5`. The page header includes the CCB logo, '中国建设银行 China Construction Bank', and '个人网银支付'. A customer service hotline (95533) and the website (WWW.CCB.COM) are also visible. The main content area is divided into two sections: '我的订单' (My Order) and '网上银行支付' (Online Bank Payment). The '我的订单' section lists order details: '商户名称: 支付宝(中国)网络技...', '订单号: 2015...', '订单金额: 1.00', '支付币种: 人民币', '商户分行: 浙江省建行', and '支付日期: 2015-08-22'. The '网上银行支付' section has two tabs: '网上银行支付' and '账号支付'. The '账号支付' tab is active, showing instructions for account payment and input fields for '支付账号' (Payment Account) and '* 附加码' (Additional Code). A CAPTCHA image with the text 'wwvp9' is displayed next to the additional code field. A blue button labeled '下一步' (Next Step) is at the bottom. A red box highlights the merchant name and order number in the '我的订单' section, with a red text annotation '断开的“识别链”' (Broken 'Identification Chain') pointing to it.

题外话：国内许多电商跳转页面为HTTPS...

正确看待HTTPS操作流程

- 用户操作可能并非一次“原子”操作
- 往往由多个Ajax以及多次自动跳转请求组成
- Cookie可对中间请求进行身份篡改



Amazon恶意购物
UnionPay银行卡绑定
JD恶意充值
Facebook支付绑定
Bitbucket OAuth

.....

惯性思维5： 总能清理的Cookie

- 服务端总能主动地、准确地清理Cookie吗？

写时带属性，读时无属性

Browser → Cookie: user=bob;

服务器无法得知Cookie的具体domain/path

如果希望删除或重新赋值...

Set-Cookie: user=bob; domain=?; path=?; ← Server

无法确定domain与path

信息丢失，难以准确地清理！

Github认为...

```
GitHub, Inc. [US] https://github.com/blog/1466-yummy-cookies-across-domains
The solution is pretty straightforward, albeit rather inelegant: for any given request URL, the
web browser would only send a malicious JavaScript cookie if its Path matches partially the
path of the request URL. Hence, we only need to attempt to drop the cookie once in each
component of the path:

HTTP/1.1 302 Found
Location: /libgit2/libgit2/pull/1457
Content-Type: text/html
Set-Cookie: _session=; Expires=Thu, 01-Jan-1970 00:00:01 GMT; Path=/; Domain=.github.com;
Set-Cookie: _session=; Expires=Thu, 01-Jan-1970 00:00:01 GMT; Path=/libgit2; Domain=.github
Set-Cookie: _session=; Expires=Thu, 01-Jan-1970 00:00:01 GMT; Path=/libgit2/libgit2; Domain
Set-Cookie: _session=; Expires=Thu, 01-Jan-1970 00:00:01 GMT; Path=/libgit2/libgit2/pull; D
Set-Cookie: _session=; Expires=Thu, 01-Jan-1970 00:00:01 GMT; Path=/libgit2/libgit2/pull/14
```

遍历嘛！总能清除异常的Cookie

但是，遍历的开销.....

<https://sub.domain.bank.com/admin/users/list.php?name=1#any>

Domains:

- .sub.domain.bank.com
- sub.domain.bank.com
- .domain.bank.com
- .bank.com

Paths:

- /
- /admin
- /admin/
- /ad (IE/Safari)
- /admin/**us** (IE/Safari)
- /admin/users/list.php?name=1#**any** (Firefox)

驻留式攻击：跨越时间和空间

几周前...



Set-Cookie: session=attacker; domain=.pay.jd.com; path=/payment/;



几周后...



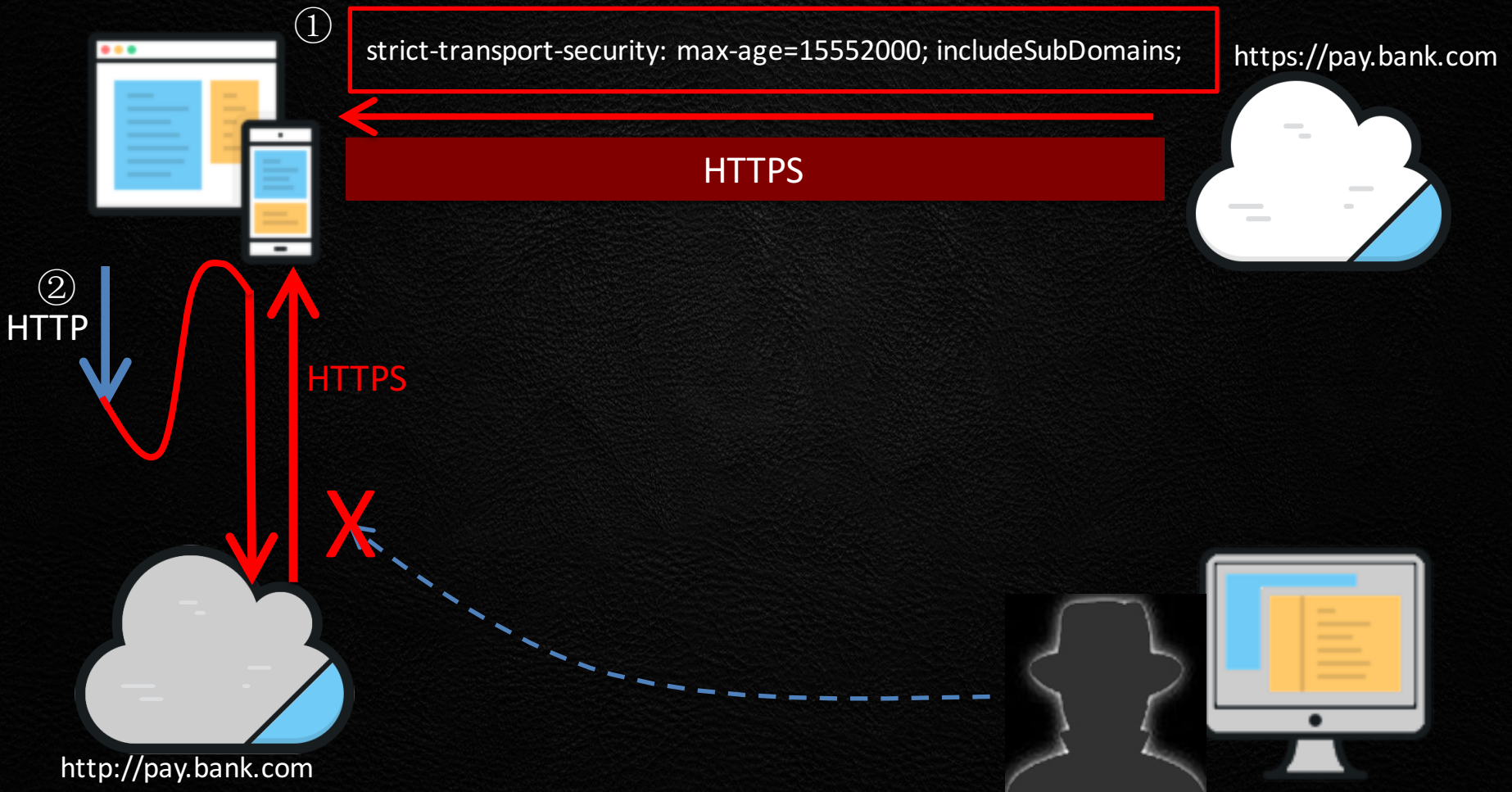
<https://pay.jd.com/payment/>



该HSTS登场了

看上去强悍的HSTS

HSTS: 浏览器对特定域名**强制**进行HTTPS访问



HSTS: 面对Cookie的尴尬

- HSTS并非为Cookie量身定制

对于Web页面

对敏感子域进行includeSubdomains设置即可；

例如gmail，

只需要在mail.google.com进行includeSubdomains

对于Cookie

由于Cookie的Domain是通配的

例如gmail，

虽然对mail.google.com进行includeSubdomains

攻击者即可伪造non.google.com，注入domain为.google.com的Cookie

在mail.google.com有效，可进行攻击

必须对整个域名标注includeSubdomains (Full HSTS)

- 部署现状

Full HSTS: 8/1000; 1252/1000000 (Alexa)

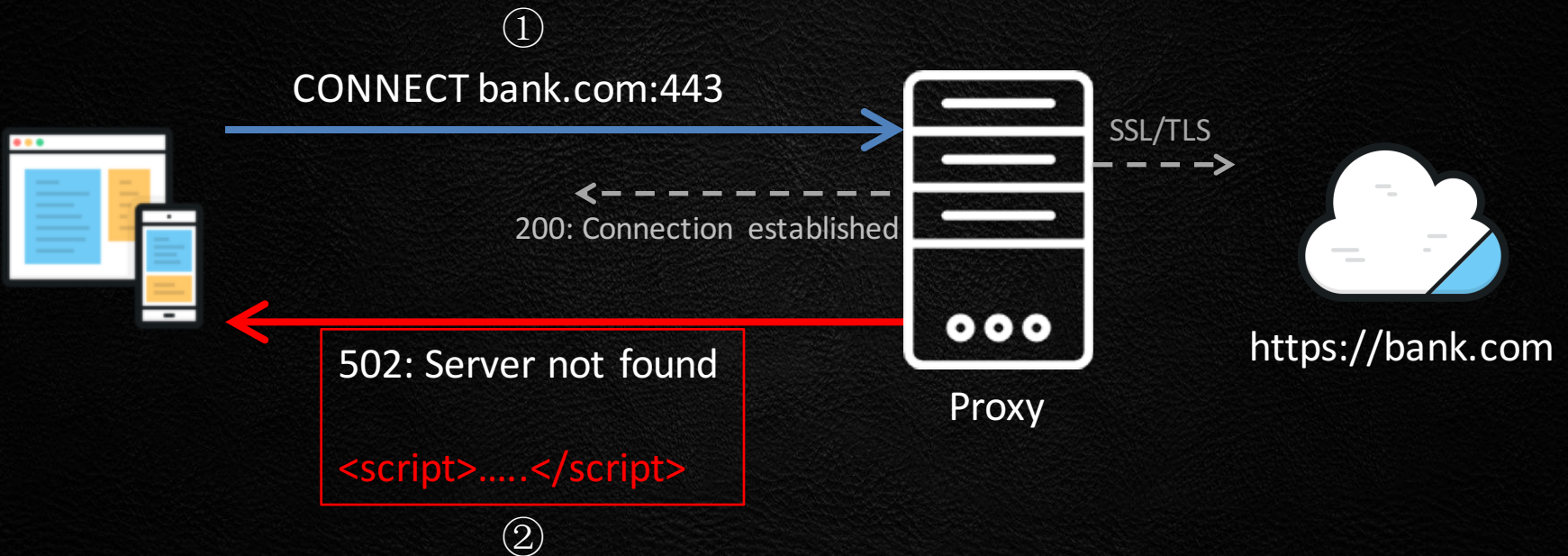
IE11才开始支持

假设Full HSTS，又如何？

407注入攻击: Pretty Bad Proxy

Pretty-Bad-Proxy: An Overlooked Adversary in Browsers' HTTPS Deployments

Shuo Chen, Ziqing Mao

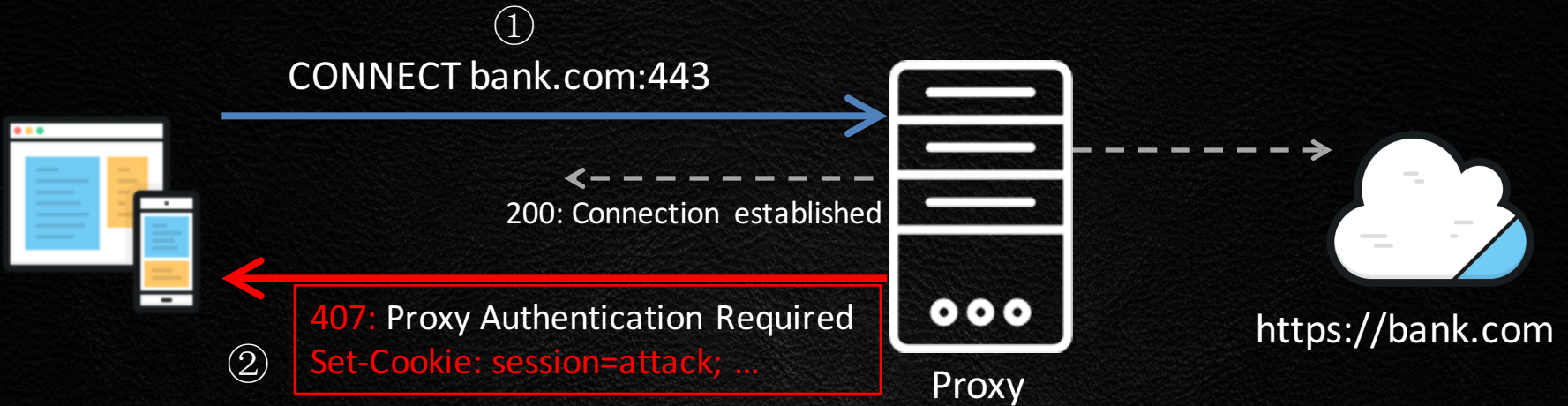


Fixed

407注入攻击: Firefox

```
1225     if (mTransaction->ProxyConnectFailed()) {
1226         // Only allow 407 (authentication required) to continue
1227         if (httpStatus != 407)
1228             return ProcessFailedProxyConnect(httpStatus);
1229         // If proxy CONNECT response needs to complete, wait to process connection
1230         // for Strict-Transport-Security.
1231     } else {
1232         // Given a successful connection, process any STS data that's relevant.
1233         rv = ProcessSTSHeader();
1234         MOZ_ASSERT(NS_SUCCEEDED(rv), "ProcessSTSHeader failed, continuing load.");
1235     }
1236
1237     MOZ_ASSERT(!mCachedContentIsValid);
1238
1239     ProcessSSLInformation();
1240
1241     // notify "http-on-examine-response" observers
1242     gHttpHandler->OnExamineResponse(this);
1243     SetCookie(mResponseHead->PeekHeader(nsHttp::Set_Cookie));
1244 }
```

407注入攻击



	Windows	Mac OS	Linux	Android	iOS
IE	-	N/A	N/A	N/A	N/A
Chrome	①	①	①	①	①
Firefox	②	②	②	②	N/A
Safari	②	③	N/A	N/A	②
Opera	①	①	N/A	①	N/A

- ①: cookie injection with pop-up window.
- ②: cookie injection without pop-up window.
- ③: cookie injection and script injection.

CVE-2014-8639
CVE-2015-1229

Full HSTS下仍可注入!

惯性思维6

Cookie而已

通用攻击

有无可能：攻击框架/商业Web软件？

假设WordPress的wp_vul_path/vul.php 存在Cookie漏洞

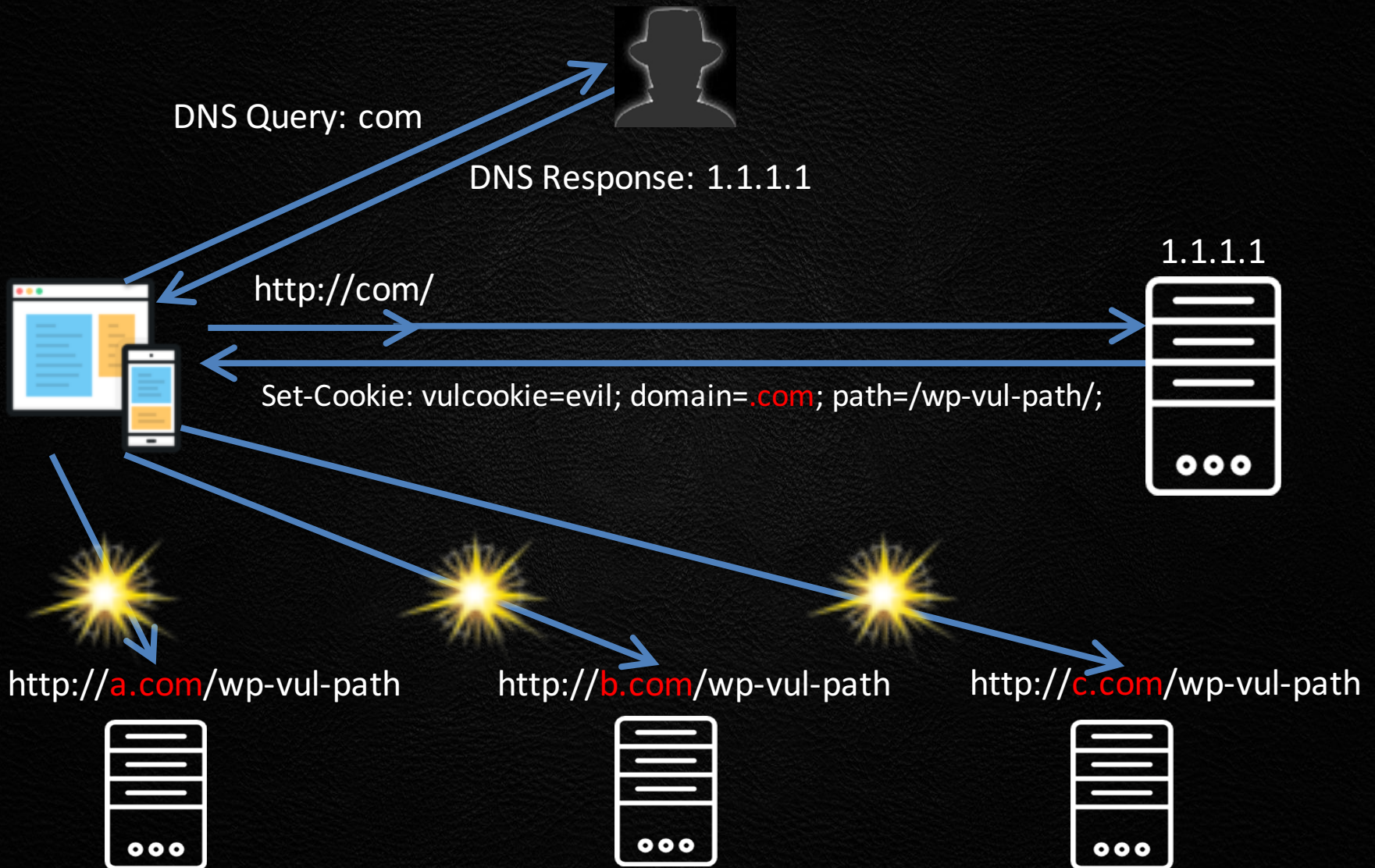
http://a.com/wp_vul_path/vul.php

http://b.com/wp_vul_path/vul.php

http://c.com/wp_vul_path/vul.php



通用攻击: Blind Cookie Attack (Safari)



再谈反射: Cookie BREACH

- Cookie BREACH
phpMyAdmin(CVE-2015-2206)



Cookie之困，困于

- 协议本身
 - 宽松的SOP、缺乏完整性约束
- 浏览器实现
 - 千奇百怪
- Cookie不可信
 - 检查过滤缺乏严谨，XSS/SQLi
- Cookie不唯一
 - 身份替换
- Cookie拆解“一个”HTTPS页面
 - 页面局部劫持
- Cookie拆解“一次”HTTPS操作
 - 业务流程劫持
- Cookie难以被Server清理
 - 持久化攻击
- 并非Cookie而已
 - 与其他攻击形式结合，BREACH/DNS Binding



离开星巴克时需要做什么？

谢谢大家

温馨提示

正在连WIFI的童鞋，会后请务必清理电脑或手机Cookie 😊