一文学会 Web Service漏洞挖掘!

原创 li'mu 黑白天 昨天

00×01 什么是Web Service

Web Service是一个平台独立的，低耦合的，自包含的、基于可编程的web的应用程序，可使用开放的XML（标准通用标记语言下的一个子集）标准来描述、发布、发现、协调和配置这些应用程序，用于开发分布式的交互操作的应用程序。

Web Service技术， 能使得运行在不同机器上的不同应用无**须借助附加的**、**专门的第三方软件**或**硬件**， 就可相**互交换数据或集成**。依据**Web Service**规范实施的应用之间， 无论它们所使用的**语言**、 **平台**或**内部协议**是什么， 都**可以相互交换数据**。

简单的说，WebService就是一种跨编程语言和跨操作系统平台的远程调用技术。就是说服务端程序采用java编写，客户端程序则可以采用其他编程语言编写。跨操作系统平台则是指服务端程序和客户端程序可以在不同的操作系统上运行。

远程调用，就是一台计算机的应用可以调用其他计算机上的应用。例如：支付宝，支付宝并没有银行卡等数据，它只是去调用银行提供的接口来获得数据。还有天气预报等，也是气象局把自己的系统服务以webservice服务的形式暴露出来，让第三方网站和程序可以调用这些服务功能。

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。

00×02 Web Service基础

WebService采用Http协议来在客户端和服务端之间传输数据。WebService使用XML来封装数据，XML主要的优点在于它是跨平台的。

WebService通过HTTP协议发送请求和接收结果时，发送的请求内容和结果内容都采用XML格式封装，并增加了一些特定的HTTP消息头，以说明HTTP消息的内容格式，这些特定的HTTP消息头和XML内容格式就是SOAP协议规定的。

WebService服务器端首先要通过一个WSDL文件来说明自己有什么服务可以对外调用。WSDL就像是一个说明书，用于描述**WebService**及其方法、参数和返回值。WSDL文件保存在Web服务器上，通过一个url地址就可以访问到它。客户端要调用一个**WebService**服务之前，要知道该服务的WSDL文件的地址。**WebService**服务提供商可以通过两种方式来暴露它的WSDL文件地址：1.注册到UDDI服务器，以便被人查找；2.直接告诉给客户端调用者。
WebService交互的过程就是,WebService遵循SOAP协议通过XML封装数据，然后由Http协议来传输数据。

00×03 Web Service分类

SOAP

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。

是使用http发送的XML格式的数据，它可以跨平台，跨防火墙，SOAP不是webservice的专有协议。

可以理解为 SOAP = http+xml

SOAP结构

必需的 Envelope 元素，可把此 XML 文档标识为一条 SOAP 消息
可选的 Header 元素，包含头部信息
必需的 Body 元素，包含所有的调用和响应信息
可选的 Fault 元素，提供有关在处理此消息所发生错误的信息

REST

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。不需要WSDL,UDDI。

00×03 Web Service技术支持

XML

可扩展的标记语言（标准通用标记语言下的一个子集）是Web service平台中表示数据的基本格式。

XSD数据类型

Web service平台就是用XSD来作为其数据类型系统的。当你用某种语言(如VB. NET或C#)来构造一个Web service时，为了符合Web service标准，所有你使用的数据类型都必须被转换为XSD类型。

WSDL

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，说明服务端接口、方法、参数和返回值，WSDL是随服务发布成功，自动生成，无需编写。少数情况下，WSDL也可以用来描述REST型Web Service。SOAP也是基于XML（标准通用标记语言下的一个子集）和XSD的，XML是SOAP的数据编码方式。

文档结构

```
    </wsdl:operation>
  </wsdl:binding>
▼<wsdl:service name="Service">
  ▼<wsdl:port name="ServiceSoap" binding="tns:ServiceSoap">
      <soap:address location="http[redacted]/service.asmx"/>
    </wsdl:port>
  ▼<wsdl:port name="ServiceSoap12" binding="tns:ServiceSoap12">
      <soap12:address location="http:[redacted]18081/service.asmx"/>
    </wsdl:port>
  ▼<wsdl:port name="ServiceHttpPost" binding="tns:ServiceHttpPost">
      <http:address location="http:[redacted]18081/service.asmx"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

阅读方法

先看service标签，看相应port的binding属性，然后通过值查找上面的binding标签。

通过binding标签可以获得具体协议等信息，然后查看binding的type属性

通过binding的type属性，查找对应的portType，可以获得可操作的方法和参数、返回值等。

通过portType下的operation标签的message属性，可以向上查找message获取具体的数据参数信息

参数解析

Service：相关端口的集合，包括其关联的接口、操作、消息等。
Binding：特定端口类型的具体协议和数据格式规范
portType：服务端点，描述 web service可被执行的操作方法，以及相关的消息，通过binding指向portType
message：定义一个操作（方法）的数据参数
types：定义 web service 使用的全部数据类型

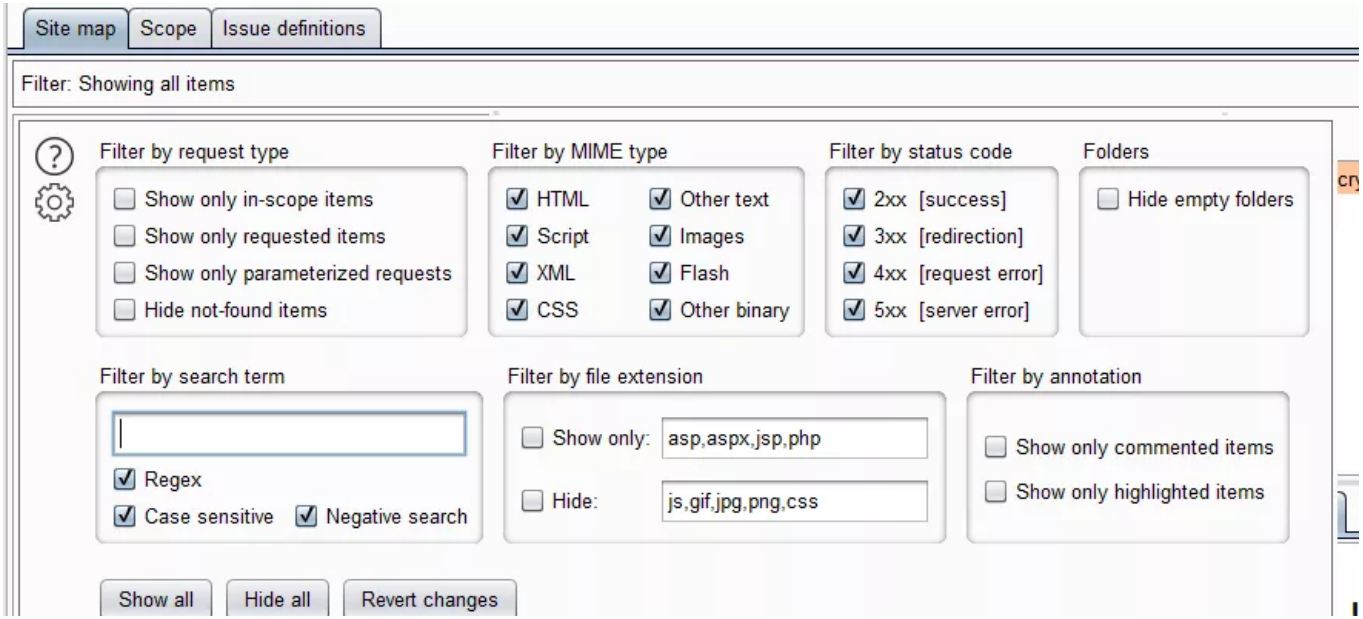
WSDL文档是从下往上阅读。

00×04 如何发现Web Service

使用burp等等代理软件，检查所捕获的数据。
使用Google语法。
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）。
爬取并解压swf、jar等类似文件。
模糊测试。

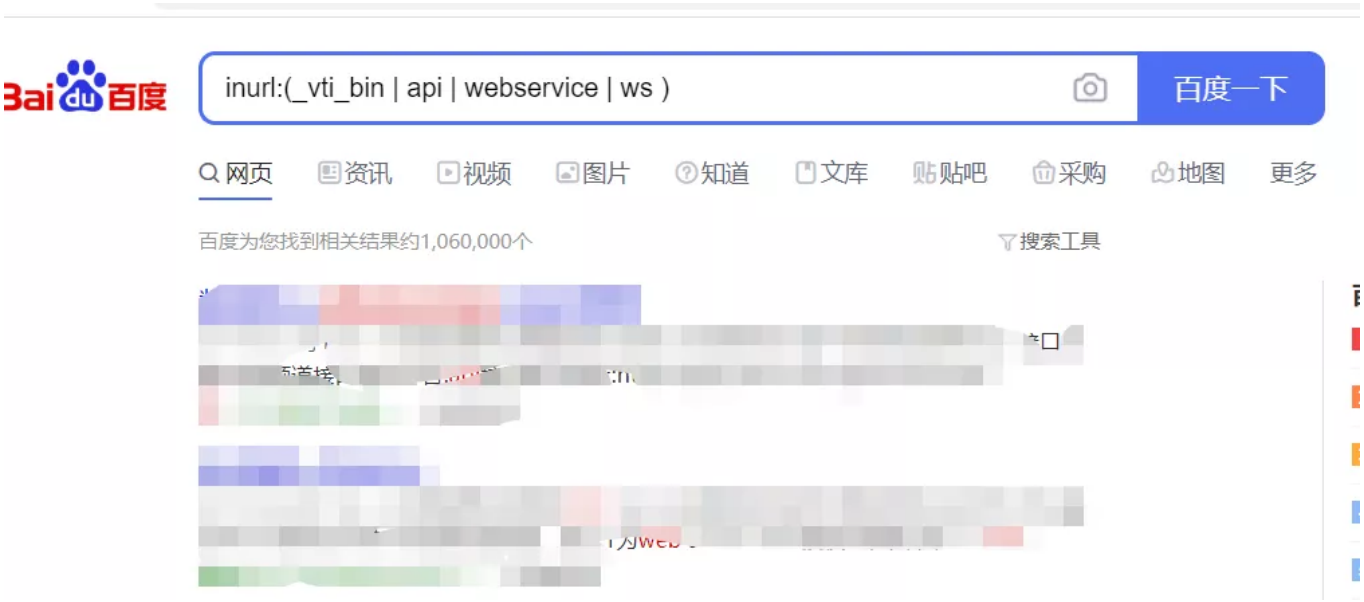
Eg：
使用burp等等代理软件，检查所捕获的数据。

在BurpSuite中设定的过滤规则，用来筛选抓包数据中的Web Service地址。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。



2.使用 Google语法。

```
inurl:(_vti_bin | api | webservice | ws )
```



00×04 Web Service渗透测试

很多人误以为 Web Service没有界面，黑客就无法进行攻击。事实上，Web service通常仅是对现有应用层功能进行了封装，其后台应用层代码如果存在安全漏洞，我们完全可以使用 Web service进行攻击。绝大多数情况下，我们可以通过查看WSDL 从而了解 Web Service可以提供的操作及 SOAP 消息格式，所以说，Web 中所面临的安全威胁同样存在于 Web Service中。

Web Service的漏洞分类可以分为2种：

1.Web 应用安全漏洞：

sql注入

xss攻击

命令执行

越权

LDAP注入

缓冲区溢出

逻辑漏洞

等等

2.XML 相关的特殊安全漏洞：

XPath注入

XQuery注入

拒绝服务攻击（SOAP 数组溢出、递归的 XML 实体声明、超大消息体）

信息泄漏（XML External Entity File Disclosure）

等等

sql注入（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。）

XPath注入

XPath 作为用来查询 XML 数据的语言，同样容易存在很多注入漏洞。某种程度来说，XPath 注入比 SQL 注入更简单，因为不同数据库产品的 SQL 语句有不同的语法，而 XPath 只有一个标准。我们假定某 Web 服务后台采用了这段代码来查询某 XML 数据文件中的记录。

存在注入漏洞的 XPath 查询

```
Stmt = "//users/user[username/text()='" + username+ "' and password/text()='" + password + "']/id/text()";
```

其中 username 和 password 是通过 SOAP 消息进行传输，如下文：

```
传递 XPath 查询参数的 SOAP 消息片段
<soap:Envelope xmlns:soap="">
  <soap:Body>
  <fn:PerformFunction xmlns:fn="">
  <fn:uid>testuser</fn:uid>
  <fn:password>testpassword</fn:password>
  </fn:PerformFunction>
  </soap:Body>
</soap:Envelope>
```

假如黑客利用 SOAP 传入 username="admin", password="' or '1'='1"，以上 XPath 查询就变为：

```
遭注入的 XPath 查询1Stmt="//users/user[username/text()='admin' and password/text()='' or '1'='1']/id/text()";
```

这样我们可以访问到 admin 用户信息。

拒绝服务攻击

由于 Web 服务基于 XML 格式的协议进行通信（例如 SOAP 消息）。当 SOAP 消息到达 Web 服务器段时，服务器端会调用 XML Parser 解析 XML 数据（包括 DTD 声明），黑客可以利用大量的超大消息体或者递归的 XML 实体声明，让服务器端长时间解析 XML 数据，直至服务器资源耗竭，从而形成拒绝访问攻击，导致 Web 服务停止服务。

例如，SOAP 消息中可以加入以下大量无意义的实体声明，导致 SOAP 消息解析缓慢。

SOAP 消息中无意义的实体声明示例

```
<!DOCTYPE root [
    <!ENTITY ha "Ha !">
    <!ENTITY ha2 "&ha; &ha;">
    <!ENTITY ha3 "&ha2; &ha2;">
    ...
    <!ENTITY ha127 "&ha126; &ha126;">
    <!ENTITY ha128 "&ha127; &ha127;">
]>
```

信息泄漏

某些 Web 服务会返回客户端指定的资源信息时，如果服务器端防范不当，则可能存在信息泄漏隐患。举个简单的类似 HelloWorld 的例子，假设某个 Web 服务会接受用户传来的名字“Jeremy”，然后返回“Hello, Jeremy!”。但，如果黑客传入如下参数：

SOAP 消息中声明外部文件引用

```
<!DOCTYPE root [
    <!ENTITY myfile SYSTEM "file://c:/windows/win.ini">
]>
    ...
    <name>&myfile;</name>
```

服务器端如果疏于参数校验及文件访问权限控制，该 Web 服务可能返回系统文件的内容。

00×05 使用soap ui+burp对Web Service渗透测试

我们可以对Web Service方法的具体参数进行Fuzz测试，挖掘其中存在的各种技术漏洞和逻辑漏洞。也可以使用一些专业工具对常见的Web Service进行渗透测试。

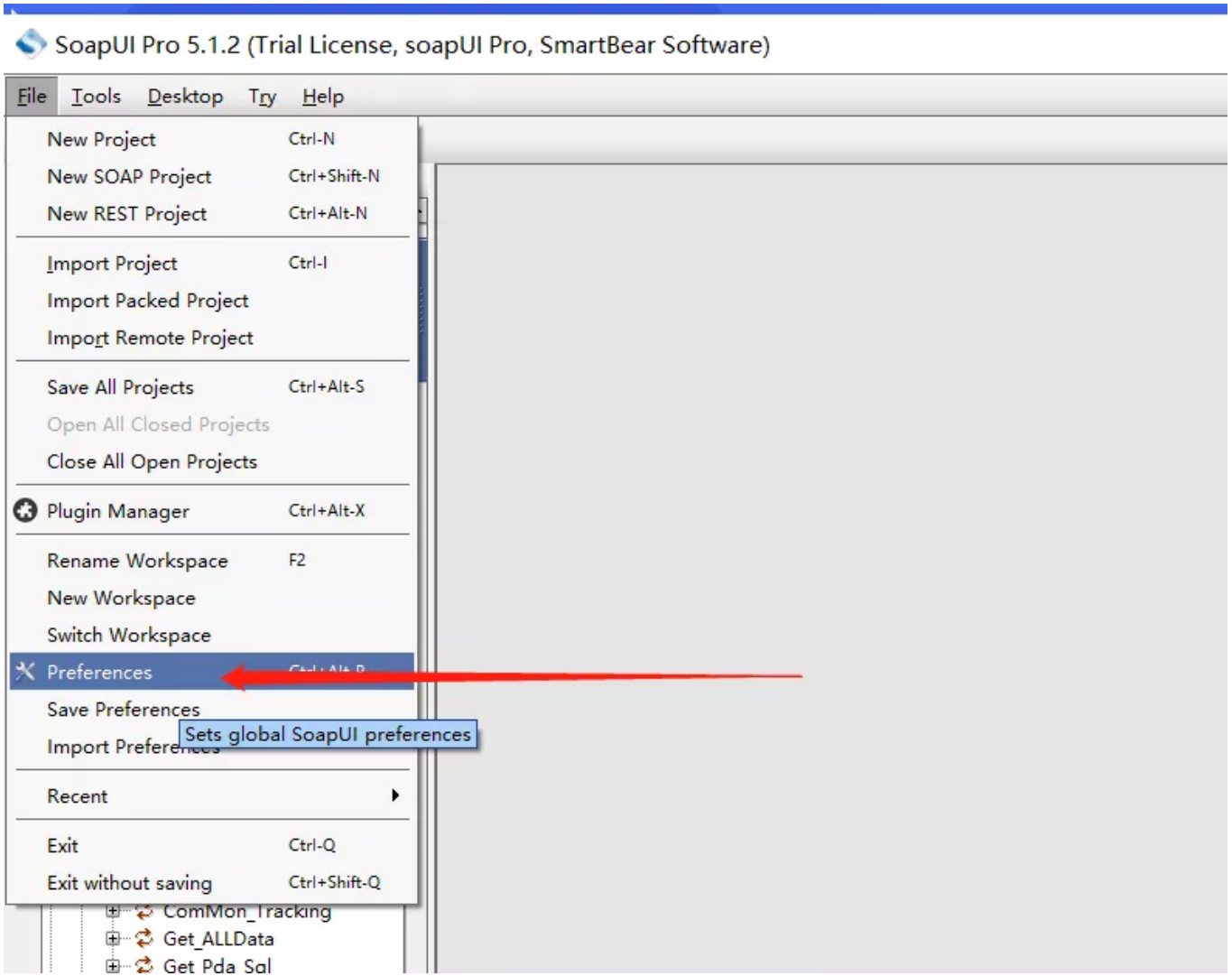
```
WebScarap
SoapUI
WCFStorm
SOA Cleaner
WSDigger
wsScanner
Wfuzz
RESTClient
BurpSuite
WS-Attacker
ZAP
Metasploit
WSDL Analyze
```

一般我们是使用burp和SoapUI联动进行对web sevice渗透测试。通过SoapUI访问Web Service，并将请求转发给BurpSuite。

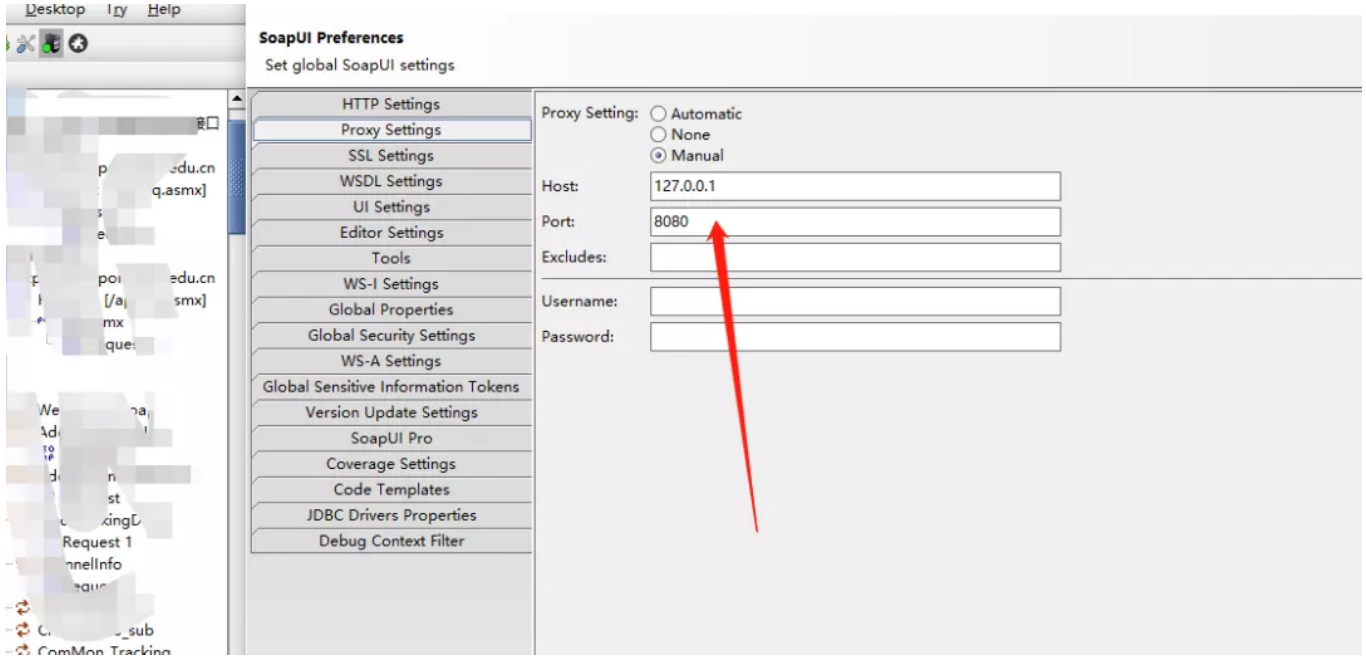
-

```
SoapUI NG Pro：渗透测试流程的发起，通信报文的解析、集合payload之后通信报文的重新组装等。
Burp Suite：代理拦截，跟踪通信过程和结果，对通信进行重放和二次处理等。
```

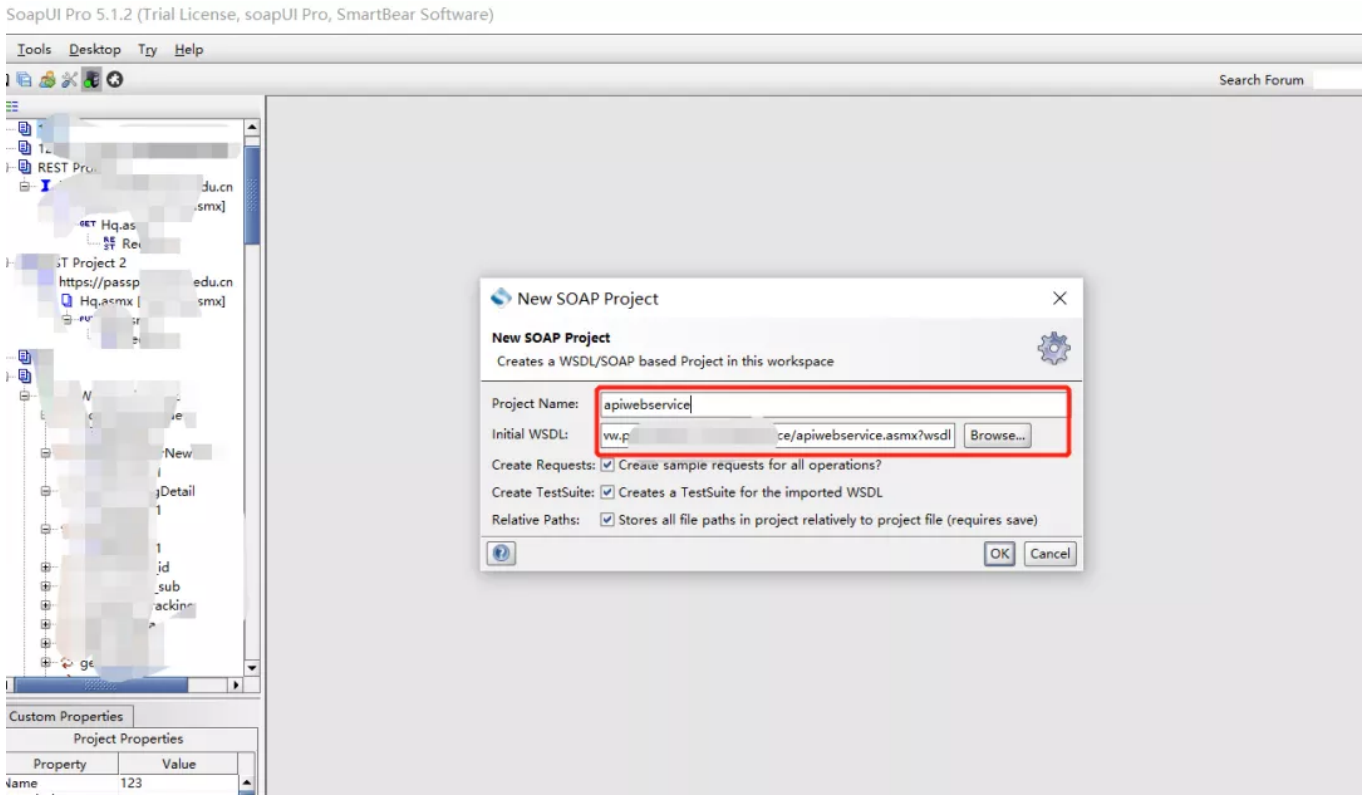
首先启动SoapUI软件，然后设置代理。



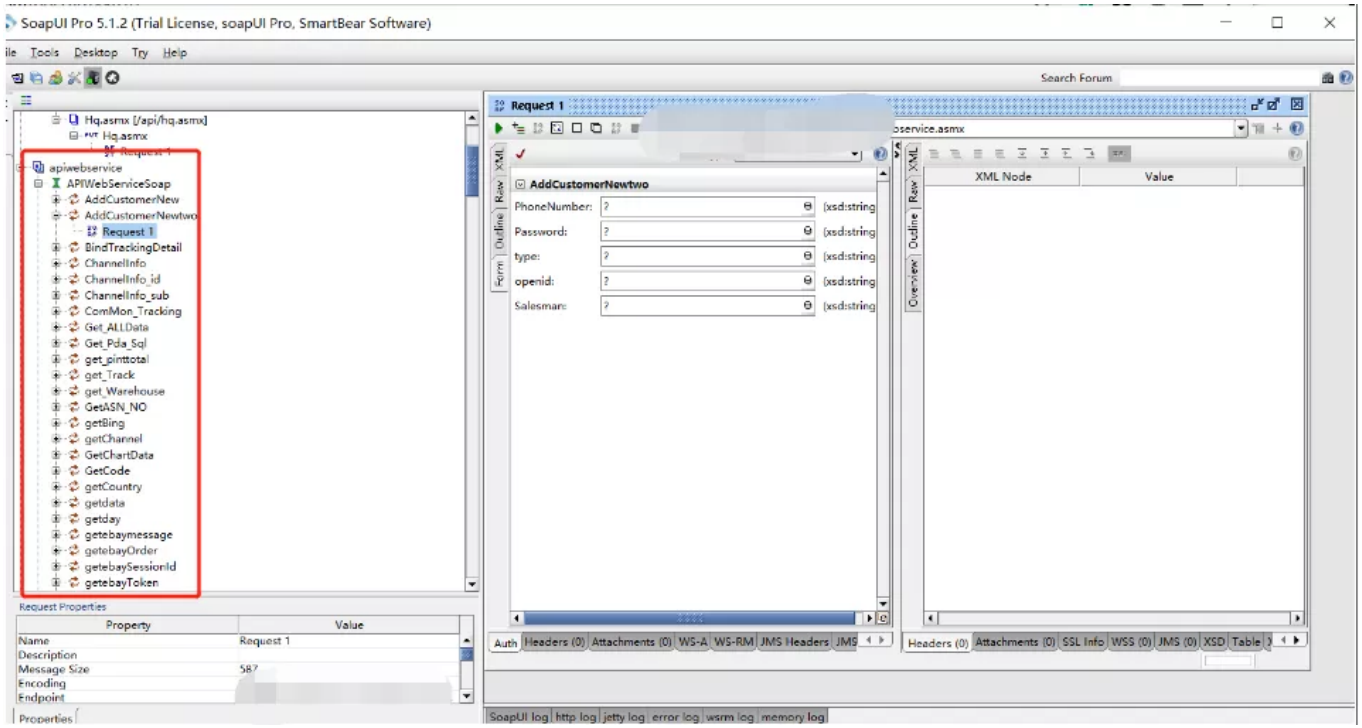
填入burp的代理IP



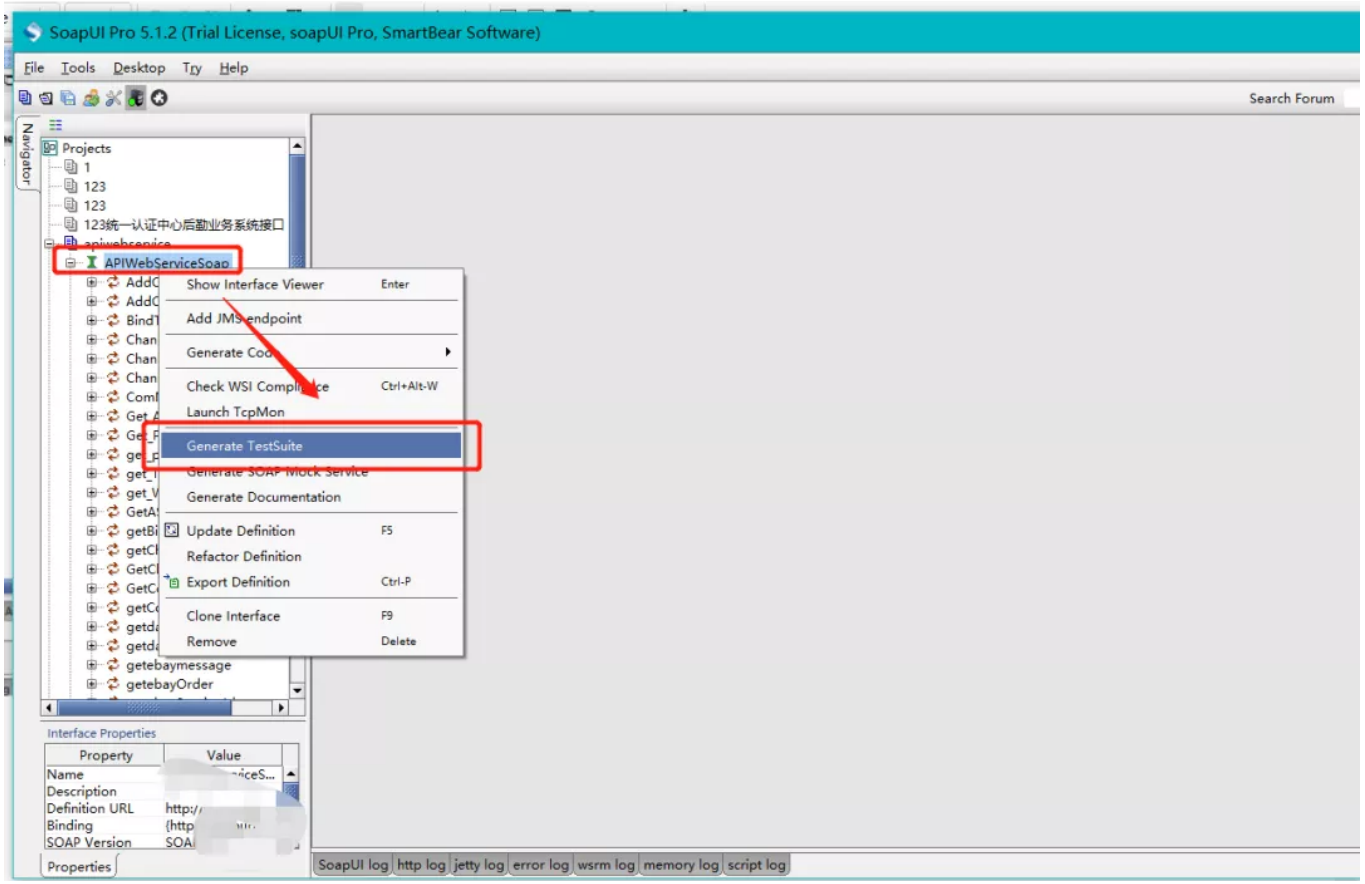
创建一个新的SOAP工程。在“Initial WSDL”一栏填入WSDL地址



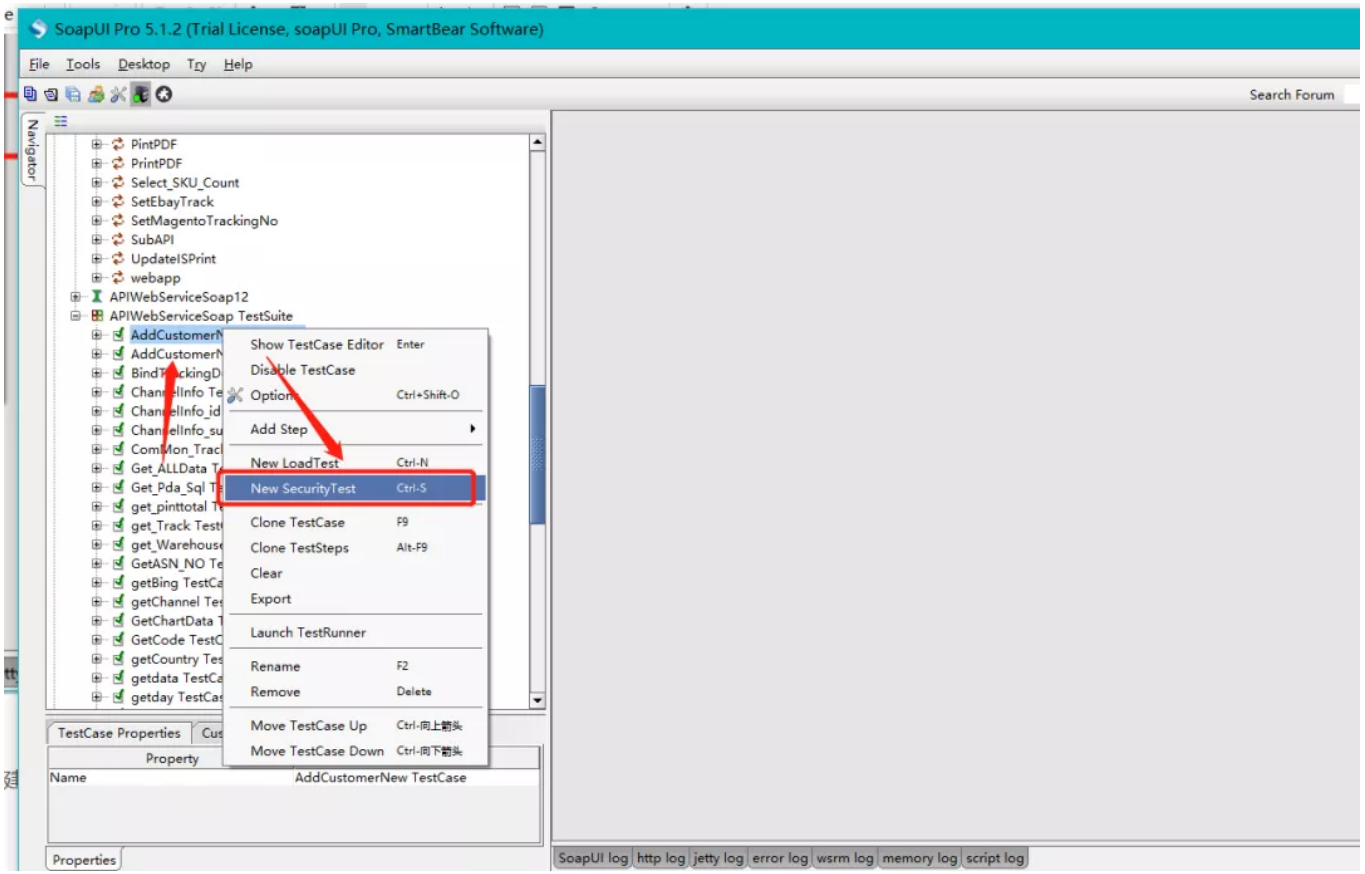
导入成功，SoapUI对给定的WSDL地址进行解析，以创建Web Service函数及请求



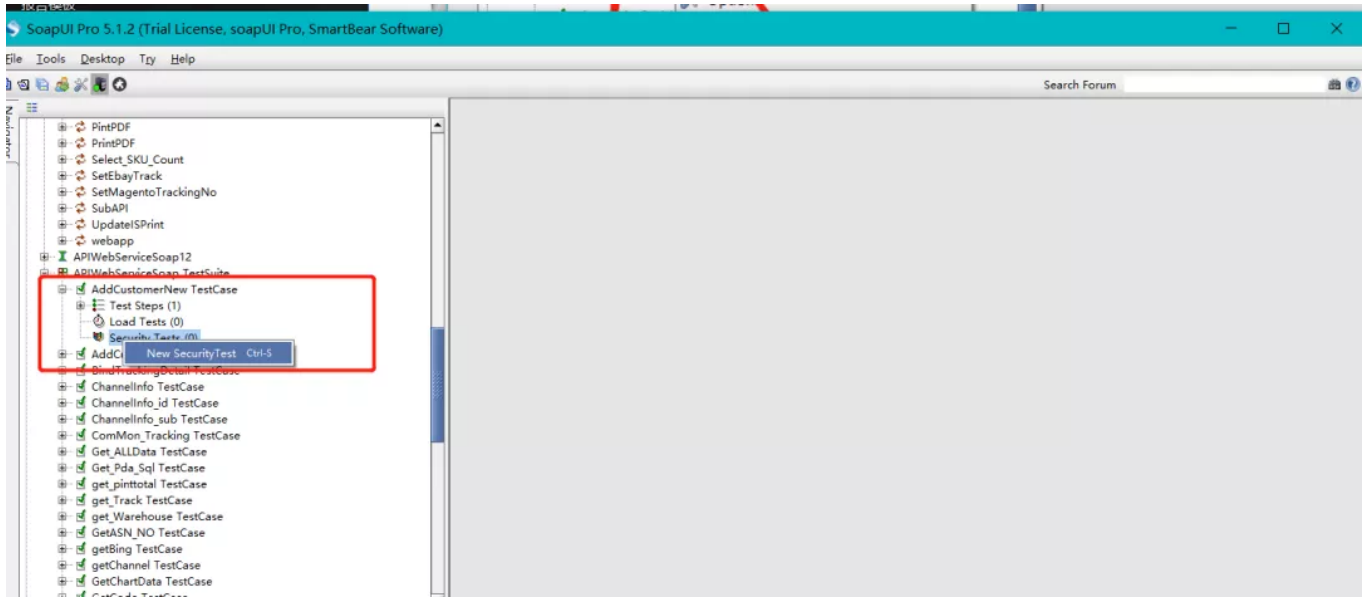
接下来我们创建一个Generate TestSuite



创建好了Generate TestSuite后，我们再来对其中一个接口来创建一个new SecurityTest.



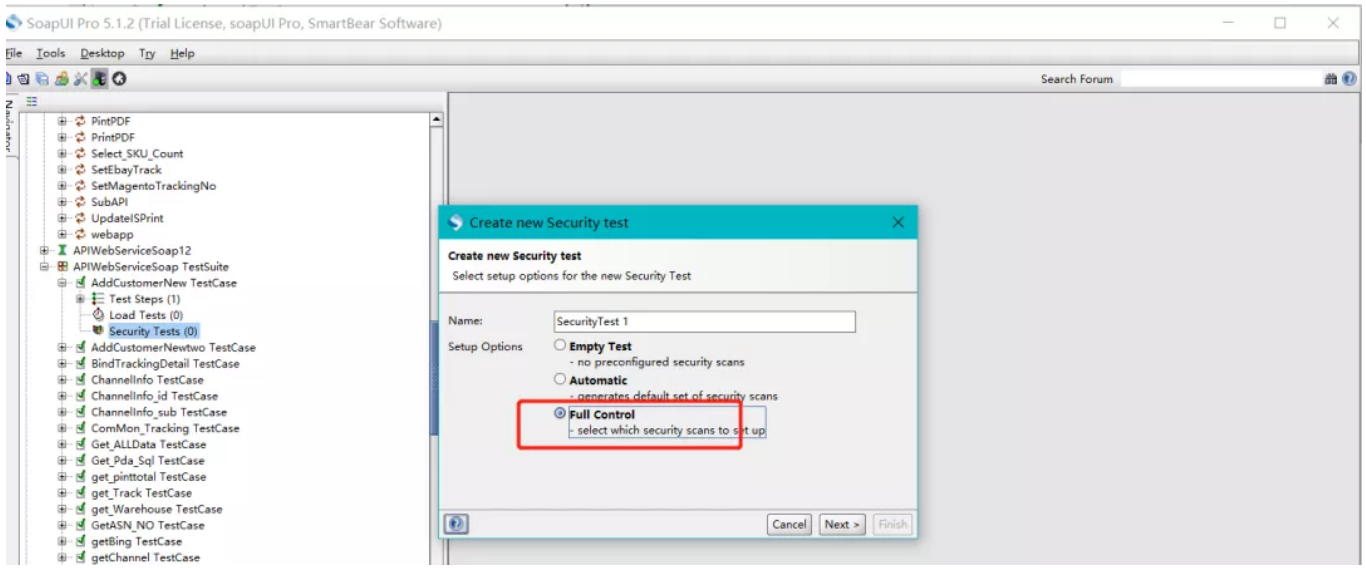
例如：我们对 AddCustomerNew TestCase接口进行SecurityTest（安全测试）



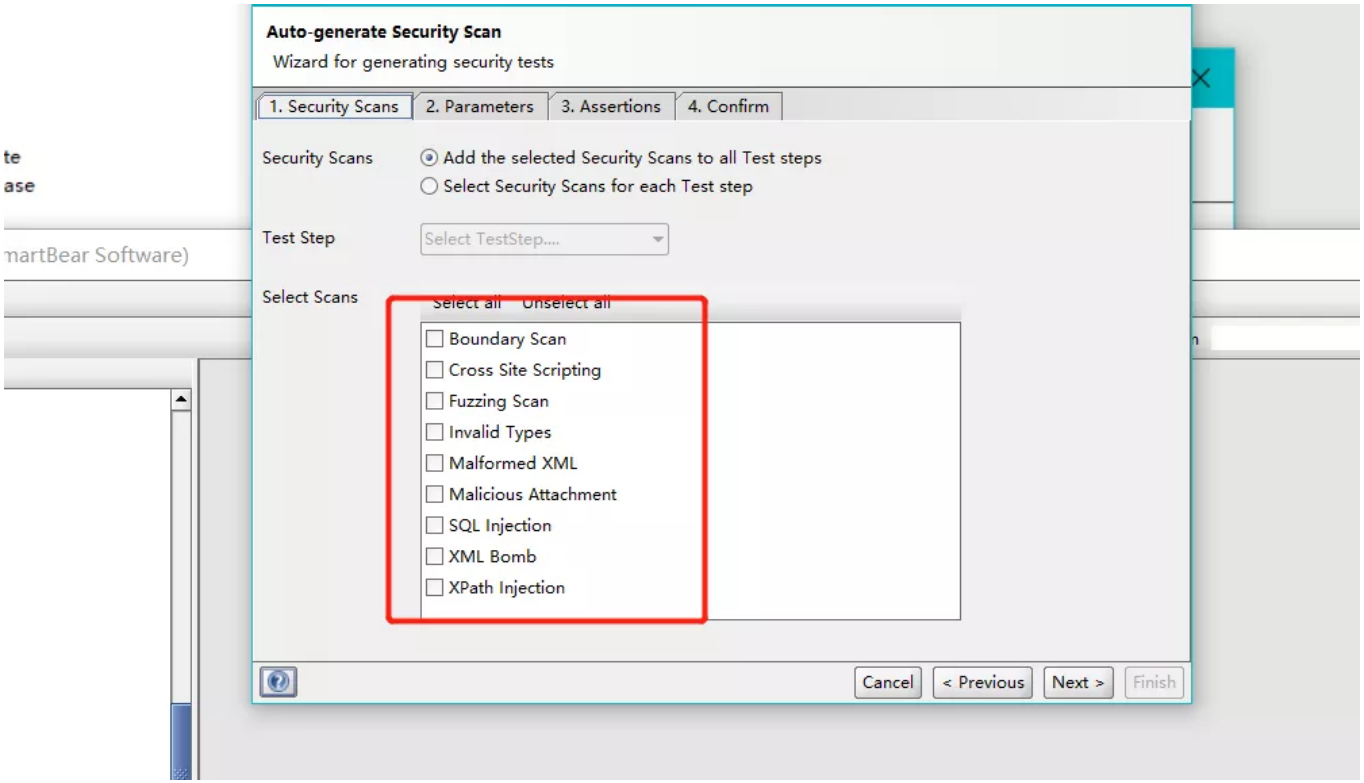
Empty 空测试

Automatic 默认安全测试，这个会默认对webservice 接口加载soapui中的所有测试模块。

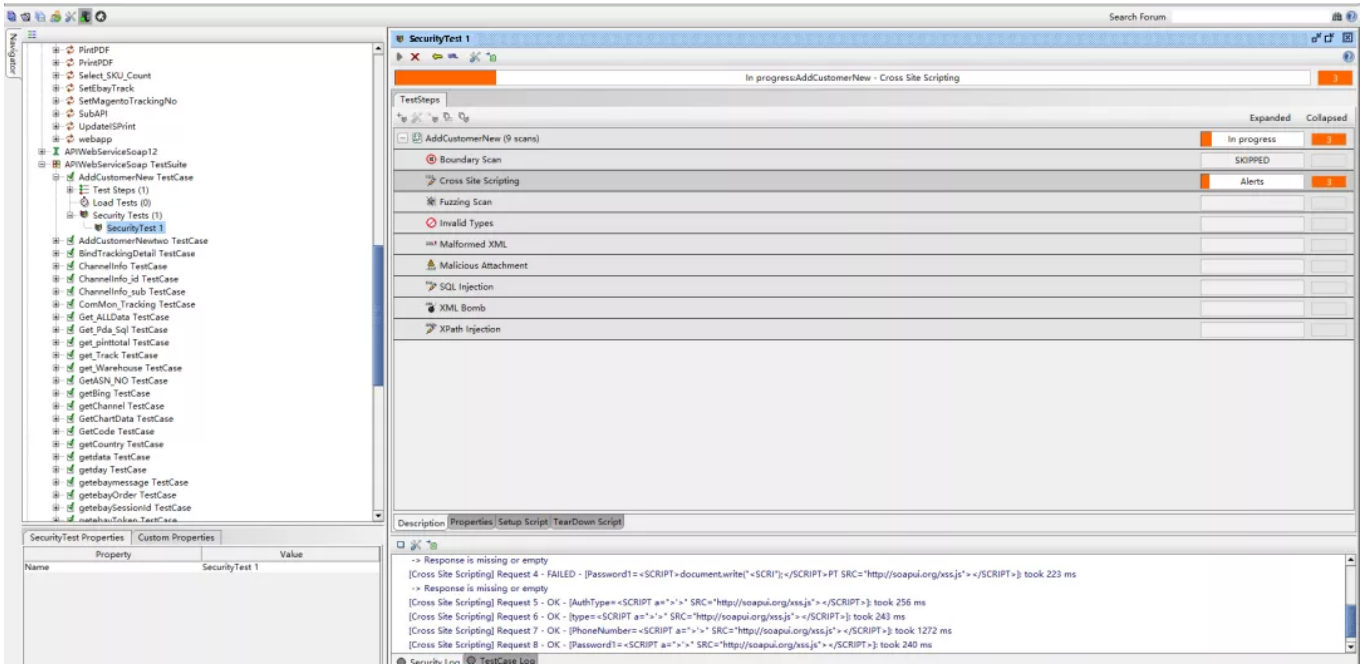
Full Control 可以自行选择测试模块



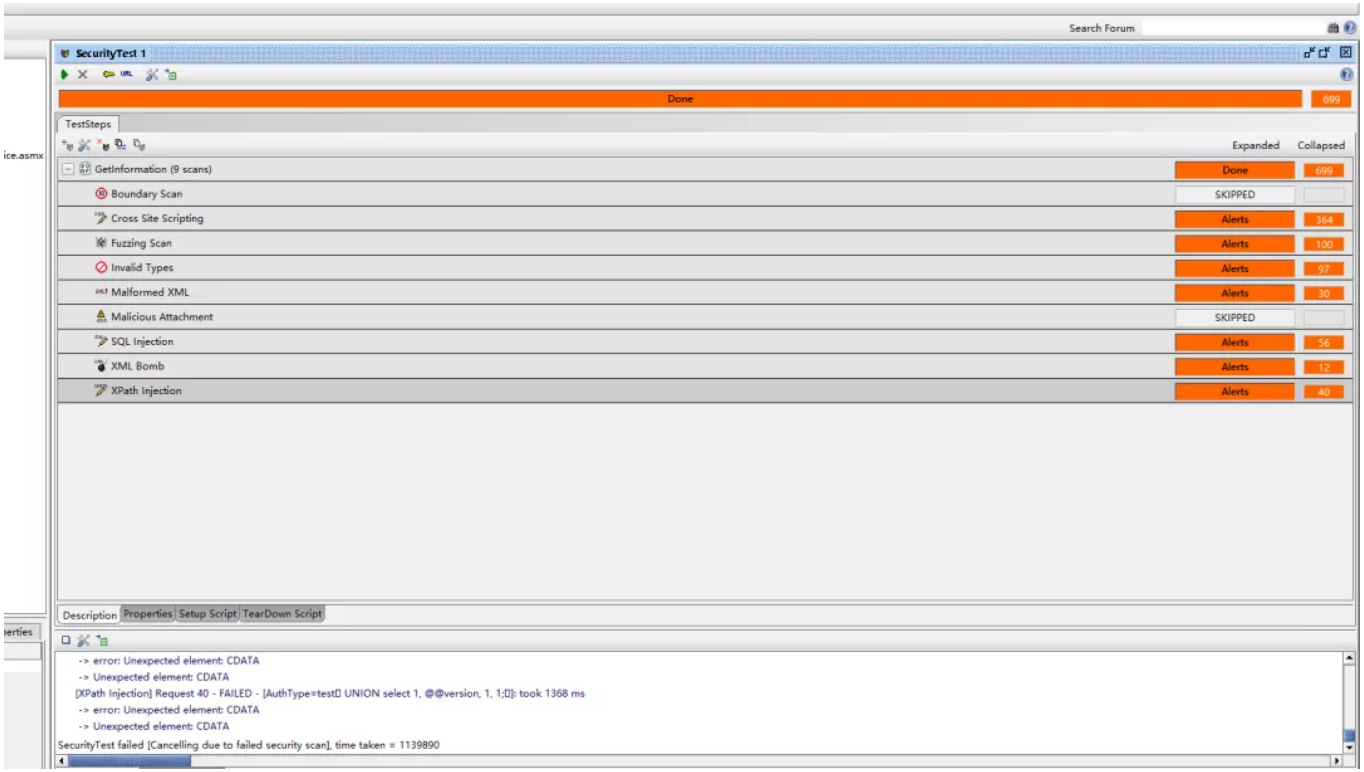
我们来看看soapui中有那些测试模块。



选择好要使用的测试模块好，我们就可以使用soapui执行自动测试。

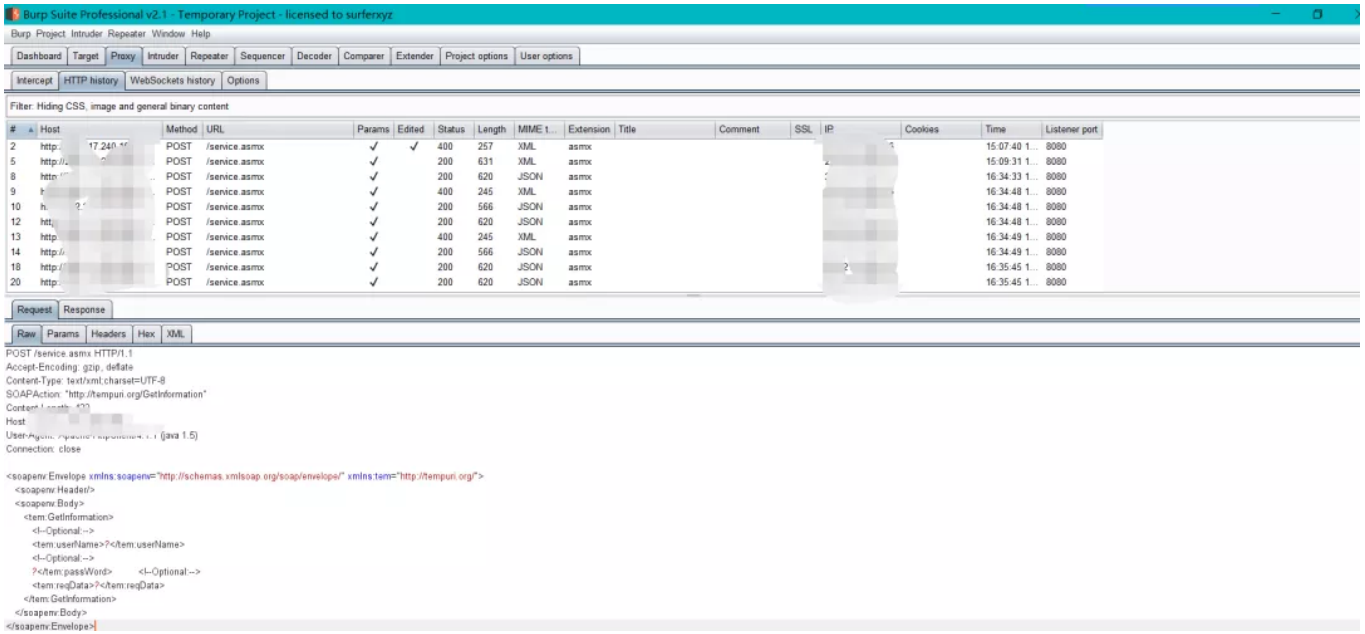

SecurityTest 1
In progress:AddCustomerNew - Cross Site Scripting
TestSteps
Expanded
Collapsed
AddCustomerNew (9 scans)
In progress
Boundary Scan
SKIPPED
Cross Site Scripting
Alerts
Fuzzing Scan
Invalid Types
Malformed XML
Malicious Attachment
SQL Injection
XML Bomb
XPath Injection
Description
Properties
Setup Script
TearDown Script
SecurityTest Properties
Custom Properties
Property
Value
Name
SecurityTest 1
-> Response is missing or empty
[Cross Site Scripting] Request 4 - FAILED - [Password1=<SCRIPT>document.write("<SCRI");</SCRIPT>PT SRC='http://soapui.org/xss.js'></SCRIPT>]: took 223 ms
-> Response is missing or empty
[Cross Site Scripting] Request 5 - OK - [AuthType=<SCRIPT a='>' SRC="http://soapui.org/xss.js"></SCRIPT>]: took 256 ms
[Cross Site Scripting] Request 6 - OK - [type=<SCRIPT a='>' SRC="http://soapui.org/xss.js"></SCRIPT>]: took 243 ms
[Cross Site Scripting] Request 7 - OK - [PhoneNumber=<SCRIPT a='>' SRC="http://soapui.org/xss.js"></SCRIPT>]: took 1272 ms
[Cross Site Scripting] Request 8 - OK - [Password1=<SCRIPT a='>' SRC="http://soapui.org/xss.js"></SCRIPT>]: took 240 ms

ok,扫描好了。



可以看到疑似漏洞699个，最终结果要我们配合burp手工去确认。

联动之后，我们可以在burp中可以看到所有的soap发送的测试数据包。



我们可以通过查看数据包和返回包来确认漏洞。

开发安全的 Web 服务是一项系统而复杂的工作。实际项目中 Web 服务的开发往往依赖于一些框架及中间件。因此如何开发安全的 Web 服务，需要结合各个框架和中间件进行具体分析。

最近应粉丝要求 我们黑白天就决定开了个星球主要是分享一些红队实战演练技巧，免杀绕过等等同时也分享翻译国外最新的技术。

目前是50/年。以后随着人数而升价，主要避免一些乱七八糟的人进来。

因为团队中三个人，两个在准备HW,一个在准备一个重要比赛都比较忙，所以星球在9月之后才慢慢放一些文章教程。望理解！

欢迎加入，一起成长！

进去不了的同学不要着急，等几天星球过了审核就行了。

资料参考；

https://www.ibm.com/developerworks/cn/rational/r-cn-appscanwebservicessecurity/index.html