Web Services Description Language (WSDL)

WSDL stands for Web Service Description Language. WSDL is an XML-based file exposed by web services that informs clients of the provided services/methods, including where they reside and the method-calling convention.

A web service's WSDL file should not always be accessible. Developers may not want to publicly expose a web service's WSDL file, or they may expose it through an uncommon location, following a security through obscurity approach. In the latter case, directory/parameter fuzzing may reveal the location and content of a WSDL file.

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 service and follow along.

Suppose we are assessing a SOAP service residing in `http://<TARGET IP>:3002`. We have not been informed of a WSDL file.

Let us start by performing basic directory fuzzing against the web service.

Web Services Description Language (WSDL)

```
MisaelMacias@htb[/htb]$ dirb http://<TARGET IP>:3002

-----------------
DIRB v2.22
By The Dark Raver
-----------------

START_TIME: Fri Mar 25 11:53:09 2022
URL_BASE: http://<TARGET IP>:3002/
WORDLIST_FILES: /usr/share/dirb/wordlists/common.txt

-----------------

GENERATED WORDS: 4612

---- Scanning URL: http://<TARGET IP>:3002/ ----
+ http://<TARGET IP>:3002/wsdl (CODE:200|SIZE:0)

-----------------
END_TIME: Fri Mar 25 11:53:24 2022
DOWNLOADED: 4612 - FOUND: 1
```

It looks like `http://<TARGET IP>:3002/wsdl` exists. Let us inspect its content as follows.

Web Services Description Language (WSDL)

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

The response is empty! Maybe there is a parameter that will provide us with access to the SOAP web service's WSDL file. Let us perform parameter fuzzing using *ffuf* and the burp-parameter-names.txt list, as follows. *-fs 0* filters out empty responses (size = 0) and *-mc 200* matches *HTTP 200* responses.

Web Services Description Language (WSDL)

```
MisaelMacias@htb[/htb]$ ffuf -w /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt -u 'http://<TARGET

        /'___\  /'___\           /'___\
       /\ \__/ /\ \__/  __  __  /\ \__/
       \ \ ,__\\ \ ,__\/\ \/\ \ \ \ ,__\
        \ \ \_/ \ \ \_/\ \ \_\ \ \ \ \_/
         \ \_\   \ \_\  \ \____/  \ \_\
          \/_/    \/_/   \/___/    \/_/

       v1.3.1 Kali Exclusive <3
________________________________________________

 :: Method           : GET
 :: URL              : http://<TARGET IP>:3002/wsdl?FUZZ
 :: Wordlist         : FUZZ: /usr/share/seclists/Discovery/Web-Content/burp-parameter-names.txt
 :: Follow redirects : false
 :: Calibration      : false
 :: Timeout          : 10
 :: Threads          : 40
 :: Matcher          : Response status: 200
 :: Filter           : Response size: 0
________________________________________________

:: Progress: [40/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Error
:: Progress: [537/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Erro
wsdl [Status: 200, Size: 4461, Words: 967, Lines: 186]
:: Progress: [982/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Erro::
Progress: [1153/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Err::
Progress: [1780/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Err::
Progress: [2461/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Err::
Progress: [2588/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Err::
Progress: [2588/2588] :: Job [1/1] :: 0 req/sec :: Duration: [0:00:00] :: Errors: 0 ::
```

It looks like *wsdl* is a valid parameter. Let us now issue a request for `http://<TARGET IP>:3002/wsdl?wsdl`

Web Services Description Language (WSDL)

```
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/"
```

? 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

```
    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>
    <!-- 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>
```

We identified the SOAP service's WSDL file!

Note: WSDL files can be found in many forms, such as `/example.wsdl`, `?wsdl`, `/example.disco`, `?disco` etc. DISCO is a Microsoft technology for publishing and discovering Web Services.

WSDL File Breakdown

Let us now go over the identified WSDL file above together.

The above WSDL file follows the WSDL version 1.1 layout and consists of the following elements.

- `Definition`
  - The root element of all WSDL files. Inside the definition, the name of the web service is specified, all namespaces used across the WSDL document are declared, and all other service elements are defined.
  - Code: `xml`

```
<wsdl:definitions targetNamespace="http://tempuri.org/"

    <wsdl:types></wsdl:types>
    <wsdl:message name="LoginSoapIn"></wsdl:message>
    <wsdl:portType name="HacktheBoxSoapPort">
      <wsdl:operation name="Login"></wsdl:operation>
    </wsdl:portType>
    <wsdl:binding name="HacktheboxServiceSoapBinding" type="tns:HacktheBoxSoapPort">
```

```
        <wsdl:operation name="Login">
            <soap:operation soapAction="Login" style="document"/>
            <wsdl:input></wsdl:input>
            <wsdl:output></wsdl:output>
        </wsdl:operation>
    </wsdl:binding>
    <wsdl:service name="HacktheboxService"></wsdl:service>
</wsdl:definitions>
```

- Data Types
  - The data types to be used in the exchanged messages.
  - Code: xml

```
<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>
```

- Messages
  - Defines input and output operations that the web service supports. In other words, through the *messages* element, the messages to be exchanged, are defined and presented either as an entire document or as arguments to be mapped to a method invocation.
  - Code: xml

```
<!-- 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>
<!-- 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>
```

- Operation
  - Defines the available SOAP actions alongside the encoding of each message.
- Port Type
  - Encapsulates every possible input and output message into an operation. More specifically, it defines the web service, the available operations and the exchanged messages. Please note that in WSDL version 2.0, the *interface* element is tasked with defining the available operations and when it comes to messages the (data) types element handles defining them.
  - Code: xml

```
<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>
```

- Binding
  - Binds the operation to a particular port type. Think of bindings as interfaces. A client will call the relevant port type and, using the details provided by the binding, will be able to access the operations bound to this port type. In other words, bindings provide web service access details, such as the message format, operations, messages, and interfaces (in the case of WSDL version 2.0).

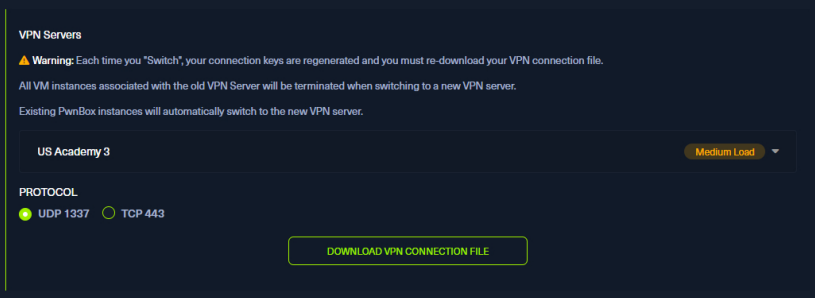
- Code: xml

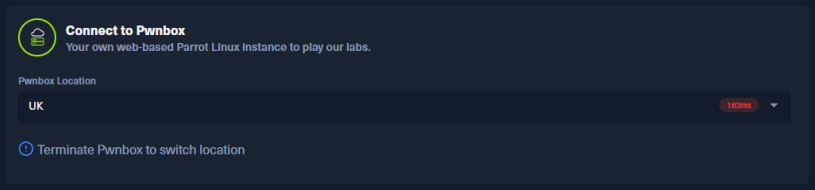
```xml
<wsdl:binding name="HacktheboxServiceSoapBinding" type="tns:HacktheBoxSoapPort">
    <soap:binding transport="http://schemas.xmlsoap.org/soap/http"/>
    <!-- SOAP Login Action -->
    <wsdl:operation name="Login">
      <soap:operation soapAction="Login" style="document"/>
      <wsdl:input>
          <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>
```

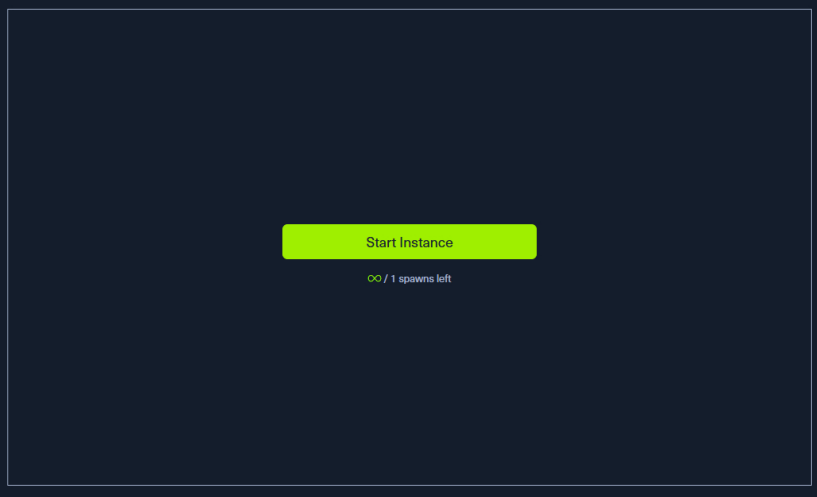
- Service
  - A client makes a call to the web service through the name of the service specified in the service tag. Through this element, the client identifies the location of the web service.
  - Code: xml

```xml
    <wsdl:service name="HacktheboxService">

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

    </wsdl:service>
```

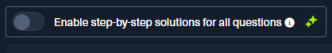
In the `SOAP Action Spoofing` section, later on, we will see how we can leverage the identified WSDL file to interact with the web service.

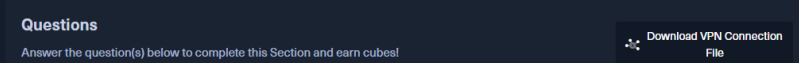


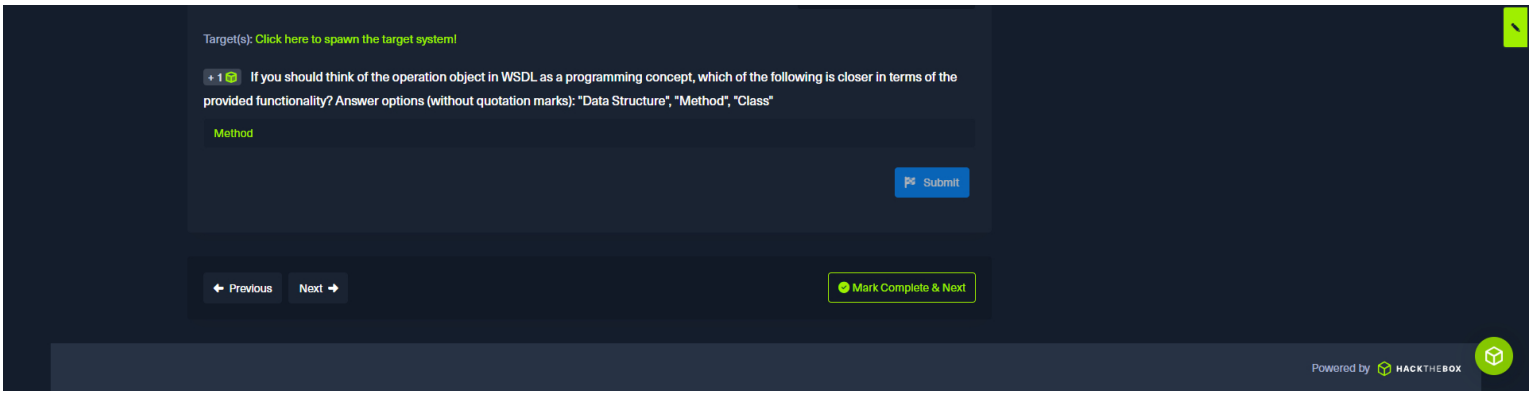




Waiting to start...






Target(s): Click here to spawn the target system!
+ 1
If you should think of the operation object in WSDL as a programming concept, which of the following is closer in terms of the provided functionality? Answer options (without quotation marks): "Data Structure", "Method", "Class"
Method
Submit
Previous
Next
Mark Complete & Next
Powered by HACKTHEBOX