# Offensive Security Exploit Developer Exam Report

OSED Exam Report

student@youremailaddress.com, OSID: XXXX

2021-03-25

# Contents

# 1 Offensive Security OSED Exam Report

## 1.1 Introduction

The Offensive Security OSED exam documentation contains all efforts that were conducted in order to pass the Offensive Security Exploit Developer exam. This report 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 the technical knowledge required to pass the qualifications for the Offensive Security Exploit Developer certification.

### 1.1.1 Objective

The objective of this exam is to solve three given assignments as described in the control panel. The student is tasked with following a methodical approach in analyzing and solving the assignments. The exam report is meant to be a writeup of the steps taken to solve the assignment, including any analysis performed and code written.

An example page has already been created for you at the latter portions of this document that should give you ample information on what is expected to pass this exam. Use the sample report as a guideline to get you through the reporting, while removing any headlines that are note relevant to a specific assignment.

### 1.1.2 Requirements

The student will be required to fill out this exam documentation fully and to include the following sections:

- High-Level summary of assignment solutions.
- Methodology walkthrough and detailed outline of steps taken through analysis and all written code.
- Each finding with included screenshots, walkthrough, sample code or reference.
- Screenshot of proof.txt.

## 1.2 High-Level Summary

A brief description of the assignments that were solved, including the overall exploitation steps.

| | IP (Hostname) | Vulnerable Service/App | Proof.txt Contents |
|---|---|---|---|
| Assignment X | 192.168.xx.xx | app_name | `xxx` |
| Assignment Y | 192.168.yy.yy | app_name | `yyy` |
| Assignment Z | 192.168.zz.zz | app_name | `zzz` |

## 1.3 Assignment X

### 1.3.1 Proof.txt

- proof.txt: `xxx`

Provide a screenshot of running `type proof.txt` and the `ipconfig` command from the directory where proof.txt is stored.

**Figure 1.1:** local.txt

### 1.3.2 Initial Analysis

Provide relevant techniques and methods used to perform enumeration of the application, including **network ports**, **security mitigations** etc. The steps taken should be reproducible and easy to understand. Include any custom code or references to public tools.

Listening Ports:

- `192.168.xx.xx:yyyy`
- `127.0.0.1:zzzz`
- `0.0.0.0:zzzz`

Security Mitigations:

- `xxx.dll`: ASLR, DEP, SafeSEH
- `yyy.dll`: ASLR, DEP, SafeSEH
- `zzz.dll`: ASLR, DEP, SafeSEH

References:

- tool xxx

### 1.3.3 Application Analysis

Provide a description of the analysis performed against the application, this includes both **dynamic** and **static** analysis.

The analysis should include any reverse engineering performed to understand network protocols or file formats as well as how the application may be triggered to dispatch available commands.

#### 1.3.3.1 Static Analysis

- static analysis

#### 1.3.3.2 Dynamic Analysis

- dynamic analysis

### 1.3.4 Vulnerability Discovery

Provide relevant analysis steps to locate vulnerabilities inside the application, this includes both results from static analysis and dynamic analysis.

As part of the documentation, proof of concept **Python3** code must be **created** and **explained** that triggers the vulnerabilities.

Only the steps that ended up working are required.

#### 1.3.4.1 Analysis

- vuln discovery analysis

#### 1.3.4.2 Initial PoC

To install the dependencies required for PoC execution:

```
package_manager install dependency1 dependency2
```

Provide the proof of concept code used to **trigger the vulnerability**.

```
#!/usr/bin/env python3

print('[+] Triggering vulnerability')
```

### 1.3.5 Exploit Creation

Provide a description of steps to create the exploit, this includes how to combine vulnerabilities, how to bypass DEP and how to write any custom shellcode. At the end of this section the full exploit code should be developed while an explanation of each step should be performed.

Steps to Create the Exploit:

1. step one
2. step two

#### 1.3.5.1 Full PoC

Provide the proof of concept code used to **gain access to the server**.

```
#!/usr/bin/env python3

print('[+] Exploit sent, awaiting shell')
```

## 1.4 Assignment Y

### 1.4.1 Proof.txt

- proof.txt: `xxx`

Provide a screenshot of running `type proof.txt` and the `ipconfig` command from the directory where proof.txt is stored.

**Figure 1.2:** local.txt

### 1.4.2 Initial Analysis

Provide relevant techniques and methods used to perform enumeration of the application, including **network ports**, **security mitigations** etc. The steps taken should be reproducible and easy to understand. Include any custom code or references to public tools.

Listening Ports:

- `192.168.xx.xx:yyyy`
- `127.0.0.1:zzzz`
- `0.0.0.0:zzzz`

Security Mitigations:

- `xxx.dll`: ASLR, DEP, SafeSEH
- `yyy.dll`: ASLR, DEP, SafeSEH
- `zzz.dll`: ASLR, DEP, SafeSEH

References:

- tool xxx

### 1.4.3 Application Analysis

Provide a description of the analysis performed against the application, this includes both **dynamic** and **static** analysis.

The analysis should include any reverse engineering performed to understand network protocols or file formats as well as how the application may be triggered to dispatch available commands.

#### 1.4.3.1 Static Analysis

- static analysis

#### 1.4.3.2 Dynamic Analysis

- dynamic analysis

### 1.4.4 Vulnerability Discovery

Provide relevant analysis steps to locate vulnerabilities inside the application, this includes both results from static analysis and dynamic analysis.

As part of the documentation, proof of concept **Python3** code must be **created** and **explained** that triggers the vulnerabilities.

Only the steps that ended up working are required.

#### 1.4.4.1 Analysis

- vuln discovery analysis

#### 1.4.4.2 Initial PoC

To install the dependencies required for PoC execution:

```
package_manager install dependency1 dependency2
```

Provide the proof of concept code used to **trigger the vulnerability**.

```
#!/usr/bin/env python3

print('[+] Triggering vulnerability')
```

### 1.4.5 Exploit Creation

Provide a description of steps to create the exploit, this includes how to combine vulnerabilities, how to bypass DEP and how to write any custom shellcode. At the end of this section the full exploit code should be developed while an explanation of each step should be performed.

Steps to Create the Exploit:

1. step one
2. step two

#### 1.4.5.1 Full PoC

Provide the proof of concept code used to **gain access to the server**.

```
#!/usr/bin/env python3

print('[+] Exploit sent, awaiting shell')
```

## 1.5 Assignment Z

### 1.5.1 Proof.txt

- proof.txt: `xxx`

Provide a screenshot of running `type proof.txt` and the `ipconfig` command from the directory where proof.txt is stored.

**Figure 1.3:** local.txt

### 1.5.2 Initial Analysis

Provide relevant techniques and methods used to perform enumeration of the application, including **network ports**, **security mitigations** etc. The steps taken should be reproducible and easy to understand. Include any custom code or references to public tools.

Listening Ports:

- `192.168.xx.xx:yyyy`
- `127.0.0.1:zzzz`
- `0.0.0.0:zzzz`

Security Mitigations:

- `xxx.dll`: ASLR, DEP, SafeSEH
- `yyy.dll`: ASLR, DEP, SafeSEH
- `zzz.dll`: ASLR, DEP, SafeSEH

References:

- tool xxx

### 1.5.3 Application Analysis

Provide a description of the analysis performed against the application, this includes both **dynamic** and **static** analysis.

The analysis should include any reverse engineering performed to understand network protocols or file formats as well as how the application may be triggered to dispatch available commands.

#### 1.5.3.1 Static Analysis

- static analysis

#### 1.5.3.2 Dynamic Analysis

- dynamic analysis

### 1.5.4 Vulnerability Discovery

Provide relevant analysis steps to locate vulnerabilities inside the application, this includes both results from static analysis and dynamic analysis.

As part of the documentation, proof of concept **Python3** code must be **created** and **explained** that triggers the vulnerabilities.

Only the steps that ended up working are required.

#### 1.5.4.1 Analysis

- vuln discovery analysis

#### 1.5.4.2 Initial PoC

To install the dependencies required for PoC execution:

```
package_manager install dependency1 dependency2
```

Provide the proof of concept code used to **trigger the vulnerability**.

```
#!/usr/bin/env python3

print('[+] Triggering vulnerability')
```

#### 1.5.5 Exploit Creation

Provide a description of steps to create the exploit, this includes how to combine vulnerabilities, how to bypass DEP and how to write any custom shellcode. At the end of this section the full exploit code should be developed while an explanation of each step should be performed.

Steps to Create the Exploit:

1. step one
2. step two

##### 1.5.5.1 Full PoC

Provide the proof of concept code used to **gain access to the server**.

```
#!/usr/bin/env python3

print('[+] Exploit sent, awaiting shell')
```

### 1.6 Additional Items Not Mentioned in the Report

This section is placed for any additional items that were not mentioned in the overall report.