BÁO CÁO THỰC HÀNH

**Bài thực hành số xx: Tên bài thực hành**

**Môn học:** Nhập môn Mạng máy tính

**Lớp:** NT100.ATTT.2

**THÀNH VIÊN THỰC HIỆN (Nhóm xx):**

|  |  |  |
| --- | --- | --- |
| **STT** | **Họ và tên** | **MSSV** |
| 1 | Nguyễn Thị A | 20520001 |
| 2 | Phan Văn B | 20520002 |

|  |
| --- |
| **Điểm tự đánh giá** |
|  |

**ĐÁNH GIÁ KHÁC:**

|  |  |
| --- | --- |
| Tổng thời gian thực hiện |  |
| Phân chia công việc |  |
| Ý kiến *(nếu có)*  + Khó khăn  + Đề xuất, kiến nghị |  |

Phần bên dưới của báo cáo này là báo cáo chi tiết của nhóm thực hiện

MỤC LỤC

[**A.** BÁO CÁO CHI TIẾT 2](#_Toc117536117)

[1. Nội dung 1 2](#_Toc117536118)

[a. Nội dung a 2](#_Toc117536119)

[b. Nội dung b 2](#_Toc117536120)

[2. Khai thác lỗ hổng insecure deserialization 2](#_Toc117536121)

[a. Tổng quan về PHP serialization 2](#_Toc117536122)

[b. Khai thác chức năng serialize và deserialize của PHP 3](#_Toc117536123)

[c. Khai thác định dạng Java serialization 6](#_Toc117536124)

[d. Khai thác định dạng python serialization 7](#_Toc117536125)

[e. Bài tập CTF 10](#_Toc117536126)

# BÁO CÁO CHI TIẾT

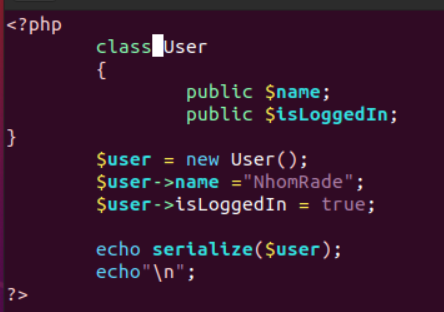
## Nội dung 1

### Nội dung a

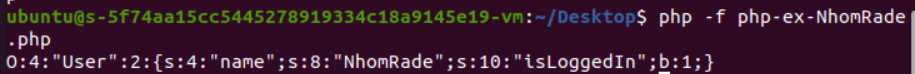
### Nội dung b

## Khai thác lỗ hổng insecure deserialization

### Tổng quan về PHP serialization



+ Bước 1: Tạo 1 file php-ex-NhomRade.php



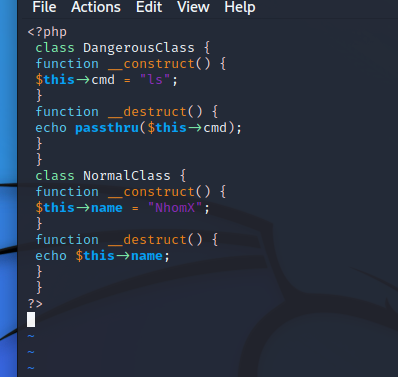
+ Bước 2: Thực thi file php-ex-NhomRade.php với câu lệnh giống như trên

* Dòng ouput trên cung cấp các thông tin về đối tượng user đã tạo và serialize
* Dòng output trên trả về dạng byte của giữ liệu
* s:4: Nghĩa là chuỗi name dài 4 ký tự và tương tự như những chuỗi khác
* b:1 Nghĩa là giá trị boolean

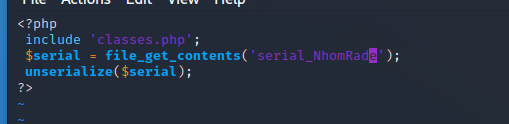
### Khai thác chức năng serialize và deserialize của PHP

+ Tạo 2 file php có nội dung như sau

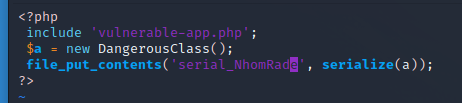
* Classes.php



* Vulnerable-app-1.php



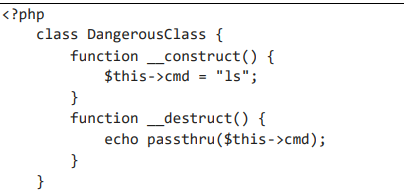
* Normal-user.php



Tạo và serialize đối tượng của class và lưu vào file serial\_NhomRade



+ Kết quả của các file là hoạt động bình thường của ứng dụng



* Vulnerable-app-1.php

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Trong DangerousClass có 1 phương thức là \_destruc() được dùng khi 1 object bị hủy bỏ hay không còn tồn tại trong chương trình nữa. Nên khi hủy bỏ chương trình thì những code bên trong sẽ được thực thi

+ File vulnerable-app-1.php đọc file người dùng nhưng không xử lý gì cả nên có thể chèn sữ liệu độc hại vào

+ Khi bị khai thác có thể dẫn đến nhựng loại tấn công như: Code Injection, SQL Injection, DoS

* Khai thác
* 

+ Ta định nghĩa lại DangerouseClass để thực hiện chức năng mong muốn ( ở đây là lệnh “id”

+ Sau đó lưu lại vào file serial\_NhomRade

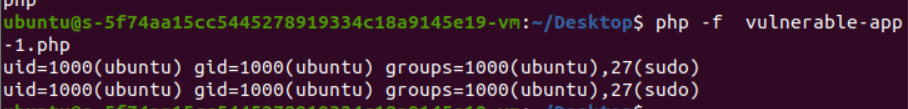
Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Do ta có thể truy cập vào dữ liệu đã serial ở trong file serial\_NhomRade

+ Nên code khai thác ta chỉ cần đọc file đạ được serialize là có thể khai tác thành công

* Kết quả



### Khai thác định dạng Java serialization

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Tạo file a JavaDeserial.java có lỗ hổng

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Code dùng để tấn công tên JavaSerial.java.

Ảnh có chứa văn bản

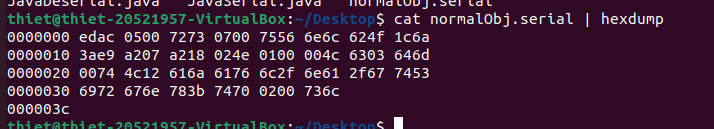
Mô tả được tạo tự động

+ Thực hiện chạy code tấn công



+ Đọc đối tượng sau khi serialize

* Phân tích



+ Trong java object khi được serialized sẽ được lưu dưới dạng byte và cũng lưu theo một định dạng nhất định. Cụ thể, các class kế thừa từ java.io.Serializable đều có thể được serialize và định dạng sau khi serialize và sẽ bắt đầu bằng ac ed 00 05 ở dạng hexa

+ Ở little edian các byte này sẽ được biểu diễn là edac 0500

+ Nếu ta thử decode các byte này theo thứ tự ac ed 00 05 thì ta được chuỗi bit sau: 10101100111011010000000000000101

+ Chuỗi này khi ở base64 tương tứng là

* 101011 - r
* 001110 - O
* 110100 - 0
* 000000 - A
* 000001 - B

+ Vậy bất kỳ khi nào thấy một chuỗi base64 bắt đầu với rO0AB ta có thể nghĩ ngay đến việc chuỗi này là 1 chuỗi serialize của 1 object nào đó trong java. Từ đó có thể khai thác điểm yếu này nếu không kiểm soát việc serialize là deserialize

### Khai thác định dạng python serialization

* Khai thác lỗ hổng thư viện pickle đơn giản

Ảnh có chứa văn bản

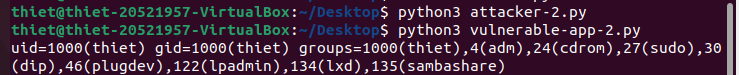
Mô tả được tạo tự động

+ Tạo 1 ứng dụng có chức năng deserialize 1 đối tượng được đọc tù file

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Code khai thác bằng cách định nghĩa 1 class đối tượng độc hại VulnPicle, tạo và dumps đối tượng và lưu vào file sẽ được đọc bởi ứng dụng trên

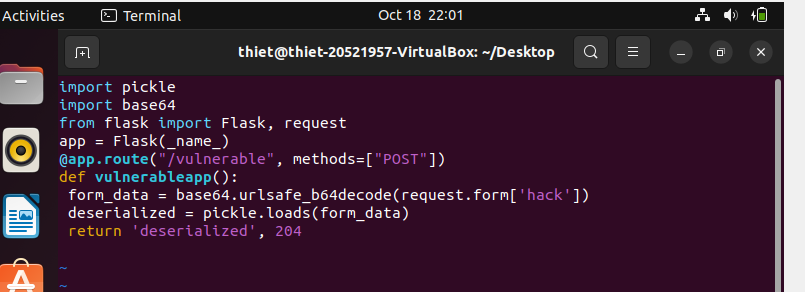


+ Thực thi và kết quả

+ định nghĩa class VulnPickle, khi vulnerable-app-2 thực hiện load đối tượng tù file, ta có kết quả như trên vì:

+ Ở class VulnPickcle, có sử dụng hàm \_\_reduce\_\_(), hàm này được hiểu là hàm không đối số và sẽ trả về 1 chuỗi hoặc 1 giá trị

+ Triển khai này có thể cung cấp cho quá trình pickling “callable” và 1 đối số để chạy là “import os”, thực thi gọi lệnh xác định id của quá trình thực thi thông qua os.system()

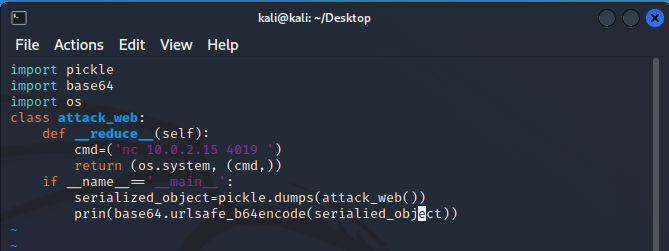
* Khai thác lỗ hổng thư viện pickle nâng cao
* 

+ Tạo 1 web server đơn giản với code như trên

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Chạy chương trình

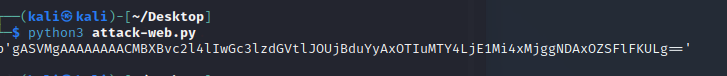


+ Tạo một file để tấn công tên attack-web.py để mở reverse shell port 4019

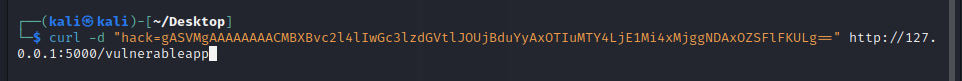
Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Mở port để lắng nghe



+ Chạy file attack-web.py để lấy cookie



+ Sử dụng câu lệnh curl -d để POST data và set hack bắng cookie đã lấy ở trên

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Đã reverse shell để kết nối tới máy qua port 4019

### Bài tập CTF

* Modifying serialized objects



+ Truy cập vào trang web và access bài lab

+ Ta thấy phần login và nhao65 username: wiener và pass: peter như đã cho

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Sau khi nhập thành công sẽ ra giao diện như thế

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Sử dụng phần mềm để xem các gói tin Http và xem gói phản hồi của login ta thấy có 1 trường là set-cookie

+ Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Ta thử decode phần set-cookie

+ Từ kết quả ta thấy rằng session nacookie này được tạo từ 1 User object với username là wiener và admin=false

+ Vậy để có thể leo thang đặc quyền ta có thể set cookie admin=1 và thực hiện decode lại để tái tạo base64

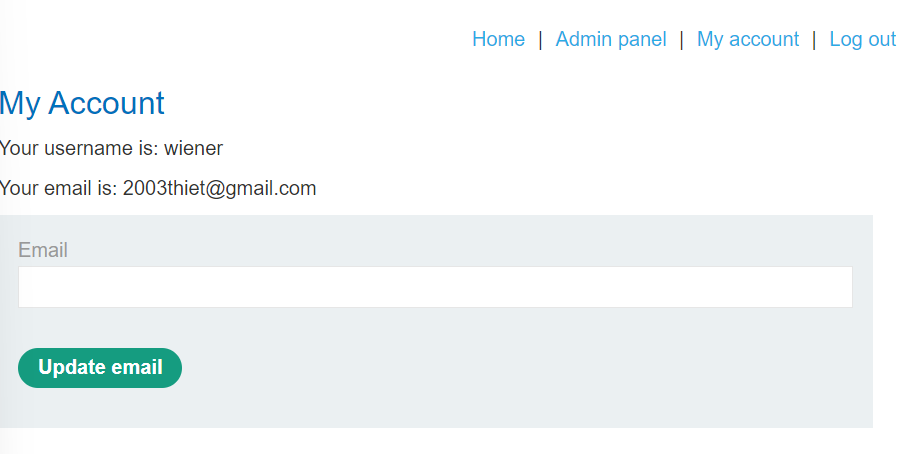
Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Sau đó ta dùng cookie và lấy được và thay vào chỗ session



+ Ta thấy panel xuất hiện 1 trường là admin panel

+ Tiếp tục thực hiện với cookie admin ta vào Admin panel

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Đã giải thành công

* Insecure Deserialization trong WebGoat

Ảnh có chứa văn bản

Mô tả được tạo tự động

+Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Ta thử copy chuỗi đã cho và được 1 kết quả trả về như trên

+ Thấy được là chương trình sẽ deserialize và tạo VulnerableTaskHolder object. Ta xem lại lesson 3

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Làm tương tự câu Java Deserialize ta dựng lại code exploit như sau

Ảnh có chứa văn bản

Mô tả được tạo tự động

Ảnh có chứa văn bản

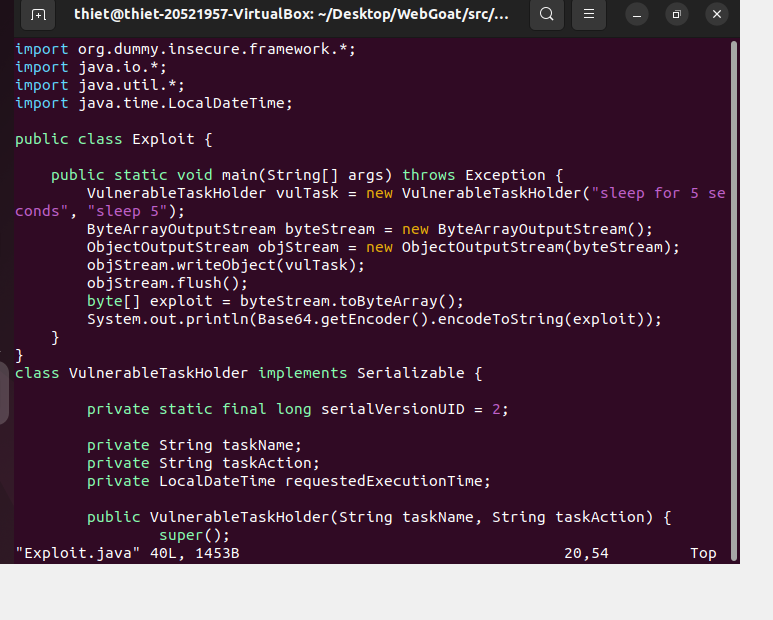
Mô tả được tạo tự động+ Khi chạy ta được payload sau

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Nhưng khi ta submit thì lại thông báo như trên

* Do Code Vulnerable của WebGote thì SerialVersionUID = 2 nên ta sửa lại giá trị này
* Class của VulnerableTaskHolder khi compile sẽ được đóng gói trong packge org.dummy.insrcure.freamework. Vậy nên ta clone sorce của webgoat về và tạo file Exploit.java tại path WebGoat/webgoat-lessons/insecure-deserialization/src/main/java bởi vì thư mục này là điểm bắt đầu của package org.dummy.insecure.framework



Ảnh có chứa văn bản

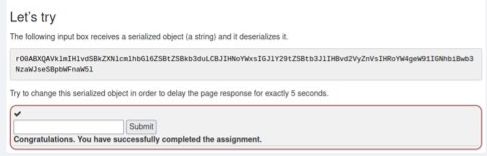
Mô tả được tạo tự động

+ Code Exploit mới

Ảnh có chứa văn bản

Mô tả được tạo tự động

+ Ta có payload mới



* TÀI LIỆU THAM KHẢO