BỘ THÔNG TIN VÀ TRUYỀN THÔNG

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

------o0o------

BÁO CÁO NỘI DUNG 3

XÂY DỰNG LAB THỰC HÀNH

FORMAT64_2 – Nhóm 08

Môn học: Chuyên đề an toàn phần mềm

Giảng viên: ThS. Ninh Thị Thu Trang

Thành viên: Phạm Thị Thu Hương – B19DCAT098

Đỗ Đức Quốc Anh – B19DCAT003

Đoàn Việt Hưng – B19DCAT094

Bùi Thanh Phong – B19DCAT135

Hà Nội - 2023

MỤC LỤC

DANH MỤC CÁC HÌNH VẼ VÀ BẢNG

I. Nội dung và hướng dẫn thực hiện bài thực hành

1. Mục đích

- Giúp sinh viên hiểu về lỗ hổng bảo mật format string thông qua việc thực hiện tấn công dịch vụ chứa lỗi.

2. Yêu cầu đối với sinh viên

- Có kiến thức cơ bản về hệ điều hành Linux, mô hình mạng khách chủ
- Có kiến thức cơ bản về ngôn ngữ assembly và C/C++
- Có kiến thức cơ bản về lỗ hổng bảo mật format string

3. Nội dung thực hành

- Khởi động bài lab: *labtainer – r ptit-format64_2*

a. Find buffer index

Mục đích: Tìm được index trỏ đến đầu buffer

Các bước thực hiện:

- Thực hiện debug tiến trình
- Tiến hành tạo một khối với nội dung bất kì
- Chỉnh sửa khối để trigger được bug
- Output sẽ in ra giá trị trên stack
- Tìm phần tử %p tương ứng khi thấy bắt đầu có sự lặp lại các giá trị in ra
- Sử dụng Ctrl-C để thoát tiến trình debug

b. Leak

Mục đích: Leak được địa chỉ trên stack

Các bước thực hiện:

- Thực hiện debug lại chương trình:
- Tiến hành nhập nội dung tạo một khối:
- Tiến hành leak thông tin sử dụng hàm modify:
- Output thu được theo payload trên là giá trị hex là địa chỉ của _IO_2_1_stdout_ trong libc
- Sinh viên có thể tự do chọn giá trị leak khác trên stack để thực hiện bài

c. VMMAP

Mục đích: Tính địa chỉ base address

Các bước thực hiện:

- Tại cửa số đang debug thực hiện câu lệnh vmmap để in ra thông tin phân vùng bộ nhớ
- Địa chỉ ở cột start tại dòng đầu tiên trỏ tới libc là địa chỉ base của libc
- Lưu lại giá trị này để tiến hành thực hiện tính offset cho hàm system ở bước sau
- Thực hiện tính offset của giá trị vừa leak ở bước trên bằng cách lấy giá trị đó trừ đi giá trị của libc base address vừa tìm được

d. System address

Mục đích: Tìm địa chỉ system để tính system offset

Các bước thực hiện:

- Thực hiện in ra địa chỉ của hàm system
- Tiến hành lấy địa chỉ system trừ đi địa chỉ libc base vừa tìm được bên trên để tìm được offset của system

e. Got address

Mục đích: Tìm được địa chỉ của got để ghi đè vào

Các bước thực hiện:

- Từ cửa sổ debug hiện tại chạy lệnh got
- Lưu lại địa chỉ got của hàm free (được đặt trong cặp '[ ]'
- Thực hiện thoát cửa sổ debug

f. Overwrite got

Mục đích: Ghi đè got bằng system

Các bước thực hiện:

- Thực hiện chỉnh sửa payload để thực hiện ghi đè free@got.plt bằng system
- Tạo 1 khối bất kì sau đó xóa đi để thực hiện resolve địa chỉ cho hàm free
- Tạo khối mới dùng để trigger bug với content bất kì
- Tiến hành input để leak dữ liệu
- Tính lấy địa chỉ leak được trừ đi offset1 đã tìm ở trên để lấy được địa chỉ base
- Cộng địa chỉ base với offset của hàm system đã tính ở trên để ra được địa chỉ của hàm system
- Sau khi tìm được các địa chỉ offset1, offset_system, free_got tiến hành thay vào payload sao đó chạy file solve.py

- Khi pwnserver đã bắt được tiến trình debug sử dụng lệnh continue để tiếp tục
- Chạy lệnh got để thấy được địa chỉ của free lúc này đã được thay bằng địa chỉ của hàm system

g. Secret

Mục đích: Tìm được nội dung flag được giấu trên server

Các bước thực hiện:

- Thay đổi payload để thực hiện tấn công lên server
- Đọc file secret sau đó copy chuỗi số bí mật
- Thoát khỏi chương trình sau đó submit flag tại cửa sổ terminal của attacker echo “flag <secret number>”

II. Phân tích, thiết kế bài thực hành

1. Phân tích yêu cầu bài thực hành

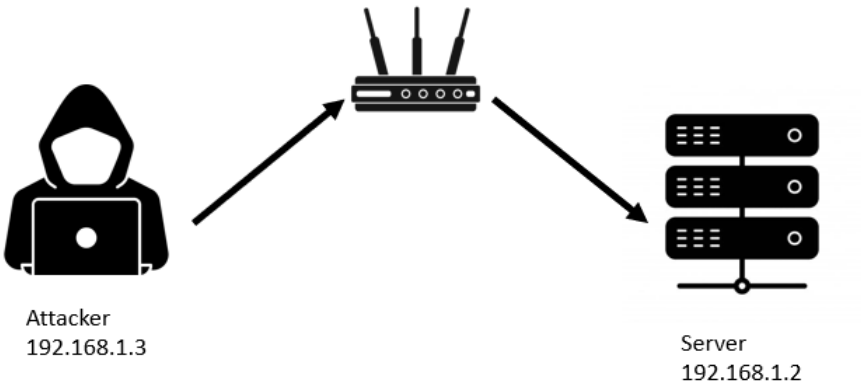
Bài thực hành cần có ba máy tính, trong đó có hai máy tính nằm trong cùng mạng LAN, một máy tính đóng vai trò là công cụ hỗ trợ. Cụ thể như sau, trong mạng LAN gồm có 2 máy, máy attacker là nơi mà sinh viên sẽ tương tác chủ yếu nhằm khai thác thành công binary, máy server sẽ chạy service là binary mà sinh viên cần khai thác, máy ảo còn lại là công cụ hỗ trợ decompile file binary. Để hoàn thành bài thực hành, sinh viên cần sử dụng máy attacker tiến hành khai thác thành công lỗ hổng buffer overflow trên máy chủ để lấy được chuỗi số bí mật.

Để đáp ứng yêu cầu bài thực hành, cần cung cấp 3 container docker. Trong đó, một container đóng vai trò là server, một container đóng vai trò là attacker, container còn lại đóng vai trò là công cụ hỗ trợ, sinh viên có thể dùng đến hoặc không (trong trường hợp sinh viên muốn dùng công cụ decompiler trực tiếp trên máy attacker). Hệ thống cần ghi lại được thao tác sử dụng các lệnh trên terminal container của sinh viên để tạo ra được kết quả đánh giá. Hệ thống yêu cầu sinh viên nhập email gắn liền với danh tính của sinh viên để thực hiện việc cá nhân hóa cho từng sinh viên.

Để bắt đầu bài thực hành sinh viên cần phải sử dụng các câu lệnh khởi tạo (labtainer -r <tên bài lab>) và câu lệnh kết thúc (stoplab <tên bài lab>) để hệ thống chạy bài lab cũng như lưu lại kết quả.

2. Thiết kế bài thực hành

Trên môi trường máy ảo Ubuntu được cung cấp, sử dụng docker tạo ra 3 container: 1 container mang tên "format642" đóng vai trò là attacker và 1 container mang tên "server" đóng vai trò là server, 2 máy đều được mở các cổng cần thiết. Hình 1.1 mô tả sơ đồ thiết kế bài thực hành.



Hình 1. Sơ đồ thiết kế bài thực hành

- Tạo mạng LAN "TEST" có cấu hình: 192.168.1.0/24 và gateway 192.168.1.1
- Cấu hình docker gồm có:
    - Buf64_2: Lưu cấu hình cho máy attacker, trong đó gồm có:
        - Tên máy: attacker
        - Địa chỉ trong mạng LAN: 192.168.1.3
        - Gateway: 192.168.1.1
    - Server: Lưu cấu hình cho máy server, trong đó gồm có:
        - Tên máy: server
        - Địa chỉ trong mạng LAN: 192.168.1.2
        - Gateway: 192.168.1.1
    - Ghidra: sử dụng để decompile binary:
        - Tên máy: ghidra-vm
        - Địa chỉ trong mạng LAN: N/A
    - Config: Lưu cấu hình hoạt động của hệ thống
    - Dockerfiles: Mô tả cấu hình của 3 container: attacker, ghidra và server, trong đó:

        - Attacker sử dụng base image cần cài thêm netcat và git để setup tools
        - Server sử dụng network base image cần cài thêm rsync để setup service xinetd
        - Ghidra sử dụng ghidra base image không cần config gì thêm
    - Docs: Lưu phần mô tả hướng dẫn làm bài thực hành cho sinh viên
    - Instr_config: Lưu cấu hình cho phần nhận kết quả và chấm điểm
- Thiết lập hệ thống mạng sao cho 2 container cùng một mạng LAN.
- Để đánh giá được sinh viên đã hoàn thành bài thực hành hay chưa, cần chia bài thực hành thành các nhiệm vụ nhỏ, mỗi nhiệm vụ cần phải chỉ rõ kết quả để có thể đưa vào đó đánh giá, chấm điểm. Do vậy, trong bài thực hành này hệ thống cần ghi nhân các thao tác, sự kiện được mô tả và cấu hình như bên dưới:

  _choice_modify = format64_2:.stdout : CONTAINS : > M*

  _format_content = format64_2:.stdout : CONTAINS : %p*

  _leak1 = format64_2:.stdout : FILE_REGEX : RIP.*0x400ed6*

  vmmap = format64_2:.stdout : CONTAINS : ld-2.31.so*

  system_address = format64_2:.stdout : CONTAINS : __libc_system*

  got_address = format64_2:.stdout : CONTAINS : GOT entries passing the filter*

  _overwrite_got1 = format64_2:.stdout : CONTAINS : 0x6020*

  _overwrite_got2 = format64_2:.stdout : CONTAINS : 290 (system)*

  _echo = format64_2:echo.stdout : TOKEN : LAST : STARTSWITH : flag

  _leak2 = format64_2:.stdout : FILE_REGEX : RDI.*%\d+\$p*
- Sau khi có đầu vào thao tác, tiến hành kiểm tra để cho ra kết quả cuối ‘Goal result’:

  find_buffer_idx = boolean : (_choice_modify and _format_content)

  *secret = matchany : string_contains : _echo : parameter.secret_*

  leak = boolean : (_leak1 and _leak2)

  overwrite_got = boolean : (_overwrite_got1 and _overwrite_got2)
- Sau khi nhận được file đóng gói từ sinh viên, giảng viên sử dụng chức năng chấm điểm để xem kết quả được thiết kế dưới dạng bảng trong đó có ghi rõ

email của sinh viên thực hiện, từng tiêu chí chấm điểm được ghi nhận (ví dụ: 'Y' là đã hoàn thành, nếu không có là chưa hoàn thành) và kết luận là sinh viên đã hoàn thành bài thực hành đó hay chưa. Kiểm tra bài thực hành đúng do sinh viên làm bằng cách kiểm tra email (xem bảng 2).

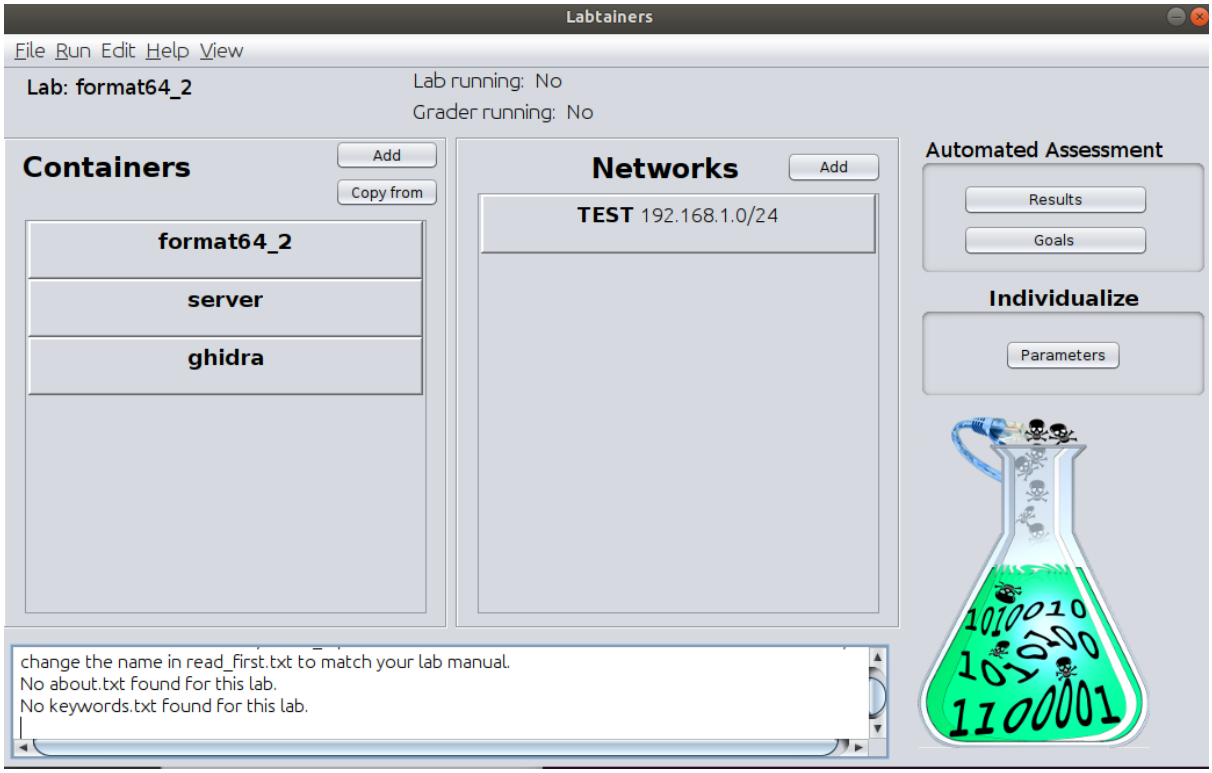
Bảng 1. Kết quả chấm điểm

Student	**find_buffer_idx**	**secret**	**leak**	**overwrite_got**	**vmmap**	**system_address**	**got_address**
Mã sinh viên	Y	Y	Y	Y	Y	Y	Y

Y: đã hoàn thành

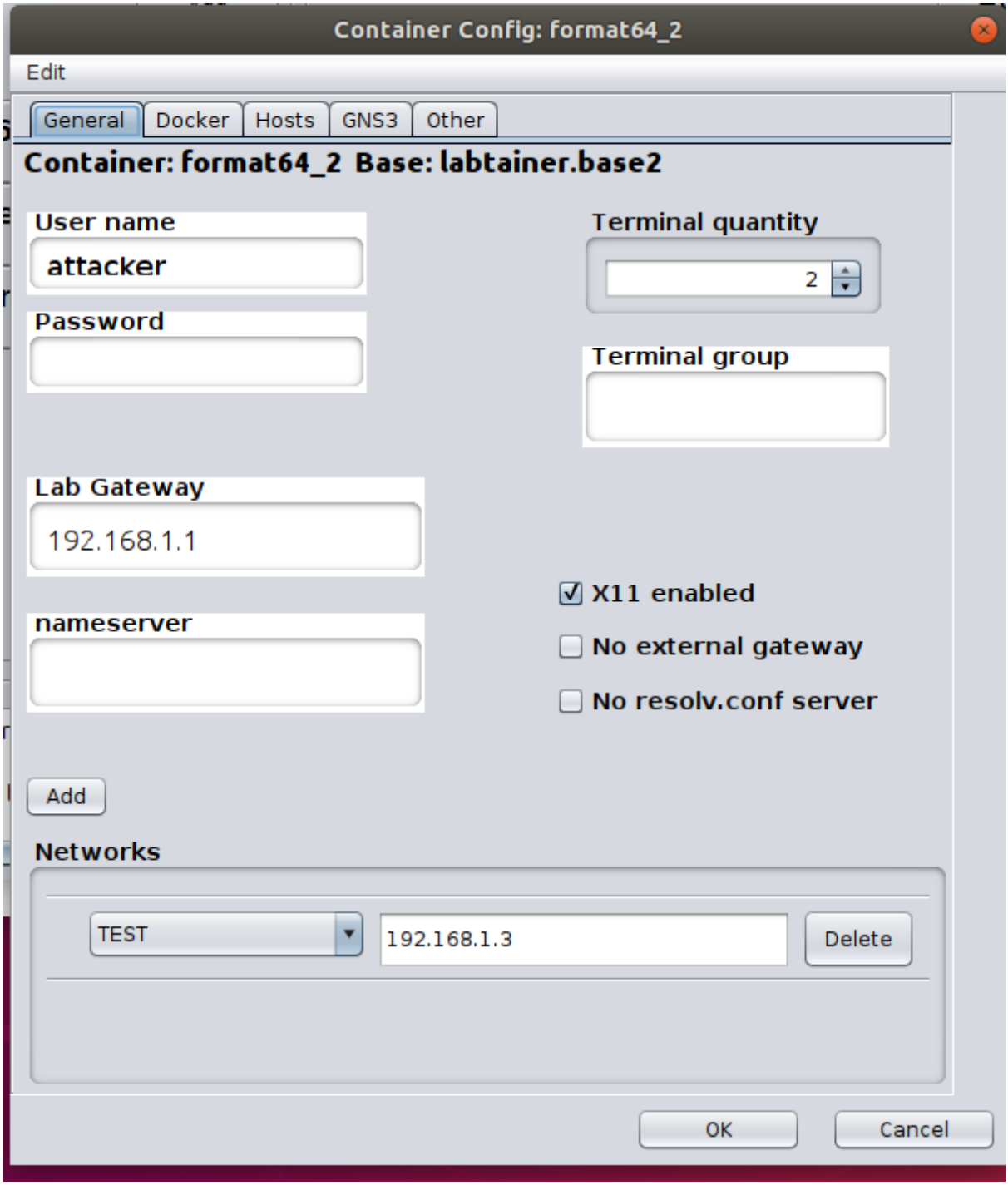
III. Cài đặt và cấu hình các máy ảo

- Cần chạy file update-designer.sh để cập nhật bản mới nhất của labedit
- Từ đường dẫn bất kì trên terminal gõ lệnh labedit
- Thực hiện tạo lab mới: File -> New Lab và đặt tên cho bài lab



Hình 2. Tạo lab với tên format64_2

- Sau khi lab được khởi tạo, tạo 3 container với base image lần lượt là: base2, network2, ghidra

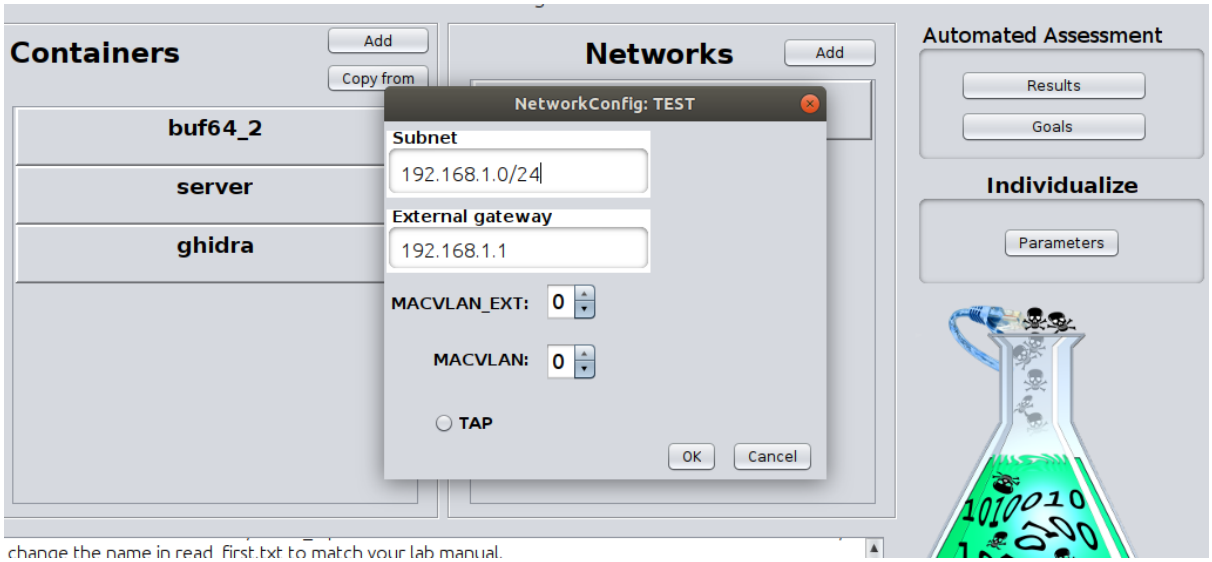


Hình 3. Tạo container attacker

Hình 4. Tạo container server

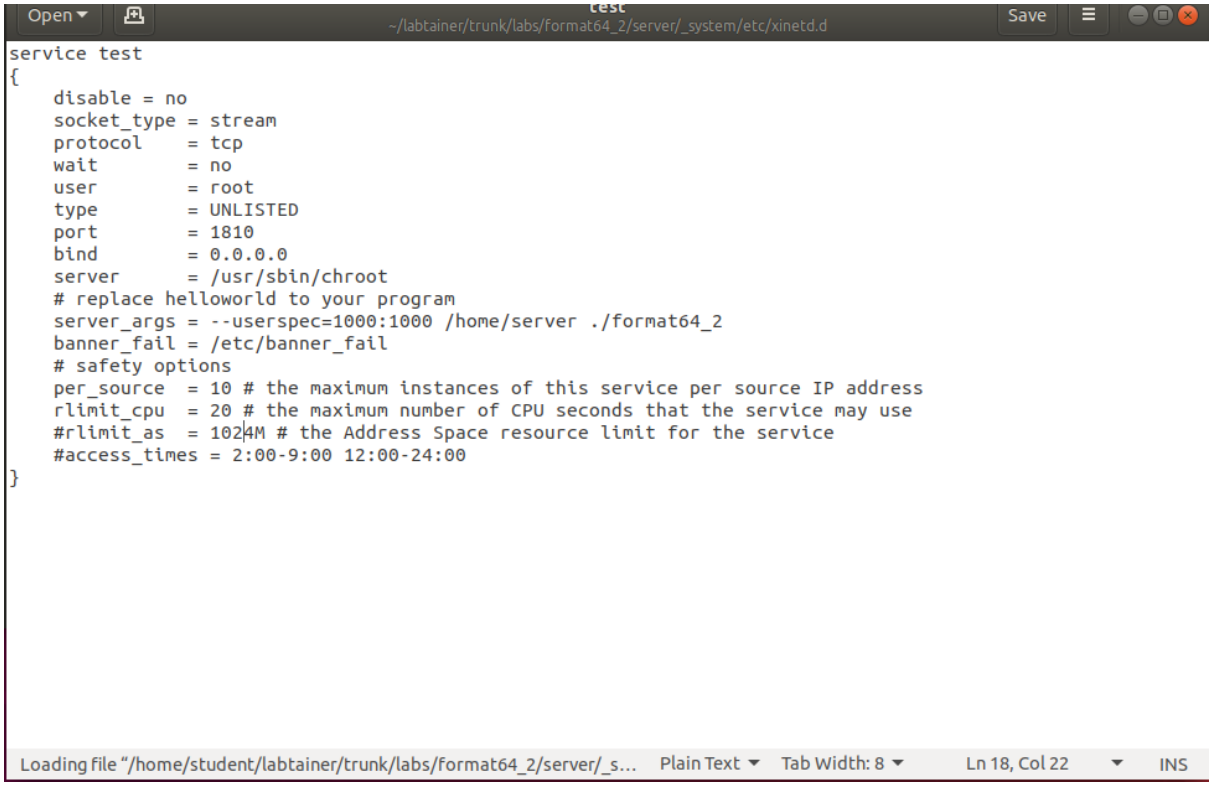
Hình 5. Tạo container ghidra

- Tạo networks mới cho lab, đặt tên network, cài đặt giải mạng con và gateway cho mạng



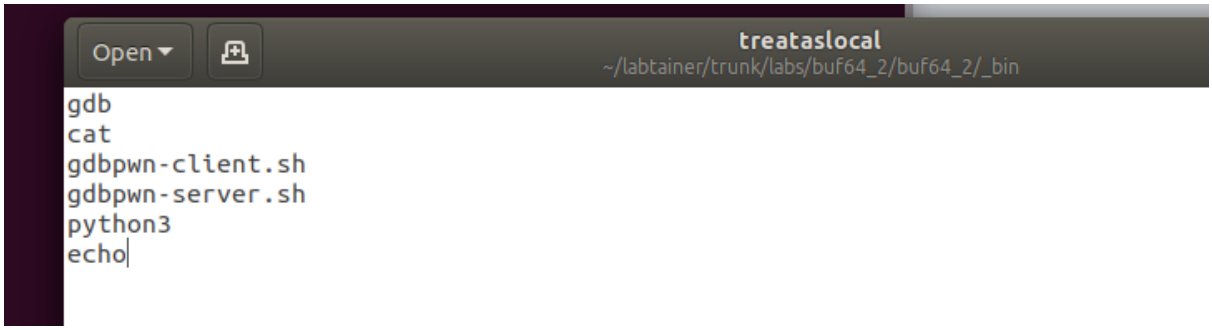
Hình 6. Tạo mạng mới cho lab

- Cấu hình mạng cho Attacker và Server:
    - Attacker:
        - IP: 192.168.1.3/24
        - GW: 192.168.1.1
    - Server:
        - IP 192.168.1.2/24
        - GW: 192.168.1.1
        - Service port: 1810
- Cài đặt thêm công cụ pwntools và extension pwndbg cho container attacker
- Cài đặt service binary cho container server



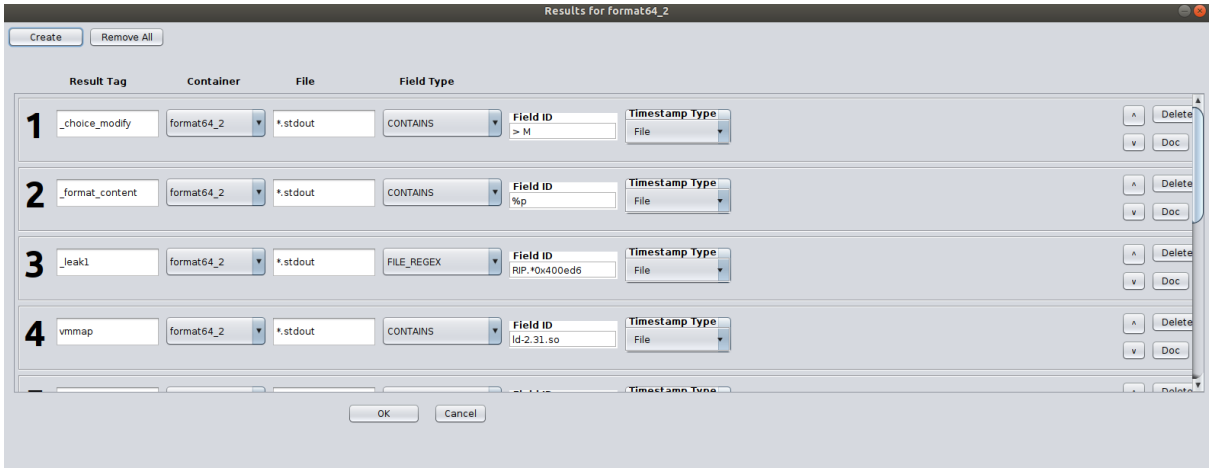
Hình 7. Cài đặt service binary tại port 1810

- Chỉnh sửa file treataslocal để tạo task đánh giá cho từng container bằng cách chỉnh sửa file treataslocal từ labedit hoặc chỉnh sửa trực tiếp theo đường dẫn ~/labtainer/trunk/labs/<tên-lab>/<tên-container>/_bin/treataslocal



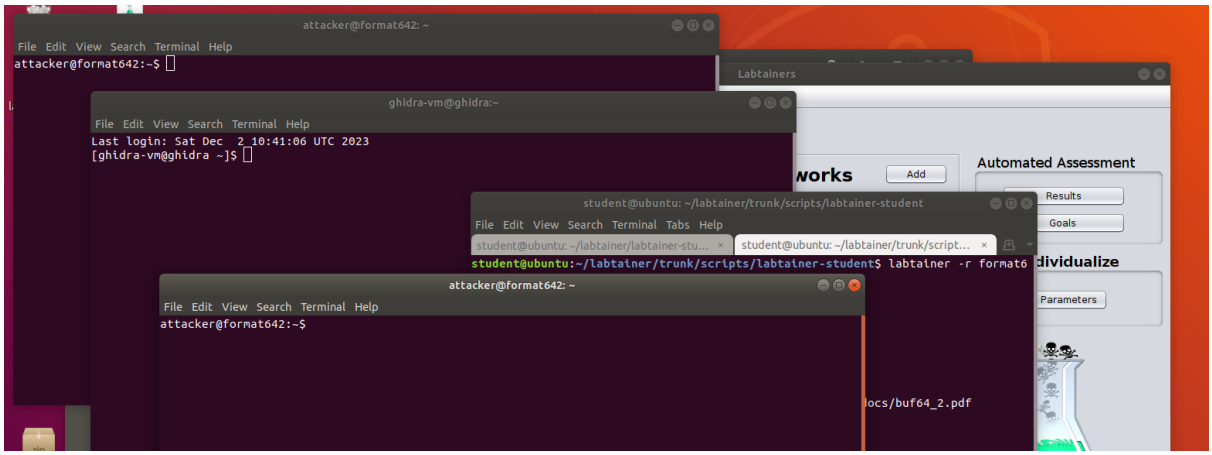
Hình 8. Chỉnh sửa file treataslocal trên máy attacker

- Cài đặt Results ở phần Automated Assesment:



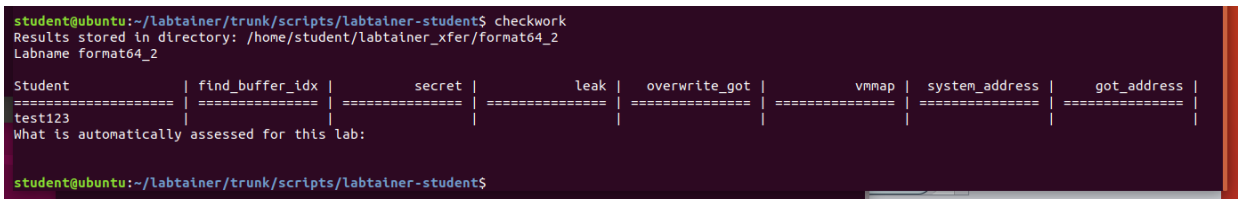
Hình 9. Cài đặt chấm điểm tự động

- Tiến hành save lab, sau đó build và chạy thử lab

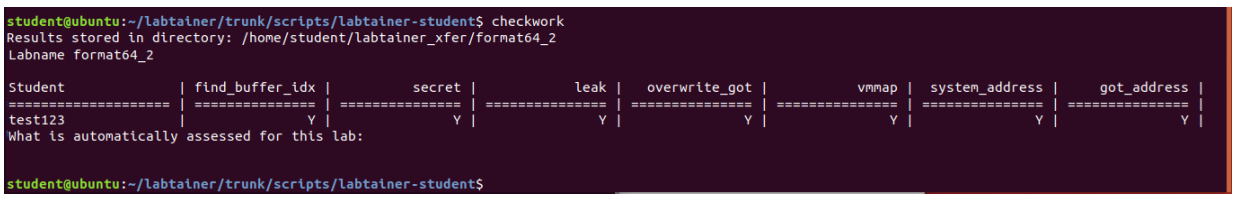


Hình 10. build lab và chạy bài lab

IV.Thử nghiệm lab và kết quả



Hình 11. Run lab và thực hiện check work trước khi thực hiện bài lab



Hình 12. Thực hiện bài lab sau đó check work

V. Triển khai bài lab

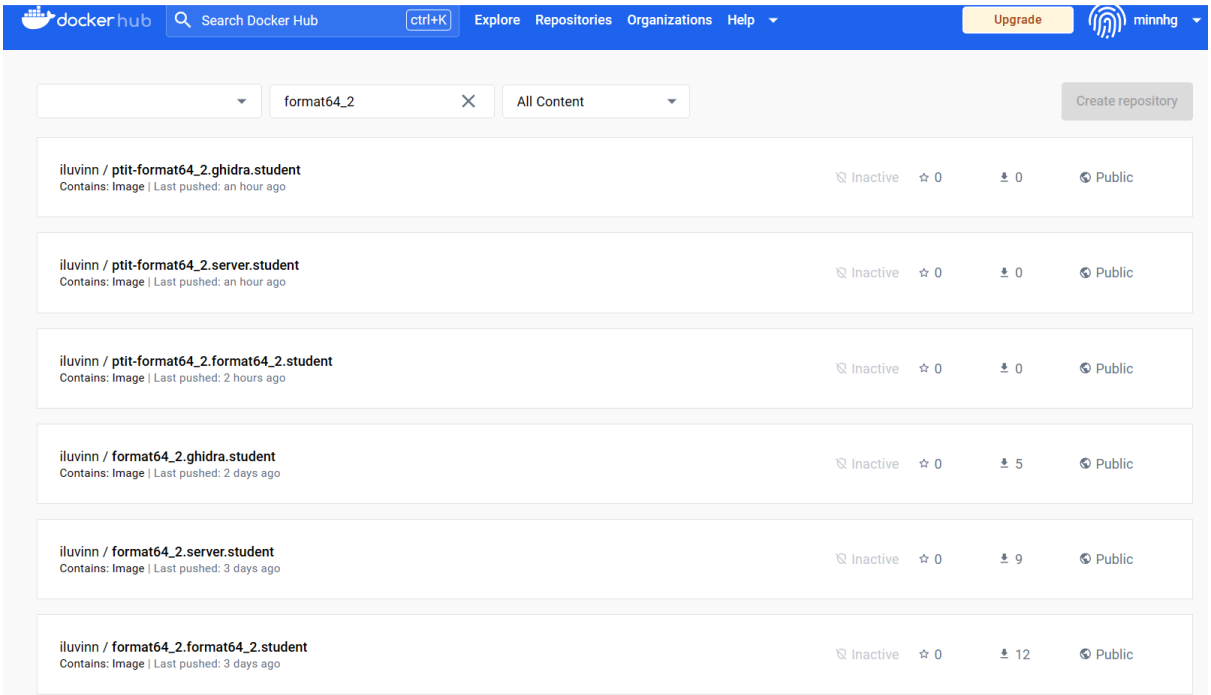
- Chuyển tới thư mục chứa các bài thực hành: **labtainer/trunk/labs**

- Khởi tạo git: **git init** (chỉ khởi tạo một lần, không lặp lại với mỗi lần).
- Lấy tên của Docker Hub để đăng ký cho registry bài lab mới ở config/start.config (tại Labtainers GUI: Edit / Config (registry))
- Trong đường dẫn thư mục của bài lab bai-lab-moi. Chạy cleanlab4svn.py để xóa những files tạm.
- Sau đó trong đường dẫn cha của bài lab:
  git add <tên bài lab> git commit <tên bài lab> -m
  "Adding an IModule"

```
File  Edit  View  Search  Terminal  Help
student@ubuntu:~/labtainer/trunk/labs$ rm -fr .git
student@ubuntu:~/labtainer/trunk/labs$ clear
student@ubuntu:~/labtainer/trunk/labs$ git init
Initialized empty Git repository in /home/student/labtainer/trunk/labs/.git/
student@ubuntu:~/labtainer/trunk/labs$ git add ptit-buf64_2
student@ubuntu:~/labtainer/trunk/labs$ git add ptit-buf64_3
student@ubuntu:~/labtainer/trunk/labs$ git add ptit-format64_2
student@ubuntu:~/labtainer/trunk/labs$ git add ptit-format64_3
student@ubuntu:~/labtainer/trunk/labs$ git add ptit-fastbin
student@ubuntu:~/labtainer/trunk/labs$ git commit -m "add n08 labs"
[master (root-commit) 198181d] add n08 labs
 235 files changed, 7106 insertions(+)
```

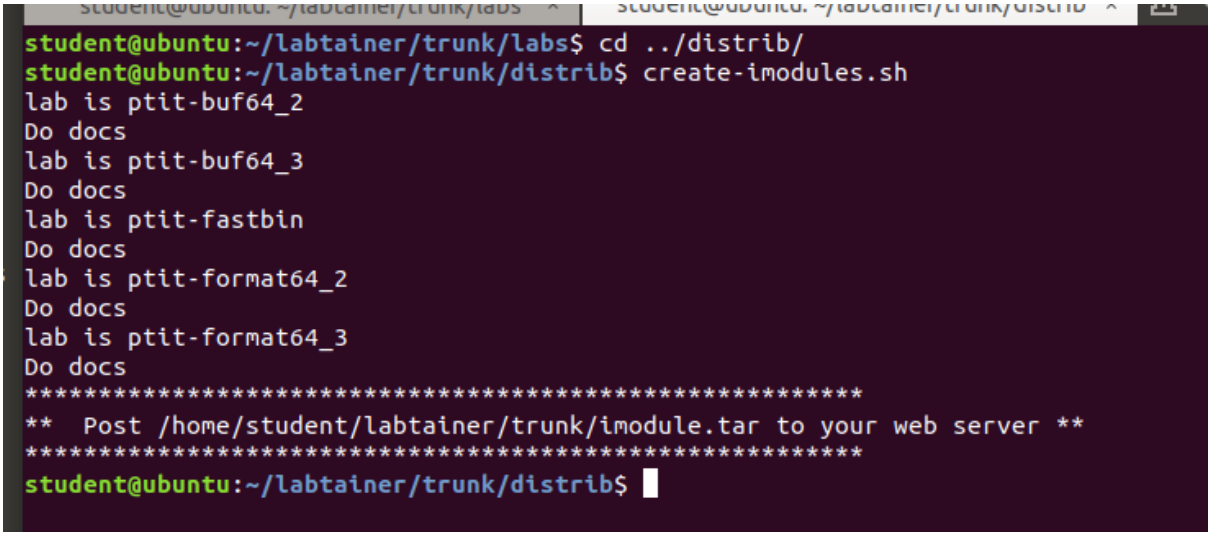
Hình 13. Đẩy bài lab lên git

- Đẩy images của vùng chứa (container) lên DockerHub
  cd $LABTAINER_DIR/distrib
  ./publish.py -d -l my-new-lab



Hình 14. Các images của vùng chứa được đẩy lên DockerHub

- Tạo file IModule tar chứa bài thực hành: create-imodules.sh



Hình 15. Tạo file Imodule tra chứa bài thực hành

- Sau đó, copy và lưu lại file imodule.tar. Đường dẫn URL vào link imodule: https://github.com/iluvinn/cdatpm-ptit.git

TÀI LIỆU THAM KHẢO

Labtainer Lab Designer User Guide