

BỘ THÔNG TIN VÀ TRUYỀN THÔNG
HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

-----o0o-----



BÁO CÁO NỘI DUNG 2
XÂY DỰNG LAB THỰC HÀNH
BUF64_3 – 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

MỤC LỤC	1
DANH MỤC CÁC HÌNH VẼ VÀ BẢNG.....	2
I. Nội dung và hướng dẫn thực hiện bài thực hành.....	3
1. Mục đích.....	3
2. Yêu cầu đối với sinh viên.....	3
3. Nội dung thực hành	3
II. Phân tích, thiết kế bài thực hành	5
1. Phân tích yêu cầu bài thực hành.....	5
2. Thiết kế bài thực hành	6
III. Cài đặt và cấu hình các máy ảo	8
IV. Thử nghiệm lab và kết quả	16
V. Triển khai bài lab.....	16
TÀI LIỆU THAM KHẢO	17

DANH MỤC CÁC HÌNH VẼ VÀ BẢNG

Hình 1. Sơ đồ thiết kế bài thực hành	6
Hình 2. Giao diện labedit.....	8
Hình 3. Tạo lab với tên buf64_3.....	9
Hình 4. Tạo container attacker	10
Hình 5. Tạo container server	11
Hình 6. Tạo container ghidra.....	12
Hình 7. Tạo mạng mới cho lab	13
Hình 8. Cài đặt service binary tại port 1810.....	14
Hình 9. Chỉnh sửa file treataslocal trên máy attacker	14
Hình 10. Cài đặt chấm điểm tự động.....	15
Hình 11. Build lab và chạy bài lab	15
Hình 12. Run lab và thực hiện check work trước khi thực hiện bài lab.....	16
Hình 13. Thực hiện bài lab sau đó check work	16
Hình 14. Đẩy bài lab lên git.....	16
Hình 15. Các images của vùng chứa được đẩy lên DockerHub.....	17
Hình 16. Tạo file Imodule tra chứa bài thực hành	17

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

1. Mục đích

- Giúp sinh viên hiểu về lỗ hổng bảo mật buffer overflow 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 buffer overflow
- Lý thuyết về calling convention
- Lý thuyết về thanh ghi

3. Nội dung thực hành

- Khởi động bài lab: `labtainer -r ptit-buf64_3`

a. Found pattern

Mục đích: Tìm được độ dài pattern gây ra lỗi

Các bước thực hiện:

- Thực hiện sinh chuỗi pattern làm đầu vào bằng cách sử dụng lệnh trên cửa sổ terminal của attacker. Output của câu lệnh trên sẽ là chuỗi pattern có độ dài 200, copy chuỗi trên làm input để debug
- Tiến hành chạy trình debug để debug file binary một lần nữa
- Paste pattern đã gen khi tiến trình yêu cầu nhập input
- Sau khi ấn enter, chương trình sẽ báo lỗi segmentation fault
- Copy 4 ký tự đầu của thanh ghi RSP ở phần REGISTERS
- Thoát tiến trình debugger:
- Tìm độ dài pattern gây ra lỗi. Output sẽ là độ dài của pattern được sử dụng để tạo payload

b. RSI gadget

Mục đích: Tìm được địa chỉ của gadget pop rsi

Các bước thực hiện:

- Thực hiện list toàn bộ gadget của binary vào file
- Tìm kiếm gadget liên quan đến RSI

- Lấy địa chỉ gadget liên quan đến pop rsi và có lệnh ret ở cuối

Task 2 quest: Tại sao cần tìm địa chỉ của read@plt mà không dùng trực tiếp địa chỉ câu lệnh call read trong hàm vuln?

c. RDI gadget

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

Các bước thực hiện:

- Tìm kiếm gadget liên quan đến RDI
- Lấy địa chỉ gadget liên quan đến pop rdi và có lệnh ret ở cuối

Task3 quest 1: Tại sao lại cần thêm 8 bytes '0x00' (p64(0)) trước địa chỉ của read@plt trong payload

Task 3 quest 2: Giải thích đầy đủ luồng hoạt động của chương trình và payload khi bug được trigger?

d. VM MAP

Mục đích: Tìm phân vùng chứa quyền read/write phù hợp trên memory

Các bước thực hiện:

- Thực hiện debug tiến trình
- Chạy lệnh và lấy địa chỉ của hàm read@plt
- Chạy lệnh và lấy địa chỉ của call system:
- Đặt breakpoint tại main
- Chạy chương trình và thực hiện chạy lệnh vmmap để liệt kê các phân vùng bộ nhớ
- Tại phân vùng dành cho bss có quyền read/write của binary (cột perm) tìm ô địa chỉ phù hợp để chọn ghi dữ liệu và không làm ảnh hưởng đến các giá trị khác lưu trữ trên phân vùng đó

e. SECRET

Mục đích: lấy được giá trị bí mật trên server

Các bước thực hiện:

- Thực hiện chỉnh sửa file payload
- Luồng chương trình mong muốn khi payload được thực thi:

- Chương trình sẽ bị điều khiển luồng thực thi để thực thi hàm read cho phép nhập dữ liệu đầu vào
- Sau khi thoát khỏi hàm read tiến hành gọi hàm system với đầu vào là địa chỉ đã nhập vào từ hàm read
- Sau khi gửi payload chương trình thực thi và cho phép ta nhập thêm input một lần nữa (hàm read được thực thi và đang chờ đọc dữ liệu) khi này câu lệnh thực thi bởi system là input được nhập vào
- Để có thể thực thi thành công sinh viên cần hiểu về calling convention trong kiến trúc 64-bit
- Payload thực thi hàm read
- Payload thực thi hàm system
- Thực hiện tương tự việc debug tại local như với lab buf64_2 để xác nhận rằng payload hoạt động tốt
- Thay đổi payload để tấn công lên server
- Sau khi thực thi payload thành công sinh viên sẽ có một interactive shell để tương tác, thực hiện đọc nội dung file .secret
- Sao chép số bí mật và submit tại cửa sổ của attacker:

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.

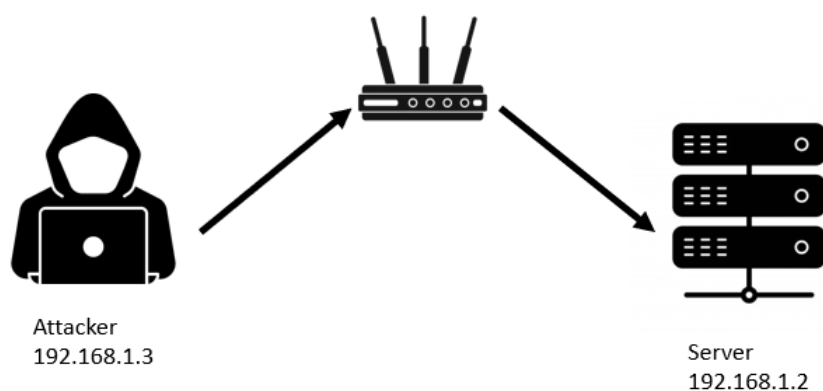
Để đáp ứng yêu cầu bài thực hành, cần cung cấp 3 container docker. Trong đó, một container đóng vai trò là server, một container đóng vai trò là attacker, container còn lại đóng vai trò là công cụ hỗ trợ, sinh viên có thể dùng đến hoặc không (trong trường hợp sinh viên muốn dùng công cụ decompiler trực 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 “buf643” 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 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_3: 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 nhận các thao tác, sự kiện được mô tả và cấu hình như sau:


```
_crash = buf64_3:*.stdout : CONTAINS : Program received signal SIGSEGV, Segmentation fault. _stack_pattern = buf64_3:*.stdout : FILE_REGEX : RSP.*saaataaa _find_pattern = buf64_3:*.stdout : CONTAINS : 72 rsi_gadget = buf64_3:*.stdout : CONTAINS : 400841 rdi_gadget = buf64_3:*.stdout : CONTAINS : 400843 _echo = buf64_3:echo.stdout : TOKEN : LAST : STARTSWITH : flag mmap = buf64_3:*.stdout : CONTAINS : 0x602000 rw
```
- Sau khi có đầu vào thao tác, tiến hành kiểm tra để cho ra kết quả cuối ‘Goal result’ như sau:


```
_crash_goal = boolean : (_crash and _stack_pattern) found_pattern = boolean : (_crash_goal and _find_pattern) secret = matchany : string_contains : echo : parameter.secret
```
- 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.

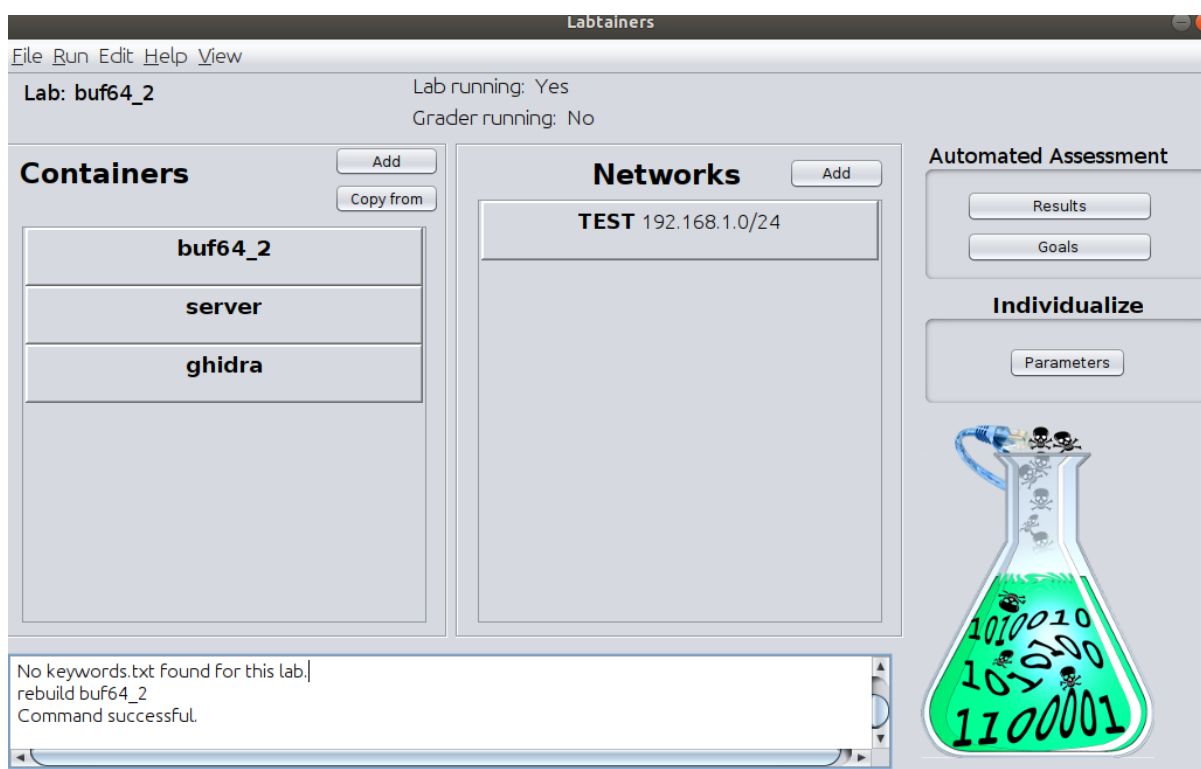
Bảng 1. Kết quả chấm điểm

Student	found_pattern	rsi_gadget	rdi_gadget	secret	vmmmap
Mã sinh viên	Y	Y	Y	Y	Y

Y: đã hoàn thành

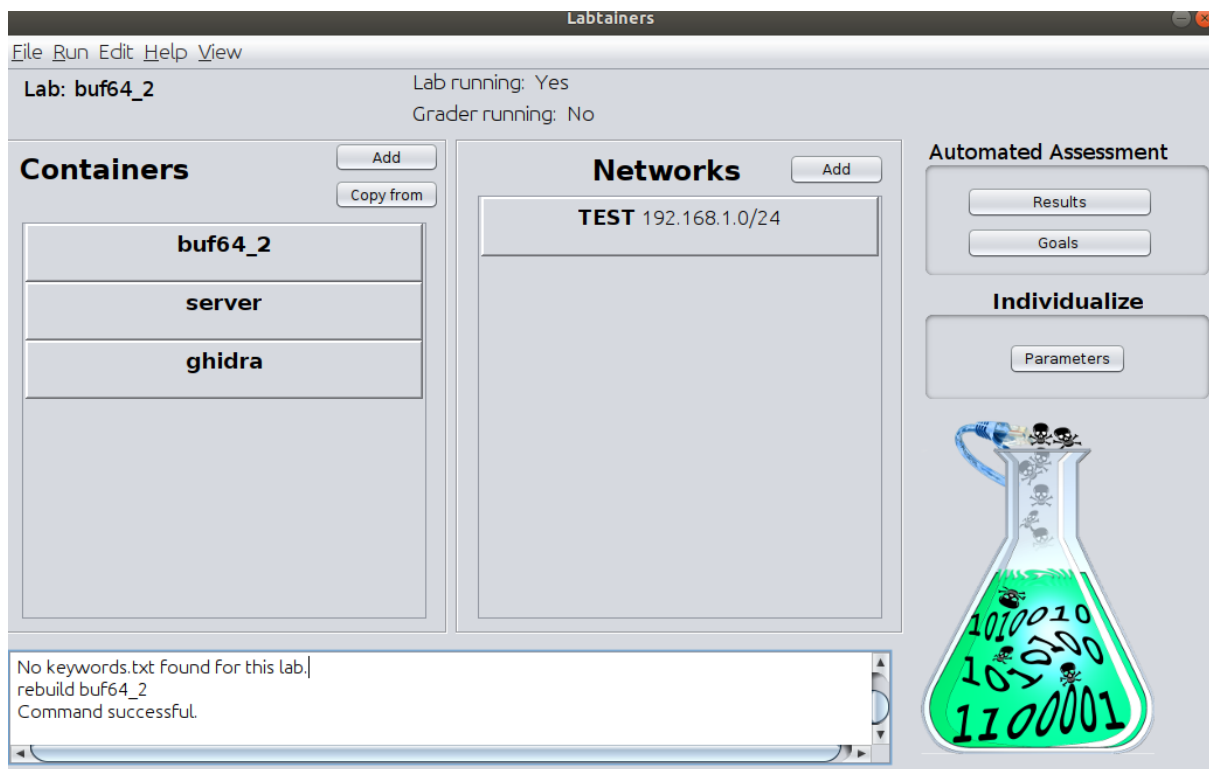
III. Cài đặt và cấu hình các máy ảo

- Cần chạy file update-designer.sh để cập nhật bản mới nhất của labedit
- Từ đường dẫn bất kì trên terminal gõ lệnh labedit, sau đó màn hình sẽ hiển thị giao diện như hình 2:



Hình 2. Giao diện labedit

- Thực hiện tạo lab mới: File -> New Lab và đặt tên cho bài lab



Hình 3. Tạo lab với tên buf64_3

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

Container Config: buf64_2

Edit

General Docker Hosts GNS3 Other

Container: buf64_2 Base: labtainer.base2

User name
attacker

Password

Terminal quantity
2

Terminal group

Lab Gateway
192.168.1.1

nameserver

☒ X11 enabled
☐ No external gateway
☐ No resolv.conf server

Add

Networks

Network Name	IP Address	Action
TEST	192.168.1.3	Delete

OK Cancel

Hình 4. Tạo container attacker

Container Config: server

Edit

General Docker Hosts GNS3 Other

Container: server Base: labtainer.network2

User name
server

Password
supersecretbassword

Terminal quantity
0

Terminal group

Lab Gateway
192.168.1.1

nameserver

☒ X11 enabled
☐ No external gateway
☐ No resolv.conf server

Add

Networks

Network	IP	Action
TEST	192.168.1.2	Delete

OK Cancel

Hình 5. Tạo container server

Container Config: ghidra

Edit

General Docker Hosts GNS3 Other

Container: ghidra Base: labtainer.ghidra

User name
ghidra-vm

Password
1

Terminal quantity
1

Terminal group

Lab Gateway

nameserver

☒ X11 enabled
☐ No external gateway
☐ No resolv.conf server

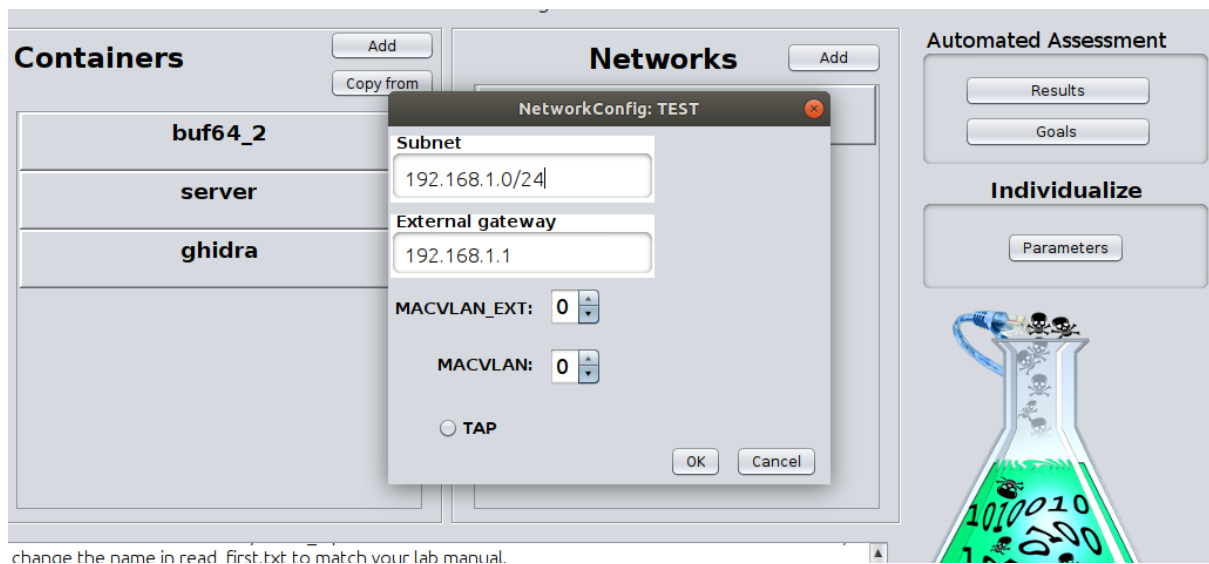
Add

Networks

OK Cancel

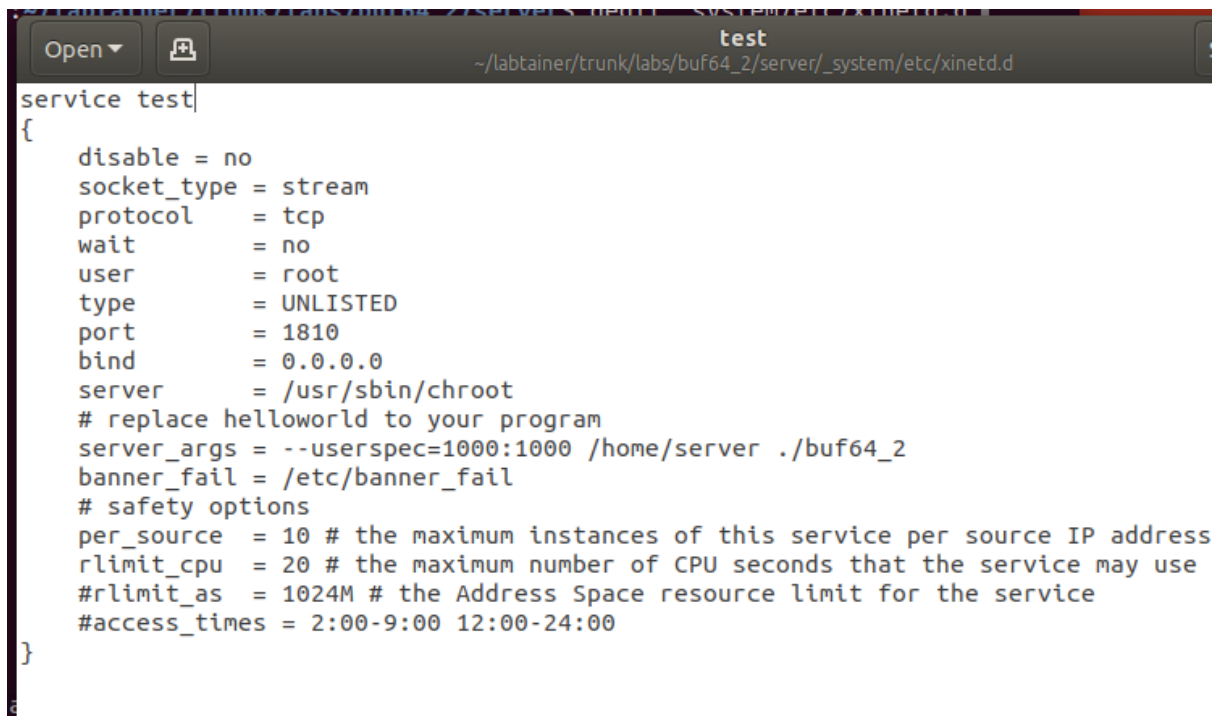
Hình 6. Tạo container ghidra

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



Hình 7. Tạo mạng mới cho lab

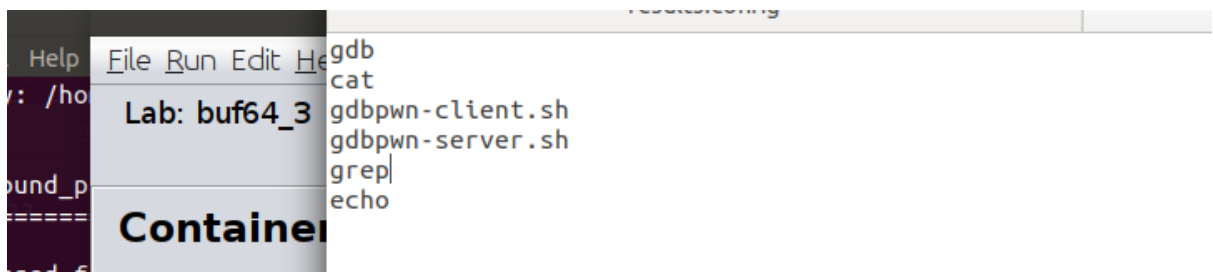
- Cấu hình mạng cho Attacker và Server:
 - Attacker:
 - IP: 192.168.1.3/24
 - GW: 192.168.1.1
 - Server:
 - IP 192.168.1.2/24
 - GW: 192.168.1.1
 - Service port: 1810
- Cài đặt thêm công cụ pwntools và extension pwndbg cho container attacker
- Cài đặt service binary cho container server



```
service test
{
    disable = no
    socket_type = stream
    protocol = tcp
    wait = no
    user = root
    type = UNLISTED
    port = 1810
    bind = 0.0.0.0
    server = /usr/sbin/chroot
    # replace helloworld to your program
    server_args = --userspec=1000:1000 /home/server ./buf64_2
    banner_fail = /etc/banner_fail
    # safety options
    per_source = 10 # the maximum instances of this service per source IP address
    rlimit_cpu = 20 # the maximum number of CPU seconds that the service may use
    #rlimit_as = 1024M # the Address Space resource limit for the service
    #access_times = 2:00-9:00 12:00-24:00
}
```

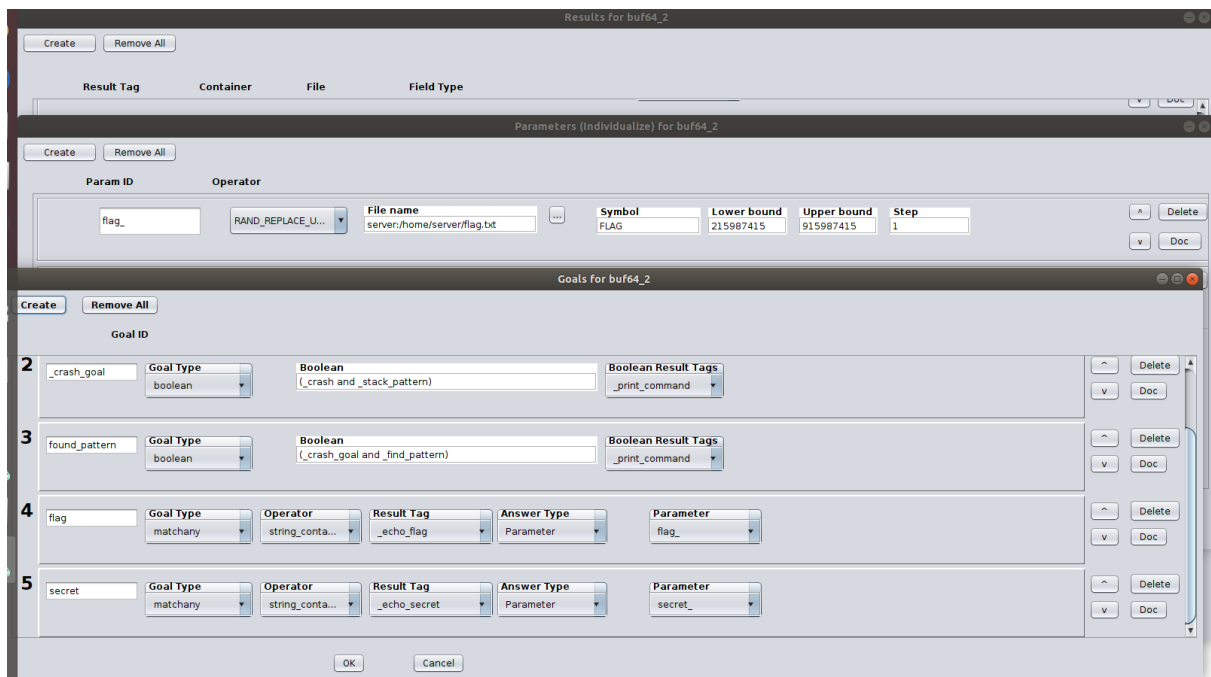
Hình 8. Cài đặt service binary tại port 1810

- 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 `~/lbtainer/trunk/labs/<tên-lab>/<tên-container>/_bin/treataslocal`



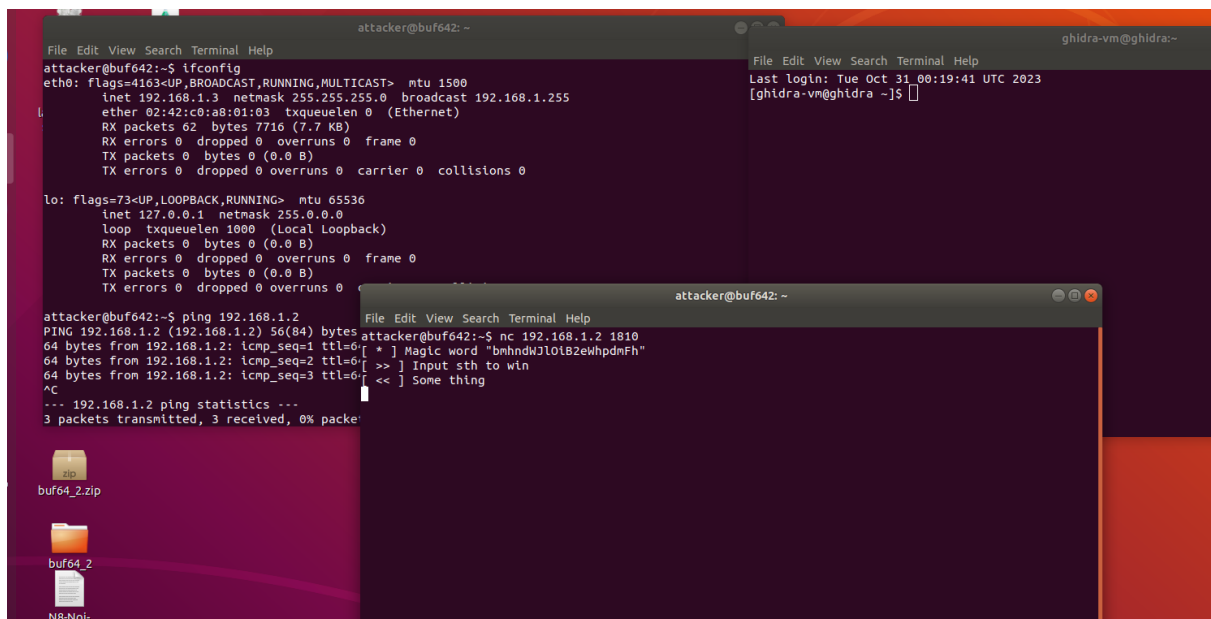
Hình 9. Chỉnh sửa file treataslocal trên máy attacker

- Cài đặt Results ở phần Automated Assesment:



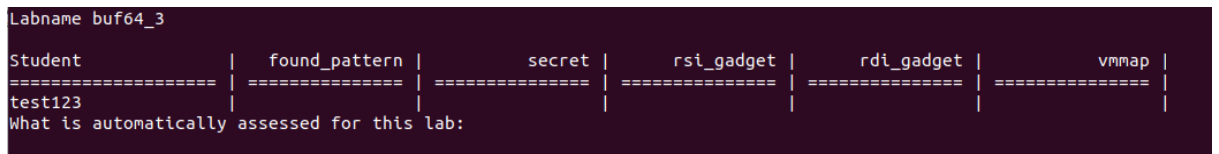
Hình 10. Cài đặt chấm điểm tự động

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



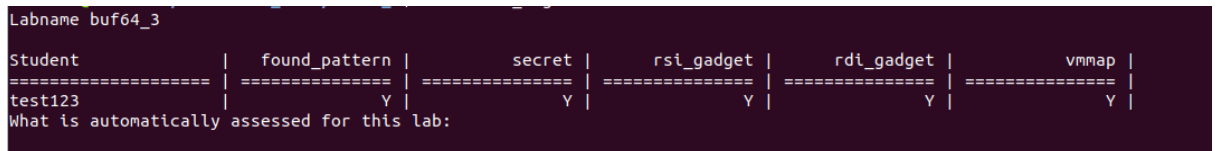
Hình 11. Build lab và chạy bài lab

IV. Thử nghiệm lab và kết quả



Labname	buf64_3				
Student	found_pattern	secret	rsi_gadget	rdi_gadget	vnmmap
=====	=====	=====	=====	=====	=====
test123					
What is automatically assessed for this lab:					

Hình 12. Run lab và thực hiện check work trước khi thực hiện bài lab



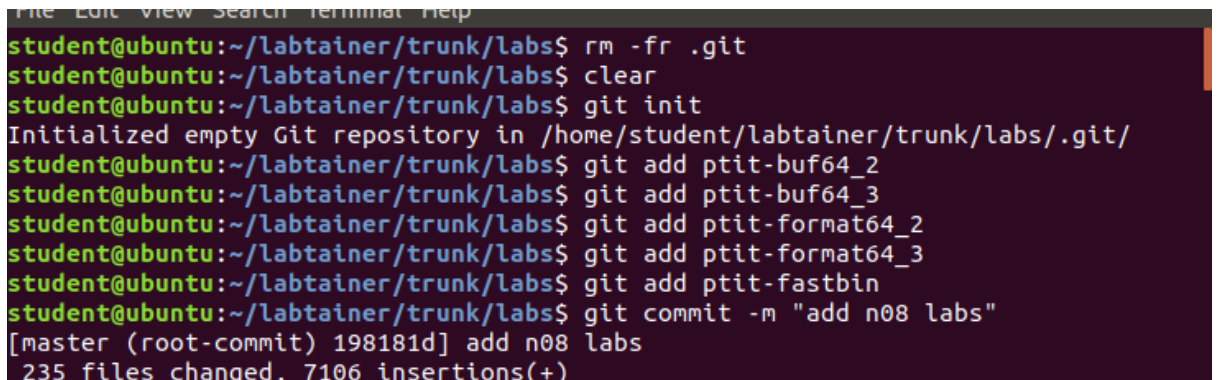
Labname	buf64_3				
Student	found_pattern	secret	rsi_gadget	rdi_gadget	vnmmap
=====	=====	=====	=====	=====	=====
test123	Y	Y	Y	Y	Y
What is automatically assessed for this lab:					

Hình 13. 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 bài-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

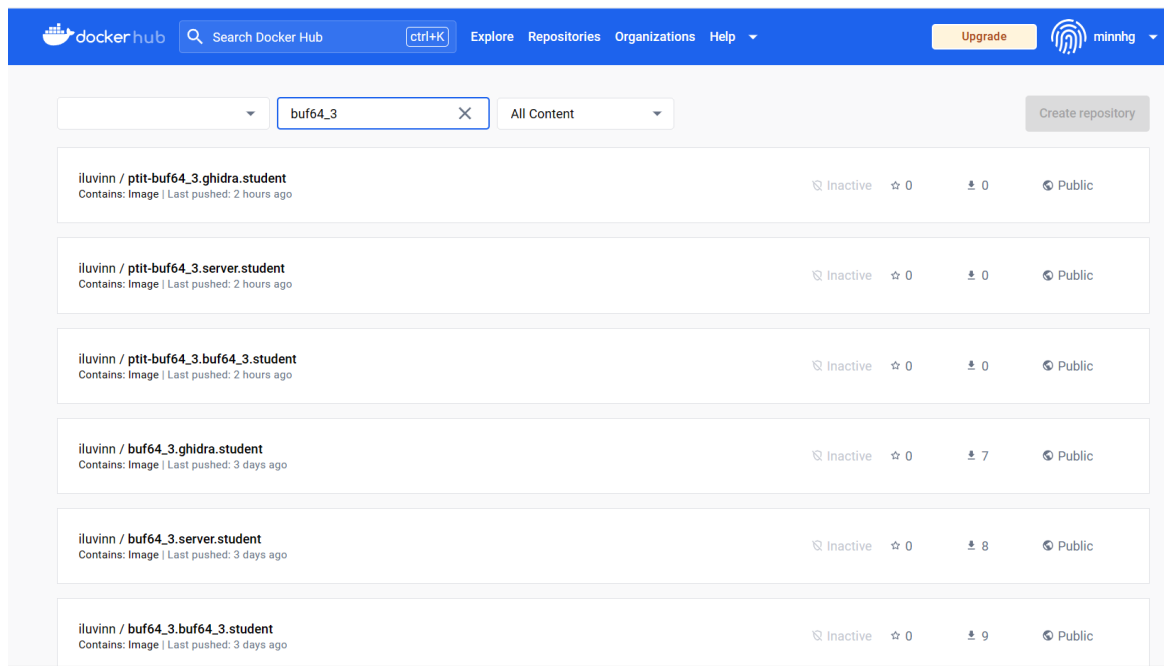
"Adding an IModule"



```
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 14. 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 15. Các images của vùng chứa được đẩy lên DockerHub

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

```
student@ubuntu: ~/labtainer/trunk/labs$ cd ../distrib/
student@ubuntu: ~/labtainer/trunk/distrib$ create-imodules.sh
lab is ptit-buf64_2
Do docs
lab is ptit-buf64_3
Do docs
lab is ptit-fastbin
Do docs
lab is ptit-format64_2
Do docs
lab is ptit-format64_3
Do docs
*****
** Post /home/student/labtainer/trunk/imodule.tar to your web server **
*****
student@ubuntu: ~/labtainer/trunk/distrib$
```

Hình 16. 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](#)