

Chương 1: Tổng quan về môi trường thực hành Labtainers	4
1.1 Ưu nhược điểm của Labtainers	4
1.2 Kiến trúc hệ thống	6
1.2.1 Quy trình thực hành của sinh viên	6
1.2.2 Cá nhân hóa bài thực hành	7
1.2.3 Đánh giá kết quả tự động	8
1.2.4 Mạng lưới của vùng chứa	9
1.3 Lý thuyết Docker và đóng gói máy ảo	10
1.3.1 Lý thuyết Docker	10
1.3.2 Đóng gói máy ảo	12
1.4 Môi trường thực thi Labtainers (Dockerfile và tệp cấu hình của Containers)	14
1.4.1 Dockerfile.....	14
1.4.2 start.config - Cấu hình các vùng chứa.....	16
1.5 Chi tiết một số mô đun	17
1.5.1 Giao diện labedit	17
1.5.2 Cá nhân hóa.....	17
1.5.3 Đánh giá tự động.....	19
Chương 2. Xây dựng và quản lý bài thực hành.....	24
2.1 Cách xây dựng một bài thực hành mới	25
2.1.1 Tạo bài thực hành.....	25
2.1.2 Cấu hình cho đánh giá tự động	27
2.1.3 Cá nhân hóa cho bài thực hành	29
2.2 Quản lý bài thực hành.....	30
2.3 Xây dựng mô đun hỗ trợ đánh giá kết quả: .json sang .csv.....	32
Chương 3. Thiết kế một số bài thực hành an toàn thông tin	34
3.1 Danh sách các bài thực hành	34
3.2 Phân tích và thiết kế các bài thực hành	35
3.2.1 Honeypot.....	35
3.2.2 Tcpdump	37
3.2.3 Arpspoof & scapy để thực tấn công người ở giữa (man-in-the-middle)	39

3.2.4 Truy tìm mật mã bằng khai thác lỗ hổng rlogin và mật mã caeser.....	42
3.3 Chi tiết xây dựng bài thực hành - honeypot	45
3.4 Cách thức sử dụng bài thực hành	52
KẾT LUẬN	Error! Bookmark not defined.
TÀI LIỆU THAM KHẢO	53

DANH MỤC HÌNH ẢNH

Hình 1.1: Sơ đồ mạng của bài thực hành routing-basics	10
Hình 1.2: Quy trình thực hiện bài thực hành	13
Hình 1.3: Sinh viên sử dụng lại bài thực hành của bạn khác và thay đổi tên.	14
Hình 1.4: Vòng đời của Docker	16
Hình 1.5: Docker images - thông tin của các ảnh.....	17
Hình 1.6: Labtainers thông qua phần mềm ảo hóa VMware, VirtualBox,	19
Hình 1.7: Labtainers với người dùng hệ thống Linux	20
Hình 1.8: Giao diện labedit.....	23
Hình 1.9: File đóng gói kết quả của sinh viên được chuyển vào thư mục như tên bài thực hành.	28
Hình 1.10: gradelab <tên bài thực hành>	29
Hình 1.11: Sử dụng cờ -w với lệnh gradelab	29
Hình 1.12: Kết quả được hiển thị trên trình duyệt localhost:8008	30
Hình 2.1: Giao diện labedit.....	31
Hình 2.2: Đặt tên cho bài thực hành mới	32
Hình 2.3: Docker image cơ sở base	32
Hình 2.4: File read_first.txt	32
Hình 2.5: kết quả lưu trữ trong thư mục student.zip/.local/result	33
Hình 2.6: Results trên giao diện labedit	33
Hình 2.7: Giao diện result của gitlab	34
Hình 2.8: Giao diện goals của gitlab	35
Hình 2.9: Giao diện cá nhân hóa bài thực hành gitlab trên labedit	36
Hình 2.10: Các image vùng chứa (container) được tải lên DockerHub.	37
Hình 2.11: Kết quả chuyển từ .json sang .csv	39
Hình 3.1: Sơ đồ thiết kế bài thực hành honeypot	42
Hình 3.2: Sơ đồ mạng bài thực hành honeypot	42
Hình 3.3: Sơ đồ thiết kế bài thực hành tcpdump	45
Hình 3.4: Sơ đồ thiết kế bài thực hành arpspoof & scapy	47
Hình 3.5: Sơ đồ thiết kế bài thực hành truy tìm mật mã caesar	50

Hình 3.6: Thêm gói thư viện ở Dockerfile của vùng chứa (container) server	52
Hình 3.7: Tập treataslocal của server	53
Hình 3.8: Checkwork trong labedit (GUI)	53
Hình 3.9: Edit / Config (registry)	54
Hình 3.10: điền tên DockerHub vào ô Registry	54
Hình 3.11: git init	54
Hình 3.12: cleanlab4svn.py	54
Hình 3.13: Cấu hình git email và tên	54
Hình 3.14: ./publish.py -d -l honeypot	55
Hình 3.15: các image đã được đẩy lên DockerHub	55
Hình 3.16: Nhập tài khoản, mật khẩu DockerHub	55
Hình 3.17: Sơ đồ mạng	56
Hình 3.18: Bài thực hành demo được lấy images từ DockerHub	58

DANH MỤC BẢNG

Bảng 3.1: Danh sách bài thực hành	40
---	----

Chương 1: Tổng quan về môi trường thực hành Labtainers

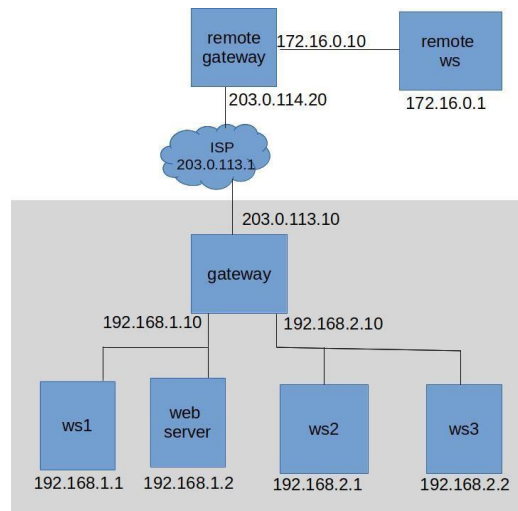
Labtainers cung cấp môi trường mà sinh viên thực hiện các bài thực hành thực hành khoa học máy tính dựa trên Linux với trọng tâm về an toàn thông tin, an ninh mạng. Đồng thời cho phép giảng viên muốn tạo hoặc điều chỉnh bài thực hành an toàn thông tin trong Labtainers. Labtainer framework được thiết kế để sử dụng cho các bài tập thí nghiệm an toàn thông tin nhằm mục tiêu vào môi trường Linux và nó được xây dựng xung quanh các vùng chứa (container) Linux Docker tiêu chuẩn (standard container).

Các bài thực hành Labtainer không chỉ thực hiện thông qua các dòng lệnh Linux mà các ứng dụng giao diện dựa trên GUI, chẳng hạn như trình duyệt hay Wireshark cũng được hỗ trợ.

1.1 Ưu nhược điểm của Labtainers

Việc triển khai các bài thực hành thực hành an toàn thông tin dựa trên framework này mang lại một số lợi ích chính:

1. Môi trường Labtainers được kiểm soát và nhất quán trên tất cả các máy tính của sinh viên. Cho phép giảng viên kiểm soát các gói phần thư viện (packages), phiên bản của các thư viện. Các bài thực hành khác nhau có thể cấu hình khác nhau hay thậm chí trong một bài thực hành có nhiều máy tính khác nhau với cấu hình mỗi máy là khác nhau.
2. Việc chấm điểm, đánh giá bài làm của sinh viên có thể được chấm tự động thông qua các tệp cấu hình trước để xác định các kết quả mà giảng viên mong muốn sinh viên thực hiện. Do đó, giảng viên sẽ không phải tự mình kiểm tra chi tiết từng sinh viên.
3. Cá nhân hóa - các bài thực hành sẽ được lưu và gán với từng bạn sinh viên. Vì vậy, sinh viên khó có thể sao chép kết quả của sinh viên khác.



Hình 1.1: Sơ đồ mạng của bài thực hành routing-basics

Labtainers cung cấp môi trường thực hành nhất quán mà không yêu cầu một máy ảo (VMs) riêng lẻ cho mỗi bài thực hành. Lợi ích này dễ dàng nhận thấy thông qua Hình 1.1 bài thực hành bao gồm nhiều máy tính kết nối mạng. Khi sử dụng vùng chứa (container) sẽ cần ít tài nguyên hơn so với máy ảo VMs. Một máy tính cá nhân của sinh viên sẽ gặp khó khăn trong việc chạy hai hoặc nhiều máy ảo nhưng có thể dễ dàng chạy đồng thời nhiều containers [1].

Giảng viên về ngành an toàn thông tin nói riêng và ngành công nghệ thông tin nói chung cần môi trường hay hệ thống để triển khai bài thực hành giúp sinh viên được trải nghiệm thực tế và nâng cao kỹ năng. Có rất nhiều các nền tảng thực hành an toàn thông tin như: DeterLab [2] cung cấp 24 bài tập chạy trên nền tảng thử nghiệm DETERLab, sinh viên, giảng viên truy cập và làm bài thực hành qua web. Tương tự thế, RAVE [3] (Môi trường ảo truy cập từ xa - Remote Access Virtual Environment) cung cấp nhiều trung tâm cơ sở hạ tầng và các bài tập thí nghiệm có thể chia sẻ. Tuy nhiên, sự khó khăn đối với giảng viên hay người hướng dẫn là để xây dựng, thực hiện và duy trì các bài thực hành với mức độ phức tạp. Không những thế, các nền tảng trên yêu cầu sinh viên phải được kết nối với nền tảng cơ sở hạ tầng lưu trữ các máy ảo. Trái lại, đối với Labtainers, việc sử dụng vùng chứa, xây dựng hệ thống xung quanh các vùng chứa (container) Linux Docker giúp việc triển khai các phòng thí nghiệm an ninh mạng mang lại sự đơn giản, nhất quán, cá nhân hóa bài thực hành cho mỗi sinh viên và hỗ trợ đánh giá kết quả của sinh viên.

Bên cạnh những ưu điểm thì Labtainers cũng có những hạn chế. Labtainer framework giới hạn bài thực hành trong môi trường hệ điều hành Linux. Tuy nhiên, người giảng viên thiết kế bài thực hành cũng có thể thiết kế máy ảo riêng biệt ví dụ: hệ thống windows và máy ảo đó có thể được nối mạng với máy ảo Linux lưu trữ vùng chứa (container) Docker.

Hiện tại, các thao tác, kết quả trên hệ thống Windows đó chưa thể được đánh giá hay cá nhân hóa tự động.

1.2 Kiến trúc hệ thống

Kiến trúc Labtainers [4], [5] được phát triển và triển khai dựa trên Linux, sử dụng vùng chứa (container) Docker (Docker containers) để tạo môi trường nhất quán, đồng nhất trên tất cả các máy tính của từng sinh viên dù phiên bản và cấu hình Linux là khác nhau. Việc sử dụng vùng chứa (container) Docker giúp giảng viên hay người xây dựng bài thực hành cài đặt, thiết kế môi trường, tài nguyên (chương trình, thư viện sẽ được cài đặt, phiên bản các gói thư viện, cấu hình mạng, ...) thích hợp cho mỗi bài thực hành. Các cấu hình có thể khác nhau giữa các bài thực hành và chúng có thể khác nhau giữa nhiều vùng chứa (container) được triển khai trong một bài thực hành. Do đó, sinh viên có thể dùng bất kỳ phiên bản Linux nào có hỗ trợ Docker mà không phải yêu cầu một phiên bản hay cấu hình cụ thể. (Lý thuyết Docker và Đóng gói máy ảo sẽ được thể hiện chi tiết ở phần [Lý thuyết Docker và đóng gói máy ảo](#)).

Kiến trúc Labtainer phục vụ cho 3 kiểu người dùng. Đầu tiên là người thiết kế, tạo ra những bài thực hành. Người này sẽ xác định, xây dựng cấu hình, kết quả, cấu trúc của bài thực hành. Thứ hai là người hướng dẫn. Họ sẽ giao bài thực hành cho sinh viên và thu thập kết quả của sinh viên. Người hướng dẫn cũng có thể làm việc với người thiết kế để tạo ra những bài thực hành mới. Cuối cùng là sinh viên, người làm bài thực hành.

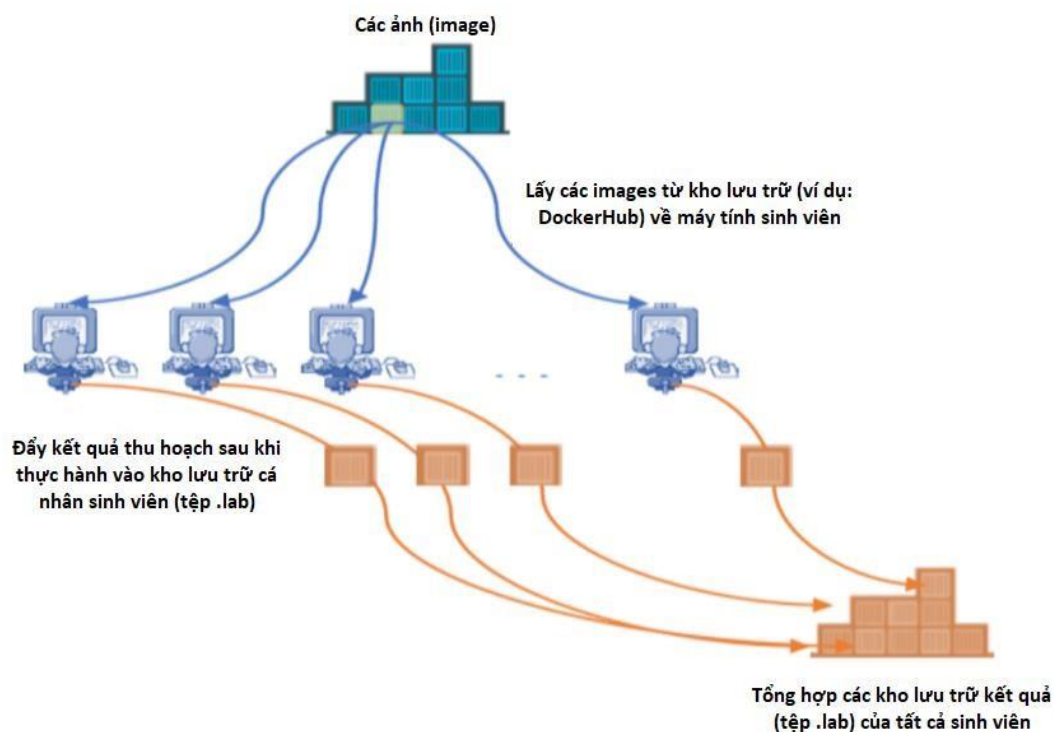
1.2.1 Quy trình thực hành của sinh viên

Sinh viên bắt đầu bài thực hành Labtainer từ bất kỳ hệ thống Linux nào (ví dụ: máy ảo VMware trên máy tính xách tay). Lệnh bắt đầu bài thực hành bằng tên của bài thực hành và hệ thống Labtainers sẽ lấy tất cả vùng chứa (container) Docker cần thiết và cấu hình của chúng từ Docker Hub để sinh viên sử dụng. Sau khi cấu hình hoàn tất, sinh viên được cung cấp một bộ thiết bị đầu cuối ảo (virtual terminals) cung cấp quyền vào môi trường bài thực hành. Các thiết bị đầu cuối này thường là bash shell mà qua đó sinh viên tương tác với các vùng chứa, nó thường xuất hiện với sinh viên dưới dạng các máy Linux riêng lẻ được kết nối với một hoặc nhiều mạng (ví dụ: máy khách, bộ định tuyến, máy chủ). Khi lệnh kết thúc bài được thực thi, hệ thống sẽ thu thập tập hợp các thao tác từ hoạt động của sinh viên sau đó đặt chúng vào tệp zip và sinh viên sẽ chuyển tiếp tệp zip (.lab) đó cho người hướng dẫn. Sau khi tập hợp các tệp zip từ tất cả các học viên vào một thư mục, người hướng dẫn bắt đầu một vùng chứa (container) dành cho người hướng dẫn đặc biệt được tạo cho bài thực hành. Vùng chứa (container) này tự động đánh giá các bài của sinh viên và cung cấp cho giảng viên kết quả của từng sinh viên. Giảng viên cũng được cung cấp bản sao thư mục chính của mỗi sinh viên và cho phép kiểm tra và xem kết quả của sinh viên. Quy trình bài thực hành của sinh viên được thể hiện trong Hình 1.2.

Cách cài đặt Labtainers tham khảo tại [6].

Trong Labtainer, quy trình của sinh viên được miêu tả ngắn gọn dưới đây:

- 1) Sinh viên lấy vùng chứa (container) Linux từ Docker Hub (bằng lệnh pull).
- 2) Sinh viên thực hiện bài thực hành của mình.
- 3) Kết thúc bài thực hành, kết quả thao tác được lưu dưới dạng tệp zip và gửi tệp zip cho giảng viên.
- 4) Giảng viên khởi chạy vùng chứa (container) dành cho người hướng dẫn để chấm điểm và đánh giá.



Hình 1.2: Quy trình thực hiện bài thực hành

1.2.2 Cá nhân hóa bài thực hành

Cá nhân hóa bài thực hành nhằm mục đích ngăn cản sinh viên chia sẻ hoặc sao chép cách giải, giải pháp của bài thực hành. Các bài thực hành được cá nhân hóa thông qua sự thay thế tương trưng của các giá trị trong mã nguồn hoặc tệp dữ liệu của bài thực hành. Khi thiết kế bài thực hành, người tạo sẽ xác định tệp, ký hiệu, kiểu thay thế cho quá trình cá nhân hóa. Các giá trị ngẫu nhiên được tạo bằng cách sử dụng trình tạo số ngẫu nhiên được tạo bằng một chuỗi cụ thể cho từng sinh viên và bài thực hành. Thực tế, thường một chuỗi

được xác định trước cho bài thực hành của sinh viên là địa chỉ email của sinh viên được đăng ký khi bắt đầu bài thực hành.

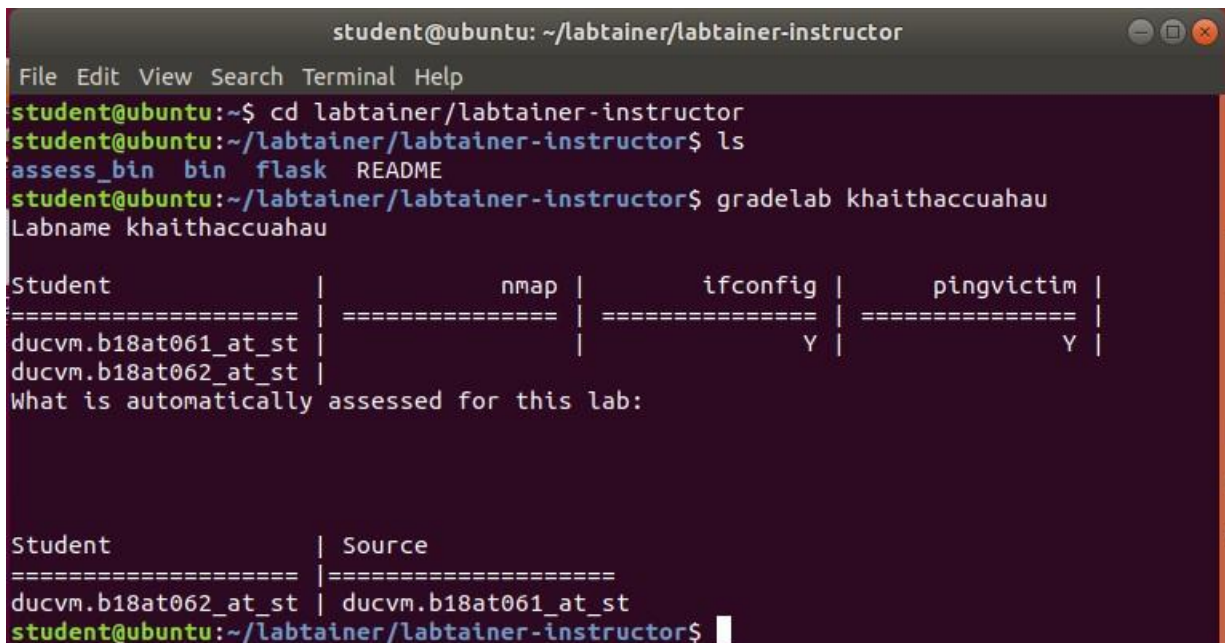
Trong bài thực hành telnet, sinh viên được hướng dẫn kết nối tới một máy chủ và mở, hiển thị nội dung của tệp cụ thể. Việc cá nhân hóa sẽ làm cho nội dung của tệp này là duy nhất đối với sinh viên đó, ví dụ: kết quả bởi hàm băm của địa chỉ email của sinh viên đăng ký trước đó.

Ví dụ:

```
FSTRING : HASH_REPLACE : \
telnetlab.server.student=filetoview.txt : \
TELNET_STRING : mytelnetfilestring
```

Cú pháp trên làm xâu TELNET_STRING trong tệp filetoview.txt được thay thế hàm băm MD5 của chuỗi “mytelnetfilestring” bằng email của sinh viên.

Cách cá nhân hóa đơn giản nhất, để đảm bảo sinh viên nộp bài của chính mình, là cá nhân hóa một số tệp trên một trong các vùng chứa (container) và sau đó kiểm tra tệp đó và tên tệp lưu trữ khi chấm điểm tự động – **gradelab**.



```
student@ubuntu: ~/labtainer/labtainer-instructor
File Edit View Search Terminal Help
student@ubuntu:~$ cd labtainer/labtainer-instructor
student@ubuntu:~/labtainer/labtainer-instructor$ ls
assess_bin bin flask README
student@ubuntu:~/labtainer/labtainer-instructor$ gradelab khaiithaccuahau
Labname khaiithaccuahau

Student | nmap | ifconfig | pingvictim |
=====|=====|=====|=====|
ducv.m.b18at061_at_st | | Y | Y |
ducv.m.b18at062_at_st | | | |
What is automatically assessed for this lab:

Student | Source
=====|=====
ducv.m.b18at062_at_st | ducvm.b18at061_at_st
student@ubuntu:~/labtainer/labtainer-instructor$
```

Hình 1.3: Sinh viên sử dụng lại bài thực hành của bạn khác và thay đổi tên.

1.2.3 Đánh giá kết quả tự động

Các mục tiêu được theo dõi bởi việc chấm điểm tự động sẽ phản ánh liệu các thao tác đầu vào và kết quả đầu ra đã được thực hiện, tạo ra, hay hiển thị hay chưa. Labtainers có thể tạo các bài thực hành trong đó sinh viên phải thực hiện một chuỗi các thao tác cụ thể và người xây dựng bài thực hành có thể dễ dàng tạo các đánh giá cho các bài thực hành đó.

Mỗi khi sinh viên gọi một chương trình hoặc tiện ích, hệ thống sẽ ghi lại các bản sao của tiêu chuẩn đầu vào và tiêu chuẩn đầu ra (stdin và stdout) vào các tệp được đánh dấu thời gian. Các tệp và mọi thứ liên quan tới bài tập sẽ được tự động đóng gói khi sinh viên kết thúc bài thực hành. Các tệp stdin và stdout được đánh dấu thời gian và được ghi lại trong \$HOME/.local/result.

1.2.4 Mạng lưới của vùng chứa

Trong phần này sẽ mô tả hệ thống Labtainer tạo một mạng đơn giản của vùng chứa (container) bao gồm máy khách và máy chủ.

Mỗi vùng chứa (container) trong bài thực hành được xác định bởi Dockerfile, Dockerfile sẽ chỉ định các gói thư viện và tệp có trong hệ thống tệp của hình ảnh vùng chứa (container image). Hình ảnh cơ sở Labtainer bao gồm một tập hợp các gói thư viện hữu ích cho nhiều bài thực hành, ví dụ: gcc, vim, python, ... Mọi Dockerfile dành riêng cho bài thực hành sẽ tham chiếu tới hình ảnh cơ sở đó hoặc hình ảnh bắt nguồn từ cơ sở đó.

Dockerfile sau đó xác định các gói thư viện và tệp bổ sung cho vùng chứa. Dockerfile dành riêng cho bài thực hành của máy chủ được xây dựng dựa trên hình ảnh này bao gồm cả dịch vụ telnet. Ngoài ra, một Dockerfile thường bắt đầu một dịch vụ bằng các sử dụng lệnh ENTRY để các mục nhật ký hệ thống sẽ xuất hiện ở vị trí quen thuộc /var/log. ENTRY cho vùng chứa (container) máy chủ trong ví dụ bắt đầu một tập lệnh đơn giản để khởi chạy rsyslog và xinetd việc này sẽ khởi chạy các dịch vụ telnet và sshd để đáp ứng với các kết nối mạng đến.

Hình ảnh Docker được tạo từ Dockerfiles cho từng vùng chứa (container) của bài thực hành được tham chiếu trong tệp start.config do người tạo bài thực hành tạo cho mỗi bài thực hành. Tệp này xác định các vùng chứa và xác định mạng trong bài thực hành. Ví dụ cho tệp cấu hình mạng:

```
NETWORK SOME_NETWORK
MASK 172.20.0.0/24
GATEWAY 172.20.0.100
```

Một vùng chứa (container) kết nối với các mạng bằng cách đặt tên cho các mạng trong tệp cấu hình cho vùng chứa (container) đó.

```
CONTAINER client
USER ubuntu
TERMINALS 2
SOME_NETWORK 172.20.0.2
CONTAINER server
USER ubuntu
TERMINALS 1
```

SOME_NETWORK 172.20.0.3

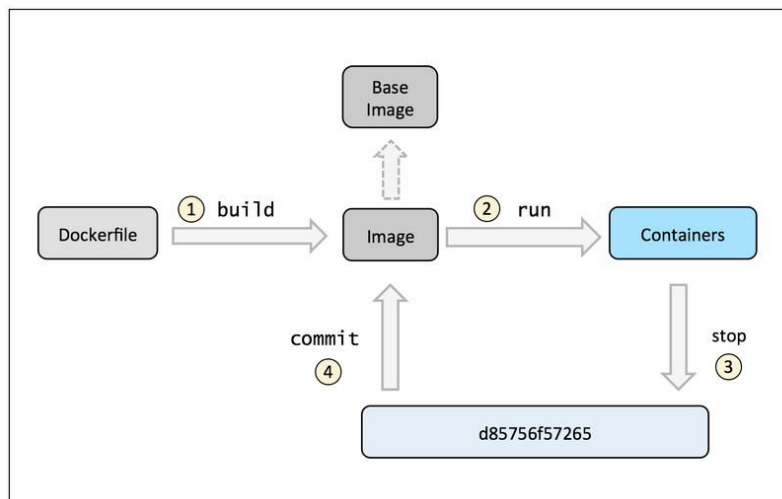
Tên vùng chứa (container) của máy khách và máy chủ phân giải thành Dockerfiles tương ứng theo quy ước đặt tên của Labtainer. Các mục đầu vào này chỉ định địa chỉ mạng cho các vùng chứa (SOME_NETWORK 172.20.0.2) và xác định số lượng thiết bị đầu cuối ảo sẽ được tạo và gắn (TERMINALS 2: sẽ có 2 terminal được khởi tạo) vào mỗi vùng chứa (container) khi bài thực hành chạy.

Ba đầu vào của tệp cấu hình này đủ để xác định mạng đơn giản mà sinh viên nhìn thấy khi thực hiện thí nghiệm ví dụ. Khi bài thực hành bắt đầu, các thiết bị đầu cuối ảo được tạo và được hiện thị dạng bash shell (terminal), cho phép sinh viên tương tác với các vùng chứa (container) như là các hệ thống Linux độc lập được kết nối mạng. Vùng chứa (container) máy chủ (server) cung cấp dịch vụ telnet mà sinh viên có thể tiếp cận bằng cách đưa ra lệnh telnet từ terminal của máy khách (client). Tất cả sinh viên sẽ thấy cùng một máy chủ và máy khác telnet, bất kể họ đang chạy bản Linux nào hay bất kể gói phần mềm (packages) nào được cài đặt trên máy chủ Linux của họ.

1.3 Lý thuyết Docker và đóng gói máy ảo

1.3.1 Lý thuyết Docker

1.3.1.1 Vòng đời của Docker



Hình 1.4: Vòng đời của Docker

Docker được bắt đầu từ tạo chạy dockerfile, một vùng chứa (container) bắt nguồn từ 1 hình ảnh (image), mỗi lần chạy một image sẽ nhận được 1 container khác nhau. Image sẽ ghi lại chính xác và đồng thời trạng thái của cơ sở dữ liệu tại một thời điểm, khi dữ liệu được cập nhật thì image cũng được cập nhật theo. Image có những tệp hệ thống, thư viện cơ bản để sẵn sàng sử dụng. Tiếp theo trong vòng đời của Docker là vùng chứa (container). Khi vùng chứa kết thúc, mọi tiến trình đang sử dụng vùng chứa đều được giữ lại, tức là sau

khi container đã dừng dữ liệu không bị mất mà được lưu trữ lại. Tiếp theo, chúng ta có thể tạo một image mới từ container bằng việc chụp lại một hình ảnh (image) từ thay đổi container trong quá trình sử dụng. Tuy nhiên, trong khuôn khổ với những bài thực hành của Labtainer thì sinh viên chỉ khởi chạy bài thực hành (khởi chạy Dockerfile, Image), sử dụng, kết quả và các sửa đổi được lưu trữ ra tệp zip, sau đó gửi tệp zip cho giảng viên mà không tạo ra image mới từ container (các vùng chứa trong bài thực hành).

1.3.1.2 Image - Ảnh

Docker image là tệp chứa đủ những phần tử cần thiết để tạo nên một hệ điều hành với các thư viện, phần mềm phục vụ cho bài thực hành.

REPOSITORY	TAG	IMAGE ID	CREATED	SIZE
ubuntu	latest	ea4c82dcd15a	3 weeks ago	85.8MB
hello-world	latest	4ab4c602aa5e	2 months ago	1.84kB

Hình 1.5: Docker images - thông tin của các ảnh

Trên đây thông tin của 2 hình ảnh là ubuntu và hello-world.

Trong đó:

- **Repository:** là nơi tạo ra hình ảnh đó.
- **Tag:** là phiên bản của image, mặc định khi khởi tạo: “latest” sẽ đi kèm với tên của ảnh.
- **IMAGE ID:** là mã của ảnh.

1.3.1.3 Container

Docker Container được tạo ra từ Docker image. Container (vùng chứa) là một môi trường thực hiện cung cấp cho một ứng dụng hoặc phần mềm của hệ điều hành, người dùng có thể chạy một ứng dụng độc lập. Những container này rất gọn nhẹ và cho phép chạy ứng dụng rất nhanh chóng và dễ dàng. Với một máy chủ vật lý, thay vì chỉ chạy được số lượng ít các máy ảo VMware hay VirtualBox thì có thể chạy vài chục Docker Container.

1.3.1.4 Dockerfile

Dockerfile là tệp cấu hình (config) cho Docker để tạo ra image. Nó sẽ dùng một image cơ sở để xây dựng image mới. Một số image cơ sở trong Labtainers là: base, network2, ... Sau đó nếu có các lớp, gói thư viện bổ sung thì sẽ được xếp chồng lên lớp cơ sở.

Các lệnh cấu hình của Dockerfile:

FROM — Chỉ định image gốc: base, network2, ...

LABEL — Cung cấp metadata cho image. Có thể sử dụng để thêm thông tin bảo trì.
Để xem các label của images, dùng lệnh docker inspect.

ENV — Thiết lập một biến môi trường.

RUN — Có thể tạo một lệnh khi tạo image. Được sử dụng để cài đặt các package – gói thư viện vào container.

COPY — Sao chép các file và thư mục vào container.

ADD — Sao chép các file và thư mục vào container.

CMD — Cung cấp một lệnh và đối số cho container thực thi. Các tham số có thể được ghi đè và chỉ có một CMD.

WORKDIR — Thiết lập thư mục đang làm việc cho các chỉ thị khác như: RUN, CMD, ENTRYPOINT, COPY, ADD,...

ARG — Định nghĩa giá trị biến được dùng trong khi xây dựng và tạo image.

ENTRYPOINT — Cung cấp lệnh và đối số cho một container thực thi.

EXPOSE — Khai báo port của image.

VOLUME — Tạo một điểm gắn thư mục để truy cập và lưu trữ data.

1.3.2 Đóng gói máy ảo

Đóng gói máy ảo (Containerization) là một khái niệm ảo hóa sử dụng hình ảnh (images) và vùng chứa (container) (containers) để chạy các ứng dụng phân tán. Đặc biệt, chúng ta có một hình ảnh để chạy bất kỳ ứng dụng nào và có nhiều phiên bản bằng cách sử dụng vùng chứa (container) [6]. Ví dụ: Docker là một phần mềm mã nguồn mở xử lý việc triển khai các ứng dụng ở cấp hệ điều hành bằng cách sử dụng các vùng chứa (container) như đã đề cập ở [4.1 Lý thuyết Docker](#). Hơn nữa, Docker sử dụng công nghệ vùng chứa (container) để chạy các ứng dụng mà không cần phải cấu hình cụ thể môi trường máy ảo cho từng ứng dụng. Điều này mang lại sự cải thiện lớn về hiệu suất do cần ít tài nguyên hơn để chạy vùng chứa (container) so với phương pháp sử dụng nhiều máy ảo. Các giải pháp đóng gói máy ảo cũng có thể mang lại nhiều lợi ích khi triển khai ứng dụng. Vì các vùng chứa (container) phụ thuộc vào một máy chủ duy nhất và sử dụng cùng một nhân, nên chúng ta có thể có một số vùng chứa (container) chạy trên hệ điều hành ở cấp độ máy chủ. Tất cả những gì người dùng cần là cài đặt công cụ docker vào máy chủ để chạy và quản lý các vùng chứa.

Ngoài ra, các vùng chứa (container) hiệu quả hơn và di động hơn các máy ảo, đặc biệt là khi triển khai cùng một ứng dụng cho các môi trường khác nhau. Điều này có thể dễ dàng được thực hiện bằng cách sử dụng các vùng chứa (container) mà không cần cài đặt cụ thể hệ điều hành hoặc cơ sở hạ tầng. Tương tự, nền tảng máy chủ Linux của sinh viên có thể là bất kỳ bản Linux nào hỗ trợ Docker và không cần phải là một phiên bản hoặc cấu hình

cụ thể. Các ưu điểm của môi trường Linux Docker và tác động của nó đối với hệ thống Labtainer được liệt kê dưới đây.

“Tập hệ thống độc lập (File system isolation): mỗi vùng chứa (container) chạy trong một hệ thống tệp gốc (root file system) hoàn toàn riêng biệt”. Có nghĩa là mỗi bài thực hành có thể được cung cấp và đóng gói với tất cả các tệp, bao gồm các tệp lệnh khởi động và cá nhân hóa cần thiết để sinh viên bắt đầu bài thực hành.

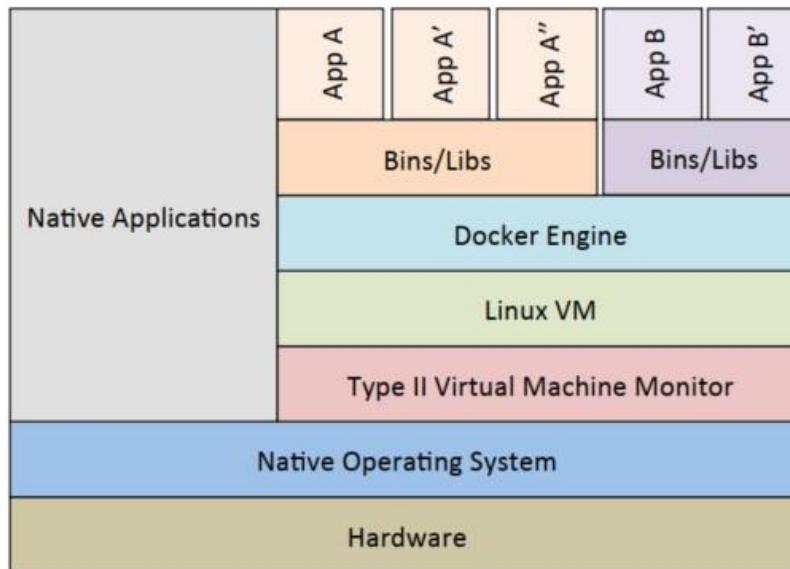
“Tài nguyên độc lập (Resource isolation): các tài nguyên hệ thống như CPU và bộ nhớ có thể được phân bổ khác nhau cho từng vùng chứa”. Các vấn đề liên quan đến bài thực hành sẽ không tràn ra khỏi vùng chứa. Khi làm bài thực hành sẽ chỉ làm thay đổi trong vùng chứa mà không ảnh hưởng tới các tiến trình khác ngoài vùng chứa.

“Mạng độc lập (Network isolation): mỗi vùng chứa (container) chạy trong không gian mạng riêng, với gian diện ảo và địa chỉ IP của riêng nó”. Đối với các bài tập thực hành, có thể tạo các vùng chứa (container) được nối mạng, do đó cho phép thực hiện bài tập về các chủ đề an ninh mạng.

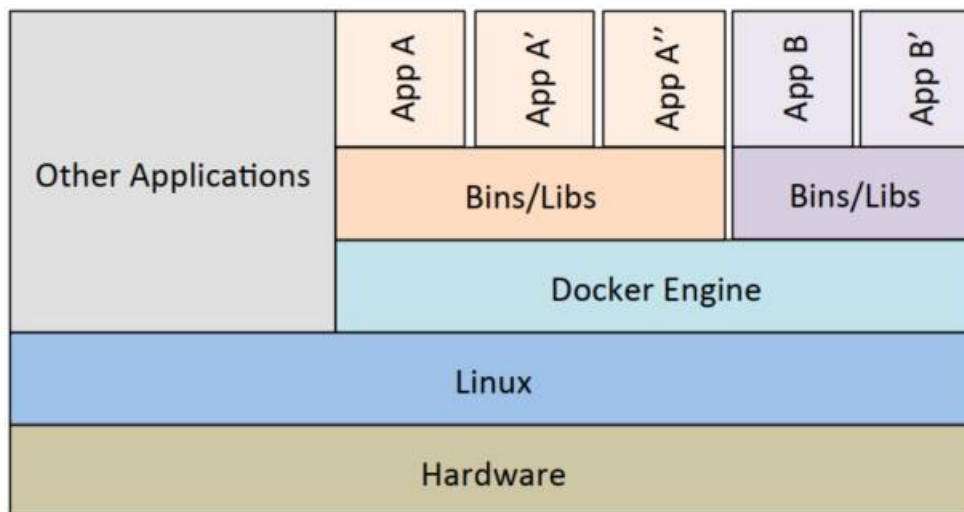
“Copy-on-write: hệ thống tập tin gốc được tạo bằng cách sử dụng copy-on-write, giúp triển khai cực kỳ nhanh, tiết kiệm bộ nhớ và ổ cứng”. Việc này cực kỳ hữu ích với nền tảng của sinh viên và giảng viên bị hạn chế về tài nguyên.

“Thay đổi sự quản lý (Change management): các thay đổi đối với hệ thống tệp của vùng chứa (container) có thể được tạo (commit) thành một hình ảnh mới (image) và được sử dụng lại để tạo nhiều vùng chứa (container) hơn” như đề cập trong [4.1 Lý thuyết Docker](#). Giảng viên có thể dễ dàng tạo ra các biến thể của các bài thực hành, do đó các bài tập có thể được mở rộng và phát triển.

Giảng viên, sinh viên sử dụng Labtainers có thể thông qua phần mềm ảo hóa ví dụ: VMware, VirtualBox, ... để cài máy ảo Linux (có hỗ trợ Docker). Các vùng chứa (container) chỉ bao gồm các phần tử cần thiết cho bài thực hành ví dụ như tệp thực thi, thư viện như trong Hình 1.6. Với người dùng sử dụng hệ thống Linux thì được mô tả như trong Hình 1.7.



Hình 1.6: Labtainers thông qua phần mềm ảo hóa VMware, VirtualBox, ...



Hình 1.7: Labtainers với người dùng hệ thống Linux

1.4 Môi trường thực thi Labtainers (Dockerfile và tệp cấu hình của Containers)

1.4.1 Dockerfile

Tệp Dockerfile của bài thực hành được đặt trong thư mục “Dockerfiles”. Và tệp Dockerfiles được thêm mỗi khi thêm máy tính mới thông qua GUI. Labtainers sử dụng cú pháp Docker tiêu chuẩn [7]. Dockerfile ban đầu để mặc định với một image Labtainer chứa những gói thư viện Linux cần thiết tối thiểu cho một bài thực hành. Tệp Dockerfile của từng container mở bằng cách click vào container và chọn tab Docker hoặc có thể mở trong thư mục: `$LABTAINER_DIR/labs/[tên bài lab]/dockerfiles` Ví dụ: `/home/student/labtainer/trunk/labs/arp-spoof/dockerfiles`

Trong đó:

\$LABTAINER_DIR là: /home/student/labtainer/trunk/

[tên bài lab] là: arp-spoof

Quy ước đặt tên cho Dockerfiles là: **Dockerfile.[tên bài lab].[tên_container].student**

Dòng đầu tiên của mỗi Dockerfile xác định image Labtainer cơ sở sẽ được lấy từ Docker Hub. Image cơ sở bao gồm:

- labtainer.base - Hệ thống Ubuntu tối thiểu.
- labtainer.network - Các gói mạng đã được cài đặt và xinetd đang chạy, nhưng các dịch vụ mạng chưa được kích hoạt.
- labtainer.network.ssh - Tương tự như labtainer.network, nhưng với ssh hoạt động trong cấu hình xinetd.
- labtainer.centos - Máy chủ CentOS với systemd.
- labtainer.lamp - Một máy chủ CentOS với Apache, Mysql và PHP, (ngăn xếp LAMP)
- labtainer.firefox - Một Ubuntu container với trình duyệt Firefox.
- labtainer.wireshark - labtainer.network được thêm wireshark.
- labtainer.java - Ubuntu container với trình duyệt Firefox và JDK mở.
- labtainer.kali - Hệ thống Kali Linux với Metasploit framework .
- labtainer.metasploitable - Máy chủ có lỗ hổng Metasploitable-2.
- labtainer.bird - Bộ định tuyến Bird (Xem bài thực hành bird - bird labs).
- labtainer.owasp - firefox với bộ công cụ OWASP zap.
- labtainer.juiceshop - Máy chủ web Juice Shop với lỗ hổng OWASP. Để xem những gói thư viện được thêm trong những image cơ sở này, đi tới thư mục:

\$LABTAINER_DIR/scripts/designer/base dockerfiles

Tập Dockerfile được để thêm gói thư viện cho container trong bài thực hành theo cú pháp:

RUN apt-get update && apt-get install -y some_package Ví

dụ:

```
RUN apt-get update && apt-get install -y --no-install-recommends \
openssl \  tcpdump \
```


1.4.2 start.config - Cấu hình các vùng chứa

Hầu hết các container của phòng thí nghiệm đều có thể sử dụng tệp start.config được tạo tự động mà không cần sửa đổi. Việc thêm mạng container hay xác định tên người dùng không phải người dùng "ubuntu" mặc định thì cần phải sửa đổi tệp start.config. Phần sau đây mô tả các phần chính của tệp cấu hình đó. Hầu hết các mục cấu hình có thể không phải sửa đổi cho hầu hết các bài thực hành.

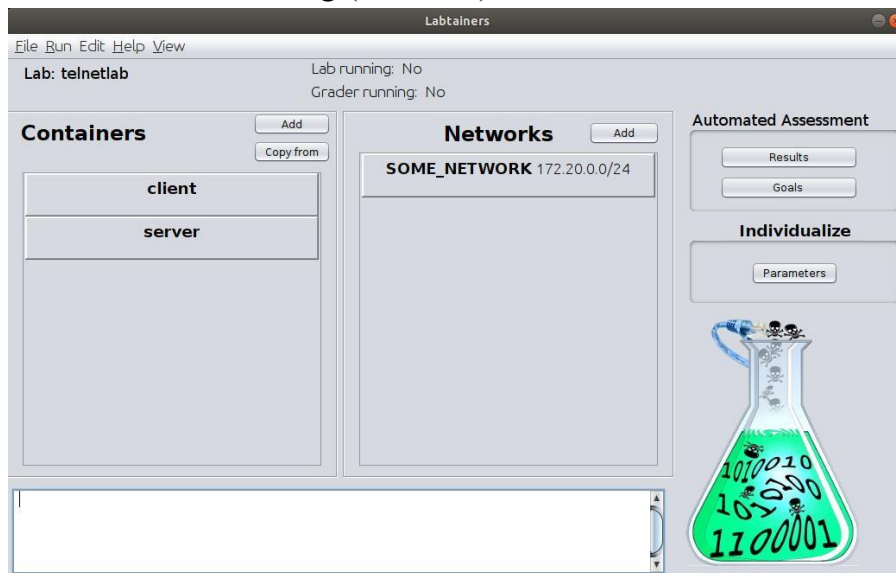
- GLOBAL SETTINGS - Biến toàn cục của bài thực hành bao gồm:
 - LAB MASTER SEED [seed] - chuỗi nguồn gốc cho bài thực hành.
 - REGISTRY [registry] - Mã hay tài khoản Docker Hub chứa các image của bài thực hành. Giá trị này mặc định này được xác định trong tệp labtainers.config tại thư mục \$LABTAINER_DIR/config.
 - BASE_REGISTRY [base_registry] - Mã hay tài khoản Docker Hub chứa các image cơ sở cho container của bài thực hành.
 - COLLECT DOCS [yes/no]
 - CHECKWORK [yes/no] - Tùy chọn để vô hiệu hóa (để “no”) khả năng sinh viên tự kiểm tra bài làm của mình từ thư mục labtainer-student.
- NETWORK [tên mạng] Phần này là bắt buộc đối với mỗi mạng trong bài thực hành. Nó được sử dụng trong file start.config khi cần truy xuất tới mạng, các giá trị sau đây được xác định cho NETWORK:
 - MASK [địa chỉ mask] - ví dụ: 172.25.0.0./24
 - GATEWAY [địa chỉ cổng] - Địa chỉ IP của cổng mạng được Docker sử dụng để giao tiếp với máy chủ. Trường GATEWAY không được đặt trùng tên IP với những thành phần khác.
- CONTAINER [container name - tên của container] - Tên của container là bắt buộc, và các giá trị mặc định cho container được tạo tự động bởi tập lệnh lab setup.py.
 - TERMINALS [số lượng] - Số lượng thiết bị đầu cuối ảo sẽ mở và gắn vào vùng chứa (container) này khi bài thực hành bắt đầu. Nếu thiếu, tương đương với giá trị mặc định là 1. Giá trị 0 ngăn chặn việc tạo thiết bị đầu cuối. Giá trị -1 thì sinh viên không thể gắn thiết bị đầu cuối vào vùng chứa.
 - TERMINAL_GROUP [tên] - Các thiết bị đầu cuối trong cùng một nhóm được sắp xếp dưới dạng các tab trong một thiết bị đầu cuối duy nhất. [name - tên] là chuỗi tùy ý.
 - USER [tên người dùng] - Tên người dùng, mặc định là “ubuntu”.
 - PASSWORD [mật khẩu] - Mật khẩu cho tài khoản tên người dùng được xác định ở USER.
 - [Tên mạng] [địa chỉ mạng] - Địa chỉ mạng được kết nối với vùng chứa (container) - container này. Một dòng riêng biệt được nhập cho mỗi mạng.

- X11 [YES/NO] - Mặc định là NO. Nếu là YES, sẽ cho phép vùng chứa (container) - container chạy được các ứng dụng có GUI, ví dụ: trình duyệt (firefox) hoặc Wireshark.

1.5 Chi tiết một số mô đun

1.5.1 Giao diện labedit

Labtainers hỗ trợ giao diện GUI để tương tác, giúp giáo viên tiện thao tác để thiết kế và duy trì bài thực hành tiện lợi dễ dàng (Hình 1.8).



Hình 1.8: Giao diện labedit

Khởi động trình điều khiển Labtainers Lab bằng cách mở terminal từ bất kỳ đường dẫn nào và khởi động bằng lệnh **labedit**.

1.5.2 Cá nhân hóa

Phần này mô tả cách cá nhân hóa bài thực hành cho từng sinh viên để tránh sao chép hoặc dùng lại. Bằng cách xác định các ký hiệu trong tệp dữ liệu của các containers của sinh

viên.

1.5.2.1 Cú pháp của tệp cấu hình cá nhân hóa: parameter.config

Hệ thống sẽ thay thế các ký hiệu này bằng các giá trị ngẫu nhiên dành riêng cho từng sinh viên. Các ký hiệu này sẽ được khai báo và xác định trong tệp config/parameter.config.

Các toán tử được sử dụng:

1. RAND_REPLACE

RAND_REPLACE : <tên tệp> : <ký tự> : <Cận dưới> : <Cận trên> : <Bước nhảy>

Toán tử RAND_REPLACE có chức năng sẽ thay thế ký tự <ký tự> trong tệp <tên tệp> bằng một giá trị ngẫu nhiên trong khoảng <Cận dưới> và <Cận trên>.

Trong đó:

- <tên tệp>: là tên tệp (tệp này phải tồn tại). <tên tệp> có định dạng như sau:

<tên vùng chứa> : <tên tệp1> ; <tên tệp2> ; ...

Có thể nhiều tệp cùng lúc, được phân cách với nhau bằng dấu “;”.

- <ký tự>: là chuỗi hoặc xâu ký tự sẽ được thay thế.
- <Cận dưới> và <Cận trên> chỉ ra khoảng giá trị sẽ được sinh ngẫu nhiên.
- <Bước nhảy>: (không bắt buộc) Ví dụ:

tham_số : RAND_REPLACE : client:/home/ubuntu/stack.c : BUFFER_SIZE : 200 : 2000

Chuỗi ký tự “BUFFER_SIZE” có trong stack.c của container client sẽ được thay thế bằng một giá trị ngẫu nhiên giữa khoảng 200 – 2000.

2. RAND_REPLACE_UNIQUE

RAND_REPLACE_UNIQUE : <tên tệp> : <ký tự> : <Cận dưới> : <Cận trên>

Cũng tương tự như RAND_REPLACE là thay thế bằng các giá trị, tuy nhiên các giá trị được chọn sẽ không được sử dụng lại. Phép thay thế này tương đương đối với các địa chỉ IP.

3. HASH_CREATE

HASH_CREATE : <tên tệp> : <chuỗi>

Mã băm của <chuỗi> và chuỗi seed của LAB_MASTER_SEED trong tệp start.config sẽ được tạo hoặc ghi đè lên tệp <tên tệp>.

Trong đó:

- <tên tệp>: là tên tệp (tệp này phải tồn tại). <tên tệp> có định dạng như sau:

<tên vùng chứa> : <tên tệp1> ; <tên tệp2> ; ...

Có thể nhiều tệp cùng lúc, được phân cách với nhau bằng dấu “;”.

- <chuỗi>: (<chuỗi> sau khi được ghép với seed của bài thực hành) để thực hiện nhúng hóa bằng MD5.

Ví dụ:

some_parameter_id	:	HASH_CREATE	:	:
client:/home/ubuntu/myseed bufferoverflowinstance				

Tệp myseed sẽ được tạo (nếu không tồn tại). Trong tệp chứa mã băm MD5 (của chuỗi seed và chuỗi “bufferoverflowinstance”)

4. HASH_REPLACE

HASH_REPLACE : <tên tệp> : <ký tự> : <chuỗi>
--

Mã băm MD5 của <chuỗi> và seed sẽ thay thế ký tự <ký tự> có trong tệp <tên tệp>.

Trong đó:

- <tên tệp>: là tên tệp (tệp này phải tồn tại). <tên tệp> có định dạng như sau:

<tên vùng chứa> : <tên tệp1> ; <tên tệp2> ; ...

Có thể nhiều tệp cùng lúc, được phân cách với nhau bằng dấu “;”.

- <chuỗi>: (<chuỗi> sau khi được ghép với seed của bài thực hành) để thực hiện mã hóa bằng MD5.
- <ký tự>: chuỗi ký tự sẽ được thay thế bằng mã băm.

1.5.2.2 Cá nhân hóa tệp start.config

Việc cá nhân hóa tệp start.config sẽ được thực hiện trước khi Docker container được tạo. Hệ thống sẽ tạo một bản sao và lưu trữ ở /tmp/start.config. Hiện tại, chỉ địa chỉ IP trong tệp start.config được cá nhân hóa.

1.5.2.3 Cách cá nhân hóa đơn giản

Với hệ thống Labainers, cách đơn giản nhất mà hệ thống dùng đó là cá nhân hóa một vài tệp của một số container trong bài thực hành và sau đó sẽ kiểm tra các tệp đó khi thực hiện khâu chấm điểm tự động (gradelab). Hệ thống sẽ tự động thực hiện việc này với mọi bài thực hành tạo ra, vì thế hầu hết sẽ phát hiện những sinh viên không trung thực.

1.5.3 Đánh giá tự động

Bên cạnh mô đun cá nhân hóa, hệ thống còn có mô đun đánh giá tự động hay là chấm điểm tự động. Trong quá trình làm bài thực hành, hệ thống sẽ tạo ra một tệp lưu kết quả, thao tác của sinh viên với bài thực hành (artifact).

1.5.3.1 Tệp lưu kết quả Trong tệp lưu kết quả này (.lab), sẽ ghi lại log và .bash_history cũng như mốc thời gian thực hiện ví dụ như dữ liệu được hiển thị sau khi chạy một chương trình nào đó (được gọi là stdout) hoặc lệnh, cú pháp để khởi chạy chương trình (stdin).

stdin và stdout của chương trình và tiện ích sẽ được ghi lại nếu tên của chương trình, tiện ích được bắt đầu ở đầu dòng (ví dụ: nmap, ping, msfconsole, ...) được khai báo trong tệp treataslocal ở

```
$LABTAINER_DIR/labs/[labname]/[container name]/_bin/treataslocal
```

Ví dụ một nội dung trong tệp **treataslocal**:



Hệ thống sẽ ghi lại các chuẩn đầu ra và vào của chương trình tcpdump và arp. Từ đó có thể sử dụng cho việc đánh giá kết quả tự động. Hệ thống thu thập tất cả lịch sử bash của sinh viên vào tệp \$HOME/.bash history và /root/.bash history.

1.5.3.2 Tiêu chuẩn đánh giá - results.config

Các hàm đánh giá tự động cho bài thực hành được thiết kế cho các mục tiêu riêng biệt. Đối với mỗi mục tiêu, người phát triển bài thực hành xác định một hoặc nhiều trường hoặc thuộc tính cho các kết quả theo ý định của giảng viên mong muốn sinh viên hoàn thành. Những cấu hình này được xác định trong tệp:

```
labtainer/trunk/labs/<lab>/instr_config/results.config
```

Các trường giá trị trong tệp results.config:

Mỗi lệnh trong tệp results.config có định dạng sau:

```
<result> = <file_id> : <field_type> : <field_id> [: <line_type> : <line_id>]
```

Trong đó:

- Result: Được hiểu là tên biến hay tên tượng trưng của kết quả. Phải là chữ và số, có thể chứa dấu gạch dưới “_”.
- File_id: xác định nội dung ở một tệp sẽ được so sánh. file_id theo định dạng sau:

```
[container_name:]<prog>.[stdin | stdout | prgout]
```

Trong đó, prog là tên một chương trình hay một tiện ích. Container_name xác định vùng chứa (container) lưu trữ tệp hay Docker container chạy chương trình đó. Ký tự “*” có thể được sử dụng thay cho prog, tức là *.stdin ghi lại log đầu vào của tất cả các chương trình và *.stdout ghi lại log đầu ra của tất cả các chương trình đã được khai báo trong file treataslocal.

- **Field_type:** các loại `field_type` dưới đây được sử dụng để xác định các trường trong một dòng đã chọn trong tệp. Sau dòng đó được xác định, thì `field_type` và `field_id` sẽ xác định giá trị trong dòng đó.
 - **TOKEN:** dòng này là mã thông báo được phân cách bằng dấu cách
 - **PARENS:** giá trị cần tìm được chứa trong dấu ngoặc đơn “()”
 - **SLASH:** Giá trị mong muốn nằm trong dấu gạch chéo, ví dụ: `/foo/`
 - **SEARCH:** Kết quả được gán giá trị `search` được định nghĩa bằng `field_id`, giá trị này được coi là một biểu thức có cú pháp của hàm `parse.search` của python. Ví dụ: `frame.number=={:d}` sẽ nhận được số `frame`.
 - **GROUP:** Được sử dụng với các loại **REGEX**, kết quả được đặt thành các thứ tự nhóm biểu thức chính quy được đặt theo “`field_id`”.
- **Line_type:** Mỗi `field_type` bên trên sẽ phải đi kèm với `line_type` và `line_id` để định vị dòng trong một tệp. Giá trị của `line_type` có các giá trị sau:
 - **LINE:** `line_id` là thứ tự dòng (bắt đầu từ 1).
 - **STARTSWITH:** `line_id` là một chuỗi. Những ký tự xuất hiện đầu tiên của một dòng bằng chuỗi đó.
 - **HAVESTRING:** `line_id` là một chuỗi. Có nghĩa dòng này có chứa một xâu.
 - **REGEX:** `line_id` là một biểu thức chính quy. Cái này sẽ khớp với biểu thức chính quy được khớp đầu tiên.
 - **NEXT_STARTSWITH:** `line_id` là một chuỗi. Dòng đứng trước của dòng này bắt đầu bằng một chuỗi đang nhập.
 - **HAVESTRING_TS:** mục đích sử dụng với các tệp nhật ký đã được gán dấu thời gian. Mỗi mục chứa chuỗi được xác định trong `line_id` sẽ có kết quả được lưu giá trị gán với mốc thời gian.
 - **REGEX_TS:** tương tự như **HAVESTRING_TS**, nhưng là biểu thức chính quy, bao gồm cả tùy chọn sử dụng trường **GROUP** `field_type`.
- **Line_id:** có thể là một giá trị được cá nhân hóa từ tệp `param.config`. Bắt đầu bằng “\$”.
- **Field_type (không có line_id):** Các `field_types` thực hiện trên toàn bộ tệp, không chỉ trên dòng đã chọn. Vì thế sẽ không có trường `line_type` hoặc `line_id`.
 - **LINE_COUNT:** số lượng dòng trong file.
 - **SIZE:** Dung lượng của tệp
 - **CHECKSUM:**
 - **CONTAINS:** Nếu tệp chứa chuỗi được nhập trong trường `field_id` thì giá trị trả về **TRUE**.
 - **FILE_REGEX:** Nếu tên tệp khớp với giá trị của biểu thức chính quy thì sẽ trả về **TRUE**.

- LOG_TS: Kiểu này được sử dụng trong tệp nhật ký, mỗi dòng trả về TRUE nếu dòng đó chứa giá trị trong trường field_id.
- FILE_REGEX_TS: tương tự như LOG_TS, tuy nhiên sử dụng biểu thức chính quy.
- LOG_RANGE: Giống như LOG_TS, tuy nhiên sẽ chỉ trong khoảng thời gian nhất định.
- RANGE_REGEX: Tương tự như LOG_RANGE, nhưng sử dụng biểu thức chính quy.
- STRING_COUNT: Số lần xuất hiện của chuỗi trong trường field_id.
- COMMAND_COUNT: Được dùng với các tệp bash_history, đếm số lần xuất hiện của lệnh được cung cấp trong trường field_id.
- Filed_id: Tùy thuộc vào field_type. Nếu field_type là SEARCH thì file_id sẽ là biểu thức tìm kiếm. Nếu field_type là “CONTAINS” thì field_id sẽ được coi là chuỗi.

Ví dụ:

```
fileview = client:telnet.stdout : \ 4 : STARTSWITH : My string is:
```

Cú pháp trên gán cho fileview một giá trị bằng với mã thông báo được phân tách bằng dấu cách thứ tư trên dòng đầu tiên bắt đầu bằng chuỗi “My string is:” trong đầu ra của chương trình có tên telnet. Các kết quả được trích xuất từ các lệnh, thao tác của sinh viên được so sánh với các giá trị kết quả dự kiến để xác định xem các mục tiêu đã được đáp ứng hay chưa. Các mục tiêu đánh giá là đúng hoặc sai.

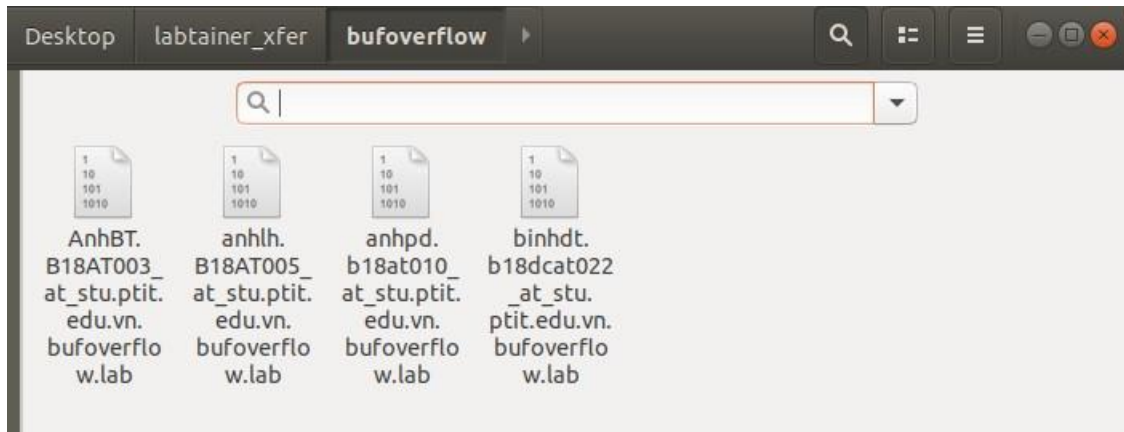
- Ví dụ:

```
telnetview = matchany : \ string_equal  
: fileview : \ parameter.FSTRING
```

Sẽ chỉ ra rằng sinh viên đã đạt được mục tiêu “telnetview” nếu bất kỳ ký hiệu “fileview” nào khớp với giá trị của tham số FSTRING. Trong ví dụ này, tham số FSTRING là chuỗi ký tự thu được sau khi đã qua mã băm MD5, chuỗi ký tự này là duy nhất cho mỗi sinh viên.

1.5.3.3 Cách thức đánh giá kết quả tự động

Sau khi nhận được các file đóng gói của sinh viên, giảng viên sẽ chuyển những tệp đó vào thư mục labtainer_xfer trong Labtainers. Ví dụ: các file đóng gói của bài thực hành bufferoverflow của sinh viên sẽ được chuyển vào trong thư mục có tên giống tên bài thực hành.



Hình 1.9: File đóng gói kết quả của sinh viên được chuyển vào thư mục như tên bài thực hành.

Để tiến hành chấm điểm tự động sẽ truy cập vào đường dẫn thư mục:

```
cd ~/labtainer/trunk/scripts/labtainer-instructor
```

Sử dụng lệnh **gradelab** để chấm điểm tự động (như Hình 1.10):

```
gradelab <tên bài thực hành>
```

```
student@ubuntu: ~/labtainer/labtainer-instructor
File Edit View Search Terminal Help
student@ubuntu:~/labtainer/labtainer-instructor$ gradelab bufoverflow
Labname bufoverflow

Student | gain_root_priv | while_run | stack_protect |
=====|=====|=====|=====|
AnhBT.B18AT003_at_st | Y | Y | Y |
anhlh.B18AT005_at_st | Y | Y | Y |
anhp.d.b18at010_at_st | | Y | Y |
binhdt.b18dcat022_at | | Y | Y |
What is automatically assessed for this lab:

gain_root_priv: Did the student get a root shell & display the /root/.se
cret file?
while_run: Did the student run the whilebash.sh with aslr on?
stack_protect: Experimented with enabling stack guard?
student@ubuntu:~/labtainer/labtainer-instructor$
```

Hình 1.10: gradelab <tên bài thực hành>

Để xem chi tiết về kết quả của từng sinh viên, sử dụng cờ -w với lệnh gradelab (Hình 1.11). Khi đó hệ thống chấm điểm của Labtainers sẽ được cập nhật ở cổng 8008 trên trình duyệt và mở đường dẫn <http://localhost:8008> để xem kết quả (Hình 1.12).


```

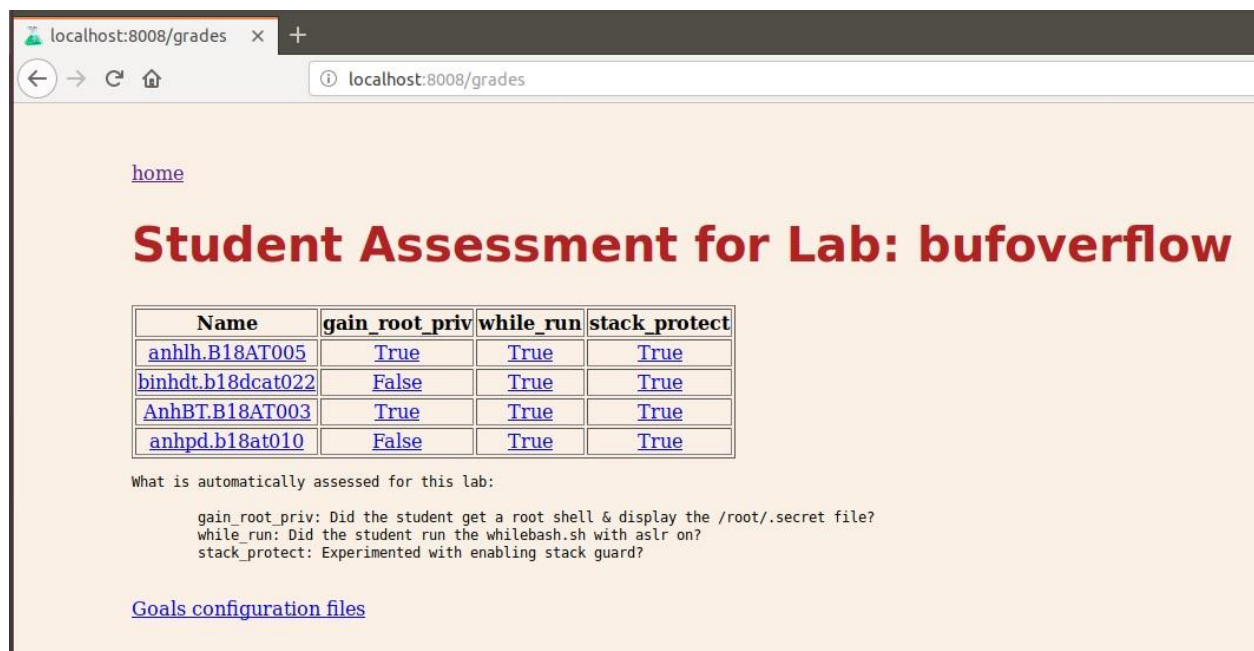
student@ubuntu:~/labtainer/labtainer-instructor$ gradelab bufoverflow -w
Labname bufoverflow

Student      | gain_root_priv | while_run | stack_protect |
===== | ===== | ===== | ===== |
AnhBT.B18AT003_at_st | Y | Y | Y |
anhlh.B18AT005_at_st | Y | Y | Y |
anhpd.b18at010_at_st |  | Y | Y |
binhdt.b18dcat022_at |  | Y | Y |
What is automatically assessed for this lab:

gain_root_priv: Did the student get a root shell & display the /root/.secret file?
while_run: Did the student run the whilebash.sh with aslr on?
stack_protect: Experimented with enabling stack guard?
Point your browser to http://localhost:8008
student@ubuntu:~/labtainer/labtainer-instructor$

```

Hình 1.11: Sử dụng cờ -w với lệnh gradelab



Hình 1.12: Kết quả được hiển thị trên trình duyệt localhost:8008

Chương một của đồ án trình bày tổng quan về kiến trúc môi trường thực hành Labtainers dựa trên docker container, các file môi trường thực thi bài thực hành (Dockerfile, start.config). Đồng thời tìm hiểu chi tiết các mô đun cá nhân hoá, đánh giá tự động. Chi tiết cách xây dựng và quản lý bài thực hành sẽ được trình bày trong chương hai.

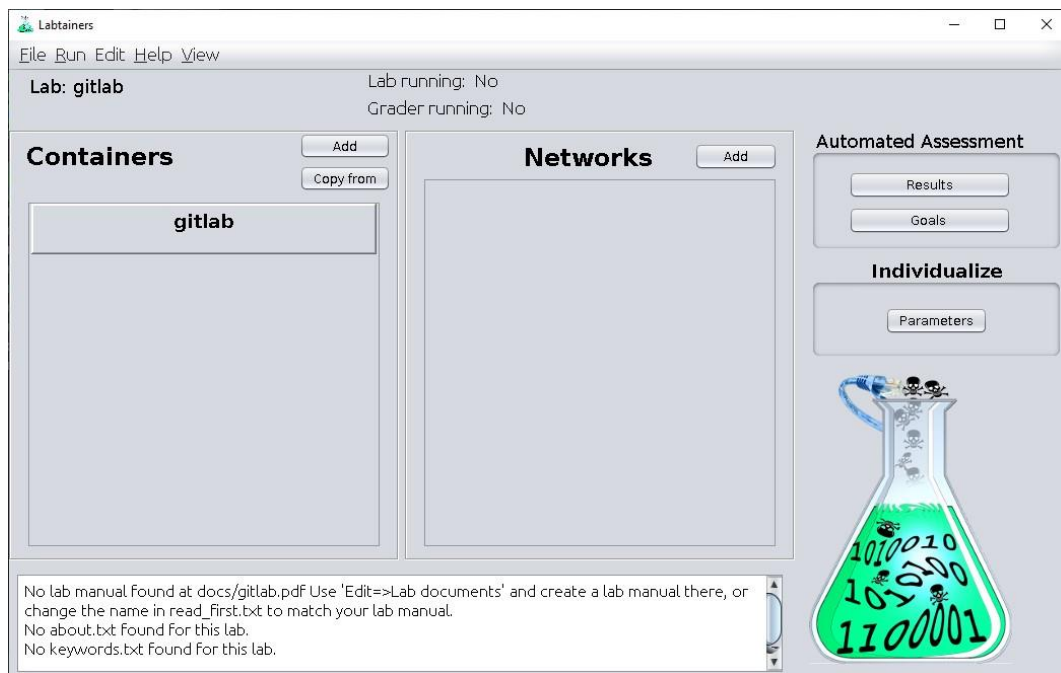
Chương 2. Xây dựng và quản lý bài thực hành

Người thiết kế bài thực hành thường bị trở ngại khi làm việc với dòng lệnh (command line) và việc quản lý các bài thực hành sẽ khó khăn để học sinh tiếp cận với bài thực hành.

Với labtainer, hệ thống hỗ trợ giao diện dễ tiếp cận với người thiết kế và dễ dàng quản lý bài thực hành khi có thể lưu trữ các vùng chứa của lên DockerHub.

2.1 Cách xây dựng một bài thực hành mới

Nền tảng thực hành số cung cấp chức năng tạo bài thực hành dựa trên chương trình Lab Editor của Labtainer. Chương trình này cung cấp giao diện đồ họa để tạo bài thực hành. Người dùng cần chạy lệnh labedit trong terminal để truy cập giao diện này. Trong trường hợp không mở được labedit



Hình 2.1: Giao diện labedit

Trong trường hợp lệnh labedit không chạy được hay không nhận diện được cú pháp thì hãy tới chuyên đường dẫn tới thư mục labtainer, sau đó sử dụng update-designer.sh.

```
~/labtainer$ ./update-designer.sh ~/labtainer$ bash ~/labtainer$ labedit
```

2.1.1 Tạo bài thực hành

Dưới đây sẽ mô tả sơ lược các bước cơ bản để tạo mới và cấu hình cho một bài thực hành về cách dùng git.

Để tạo bài thực hành, vào File => New Lab sẽ có giao diện này:



Hình 2.2: Đặt tên cho bài thực hành mới

Theo mặc định, có hai docker image cơ sở base:

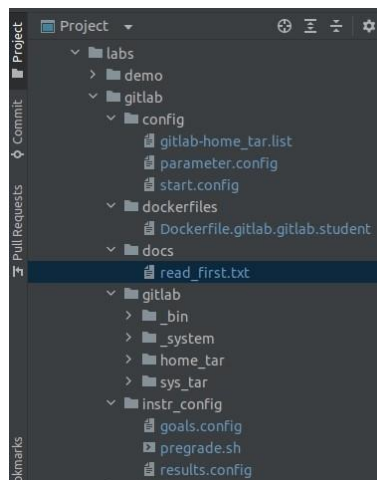
```
student@student-Virtual-Machine:~/labtainer/labtainer-student$ docker images | grep
base
labtainers/labtainer.base2      latest       739eee77f8d3   19 months ago   1.24GB
labtainers/labtainer.base      latest       cb48e785f464   4 years ago     714MB
```

Hình 2.3: Docker image cơ sở base

Tất cả các bài thực hành đều có dockerfiles, tệp này sẽ tạo ra một image khi sinh viên khởi chạy bài thực hành. Đối với bài thực hành được tạo mới, ta cần các phần phụ thuộc do không được sửa đổi các image cơ sở. Do đó, cần phải sửa đổi các dockerfile của bài thực hành. Để làm điều này, nhấp vào tên của bài thực hành rồi thư mục docker, chỉnh sửa dockerfile.

Giảng viên cũng sẽ phải thêm tài liệu để sinh viên đọc khi bắt đầu công việc thực hành, chẳng hạn như báo cáo về công việc thực tế. Để làm điều này, chỉ cần tạo một thư mục /docs trong .../trunks/labs/testlab/ và thêm mọi thứ sinh viên cần cho quá trình thực hành.

Trong ví dụ về gitlab, chúng tôi đã thêm một tài liệu read_first.txt trong đó mô tả một bổ sung mà sinh viên phải thực hiện.



Hình 2.4: File read_first.txt

2.1.2 Cấu hình cho đánh giá tự động

Để hoàn thành bài thực hành, ta cần nắm bắt các thành phần đối tượng và các quy tắc cần để thực hiện tính điểm thực hành tự động.

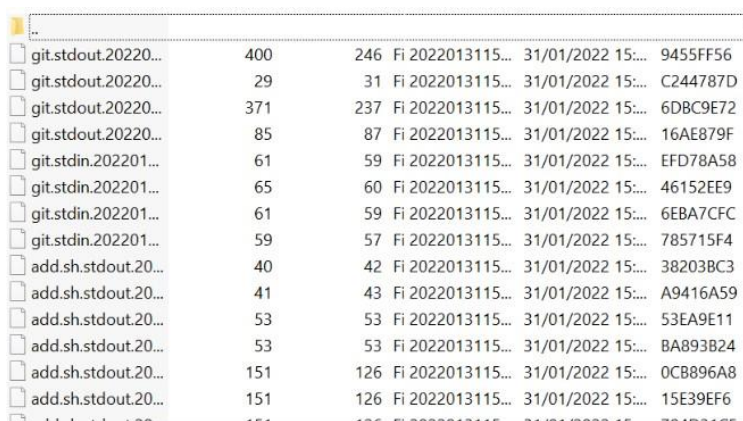
Để có được các thành phần mới trong bài thực hành, nếu đó là thiết bị xuất chuẩn của chương trình hệ thống, ta phải thêm tên chương trình định cho vào tệp treataslocal trong **trunk/labs/[tên bài thực hành]/[tên vùng chứa]/_bin/ treataslocal**.

Ví dụ: đối với bài thực hành gitlab, ta cần đầu vào và đầu ra của git, để kiểm tra xem các lệnh git có được thực hiện hay không, do đó, cần thêm tên của git vào tệp. Hơn nữa, ta cũng có thể truy xuất stdin và stdout của tệp chương trình. Ví dụ: trong bài thực hành sẽ cần tạo một script add.sh để thực hiện các phần bổ sung trong bash, ta có thể thêm tên của tệp add.sh vào tệp treataslocal để bắt tất cả stdin và stdout của bash script này.

```
trunk > labs > gitlab > gitlab > _bin > ≡ treataslocal
1  git
2  add.sh |
```

Chú ý quan trọng: File treataslocal PHẢI có ký hiệu kết thúc file là LF, chứ không dùng CRLF.

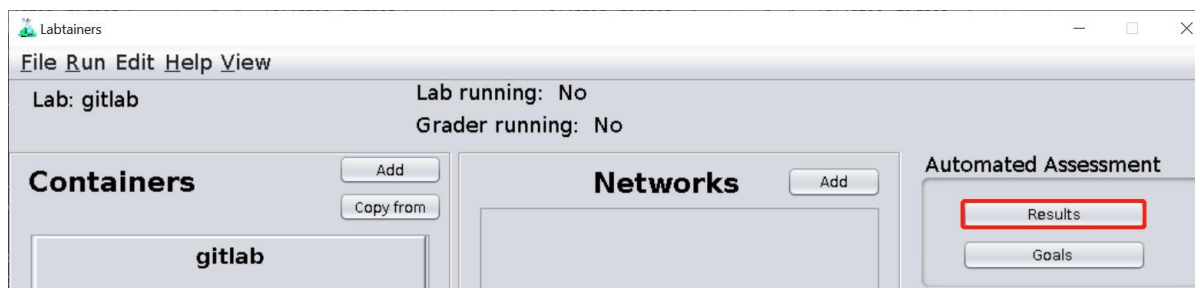
Kết quả của các lệnh này được lưu trữ trong thư mục [địa chỉ mail của sinh viên đã khai báo]=[tên bài thực hành].student.zip/.local/result



File Name	Size	Permissions	Identifier
git.stdout.202201...	400	246	Fi 2022013115...
git.stdout.202201...	29	31	Fi 2022013115...
git.stdout.202201...	371	237	Fi 2022013115...
git.stdout.202201...	85	87	Fi 2022013115...
git.stdin.202201...	61	59	Fi 2022013115...
git.stdin.202201...	65	60	Fi 2022013115...
git.stdin.202201...	61	59	Fi 2022013115...
git.stdin.202201...	59	57	Fi 2022013115...
add.sh.stdout.20...	40	42	Fi 2022013115...
add.sh.stdout.20...	41	43	Fi 2022013115...
add.sh.stdout.20...	53	53	Fi 2022013115...
add.sh.stdout.20...	53	53	Fi 2022013115...
add.sh.stdout.20...	151	126	Fi 2022013115...
add.sh.stdout.20...	151	126	Fi 2022013115...

Hình 2.5: kết quả lưu trữ trong thư mục student.zip/.local/result

Sau khi tất cả kết quả được tạo ra, ta có thể xác định kết quả mà chúng ta muốn biết để đánh giá bài thực hành. Thao tác này phải được thực hiện với giao diện labedit.



Hình 2.6: Results trên giao diện labedit

Chuyển tới tab Results:

The screenshot shows a web interface titled "Results for gitlab". At the top, there are "Create" and "Remove All" buttons. Below is a table with columns: "Result Tag", "Container", "File", "Field Type", "Field ID", "Timestamp Type", "Line Type", "Line ID", and "Timestamp Type". There are four rows of configuration, numbered 1 to 4.

	Result Tag	Container	File	Field Type	Field ID	Timestamp Type	Line Type	Line ID	Timestamp Type
1	git_init	gitlab	git.stdin	CONTAINS	init	File			
2	new_file	gitlab	.bash_history	CONTAINS	nano add.sh	File			
3	new_file_2	gitlab	.bash_history	CONTAINS	touch add.sh	File			
4	add	gitlab	add.sh.stdout	TOKEN	1		HAVESTRING	Usage	File

At the bottom of the table, there are "OK" and "Cancel" buttons.

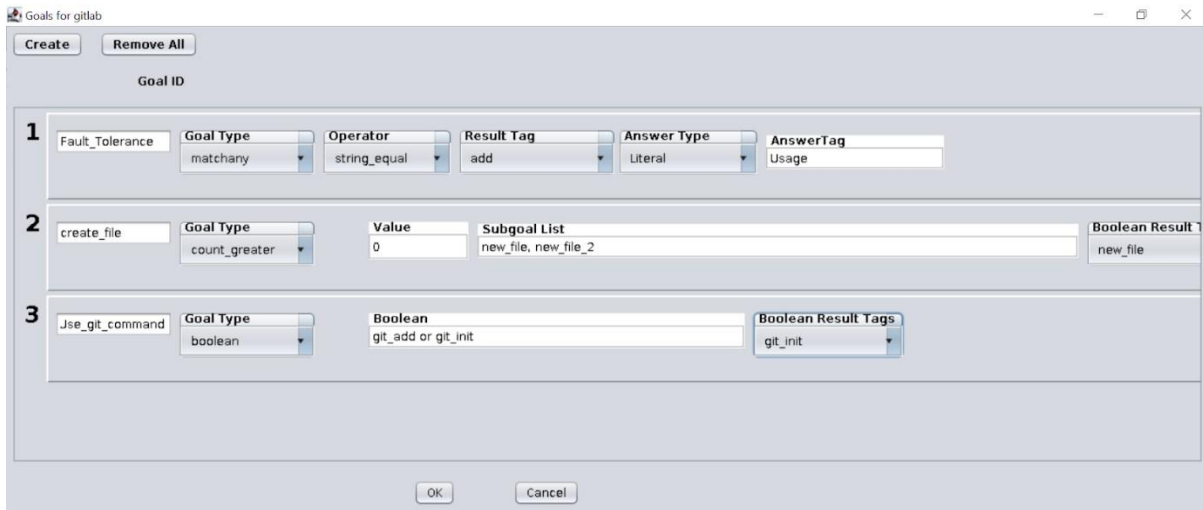
Hình 2.7: Giao diện result của gitlab

Các tiêu chí này được xác định theo cách sau (chi tiết cú pháp của xác định kết quả được trình bày ở [6.3.2. Tiêu chuẩn đánh giá - results.config](#)):

- Result tag: là tên của kiểm tra được sử dụng trong định nghĩa Goals.
- Container: là tên của container muốn lấy thông tin.
- File: là tên của tệp mà muốn lấy thông tin.
- Filed Type: là loại so sánh muốn tiến hành. Có 2 loại. Loại đầu tiên là các thao tác được thực hiện trên toàn bộ tệp như ví dụ trên với CONTAINS. Loại thứ hai là nơi phải chỉ định dòng để thực hiện thao tác. Ví dụ tới tag “add”. Ta sẽ truy xuất từ đầu tiên của dòng (Field ID là số từ muốn truy xuất) từ các tệp add.sh.stout có chứa từ (HAVESTRINGS) “Usage”.
- Field ID: trường thay đổi tùy theo Filed Type, nếu Filed Type là CONTAINS hoặc FILE_REGEX thì Filed ID là mẫu ta muốn tìm kiếm.

Để có nhiều thông tin tham khảo, cần đọc thêm tài liệu của Labtainer để biết chính xác hơn, trong phần 6.2.1 trang 28 của labdesigner.pdf - <https://nps.edu/web/c3o/labtainers>.

Từ những kết quả này, ta có thể xác định mục đích của công việc thực tế. Đó chính là kết quả mà ta phải đạt được ở cuối bài thực hành. Ví dụ trong bài thực hành gitlab này, ta phải sử dụng lệnh git init cũng như lệnh git add. Vì vậy, trong giao diện định nghĩa Goals, có thể định nghĩa một mục tiêu như Use_git_command trong hình bên dưới:



Hình 2.8: Giao diện goals của gitlab

Trong định nghĩa Goal, cũng có một số trường:

- Goal ID: là tên của mục tiêu.
- Goal Type: là loại hoạt động ta muốn thực hiện trên các kết quả theo các loại khác nhau mà ta có thể có các trường khác nhau.

Đối với kiểu matchany, matchlast hoặc matchacross, ta phải xác định một toán tử của kết quả ta đã truy xuất ở trên và so sánh nó với một biểu thức bằng chữ đã xác định như trong mục tiêu đầu tiên của hình. Ở ví dụ này, ta đã lấy kết quả “add” được xác định trước đó và so sánh nó với chuỗi “Usage”.

Đối với kiểu count_greater, ta sẽ xác định Subgoal List chứa Goal ID được xác định ở trên hoặc Result Tag đã xác định trong tab Result và giá trị nằm trong khoảng từ 0 đến độ dài của danh sách trừ đi một. Toán tử sẽ đếm số lượng True trong danh sách, nếu số lượng này vượt quá giá trị mà ta đã xác định thì mục tiêu này sẽ trả về True ngược lại sẽ là False.

Nhận xét: danh sách phải được phân tách bằng dấu phẩy + dấu cách. Tức là el_1, el_2, el_3 chứ không phải el_1,el_2,el_3! Thêm nữa, Result tag Boolean hoàn toàn vô dụng, nó hoàn toàn không ảnh hưởng đến kết quả của Goal và việc chọn một tag cụ thể cũng vô ích, vì khi mở lại tab, tag sẽ được đặt lại trên tag đầu tiên.

Đối với kiểu boolean, ta phải đặt một biểu thức boolean trong Boolean và có thể bao quanh biểu thức bằng dấu ngoặc đơn hoặc không. Tương tự như vậy, tag kết quả Boolean là cũng vô dụng ở đây.

2.1.3 Cá nhân hóa cho bài thực hành

Hệ thống cung cấp khả năng cá nhân hóa các giá trị hoặc tệp băm để người học có thể có các mục tiêu khác nhau. Để thực hiện việc này, trong giao diện tạo bài thực hành, trong phần Individualize, vào Parameters, chọn tab cá nhân hóa.

Như có thể thấy trong hình bên dưới, cá nhân hóa phải chứa (chi tiết thêm về cấu trúc của mô đun được trình bày ở [6.2. Cá nhân hóa](#)):

- Param ID: là tên của cá nhân hóa
- Operator: là kiểu thay thế mà chúng ta muốn thực hiện, ví dụ: thay thế biến bằng một số nguyên hoặc mã băm, v.v.
- File name: là tên file chứa biến cần thay thế (nút cho phép chọn file có giao diện người dùng)
- Symbol: là biến cần thay thế.
- Lower bound: Giới hạn dưới là giá trị nhỏ nhất mà biến có thể nhận
- Upper bound: Giới hạn trên là giá trị lớn nhất mà biến có thể nhận
- Step: là bước mà tại đó chúng ta có thể di chuyển từ giá trị nhỏ nhất đến giá trị lớn nhất.

Param ID	Operator	File name	Symbol	Lower bound	Upper bound	Step
first	RAND_REPLACE	doc/description.txt	FIRST_VALUE	1	20	1
second	RAND_REPLACE	doc/description.txt	SECOND_VALUE	20	40	1

Hình 2.9: Giao diện cá nhân hóa bài thực hành gitlab trên labedit

Trong ví dụ này, trong tệp description.txt, ta thay thế biến FIRST_VALUE bằng một số nguyên từ 1 đến 20 và SECOND_VALUE bằng một số nguyên từ 20 đến 40. Sự ngẫu nhiên này được tạo khi image được tạo, vì vậy nó không chỉ chạy một lần lúc đầu khi chưa có image bài thực hành.

2.2 Quản lý bài thực hành

Sau khi đã thiết kế được bài thực hành thì chúng ta sẽ lưu những images của bài thực hành trên <https://hub.docker.com/> để những images này sẽ được tải về máy tính của sinh

viên khi làm bài thực hành. Các thực hiện cụ thể được nêu ở dưới:

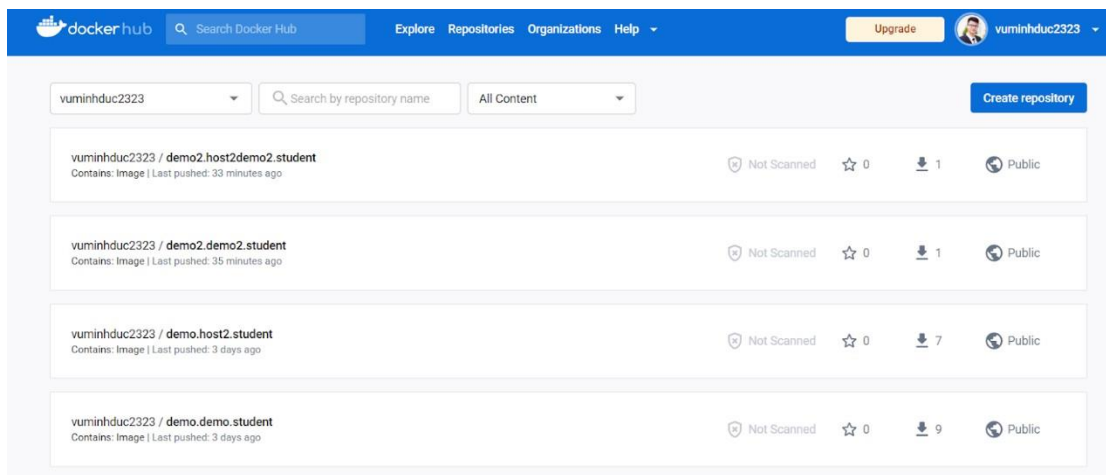
- Chuyển tới thư mục chứa các bài thực hành: **labtainer/trunk/labs**

- 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 ví dụ bài thực hành mới là bai-lab-moi 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 bai-lab-moi:

```
git add bai-lab-moi
git commit bai-lab-moi -m "Adding an IModule"
```

- Đẩy images của vùng chứa (container) lên DockerHub

```
cd $LABTAINER_DIR/distrib
./publish.py -d -l my-new-lab
```



Hình

2.10: Các image vùng chứa (container) được tải lên DockerHub.

- Tạo file IModule tar chứa bài thực hành:

```
create-imodules.sh
```

- Sau đó, copy và lưu lại file imodule.tar và chia sẻ đường dẫn URL tới sinh viên để thực hiện bài thực hành.

2.3 Xây dựng mô đun hỗ trợ đánh giá kết quả: .json sang .csv

Để thuận tiện cho giảng viên trong quá trình đánh giá thì tôi xây dựng mô đun chuyển đổi dữ liệu từ .json sang .csv. Sau khi chạy xong quá trình chấm điểm tự động thì trong thư mục labtainer_xfer/<tên bài thực hành>, trong ví dụ này là labtainer_xfer/bufoverflow được tạo ra file bufoverflow.grades.json. Để tạo ra file csv cần cài đặt chương trình python có tên jsonToCSV.py (tên có thể đặt tùy ý) trong thư mục labtainer-instructor. Nội dung của file jsonToCSV.py có thể tải về tại: <https://github.com/Vuduchihi/labtainerlabs/blob/main/jsonToCSV.py> File .grades.json có dạng:

```
{
  "[email của sinh viên].[tên bài thực hành]": {
    "parameter": {
      "rand1": "171",
      "roothash": "d25f175d8087ab759c03088c1868de2c",
      "userhash": "96107ca8d0996ee5ff387a6e9ea9b28d"
    },
    "grades": {
      "result tag": [true/false],
      "result tag2": true,
      ...
    },
    "firstlevelzip": {},
    "secondlevelzip": {},
    "actualwatermark": "ce498d68b00e8459915cf09c2a1493c6",
    "expectedwatermark": "ce498d68b00e8459915cf09c2a1493c6",
    "labcount": {
      "start": [],
      "redo": []
    }
  },
}
```

Đọc dữ liệu từ file json và chuyển đổi thành kiểu dữ liệu danh sách kết quả sinh viên (list):

Dữ liệu của danh sách – list theo định dạng sau:

```
[{sinh viên 1}, {sinh viên 2}, ... ]
```

Trong đó, {sinh viên} có dữ liệu các khóa và giá trị như sau: **‘khóa’**: **‘giá trị’**

- 'email': '[email của sinh viên].[tên bài thực hành]'
- 'tên tiêu chí': [True/False]
- 'score': '[số tiêu chí đã hoàn thành] / [tổng số tiêu chí]'

```
[
  {'email': '[email của sinh viên].[tên bài thực hành]', 'tên tiêu chí': [True/False], 'score':
  '[số tiêu chí đã hoàn thành] / [tổng số tiêu chí]'}
]
```

Có thể có nhiều tiêu chí khác nhau, các tiêu chí được phân cách với nhau bằng dấu “,” Ví dụ:

```
{'email': 'anhlh.B18AT005_at_stu.ptit.edu.vn.bufoverflow',
  'gain_root_priv': True, '_aslr': True, 'while_run': True, 'stack_protect': True,
  '_whiledump': True, '_whileroot': False, 'score': '5/6'}
```

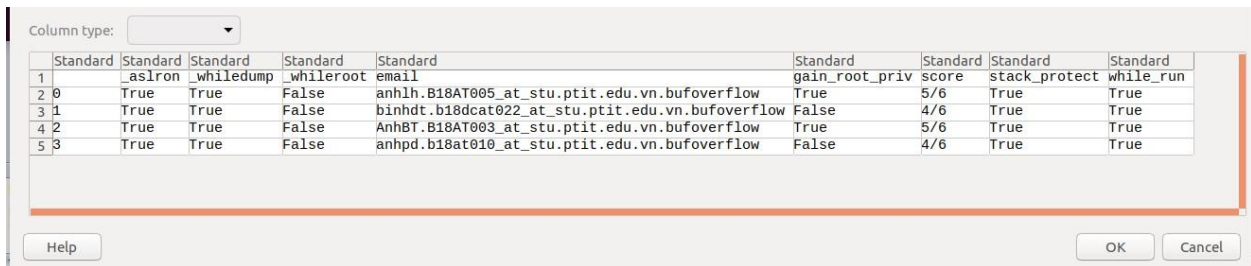
Từ danh sách kết quả sinh viên này, sử dụng thư viện **pandas** Python để chuyển từ dạng danh sách (list) sang dữ liệu kiểu hàng và cột bằng hàm:

```
pandas.DataFrame.from_dict([danh sách kết quả sinh viên])
```

Sau đó ghi dữ liệu ra file có dạng .csv bằng hàm

```
.to_csv().
```

Sau khi file .csv được tạo, giảng viên có thể sao chép, hoặc di chuyển sang máy tính thật của mình để dễ dàng sử dụng để tùy chỉnh, xem xét.



	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard	Standard
1	_aslr	_whiledump	_whileroot	email	gain_root_priv	score	stack_protect	while_run	
2	0	True	True	False	anhlh.B18AT005_at_stu.ptit.edu.vn.bufoverflow	True	5/6	True	True
3	1	True	True	False	binhdt.b18dcat022_at_stu.ptit.edu.vn.bufoverflow	False	4/6	True	True
4	2	True	True	False	AnhBT.B18AT003_at_stu.ptit.edu.vn.bufoverflow	True	5/6	True	True
5	3	True	True	False	anhpd.b18at010_at_stu.ptit.edu.vn.bufoverflow	False	4/6	True	True

Hình 2.11: Kết quả chuyển từ json sang .csv

Như vậy, qua chương 2 đã biết cách tạo mới bài thực hành đơn giản có cấu hình cho cá nhân hoá và đánh giá tự động. Cách quản lý bài thực hành, đẩy các image của bài thực hành lên DockerHub để học sinh lấy dữ liệu về khi làm bài thực hành. Sử dụng mô đun tự xây dựng để tinh chỉnh và chuyển đổi dữ liệu từ dạng .json sang .csv để thuận tiện cho giảng viên đánh giá và xem kết quả. Một số bài thực hành an toàn thông tin sẽ được thiết kế và triển khai trong chương tiếp theo.

Chương 3. Thiết kế một số bài thực hành an toàn thông tin

Thời đại 4.0 hiện nay, vấn đề an toàn thông tin trở nên nóng hơn bao giờ hết. Cuộc chiến không hồi kết giữa tin tặc và các nhà an ninh mạng, những cuộc tấn công của tin tặc vào hệ thống ngày càng gia tăng và gây ảnh hưởng rất lớn tới các tổ chức. Trọng trách của những người an ninh cũng vì thế mà nhiều hơn. Để có thể phòng tránh, ngăn chặn những cuộc tấn công mạng xảy thì bản thân chuyên viên an ninh mạng phải hiểu rõ được quy trình của cuộc tấn công để có thể đưa ra những biện pháp phòng thủ hay vá những lỗ hổng đang tồn tại. Với bài thực hành “Truy tìm mật mã” sinh viên sẽ được tìm hiểu, thực hành về cách rà quét lỗ hổng bằng công cụ Nmap và sử dụng môi trường Metasploit Framework để kiểm tra và thực thi tấn công. Bên cạnh đó hiểu rõ về tầm quan trọng của việc bảo mật thông tin thông qua mật mã Caesar. Để phòng tránh những cuộc tấn công thì chúng ta cần theo dõi và giám sát lưu lượng truyền đi trong môi trường mạng, vì thế cần phải thuần thục những công cụ theo dõi và giám sát mạng thông qua bài thực hành “tcpdump” và “arpspoof & scapy”. Cuối cùng là biện pháp phòng tránh các tin tặc bằng việc giả mạo những hệ thống máy ảo qua bài “Honeypot”. Từ đó, các bạn sinh viên được trải nghiệm đầy đủ mọi quá trình trong quy trình một cuộc tấn công từ đó hiểu rõ tầm quan trọng và có thể vận dụng kiến thức để giữ gìn cho môi trường mạng ngày càng trong sạch, an toàn.

3.1 Danh sách các bài thực hành

STT	Tên bài thực hành	Lý do xây dựng bài thực hành
1	Honeypot	Giúp sinh viên có môi trường thực hành để hiểu về hệ thống honeypot, và cài đặt thành công honeypot
2	Tcpdump	Giúp sinh viên có môi trường thực hành thực tế để hiểu và thành thạo vận dụng sử dụng công cụ thu thập dữ liệu mạng tcpdump
3	arpspoof & scapy	Giúp sinh viên có môi trường thực hành thực tế và khám phá cách sử dụng arpspoof và thư viện scapy python để lắng nghe lưu lượng mạng.
4	Truy tìm mật mã	Giúp Sinh viên có môi trường thực hành thực tế để nắm được quy trình và thực hiện tấn công khai thác lỗ hổng rlogin và hiểu về mật mã caesar.

Bảng 3.1: Danh sách bài thực hành

Các bài thực hành được lưu ở [9]. Cách thực hiện bài thực hành được trình bày ở [Cách thức sử dụng bài thực hành](#).

3.2 Phân tích và thiết kế các bài thực hành

3.2.1 Honeypot

3.2.1.1 Phân tích Với sự phát triển mạng mẽ của internet đã mang lại những lợi ích to lớn trên nhiều lĩnh vực của đời sống xã hội. Tuy nhiên, hiện trạng ngày càng nhiều vụ tấn công vào hệ thống thông tin quan trọng của các quốc gia, đánh cắp thông tin cá nhân và dữ liệu người dùng để sử dụng vào mục đích không chính đáng. Chính vì vậy sinh viên an toàn thông tin cần tìm hiểu về các hệ thống tài nguyên giúp ngăn chặn, phòng ngừa, phát hiện các cuộc tấn công mạng. Bài thực hành để sinh viên tìm hiểu về honeypot – một hệ thống tài nguyên thông tin được xây dựng với mục đích là giả lập, đánh lừa những kẻ hacker, người xâm nhập không hợp pháp.

Honeypots là một hệ thống máy tính được thiết kế để thu thập tất cả hoạt động, tệp mà các kẻ tấn công khởi tạo, có ý định giành quyền truy cập trái phép. Honeypot có thể giả dạng bất cứ loại máy chủ tài nguyên nào như Mail Server, Webserver, ..., được cài đặt chạy trên mọi hệ điều hành [8].

Honeypot gồm hai loại chính khi chia theo mức độ tương tác: Tương tác thấp và tương tác cao

- Honeypot tương tác thấp: Chỉ được thiết kế một số dịch vụ cụ thể, vì thế thông tin thu được không đa dạng. Tuy nhiên dễ triển khai, mức độ rủi ro thấp.
- Honeypot tương tác cao: Được thiết kế chạy tất cả các dịch vụ mà một hệ thống có. Mức độ thông tin quan sát và thu thập cao. Cũng vì thế mà cần tốn rất nhiều tài nguyên và thời gian để vận hành và bảo trì.

Honeyd là một loại hình Honeypot tương tác thấp, có nhược điểm là không thể cung cấp một hệ điều hành hoàn chỉnh đầy đủ dịch vụ để tương tác với tin tặc. Honeyd có thể mô phỏng cùng một lúc nhiều hệ điều hành khác nhau và lắng nghe trên tất cả các cổng TCP và UDP, những dịch vụ mô phỏng được thiết kế với mục đích ngăn chặn và ghi lại những cuộc tấn công tương tác với kẻ tấn công với vai trò một hệ thống nạn nhân.

3.2.1.2 Thiết kế bài thực hành

3.2.1.2.1 Yêu cầu

- Một máy ảo cho sinh viên làm nhiệm vụ tấn công (attacker).
- Một máy ảo để cài Honeyd (server),
- Thiết lập hệ thống mạng sao cho máy tấn công và máy nạn nhân cùng một mạng.
- Các thư viện cần có (lấy từ labedit của Labtainers và github [9])
- Cấu hình docker gồm có:

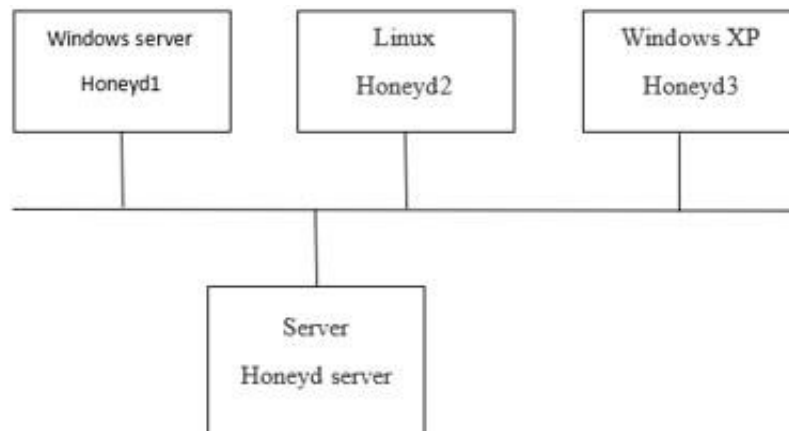
- Attacker: lưu cấu hình cho máy attacker
- Config: lưu cấu hình hoạt động của hệ thống bài thực hành
- Dockerfiles: thêm các gói thư viện (packages), các tham số cần có cho container của attacker và server
- Docs: lưu các mô tả cho bài thực hành
- Instr_config: lưu cấu hình cho phần nhận kết quả và chấm điểm.
- Server: lưu cấu hình cho máy server

3.2.1.2.2 Sơ đồ thiết kế bài thực hành



Hình 3.1: Sơ đồ thiết kế bài thực hành honeypot

Sơ đồ mạng: 1 máy chủ Server, cấu hình honeyd tạo ra thêm 3 máy gồm windows server, máy linux và máy windows XP (xem hình dưới):



Hình 3.2: Sơ đồ mạng bài thực hành honeypot

Chi tiết cấu hình cơ bản của các máy:

- Windows server:

IP: 192.168.2.50

Cổng 135: mở

- Linux:

IP: 192.168.2.51

Cổng 21: mở

- Windows XP:

IP: 192.168.2.52

Cổng 80: mở

3.2.1.2.3 Kịch bản

Hệ thống gồm có 1 máy server và 1 máy tấn công được cài đặt những thư viện cần thiết. Bài thực hành sử dụng hai máy ảo Linux: một máy để cài đặt honeyd (máy server có địa chỉ 192.168.2.2) và cái còn lại là máy dùng để tấn công (máy attacker có địa chỉ 192.168.2.1)

Sinh viên tải file image về chạy trên máy ảo. Sau đó chạy bài thực hành theo hướng dẫn. Trong bài thực hành này, 1 container mang tên “attacker” đóng vai trò là máy tấn công và 1 container mang tên “server” đóng vai trò là máy cài đặt hệ thống honeyd.

Các nhiệm vụ cần thực hiện để cài đặt thành công:

1. Cài đặt honeyd trên máy server. Tải mã nguồn từ git [9], tạo tệp cấu hình để tạo ra 3 honeyd client. Kết quả cần đạt được trên máy server khởi động và hệ thống honeyd đang lắng nghe lưu lượng mạng.
2. Trên máy attacker sử dụng công cụ “nmap” để thực hiện quét các địa chỉ máy có trên mạng. Kết quả cần đạt được trên máy server xuất hiện thông tin địa chỉ ip của các máy ảo do honeyd tạo ra trên máy attacker.

Sau khi sinh viên hoàn thành bài thực hành, hệ thống cần tự động lưu lại kết quả vào 1 file và tự động đóng gói lại để cho sinh viên chỉ cần gửi file đóng gói cho giảng viên.

3.2.2 Tcpcdump

3.2.2.1 Phân tích

Với sự phát triển của internet, nên càng nhiều vụ tấn công vào hệ thống thông tin quan trọng của các quốc gia, đánh cắp thông tin cá nhân thông qua môi trường mạng. Chính vì vậy sinh viên an toàn thông tin cần hiểu về giám sát và theo dõi dữ liệu trên mạng giúp

ngăn chặn, phòng ngừa, phát hiện các cuộc tấn công mạng. Bài thực hành để sinh viên tìm hiểu và vận hành công cụ thu thập dữ liệu mạng: tcpdump.

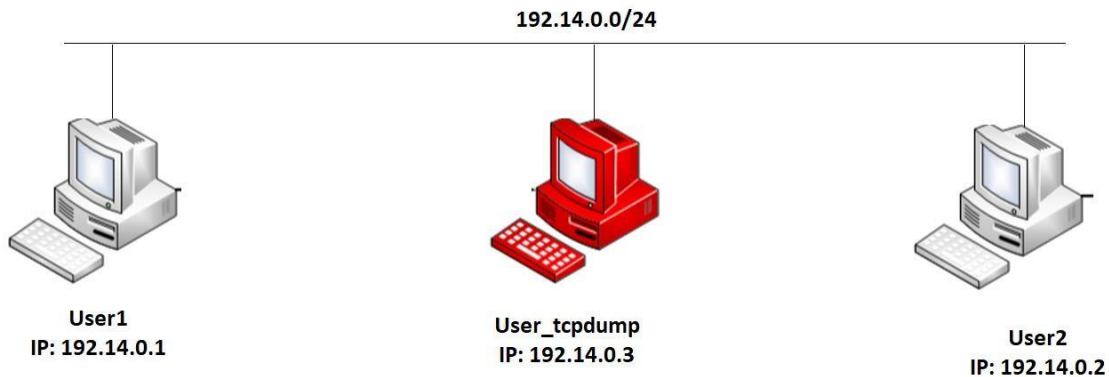
Tcpdump là công cụ hỗ trợ phân tích các gói dữ liệu mạng theo dòng lệnh, cho phép người dùng chặn bắt và lọc các gói tin được lưu chuyển trên mạng. Tcpdump sẽ lưu lại những gói tin bắt được và sau đó dùng để phân tích [10].

3.2.2.2 Thiết kế bài thực hành

3.2.2.2.1 Yêu cầu

- Một máy ảo có công cụ tcpdump (máy Kali Linux cài thư viện tcpdump) cho sinh viên làm nhiệm vụ thu thập trên đường truyền mạng (may_tcpdump).
- Hai máy ảo có cùng một mạng với user_tcpdump.
- Thiết lập hệ thống mạng sao cho máy tấn công và máy nạn nhân cùng một mạng LAN.
- Cấu hình docker gồm có:
 - User_tcpdump: lưu cấu hình cho máy User_tcpdump
 - Config: lưu cấu hình hoạt động của hệ thống bài thực hành
 - Dockerfiles: thêm các gói thư viện (packages), các tham số cần có cho container của attacker và server
 - Docs: lưu các mô tả cho bài thực hành
 - Instr_config: lưu cấu hình cho phần nhận kết quả và chấm điểm.
 - User1: lưu cấu hình cho máy user1
 - User2: lưu cấu hình cho máy user2

3.2.2.2 Sơ đồ thiết kế bài thực hành



Hình 3.3: Sơ đồ thiết kế bài thực hành tcpdump

3.2.2.3 Kịch bản

Hệ thống gồm có 3 máy ảo Linux được cài đặt những thư viện cần thiết. Bài thực hành sử dụng ba máy ảo Linux: một máy để sử dụng công cụ tcpdump (máy User_tcpdump có địa chỉ 192.14.0.3) và 2 cái còn lại là máy dùng tương tác trong mạng (User1: 192.14.0.1 và User2: 192.14.0.2).

Sinh viên tải file image về chạy trên máy ảo. Sau đó chạy bài thực hành theo hướng dẫn.

Các nhiệm vụ cần thực hiện:

1. Tìm hiểu ý nghĩa và chạy các câu lệnh yêu cầu. Kết quả cần đạt được chạy thành công các lệnh và thu được thông tin gói tin trên mạng.
2. Thực hiện ssh từ user1 tới máy User_tcpdump và thực hiện lọc filter gói TCP. Kết quả cần đạt được tcpdump thu được gói tin TCP từ user1.
3. Tương tự, viết các bộ lọc theo yêu cầu để lọc được những gói tin đúng theo quy định.

Sau khi sinh viên hoàn thành bài thực hành, hệ thống cần tự động lưu lại kết quả vào 1 file và tự động đóng gói lại để cho sinh viên chỉ cần gửi file đóng gói cho giảng viên.

3.2.3 Arpspoof & scapy để thực tấn công người ở giữa (man-in-the-middle)

3.2.3.1 Phân tích

Với sự phát triển của internet, nên càng nhiều vụ tấn công vào hệ thống thông tin quan trọng của các quốc gia, đánh cắp thông tin cá nhân thông qua môi trường mạng. Chính

vì vậy sinh viên an toàn thông tin cần hiểu về giám sát và theo dõi dữ liệu trên mạng giúp ngăn chặn, phòng ngừa, phát hiện các cuộc tấn công mạng. Bài thực hành để sinh viên tìm hiểu và sử dụng thư viện: Scapy để giám sát và thu thập dữ liệu mạng.

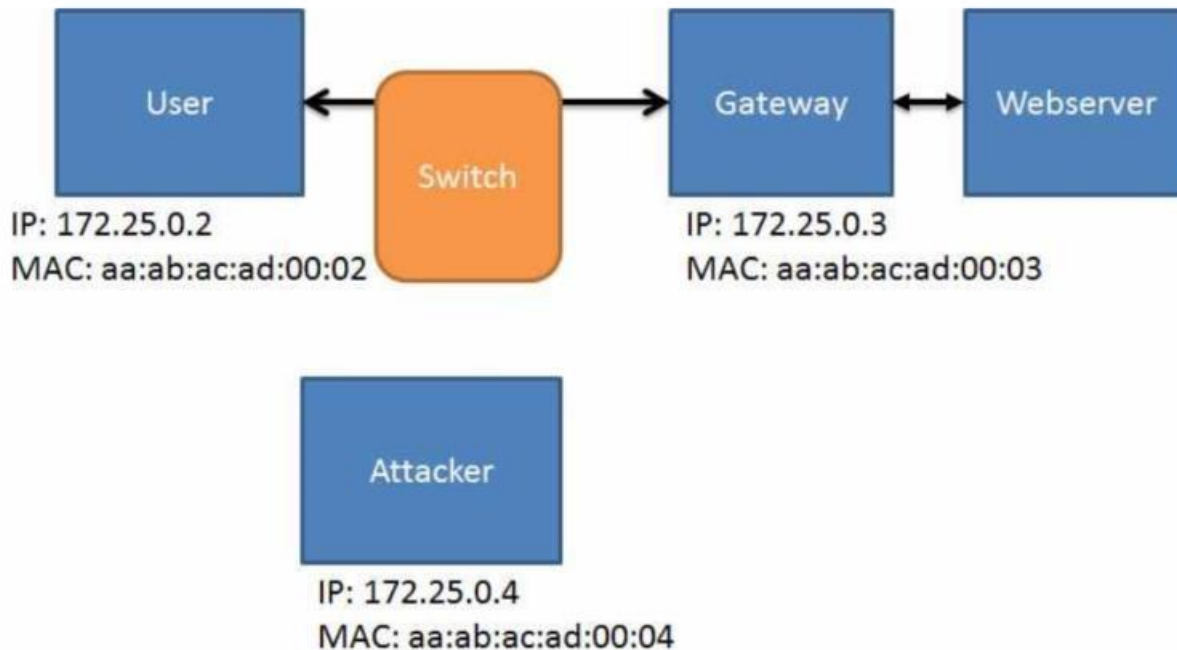
Scapy là một thư viện Python để giám sát, thu thập, phân tích và làm giả mạo gói tin trong mạng. Scapy có thể xử lý các nhiệm vụ cơ bản như rà quét, truy vết các cuộc tấn công mạng. Vì thế trong bài này, sinh viên viết một số chương trình đơn giản sử dụng scapy để thu thập, lọc gói tin trong mạng.

3.2.3.2 Thiết kế bài thực hành

3.2.3.2.1 Yêu cầu

- Một máy ảo có cài đặt thư viện scapy để lắng nghe với vai trò là người đứng giữa – man-in-the-middle (User). Một máy ảo có vai trò Webserver để User có thể truy cập tới trang web, và một máy Attacker để cho sinh viên làm nhiệm vụ thu thập trên đường truyền mạng, một container đóng vai trò là Gateway hỗ trợ cho truy cập tới Webserver.
- Thiết lập hệ thống mạng sao cho máy tấn công và máy nạn nhân cùng một mạng LAN.
- Cấu hình docker gồm có:
 - User: lưu cấu hình cho máy User
 - Config: lưu cấu hình hoạt động của hệ thống bài thực hành
 - Dockerfiles: thêm các gói thư viện (packages), các tham số cần có cho container của attacker và server
 - Docs: lưu các mô tả cho bài thực hành
 - Instr_config: lưu cấu hình cho phần nhận kết quả và chấm điểm.
 - Gateway: lưu cấu hình cho máy Gateway
 - Webserver: lưu cấu hình cho máy Webserver
 - Attacker: lưu cấu hình cho máy Attacker

3.2.3.2.2 Sơ đồ thiết kế bài thực hành



Hình 3.4: Sơ đồ thiết kế bài thực hành arpspoof & scapy

3.2.3.2.3 Kịch bản

Hệ thống gồm có 3 máy ảo Linux được cài đặt những thư viện cần thiết. Bài thực hành sử dụng ba máy ảo Linux: một máy để sử dụng chương trình bởi scappy có vai trò lắng nghe trong mạng (man-in-the-middle) (máy Attacker có địa chỉ 172.24.0.4) máy User hoạt động truy cập trang web tới Webserver (User: 172.24.0.2 và Webserver: 172.35.0.3).

Sinh viên tải file image về chạy trên máy ảo. Sau đó chạy bài thực hành theo hướng dẫn.

Các nhiệm vụ cần thực hiện:

1. Sử dụng công cụ arpspoof để cấu hình máy tính Attacker nhận gói tin trung gian giữa User và Gateweb. Kết quả cần đạt được cấu hình thành công và có dữ liệu trao đổi.
2. Chỉnh sửa chương trình python Task1.1A.py để thu thập gói tin trong mạng.
Sau đó, “ping” tới máy attacker. Kết quả cần đạt được là chương trình thu được gói tin dữ liệu tới địa chỉ đích là attacker
3. Chỉnh sửa chương trình python Task1.1B-icmp.py để thu thập là lọc gói tin ICMP trong mạng. Kết quả cần đạt được là chương trình thu được gói tin dữ liệu có kiểu proto = icmp.

4. Chỉnh sửa chương trình python Task1.1B-icmp.py để thu thập là lọc gói tin TCP trong mạng. Cần sử dụng thêm công cụ khác như ssh để thu được gói tin TCP trên máy Attacker. Kết quả cần đạt được là chương trình thu được gói tin dữ liệu có kiểu tcp.

Sau khi sinh viên hoàn thành bài thực hành, hệ thống cần tự động lưu lại kết quả vào 1 file và tự động đóng gói lại để cho sinh viên chỉ cần gửi file đóng gói cho giảng viên.

3.2.4 Truy tìm mật mã bằng khai thác lỗ hổng rlogin và mật mã caesar.

3.2.4.1 Phân tích

Các lỗ hổng bảo mật tồn tại trong hệ thống máy tính nói riêng và hệ thống mạng nói chung là vấn đề đã và đang được quan tâm, đặc biệt trong mối liên hệ mật thiết với vấn đề nóng hiện nay là đảm bảo an toàn thông tin, hệ thống, mạng. Trên thực tế, gần như không có hệ thống nào là hoàn hảo, không có điểm yếu, hoặc khiếm khuyết. Các hệ thống máy tính, hoặc hệ thống thông tin là các hệ thống rất phức tạp, được cấu thành từ nhiều thành phần phần cứng, phần mềm, do vậy chúng luôn tồn tại các lỗi, các khuyến khuyết, hay các điểm yếu. Các điểm yếu có thể tồn tại trong các mô đun phần cứng cũng như trong các mô đun phần mềm. Nhìn chung, các hệ thống càng phức tạp, có nhiều thành phần với nhiều tính năng thì khả năng xuất hiện các lỗi và điểm yếu ngày càng tăng. Lỗ hổng bảo mật là một điểm yếu tồn tại trong hệ thống cho phép kẻ tấn công khai thác gây tổn hại đến các thuộc tính an toàn, an ninh của hệ thống đó, trong đó bao gồm tính bí mật, tính toàn vẹn và tính sẵn sàng.

Chính vì vậy sinh viên an toàn thông tin cần phải tìm hiểu về các lỗ hổng bảo mật và hiểu được cách khai thác lỗ hổng này - rlogin từ đó hiểu được cách ngăn chặn. Bài thực hành khai thác các lỗ hổng bảo mật cung cấp cho sinh viên môi trường để sinh viên tìm hiểu về các lỗ hổng bảo mật và giúp sinh viên biết cách khai thác các lỗ hổng này thông qua các công cụ có sẵn. Đồng thời từ kiến thức lý thuyết về mật mã caesar xây dựng chương trình C++ cho thuật toán đó.

Có rất nhiều công cụ khai thác lỗ hổng hiện nay như Wireshark, Nmap, ... Tuy nhiên, trong bài thực hành này công cụ Metasploit được lựa chọn bởi vì:

Metasploit Framework là một môi trường dùng để kiểm tra, tấn công và khai thác lỗi của các service. Metasploit được xây dựng từ ngôn ngữ hướng đối tượng Perl, với những components được viết bằng C, assembler, và Python. Metasploit có thể chạy trên hầu hết các hệ điều hành: Linux, Windows, MacOS. Công cụ Metasploit Framework (MSF) chứa một tập hợp các công cụ khai thác các lỗ hổng. Công cụ này đã được xây dựng sẵn các cơ sở hạ tầng để phục vụ việc khai thác lỗ hổng và có thể sử dụng cho các nhu cầu tùy chỉnh của mình. Điều này giúp sinh viên tập trung vào việc thiết lập môi trường khai thác của mình. MSF là một trong những công cụ phổ biến nhất dành cho các chuyên gia bảo mật thực hiện các nghiên cứu về khai thác lỗ hổng thực tế. Nó chứa một công cụ khai thác và môi trường làm việc có thể mở rộng. Ngoài ra, nó được cung cấp miễn phí cho người dùng. Hơn nữa, Metasploit hỗ trợ nhiều giao diện với người dùng:

- Console interface: dùng *msfconsole.bat*. Msfconsole interface sử dụng các dòng lệnh để cấu hình, kiểm tra nên nhanh hơn và mềm dẻo hơn.
- Web interface: dùng *msfweb.bat*, giao tiếp với người dùng qua giao diện web.
- Command line interface: dùng *msfcli.bat*

Mật mã Caesar (hay còn được gọi là Mật mã của Caesar, Mật mã chuyển vị, Mã của Caesar hay Chuyển vị Caesar) là một trong những kỹ thuật mã hóa đơn giản và phổ biến nhất. Đây là một dạng mật mã thay thế, trong đó mỗi ký tự trên văn bản thô sẽ được thay bằng một ký tự khác, có vị trí cách nó một khoảng xác định trong bảng chữ cái.

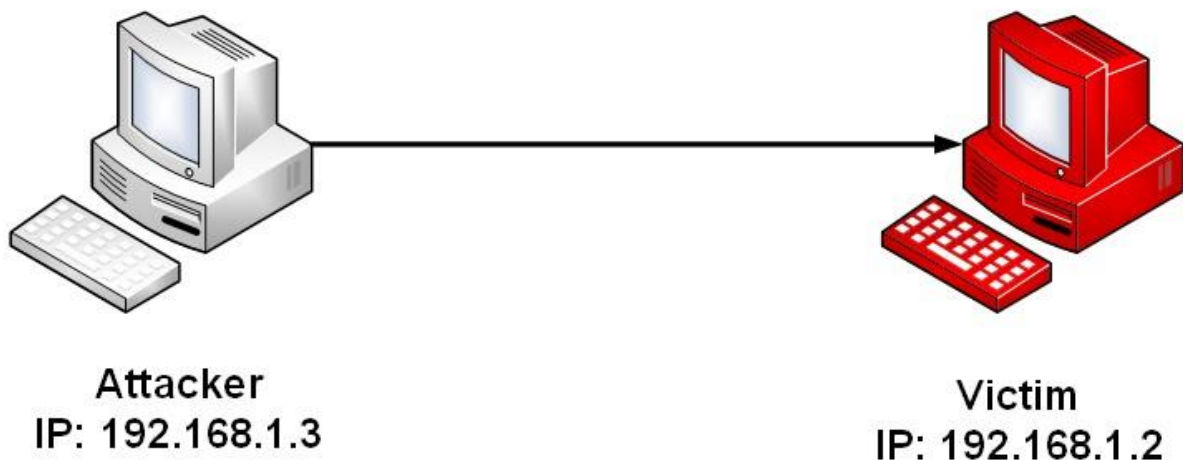
Bài thực hành này sinh viên cần tìm hiểu về lỗ hổng bảo mật dịch vụ rlogin và hiểu được cách khai thác lỗ hổng này. Không chỉ thế, sinh viên cần áp dụng kiến thức đã học về Mật mã Caesar viết chương trình C++, cách thực thi một chương trình C++ bằng dòng lệnh.

3.2.4.2 Thiết kế bài thực hành

3.2.4.2.1 Yêu cầu

- Một máy ảo có công cụ khai thác (máy cài Kali Linux cài metasploit) cho sinh viên làm nhiệm vụ tấn công (attacker)
- Một máy ảo có mở sẵn các lỗ hổng rlogin cần thiết làm máy nạn nhân (victim),
- Thiết lập hệ thống mạng sao cho máy tấn công và máy nạn nhân cùng một mạng LAN.
- Cấu hình docker gồm có:
 - Attacker: lưu cấu hình cho máy attacker
 - Config: lưu cấu hình hoạt động của hệ thống
 - Dockerfiles: mô tả các gói dữ liệu (packages), các tham số cần có cho docker của attacker và victim
 - Docs: lưu các mô tả cho bài thực hành
 - Instr_config: lưu cấu hình cho phần nhận kết quả và chấm điểm
 - Victim: lưu cấu hình cho máy victim

3.2.4.2.2 Sơ đồ thiết kế bài thực hành



Hình 3.5: Sơ đồ thiết kế bài thực hành truy tìm mật mã caesar

3.2.4.2.3 Kịch bản

Hệ thống gồm có 1 máy tấn công và 1 máy nạn nhân có mở các cổng cần thiết. Bài thực hành sử dụng hai máy ảo Linux: Một là máy Kali Linux có cài đặt MSF (máy attacker có địa chỉ IP: 192.168.1.3); và cái còn lại là máy Linux dễ bị tấn công (máy

Victim có địa chỉ IP: 192.168.1.2). Sinh viên sẽ sử dụng MSF trên máy Kali Linux để truy cập từ xa trên máy Linux dễ bị tấn công.

Sinh viên chỉ cần tải file image thực hành về chạy trên máy ảo. Sau đó chạy bài thực hành theo hướng dẫn. Trong bài thực hành này, 1 container mang tên “attacker” đóng vai trò máy tấn công và 1 container mang tên “victim” đóng vai trò máy nạn nhân. Trong đó ta thấy rằng máy attacker được cấu hình IP là 192.168.1.3, máy victim được cấu hình IP là 192.168.1.2 đồng thời một mạng LAN giữa 2 máy có subnet: 192.168.1.0/24 và external gateway: 192.168.1.1 (Hình 3.5).

- Trong máy của Victim cần thiết kế ra một file có chứa một chuỗi đã được bị mã hóa bởi mật mã Caesar.
- Trong máy của Attacker cần thiết kế ra một file zip đã bị khóa. Bên trong chứa thông điệp cần tìm đồng thời cũng là đoạn mã hóa cá nhân được tạo ra sau mỗi lần khởi động bài thực hành để giúp cá nhân hóa bài thực hành của sinh viên.
- Các nhiệm vụ cần thực hiện: 1. Sử dụng câu lệnh “ping” để kiểm tra kết nối từ máy attacker đến máy Victim. Kết quả cần đạt được “ping” thực hiện thành công, có phản hồi từ máy Victim.
 2. Sử dụng công cụ “nmap” để quét các dịch vụ có thể tấn công. Kết quả cần đạt được tìm ra các cổng có thể tấn công vào máy Victim nếu có (trong bài lab mặc định các cổng dịch vụ đều mở để có thể tấn công).
 3. Khai thác dịch vụ cấu hình rlogin (cổng 513) để truy nhập từ xa đến máy của Victim (với đặc quyền root). Kết quả cần đạt được truy cập thành công đến máy Victim với quyền root và mở được file trên máy Victim và thu được chuỗi ký tự.
 4. Viết chương trình C++ dùng thuật toán mô phỏng lại thuật toán mật mã caesar để giải mã chuỗi ký tự - đó là mật khẩu để giải nén file zip. Kết quả cần đạt được là giải nén thành công được file zip.
- Sau khi sinh viên hoàn thành bài thực hành, hệ thống cần tự động lưu lại kết quả vào 1 file và tự động đóng gói lại để cho sinh viên chỉ cần gửi file đóng gói cho giảng viên.

3.3 Chi tiết xây dựng bài thực hành - honeypot

Thực hiện xây dựng bài thực hành mới bằng GUI và tạo bài thực hành với tên honeypot (như phần [3.2.1 Thử nghiệm bài thực hành mới](#)). Sau đó sẽ thêm hai vùng chứa (container) (containers) với image cơ sở là network2 (các image được mô tả ở [2.1 Tập Dockerfile](#)). Hai vùng chứa (container) lần lượt là server và attacker.

Tiếp đó, tạo mới một mạng đặt tên là hnetwork và xác định dải mạng là 192.168.2.0/24, đồng thời cũng định danh cho địa chỉ cổng ngoài (external gateway) là 192.168.2.100. Thử khởi tạo, chạy (Run / Build and Run) bài thực hành để xác nhận việc thiết lập là đúng. Khi terminal của hai vùng chứa (container) được khởi tạo, sử dụng lệnh ifconfig và ping giữa các máy để chắc chắn việc kết nối thành công.

Tiến hành cài đặt thiết lập bài thực hành honeypot theo từng bước kịch bản được dự tính. Đặc biệt lưu ý trong quá trình thiết kế, khi bị thiếu thư viện (packages) sẽ cần ghi chú lại để thêm các thư viện cần thiết vào file Dockerfile [2.1 Tập Dockerfile](#). Những thư viện cần được cài đặt cho bài thực hành này là: git, zlib1g, zlib1g-dev, libevent-dev, libdumbnet-dev, libpcap-dev, libpcr3-dev, libedit-dev, bison, flex, libtool, automake, gcc. Trong Dockerfile của vùng chứa (container) cần phải thêm lệnh, trong trường hợp này là server tại phần ghi chú “put package installation here” (thêm các gói dữ liệu thư viện ở đây):

```
RUN apt-get update && apt-get install git zlib1g zlib1g-dev libevent-dev libdumbnet-dev libpcap-dev libpcr3-dev libedit-dev bison flex libtool automake gcc -y
```

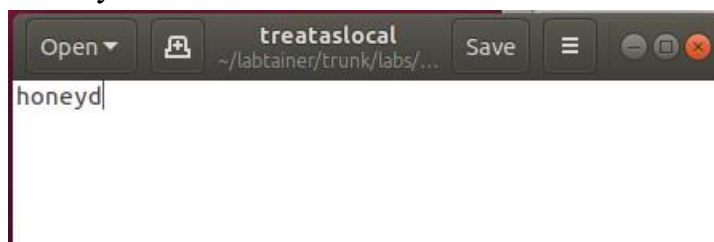
A screenshot of a text editor window titled "Dockerfile.honeypot.server.student" with the path "~/labtainer/trunk/labs/honeypot/dockerfiles". The editor contains a Dockerfile with the following content:

```
ENV APT_SOURCE $apt_source
RUN /usr/bin/apt-source.sh
#
# put package installation here, e.g.,
# RUN apt-get update && apt-get install -y --no-install-recommends somepackage
|
RUN apt-get update && apt-get install git zlib1g zlib1g-dev libevent-dev
libdumbnet-dev libpcap-dev libpcr3-dev libedit-dev bison flex libtool autotools-
dev automake gcc -y
#
```

The status bar at the bottom indicates "Plain Text", "Tab Width: 8", "Ln 40, Col 1", and "INS".

Hình 3.6: Thêm gói thư viện ở Dockerfile của vùng chứa (container) server

Để thu thập dữ liệu nhập vào và ra (stdout, stdin) của chương trình hay tiện ích thì cần khai báo chương trình, tiện ích đó vào tệp treataslocal [3.4 Đánh giá kết quả tự động](#). Trong bài này, chúng ta cần khai báo chương trình honeyd trên máy server như Hình 3.7 và tương tự nmap trên máy attacker.

A screenshot of a text editor window titled "treataslocal" with the path "~/labtainer/trunk/labs/...". The editor contains the text "honeyd".

Hình 3.7: Tệp treataslocal của server

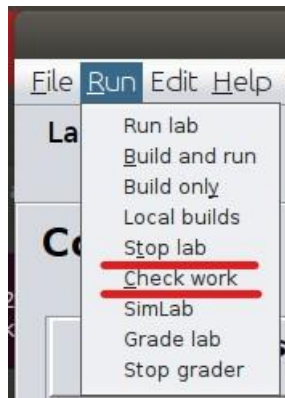
Xác định kết quả mà sinh viên cần đạt được:

	Result Tag	Container	File	Field Type	Field ID
1	ip_server_nmap	attacker	nmap.stdin	CONTAINS	192.168.2.0/24
2	honeyd_nghe1	server	honeyd.stdout	CONTAINS	192.168.2.50
3	honeyd_nghe2	server	honeyd.stdout	CONTAINS	192.168.2.51
4	honeyd_nghe3	server	honeyd.stdout	CONTAINS	192.168.2.52

Một kết quả cần đạt được thể hiện trong bảng:

- ip_server_nmap: sinh viên thực hiện quét nmap trên dải mạng 192.168.2.0/24
- honeyd_nghe1, 2, 3: honeyd thu thập được dữ liệu địa chỉ honeyd khách khi bị tấn công.

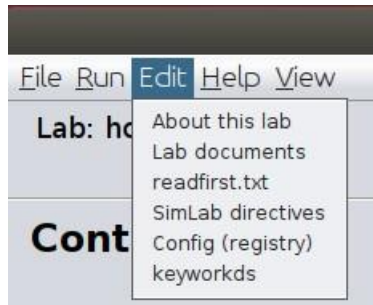
Chúng ta cần chạy, làm lại bài thực hành và sử dụng lệnh checkwork (Hình 3.8) để kiểm chứng những kết quả dự kiến thu thập theo đúng ý định của mình.



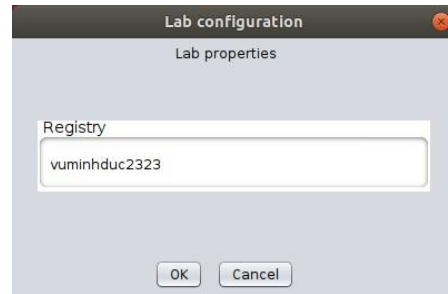
Hình 3.8: Checkwork trong labedit (GUI)

Sau khi đã thiết kế xong bài thực hành thì sẽ tiến hành công khai bài thực hành (như đề cập ở [3.2.4. Công khai bài thực hành](#)) của mình để sinh viên có thể tiếp cận, tải về, dễ dàng sử dụng:

5. Xác định Registry (tên DockerHub để tải ảnh (image) lên đó) bằng cách chọn Edit / Config (registry). (Cho rằng đã có tài khoản <https://hub.docker.com/>)

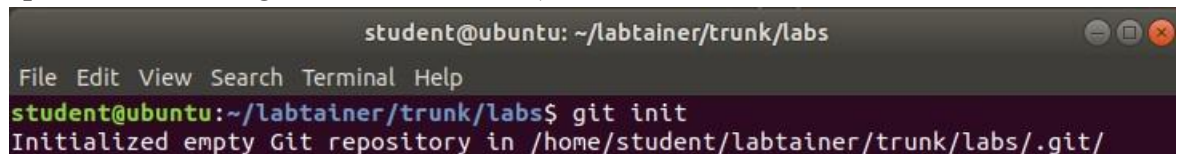


Hình 3.9: Edit / Config (registry)



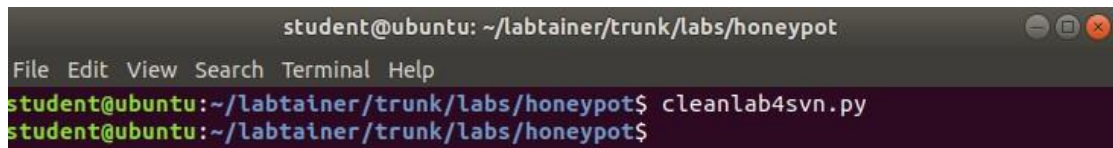
Hình 3.10: điền tên DockerHub vào ô Registry

- Chuyển tới thư mục trunk/lab khởi tạo git (chỉ khởi tạo 1 lần duy nhất, không lặp lại mỗi lần công khai bài thực hành)



Hình 3.11: git init

- Tránh cho file ảnh (image) có dung lượng lớn bị gây ra bởi những fille tạm, không thiết yếu, dùng chương trình cleanlab4svn.py để làm điều đó ở đường dẫn thư mục bài thực hành :



Hình 3.12: cleanlab4svn.py

- Sau đó trong đường dẫn cha của honeypot thêm bài thực hành vào git:

```
git add honeypot
git commit honeypot -m "Adding an IModule"
```

Với lần đầu khởi tạo git thì sẽ cần cấu hình email và tên của bạn:



Hình 3.13: Cấu hình git email và tên

- Vào thư mục trunk/distrib/ và sử dụng lệnh `./publish.py -d -l honeypot` để đẩy image lên DockerHub

```

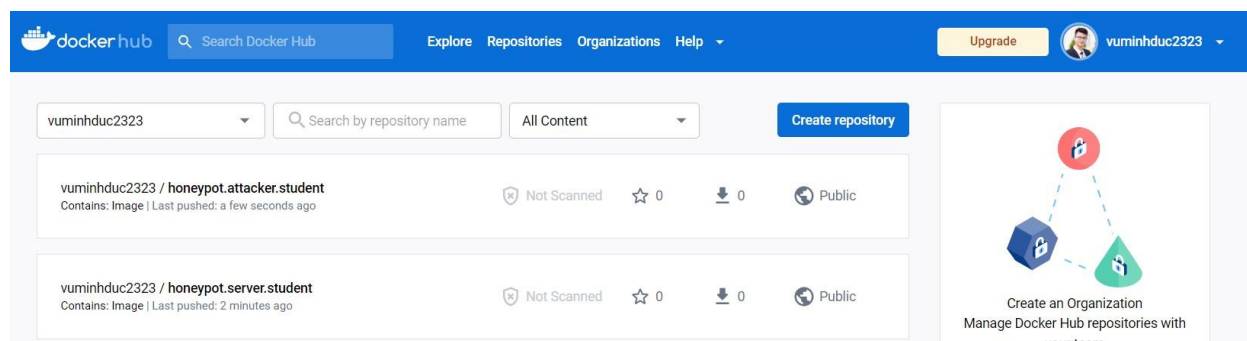
student@ubuntu:~/labtainer/trunk$ cd distrib/
student@ubuntu:~/labtainer/trunk/distrib$ ./publish.py -d -l honeypot
adding [nmaplab]
adding [httplab]
adding [liveforensics]
adding [bind-shell]
adding [tlab]
adding [metasploitable-test]
adding [kali-test]
adding [my-remote-dns]

```

Hình 3.14: ./publish.py -d -l honeypot

Với lần đầu tiên công khai bài thực hành, đẩy image lên DockerHub thì hệ thống sẽ yêu cầu nhập tài khoản, mật khẩu của DockerHub (Hình 3.16).

Sau khi hoàn tất, image của các vùng chứa (container) bao gồm attacker và server sẽ được đẩy lên <https://hub.docker.com/>



Hình 3.15: các image đã được đẩy lên DockerHub

```

student@ubuntu: ~/labtainer/trunk/distrib
File Edit View Search Terminal Help
Username: vuminhduc2323
Password:
WARNING! Your password will be stored unencrypted in /home/student/.docker/config.json.
Configure a credential helper to remove this warning. See
https://docs.docker.com/engine/reference/commandline/login/#credentials-store

Login Succeeded
Sending build context to Docker daemon 126.5kB
Step 1/4 : FROM honeypot.server.student
--> d4305e6d2d65
Step 2/4 : ARG version
--> Running in 7acdef56e1b8
Removing intermediate container 7acdef56e1b8
--> 2825c32bc13c
Step 3/4 : LABEL version=3
--> Running in 8bb6762c1f76
Removing intermediate container 8bb6762c1f76
--> 16512f18c5e1
Step 4/4 : LABEL base=labtainers/labtainer.network2.776f1cfababa
--> Running in f0e4c62beef2
Removing intermediate container f0e4c62beef2
--> 9b28ed9bb670
Successfully built 9b28ed9bb670

```

Hình 3.16: Nhập tài khoản, mật khẩu DockerHub

Các bước thực hiện bài thực hành:

1. Cài đặt Honeyd trên Server

- Tải mã nguồn: từ git <https://github.com/DataSoft/Honeyd>

- Tiến hành cài đặt honeyd:

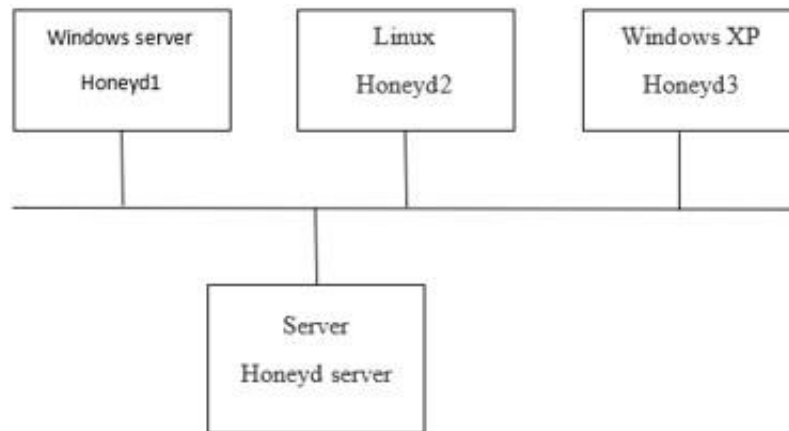
○ \$./autogen.sh ○ \$

./configure ○ \$ make ○

\$ sudo make install

2. Cấu hình honeyd

- Sơ đồ mạng: 1 máy chủ Server, cấu hình honeyd tạo ra thêm 3 máy gồm windows server, máy linux và máy windows XP (xem hình dưới):



Hình 3.17: Sơ đồ mạng

- Trong trường hợp này, máy Server đồng thời là máy chủ honeyd tạo ra 3 honeyd client 1, 2, 3.

- Thiết lập file cấu hình honeyd.conf ○ \$ nano honeyd.conf

- Cách tạo file cấu hình cho honeyd để được sơ đồ như sau:

```
create windows
```

```
set windows personality "Microsoft Windows Server
2003" set windows default tcp action reset add windows
tcp port 135 open set windows ethernet
"11:11:11:11:11:11" bind 192.168.2.50 windows

create linux set linux personality
"Linux 2.4.7" set linux default tcp
action reset add linux tcp port 21 open
set linux ethernet
"11:11:11:11:11:22" bind
192.168.2.51 linux

create xp set xp personality "Microsoft Windows XP
Professional" set xp default tcp action reset add xp tcp
port 80 open set xp ethernet "22:22:22:22:22:22" bind
192.168.2.52 xp
```

3. Sử dụng honeyd phát hiện tấn công mạng

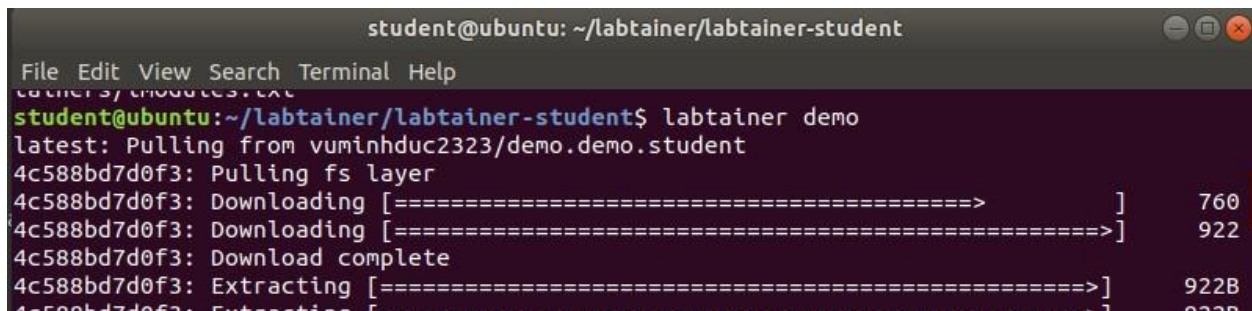
- Honeyd yêu cầu phải được chạy ở quyền root. Vì thế Ở bên máy Server sử dụng lệnh:

```
$ sudo honeyd -d -f ~/honeyd.conf
```

- Dùng máy Attacker để ra quét dải mạng 192.168.2.0/24. Khi này bên máy Server phát hiện mạng đang bị rà quét. Và nhận biết được máy nào tấn công sau một khoảng thời gian.

3.4 Cách thức sử dụng bài thực hành

1. Sinh viên download file imodule.tar từ URL được giảng viên cung cấp
2. Giải nén file imodule.tar và chuyển bài thực hành tới thư mục labs (labtainer/trunk/labs)
3. Mở terminal tại thư mục labtainer-student: **labtainer [tên bài lab]** - để bắt đầu bài thực hành



```
student@ubuntu: ~/labtainer/labtainer-student
File Edit View Search Terminal Help
student@ubuntu:~/labtainer/labtainer-student$ labtainer demo
latest: Pulling from vuminhduc2323/demo.demo.student
4c588bd7d0f3: Pulling fs layer
4c588bd7d0f3: Downloading [=====>] 760
4c588bd7d0f3: Downloading [=====>] 922
4c588bd7d0f3: Download complete
4c588bd7d0f3: Extracting [=====>] 922B
4c588bd7d0f3: Extracting [=====>] 922B
```

Hình 3.18: Bài thực hành demo được lấy images từ DockerHub

Chương này đã trình bày sự phân tích và thiết các bài thực hành Honeypot giúp phát hiện và đánh lừa kẻ tấn công. Bài thực hành Tcpdump trang bị cho sinh viên kiến thức về công cụ giám sát thu thập thông tin và phân tích gói tin, Arpspoof & Scapy hướng dẫn sử dụng thư viện scapy để tạo chương trình thu thập giám sát lưu lượng mạng, Truy tìm mật mã giúp sinh viên ứng dụng thuật toán mật mã Caesar và tấn công lỗ hổng rlogin.

TÀI LIỆU THAM KHẢO

- [1] mftthom, 14 January 2018. [Trực tuyến]. Available: <https://youtu.be/JDV6jGF3Szw>.
- [2] T. B. Jelena Mirkovic, “Teaching Cybersecurity with DeterLab,” *IEEE Security & Privacy*, tập 10, số 1, pp. 73-76, 2012.
- [3] V. N. M. E. L. J. M. B. H. Richard Weiss, “Hands-on cybersecurity exercises and the rave virtual environment,” *SIGCSE '13: Proceeding of the 44th ACM technical symposium on Computer science education*, p. 759, 2013.
- [4] M. F. T. J. K. Cynthia E. Irvine, “Labtainers: A Framework for Parameterized Cybersecurity Labs Using Containers,” 2017.
- [5] M. F. T. M. M. K. Cynthia E. Irvine, “Labtainers: A Docker-based Framework for Cybersecurity Labs,” 2017.
- [6] [Trực tuyến]. Available: <https://nps.edu/web/c3o/labtainers>.
- [7] A. K. Alsalamah, “Applying Virtualization and Containerization Techniques in Cybersecurity Education,” 2018.
- [8] “Dockerfile reference,” [Trực tuyến]. Available: <https://docs.docker.com/engine/reference/builder/>.
- [9] [Trực tuyến]. Available: <https://github.com/Vuduchihi/labtainerlabs>.
- [10] Triển khai hệ thống Honeypot kết hợp IDS, HVCNBCVT.
- [11] [Trực tuyến]. Available: <https://github.com/DataSoft/Honeyd>.
- [12] NSM_Bài thực hành 1: công cụ thu thập dữ liệu mạng: tcpdump, wireshark, silk, HVCNBCVT.