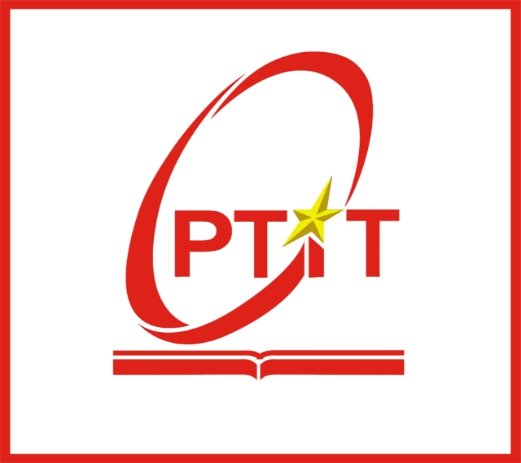


**HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG**

**KHOA AN TOÀN THÔNG TIN**



**ptit-yara**

**yarGen**

Giảng viên: Đinh Trường Duy

Lớp : D20CQAT01-B

## Mục đích

* Giúp sinh viên hiểu được cấu trúc luật của Yara từ đó biết cách viết một tập luật đơn giản để phát hiện file mã độc trong thư mục.
* Giới thiệu đến sinh viên công cụ yarGen – công cụ giúp sinh viên tạo file rule tự động đối với một hoặc nhiều mã độc.

## Yêu cầu với sinh viên

* Có kiến thức cơ bản về hệ điều hành Linux
* Tìm hiểu về cấu trúc luật của Yara

## Nội dung thực hành

* Khởi động bài lab:

Tại terminal, sinh viên gõ:

***labtainer ptit-yara***

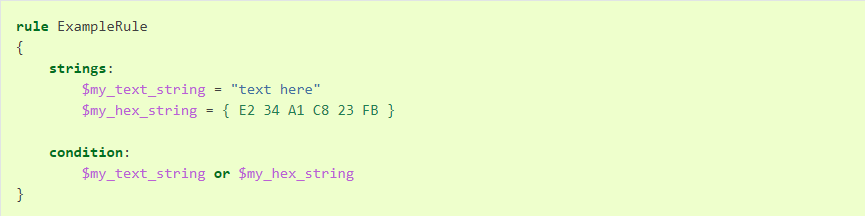
*Chú ý: Sinh viên sử dụng mã sinh viên của mình để nhập email thông tin người thực hiện lab khi có yêu cầu để sử dụng khi chấm điểm*

Sau khi ấn enter, 2 terminal của máy ảo có tên ***ptit-yara@client*** và ***ptit-yara@server*** sẽ xuất hiện.

Sinh viên gõ câu lệnh ***ls*** kiểm tra các file/folder đang có sẵn trên máy ảo, bao gồm:

* Trên ***ptit-yara@client (IP: 172.20.0.2):*** file example.txt, folder yarGen-0.23.4, folder ghidra\_11.0.3\_PUBLIC\_20240410
* Trên ***ptit-yara@server (IP: 172.20.0.3):*** file filetoview.txt

Sinh viên thực hiện tìm hiểu cấu trúc của 1 luật yara như sau:



Hình 1‑1: Cấu trúc 1 luật yara cơ bản

Một luật cơ bản của yara bắt đầu bằng từ khóa “rule” theo sau là tên của luật. Mỗi luật thường gồm phần định nghĩa chuỗi (strings) và phần điều kiện (condition). Phần định nghĩa chuỗi là nơi xác định các chuỗi sẽ là một phần của quy tắc. Mỗi chuỗi có một mã định danh bao gồm ký tự $ theo sau là một chuỗi các ký tự chữ và số và dấu gạch dưới, những mã định danh này có thể được sử dụng trong phần điều kiện để chỉ chuỗi tương ứng. Chuỗi có thể được xác định ở dạng văn bản hoặc thập lục phân.

Chuỗi văn bản được đặt trong dấu ngoặc kép giống như trong ngôn ngữ C. Chuỗi hex được bao quanh bởi dấu ngoặc nhọn và chúng được tạo thành bởi một chuỗi các số thập lục phân có thể xuất hiện liền kề hoặc phân tách bằng dấu cách. Số thập phân không được phép trong chuỗi hex.

Phần điều kiện là nơi chứa logic của quy tắc. Phần này phải chứa một biểu thức boolean cho biết trong trường hợp nào một tệp hoặc quy trình có đáp ứng quy tắc hay không. Nói chung, điều kiện sẽ tham chiếu đến các chuỗi được xác định trước đó bằng cách sử dụng mã định danh của chúng. Trong ngữ cảnh này, mã định danh chuỗi hoạt động như một biến boolean đánh giá là đúng nếu chuỗi được tìm thấy trong tệp hoặc bộ nhớ xử lý hoặc sai nếu ngược lại.

Ngoài ra, sinh viên có thể tìm hiểu thêm về Yara tại đường dẫn sau: <https://yara.readthedocs.io/en/latest/gettingstarted.html>

Sau khi đã hiểu về cấu trúc, các quy tắc viết 1 luật yara, sinh viên sẽ thực hành viết 1 luật cơ bản để phát hiện file strings.txt cho sẵn. Sinh viên có thể đọc trực tiếp file **example.txt** để xem file này có chứa những loại chuỗi nào hoặc sử dụng công cụ strings như sau:

***strings example.txt***

Sinh viên tạo 1 file luật có tên **example.yar**, sau đó viết 1 luật để phát hiện chuỗi có trong **example.txt**. Sinh viên thực hiện câu lệnh sau để chạy yara:

*yara -r <đường dẫn tới file yara> <đường dẫn tới folder cần quét>*

Ví dụ trong bài này:

***yara -r example.yar ./***

Để đọc thêm các tùy chọn có trong yara, sinh viên có thể gõ *yara -h*

Tiếp theo, sinh viên sử dụng gợi ý có trong file **example.txt**.

**Câu hỏi:** Gợi ý trong file **example.txt** là gì?

Sau khi đã nắm được gợi ý, sinh viên sẽ dùng nó để rà quét, tìm kiếm các file khác có chứa gợi ý này bằng cách viết luật yara. Có 2 file mà sinh viên cần tìm được

Sử dụng gợi ý trong file **example.txt**, sinh viên sẽ viết tiếp luật mới vào file **example.yar** đã tạo, sau đó sử dụng câu lệnh: ***yara -r /home/ptit-yara/example.yar ./*** để rà quét các file trong thư mục hiện tại. Sau khi đã tìm ra 2 file trong đó có 1 file là file thực thi. Sinh viên không cần chạy file đó, mà cần phân tích xem trong file này có gợi ý gì (có thể dùng công cụ strings). File còn lại cũng chứa gợi ý.

**Câu hỏi:** Tên của 2 file đó là gì? Hãy chụp minh chứng của bạn.

**Câu hỏi:** Gợi ý của từng file là gì? Hãy chụp minh chứng của bạn.

Dựa vào 2 gợi ý vừa tìm được, sinh viên sẽ tìm thấy 1 file thực thi có tên **crackme0x03** và 1 file zip có tên **malwares.zip**. Sinh viên thử mở file zip đó bằng công cụ unzip sẽ được yêu cầu nhập mật khẩu. Sinh viên sẽ cần phân tích file **crackme0x03** để tìm ra mật khẩu của file zip đó sử dụng công cụ Ghidra đã được cài sẵn.

Sinh viên vào thư mục **ghidra\_11.0.3\_PUBLIC,** thực hiện gõ câu lệnh:

***./ghidraRun***

để chạy chương trình Ghidra. Sinh viên thực hiện tạo project mới bằng cách ấn vào File, chọn New Project rồi đặt tên cho project của mình. Sau đó tiếp tục ấn vào File, chọn Import File rồi thực hiện import file **crackme0x03** vào. Cuối cùng, sinh viên click đúp chuột vào file crackme0x03 trong giao diện Ghidra để phần mềm thực hiện phân tích.

**Câu hỏi:** File crackme0x03 là loại file gì?

Trong lần đầu tiên tạo project, có thể sinh viên sẽ gặp phải lỗi không cho phép import file **crackme0x03**.

**Câu hỏi:** Nguyên nhân là gì? Cách để import được file **crackme0x03** vào Ghidra?

Sinh viên thực hiện phân tích mã nguồn đã dịch ngược bằng công cụ Ghidra, tìm đến hàm main để đọc mã nguồn và trả lời câu hỏi sau:

**Câu hỏi:** Hãy phân tích chi tiết chức năng của hàm “\_test”? Mật khẩu bạn tìm ra là gì? Hãy chụp minh chứng của bạn.

Sau khi có được mật khẩu, sinh viên sẽ thực hiện giải nén file **malwares.zip** bằng câu lệnh ***sudo unzip malwares.zip***. Lúc này, sinh viên sẽ thấy 1 folder có tên malwares chứa 1 vài file exe có tên ngẫu nhiên. Nhiệm vụ của sinh viên là viết luật để tìm ra mã độc theo một vài mô tả sau:

* Đây là một mã độc giả dạng là một file excel
* Mã độc này sử dụng trình đóng gói UPX
* Khi người dùng click chạy mã độc, nó sẽ thực hiện một loạt các hành vi bao gồm: hiện thông báo giả mạo tới người dùng, kết nối ra Internet để tải thêm phần mềm khác về máy người dùng, thực hiện persistent hệ thống, thêm hoặc sửa đổi các Registry Key,...

Dựa vào các mô tả này, sinh viên hãy tạo 1 file luật mới có tên **malware.yar** rồi viết các luật để phát hiện file mã độc mục tiêu có trong folder malwares. ***Viết tối thiểu 1 luật và có chứa nhiều chuỗi, kiểu chuỗi càng tốt*** (tham khảo thêm tại đường dẫn phía trên).Ví dụ: 1 chuỗi phát hiện mã độc kết nối đến url, 1 chuỗi phát hiện mã độc sử dụng trình đóng gói,...Sau đó, sử dụng câu lệnh:

***yara -r malware.yar -m /home/malwares***

Sau khi phát hiện ra mã độc mục tiêu, sinh viên sẽ tạo 1 folder có tên **madoc** rồi copy mã độc vừa tìm thấy vào folder đó.

Tiếp theo, bài lab này sẽ giới thiệu đến sinh viên công cụ yarGen dùng để tạo luật tự động cho mã độc bằng cách tham chiếu đến CSDL của công cụ. Sinh viên truy cập vào thư mục **yarGen-0.23.4** để tìm hiểu các file của công cụ này. Sinh viên sử dụng công cụ để sinh 1 file luật cho mã độc vừa tìm được. Sử dụng câu lệnh:

***python3 yarGen.py -m /home/madoc***

để tạo luật cho file mã độc vừa tìm được. Một file luật mới có tên **yargen\_rules.yar** được tạo (sinh viên có thể tự đặt tên cho file luật bằng tùy chọn -o của công cụ này). Tiến hành đọc file luật, sinh viên có thể thấy có 20 chuỗi được sinh ra và sinh viên có thể tùy chỉnh số lượng chuỗi mặc định trong mã nguồn của file yarGen.py.

Sau khi đã tạo xong file luật, sinh viên thực hiện quét folder **malwares** trước đó để kiểm tra xem có phát hiện ra mã độc mục tiêu hay không.

Trước khi hoàn thành bài lab, sinh viên thực hiện ssh tới server. Sinh viên nên kiểm tra trạng thái của dịch vụ ssh, rồi start hoặc restart dịch vụ ở server trước khi ssh từ client, ví dụ: ***sudo systemctl status ssh, sudo systemctl start/restart ssh,*** sử dụng mật khẩu chính là tên của mã độc mục tiêu đã tìm được để đăng nhập, sau đó thực hiện in ra màn hình nội dung của file **filetoview.txt.**

* Kết thúc bài lab:

Tại terminal, sinh viên gõ:

***stoplab ptit-yara***

Khi bài lab kết thúc, một tệp zip lưu kết quả được tạo và lưu vào một vị trí được hiển thị bên dưới stoplab.

* Khởi động lại bài lab:

Trong quá trình làm bài sinh viên cần thực hiện lại bài lab, dùng câu lệnh:

***labtainer –r ptit-yara***