



l4w.io



JULY 20, 2017 / MANHLUAT / Z

BẮT ĐẦU HỌC AN TOÀN THÔNG TIN NHƯ THẾ NÀO?

Bạn thắc mắc rằng học an toàn thông tin cần gì ? học những gì và học như thế nào ?

Hôm nay mình muốn viết một cái note này để dành cho những bạn mới bắt đầu nghiên cứu về bảo mật máy tính. Mình sẽ không đi quá chi tiết vào từng mảng cũng như khái niệm, công cụ, mà đơn giản chỉ là một cái note, một cái roadmap cho bạn để dễ hình dung mình nên bắt đầu từ gì và bắt đầu như thế nào.

Về cơ bản bảo mật máy tính hiện tại chia thành các mảng sau đây:

- Web Application Security (Ứng dụng web)
- Reverse Engineering (Dịch ngược)
- Software Exploitation (Khai thác lỗi phần mềm)
- Cryptography (Mật mã học)

- Forensics (Điều tra chứng cứ số)
- Networking Security (An toàn mạng)

Hãy học lập trình.

Bạn sẽ dẫn đo liệu rằng mình nên bắt đầu từ mảng nào, mình chỉ có lời khuyên bạn nên bắt đầu từ **Coding – Lập trình**. Đây là điều mình luôn khuyên tất cả các bạn trẻ dù đang theo học bất kì lĩnh vực nào trong thế giới máy tính. Vì đơn giản, một khi bạn có tư duy logic của một lập trình viên, mọi thứ sẽ trở nên dễ dàng hơn. Nên ghi nhớ, máy tính được cấu tạo từ 2 bit logic 1 và 0, đúng và sai, true and false. Cho nên hiển nhiên 100%, logic là thứ bạn cần để đi tiếp con đường này. Ít nhất hãy trang bị cho mình những kiến thức cơ bản về: Hệ điều hành (Operating System), Cấu trúc dữ liệu (structures), cũng như có khả năng lập trình được từ 03 ngôn ngữ trở lên.

Tìm ra cách học và theo đuổi đam mê cho riêng mình.

Về phần mình, mình đã bắt đầu với việc lập trình PHP, việc tạo ra những công cụ nhỏ lẻ mà không thể tìm được trên mạng là thứ mình thích. Bạn cũng nên thế, hãy tìm ra sở thích của mình, và dùng công cụ để tạo nên nó, nó sẽ là động lực để bạn có hứng thú tìm hiểu sâu hơn mà không ngán ngẫm. Mình chỉ đơn giản là một thằng nhóc theo đuổi thứ mình thích lâu nhất và không bỏ cuộc.

Hãy đừng so sánh ngôn ngữ nào hay hơn, mạnh hơn ngôn ngữ nào. Suy cho cùng nó chỉ là một công cụ để cho con người thực thi những việc mình muốn một cách nhanh và tự động. Mình đã từng nghe câu như này:

Máy tính, nhanh và chính xác, nhưng lại kém thông minh. Con người cực kì thông minh, nhưng lại quá chậm và sai sót.

Kết hợp cả hai thứ trên, thì mới là cao nhân. Trí tuệ nhân tạo được như ngày hôm nay thì cũng là do con người setup cho, chứ nó chẳng tự dựng thông minh lên.

Okay, quay trở lại việc bắt đầu với các mảng chính kể trên.

Bạn nên chọn mảng nào ?

Đây có lẽ là điều băn khoăn nhất với những người bắt đầu. Về phần mình, là người đã tham gia vào nó một thời gian đủ lâu, quan sát từng mảng và cả những người chuyên về lĩnh vực đó. Mình nhận thấy, mảng nào cũng hay, mảng nào cũng là cả một nghệ thuật và đòi hỏi quá trình học hỏi không ngừng.

Quan trọng là: bạn thích gì, bạn cảm thấy mình hợp, bạn cảm thấy vui khi làm cái nào nhất ?

Như đã kể trên, mình bắt đầu với việc lập trình web trước, vì hồi nhỏ mình thấy anh trông net nhà kể bên, gõ gõ notepad html mà ra một trang web. Nên thấy nó thú vị, và bản thân mình cũng thích làm một website thật bắt mắt, đó là điều mình thích thú và cảm thấy vui vẻ khi làm. Cho nên việc bắt đầu với mảng Web Application Security với bản thân mình tại thời điểm đó là hợp nhất.

Nếu bạn suy nghĩ về một thứ gì đó trong khoảng một thời gian dài, và việc suy nghĩ về nó chiếm hơn 1/4 ngày của bạn, đó chính là thứ bạn thích, theo mình là vậy.

Còn nếu bạn thật sự chưa biết gì về các mảng trên thì hãy... thử hết, nhưng hãy thử thật nhanh, thử chơi CTF (Capture The Flag) mảng đó xem, xem mình thích không, không thích thì thử mảng khác. Sau đó ngồi ngẫm xem, mình thấy vui nhất khi làm cái nào. Vậy thôi, mọi thứ trên đời cái gì cũng có cái hay/dở của nó, quan trọng là **bạn thích cái gì**. Vì bạn phải vui, thì bạn mới bỏ công sức, tâm huyết mình vào với nó, lúc đó mới có thể phát triển xa được, nếu không mọi thứ nó sẽ trở nên hỗn độn, không gì ra gì, chỉ tốn phí thời gian mình thôi.

Dù bạn có thiên bẩm, mọi thứ vẫn phải tuân theo quy luật **10,000 hours**. Tức là bạn phải luyện tập 10,000 tiếng đồng hồ mới có thể nên cơm cháo được.

Lí do tại sao, mình nghĩ con người nên phát triển đam mê từ bé, vì lúc đó bạn sẽ không ngại ngại gì nhiều, không phải đắn đo việc xung quanh, cơm áo gạo tiền,...

Hãy tập thói quen hỏi “Tại sao?”


Mọi thứ bạn làm, mọi công cụ bạn chạy, hãy tự hỏi bản thân “Tại sao?”, nguyên lý nó như thế nào, tại sao nó lại bị lỗi, tại sao phải khai thác như thế, tại sao và tại sao.....

Cũng thay vì bạn tìm hiểu khái niệm về một lỗi bảo mật nào đó, đừng!. Đây là cách học mình cho là hay nhất: Hãy tạo ra chính lỗi đó, hãy là người lập trình mắc phải những vấn đề tương tự, rồi bạn sẽ hiểu được nó một cách rõ ràng, hệ thống, và logic nhất. Đó là cách học theo mình là đúng đắn cho một chặng đường dài mà bạn phải theo đuổi.

Tin mình, hãy học bằng một cách khác đi. Đừng chỉ chạy công cụ nữa 😞

Bắt đầu việc phân tích những điểm hay ho trong từng mảng.


Web Application Security (Ứng dụng web)

- **Dễ tiếp cận**, vì kiến thức nó không quá sâu. Bạn không cần phải rành mạch về Hệ điều hành, Cấu trúc dữ liệu gì cả, mà vẫn có thể làm nó. Chỉ cần biết **lập trình**. Vì khi bạn biết lập trình, bạn sẽ chính là người mắc vào những lỗi đó, cách học nhanh nhất và dai nhất là tự vấp và sửa sai lầm của chính mình.
- **Target rất nhiều**, nói tới đây không phải cổ súy. Nhưng rõ ràng thế giới www (World-Wide-Web) ngày này cực kì to lớn, ai cũng có thể lập cho mình một website. Chính vì thế mà “bao cát” luyện tập cho mọi người cũng rất nhiều. Nhưng hãy nhớ, xây dựng mới khó, đập phá rất dễ. Hãy là những người trẻ có suy nghĩ và ý thức.
- **Công cụ/tài liệu rất rất nhiều**, vì đây là mảng nhiều người tiếp cận nhất, sinh ra công cụ mà họ làm ra cũng tỉ lệ thuận. Hàng ngàn công cụ được tạo ra chỉ cho việc khai thác một lỗi duy nhất (điển hình: SQL Injection). Cho nên, việc script-kiddie, những người không hề biết tí nguyên lý về nó vẫn có thể khai thác thành công. Đây là điểm chí mạng, mình sẽ đề cập sau.
- Các kĩ thuật / nguyên lý hoạt động của www ngày nay bạn cần biết, nó là rất nhiều, và thay đổi theo từng ngày, nhưng cũng chỉ xoay quanh những kiến thức nền tảng, nếu bạn nắm bắt tốt, việc tiếp thu một kĩ thuật hay nguyên lý mới cũng không quá khó. Bạn cũng nên cập nhật nguyên lý của các biện pháp bảo vệ (mitigation) trên trình duyệt hiện đại như: Chrome/Edge/Firefox/...
 - Các lỗi thường gặp: Injection (SQL/Command/Template/...), Cross-Site Scripting (XSS), Cross-Site Request Forgery (CSRF), Server Side Request Forgery (SSRF), Insecure Direct Object References (IDOR), XML External Entity (XXE),...
 - HTTPS
 - Content-Security-Policy (CSP)
 - Cross-Origin Resource sharing (CORs)
 - X-Frame
-  **The Tangled Web: A Guide to Securing Modern Web Applications**
(Cuốn này bao xô gần hết www này, và được viết từ một hacker huyền thoại 😊)
- Ngôn ngữ nên thành thạo: HTML / Javascript / Java / PHP / ASPX / Ruby / Python
- Tips: Một cách hay để cập nhật, thấy được cái hay và sáng tạo của các web ninja khác là theo dõi các report bug bounty trên hackerone / bugcrowd hoặc writeup của họ. Hãy tìm hiểu cách thức họ suy nghĩ, cách suy luận để ra được kết quả.

P/S: Trong phạm vi bài viết này, thay vì đưa chục cuốn sách cho bạn, mình sẽ đưa cho bạn ít nhất có thể, mình nghĩ các bạn nên cố gắng đọc hiểu kiên trì một cuốn nào đó, thay vì tải một đồng vế, mỗi thứ một chút, chẳng được gì (mình đã từng như vậy nên đây là lời khuyên). Một khi bạn biết mình đã thấu hiểu được lượng kiến thức nhất định, tự khắc lúc đó bạn sẽ biết mình cần làm, cần đọc cái gì tiếp theo thôi.

☹ Reverse Engineering (Dịch ngược)

- **Cần một lượng kiến thức nhất định**, nôm na công việc dịch ngược sẽ giúp bạn thấu hiểu về cách thức hoạt động của một chương trình, chẳng hạn bằng cách nào mà từ một file mp3, khi được tải vào một chương trình chơi nhạc lại phát ra những âm thanh, cấu tạo của tệp mp3 đó là gì ? làm thế nào chương trình biết tới đoạn nào sẽ chơi nốt nhạc gì. Thậm chí ngày nay, việc nghe nhạc bản quyền đã trở nên nghiêm túc, chẳng hạn chương trình chơi nhạc như **Spotify** đã áp dụng cả mã hóa đối với những tệp nhạc đấy, để tránh người dùng tải, lưu giữ, phát tán trái phép. Nhiều người nghĩ rằng RE (Reverse Engineering) chỉ gom gọn trong việc bẻ khóa / crack phần mềm để xài “chùa”. Có lẽ bạn đã hiểu sai, mục đích của RE thậm chí có thể là phân tích cách thức hoạt động, thuật toán đẳng cấp đẳng sau một chương trình nào đó của công ty đối thủ mà mình muốn “học hỏi”, hoặc thậm chí như ví dụ nêu trên, là việc phân tích cách giải mã một tệp nhạc của Spotify. Vậy cho nên, kiến thức để giúp bạn RE một chương trình, nó tùy thuộc vào độ phức tạp, loại đối tượng mà bạn đang làm. Ví dụ bạn phải dịch ngược một kernel module nào đó, tất nhiên bạn phải có kiến thức bổ sung về Filesystem/Device Control IO/System Calls/... Còn về kiến thức cơ bản sẽ xoay quanh:
 - Kiến thức về mã máy (machine code, opcode).
 - Đọc hiểu hợp ngữ (assembly) và có thể viết cả một hàm (ví dụ strlen ?) nào đó bằng ASM.
 - Kiến thức về CPU/ thanh ghi (register).
 - Cách thức hoạt động chung và cơ bản của một hàm (buffer, calling convention, stack frame, call stack,...)
- Song đó, cũng tồn tại những kĩ thuật chống lại quá trình dịch ngược/debug, các kĩ thuật/ thủ thuật rất rất nhiều, và hàng ngày hai phe vẫn cố gắng tìm cách ngăn chặn phe kia. Hơi giống với trò chơi mèo đuổi chuột, cứ chạy vòng vòng, ít nhất là tại thời điểm hiện tại.
 - Anti-VM / Anti-sandbox, chống lại việc chạy tệp nhị phân trong một môi trường máy ảo hoặc sandbox nào đấy, người viết ra tệp nhị phân sẽ tìm cách để phát hiện ra điều đó.
 - Anti-debug, sử dụng những kĩ thuật/ thủ thuật ngăn chặn người phân tích có thể dễ dàng chạy dưới hoặc đánh vào debugger.

- Obfuscation, làm rối các đoạn mã máy/hợp ngữ, làm cho những chương trình phân tích tự động cũng khó phân tích được.
- Packing, nhồi nhét như nghĩa của nó, kĩ thuật “đóng gói” một binary vào trong một cái hộp nào đó, quấn dây, băng keo lại, một cách rối rắm, mà chỉ người chế ra cách đóng gói nó mới có thể dễ dàng mở...nhưng đôi khi bạn chỉ cần dùng rọc giấy để mở nó... :hacknào:
- Còn nhiều kĩ thuật khác, bạn có thể tìm hiểu trong cuốn sách bên dưới.
- Ngoài ra có một cách học RE hay ho là, bạn hãy lập trình nhiều chương trình, hàm khác nhau, thuật toán khác nhau, compile nó và quăng vào các chương trình dịch ngược để thấy đoạn hợp ngữ được sinh ra, nếu bạn có đang dùng IDA Hex-rays hoặc chương trình Decompiler tương tự để sinh ra mã giả C thì...tạm thời quên nó đi mà hãy tập làm quen với việc đọc hợp ngữ, mã máy, cách thức các instruction hoạt động.
- Hãy ráng học những nền tảng cơ bản thật vững chắc, vì để đi đường dài, bạn không thể nào chỉ phụ thuộc vào nó được, sẽ có những trường hợp bạn cần giải quyết bằng những kiến thức thật chắc bài, hoặc thậm chí phải tự làm ra công cụ của riêng mình dựa trên những kiến thức ấy.
- Tutorial/Blog Tiếng Việt: [kienmanowar](#) / [yeuchimse](#) / ...
-  Reversing: Secrets of Reverse Engineering

Hey yo, chưa gì được gần 1 tháng rồi, đợt rồi mình hơi ít thời gian vì đã phải chuẩn bị và tham gia DEFCON 25 CTF diễn ra thường niên tại Las Vegas. Được cho là hội thảo underground có số lượng người tham gia lớn nhất thế giới hiện tại.

Mình cũng vừa có một tuần được ở trong tiểu bang California, nơi mà người Việt hải ngoại sinh sống khá nhiều ở đây, được ghé thăm quận Cam, Phước Lộc Thọ,... May mắn cũng được đi chơi ké một gia đình bạn bè người Việt đang sinh sống ở Mỹ nên mình cũng được nghe những câu chuyện và tìm hiểu được tí xít về đời sống, sinh hoạt của người dân ở đó. Tính viết một bài nói về cảm nghĩ chuyến đi, mà không biết nên không 😊.

Riêng về cảm nghĩ bản thân, mình rất thích tiểu bang đầy nắng này ☀️, mình cũng đi được vài nước tiên tiến khác, nhưng hiện tại California đang là nơi mình muốn quay lại nhiều nhất. Thời tiết ôn hòa là lí do nhiều người muốn đến đây sống mặc dù là một trong các tiểu bang có mức sống cao nhất nước Mỹ, nắng nóng nhưng lại không khó chịu như Việt Nam và Singapore (chắc do độ ẩm ?). Biển bờ tây tuyệt đẹp, đường phố cây cối, nhà cửa được trang trí 2 bên đường cực kì hợp lý. Đặc biệt, có rất nhiều hàng quán, đồ ăn Việt Nam.

Thôi quay lại chủ đề chính.

Software Exploitation (Phần mềm)

- Đơn giản nó là tìm kiếm và khai thác lỗi bảo mật ở các phần mềm. Công đoạn được cho là quan trọng và cũng là điểm chung lớn nhất của mảng này và Web Application Security là bạn phải tìm bug. Vì phải có bug (lỗi) thì bạn mới có thể khai thác (exploit). Ở mảng này, có hai cách để tìm bug mà được sử dụng nhiều nhất:
 - **Fuzzing**, hm hôm nay, là bạn viết một chương trình tự động gọi là fuzzer để cung cấp input vào một chương trình nào đó một cách ngẫu nhiên để tìm ra lỗi. Giả sử mình có một đoạn hàm C như sau:

```
1 int main(){
2     char buffer[256];
3     scanf("%256s",buffer);
4     printf(buffer);
5 }
```

Bạn có thể sử dụng python để viết ra một fuzzer cho chương trình này để dễ dàng tìm ra được bug format string (nhưng thật ra chương trình này còn một bug nữa). Về tài liệu thì có rất nhiều, bạn có thể tham khảo video này: [Hacking Livestream #17: Basics of fuzzing by Gynvael Coldwind](#). Anh này có cả một series nói về hacking và CTF. Có thời gian các bạn nên tìm hiểu.

Sơ bộ và nhìn chung thì một fuzzer có các bước sau đây:

1. Cung cấp corpus (tùy chỉnh)
2. Biến đổi input (Mutate)
3. Chạy chương trình với input trên.
4. Nhận biết xem chương trình có crash hay không.
5. Quay lại bước 2

Ở trong các fuzzer đương đại thì ở bước 2 và 3, họ còn sử dụng chiến thuật Feedback driven, tức là nhận biết rằng input đó đã đi tới đâu trong chương trình, và thực hiện biến đổi input một cách “thông minh” dựa trên dữ kiện đó. Thường là họ dùng coverage để làm chuyện đó. Bạn nên đọc bài này: [15 minute guide to fuzzing](#).

Hiện tại fuzzing cũng đang trở nên mainstream, được cả cộng đồng đang quan tâm, sinh ra cũng có rất nhiều công cụ cực chuẩn và open source, cùng các thuật toán rất hay ho như: afl-fuzz, libFuzzer, honggfuzz,... Nếu bạn đang muốn viết fuzzer cho riêng một chương trình nào đấy thì nên đọc qua mã nguồn các công cụ trên để có những ý tưởng cho mình. Kèm theo là sự hỗ trợ từ các thư viện/công cụ để giúp cho việc detect các lỗi trở nên chính xác và tinh vi hơn (vd: AddressSanitizer, valgrind, ...). Và đừng quên hỏi tại sao cách các

công cụ này làm việc như thế nào, hãy tìm hiểu bằng cách đọc document cũng như mã nguồn của nó.



Thật ra mình tính viết, hoặc trình bày ở một hội thảo nào đó sơ bộ về fuzzing và cách dùng libFuzzer để giúp những developer có thể hiểu và tự fuzz chính sản phẩm của mình, để chủ động hơn trong việc bảo mật các chương trình được viết bằng C/C++. Nhưng cũng không biết khi nào có dịp.

Hiệu quả của fuzzing cũng không cần phải bàn, đa số các bug đã và đang được tìm phần lớn đến từ fuzzing, vì việc này giúp bạn setup cho máy tính có thể tự động chạy 24/7, còn bạn thì thường làm việc một ngày 8-12h thôi đúng không ;p

Mình có ví dụ thế này, giả sử ông Beckham ổng phải sút 3 trái banh như trong clip này. Có người cho rằng clip này là giả, mà thôi kệ đi :khongquantam: , nhưng việc những đường chuyển long pass như đặt của ổng là có thật . Để làm được chuyện này ổng phải luyện ngày đêm, bao nhiêu năm tháng, mới làm được những chuyện đó, mà chưa chắc là 100% lúc nào cũng làm được. Nhưng giả sử luật chơi chỉ là, trong 5 phút bạn phải câu được 3 quả vào trong thùng như trong clip, mà không giới hạn số lần thử. Thì đây mình sẽ tạo nên một cái máy bắn bóng nhưng kiểu máy tập trong bộ môn tennis, bóng chày chẳng hạn. Mình chế làm sao, 1 giây nó bắn được ít nhất 2 quả, thì trong 5 phút mình bắn ít nhất được 600 quả. Và mình tin rằng, với từng ấy cơ hội mình sẽ được 3 quả mà không phải tốn sức tập luyện sút cho chính xác, chỉ cần biết cách thức hoạt động của đối tượng, và làm ra cái công cụ tự động đó. Nhưng để thành cao nhân, bạn phải kết hợp code review để hiểu rõ thêm về target (đối tượng) mình đang làm để dẫn dắt công cụ đi một cách hiệu quả nhất. Code review ra sao, ta cùng tìm hiểu tiếp.

- Code review là công việc bạn sẽ phải đọc hiểu mã nguồn và tìm kiếm lỗi dựa trên những hiểu biết đó. Đôi khi bạn cũng chẳng có mã nguồn để mà đọc, phải dùng kĩ thuật RE để hiểu nó, vì vậy tùy vào đối tượng bạn đang làm mà mảng này cũng cần đòi hỏi kĩ năng Dịch ngược, nhưng mình nghĩ dù thế nào thì nếu bạn đã tham gia mảng này thì hãy nên trang bị cho mình những kiến thức xoay quanh nó. Sau khi đọc hiểu một chương trình, đối với những lỗi cơ bản, dễ tìm thấy (có thể bằng các signature) thì song đó với những bug phức tạp, logic, bạn phải chơi trò lắp ghép, từ input, làm sao có thể đi qua các step, để tìm đến dòng code bị lỗi, cái này hoàn toàn là kĩ năng, được rèn luyện bằng thời gian. Quá trình này có thể giúp bạn tìm ra những lỗi phức tạp hơn, mà nhiều khả

năng fuzzer không thể chạm đến. Vì dù gì ở một góc độ nào đó, não người vẫn thông minh hơn công cụ.

- Tóm lại, hãy cân đối giữa việc fuzzing và code review cho chính bản thân mình, nhận ra mình thích và mạnh ở điểm nào, và mình nghĩ kết hợp cả 2 thì mới là tốt nhất.
- Sau đó sẽ là các bước khai thác lỗi đã có, bypass các mitigation như thế nào, chiến thuật để exploit chương trình đó ra sao, điều khiển thanh ghi, blah blah. Cách tốt nhất để học nó cho một người chưa biết gì là chơi CTF và đọc writeup. Bạn có thể tìm đọc các blog của những người như: suto, peternguyen, meepwn/piggybird/babyphd/nightstorm ctf team, ... bằng tiếng Việt. còn tiếng Anh thì bao la. Nhưng nhớ là hãy tự thân vận động để giải bằng khả năng của mình, rồi hãy tham khảo các bài viết.
- Giai đoạn hiện tại, mình đã tạm gác việc tập trung hết thời gian cho mảng Web để chuyển sang tìm hiểu sâu hơn ở mảng này, nhưng tương lai mình nghĩ Application Security (là tất tần tạt về bảo mật ứng dụng, có thể là web, có thể là một ứng dụng được viết bằng bất kì ngôn ngữ nào) sẽ là một mảng mạnh và cơ hội được làm công việc bằng đam mê nhiều hơn. Nhìn chung, đòi hỏi kĩ năng đọc và tìm ra bugs rất nhiều. Và đối với bản thân mình, tìm bug là một công việc thú vị nhất trong An toàn thông tin. Vì nó mang đầy tính sáng tạo trong công việc đó, và mình thì không mấy thích thú với những có xu hướng việc lặp đi lặp lại, có phần hơi nhàm chán.
-  Nghệ thuật tận dụng lỗi phần mềm – Nguyễn Thành Nam
-  Hacking: The Art of Exploitation
- Lab và course thì trên mạng có rất nhiều, bạn có thể tham khảo: <https://github.com/RPISEC/MBE> . Về phần mình, mình chỉ đọc duy nhất cuốn của anh NamNT và tập luyện bằng việc chơi CTF, vì cơ bản cuốn đó đã nói hết những kiến thức cần có rồi, còn lại là bạn sẽ phải tự mày mò, nâng cao trình độ.

 Cryptography (Mật mã học)

 Forensics (Điều tra chứng cứ số)

Networking Security (An toàn mạng)

...

Tạm thời đến đây, có thời gian mình sẽ viết tiếp.

20/07/2017

Updated 09/08/2017, à theo thời gian mình có thể thêm thắt ý vào các phần, nên rất có thể nội dung những mảng bạn đã đọc qua trước đó đã thay đổi, nên đôi khi nên check lại xem nhé.

Edit: Sorry, lúc viết bài này mình tính mặc định trong đầu Reverse và Exploit nói chung 1 mảng, mà quên edit. Thôi mình split thành 2 nhé, hix. Hèn chi thấy thiếu thiếu

PREVIOUS POST

[GOOGLE CTF 2017] THE X SANITIZER – WRITEUP

NEXT POST

SVATTT 2017 (VÒNG LOẠI): SAD BOOK WRITEUP

CATEGORIES

Z

TAGS

AN TOÀN THÔNG TIN COMPUTER SECURITY INFOSEC



WRITTEN BY:

MANHLUAT



21 COMMENTS