Tạp chí Khoa học ĐHQGHN, Khoa học Tự nhiên và Công nghệ 28 (2012) 255-263

Song song hóa thuật toán so khớp mẫu QuickSearch trong NIDS sử dụng mô hình chia sẻ bộ nhớ trên OpenMP và PThreads

Lê Đắc Nhường[1], Nguyễn Gia Như[2], Lê Đăng Nguyên[1], Lê Trọng Vĩnh[3]

[1]*Khoa Công nghệ thông tin, Trường Đại học Hải Phòng*
[2]*Trường Đại học Duy Tân, Đà Nẵng*
[3]*Trường Đại học Khoa học Tự nhiên, ĐHQGHN, 334 Nguyễn Trãi, Hà Nội, Việt Nam*

Nhận ngày 3 tháng 8 năm 2012

Tóm tắt. Hệ thống phát hiện xâm nhập mạng (NIDS: Network Intrustion Detection System) có nhiệm vụ theo dõi và phát hiện sự xâm nhập cũng như các hành vi khai thác trái phép tài nguyên làm tổn hại đến tính bảo mật, tính toàn vẹn và tính sẵn sàng của hệ thống. Việc phát hiện các nguy cơ tiềm ẩn trong NIDS được thực hiện bằng cách so khớp nội dung gói tin với các mẫu đã biết. Với sự đa dạng về số lượng các đợt tấn công, hình thức tấn công và sự phát triển nhanh của các loại virus, spam, trojan...việc thu thập đầy đủ các mẫu đang gặp rất nhiều khó khăn làm cho kích thước tập mẫu ngày càng tăng. Vấn đề đặt ra là cần có những thuật toán so khớp nhanh có thể thực thi trên một tập lớn các mẫu. Trong bài báo này, chúng tôi sẽ thực hiện song song hóa thuật toán so khớp mẫu QuickSearch sử dụng mô hình chia sẻ bộ nhớ trên OpenMP và PThreads nhằm nâng cao hiệu năng và tốc độ xử lý gói tin trong NIDS với các tập luật của Snort.

Từ khóa: Pattern Matching, QuickSearch, Hệ thống phát hiện xâm nhập mạng, OpenMP, PThreads.

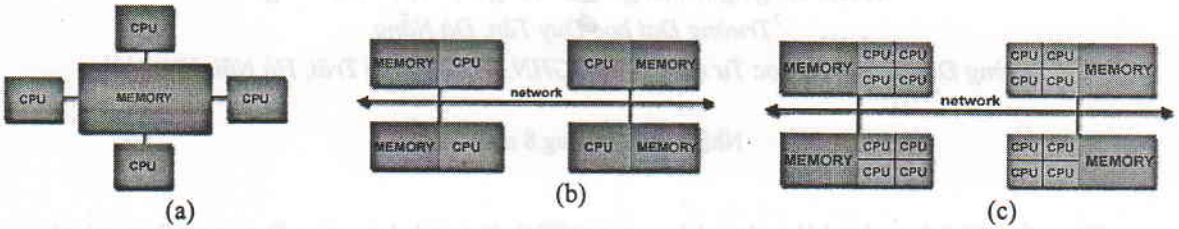
1. Giới thiệu

Theo tiếp cận truyền thống, các chương trình được viết cho máy tính sẽ được thực thi trên một máy tính chỉ có một bộ vi xử lý (*CPU*). Chương trình đó được xử lí một cách tuần tự từng lệnh, tại một thời điểm chỉ có một chỉ thị được xử lí. Cùng với sự phát triển của các công nghệ chế tạo bộ CPU nhiều lõi và các kiến trúc song song, hướng tiếp cận song song các chương trình đang thu hút được rất nhiều sự quan tâm nghiên cứu. Theo cách hiểu đơn giản nhất, tính toán song song là đồng thời sử dụng nhiều tài nguyên để giải quyết một bài toán. Các tài nguyên tính toán là một máy tính được lắp đặt nhiều CPU hay một số máy tính được bó song song với nhau (*PC-Clustering*). Các bài toán thực hiện song song có đặc tính chung là cho phép chia nhỏ một công việc lớn thành nhiều phần việc nhỏ hơn và có thể được giải quyết đồng thời. Tức là tại một thời điểm, có thể thực thi nhiều chỉ thị chương trình. Khi đó, thời gian xử lí bài toán sẽ giảm xuống bởi vì nhiều tài nguyên tính toán được sử dụng.

* Tác giả liên hệ. ĐT: 84-987394900.
E-mail: Nhuongld@hus.edu.vn

1.1. Kiến trúc song song

Kiến trúc chia sẻ bộ nhớ (Shared memory): tất cả các CPU hoạt động độc lập đều có thể truy cập đến một không gian địa chỉ chung gọi là chia sẻ chung tài nguyên bộ nhớ (Hình 1.a). Các CPU khác có khả năng nhìn thấy các thay đổi trong bộ nhớ do một CPU tác động. Ưu điểm của kiến trúc này là không gian địa chỉ toàn cục cho phép lập trình bộ nhớ trở nên thân thiện, dễ dàng hơn. Việc chia sẻ dữ liệu giữa các tác vụ nhanh và đồng nhất. Nhược điểm là khả năng phát triển mở rộng bộ nhớ và CPU. Việc bổ sung thêm CPU dẫn đến gia tăng hoạt động trên bộ nhớ chia sẻ, tăng hoạt động trên đường nối giữa bộ nhớ - CPU. Chi phí sẽ tăng khi tăng số lượng CPU và dung lượng bộ nhớ chia sẻ [1].



Hình 1. Các kiến trúc bộ nhớ song song.

Kiến trúc bộ nhớ phân tán (Distributed Memory): các hệ thống riêng rẽ được kết nối với nhau tạo ra một liên kết bộ nhớ và CPU. Mỗi CPU sẽ có không gian bộ nhớ cục bộ của riêng nó (Hình 1.b). Các địa chỉ bộ nhớ trong một CPU này sẽ không được ánh xạ đến CPU khác, vì vậy không có khái niệm không gian địa chỉ toàn cục trên tất cả các CPU. Các vùng bộ nhớ cục bộ sẽ hoạt động một cách độc lập, các thay đổi tạo ra trên vùng bộ nhớ cục bộ không ảnh hưởng đến bộ nhớ của các CPU khác. Khi một CPU này muốn truy cập đến dữ liệu của một CPU khác thì người lập trình phải định nghĩa một cách rõ ràng thời điểm và cách thức dữ liệu được chia sẻ. Việc đồng bộ hóa giữa các tác vụ cũng do người lập trình đảm nhiệm. Ưu điểm là việc mở rộng dung lượng bộ nhớ hoàn toàn độc lập với số lượng CPU do mỗi CPU có một vùng bộ nhớ của riêng nó. Mỗi CPU có thể truy cập nhanh chóng các vùng dữ liệu của riêng nó mà không ảnh hưởng đến các CPU khác. Nhược điểm là người lập trình sẽ phải đảm bảo đồng bộ của việc truyền thông giữa các CPU, ánh xạ các cấu trúc dữ liệu đang có trên không gian bộ nhớ toàn cục sang tổ chức bộ nhớ phân tán trở nên gặp rất nhiều khó khăn [1].

Mô hình lai (Hybrid Distributed-Shared Memory): Các máy tính lớn nhất và nhanh nhất ngày nay đều dùng cả 2 loại kiến trúc bộ nhớ phân tán và bộ nhớ chia sẻ kết hợp gọi là mô hình lai (Hình 1.c).

1.2. Mô hình lập trình song song

Mô hình chia sẻ bộ nhớ: Các tác vụ chia sẻ một vùng địa chỉ chung và sẽ đọc và viết một cách bất đồng bộ. Các cơ chế khác nhau như là khóa/truyền tin có thể được sử dụng để truy cập vùng bộ nhớ chia sẻ. Một thuận lợi của mô hình này từ quan điểm của người lập trình là không có khái niệm "quyền sở hữu", vì vậy không cần thiết phải chỉ định rõ ràng việc truyền dữ liệu giữa các tác vụ. Việc phát triển

chương trình thường đơn giản. Một trong những bất lợi lớn là tốc độ. Chúng ta sẽ gặp khó khăn trong việc hiểu rõ và quản lí dữ liệu một cách nội bộ.

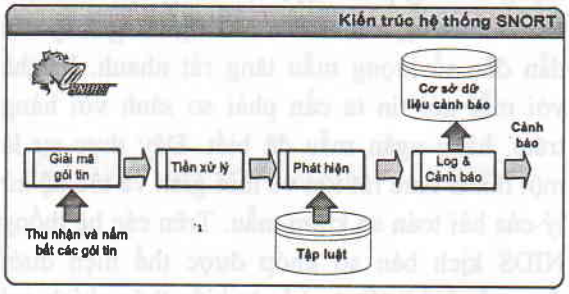
Mô hình Threads: lập trình song song với các luồng (*Thread*) cho phép một tiến trình đơn có thể có nhiều đường dẫn thực thi đồng thời. Công việc của thread giống như là chương trình con bên trong chương trình chính. Bất kỳ thread nào cũng có thể thực thi một chương trình con bất kỳ cùng thời điểm với các thread khác. Các thread liên lạc với nhau thông qua bộ nhớ toàn cục. Điều này đòi hỏi phải đồng bộ hóa để đảm bảo rằng tại một thời điểm bất kỳ không có nhiều hơn một thread cập nhật cùng một vùng bộ nhớ toàn cục. Các thread có thể được tạo ra hoặc hủy bỏ, nhưng chương trình chính sẽ vẫn hiện diện để cung cấp các tài nguyên chia sẻ cần thiết cho đến khi ứng dụng kết thúc. Các thread thường được gắn với các kiến trúc bộ nhớ chia sẻ và hệ điều hành.

Mô hình truyền thông điệp: Mô hình truyền thông điệp có đặc điểm là các tác vụ có thể sử dụng vùng bộ nhớ cục bộ của nó trong khi tính toán. Nhiều tác vụ có thể cùng nằm trên cùng một máy vật lí hoặc trên các máy chuyên biệt. Các tác vụ tráo đổi dữ liệu với nhau thông qua việc truyền tin bằng cách gửi và nhận các thông điệp. Việc truyền dữ liệu thường yêu cầu thêm các hoạt động xử lí để thực hiện bởi mỗi tiến trình.

Mô hình dữ liệu song song: Trong mô hình này, phần lớn các phần việc song song tập trung vào việc thực hiện các thao tác trên tập dữ liệu. Dữ liệu thường được sắp xếp vào các cấu trúc thông dụng, chẳng hạn như mảng hoặc khối lập phương 3 chiều. Một tập tác vụ làm việc chung trên cùng cấu trúc dữ liệu, tuy nhiên mỗi tác vụ làm việc trên một phần khác nha của cùng cấu trúc dữ liệu. Các tác vụ thực hiện cùng các thao tác trên phần việc của nó [2].

2. Hệ thống phát hiện xâm nhập mạng

Cùng với sự phát triển nhanh về số lượng các ứng dụng trên mạng Internet thì việc bảo đảm an ninh cho các hệ thống thông tin càng trở nên cấp thiết hơn bao giờ hết. Bài toán an ninh thông tin nói chung và an ninh mạng nói riêng đang rất được quan tâm không chỉ tại Việt Nam mà trên toàn thế giới. Trong các hệ thống phát hiện xâm nhập mạng (NIDS: *Network Intrustion Detection System)*, hệ thống lọc các trang web, ngăn chặn virus, spam…thì các thuật toán so khớp mẫu có vai trò quan trọng nhất. NIDS tiến hành thu thập thông tin từ rất nhiều nguồn khác nhau trong hệ thông đang bảo vệ sau đó tiến hành phân tích các thông tin đó theo nhiều các khác nhau đề phát hiện các xâm nhập trái phép. Khi NIDS có thêm khả năng ngăn chặn các nguy cơ xâm nhập được phát hiện thì gọi là hệ thống ngăn chặn xâm nhập NIPS (*Network Intrustion Prevention System*) [3].



Hình 2. Kiến trúc hệ thống phát hiện xâm nhập mạng Snort.

Có 2 cách tiếp cận cơ bản với NIDS là: phát hiện lạm dụng (*Misuse Detection Model*) và phát hiện bất thường (*Anomaly Detection Model*). Phát hiện lạm dụng là phát hiện kẻ xâm nhập đang cố gắng đột nhập vào hệ thống thông qua việc sử dụng một số kỹ thuật đã biết. Việc mô tả đặc điểm cách thức xâm nhập được thể hiện như một mẫu (*Pattern*), hệ thống có nhiệm vụ kiểm soát nội dung với các mẫu đã có. Mẫu có thể là một chuỗi bít cố định như mã một

virus trong file hay một tập các hành động nghi ngờ. Khi hoạt động, hệ thống liên tục so sánh hành động hiện tại với một tập các kịch bản xâm nhập (*Intrusion scenario*) để cố gắng dò ra kịch bản đang được thực thi. Các kỹ thuật phát hiện lạm dụng khác nhau ở cách thức mô hình hóa hành vi chỉ định một sự xâm nhập qua các luật (*Rules*), kịch bản. Sau đó sẽ tiến hành so khớp các dấu hiệu giống như các phần mềm quét virus truyền thống. Khi hacker tìm cách khai thác lỗ hổng đã biết thì NIDS cố gắng để đưa lỗi đó vào cơ sở dữ liệu của mình. Phát hiện bất thường là phân biệt giữa những hành vi bình thường và bất bình thường đang diễn ra. Ranh giới giữa dạng chấp nhận được và dạng bất thường của đoạn mã thể hiện qua sự giống và khác nhau giữa các chuỗi bit. Kỹ thuật phát hiện bất thường có 2 loại tĩnh (*Static*) và động (*Dynamic*) [4, 5].

Tuy nhiên trong thực tế cơ sở tấn công mạng thường phức tạp, có nhiều bước và qua nhiều thiết bị, mô hình tấn công cũng thay đổi dẫn đến số lượng mẫu tăng rất nhanh. Vì thế với mỗi gói tin ta cần phải so sánh với hàng trăm, hàng ngàn mẫu đã biết. Đây thực sự là một thách thức rất lớn về thời gian và tốc độ xử lý của bài toán so khớp mẫu. Trên các hệ thống NIDS kịch bản so khớp được thể hiện dưới dạng chuỗi bit (*String*) hoặc biểu thức chính qui (*Regular Expression*) nhằm tạo thuận lợi trong việc chia sẻ cơ sở dữ liệu mẫu. Một số ứng dụng mã nguồn mở như: *Snort, Source Fire, Bro, ClamAV* [6]...

3. Mô hình bài toán so khớp mẫu trong NIDS

3.1. Bài toán so khớp mẫu

So khớp mẫu (*Pattern Matching*) là tìm ra tất cả các lần xuất hiện của mẫu X trong gói tin Y. Trong [6], bài toán so khớp mẫu được mô tả như sau: *Cho một bảng chữ cái A, một mẫu P (P[1..m]) độ dài m và một gói tin M (M[1..n]) độ dài n (trong đó m<<n). Bài toán đặt ra là cần tìm các vị trí xuất hiện của P trong M hoặc P có khớp với một chuỗi con của M hay không?* Các thuật toán so khớp mẫu đều sử dụng cơ chế *cửa sổ trượt* (một khung có kích thước bằng với kích thước của mẫu cần tìm) để so sánh các ký tự của mẫu trong cửa sổ với các ký tự trong gói tin.

Có thể phân loại các thuật toán so khớp mẫu theo 2 tiêu chí:

- Dựa trên số lượng mẫu ta có so khớp mẫu đơn (*Single Pattern*) và so khớp đa mẫu (*Multiple Patterns*).
- Dựa trên cơ sở thiết kế thuật toán ta có 3 loại: so khớp dựa trên tiền tố (*prefix*), so khớp hậu tố (*suffix*) và so khớp thừa số (*factor*).
- Dựa trên kết luận ta có: so khớp chính xác (*Extract matching*) và so khớp sấp xỉ (*Approximate Matching*).
- Các thuật toán so khớp mẫu đều có 2 giai đoạn: tiền xử lý (*preprocessing phase*) và tìm kiếm (*Searching phase*). Việc đánh giá các thuật toán được thực hiện dựa trên dung lượng bộ nhớ sử dụng và tốc độ so khớp trong trường hợp trung bình. Trong bài báo này chúng tôi sẽ cài đặt song song thuật toán so khớp mẫu chính xác Quick-Search trên OpenMP và PThreads.

3.2. Thuật toán tìm kiếm nhanh (Quick Search)

Thuật toán Quick-Search (QS) là một thuật toán đơn giản hóa của BM (Boyer-Moore) chỉ sử dụng bảng dịch *"Bad-character shift"* [7]. Thuật toán QS dễ khai báo và thực hiện trên các tập mẫu lớn và ngắn. Sau mỗi lần thử, cửa sổ trượt sẽ dịch chuyển sang vị trí tiếp theo trong gói tin là $M[j..j+m-1]$, độ dài mỗi lần dịch ít nhất sẽ bằng 1.

Độ phức tạp trung bình thời gian của thuật toán Quick-Search trong giai đoạn tiền xử lý là $O(m+|\Sigma|)$ và không gian là $O(|\Sigma|)$. Độ phức tạp trong giai đoạn tìm kiếm là $O(m*n)$. Trong đó, n là kích thước gói tin *Msg*, m là kích thước tập mẫu *P*, $|\Sigma|$ là kích thước tập ký tự.

Thuật toán QS được cài đặt trên C với 2 giai đoạn tiền xử lý và tìm kiếm được mô tả trong hình 3.

```
1. void preQsBc (char *P, int m, int qsBc[])
2. {
3. int i;
4. for (i=0; i <= m; ++i)
5.         qsBc[i] = m+1;
6. for (i=0; i<m ; ++i)
7.         qsBc[P[i]] = m -i;
8. }
9. void QuickSearch (char *P, int m, char *Msg, int n)
10. {
11. int j;
12. qsBc[PSIZE];
13. preQsBc(P, m, qsBc);                    /* Preprocessing */
14. j++;                                 /* Searching */
15. while (j < n - m)
16. {
17.         if (memcmp(P, Msg + j, m)
18.                 OUTPUT (j);
19.         j+=qsBc [Msg[j+ m]]              /* shift */
20. }
21. }
```

Hình 3. Cài đặt Quick-Search trên C.

4. Song song hóa thuật toán so khớp mẫu quicksearch

Đã có rất nhiều công cụ hỗ trợ lập trình song song [8] như: PVM (*Parallel Virtual Machine*), MPI (*Message Passing Interface*), OpenMP (*Open MultiProcesing*), Pthreads... Trong bài báo này tôi sẽ đánh giá hiệu quả của 2 cách tiếp cận giữa OpenMP và Pthreads khi song song hóa thuật toán QuickSearch.

4.1. Song song QuickSearch với OpenMP

OpenMP [9] được sử dụng cho các mô hình song song chia sẻ bộ nhớ, phù hợp cho các ứng dụng song song ở mức độ vừa phải. Ưu điểm rõ ràng nhất của OpenMP chính là tính đơn giản khi viết chương trình bởi OpenMP hoàn toàn giữ nguyên cấu trúc lập trình tuần tự, song song hóa chỉ được thể hiện qua các cấu trúc dẫn hướng biên dịch vòng lặp.

OpenMP có 3 cơ chế lập tình song song là:

- Song song hóa dựa trên cơ chế luồng (*Thread based parallelism*): chương trình xử lý trên bộ nhớ toàn cục bao gồm nhiều luồng thực thi đồng thời. OpenMP dựa vào sự tồn tại của nhiều luồng trên một mô hình lập trình chia sẻ bộ nhớ chung.

- Mô hình song song hiện (*Explicit Parallelism*): là một mô hình lập trình không tự động. Người lập trình có quyền điều khiển việc song song hóa một cách độc lập.

- Mô hình Fork-Join: tất cả các chương trình song song đều bắt đầu với việc xử lý đơn bởi một luồng chủ (*master thread*). Luồng chủ này sẽ thực thi một cách tuần tự cho tới khi bắt gặp vùng song song (*parallel region*) đầu tiên.

Với hướng tiếp cận song song hóa dựa trên cơ chế luồng, quá trình kiểm soát các gói tin được thực hiện ở bên gửi và nhận theo cả hai chiều, các gói tin đến sẽ được xử lý bằng cách kiểm tra phần tiêu đề và nội dung. Nếu tiêu đề và nội dung khớp với bất kỳ một luật nào trong tập luật xem xét thì gói tin đó sẽ bị loại bỏ. Chúng tôi chia mỗi gói tin đến thành 2 phần: tiêu đề (*header*) và nội dung (*content*). Phân lớp tập luật và lưu trữ trong 2 danh sách liên kết: một danh sách lưu tiêu đề và một lưu nội dung cần kiểm soát. Mô tả cài đặt song song thuật QS với OpenMP được thể hiện trong hình 4.

```
Input:  Luồng gói tin
Output: Khớp hay không khớp với tập luật
   1. Khai báo số lượng luồng.
   2. Khởi tạo bắt đầu tính thời gian
   3. #pragma omp parallel
   4. {
   5.    Tid = = số lượng luồng;
   6.    If ( Tid = = 0)
   7.      {
   8.         N_th = Số lượng luồng;
   9.      }
   10.   #pragma omp schedule (static, chunk)
   11.      Lặp lại việc nắm bắt các gói tin
   12.      Gọi hàm PreQsBc;
   13.      Gọi hàm QuickSearch;
   14. }
   15. Dừng lại và tính thời gian xử lý.
```

Hình 4. Cài đặt song song Quick-Search với OpenMP.

4.2. Song song QuickSearch với PThreads

Thread là mô hình lập trình phổ biến cho phép nhiều thread đơn có thể chạy trên cùng một tiến trình, và các thread này có thể chia sẻ tài nguyên của tiến trình cũng như có thể tính toán độc lập. Mô hình này được áp dụng cho một tiến trình đơn lẻ để cho phép tính toán song song trên một hệ thống đa xử lý. Trong phần này, tôi sẽ trình bày mô hình Thread theo chuẩn IEEE POSIX 1003.1c, được gọi là POSIX thread hay PThread [10].

Mô tả song song hóa thuật QS với Thread được thể hiện trong hình 5.

```
Input:  Luồng gói tin
Output: Khớp hay không khớp với tập luật
   1. Khai báo số lượng luồng.
   2. Khởi tạo bắt đầu tính thời gian
   3. Tính tổng lưu lượng
   4. Phân bố các cổng cho mỗi Thread
   5. Tạo các Thread và gọi các hàm thực hiện song song
   6. {
   7.       Thread_Create(ThreadID, NULL, Thread_Function,Ptr);
   8. }
   9. Nối các Thread
   10. Dừng lại và tính thời gian xử lý.
```

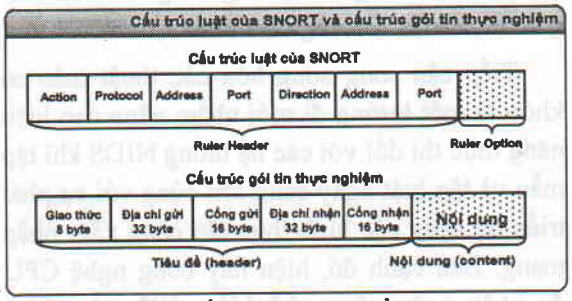
Hình 5. Cài đặt song song Quick-Search với PThreads.

Tương tự như OpenMP, việc song song hóa thuật toán QS được thực hiện bằng cách tạo ra các Thread được lưu lại trong ThreadID. Trong đó, mỗi Thread sẽ gọi đến các hàm Thread_function là Quick-Search, PreQsBc. Khi thực hiện và tạo các ThreadID được lưu trữ bởi các con trỏ *Ptr* để kết nối lại các Thread trả lại kết quả cuối cùng trong bước 8.

Đối với mỗi gói tin đến, việc so khớp gói tin với tập luật được thực hiện với các luồng khác nhau trên các CPU. Trong MPI để so khớp gói tin thì CPU phải gửi thông điệp yêu cầu thông của các gói tin trên các CPU khác. Còn OpenMP lại làm việc trên các dữ liệu chia sẻ nên các CPU hoàn toàn biết thông tin của gói tin nằm trên các CPU khác.

5. Thực nghiệm và đánh giá

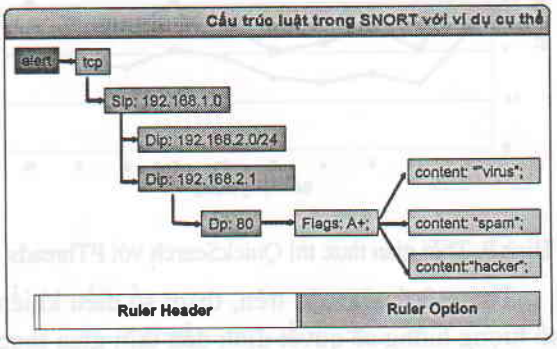
Để đánh giá thời gian thực thi và hiệu quả của việc song song hóa thuật toán với OpenMP và PThreads, chúng tôi đã cài đặt các thuật toán trên ngôn ngữ C. Các tham số thử nghiệm là kích thước chiều dài nội dung gói tin, số lượng luồng, lưu lượng truyền tải, kích thước tập luật, chiều dài của gói tin và chiều dài của tập luật. Cấu trúc gói tin được thử nghiệm được minh họa trong hình 6.



Hình 6. Cấu trúc gói tin kiểm soát.

Trong Snort, cấu trúc phần Ruler Header gồm 4 thành phần. *Action* qui định hành động nào được thực thi khi các dấu hiệu của gói tin được nhận dạng chính xác bằng luật đó. Thường nó sẽ tạo ra một cảnh báo, một log thông điệp hoặc kích hoạt một luật khác. *Protocol* qui định việc áp dụng luật cho các gói tin thuộc một giao thức cụ thể nào đó như IP, TCP, UDP, ICMP...*Address* là địa chỉ nguồn và địa chỉ đích, các địa chỉ có thể là của một hay nhiều máy hoặc của một mạng nào đó. Việc xác định nguồn hay đích phụ thuộc vào phần Direction. *Port* xác định cổng nguồn, đích của gói tin được kiểm soát.

Phần Ruler Option được đặt trong dấu ngoặc đơn. Nếu có nhiều Option thì các Option sẽ được phân cách nhau qua dấu chấm phẩy ";" và có thể được kết nối logic với nhau bằng AND. Một Option gồm 2 phần: một từ khóa và một tham số, hai phần phân cách nhau bằng dấu hai chấm ":". Ví dụ minh họa đặc tả luật trong Snort được thể hiện trong hình 7.

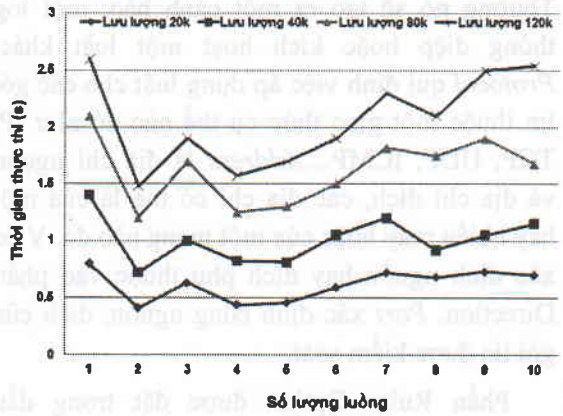


Hình 7. Biểu diễn luật SNORT với các ví dụ.

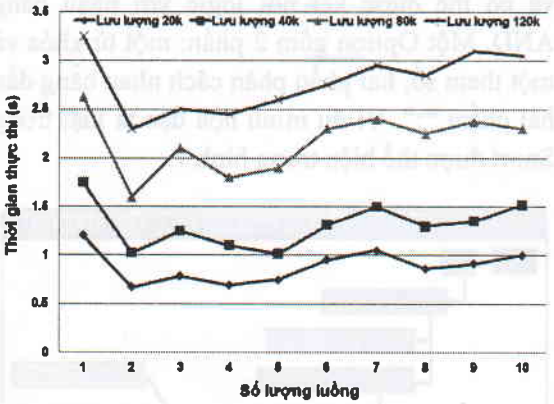
Các thực nghiệm được tiến hành trên máy tính có bộ vi xử lý Intel Core 2 Duo 2.66 GHz (E6700), Cache 4MB, Bus 1066MHz, DDR2-1066Mhz 2x2GB hỗ trợ công nghệ siêu phân luồng. Các phần mềm hệ thống sử dụng gồm: Snort 2.4.3, IDS Center 1.1 RC4, WinPcap 3.1, Ethereal 0.10.14, Packet Excalibur 1.0.2.

Kết quả thực nghiệm trên tập luật có kích thước 3kB, các lưu lượng lần lượt là 20kB,

40kB, 80kB, 120kB và số lượng luồng được thiết lập lần lượt là 1 đến 10 như sau:



Hình 8. Thời gian thực thi QuickSearch với OpenMP.



Hình 9. Thời gian thực thi QuickSearch với PThreads.

Trong 2 thuật toán trên, tham số điều khiển số lượng luồng sẽ quyết định đến thời gian thực thi của thuật toán. Qua OpenMP và PThreads giúp chúng ta thấy được hiệu quả, tiềm năng của chương trình, việc tạo ra một thread sử dụng ít tài nguyên và chi phí của hệ điều hành hơn rất nhiều so với việc tạo ra một tiến trình thông thường.

So với hướng tiếp cận song song sử dụng MPI, OpenMP và PThreads thực hiện phân đoạn mã song song, mỗi tiến trình vẫn thực hiện tính toán trên miền con dữ liệu của riêng nó. Qua kết quả thống kê ở trên ta thấy thời gian của chương trình song song trên bộ vi xử lý 2 nhân giảm được gần một nửa so với chương trình tuần tự trên 1 CPU vì công việc được chia cho 2 CPU thực hiện đồng thời. Sở dĩ thời gian không thể giảm đi đúng một nửa là vì sự thiếu đồng bộ của hai CPU và nhân của hệ điều hành mất một phần thời gian để thiết lập một vùng song song khi bắt gặp một cấu trúc song song. So sánh thời gian thực thi giữa OpenMP và PThreads trên cùng một tập luật với các tham số Thread thiết lập như nhau thì OpenMP thực hiện nhanh hơn so với PThreads và đạt hiệu quả cao nhất với số Thread là 2.

5. Kết luận

Việc thực hiện song song thuật toán Quick-Search dựa trên mô hình chia sẻ bộ nhớ đã làm giảm thời gian thực thi so với chương trình tuần tự. Các chiến lược song song khác nhau sẽ đem lại những hiệu quả khác nhau về thời gian. Thời gian thực thi giảm được gần hai lần khi thực hiện trên bộ vi xử lý 2 nhân. Tuy nhiên, khi thực hiện song song hóa không phải trong trường hợp nào cũng hiệu quả về mặt thời gian như đã thống kê trong hình 8, 9. Nếu không song song hóa một cách hợp lý có thể xảy ra nghịch lý về song song có nghĩa là thời gian thực hiện chương trình song song lớn hơn thời gian thực hiện chương trình tuần tự.

Tiếp cận song song hóa các thuật toán so khớp là một hướng đi mới nhằm nâng cao hiệu năng thực thi đối với các hệ thống NIDS khi tập mẫu và tập luật ngày càng lớn cùng với sự phát triển đa dạng các hình thức tấn công xâm nhập mạng. Bên cạnh đó, hiện nay công nghệ CPU đa nhân ngày càng phổ biến. Việc tận dụng công nghệ đa nhân làm tăng tốc độ tính toán với các chương trình đã có là hướng nghiên cứu đang rất được quan tâm hiện nay.

Tài liệu tham khảo

[1] Hwang, K., Briggs, F. *Computer Architecture and Parallel Processing*. McGrawHill, Inc. New York, NY, 1990.

[2] Quammen, C. *Introduction to Programming Shared-Memory and Distributed Memory Parallel Computers*. ACM Crossroad, Student Edition, 2000.

[3] B. Mukherjee, H. Heberlein, and K. Levitt, Network intrusion detection, *IEEE Network*, vol. 8, no. 3 (1994) 26.

[4] H. Debar, M. Dacier, A. Wespi, Towards a taxonomy of intrusion-detection systems, *Computer Networks*, 31 (1999) 805.

[5] Kedar Namjoshi và Girija Narlikar, *Robust and Fast Pattern Matching For Intrusion Detection*, INFOCOM 2010.

[6] M. Roesch, *Snort: Lightweight intrusion detection for networks*, Proc. of the 1999 USENIX LISA Systems Administration Conference, 1999.

[7] Christian Charras, Therry Lecroq, *Handbook of Exact String Matching Algorithms*, King's College Publications, 2004.

[8] Jianming Yu and Jun Li, *A Parallel NIDS Pattern Matching Engine and Its Implementation on Network Processor*, Proc. of the 2005 International Conference on Security and Management (SAM), 2005.

[9] Ranjit Noronha and D.K. Panda. "*Improving Scalability of OpenMP Applications on Multi-core Systems Using Large Page Support*", 2007.

[10] Jianming Yu, Quan Huang, and Yibo Xue, *Optimizing Multi-thread String Matching for Network Processor-Based Intrusion Management System*, Conference on Communication Network and Information Security (CNIS), 2006.

Paralleling QuickSearch Pattern Matching Algorithm in NIDS use shared Memory Model with OpenMP and PThreads

Le Dac Nhuong[1], Nguyen Gia Nhu[2], Le Dang Nguyen[1], Le Trong Vinh[3]

[1]*Faculty of Information Technology, Haiphong University*

[2]*Duytan University, Danang*

[3]*VNU University of Science, 334 Nguyen Trai, Hanoi, Vietnam*

Network Intrustion Detection System (NIDS) analyzing information about the activities performaned in a computer system or network, looking for evidence of malicious behavior to compromising the confidentiality, integrity and availability of the system. NIDS looking for evidence of malicious behavior based on matching packet contents with known patterns. When network-based attacks often conform to a multi-step process and combine many means with number of unknown viruses, spam, trojan increases in proportion of time then collection of virus signatures are difficulties. A problem is necessary to build fast pattern matching algorithms in a large rulersets. In this paper, we will use shared memory model with open-multiprocessing (OpenMP), PThreads to parallel pattern matching algorithms to improve performance for NIDS with Snort's ruleset

Keywords: Pattern Matching, QuickSearch, Network Intrustion Detection System, OpenMP, PThreads.