

HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG



PHẠM HOÀNG DUY

BÀI GIẢNG

AN TOÀN HỆ ĐIỀU HÀNH

HÀ NỘI 2017

Lời nói đầu

Hệ điều hành là một bộ phận cấu thành quan trọng của hệ thống máy tính giúp cho con người có thể khai thác và sử dụng hiệu quả hệ thống máy tính. Sự phổ biến của máy tính và sự phát triển các ứng dụng mạng khiến cho vấn đề về an toàn trở nên cấp thiết. Bài giảng môn học trước hết giới thiệu các yêu cầu và các kiến trúc cơ bản đáp ứng mục tiêu an toàn cho hệ điều hành. Đồng thời, bài giảng trình bày một số cơ chế phần cứng máy tính cho phép hạn chế, che dấu, và kiểm soát việc sử dụng các tài nguyên máy tính như bộ xử lý, bộ nhớ và các thiết bị vào/ra để hỗ trợ cho hệ điều hành có thể triển khai các dịch vụ cơ bản như quản lý tiến trình một cách an toàn. Ngoài ra, bài giảng giới thiệu các mô hình an toàn chính xác phục vụ cho các yêu cầu an toàn khác nhau của hệ thống như tính toàn vẹn hay tính bí mật. Phần cuối cùng trình bày cách thức mô tả các yêu cầu cũng như kỹ thuật kiểm chứng nhằm kiểm tra và đánh giá khả năng đáp ứng yêu cầu an toàn của hệ thống.

Cấu trúc chi tiết bài giảng như sau.

Chương I giới thiệu vấn đề cơ bản với an toàn hệ điều hành tập trung chủ yếu vào việc mô tả và xác định các mục tiêu cũng như yêu cầu an toàn với hệ điều hành nói chung. Việc xây dựng chính sách mô tả chính xác và xác định mục tiêu an toàn phù hợp đóng vai trò then chốt trong việc phát triển các cơ chế bảo vệ đảm bảo an toàn cho hệ điều hành cũng như việc đánh giá tính an toàn của các cơ chế này với hệ thống. Nhân an toàn và nền tảng tính toán tin cậy là bộ phận quan trọng và then chốt trong việc kiểm soát và đảm bảo an toàn cho các chương trình người dùng cũng như bản thân hệ điều hành.

Chương II trình bày một số cơ chế đảm bảo an toàn phần cứng dựa trên cơ chế bảo vệ theo lớp. Trong đó mỗi lớp được gắn với một mức đặc quyền cho phép tiến trình của người dùng được phép truy nhập tới những thao tác nhất định. Cơ chế này đóng vai trò cốt lõi cho kiểm soát việc thực thi, sử dụng bộ nhớ của các tiến trình cũng như là truy nhập tới các thiết bị vào/ra.

Chương III tập trung vào cơ chế an toàn của kiến trúc x86 đối với các dịch vụ cơ bản của hệ điều hành như quản lý việc thực thi của các chương trình cũng như quản lý bộ nhớ. Các yêu cầu về tính toàn vẹn và bí mật của hệ thống file nhận được sự hỗ trợ và đảm bảo tốt hơn so với việc mã hóa thông thường từ mô-đun hạ tầng tin cậy TPM nhờ vào việc bảo vệ khóa và xác thực được đóng gói bằng phần cứng. Phần cuối chương giới thiệu cách thức triển khai cơ chế bảo đảm an toàn trong hệ điều hành sử dụng kiến trúc x86 đó là Windows và Unix/Linux. Các vấn đề an toàn với các hệ điều hành được thảo luận chi tiết trong phần này. Ngoài ra, phần này cũng giới thiệu một số cách tiếp cận nhằm đảm bảo các yêu cầu an toàn và kiểm chứng ở mức độ cao các yêu cầu này bao gồm nhân an toàn SCOMP và các mô-đun an toàn Linux.

Chương IV trình bày các mô hình an toàn chính xác cho phép mô tả và kiểm chứng các yêu cầu cần phải đạt với mô hình đề xuất. Ngoài mô hình truyền thống dựa trên máy

trạng thái, chương này giới thiệu các mô hình cho phép đảm bảo các yêu cầu về tính toàn vẹn và bí mật của hệ thống máy tính. Phần này chỉ trình bày các kết quả thu được và bỏ qua các chi tiết chứng minh các kết quả này.

Chương V giới thiệu cách thức giúp cho việc đánh giá và kiểm tra các yêu cầu an toàn với hệ thống máy tính thông qua việc xây dựng các đặc tả yêu cầu hệ thống. Ngoài ra, phần này giới thiệu hai kỹ thuật kiểm chứng mã chương trình có đảm bảo yêu cầu đặt ra hay không. Đó là phương pháp phân tích tĩnh và phân tích động. Về cơ bản, phần đầu chương cho phép đánh giá tính an toàn của hệ thống xét về mặt thiết kế. Phần phân tích mã chương trình nhằm kiểm tra và đánh giá việc triển khai cũng như các hành vi của chương trình.

Mục lục

CHƯƠNG 1. TỔNG QUAN VỀ AN TOÀN HỆ ĐIỀU HÀNH.....8

| | | |
|-----|--------------------------------------|----|
| 1.1 | Giới thiệu..... | 8 |
| 1.2 | Các vấn đề về kiến trúc an toàn..... | 15 |
| 1.3 | Chính sách an toàn | 17 |
| 1.4 | Nền tảng tính toán tin cậy..... | 18 |
| 1.5 | Nhân an toàn..... | 23 |
| 1.6 | Câu hỏi ôn tập..... | 26 |

CHƯƠNG 2. CÁC CƠ CHẾ AN TOÀN PHẦN CỨNG27

| | | |
|-----|---------------------------------|----|
| 2.1 | Hỗ trợ các tiến trình..... | 27 |
| 2.2 | Bảo vệ bộ nhớ..... | 29 |
| 2.3 | Kiểm soát thao tác vào/ra | 31 |
| 2.4 | Ảo hóa | 33 |
| 2.5 | Kết luận | 34 |
| 2.6 | Câu hỏi ôn tập..... | 35 |

CHƯƠNG 3. AN TOÀN CÁC DỊCH VỤ CƠ BẢN CỦA HỆ ĐIỀU HÀNH 36

| | | |
|-----|--|----|
| 3.1 | Quản lý tiến trình..... | 36 |
| 3.2 | Quản lý bộ nhớ | 38 |
| 3.3 | Hệ thống file..... | 42 |
| 3.4 | Phân tích an toàn các dịch vụ cơ bản hệ điều hành Windows và Unix/Linux..... | 48 |
| 3.5 | Kết luận | 65 |
| 3.6 | Câu hỏi ôn tập..... | 66 |

CHƯƠNG 4. CÁC MÔ HÌNH AN TOÀN.....67

| | | |
|-----|---|----|
| 4.1 | Vai trò và đặc trưng của mô hình an toàn..... | 67 |
| 4.2 | Mô hình máy trạng thái | 69 |
| 4.3 | Mô hình Harrison-Ruzzo-Ullman..... | 72 |
| 4.4 | Các mô hình khác | 74 |
| 4.5 | Kết luận | 79 |
| 4.6 | Câu hỏi ôn tập..... | 79 |

CHƯƠNG 5. ĐÁNH GIÁ AN TOÀN.....80

| | | |
|-----|---|----|
| 5.1 | Các đặc trưng của đặc tả an toàn | 80 |
| 5.2 | Các kỹ thuật kiểm chứng đặc tả an toàn | 82 |
| 5.3 | Các phương pháp phân rã dữ liệu và chương trình | 88 |
| 5.4 | Các kỹ thuật kiểm chứng mã chương trình | 91 |
| 5.5 | Kết luận | 99 |
| 5.6 | Câu hỏi ôn tập..... | 99 |

Danh mục các hình vẽ

| | |
|--|----|
| Hình 1-1. Hệ điều hành điều phối truy nhập của các tiến trình | 8 |
| Hình 1-2. Ma trận truy nhập với hai tiến trình..... | 12 |
| Hình 1-3. Hệ thống bảo vệ bắt buộc | 14 |
| Hình 1-4. Mô đun TPM phần cứng..... | 20 |
| Hình 1-5. Các bộ phận chức năng của TPM | 22 |
| Hình 1-6. Tương tác giữa tiến trình và bộ giám sát tham chiếu | 25 |
| Hình 2-1. Các lớp bảo vệ tiêu biểu | 28 |
| Hình 2-2. Cấu trúc mô tả phần nhớ công | 29 |
| Hình 2-3. Xung đột bộ nhớ chương trình người dùng | 30 |
| Hình 2-4. Ánh xạ từ bộ nhớ ảo tới bộ nhớ vật lý | 30 |
| Hình 2-5. Mô tả các chế độ truy nhập bộ nhớ của User và System..... | 31 |
| Hình 2-6. Ánh xạ thiết bị ảo tới thiết bị vật lý | 32 |
| Hình 3-1. Thẻ <u>chọn đoạn</u> dữ liệu và lệnh..... | 37 |
| Hình 3-2. Kiểm tra <u>mức đặc quyền</u> của <u>đoạn lệnh</u> | 38 |
| Hình 3-3. Thẻ mô tả đoạn | 39 |
| Hình 3-4. Thẻ chọn đoạn..... | 39 |
| Hình 3-5. Truy nhập bộ nhớ qua GDT | 40 |
| Hình 3-6. Cấu trúc không gian nhớ của tiến trình và thông tin quản lý bộ nhớ | 41 |
| Hình 3-7. Kiểm tra <u>đặc quyền</u> truy nhập bộ nhớ | 41 |
| Hình 3-8. Tính toán cơ sở tin cậy trong quá trình khởi động | 43 |
| Hình 3-9. Khởi động được bảo vệ trong Windows với BIOS truyền thống | 45 |
| Hình 3-10. Bảo vệ và sử dụng <u>khóa mã</u> bằng TPM..... | 47 |
| Hình 3-11. Quá trình giải mã ổ đĩa BitLocker | 47 |
| Hình 3-12. Danh sách kiểm soát truy nhập và thẻ của các tiến trình..... | 50 |
| Hình 3-13. Cơ chế bảo vệ bằng bít chế độ..... | 54 |
| Hình 3-14. Kiến trúc giao tiếp LSM | 59 |
| Hình 3-15. Kiến trúc SCOMP | 62 |
| Hình 3-16. Phần cứng của SCOMP | 63 |
| Hình 4-1. Tương quan giữa các bước phát triển mô hình an toàn | 68 |
| Hình 4-2. Xây dựng luồng thông tin | 75 |
| Hình 4-3. Nguyên tắc an toàn trong Bell-La Padula..... | 76 |
| Hình 5-1. Giao tiếp giữa hai hệ thống..... | 81 |

| | |
|---|----|
| Hình 5-2. Tương quan giữa mô hình và đặc tả an toàn | 82 |
| Hình 5-3. Quan hệ giữa các đối tượng trong hệ thống file | 86 |
| Hình 5-4. Phản chứng cho thấy lỗi với thư mục tuần hoàn | 88 |
| Hình 5-5. Các mức độ chi tiết của việc đặc tả giữa mô hình và việc triển khai | 89 |
| Hình 5-6. Phân rã thuật toán theo các lớp..... | 90 |
| Hình 5-7. Phân rã thuật toán cho thao tác hệ thống file..... | 91 |
| Hình 5-8. Mô hình phân tích tinh đoạn mã | 92 |
| Hình 5-9. Bộ luật <u>sinh cây</u> phân tích..... | 93 |
| Hình 5-10. Cây cú pháp của câu lệnh | 94 |
| Hình 5-11. Cây cú pháp khái quát | 94 |
| Hình 5-12. Luồng điều khiển | 95 |
| Hình 5-13. Đồ thị gọi hàm của ba phương thức <u>larry</u> , <u>moe</u> , <u>curly</u> | 95 |
| Hình 5-14. Các công cụ phân tích tinh | 96 |

Các từ viết tắt

ACL – Access Control List: Danh sách kiểm soát truy nhập

BIOS – Basic Input Output System: Hệ thống vào ra cơ bản

CPU – Central Processing Unit: Đơn vị xử lý trung tâm

DAC – Discretionary Access Control: Kiểm soát truy nhập tùy chọn

DEP – Data Execution Prevention: Ngăn chặn thực thi dữ liệu

EAL - Evaluation Assurance Levels: Mức độ đánh giá an toàn

FAT – File Allocation Table: Bảng cấp phát file

GUI – Graphic User Interface: Giao diện người dùng đồ họa

IDE – Integrated Development Environment: Môi trường phát triển tích hợp

IDS – Intrusion Detection System: Hệ thống phát hiện xâm nhập

LSM – Linux Security Module: Mô-đun an ninh Linux

MAC – Mandatory Access Control: Kiểm soát truy nhập bắt buộc

MLS – Multi-Level Security: An toàn nhiều mức

NX – No eXecution: Cấm thực thi

OS – Operating System: Hệ điều hành

PC – Personal Computer: máy tính cá nhân

PCR – Platform Configuration Register: Thanh ghi cấu hình hệ thống

POSIX – Portable Operating System Interface: Giao tiếp hệ điều hành khả chuyền

RBAC – Role Based Access Control: Kiểm soát truy nhập theo vai trò

SCOMP – Secure Communication Processor: Bộ xử lý truyền thông an toàn

SELinux – Secure-Enhanced Linux: Linux với an ninh tăng cường.

TPM – Trusted Platform Module: Mô-đun hạ tầng tin cậy

UEFI – Unified Extensible Firmware Interface: Giao tiếp firmware mở rộng hợp nhất

UML – Unified Modeling Language: Ngôn ngữ lập mô hình hợp nhất

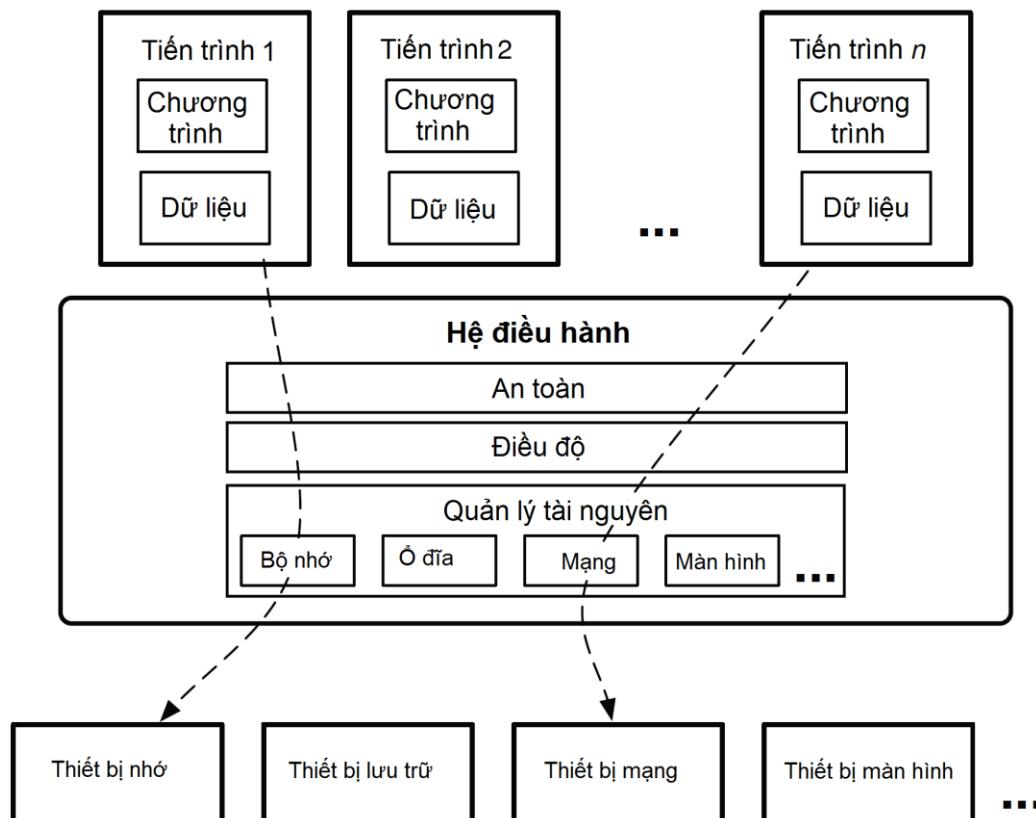
CHƯƠNG 1. TỔNG QUAN VỀ AN TOÀN HỆ ĐIỀU HÀNH

Hệ điều hành là phần mềm đặc biệt cho phép người dùng truy nhập tới các tài nguyên phần cứng khác nhau mà chúng cấu thành hệ thống máy tính cụ thể bao gồm bộ xử lý, bộ nhớ, các thiết bị ngoại vi. Bất kỳ phần mềm nào chạy trên hệ thống máy tính đều sử dụng tập lệnh được cung cấp bởi bộ xử lý của hệ thống đó và phần mềm này cũng có thể cần dùng các tài nguyên khác như truy nhập tới ổ cứng hay mạng. Người dùng thông qua các chương trình để sử dụng phần cứng máy tính như kết nối mạng, xử lý dữ liệu hay giải trí.

Một cách khái quát, vấn đề an toàn hệ điều hành liên quan đến an toàn trong việc chia sẻ hay phân chia các tài nguyên mà các phần mềm sử dụng trong quá trình hoạt động của chúng.

1.1 Giới thiệu

Hệ điều hành hiện đại cần cho phép chạy nhiều chương trình hay người dùng, ở dạng các tiến trình (process), cùng đồng thời chia sẻ sử dụng các tài nguyên của hệ thống (như trong hình dưới đây). Như vậy, hệ điều hành phải đảm bảo ba chức năng căn bản như sau:



Hình 1-1. Hệ điều hành điều phối truy nhập của các tiến trình

- Hệ điều hành phải cung cấp cơ chế sử dụng tài nguyên hiệu quả. Nói cách khác, hệ điều hành cần xác định rõ cách thức cách tiến trình sử dụng tài nguyên phần cứng và ngăn chặn các xung đột khi sử dụng tài nguyên. Các chức năng tiêu biểu gồm có: quản lý CPU, quản lý bộ nhớ, quản lý hệ thống file, giao thức mạng, ...;
- Hệ điều hành phải cung cấp cơ chế điều độ giữa các chương trình người dùng đảm bảo việc sử dụng tài nguyên công bằng;
- Hệ điều hành phải kiểm soát việc truy nhập tới các tài nguyên sao cho chương trình người dùng không ảnh hưởng một cách vô tình hay xấu tới chương trình khác. Đây chính là vấn đề đảm bảo an toàn cho các chương trình chạy trong hệ thống.

Việc đảm bảo các chương trình hoạt động một cách an toàn lệ thuộc vào việc triển khai đúng đắn các cơ chế chia sẻ và điều độ tài nguyên như sau:

- Các cơ chế truy nhập tài nguyên phải xác định ranh giới các tài nguyên và đảm bảo các thao tác tới các tài nguyên này không xung đột với nhau.
- Cơ chế điều độ phải đảm bảo tính sẵn sàng của tài nguyên cho các chương trình để ngăn chặn việc từ chối dịch vụ.

Một cách lý tưởng các cơ chế này đảm bảo các chương trình hoạt động bình thường và không gian nhớ không bị tổn thất cho dù việc xâm phạm các nguyên tắc chia sẻ hay phối hợp xảy ra một cách có chủ ý.

Vấn đề an toàn ngày càng được quan tâm do các chương trình trong máy tính hiện đại tương tác với nhau theo nhiều cách và việc chia sẻ dữ liệu giữa các người dùng là hành vi căn bản và phổ biến với hệ thống máy tính được nối mạng và gắn kết chặt chẽ. Thách thức với việc thiết kế an toàn cho hệ điều hành là thiết kế các cơ chế an toàn để bảo vệ việc thực thi của các chương trình và dữ liệu của chúng trong môi trường phức tạp. Các cơ chế an toàn chính tắc (*formal security mechanism*) giúp chứng minh về mặt toán học hệ thống đạt được các mục tiêu an toàn song không tính đến độ phức tạp của hệ thống. Trong nhiều tình huống, độ phức tạp khiến cho việc chứng minh lý thuyết là không khả thi.

Trên thực tế, an toàn hệ điều hành được tiếp cận theo hai hướng chủ yếu. Một là hệ thống hạn chế (*constrained*): đảm bảo các mục tiêu an toàn được thỏa mãn với mức độ cao. Hai là hệ thống dùng chung (*general-purpose*): chỉ đảm bảo các mục tiêu an toàn một cách giới hạn với mức độ thấp. Các hệ thống hạn chế hỗ trợ ít ứng dụng cũng như chức năng song đảm bảo mục tiêu chủ yếu là tính an toàn. Việc hạn chế các chức năng cho phép hệ thống có thể kiểm chứng được tính đúng đắn của các mục tiêu an toàn đề ra. Mặt khác, các hệ thống dùng chung hướng tới cung cấp các chức năng mềm dẻo, thân thiện người dùng, dễ triển khai và có hiệu năng cao. Các đặc điểm này dẫn đến nhiều thách thức với việc đảm bảo an toàn cho hệ thống.

Hệ điều hành an toàn, một cách lý tưởng, là hệ điều hành cung cấp cơ chế bảo vệ đảm bảo đạt được các mục tiêu an toàn của hệ thống cho dù hệ thống phải đối mặt với

các mối đe dọa. Các cơ chế an toàn này được thiết kế trong ngữ cảnh của việc chia sẻ và điều độ tài nguyên. Các mục tiêu an toàn của hệ thống phải được duy trì bất kể cách thức sử dụng hệ thống (như bị tấn công bởi người bẻ khóa).

Thông thường hệ điều hành đạt được mức độ đảm bảo cao được coi là hệ điều hành an toàn hay gọi là hệ thống tin cậy (*trusted system*). Thực tế, không có hệ thống hiện đại phức tạp nào hoàn toàn an toàn. Những khó khăn của việc ngăn ngừa các lỗi lập trình và các thách thức để loại bỏ những lỗi như vậy dẫn đến không có hệ thống nào phức tạp như hệ điều hành có thể hoàn toàn an toàn. Dù vậy, việc xây dựng hệ điều hành an toàn vẫn rất cần thiết và giúp cho người dùng có thể xác định hay đánh giá được mức độ an toàn của hệ thống họ sử dụng hay theo đuổi.

Các yếu tố giúp xây dựng hệ điều hành an toàn bao gồm mục tiêu an toàn, các mô hình đe dọa và tin cậy, và cơ chế bảo vệ.

a. **Mục tiêu an toàn**

Mục tiêu an toàn (*security goal*) xác định các thao tác có thể được thực hiện bởi hệ thống trong khi ngăn chặn các truy nhập trái phép. Các mục tiêu này cần được định nghĩa ở mức độ khái quát cao. Các mục tiêu an toàn xác định các yêu cầu mà thiết kế hệ thống cần phải thỏa mãn và việc triển khai đúng đắn phải đáp ứng đầy đủ các yêu cầu này.

Mục tiêu an toàn mô tả các truy nhập tới các tài nguyên của hệ thống mà chúng cần thỏa mãn các yếu tố sau: bí mật, toàn vẹn, và sẵn dùng. Truy nhập hệ thống được mô tả bằng **chủ thẻ** (chương trình hay người dùng) có thể thực hiện **các thao tác** (đọc hay ghi) lên các **đối tượng** (file hay *socket*). Như vậy, các thuộc tính an toàn nêu trên được hiểu như sau:

- Tính bí mật giới hạn các đối tượng có thể được truy nhập
- Tính toàn vẹn hạn chế các đối tượng mà chủ thẻ có thể ghi để đảm bảo thao tác được đúng đắn trong quan hệ với các thao tác của các chủ thẻ khác
- Tính sẵn dùng hạn chế các tài nguyên mà các chủ thẻ có thể sử dụng do các chủ thẻ có thể làm cạn kiệt tài nguyên đó

Mục tiêu an toàn có thể xây dựng dựa trên các chức năng thông qua nguyên tắc: *Đặc quyền tối thiểu*. Các chương trình được thực hiện các thao tác cần thiết cho hoạt động của chúng. Tuy nhiên, hạn chế chức năng không làm tăng tính an toàn của hệ thống mà chỉ làm giảm khả năng của việc tấn công.

b. **Mô hình tin cậy**

Mô hình tin cậy (*Trust model*) của hệ thống định nghĩa tập phần mềm và dữ liệu mà hệ thống dựa vào để đảm bảo thực hiện đúng đắn các mục tiêu an toàn của hệ thống. Với hệ điều hành, khái niệm này tương đồng với cơ sở tính toán tin cậy TCB (*trusted computing base*). Một cách lý tưởng, hệ thống tin cậy chứa số phần mềm tối thiểu để đảm bảo bắt buộc các mục tiêu an toàn. Phần mềm tin cậy bao gồm phần mềm xác định các yêu cầu an toàn của hệ thống và phần mềm thực thi các yêu cầu này. Hơn nữa, phần

mềm kích hoạt (khởi động) các phần mềm này cũng phải tin cậy. Để hệ điều hành hoạt động cần có phần mềm đăng nhập, xác thực người dùng, truy nhập tài nguyên nên các phần mềm này cần phải là phần mềm tin cậy.

Như vậy, người phát triển hệ điều hành an toàn phải chứng minh hệ thống của mình có mô hình tin cậy tồn tại. Để làm được điều này cần phải có:

- Phần mềm tin cậy phải thực hiện việc dàn xếp toàn bộ các thao tác nhạy cảm với an toàn. Nói cách khác, tất cả các thao tác nhạy cảm phải do phần mềm tin cậy đứng trung gian thực hiện.
- Chứng minh tính đúng đắn của phần mềm và dữ liệu tin cậy. Mức độ tin cậy của hệ thống thể hiện qua mức độ đánh giá (chứng minh) tính an toàn: từ một phần, kiểm thử toàn bộ hay đánh giá. Qua đó, người dùng có thể tin tưởng vào phần mềm để thực hiện công việc của mình.
- Chứng minh việc thực thi của các phần mềm không bị phá vỡ bởi các chương trình không nằm trong các phần mềm tin cậy. Nghĩa là, tính toàn vẹn của các phần mềm tin cậy phải được bảo vệ khỏi các mối đe dọa tới hệ thống. Rõ ràng, nếu phần mềm bị xâm nhập thì phần mềm đó không được tin cậy.

c. **Mô hình đe dọa**

Mô hình đe dọa (*Threat model*) xây dựng tập các thao tác mà người tấn công có thể dùng để vô hiệu hóa hệ thống. Trong mô hình này, giả định người tấn công chuyên nghiệp có khả năng chèn dữ liệu vào mạng và có thể kiểm soát một phần các phần mềm đang chạy của hệ thống. Như vậy, tập các thao tác này không hạn chế theo nghĩa người tấn công có thể áp dụng bất cứ thao tác có thể để xâm phạm mục tiêu an toàn của hệ thống. Nhiệm vụ của người xây dựng hệ điều hành an toàn là bảo vệ các phần mềm tin cậy khỏi các dạng đe dọa trong mô hình.

Mô hình đe dọa cho thấy điểm yếu cơ bản của các hệ điều hành thương mại như Windows. Bởi, các hệ điều hành này giả định các phần mềm hoạt động nhân danh chủ thẻ thì được tin tưởng bởi chủ thẻ đó. Nói cách khác, người dùng tin tưởng vào các phần mềm mà họ sử dụng. Tuy nhiên, theo mô hình đe dọa thì các phần mềm này có thể bị kiểm soát bởi người tấn công. Vậy, hệ điều hành an toàn không thể tin tưởng các phần mềm nằm ngoài phạm vi an toàn (không tin cậy). Chương trình người dùng có thể không tin cậy song hệ thống có thể hạn chế việc truy nhập tới dữ liệu nhạy cảm nhằm hạn chế rò rỉ hay sửa đổi các thông tin này.

Nhiệm vụ của hệ điều hành an toàn là bảo vệ các phần mềm tin cậy khỏi các mối đe dọa được nhận biết từ mô hình nêu trên. Mục tiêu an toàn cần được đảm bảo bất kể hành vi hay hoạt động của các chương trình người dùng. Việc bảo vệ phần mềm tin cậy thường khó khăn hơn việc hạn chế rủi ro của các phần mềm người dùng do phải tương tác với nhiều tiến trình không tin cậy. Người phát triển hệ thống phải nhận biết được các

mối đe dọa, đánh giá ảnh hưởng của chúng lên an toàn hệ thống, và cung cấp biện pháp chống trả hiệu quả những đe dọa này.

d. Cơ chế bảo vệ

Hệ điều hành cung cấp cơ chế thực thi truy nhập tới các tài nguyên chung của hệ thống để người dùng thực hiện các hoạt động của mình thông qua phần mềm. Cơ chế thực thi truy nhập cho phép các yêu cầu (lời gọi hệ thống-*system call*) từ các chủ thể (*subject*) khác nhau như người dùng, tiến trình để thực hiện các thao tác (đọc ghi,...) lên các đối tượng (*object*) là tài nguyên của hệ thống. Đây là các khái niệm căn bản của hệ thống bảo vệ nhằm đảm bảo an toàn cho hệ điều hành. Điều cần chú ý là các truy nhập hệ thống hạn chế với các chủ thể và đối tượng bên trong hệ thống không phải từ bên ngoài.

Các dữ liệu cơ bản của hệ thống bảo vệ gồm có:

- Trạng thái bảo vệ mô tả các thao tác mà các chủ thể của hệ thống có thể thực hiện lên các đối tượng hệ thống
- Tập các thao tác lên trạng thái bảo vệ làm thay đổi các trạng thái này
- Hệ thống bảo vệ xác định các yêu cầu an ninh của hệ điều hành và thực hiện việc quản lý các yêu cầu này.

Một trong những biện pháp biểu diễn các trạng thái bảo vệ là sử dụng ma trận truy nhập như dưới đây.

Ma trận truy nhập

Các trạng thái bảo vệ của hệ thống được biểu diễn bằng ma trận truy nhập được định nghĩa bằng

- Tập các chủ thể S
- Tập các đối tượng O
- Các thao tác được phép của chủ thể lên đối tượng Op

| | File 1 | File 2 | File 3 | Process 1 | Process 2 |
|-----------|--------|-------------|-------------|-----------|-----------|
| Process 1 | Read | Read, Write | Read, Write | Read | - |
| Process 2 | - | Read | Read, Write | - | Read |

Hình 1-2. Ma trận truy nhập với hai tiến trình

Bên cạnh các thao tác cơ bản như đọc (*read*), ghi (*write*), ma trận cũng mô tả các thao tác mà chủ thể có thể thực hiện lên trên các ô của ma trận, chính là quyền sở hữu (*own*). Người có quyền sở hữu có thể thay đổi trạng thái bảo vệ (nội dung) của các ô tương ứng với đối tượng. Vì vậy, biện pháp bảo vệ này còn được gọi là tùy chọn.

Ma trận truy nhập cũng được sử dụng để mô tả miền bảo vệ (*protection domain*). *Miền bảo vệ* là tập các đối tượng (tài nguyên) mà tiến trình có thể truy nhập và các thao tác mà tiến trình có thể dùng để truy nhập tới các đối tượng như vậy. Việc duyệt qua nội dung của hàng trong ma trận truy nhập cho biết thông tin về miền hoạt động của tiến

trình như trong hình trên. Với hệ điều hành an toàn, cần đảm bảo miền an toàn của mỗi tiến trình thỏa mãn các mục tiêu an toàn như tính bí mật hay toàn vẹn.

Trên thực tế ma trận truy nhập có cấu trúc thưa, nghĩa là nhiều ô không có nội dung. Người ta có thể sử dụng các cách biểu diễn khác như sử dụng các cột thì ta có danh sách kiểm soát truy nhập ACL (*Access control list*). Hay sử dụng các hàng của ma trận thì ta có danh sách năng lực C-List (*Capability list*). ACL cho phép người quản trị hình dung được chủ thể nào có quyền gì với các đối tượng còn C-List giúp nhanh chóng xác định miền bảo vệ của tiến trình.

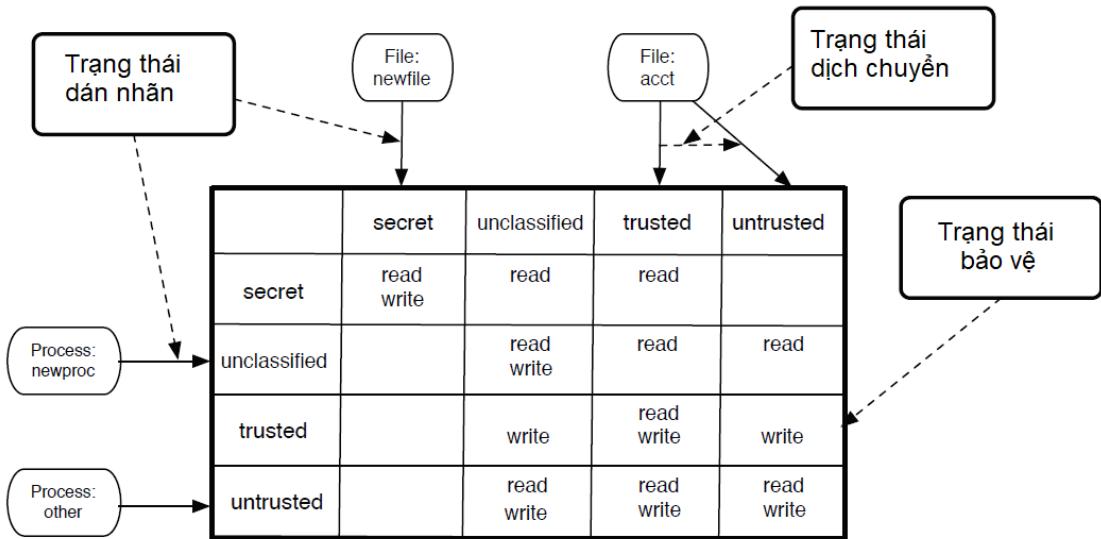
Mô hình bảo vệ sử dụng ma trận truy nhập làm nảy sinh vấn đề với an toàn hệ điều hành. Đó là các tiến trình không tin cậy có thể xâm nhập hệ thống bảo vệ. Lợi dụng các thao tác lên trạng thái bảo vệ, tiến trình người dùng không tin cậy có thể sửa đổi ma trận truy nhập bằng cách thêm chủ thể, đối tượng mới, hay các thao tác cho các ô của ma trận. Bài toán đảm bảo với trạng thái bảo vệ cụ thể và với toàn bộ các trạng thái bảo vệ có thể phái sinh từ trạng thái ban đầu không cho phép truy nhập trái phép xảy ra gọi là bài toán an toàn (*safety problem*). Người ta cho thấy rằng với các thao tác phức hợp như tạo file, thay đổi nội dung của cột và ô trạng thái người dùng, bài toán an toàn là không xác định được (*undecidable*). Như vậy, nói chung không thể kiểm chứng được trạng thái an toàn của hệ thống, tức là trạng thái xuất hiện các truy nhập không mong muốn với người sở hữu hay hệ thống. Khi đó, hệ thống cần triển khai biện pháp bảo vệ bắt buộc.

Hệ thống bảo vệ bắt buộc

Như đã nêu trên, ma trận truy nhập không thể bảo vệ mục tiêu bí mật và toàn vẹn nhất là khi các chương trình người dùng không tin cậy. Để đảm bảo mục tiêu của hệ thống, các trạng thái bảo vệ cần được định nghĩa dựa trên việc xác định chính xác mục tiêu bí mật và toàn vẹn của dữ liệu và chương trình người dùng.Thêm vào đó không cho phép bất cứ chương trình không tin cậy nào được phép thực hiện các thao tác sửa đổi thông tin bảo vệ. Các yêu cầu này làm xuất hiện hệ thống bảo vệ bắt buộc.

Hệ thống bảo vệ bắt buộc là hệ thống mà chỉ có thể được sửa đổi bởi người quản trị tin cậy thông qua phần mềm tin cậy gồm các biểu diễn trạng thái như sau:

- *Trạng thái bảo vệ bắt buộc* là trạng thái mà các chủ thể và các đối tượng được biểu diễn bằng các *nhãn*. Các trạng thái mô tả các thao tác mà các nhãn chủ thể có thể thực hiện lên các nhãn đối tượng.
- *Trạng thái dán nhãn* để ánh xạ các tiến trình và các đối tượng tài nguyên hệ thống tới các nhãn
- *Trạng thái dịch chuyển* mô tả cách thức hợp lệ mà các tiến trình và các đối tượng có thể được dán nhãn lại (thay đổi nhãn).



Hình 1-3. Hệ thống bảo vệ bắt buộc

Trong hệ điều hành an toàn, nhãn chính là các định danh khái quát. Việc gán quyền cho nhãn xác định ngữ nghĩa (semantics) an toàn của chúng. Các nhãn này chống lại việc xâm nhập (temper-proof) nhờ:

- Tập các nhãn này được xây dựng bởi người quản trị tin cậy bằng phần mềm tin cậy
- Tập các nhãn không thay đổi được bởi các tiến trình không tin cậy của người dùng

Người quản trị tin cậy xây dựng các nhãn của ma trận truy nhập và xác lập các thao tác mà chủ thể với nhãn nhất định được phép thực hiện lên đối tượng với nhãn cho trước. Hệ thống này cho phép miễn nhiễm với các tiến trình không tin cậy. Điều này là vì tập các nhãn không thể thay đổi qua việc thực thi của các tiến trình người dùng, có thể chứng minh được các mục tiêu an toàn được thực thi qua ma trận và trong suốt quá trình hoạt động của hệ thống.

Hệ điều hành an toàn cần có khả năng gán các nhãn cho các chủ thể (tiến trình) và đối tượng được tạo ra trong quá trình hoạt động và thậm chí cho phép thay đổi nhãn. Trạng thái dán nhãn chỉ là quá trình gán các nhãn cho chủ thể và đối tượng mới. Hình 1-3 cho thấy các tiến trình và các file được liên kết với các nhãn trong một tinh huống cố định. Khi newfile được tạo ra cần gán cho nó một nhãn trong trạng thái an toàn cụ thể là nhãn secret. Tương tự tiến trình newproc được gán nhãn unclassified. Như vậy, ma trận truy nhập cho thấy tiến trình mới này không có quyền gì với file mới được tạo ra.

Trạng thái chuyển dịch cho phép hệ điều hành an toàn thay đổi nhãn của tiến trình (chủ thể) hay tài nguyên hệ thống (đối tượng). Với tiến trình, việc này làm thay đổi miền bảo vệ hay các tài nguyên được phép sử dụng. Việc này cần thiết khi xét đến khả năng một tiến trình kích hoạt chương trình khác chạy. Như vậy, nhãn gắn với tiến trình cần thay đổi thể hiện các yêu cầu truy nhập hay tin cậy trong môi trường (miền) mới. Với hệ

điều hành an toàn, trạng thái chuyên dịch phải được xác định bởi người quản trị tin cậy và không bị thay đổi trong quá trình thực thi hệ thống.

1.2 Các vấn đề về kiến trúc an toàn

Việc xây dựng hệ thống máy tính cần tìm kiếm sự cân bằng giữa một loạt các yêu cầu như năng lực, tính mềm dẻo, hiệu năng, chi phí. Trong khi không có xung đột hiển nhiên giữa các yêu cầu này thì các tính năng liên quan tới an toàn thường xung đột với nhau và cần phải thỏa hiệp trong khi thiết kế hệ thống. An toàn đơn giản là một dạng yêu cầu khác và nếu có xung đột các tính năng an toàn phải cân đối với các tính năng khác tùy theo mức độ quan trọng với hệ thống. Kiến trúc an toàn (*security architecture*) đóng vai trò thiết yếu trong quá trình phân tích thiết kế hệ thống.

Kiến trúc an toàn là mô tả chi tiết toàn bộ các khía cạnh của hệ thống liên quan đến vấn đề an toàn cùng với các nguyên tắc thiết kế. Kiến trúc an toàn tốt giống như thiết kế tổng thể mô tả ở mức khái quát quan hệ giữa các bộ phận then chốt theo cách mà chúng phải thỏa mãn các yêu cầu về an toàn. Mặt khác, kiến trúc này cần mô tả các chi tiết của quá trình xây dựng hệ thống mà qua đó các yêu cầu an toàn được đảm bảo. Kiến trúc an toàn cho phép mô tả và xác định các mục tiêu an toàn mà hệ thống cần phải đạt được. Việc này không chỉ giúp trong những giai đoạn đầu thiết kế hệ thống mà cả sau này khi vận hành.

Tại thời điểm bắt đầu xây dựng hệ thống, kiến trúc an toàn có thể được mô tả bằng các vấn đề an toàn ở mức cao: chính sách an toàn, mức độ đảm bảo mong muốn, tác động của an toàn lên quá trình xây dựng hệ thống, và các nguyên tắc hướng dẫn chung. Sau này, khi được định hình rõ ràng, kiến trúc an toàn cần phản ánh cấu trúc của hệ thống và mức độ chi tiết tăng dần theo các bước thiết kế. Tuy vậy, kiến trúc cần đi trước một bước để định hướng cho việc hoàn thành công việc thiết kế. Kiến trúc an toàn đóng vai trò quan trọng trong khi phát triển và tác động tới từng cá nhân riêng lẻ thông qua các hướng dẫn thể hiện qua các tiêu chuẩn lập trình, đánh giá mã và kiểm thử.

Các vấn đề cần xây dựng kiến trúc an toàn được xem xét như dưới đây.

1.2.1 Xem xét vấn đề an toàn ngay từ đầu

Ngay từ đầu, người thiết kế hệ thống cần coi trọng vấn đề an toàn ngang bằng như các tính năng vận hành của hệ thống và phải được tích hợp đầy đủ vào hệ thống. Nhiều trường hợp, người thiết kế và phát triển sử dụng chính sách “xây dựng trước, an toàn sau”. Như vậy vấn đề an toàn không được tích hợp tốt vào hệ thống. Việc thiếu quan tâm đến vấn đề an toàn sẽ dễ dẫn đến việc không kiểm soát được các phi tần đê bổ sung các tính năng an toàn.

1.2.2 Lường trước các yêu cầu về an toàn

Kiến trúc an toàn cần có tầm nhìn xa để cập tới các tính năng an toàn tiềm năng thậm chí chưa có kế hoạch sử dụng ngay lập tức. Việc này làm tăng chi phí một chút cho việc nâng cao tính an toàn. Điểm then chốt cho việc gắn kết hợp lý các tính năng an toàn tương lai là việc hiểu rõ các yêu cầu về an toàn của hệ thống máy tính nói chung. Hơn thế cần phải mô tả một cách tường minh nhất các yêu cầu trong tương lai này trong kiến trúc an toàn. Thực tế cho thấy rằng an toàn của nhiều hệ thống máy tính không thể được cải thiện do các chức năng của hệ thống được xây dựng dựa trên các đặc điểm không an toàn. Khi các đặc điểm này thay đổi, hệ thống sẽ không hoạt động như kỳ vọng. Trong nhiều trường hợp, việc khắc phục các lỗi hỏng này có thể làm hỏng các phần mềm đang chạy tốt. Lường trước các yêu cầu an toàn không chỉ ảnh hưởng đến mức độ cần thiết làm hệ thống an toàn hơn trong tương lai mà còn giúp xác định liệu tính an toàn của hệ thống có thể được nâng cao hay không.

Vấn đề khác là chính sách an toàn. Thay đổi trong chính sách an toàn có thể dẫn đến hậu quả tai hại với các ứng dụng đang hoạt động tốt mà nay xung đột với chính sách mới. Mặt khác quan tâm tới các chính sách an toàn ngay khi xây dựng ứng dụng, ngay cả khi các chính sách chưa thực thi, làm cho việc thay đổi nếu có được trơn tru và trong suốt.

1.2.3 Giảm thiểu và cách ly các biện pháp an toàn

Để đạt được độ tin cậy cao về an toàn của hệ thống, người thiết kế cần giảm thiểu kích cỡ và độ phức tạp của các phần liên quan tới an toàn của thiết kế. Lý do chính khiến hệ điều hành không an toàn là kích cỡ quá lớn của chúng làm cho khó nắm bắt tổng thể hệ thống và hệ điều hành khó có thể hoàn toàn sach lõi và khó tin cậy. Vì vậy, ngay cả với hệ thống phức tạp, cần giữ phần cốt lõi (liên quan đến an toàn) nhỏ và định nghĩa rõ ràng.

Điểm then chốt để giảm thiểu các bộ phận liên quan tới an toàn của hệ điều hành là chỉ dùng số ít các cơ chế thực thi an toàn. Như vậy, bắt buộc các hành động liên quan tới an toàn được giữ trong một số ít phần cách ly. Thực tế với hệ điều hành rất khó đặt được điều này. Vấn đề an toàn liên quan tới rất nhiều chức năng khác nhau của hệ thống như quản lý file hệ thống, quản lý bộ nhớ ...

Khi các cơ chế an toàn đơn giản, dễ nhận biết và cách ly thì dễ dàng triển khai các cơ chế bảo vệ bổ sung để tránh các thiệt hại do lỗi tại các phần khác của hệ thống. Các đoạn mã liên quan đến an toàn có thể bảo vệ chống ghi để không bị sửa đổi.

1.2.4 Thực hiện quyền tối thiểu

Gắn với khái niệm cách ly các cơ chế an toàn là nguyên tắc quyền tối thiểu. Các chủ thể (người dùng hay chương trình) cần được cấp quyền không hơn mức cần thiết để thực hiện công việc. Như vậy, thiệt hại do lỗi hay phần mềm xấu được hạn chế. Quyền tối thiểu được biểu diễn trong nhiều khía cạnh của hệ thống.

Thông thường, quyền thể hiện ở các cơ chế phần cứng hạn chế việc sử dụng các câu lệnh đặc biệt (lệnh vào/ra) và truy nhập tới các vùng ô nhớ. Ngoài ra, quyền còn thể hiện ở các cơ chế phần mềm, như trong hệ điều hành, cho phép chương trình người dùng qua các biện pháp kiểm soát truy nhập hay thực thi các chức năng hệ thống.

Quyền tối thiểu còn thể hiện trong nguyên tắc phát triển hệ thống. Như bằng việc đặt ra các tiêu chuẩn lập trình hạn chế các truy nhập tới các dữ liệu toàn cục (*global data*), như vậy giảm khả năng lỗi từ vùng này tác động tới vùng khác. Việc quản trị người dùng và hệ thống là một khía cạnh khác của quyền tối thiểu. Người dùng và người quản trị không nên được cấp truy nhập nhiều hơn với công việc của họ.

1.2.5 Giữ các tính năng an toàn thân thiện

Các cơ chế an toàn không được ảnh hưởng tới người dùng tuân thủ quy định. Cơ chế an toàn phải trong suốt với người dùng bình thường. Việc can thiệp vào công việc hàng ngày làm giảm năng suất và khiến người dùng tìm cách bỏ qua các cơ chế an toàn.

Các cơ chế an toàn cần thuận tiện cho người dùng để cấp quyền truy nhập. Người dùng cần được đảm bảo cung cấp đủ truy nhập khi cần thiết và tránh các thủ tục rườm rà và phức tạp.

Để thuận tiện cho người dùng để hạn chế truy nhập có thể thực hiện cách xây dựng các chế độ mặc định phù hợp. Việc này hạn chế các rủi ro do vô tình khi quản lý và cấp phát các quyền truy nhập cho người dùng.

1.2.6 An toàn không dựa trên bí mật

Ngoại trừ việc quản lý mật khẩu, đính chính của kiến trúc an toàn tránh phụ thuộc vào tính bí mật để đảm bảo an toàn. Việc giả định người dùng không bẻ khóa hệ thống vì không biết mã nguồn hay tài liệu về hệ thống không an toàn chút nào.

Việc công khai mã nguồn hệ thống có khả năng cải thiện tính an toàn nhờ có khối lượng người dùng lớn hơn giúp phát hiện và sửa chữa các khiếm khuyết.

1.3 Chính sách an toàn

Từ góc độ quản lý, công việc quản trị cần mô tả và xây dựng các yêu cầu và các hành động cụ thể để đảm bảo sự tuân thủ chính sách, quy trình, tiêu chuẩn và hướng dẫn của cơ quan/tổ chức. Việc quản trị tốt mang lại sự tin tưởng và tự tin về các thành viên tuân theo các qui định đã được xây dựng. Về cơ bản, chính sách an toàn mô tả các kiểm soát, hành động, và quy trình cần được thực hiện cho hệ thống thông tin. Các chính sách cần đề cập và xử lý các mối đe dọa tới hệ thống bao gồm cả con người, thông tin và tài sản cụ thể. Hệ điều hành là hệ thống cho phép người dùng thông qua các chương trình máy tính truy nhập vào các tài nguyên của máy tính cũng như là các tài nguyên (thông tin) nhạy cảm với an toàn và thực thi các thao tác trên các tài nguyên này. Việc truy nhập cũng như

xử lý này chịu sự ràng buộc và hạn chế hiện trong các chính sách của cơ quan/tổ chức và do người quản trị thực thi qua các công cụ quản trị của hệ thống.

Sự thành công của các biện pháp bảo vệ tài nguyên của hệ thống tùy thuộc vào các chính sách được xây dựng và thái độ quản lý tới việc đảm bảo an toàn thông tin. Những người xây dựng chính sách xác lập tinh thần và tầm quan trọng của vấn đề an toàn thông tin của hệ thống nhằm để: giảm thiểu rủi ro; phù hợp với các quy định và luật pháp; đảm bảo việc hoạt động liên tục, tính bí mật và toàn vẹn thông tin. Mặt khác, các chính sách cần phải được hỗ trợ và quản trị một cách thích đáng. Các chuyên gia an toàn thông tin duy trì an ninh thông qua thực thi các chính sách hay thực hiện các hướng dẫn về các hành vi được và không được chấp nhận tại nơi làm việc một cách bắt buộc với các thành viên.

Chính sách chung dùng để mô tả và xây dựng định hướng và tầm nhìn chung. Nói cách khác, chính sách này xác lập mong muốn về việc bảo vệ các tài sản thông tin. *Chính sách cho ứng dụng cụ thể* hướng tới các ứng dụng cụ thể sau khi đã định hình được các yêu cầu cũng như biện pháp đảm bảo an toàn cần thiết. Các quyết định về an ninh hay an toàn sẽ chỉ áp dụng cho các ứng dụng cụ thể như: ai là người có thẩm quyền sửa đổi dữ liệu? Việc truy nhập từ xa được kiểm soát như thế nào? Việc sử dụng các kết nối mạng ra làm sao? Như vậy, cách chính sách ở mức này liên quan trực tiếp đến các yêu cầu an toàn cho các hoạt động cụ thể của cơ quan/tổ chức.

Một cách chặt chẽ, hệ thống máy tính cần tuân thủ các thuộc tính (mục tiêu cần đạt) về an toàn và an ninh trong khi người dùng cần tuân thủ các chính sách an toàn. Việc xác định và xây dựng các thuộc tính an toàn chỉ có thể hoàn thành khi xác định được các yêu cầu cụ thể về an toàn thông qua các chính sách được mô tả. Thiếu sót trong việc xây dựng cách chính sách rõ ràng và tường minh khiến cho việc thực thi không nhất quán và làm nảy sinh các lỗ hổng khó khắc phục.

Từ góc độ vận hành, các chính sách an toàn cần được chuyển hóa thành các luật trong các bộ phận thực hiện chức năng kiểm soát truy nhập tới các tài nguyên của hệ thống như hệ thống file, mạng, bộ nhớ... Mặt khác bộ phận kiểm soát cần thiết đánh giá và kiểm chứng các chính sách này có xung đột với nhau hay vi phạm các nguyên tắc an toàn chung của hệ thống hay không.

1.4 Nền tảng tính toán tin cậy

1.4.1 Khái niệm

Máy tính ngày nay sử dụng kiến trúc mở cho phép người dùng toàn quyền lựa chọn phần mềm cũng như khả năng đọc, xóa hay sửa dữ liệu lưu trữ trên máy tính. Điều này dẫn đến các vấn đề tiêu biểu sau:

- Không an toàn cho người dùng vì kiến trúc mở có nguy cơ bị lây nhiễm virus, sâu hay vô tình cài đặt phần mềm xấu

- Không an toàn cho mạng mà máy tính kết nối vào do máy tính này có thể chứa phần mềm xấu có thể đe dọa máy tính khác trong mạng
- Không an toàn cho người sản xuất phần mềm và nội dung do kiến trúc mở cho phép các chương trình, file âm nhạc ... có thể bị sao chép không giới hạn và không bị giảm chất lượng

Khái niệm tính toán tin cậy đã được trình bày khá lâu trong lĩnh vực an toàn máy tính và có ảnh hưởng tới việc thiết kế các thẻ hệ máy tính phổ thông như PC, thiết bị di động. Tính toán tin cậy đòi hỏi thiết kế lại kiến trúc hệ thống sao cho các thành phần riêng lẻ được định nghĩa một cách tường minh các đặc tính của mình. Điều này cho phép người thiết kế có thể xác định chính xác hành vi của hệ thống.

Tính toán tin cậy mô tả các sửa đổi cần thiết về phần cứng và phần mềm để cung cấp nền tảng ổn định để hệ thống máy tính có thể hoạt động trên đó. Hệ thống này có đặc tính:

- Độ đảm bảo cao về trạng thái (cấu hình, tình trạng hoạt động của phần mềm ...) của hệ thống máy tính cục bộ. Do vậy có thể xác định khả năng chấp nhận các tác động không mong muốn
- Mức độ đảm bảo cao tương đối về trạng thái của hệ thống ở xa. Điều này thể hiện độ tin cậy của việc tương tác trong hệ thống phân tán.

Việc triển khai tính toán tin cậy gặp nhiều thách thức khác nhau. Trước hết, các máy tính và phần mềm được cung cấp từ nhiều nhà sản xuất và phân phối khác nhau dẫn đến khó khăn trong việc xác định mức độ ổn định và tin cậy của hệ thống máy tính cũng như là phần mềm.Thêm vào đó, hầu hết hệ điều hành được thiết kế dựa trên ý tưởng có 1 người quản trị hệ thống chuyên nghiệp trong khi người dùng cuối thường thiếu kỹ năng và hiểu biết. Nhiều rủi ro xuất hiện do sự bất cẩn của người dùng cuối.

Các vấn đề tương tự cũng gặp phải với các hệ thống mạng. Người dùng không thể tiếp xúc trực tiếp với các hệ thống máy chủ cung cấp dịch vụ. Như vậy, người dùng phải chấp nhận giả định hệ thống máy tính tin cậy và được quản trị bởi đội ngũ phù hợp.

Nền tảng di động là ví dụ tiêu biểu với tính toán tin cậy vì có yêu cầu tin cậy chặt chẽ hơn máy tính cá nhân PC. Các thiết bị vô tuyến cần tuân thủ chặt chẽ các qui định quan trọng như không cho phép phần mềm thay đổi tùy ý các tham số hoạt động. Thời gian hoạt động cũng có thể được tính tiền và phải được gán một cách chính xác tới thuê bao. Hơn thế, thiết bị di động có thể có những cách sở hữu phức tạp như người dùng cuối có thể đăng ký sử dụng nhiều thiết bị và dịch vụ khác nhau. Các thiết bị này cũng dễ bị thất lạc và mất trộm hơn là máy tính để bàn truyền thống. Chính vì các yêu cầu khắt khe hơn và thu hút được sự quan tâm hơn từ nhà sản xuất cũng như người dùng, nền tảng tính toán tin cậy trên di động có nhiều kết quả hơn so với PC truyền thống.

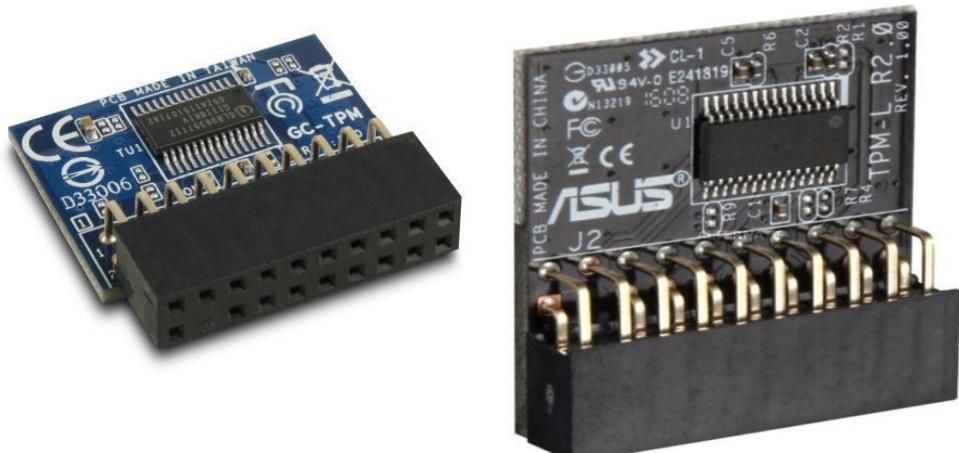
1.4.2 Mô đun hạ tầng tính toán tin cậy

a. Giới thiệu

Để có thể đảm bảo an toàn với việc sử dụng các thiết bị tính toán trên máy tính PC, về cơ bản cần:

- Các máy tính được định danh một cách chắc chắn có thể thông qua việc sử dụng khóa công khai kèm với khóa bí mật “gắn chặt” liền với hạ tầng tính toán. Như vậy cần sử dụng cơ chế phần cứng và chống xâm nhập hay giả mạo.
- Các máy tính xác định chắc chắn cấu hình và định danh chương trình nhờ sử dụng mã băm và cơ chế khác. Phần mềm, firmware, BIOS, trình nap, nhân, chương trình tham gia vào quá trình hoạt động của máy tính cần được kiểm tra thích đáng để đảm bảo mức độ tin cậy và thực thi đúng đắn chính sách an ninh mong muốn.

Mô-đun hạ tầng tin cậy TPM (*Trusted Platform Module*) được định nghĩa là các chức năng phần mềm (lõi-gic) và nhúng vào trong kiến trúc của máy tính bằng cách sử dụng chíp riêng biệt. Mô-đun này cung cấp cơ sở để đảm bảo an toàn cho các chương trình máy tính trong quá trình hoạt động cũng như tương tác với nhau thông qua việc sử dụng các cơ chế định danh và mã hóa. Các chức năng của TPM được xây dựng bằng phần mềm, song các chức năng an ninh cần được bảo vệ chặt chẽ được thực hiện thông qua thiết bị phần cứng. Hình dưới đây cho thấy các mô-đun TPM từ các nhà sản xuất khác nhau.



Hình 1-4. Mô đun TPM phần cứng

Về cơ bản, tính toán tin cậy cần thiết bị lưu trữ được bảo vệ mà thiết bị này chỉ có thể truy nhập qua các giao tiếp đặc biệt và không sử dụng các thức đọc ghi thông thường như với thanh ghi hay bộ nhớ. Cách này tương tự như việc truy nhập thông tin trên thẻ tín dụng. Việc này cho phép TPM mã hóa các thông tin bằng mã khóa chỉ tồn tại bên trong TPM và giúp bảo vệ dữ liệu được lưu trữ. Chi tiết về cách thức sử dụng mô-đun TPM

trong việc đảm bảo hoạt động an toàn cho các chương trình máy tính nói chung và hệ điều hành nói riêng sẽ được giới thiệu chi tiết trong chương sau. Phần này chỉ nhằm giới thiệu tổng quan các chức năng an toàn cơ bản mà mô-đun TPM cung cấp.

b. Các dịch vụ an toàn

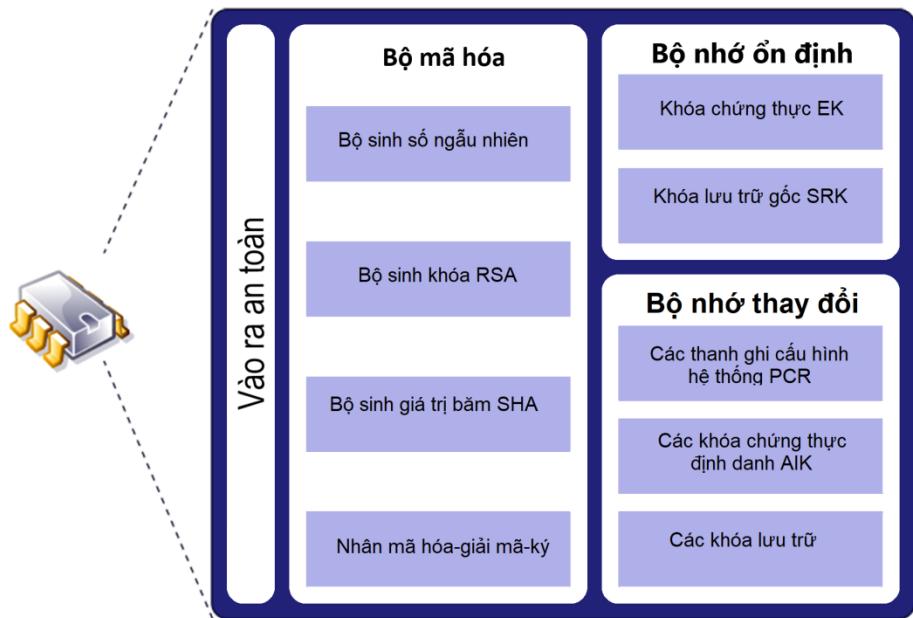
TPM khi được nhúng vào bên trong hệ thống PC nhằm cung cấp và đảm bảo các cơ sở tin cậy sau:

- Cơ sở tin cậy cho việc tính toán (*Root of Trust for Measurement- RTM*): triển khai một cách tin cậy các thuật toán băm chịu trách nhiệm cho các biện pháp bảo vệ đầu tiên với hạ tầng tính toán như tại thời điểm khởi động hay xác lập trạng thái tin cậy của hệ thống.
- Cơ sở lưu trữ tin cậy (*Root of Trust for storage - RTS*) : triển khai tin cậy vị trí được bảo vệ cho việc lưu trữ một hay nhiều khóa bí mật và một khóa lưu trữ gốc SRK (*storage root key*);
- Cơ sở tin cậy cho việc báo cáo (*Root of Trust for reporting-RTR*): triển khai tin cậy vị trí được bảo vệ để lưu khóa bí mật đại diện cho định danh duy nhất của hạ tầng, còn gọi là khóa chứng thực EK (*endorsement key*).

Các khóa SRK và EK sử dụng cách mã hóa di bô (cơ chế mã hóa dùng khóa công khai). TPM có nhiệm vụ bảo vệ khóa bí mật trong cấp khóa trên. Phần khóa công khai EK được đăng ký với nơi chứng thực. Khóa EK tồn tại không đổi trong suốt thời gian hoạt động của hạ tầng tính toán và được sử dụng cho một số hạn chế các thủ tục. Điều này là vì EK là duy nhất cho mỗi thiết bị TPM và có thể được dùng để nhận dạng thiết bị. Trong một số tình huống, người dùng có thể tự quản lý khóa EK này. Tuy nhiên khi này người dùng không thể cung cấp chứng thực cho khóa nên việc ứng dụng bị hạn chế. SRK được thiết lập khi triển khai cho người dùng và có thể được khởi tạo lại khi thay đổi người dùng thông qua cơ chế xác lập quyền sở hữu với thiết bị TPM.

Các bộ phận cơ bản cấu thành mô-đun TPM được trình bày trong Hình 1-5. Các khóa chứng thực định danh AIK (*Attestation Identity Key*) được tạo ra nhằm xác định định danh của hệ thống mà không phải cung cấp EK như là khóa dùng để ký. Việc này nhằm đảm bảo tính riêng tư của người sử dụng trong tình huống các nhà cung cấp dịch vụ yêu cầu bằng chứng về định danh người dùng.

Các thanh ghi cấu hình hệ thống PCR (*Platform Configuration Register*) được dùng để lưu lại các giá trị đo tính toàn vẹn của hệ thống. Các giá trị lưu trong các thanh ghi này có thể tính toán độ toàn vẹn của bất cứ đoạn mã nào từ BIOS tới các ứng dụng chủ yếu là trước khi đoạn mã được thực hiện. TPM có tối thiểu 16 thanh ghi kiểu này, trong đó 8 thanh ghi đầu dành riêng cho TPM.



Hình 1-5. Các bộ phận chức năng của TPM

Bộ phận RTM và RTR là các bộ phận cơ bản để thiết lập tin cậy cũng như với bên thứ ba. Nhờ vào thủ tục “niêm phong”, bộ phận RTS xác lập sự tin cậy với hệ thống cục bộ. Đồng thời RTS cũng giúp bảo vệ tính bí mật cũng như cung cấp lưu trữ được bảo mật cho các ứng dụng hay người dùng không tin cậy lẫn nhau.

Bộ phận mã hóa là phần thực thi các hàm cần thiết dùng cho việc cung cấp các dịch vụ an toàn của mô-đun TPM như tạo số ngẫu nhiên, tính toán giá trị băm, ký số,...

c. Khởi tạo TPM

Các thẻ thông minh nói chung được sở hữu và tùy chỉnh bởi bên cấp thẻ trước khi người dùng nhận được thiết bị này. Tuy vậy, nhóm đề xuất tính toán tin cậy TCG (*Trusted Computing Group*) hiểu rõ tình huống các hệ thống sử dụng TPM có thể bị kiểm soát bởi các tổ chức lớn từ xa. Vì vậy, TCG đã rất cẩn thận cung cấp cơ chế để chỉ người dùng được sở hữu và đặt cấu hình TPM. Nói cách khác, chỉ có người dùng sở hữu TPM xác định trạng thái TPM theo mong muốn của mình chứ không phải là nhà cung cấp.

Khi người dùng lấy quyền sở hữu của TPM, người ta thiết lập một bí mật dùng cho chia sẻ, thường được gọi là dữ liệu ủy quyền chủ sở hữu (*owner authorisation data*) và chèn dữ liệu này vào nơi lưu trữ an toàn trong TPM. Trên thực tế, đây chính là mật khẩu của người dùng và việc có thể chứng tỏ việc nắm giữ bí mật này cung cấp bằng chứng về quyền sở hữu. Ngay sau khi được sở hữu, TPM có thể kiểm soát việc truy nhập tới các thao tác được bảo vệ nhất định qua việc yêu cầu bằng chứng của việc sở hữu mà việc này có thể được thực hiện thông qua việc cung cấp dữ liệu ủy quyền chủ sở hữu.

Quá trình lấy quyền sở hữu yêu cầu dữ liệu ủy quyền sở hữu được bảo vệ khỏi việc nghe trộm hay bên thứ ba. Đây là tình huống chứng nhận chứng thực được sử dụng do khóa chứng thực EK là khóa duy nhất mà TPM chưa từng được sử dụng (sở hữu) có. Chính khóa EK này thiết lập kênh an toàn để trao đổi dữ liệu ủy quyền tới TPM hợp lệ.

Như vậy, trong quá trình lấy quyền sở hữu, người chủ sở hữu yêu cầu chứng nhận chứng thực và sau khi kiểm tra thông tin này xong, nhận được khóa công khai EK để mã hóa bí mật chia sẻ. Chỉ có TPM hợp lệ được xác định nhờ thông tin chứng nhận chứng thực này truy nhập vào khóa bí mật EK và như vậy chỉ có TPM được chỉ định mới có được bí mật chia sẻ này. Quá trình lấy quyền sở hữu hoàn tất bằng việc tạo ra khóa lưu trữ gốc SRK mà nó được TPM sinh ra và lưu trữ trong phần bộ nhớ ôn định (*non-volatile*). Cũng giống như khóa bí mật EK, khóa SRK không bao giờ ra khỏi TPM.

1.4.3 Các tác động

Các tiếp cận của tính toán tin cậy làm thay đổi mạnh mẽ thiết kế của hệ thống máy tính để bàn và ứng dụng phân tán. Điều mới với tính toán tin cậy là việc đảm bảo chắc chắn phần mềm chạy cục bộ hay ở xa dựa trên cơ chế mã hóa sử dụng cách thức xác thực đảm bảo. Tính toán tin cậy ngăn chặn các vụ tấn công dựa trên phần mềm nhờ vào các thao tác thiết yếu cho hoạt động phải có sự chứng thực từ phần cứng TPM.

Tính toán tin cậy chịu nhiều chỉ trích không chỉ từ cộng đồng mã nguồn mở từ những điểm tiêu biểu như sau:

- Tính riêng tư: Không bảo vệ định danh người dùng với một số giao dịch
- Kiểm soát của bên bán hàng: Bên bán hàng có thể sử dụng TPM khiến cho việc lựa chọn và thay đổi sản phẩm khó khăn hơn với người dùng cuối
- Chứng thực: Việc chứng thực sử dụng chữ ký khó khăn do số lượng phần mềm lớn, nhất là khi các phần mềm này có thể được cập nhật định kỳ, vì vậy việc chứng thực cần thực hiện trên cơ sở hành vi của chương trình.
- Không hỗ trợ mã khóa đối xứng. Thiết kế của TPM dù dùng rất nhiều phần của bộ xử lý mã hóa song không cung cấp chức năng mã hóa đối xứng nào. Điều này ảnh hưởng tới vấn đề như thực thi luật pháp.
- Thực thi luật pháp: việc mã hóa mạnh tác động cả hai bên người dùng hợp lệ và người bẻ khóa. Được dùng một cách đúng đắn, TPM giúp bảo vệ dữ liệu bí mật của người dùng. TPM mô tả một cách rõ ràng không có cửa hậu trong thiết bị hợp chuẩn. Như vậy, không có cách thức hợp lệ nào khác ngoài thông tin của chủ sở hữu để truy nhập vào các thông tin được bảo vệ. Ngược lại, khi bị lạm dụng sẽ rất khó khăn cho các cơ quan thực thi luật pháp để khai thác các thông tin bí mật này.

1.5 Nhân an toàn

Về mặt kỹ thuật, hệ điều hành an toàn cần có nền tảng nhỏ và có thể kiểm chứng được để xây dựng an toàn của hệ thống. Phần cơ sở nền tảng này còn được gọi là nhân an toàn (security kernel). Nhân an toàn được xác định bằng phần cứng và phần mềm cần thiết để thực thi các chính sách của hệ thống. Các quyết định truy nhập được mô tả bởi các chính sách dựa trên các thông tin trong cơ sở dữ liệu về kiểm soát truy nhập. Cơ sở

dữ liệu này thể hiện trạng thái an toàn của hệ thống và chứa các thông tin như quyền truy nhập và các thuộc tính an ninh. Cơ sở dữ liệu này động và thay đổi do các chủ thẻ và đối tượng được tạo và xóa cũng như quyền truy nhập của chúng được sửa đổi. Yêu cầu tối quan trọng là giám sát việc tham chiếu của từng thao tác từ chủ thẻ tới đối tượng.

1.5.1 Các yêu cầu an toàn

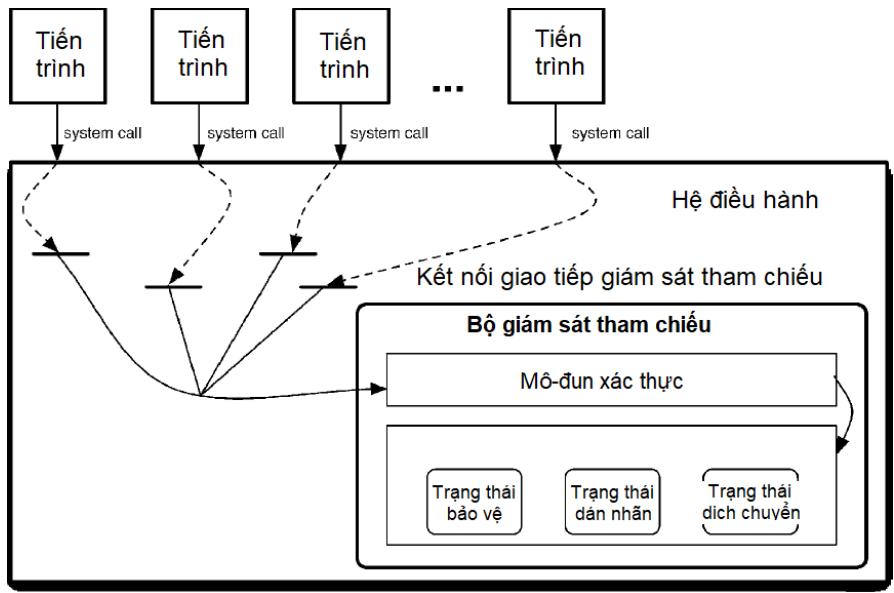
Hệ điều hành an toàn là hệ điều hành mà việc thực thi truy nhập thỏa mãn các yêu cầu của giám sát tham chiếu. Bộ giám sát tham chiếu xác định các thuộc tính cần và đủ của bất kỳ hệ thống nào để thực thi hệ thống bảo vệ một cách an toàn. Các thuộc tính của bộ giám sát tham chiếu đảm bảo yêu cầu an toàn:

- *Ngăn chặn hoàn toàn (complete mediation)*: hệ thống đảm bảo cơ chế thực thi truy nhập ngăn chặn toàn bộ các thao tác nhạy cảm với an ninh
- *Chống xâm nhập (tamper-proof)*: hệ thống đảm bảo có chế độ thực thi truy nhập, kể cả hệ thống bảo vệ, không thể bị sửa đổi bởi các tiến trình (chương trình) không tin cậy
- *Thảm tra được (verifiable)*: Cơ chế thực thi truy nhập, kể cả hệ thống bảo vệ, phải đủ nhỏ để có thể kiểm tra và phân tích, tính đúng đắn của nó có thể được đảm bảo. Nói cách khác, phải có khả năng chứng minh hệ thống thực thi các mục tiêu an toàn một cách đúng đắn

1.5.2 Giám sát tham chiếu

Giám sát tham chiếu (*Reference Monitor*) là cơ chế thực thi truy nhập cổ điển. Khi có yêu cầu truy nhập, bộ phận giám sát trả lời chấp nhận hay từ chối truy nhập dựa vào chính sách truy nhập bên trong bộ giám sát. Giám sát tham chiếu bao gồm 3 phần: *Giao tiếp* xác định vị trí mà mô-đun xác thực được gọi; *Mô-đun xác thực* xác định các truy vấn chính xác cần được gửi tới kho chính sách; *Kho chính sách* trả lời các truy vấn dựa trên hệ thống bảo vệ mà nó duy trì. Các bộ phận của bộ giám sát tham chiếu như trong hình dưới đây (Hình 1-6).

Giao tiếp xác định nơi mà các truy vấn/yêu cầu hệ thống bảo vệ được thực hiện tới bộ giám sát. Về cơ bản tất cả các thao tác nhạy cảm về an ninh được xác thực bởi cơ chế thực thi truy nhập. Các thao tác nhạy cảm là các thao tác thực hiện trên một đối tượng cụ thể (file, socket...) mà các thao tác này có thể xâm phạm các yêu cầu an ninh của hệ thống.



Hình 1-6. Tương tác giữa tiến trình và bộ giám sát tham chiếu

Mô-đun xác thực là bộ phận cốt lõi của bộ phận giám sát tham chiếu. Bộ phận này nhận các tham số đầu vào từ giao tiếp như định danh tiến trình, tham chiếu đối tượng, tên lời gọi hệ thống ... và thực hiện truy vấn kho chính sách để trả lời về tính hợp lệ của truy vấn từ giao tiếp. Thách thức của mô-đun này là ánh xạ các định danh của tiến trình thành các nhãn chủ thể, các tham chiếu đối tượng thành các nhãn đối tượng và hoạt động cụ thể được chấp thuận. Ví dụ như với yêu cầu mở file open. Mô-đun này cần có nhãn chủ thể sinh yêu cầu, nhãn của đối tượng thư mục và trạng thái bảo vệ của yêu cầu (đọc hay ghi). Trong một vài trường hợp mô-đun xác thực có thể thực thi các yêu cầu phụ tới kho chính sách về việc dán nhãn như trong tình huống tạo mới đối tượng (như file) hay di chuyển các nhãn.

Kho chính sách chính là cơ sở dữ liệu về trạng thái bảo vệ, các nhãn trạng thái và trạng thái dịch chuyển. Các câu truy vấn có cấu trúc {nhãn chủ thể, nhãn đối tượng, tập thao tác} và trả về kết quả nhị phân (*hợp lệ/không hợp lệ*). Các truy vấn về việc dịch chuyển có dạng {nhãn chủ thể, nhãn đối tượng, tập thao tác, tài nguyên}. Các tài nguyên có thể là các thực thể hoạt động như bộ xử lý hay bộ nhớ động như file.

1.5.3 Xây dựng nhân an toàn

Nhân an toàn là cách tiếp cận dựa trên giám sát tham chiếu có kết hợp phần cứng và phần mềm để đảm bảo thực thi các chính sách an toàn của hệ thống. Giám sát tham chiếu đảm bảo việc giám sát mỗi truy nhập từ các chủ thể khác nhau của hệ thống tới từng tài nguyên/đối tượng. Mục tiêu chính của hầu hết các nhân an toàn là kiểm chứng việc triển khai ở mức độ mã nguồn thỏa mãn các yêu cầu của nhân an toàn. Việc này dẫn đến ứng dụng các phương pháp chính tắc hay bán chính tắc để kiểm chứng.

Cơ sở của nhân an toàn, về mặt lý thuyết, là trong hệ điều hành lớn một phần rất nhỏ chịu trách nhiệm về an toàn. Bằng cách cấu trúc lại hệ thống sao cho toàn bộ phần mềm

liên quan đến an toàn được tách ra thành phần nhân tin cây của hệ điều hành thì hệ điều hành hầu như không chịu trách nhiệm việc thực thi an toàn. Nói cách khác, phần nhân an toàn khi này sẽ chịu trách nhiệm giám sát và thực thi các chính sách an toàn lên các hoạt động của hệ điều hành cũng như người dùng. Phần nhân phải được bảo vệ một cách thích đáng (chống xâm nhập) và không thể bỏ qua các kiểm tra truy nhập của nhân. Phần nhân cần phải nhỏ nhất có thể để có thể xác minh tính đúng đắn của nó một cách dễ dàng.

Trong mọi khía cạnh, nhân an toàn giống hệ điều hành sơ khai. Nhân an toàn thực hiện các dịch vụ phục vụ hệ điều hành cũng như hệ điều hành thực thi các dịch vụ phục vụ các ứng dụng. Và ngay khi hệ điều hành thiết lập các hạn chế lên ứng dụng, nhân an toàn đặt ra các hạn chế với hệ điều hành. Trong khi hệ điều hành không đóng vai trò gì trong việc thực thi các chính sách an toàn được triển khai bởi nhân, hệ điều hành cần giữ cho hệ thống hoạt động và ngăn chặn việc từ chối phục vụ do ứng dụng lỗi hay có mục đích xấu. Không lỗi nào trong ứng dụng cũng như trong hệ điều hành xâm phạm đến chính sách an toàn của nhân.

1.6 Câu hỏi ôn tập

- 1) Các yêu cầu cơ bản về an toàn đối với hệ điều hành?
- 2) Khái niệm về mục tiêu an toàn của hệ điều hành?
- 3) Khái niệm về mô hình tin cậy?
- 4) Khái niệm về mô hình đe dọa?
- 5) Khái niệm về hệ thống bảo vệ
- 6) Trình bày về ma trận truy nhập. Cho ví dụ?
- 7) Trình bày về hệ thống bảo vệ bắt buộc. Cho ví dụ?
- 8) Nêu các chức năng cơ bản của bộ giám sát tham chiếu?
- 9) Các khái niệm cơ bản với nhân an toàn?
- 10) Giải thích các thuộc tính của bộ giám sát truy nhập?
- 11) Nêu và giải thích các nguyên tắc của kiến trúc an toàn?
- 12) Nêu khái niệm về tính toán tin cậy?
- 13) Các yêu cầu với nền tảng tính toán tin cậy?
- 14) Trình bày các chức năng cơ bản của mô-đun hạ tầng tin cậy TPM?

CHƯƠNG 2. CÁC CƠ CHẾ AN TOÀN PHẦN CỨNG

Để hệ điều hành hoạt động ổn định và hiệu quả, hệ điều hành cần phân biệt được các hoạt động của riêng bản thân và các hoạt động của chương trình người dùng. Việc này phức tạp do bản thân hệ điều hành cũng là một chương trình. Hệ điều hành phải theo dõi toàn bộ các hoạt động và đảm bảo các hoạt động này không vi phạm chính sách an toàn. Như vậy, hệ điều hành sử dụng một số cơ chế bảo vệ với sự hỗ trợ từ phần cứng để thực thi các cơ chế bảo vệ này một cách hiệu quả. Nhiều cơ chế an ninh từng được xây dựng bằng phần mềm thì nay được phát triển bằng phần cứng. Một số tính năng vẫn nằm trong phần mềm do độ phức tạp của các tính năng này hay do người thiết kế không muốn lè thuộc vào các triển khai cụ thể. Thực sự, chúng ta không cần phần cứng tinh vi để xây dựng hệ điều hành an toàn. Các tính năng phần cứng được ưa thích hơn phần mềm là vì:

- Các tính năng an toàn này được chứng tỏ tin cậy hơn và được triển khai một cách đúng đắn. Điều này đúng chỉ là vì các chức năng thực hiện tại phần cứng đơn giản hơn và ít bị lỗi ngẫu nhiên so với phần mềm. Điều quan trọng là phần cứng sẽ không bị tác động bởi phần mềm khác.
- Đặt các chức năng an ninh vào phần cứng tạo nên kiến trúc trong sáng hơn.
- Phần cứng cho phép hiệu năng cao hơn, thực thi nhanh hơn so với phần mềm.

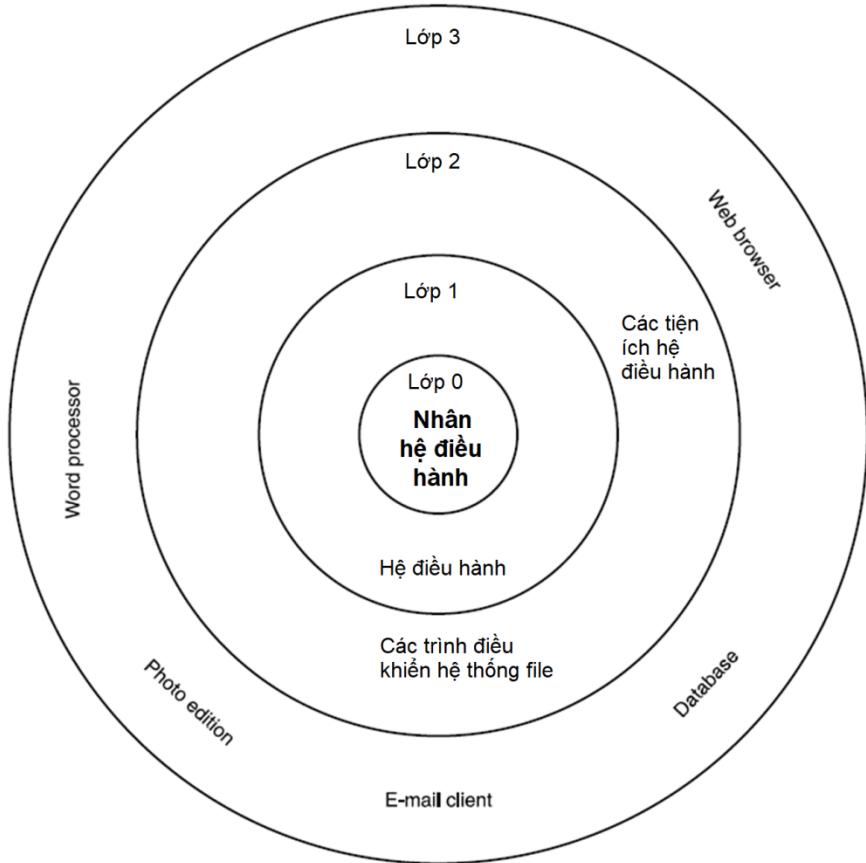
Phần dưới đây giới thiệu các chức năng cơ bản cần thiết từ phần cứng để hỗ trợ cho hệ điều hành trong các vấn đề về quản lý tiến trình, bộ nhớ và vào/rã.

2.1 Hỗ trợ các tiến trình

Một yêu cầu cơ bản của hệ điều hành an toàn là cách ly (chương trình) người dùng với nhau trong khi chấp nhận việc trao đổi thông tin qua các kênh được kiểm soát. Với các hệ điều hành hiện nay, các chương trình người dùng chạy song song với nhau nên việc chuyển đổi ngữ cảnh của các chương trình cần được thực hiện một cách nhanh chóng và hiệu quả. Phần cứng giúp làm đơn giản hóa việc này thông qua cơ chế “khởi động” lại cách chương trình bị dừng bằng cách lưu lại trạng thái của các tiến trình (chương trình đang chạy) và khôi phục đồng thời các thanh ghi mà chương trình sử dụng. Tuy nhiên, các bộ xử lý thế hệ đầu yêu cầu chương trình hệ thống lưu và khởi tạo từng thanh ghi một.

Mặt khác các bộ xử lý hỗ trợ việc cung cấp các không gian tách biệt để chạy các chương trình với các yêu cầu (đặc quyền) khác nhau. Các không gian này thường được biểu diễn như là chế độ hệ thống (đặc quyền) và người dùng (thông thường). Không gian hệ thống có được truy nhập không giới hạn tới các tài nguyên của hệ thống máy tính như toàn bộ không gian nhớ, các câu lệnh. Trong khi đó, không gian người dùng bị hạn chế

truy nhập tới bộ nhớ và tập hạn chế các câu lệnh. Một cách tổng quát, hệ thống phân cấp các không gian thực thi thành các lớp bảo vệ (*protection rings*). Các lớp đặt ra các ranh giới chặt chẽ và các mô tả những việc mà các chương trình (tiến trình) hoạt động trong từng lớp như những tài nguyên được truy nhập và những thao tác được phép thực hiện.



Hình 2-1. Các lớp bảo vệ tiêu biểu

Các chương trình nằm ở các lớp bên trong có nhiều đặc quyền hơn là nằm ở lớp ngoài. Nói cách khác, lớp có số thứ tự thấp hơn sẽ có nhiều đặc quyền hơn lớp có số lớn hơn. Số lượng các lớp tùy thuộc vào mục đích và nhu cầu cụ thể của hệ điều hành. Về cơ bản, các thành phần của hệ điều hành hoạt động tại lớp mà cung cấp các truy nhập tới vị trí bộ nhớ, thiết bị ngoại vi, trình điều khiển hệ thống và các tham số cấu hình nhạy cảm. Chính vì sử dụng các tài nguyên quan trọng nên đây là lớp được bảo vệ chặt chẽ nhất.

Các chương trình người dùng chịu hạn chế truy nhập đến bộ nhớ và các thiết bị phần cứng và chịu giám sát của hệ điều hành thông qua các chức năng của hệ điều hành hay lời gọi hệ thống. Nếu người dùng cố yêu cầu CPU thực hiện các lệnh vượt quá quyền hạn thì CPU xử lý những yêu cầu này như là lỗi hoặc cố gắng khóa chương trình lại.

Các lớp tiêu biểu

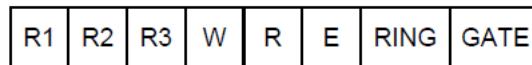
- Lớp 0 (*Ring 0*): Nhân hệ điều hành
- Lớp 1(*Ring 1*): Phần còn lại của hệ điều hành
- Lớp 2 (*Ring 2*): Các trình điều khiển vào/ra và tiện ích

- Lớp 3(*Ring 3*): chương trình ứng dụng

Thực tế, các lớp bảo vệ được triển khai bằng cách kết hợp giữa phần cứng và hệ điều hành. Phần cứng (CPU) được cấu hình để hoạt động với một số lớp nhất định và hệ điều hành được xây dựng sao cho cùng hoạt động ở các lớp này. Các lớp bảo vệ hình thành nên các rào cản giữa chủ thẻ và đối tượng, và thực thi việc giám sát truy nhập khi các chủ thẻ thực hiện việc truy nhập tới các đối tượng:

- Mỗi đối tượng và chủ thẻ được gán một số thẻ hiện cấp độ của lớp bảo vệ
- Chủ thẻ có cấp độ thấp thì không thể truy nhập trực tiếp đối tượng có cấp độ cao hơn. Trong trường hợp cần thiết thì chủ thẻ có thể yêu cầu thông qua lời gọi hệ thống. Hệ điều hành sẽ thực hiện việc kiểm soát và hoàn tất truy nhập.

Việc thay đổi không gian thực hiện của chương trình chỉ được thực hiện thông qua lời gọi hàm *call* tới các mục được phép hợp lệ cho trước. Các hệ thống triển khai cơ chế gọi hàm như vậy dưới dạng các bẫy tới các con trỏ hay vị trí xác định trước trong không gian nhớ hệ thống. Với kiến trúc lớp, việc thay đổi lớp (như *R1*, *R2*) chỉ được hoàn tất thông qua câu lệnh *call* tới vị trí xác định trước trong phần nhớ đặc biệt gọi là cổng (*gate*) mà chúng được gán làm điểm khởi đầu cho lớp bên trong. Việc sử dụng cổng để ngăn chặn các chương trình thực hiện lời gọi hàm vào lớp trong và được thực hiện tại bất kỳ vị trí nhớ nào.



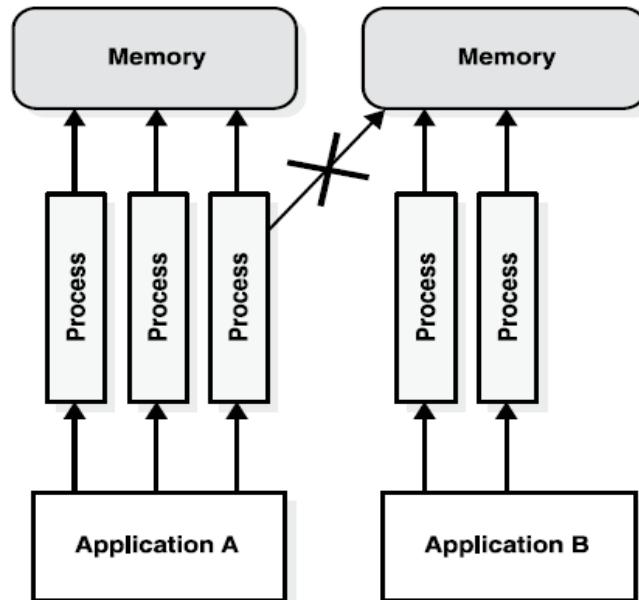
Hình 2-2. Cấu trúc mô tả phần nhớ cổng

Vị trí được chấp nhận trong phần cổng có thể được xác định bằng cách bổ sung thêm các trường trong mô tả cổng bao gồm mức truy nhập của lớp (*ring bracket*), trong đó R1 mức ghi, R2 mức đọc và R3 mức thực thi ví dụ $R1=3$ nghĩa là đoạn này có thể ghi với lớp từ 0 đến 3, và chế độ truy nhập W ghi, R đọc, và E thực thi của mỗi đoạn nhớ cho từng lớp. Tuy nhiên, có thể hạn chế đầu vào tại vị trí khởi đầu (bằng 0) của phần nhớ cổng là đủ. Số thứ tự lớp hiện thời sẽ chỉ thay đổi khi có phần nhớ cổng và câu lệnh *call* được kích hoạt.

2.2 Bảo vệ bộ nhớ

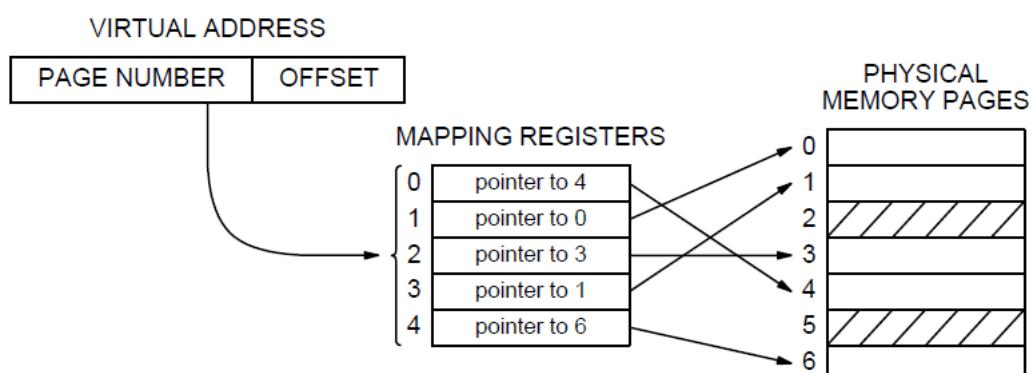
Một trong những yêu cầu phần cứng cơ bản nhất với hệ điều hành an toàn là cơ chế bảo vệ bộ nhớ. Trong hệ điều hành hiện đại, các chương trình của người dùng cần được cách ly về không gian nhớ với nhau và với các chương trình của người dùng hệ thống khác. Cơ chế bộ nhớ ảo được nhiều hệ thống hỗ trợ cho phép thay đổi không gian nhớ vật lý gán cho không gian nhớ mà chương trình có thể được sử dụng. Quá trình này hoàn toàn trong suốt với chương trình người dùng. Các chương trình thay vì truy nhập trực tiếp bộ nhớ vật lý thì thông qua các bảng chỉ số và con trỏ mô tả phần không gian nhớ lô-

gic của chương trình. Chỉ có hệ điều hành mới truy nhập trực tiếp bộ nhớ nhờ các lệnh sử dụng đặc quyền. Như đã nêu trong phần trước, đặc quyền của các câu lệnh này được bộ xử lý kiểm tra trong quá trình thực thi bằng cách so sánh mức độ đặc quyền của câu lệnh với không gian nhớ được yêu cầu. Việc này giúp ngăn chặn các truy nhập của các chương trình người dùng nhằm lấy các thông tin về bộ nhớ vật lý của hệ thống.



Hình 2-3. Xung đột bộ nhớ chương trình người dùng

Trong các hệ thống đầu tiên, không gian nhớ của chương trình được xác định thông qua con trỏ cơ sở, cho biết vị trí bắt đầu, và con trỏ giới hạn, xác định vị trí kết thúc. Như vậy, chương trình người dùng không thể vượt ra ngoài không gian được cấp cho nó trong tình huống xung đột như trong hình Hình 2-3. Để tăng tính hiệu quả trong việc quản lý, các biện pháp cấp phát theo khối nhớ (còn gọi là trang) với kích cỡ hợp lý thường được sử dụng như trong hình dưới đây thể hiện. Các con trỏ trong các thanh ghi cho biết vị trí bắt đầu của các trang trên bộ nhớ vật lý tùy thuộc theo tình hình hoạt động của hệ điều hành. Các trang nhớ của chương trình không nhất thiết phải liên tục. Hình 2-4 biểu diễn cách thức xác định địa chỉ của bộ nhớ vật lý từ địa chỉ của bộ nhớ ảo.

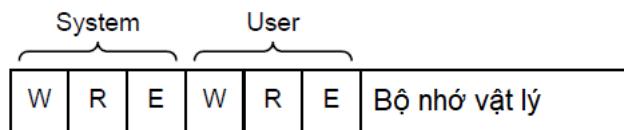


Hình 2-5. Ánh xạ từ bộ nhớ ảo tới bộ nhớ vật lý

Khi hệ thống phân cấp các chương trình theo các lớp bảo vệ (chế độ hệ thống và chế độ người dùng) thì các chương trình người dùng không được phép đọc ghi tùy tiện vào không gian nhớ của hệ thống. Mặt khác, khi ở chế độ hệ thống chương trình được phép truy nhập toàn bộ không gian nhớ vật lý của máy tính. Việc chuyển đổi không gian thực hiện chương trình được thực hiện nhờ câu lệnh đặc biệt. Để tăng độ linh hoạt, hệ điều hành có thể chỉ định chính xác các phần (trang) của bộ nhớ được phép truy nhập tùy theo ngữ cảnh thực hiện chương trình.

Trước khi có được việc quản lý bộ nhớ một cách trong suốt, quyết định truy nhập được dựa trên định danh của các trang vật lý. Mỗi trang được gán nhãn như khóa, các bít truy nhập chỉ định thao tác đọc/ghi. Mỗi chương trình người dùng khi chạy được gán một khóa. Phần cứng, hay bộ xử lý, sẽ tiến hành kiểm tra khóa mỗi khi có tham chiếu bộ nhớ. Truy nhập được phép chỉ khi khóa truy nhập trùng với khóa mô tả đúng thao tác mà người dùng mong muốn.

Với cơ chế chuyển đổi địa chỉ dựa trên các thẻ mô tả (*descriptor*), mỗi tiến trình có tập riêng các thẻ mô tả, chế độ truy nhập của các tiến trình tới phần (trang) nhớ được xác định trong các mô tả như hình dưới đây với W(Ghi), R(Đọc), E(Chạy) là các bít cho biết tiến trình có khả năng truy nhập tới vùng nhớ như thế nào.



Hình 2-6. Mô tả các chế độ truy nhập bộ nhớ của User và System

Do các mô tả được nạp trong quá trình dịch địa chỉ nhớ nên cơ chế kiểm soát dựa trên các mô tả này giảm thiểu các chi phí trong quá trình chuyển ngữ cảnh hay hoán đổi của tiến trình.

Như vậy, các cơ chế phần cứng giúp bảo vệ các thông tin quan trọng với việc quản lý bộ nhớ như cấu trúc chuyển đổi địa chỉ ảo thành địa chỉ vật lý. Những nỗ lực truy nhập thông tin bộ nhớ từ chương trình người dùng sẽ bị ngăn chặn từ mức phần cứng.

2.3 Kiểm soát thao tác vào/ra

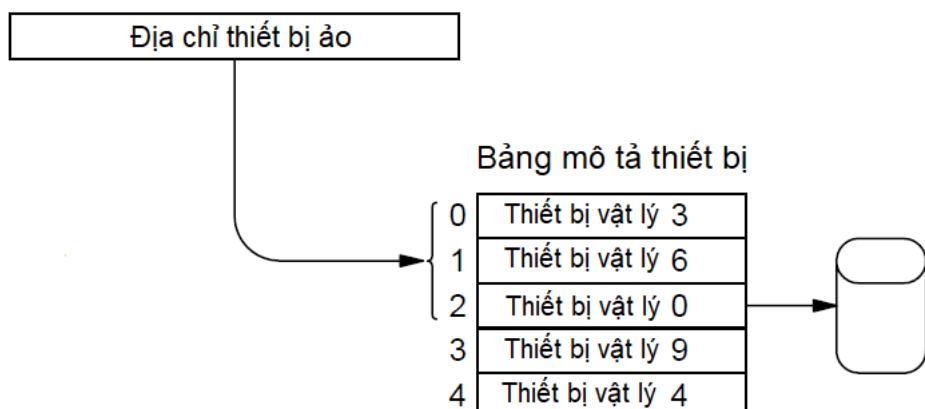
Trong các chức năng của hệ điều hành tiêu biểu, các thao tác vào/ra thường phức tạp nhất. Trong khi phần cứng hỗ trợ các tiến trình, quản lý bộ nhớ được hướng tới việc làm thuận tiện cho người lập trình thì phần cứng hỗ trợ vào/ra có vẻ như không hướng tới việc triển khai hệ thống vào/ra một cách dễ hiểu. Thay vào đây, mục tiêu của thiết kế vào/ra là hiệu năng và chi phí chứ không phải là an toàn. Nói chung, vào/ra là các thao tác đặc quyền được thực hiện chỉ bởi hệ điều hành. Hệ điều hành cung cấp các thao tác mức cao và người dùng không cần thiết kiểm soát các chi tiết của thao tác vào/ra. Hệ điều hành sẽ đơn giản hơn rất nhiều nếu nó không phải trung gian kiểm soát các thao tác này. Trên nguyên tắc, hệ điều hành không tham gia vào việc kiểm soát truy nhập tới

các thiết bị do người dùng sở hữu. Với hỗ trợ phù hợp từ phần cứng, việc kiểm soát các thiết bị này được chuyển cho các chương trình ứng dụng bên ngoài hệ điều hành.

Phần cứng hỗ trợ cho quản lý bộ nhớ chủ yếu tập trung vào các truy nhập từ bộ xử lý tới bộ nhớ. Khi kết hợp với thao tác vào/ra cần có thêm một số kênh thông tin khác như: thiết bị tới bộ nhớ, thiết bị tới bộ xử lý. Việc kiểm soát truy nhập tới các thiết bị này cần phải dựa trên định danh của chủ thể (tiến trình) đại diện cho thiết bị được sử dụng và đối tượng (phần bộ nhớ) được sử dụng. Từ góc độ kiểm soát truy nhập, phần cứng có thể hỗ trợ vào/ra theo 4 cách từ đơn giản đến phức tạp:

- Lập trình
- Không ánh xạ (*unmapped*)
- Ánh xạ trước (*premapped*)
- Ánh xạ đầy đủ

Vào/ra được lập trình là cơ chế đồng bộ theo nghĩa bộ xử lý kiểm soát trực tiếp từng đơn vị dữ liệu được trao đổi tới thiết bị vào/ra. Như vậy vấn đề an ninh duy nhất là tiến trình yêu cầu vào/ra truy nhập tới được thiết bị. Cách trực quan nhất để xử lý vấn đề này là sử dụng bảng mô tả thiết bị (*device descriptor table*) cho phép ánh xạ tên thiết bị tới thiết bị vật lý cụ thể và kèm theo là các mô tả kiểm soát giống như kiểm soát bộ nhớ như trong hình dưới đây



Hình 2-7. Ánh xạ thiết bị ảo tới thiết bị vật lý

Vào/ra không ánh xạ thường không cho phép chương trình người dùng làm việc trực tiếp với các địa chỉ vật lý mà chỉ được kích hoạt từ hệ điều hành. Hệ điều hành sẽ chuyển các địa chỉ bộ đệm dùng cho vào/ra từ chương trình người dùng thành các địa chỉ vật lý. Mặc dù phần cứng hỗ trợ việc chuyển đổi tên thiết bị ảo giống như vào/ra lập trình song việc này không giải phóng nhiệm vụ hệ điều hành phải xác thực và thực hiện các thao tác vào/ra.

Vào/ra ánh xạ trước hay ảo cho phép phần mềm xác định địa chỉ bộ đệm ảo. Khi các câu lệnh vào/ra được thực hiện, bộ xử lý sẽ chuyển các địa chỉ ảo này thành địa chỉ vật lý sử dụng bảng mô tả thiết bị và ánh xạ các thanh ghi của tiến trình hiện thời; và bộ xử lý chuyển địa chỉ vật lý thu được tới thiết bị. Trong quá trình chuyển, bộ xử lý kiểm tra liệu

tiến trình có quyền truy nhập hợp lệ tới vị trí đọc ghi. Từ phía thiết bị, các địa chỉ vào/ra là thật (vật lý) nhưng từ góc độ chương trình các địa chỉ là ảo và việc kiểm soát truy nhập được thực thi nhờ phần cứng. Các thiết bị phải được tin cậy để truy nhập tới vị trí mong muốn trong bộ nhớ.

Ngay cả khi phần cứng không hỗ trợ vào/ra, chương trình người dùng cũng không thể sinh ra các câu lệnh vào/ra mà không có sự can thiệp của hệ điều hành. Cần có cơ chế ngăn chặn hệ điều hành khỏi việc gán lại một cách vô tình (như việc hoán đổi bộ nhớ) các trang nhớ bị ảnh hưởng trong khi vào/ra do người dùng khởi xướng đang xảy ra. Các cơ chế ảo hóa giải phóng hệ điều hành khỏi việc thực hiện chuyển đổi địa chỉ và kiểm soát truy nhập song hệ điều hành vẫn phải chịu trách nhiệm quản lý và theo dõi các thao tác vào/ra.

Với vào/ra ánh xạ trước yêu cầu các thiết bị vào/ra tin cậy và các tham chiếu trả tới địa chỉ vật lý định trước cũng như tuân thủ các hạn chế vào/ra mà bộ xử lý giám sát. Tuy nhiên, có những trường hợp thiết bị vào/ra phức tạp như sử dụng vi chương trình với phần firmware được nạp xuống. Với các hệ thống cần an ninh cao, việc gán độ tin cậy cao cho phần cứng và phần firmware mà đôi khi không kiểm soát được lại không phù hợp. Đặc biệt khi bản thân các thiết bị phần cứng lại nằm ngoài phạm vi đảm bảo an ninh như các thiết bị ở xa.

Dạng vào/ra an toàn hơn gồm phần cứng thực hiện việc chuyển địa chỉ từ ảo-thực (vật lý) với mỗi tham chiếu bộ nhớ được thực hiện bởi thiết bị. Thiết bị hoạt động như đối tượng không tin cậy (có thể chứa mã trojan) sử dụng các địa chỉ ảo khi đọc ghi thông tin trong bộ nhớ; phần cứng mà thực hiện việc chuyển địa chỉ nằm trong vùng được bảo vệ thực hiện việc ánh xạ và kiểm tra truy nhập. Phần cứng sử dụng cùng các mô tả bộ nhớ mà thuộc về các tiến trình khởi tạo vào/ra. Nhờ việc chuyển địa chỉ và kiểm tra được thực hiện trên từ mức đơn vị dữ liệu nên sẽ không gặp phải vấn đề an ninh khi hệ điều hành phân phối lại bộ nhớ trong quá trình vào/ra. Do việc vào/ra là dị bộ nên địa chỉ ảo của thiết bị vào/ra không nhất thiết phải gắn với không gian nhớ của chương trình đang chạy.

2.4 Ảo hóa

Ảo hóa giúp cho việc tận dụng và chia sẻ tài nguyên máy tính được thuận tiện và dễ dàng hơn đặc biệt khi năng lực xử lý của hệ thống máy tính được nâng cao. Phần cứng hỗ trợ ảo hóa làm cho hiệu năng của các phần mềm ảo hóa được cải thiện và hiệu năng được người dùng chấp nhận hơn. Các yêu cầu cơ bản với máy tính cho phép ảo hóa bao gồm:

- **Tính hiệu quả:** Tất cả các câu lệnh bình thường được thực hiện trực tiếp bởi phần cứng mà không có sự can thiệp nào của các chương trình giám sát.
- **Kiểm soát tài nguyên:** Không cho phép bất kỳ chương trình nào ảnh hưởng tới các tài nguyên hệ thống như bộ nhớ và tính sẵn dùng của nó. Bộ cấp phát được gọi bắt cứ khi nào chương trình cần.

- **Bình đẳng:** Bất kỳ chương trình nào đang chạy với sự hiện diện của chương trình kiểm soát cũng được thực hiện mà không khác gì trường hợp không có chương trình giám sát này với truy nhập không hạn chế các câu lệnh đặc quyền mà người lập trình dự định.

Ảo hóa cũng giúp cho việc đảm bảo an toàn cho chương trình một cách linh hoạt hơn do các chương trình người dùng hoạt động trong các không gian cách ly với nhau. Tuy nhiên, có vấn đề khó khăn khi thiết kế máy tính có khả năng “bẫy” các câu lệnh có đặc quyền. Các câu lệnh có đặc quyền nhất được thực hiện ở lớp 0 và dành riêng cho hệ điều hành. Các hệ thống ảo chạy trên nền hệ điều hành không thể truy nhập đến lớp 0 này một cách trực tiếp mà phải thông qua bước chuyển không gian thực hiện (thay đổi đặc quyền). Ảo hóa một phần (paravirtualisation) giúp làm giảm chi phí cho việc chuyển đổi này bằng cách sử dụng bộ thư viện API mà phần mềm ảo có thể truy nhập các thao tác đặc quyền.

Ảo hóa phần cứng làm giảm sự can thiệp của hệ thống chủ trong việc xử lý các vấn đề quản lý việc chuyển không gian địa chỉ và đặc quyền. Intel với VT-i và AMD với AMD-V là các công nghệ ảo hóa giúp đơn giản hóa hệ thống chủ và đảm bảo hiệu năng gần như thật với hệ thống được ảo hóa. Hỗ trợ từ phần cứng cho phép chuyển đổi nhanh chóng giữa hệ thống ảo hóa và bộ phận giám sát (hệ thống chủ) và cấp các thiết bị vào/ra một cách an toàn cho các hệ thống ảo hóa.

2.5 Kết luận

Chương này trình bày cơ chế bảo vệ và cách ly các tiến trình thông qua việc sử dụng cơ chế lớp bảo vệ với sự hỗ trợ từ phần cứng. Việc phân chia các lệnh hay các thao tác theo các nhóm đặc quyền cho phép hạn chế việc sử dụng các câu lệnh của các chương trình người dùng cũng như các chương trình ở mức hệ điều hành trong việc sử dụng các tài nguyên của hệ thống. Qua đó bảo vệ các tài nguyên này khỏi các truy nhập không được phép.

Sử dụng nguyên tắc tương tự với vấn đề quản lý bộ nhớ và các thao tác vào/ra. Các tiến trình bị hạn chế truy nhập trực tiếp tới không gian nhớ vật lý thông qua các kỹ thuật như phân trang và bộ nhớ ảo. Các thông tin trực tiếp về việc cấp phát vị trí nhớ cũng như việc truy nhập vào các thẻ thông tin quản lý việc cấp phát được lưu trữ trong không gian nhớ được bảo vệ và được kiểm soát thông qua cơ chế đặc quyền. Như vậy, các tiến trình người dùng sẽ bị che dấu các thông tin vật lý và việc truy nhập chỉ dành cho các tiến trình của hệ điều hành. Điều này giúp giảm thiểu rủi ro và xung đột giữa các tiến trình cũng như việc rò rỉ thông tin bởi các tiến trình có độ tin cậy thấp. Ảo hóa là bước tiến quan trọng trong việc quản lý và phân chia các tài nguyên dùng chung của hệ thống phần cứng. Việc triển khai thành công ảo hóa giúp giảm thiểu các rủi ro về an toàn thông tin.

Các cơ chế phần cứng cung cấp công cụ tin cậy cho việc triển khai an toàn và thẩm tra được việc thoả mãn các yêu cầu an toàn của hệ thống. Việc triển khai ở mức phần

cứng giúp hạn chế tối đa việc can thiệp và sửa đổi trái phép vào quá trình kiểm soát do cách cơ chế này được thực thi từ mức phần cứng.

2.6 Câu hỏi ôn tập

- 1) Nêu khái niệm về lớp bảo vệ của CPU?
- 2) Giải thích việc thực hiện cơ chế lớp bảo vệ trong tập lệnh x86?
- 3) Cách thức bảo vệ bộ nhớ từ mức phần cứng?
- 4) Nêu các kỹ thuật đảm bảo an toàn cho các thiết bị vào/ra từ mức phần cứng?
- 5) Các yêu cầu cơ bản với máy ảo?

CHƯƠNG 3. AN TOÀN CÁC DỊCH VỤ CƠ BẢN CỦA HỆ ĐIỀU HÀNH

Chương này trình bày các công cụ và cách thức giúp hệ điều hành có thể triển khai các dịch vụ của mình một cách an toàn dựa trên kiến trúc x86. Trước hết là việc quản lý các chương trình người dùng hay tiến trình sao cho các chương trình người dùng được thực thi một cách an toàn cũng như ngăn cản việc xâm phạm tới các thông tin cần bảo vệ như đoạn mã hay đoạn dữ liệu. Tiếp theo là cách thức cách ly không gian nhớ giữa các chương trình và phần nhân của hệ điều hành. Phần thứ 3 trình bày cách thức bảo vệ tính toàn vẹn của các chương trình tham gia vào quá trình khởi động máy cũng như bảo vệ tính bí mật nội dung dữ liệu thông qua việc sử dụng mô-đun hạ tầng tin cậy TPM. Phần cuối của chương thảo luận an toàn của hệ điều hành phổ biến là Windows và Unix/Linux thông qua việc triển khai các cơ chế đảm bảo an toàn. Cách thức triển khai các cơ chế bảo vệ có thể dẫn đến các điểm yếu cố hữu. Ngoài ra, phần này cũng thảo luận một số cách thức triển khai các cơ chế nâng cao tính an toàn.

3.1 Quản lý tiến trình

Kiến trúc tập lệnh x86 hỗ trợ việc quản lý thực thi các chương trình bằng cách triển khai cơ chế bảo vệ theo lớp đặc quyền với việc kiểm soát truy nhập đến các câu lệnh cũng như dữ liệu. Cơ chế này được phối hợp giữa hệ điều hành và phần cứng để hạn chế các thao tác mà chương trình người dùng có thể thực hiện được. Các tài nguyên được bảo vệ với các thao tác của người dùng chủ yếu là bộ nhớ, các cổng vào/ra và khả năng chạy một số câu lệnh đặc biệt. Tại bất kỳ thời điểm nào, bộ xử lý x86 hoạt động tại một lớp đặc quyền nhất định mà lớp này quyết định các thao tác mà đoạn mã có thể thực thi được hay không.

Có khoảng 15 câu lệnh được bảo vệ dành riêng cho lớp có đặc quyền cao nhất (lớp 0). Các câu lệnh khác chủ yếu hạn chế việc truy nhập tới các tham số (tổng hợp) của câu lệnh đó. Các câu lệnh này có thể phá vỡ cơ chế bảo vệ hay gây ra sự hỗn loạn nếu được thực thi ở chương trình người dùng, do vậy chúng được dành riêng cho phần nhân hay hệ thống. Bất kỳ nỗ lực nào nhằm chạy các câu lệnh này bên ngoài lớp 0 sẽ gây ra lỗi như khi chương trình cố truy nhập ô nhớ không hợp lệ. Bộ xử lý x86 theo dõi mức đặc quyền (lớp bảo vệ) thông qua các trường:

- **RPL: Requested Privilege Level** trên thanh ghi đoạn dữ liệu. Giá trị của trường này không thể được gán trực tiếp bởi các câu lệnh nạp dữ liệu mà chỉ bởi các câu lệnh thay đổi luồng thực hiện chương trình như câu lệnh *call*.
- **CPL: Current Privilege Level** trên thanh ghi đoạn lệnh. Giá trị này được duy trì bởi chính CPU và nó luôn bằng với mức bảo vệ hiện thời của CPU.

Nói cách khác, giá trị CPL cho biết mức độ bảo vệ của đoạn mã được thực hiện.



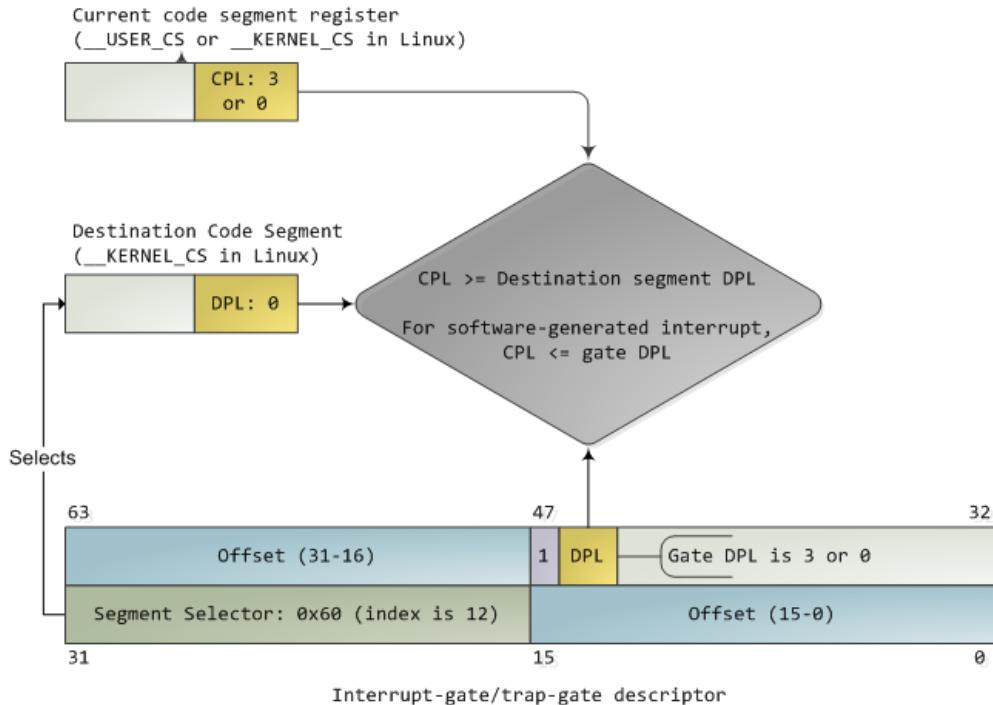
Hình 3-1. Thẻ chọn đoạn dữ liệu và lệnh

Hình trên đây mô tả cấu trúc và vị trí các thông tin về mức bảo vệ tương ứng với các thẻ chọn đoạn dữ liệu (*data*) và lệnh (*code*). Các mức độ đặc quyền của bộ xử lý không liên quan tới vai trò người dùng của hệ điều hành dù là quản trị (*administrator*) hay người dùng thông thường. Toàn bộ các đoạn mã (lệnh) của người dùng được chạy ở lớp 3 (mức ít đặc quyền nhất) và toàn bộ đoạn mã của phần nhân hoạt động ở lớp 0.

Do bị hạn chế truy nhập tới bộ nhớ, các cổng vào/ra, chế độ người dùng hầu như không thể tác động tới các tài nguyên bên ngoài trừ phi thực hiện lời gọi đến phần nhân. Chương trình người dùng không thể mở file, gửi các gói dữ liệu hay cấp phát bộ nhớ. Chương trình người dùng bị đóng trong “hộp kín” (*sandbox*) do đoạn mã của phần nhân đặt ra. Về mặt thiết kế không thể xảy ra chuyện rò rỉ bộ nhớ của chương trình hay truy nhập trực tiếp đến các cấu trúc dữ liệu quan trọng của hệ thống. Khi chương trình người dùng kết thúc thì “hộp kín” này cũng bị xóa bỏ.

Nếu như các đoạn mã với đặc quyền thấp có thể chuyển quyền điều khiển tới bất kỳ vị trí nào trong nhân thì có thể dễ dàng phá hỏng hệ điều hành bằng cách chuyển câu lệnh tới vị trí sai lệch qua câu lệnh nhảy *jmp*. Thực tế điều này được kiểm soát thông qua thẻ mô tả cổng (*gate descriptor*). Thẻ mô tả cổng có thể là một trong 4 loại: mô tả cổng gọi hàm (*call-gate*), cổng ngắt (*interrupt-gate*), cổng bẫy (*trap-gate*) và cổng nhiệm vụ (*task-gate*). Cổng gọi hàm có được sử dụng với các lời gọi hàm thông thường như *call* hay *jump*. Cổng nhiệm vụ thường được sử dụng khi bị lỗi kép do nhân hay phần cứng. Cổng ngắt và bẫy quan trọng hơn và được dùng để xử lý các ngắt phần cứng như bô đinh thời, ô cứng và các ngoại lệ như lỗi trang nhớ, chia phải 0. Các cổng này được lưu trữ trong bảng mô tả ngắt IDT (*Interrupt Descriptor Table*) và cung cấp thông tin về mức đặc quyền của cổng thông qua trường DPL (*Descriptor Privilege Level*).

Khi xảy ra việc dịch chuyển đoạn mã, việc kiểm tra mức đặc quyền được thể hiện như trong hình dưới đây.



Hình 3-2. Kiểm tra mức đặc quyền của đoạn lệnh

Phần nhân thông thường sử dụng thanh ghi đoạn mã cho phần mã của nhân trong phần mô tả cổng. Việc chuyển đoạn mã thực hiện sẽ không xảy ra từ lớp bảo vệ cao sang mức bảo vệ thấp. Đặc quyền phải ở mức ngang bằng hay cần được nâng cấp. Trong cả hai trường hợp mức đặc quyền của đoạn mã hiện thời CPL phải bằng với mức đặc quyền của đoạn mã được chuyển tới DPL, mức đặc quyền của đoạn mã lệnh hiện thời. Nếu CPL thay đổi đồng thời dẫn đến việc thay đổi ngăn xếp và nếu việc chuyển đổi đoạn mã được thực hiện thông qua lệnh *INT* thì cần đảm bảo thêm việc mức đặc quyền DPL phải giống hoặc thấp hơn của CPL. Việc này ngăn cản chương trình người dùng kích hoạt ngẫu nhiên các đoạn mã ngắt. Nếu việc kiểm tra không thành công thì lỗi bảo vệ sẽ được sinh ra.

Với các câu lệnh sử dụng các thiết bị vào/ra cũng được kiểm tra theo cách tương tự. Mỗi câu lệnh IN/OUT được liên kết với cờ trạng thái mô tả mức đặc quyền IOPL (I/O privilege level) cho biết lớp bảo vệ tương ứng. Câu lệnh vào/ra sẽ chỉ được thực hiện khi mức đặc quyền của đoạn mã nhỏ hơn hoặc bằng với mức đặc quyền của lệnh vào/ra.

3.2 Quản lý bộ nhớ

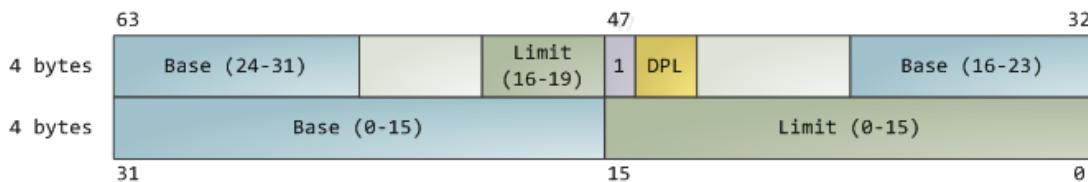
3.2.1 Bảo vệ bộ nhớ qua đặc quyền

Với chức năng quản lý bộ nhớ của hệ điều hành, không gian nhớ của các tiến trình cần được cách ly với nhau sao cho mỗi tiến trình chỉ có thể truy nhập được không gian nhớ của riêng mình. Các không gian nhớ của các tiến trình phải được cách biệt với nhau và với phần nhân của hệ điều hành. Việc này ngăn chặn lỗi hay phần mềm xâm bén trong

tiến trình không ảnh hưởng tới các tiến trình khác và bản thân hệ điều hành. Việc cố truy nhập bộ nhớ không phải của tiến trình sẽ gây ra lỗi phần cứng, như là lỗi xâm phạm ô nhớ được bảo vệ, và làm cho tiến trình vi phạm phải chấm dứt hoạt động.

Các tiến trình sử dụng địa chỉ bộ nhớ lô-gíc thay vì địa chỉ bộ nhớ vật lý mà bộ xử lý kết nối tới thông qua buýt hệ thống. Như vậy, cần phải thực hiện việc chuyển đổi (ánh xạ) từ địa chỉ lô-gíc của chương trình thành địa chỉ vật lý thực sự trước khi thao tác đọc/ghi bộ nhớ được diễn ra.

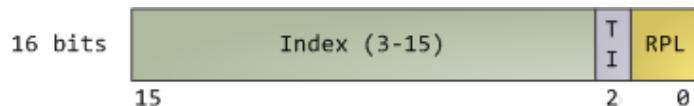
Kiến trúc x86 sử dụng phương pháp phân đoạn để quản lý không gian nhớ chương trình nhờ vào việc tách biệt các chức năng của không gian nhớ (đoạn mã, đoạn dữ liệu, đoạn ngắn xếp) cũng như việc chia sẻ. Các đoạn chức năng được truy nhập nhờ vào các thanh ghi đoạn như thanh ghi đoạn lệnh CS hay đoạn dữ liệu DS. Các thông tin quản lý của các thanh ghi đoạn được lưu trong thẻ mô tả thanh ghi đoạn (segment descriptor).



Hình 3-3. Thẻ mô tả đoạn

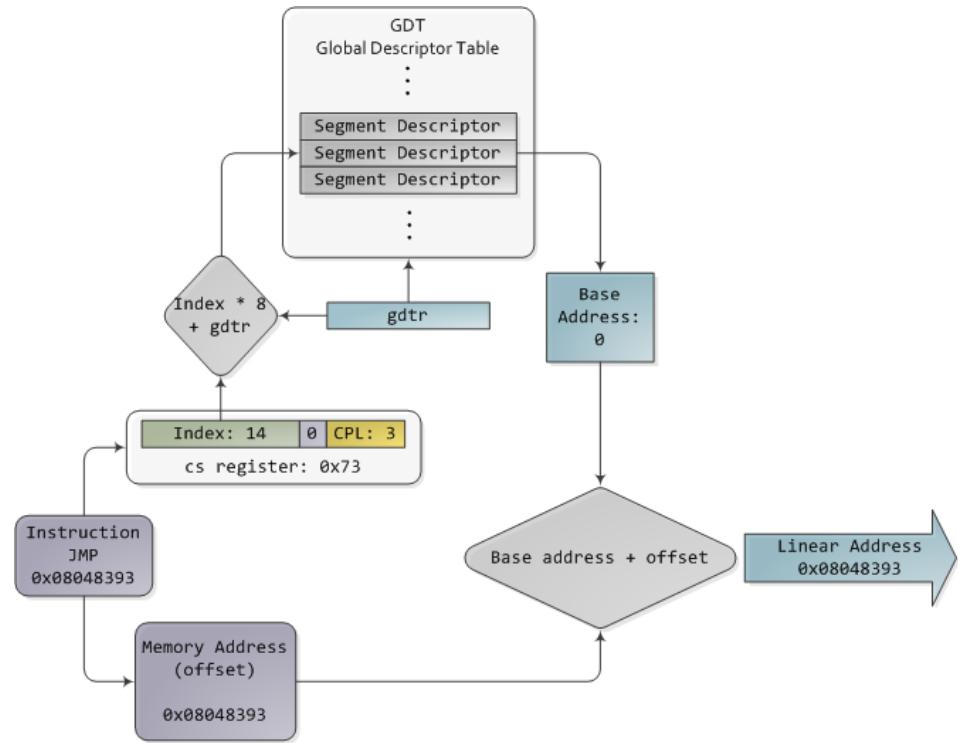
Địa chỉ cơ sở (base) sử dụng cấu trúc tuyến tính 32 bit cho biết vị trí bắt đầu của đoạn và giới hạn (limit) cho biết độ lớn của đoạn. DPL là mức đặc quyền của đoạn: 0 ứng với nhiều đặc quyền nhất; 3 ít đặc quyền nhất và kiểm soát việc truy nhập tới đoạn.

Các thẻ mô tả đoạn được lưu trong hai bảng: bảng mô tả toàn cục GDT (*Global Descriptor Table*) và bảng mô tả cục bộ LDT (*Local Descriptor Table*). Để chọn đoạn, chương trình cần sử dụng thẻ chọn đoạn (segment selector) có cấu trúc như dưới đây.



Hình 3-4. Thẻ chọn đoạn

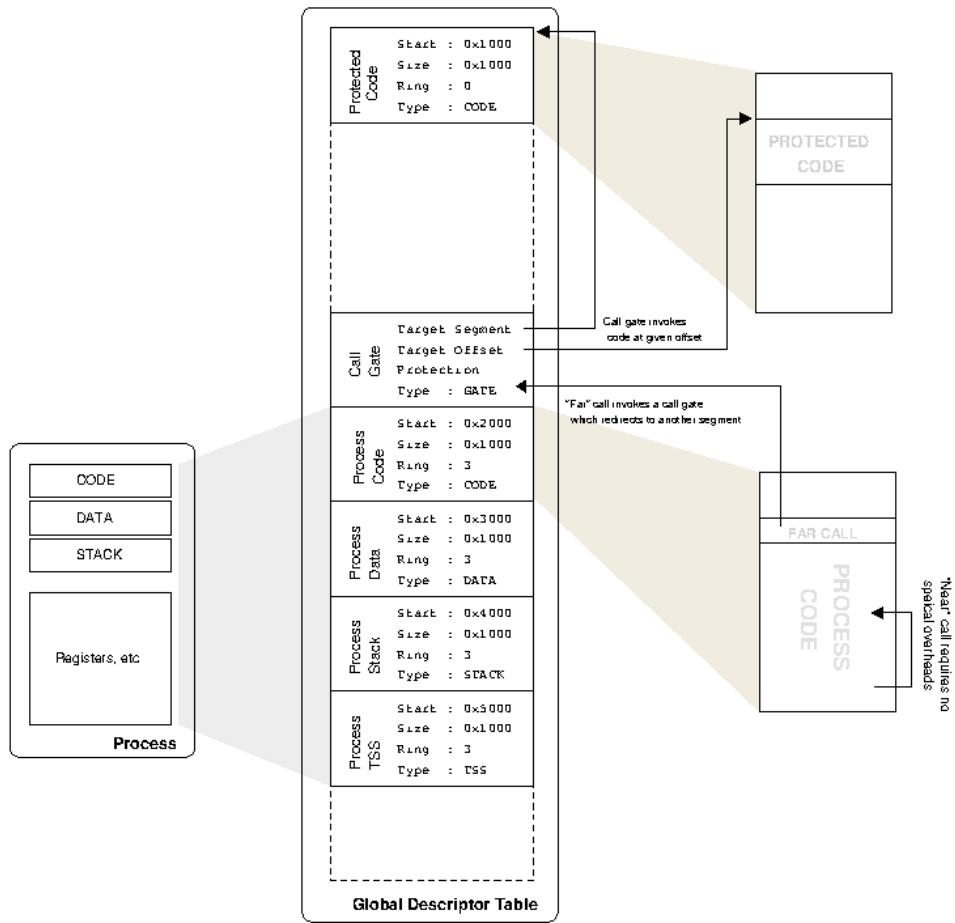
Bít TI giúp phân biệt GDT (TI=0) và LDT (TI=1) còn RPL mô tả mức đặc quyền cần thiết khi truy nhập vào đoạn tương ứng. Thông thường cần 1 bảng GDT để truy nhập trực tiếp vào không gian nhớ phẳng. Vị trí đầu tiên của bảng GDT được lưu trong thanh ghi gdtr. Cách thức ánh xạ địa chỉ phẳng từ bảng GDT như trong hình dưới đây.



Hình 3-5. Truy nhập bộ nhớ qua GDT

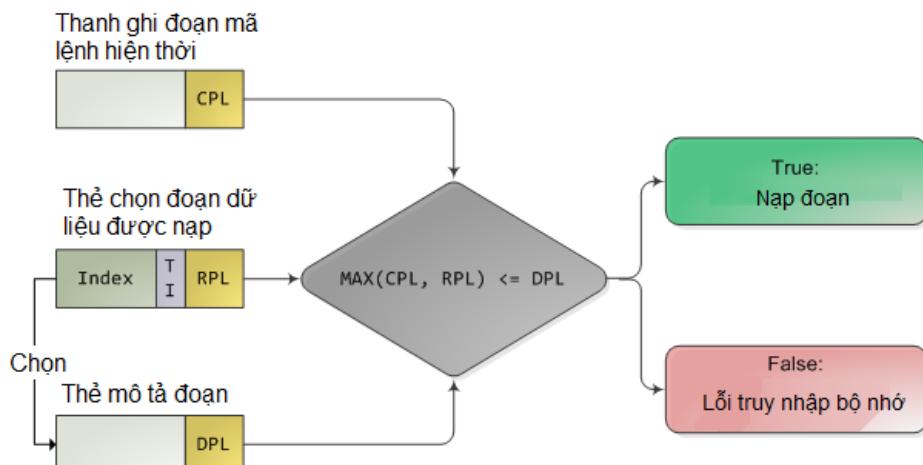
Nếu chương trình người dùng có khả năng sửa đổi thông tin trong bảng GDT thì tự bản thân chương trình này có khả năng thay đổi đặc quyền của mình. Như vậy, hệ điều hành cần phải lưu bảng này trong phần bộ nhớ dành cho nhân và không thể truy nhập trực tiếp từ chương trình người dùng. Nói cách khác, các thông tin này cần được lưu vào không gian nhớ được bảo vệ.

Hình 3-6 dưới đây minh họa cấu trúc không gian nhớ của chương trình với mô tả toàn cục GDT. Ngoài không gian thông thường như dữ liệu và đoạn mã, bảng GDT cung cấp thông tin về các đoạn mã được bảo vệ (*protected code*) và các cổng gọi hàm (*call gate*) giúp chương trình truy nhập tới bộ nhớ được bảo vệ. Khi việc bảo vệ truy nhập bộ nhớ thông qua việc kiểm soát đặc quyền được kích hoạt, toàn bộ việc kiểm tra được thực hiện thông qua phần cứng nên chống lại can thiệp vào cơ chế kiểm tra từ chương trình người dùng.



Hình 3-6. Cấu trúc không gian nhớ của tiến trình và thông tin quản lý bộ nhớ

Như trong hình mô tả đặc quyền (Hình 3-7), khi đoạn dữ liệu được nạp việc kiểm tra được diễn ra theo các bước như sau. Mức độ bảo vệ của đoạn bộ nhớ được yêu cầu truy nhập DPL được so sánh với CPL và RPL. Nếu giá trị DPL mà nhỏ hơn thì việc truy nhập là hợp lệ. Nếu không khi này sẽ phát sinh lỗi truy nhập bộ nhớ được bảo vệ. Việc sử dụng RPL cho phép đoạn mã nhanh nạp đoạn bộ nhớ dùng mức đặc quyền thấp hơn. Riêng với đoạn ngắn xếp, cả ba mức CPL, RPL và DPL phải giống nhau hoàn toàn.



Hình 3-7. Kiểm tra đặc quyền truy nhập bộ nhớ

Thực tế, việc bảo vệ đoạn bộ nhớ không quá quan trọng bởi các nhân của hệ điều hành hiện đại sử dụng không gian địa chỉ phẳng còn chương trình người dùng truy nhập toàn bộ không gian địa chỉ tuyến tính (ảo). Cơ chế bảo vệ bộ nhớ hữu ích được thực hiện tại bộ phận quản lý trang khi thực hiện việc chuyển đổi địa chỉ phẳng thành địa chỉ vật lý. Mỗi trang nhớ được mô tả bởi khoản mục bảng trang PTE (page table entry) chứa hai trường phục vụ cho việc bảo vệ là cờ giám sát (supervisor) và đọc/ghi. Cờ giám sát cho phép bảo vệ không gián nhở của nhân. Khi cờ này bật lên, trang nhớ tương ứng sẽ không truy nhập được từ mức người dùng (mức đặc quyền 3). Cờ đọc/ghi không thực sự quan trọng với việc thực thi đặc quyền. Khi tiến trình được nạp vào bộ nhớ, trang chứa mã được bật cờ chỉ đọc. Như vậy các thao tác ghi lên trang nhớ này sẽ bị báo lỗi và từ chối thực hiện.

3.2.2 Cấm thực thi dữ liệu

Tân công tràn bộ đệm là một trong những kỹ thuật khai thác lỗ hổng trong việc tổ chức và quản lý không gian nhớ của chương trình. Ngăn chặn thực thi dữ liệu DEP (*Data Execution Prevention*) là kỹ thuật bảo vệ bộ nhớ ở mức hệ thống được tích hợp vào hệ điều hành. Về cơ bản, DEP cho phép hệ thống đánh dấu một hay nhiều trang bộ nhớ là không thực thi được. Như vậy đoạn mã tại vùng nhớ này sẽ không được thực thi và tạo thêm rào cản cho việc khai thác tràn bộ đệm.

Kiến trúc x86 hỗ trợ cơ chế này thông qua bit cấm chạy (*execute-disable* hay *no-execute*) đặt tại trang nhớ. Khi chương trình cố thực thi đoạn mã từ trang nhớ có bít cấm chạy được bật sẽ gây ra lỗi trang. Ở mức độ hệ điều hành nếu lỗi này không được xử lý thì chương trình đang chạy sẽ bị chấm dứt việc thực thi. Cơ chế cấm chạy có thể được kích hoạt bằng bít cấm chạy tại bất kỳ mức nào trong cấu trúc trang. Khi cơ chế bảo vệ này không được kích hoạt, trang nhớ có thể được dùng lẩn giữa mã và dữ liệu.

Mặt khác, DEP có thể được thực hiện ở mức hệ điều hành như trong Windows XP nhằm ngăn chặn mã độc lợi dụng cơ chế xử lý lỗi (*exception*) trong Windows. DEP từ phần mềm có thể hoạt động trên bất kỳ bộ xử lý bất kể bộ xử lý có hỗ trợ DEP từ phần cứng hay không. Tuy nhiên, DEP bằng phần mềm chỉ giới hạn bảo vệ các file nhị phân của hệ thống

3.3 Hệ thống file

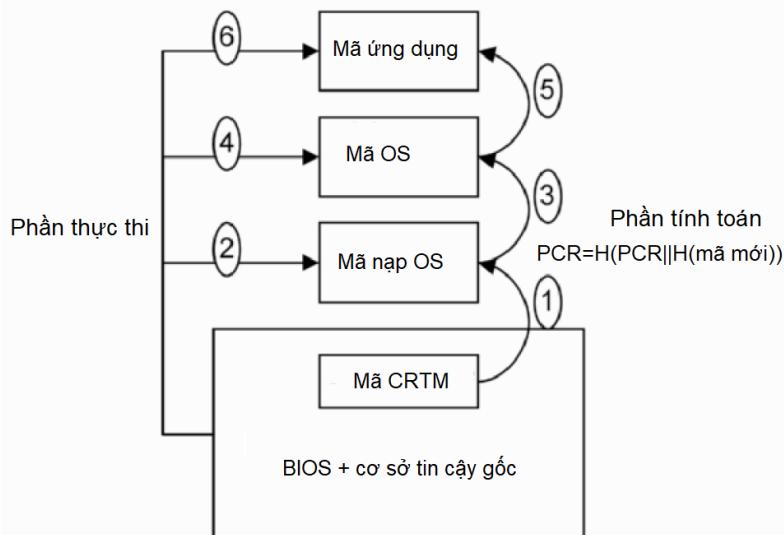
Phần này trước hết giới thiệu cách thức bảo vệ quá trình nạp các file khởi động của hệ điều hành thông qua việc sử dụng môđun tính toán tin cậy TPM. Tiếp theo giới thiệu cách thức sử dụng TPM trong việc mã hóa hệ thống file.

3.3.1 Khởi động được bảo vệ

Trong kiến trúc x86, BIOS (*Basic Input Output System*) chứa các phần mềm quan trọng được chạy trước tiên và cung cấp các dịch vụ cơ bản cho hệ điều hành. Phần BIOS có tất cả các đặc quyền truy nhập tới toàn bộ phần cứng máy tính. Trong một số trường

hợp BIOS có thể lập trình lại các thiết bị như thay đổi vị trí nhớ để nạp các đoạn mã của hệ điều hành hay cách thức thực hiện các giao dịch truy nhập bộ nhớ trực tiếp DMA. Ngoài ra, BIOS cung cấp các đoạn mã cho chế độ quản lý hệ thống SMM (*System Management Mode*) mà các đoạn mã này được sử dụng trong toàn bộ thời gian hoạt động của hệ thống. Có thể thấy BIOS đóng vai trò quan trọng trong việc xác lập sự tin cậy trong suốt quá trình hoạt động của hệ điều hành. Tuy vậy, BIOS có thể chứa mã xấu do bên cung cấp xây dựng một cách có chủ đích như cửa hậu (*backdoor*) hoặc do người dùng sửa đổi BIOS mà không có cơ chế bảo vệ việc lập trình lại BIOS.

Việc bảo vệ các phần mềm được sử dụng trong quá trình khởi động máy tính, bao gồm BIOS, có thể được thực hiện thông qua mô-đun TPM. Trong đó, bộ phận RTM (*Root of Trust for Measurement*) và RTR (*Root of Trust for Reporting*) là các thành phần căn bản để tạo dựng được sự tin cậy thông qua quá trình khởi động được bảo vệ (*measured boot*) và xác lập khởi động được xác thực (*authenticated boot*). Quá trình khởi động được bảo vệ được biểu diễn trong hình dưới đây.



Hình 3-8. Tính toán cơ sở tin cậy trong quá trình khởi động

Tại thời điểm khởi động, phần BIOS được kích hoạt và chạy trước. Sau đó, tại bước 2 BIOS tiến hành nạp đoạn mã nạp OS giúp khởi động hệ điều hành. Bước 4 là thực thi mã hệ điều hành. Cuối cùng, bước 6 là thực thi ứng dụng. Các bước 1, 3, và 5 tiến hành tính toán cơ sở tin cậy cho các đoạn mã và theo công thức $PCR \equiv H(PCR|H(\text{mã mới}))$ với H là hàm băm của RTM và PCR là các thanh ghi bên trong của TPM. Chi tiết tính toán giá trị cơ sở tin cậy cho quá trình khởi động được bảo vệ như sau:

1. RTM lưu lại chỉ số định danh của hệ thống vào vị trí an toàn là các thanh ghi PCR. Chỉ số này đóng vai trò là giá trị tin cậy gốc CRTM (*Core Root of Trust for Measurement*) cho việc tính toán mức độ toàn vẹn của quá trình khởi động. Đây có thể chỉ là biện pháp bảo vệ mã của hạ tầng tính toán hay đơn giản chỉ là định danh.

2. Trước khi khởi tạo các phần mã tiếp theo trong chuỗi khởi động, RTM tính toán mã băm của bộ phận đó và lưu lại vào nơi an toàn. Sau đó chuyển quyền điều khiển cho phần mã đó.

3. Lặp lại bước 2 cho từng liên kết trong chuỗi

Như vậy với bất kỳ chương trình nào và bất cứ khi nào đều có thể nhận được đảm bảo về tính toàn vẹn của bản thân chương trình đó và các chương trình khác tham gia vào hoạt động của nó. Nói cách khác, bằng cách kiểm tra các giá trị được lưu trữ một cách an toàn trong TPM với giá trị mà chương trình tính toán được, chương trình có thể chắc chắn về tình trạng của chương trình khác mà nó phối hợp với. Điều này cho phép các chương trình tin cậy chống lại việc giả mạo cũng như xâm nhập vào quá trình hoạt động bình thường của hệ thống. Quá trình tính toán cơ sở tin cậy như trên còn được gọi là cơ sở tin cậy tĩnh vì toàn bộ các chương trình tham gia đều được tính toán và tạo nên một chuỗi tin cậy.

Các thông tin được coi là lưu trữ an toàn khi sử dụng các thanh ghi cấu hình của hạ tầng PCR (*Platform Configuration Registers*) bên trong TPM là nhờ các yếu tố sau:

- Các ô nhớ này được bảo vệ bằng cách có thể đọc nhưng không thể ghi tùy ý
- Dữ liệu được ghi vào theo dang tò hợp với giá trị băm của dữ liệu hiện thời và giá trị trước đó.

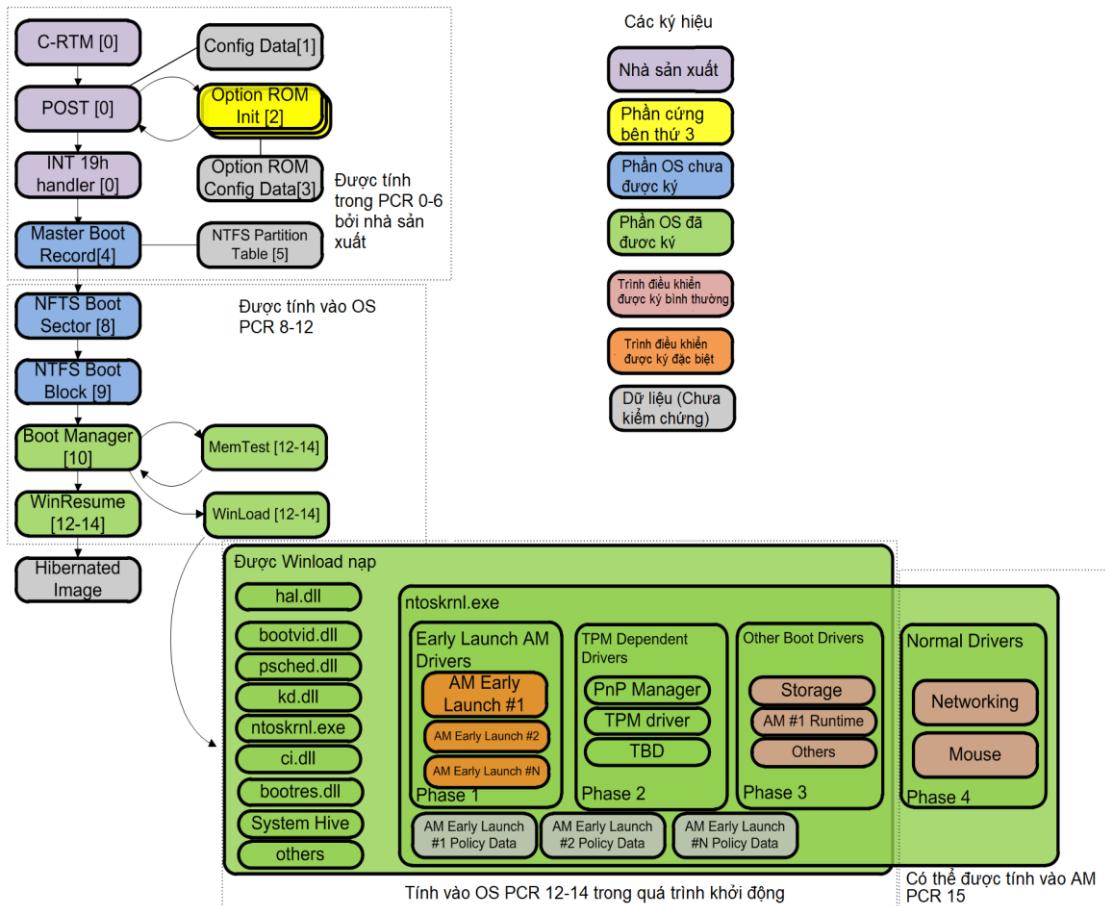
Bên cạnh đó, TPM cung cấp bản sao có xác nhận trạng thái PCR đảm bảo đối tác (chương trình) có thể kiểm tra trạng thái của hạ tầng tính toán. Điều quan trọng là việc xác nhận diễn ra bên trong TPM như vậy chống lại việc xâm nhập và giả mạo dữ liệu.

Do chuỗi chương trình tham gia vào quá trình khởi động rất lớn nên việc tính toán cho toàn bộ chuỗi là không đơn giản nhất là trong trường hợp các đoạn mã và chương trình từ hệ điều hành liên tục được cập nhật và sửa chữa. Các nhà sản xuất TPM đưa ra giải pháp linh hoạt bằng cách sử dụng cơ sở tin cậy động (*dynamic root of trust*). Khi này chuỗi tính toán sử dụng đoạn mã cố định từ nguồn tin cậy được phép nạp và chạy, chương trình được chọn như vậy làm giảm độ dài của chuỗi khởi động.

Hình 3-9 thể hiện quá trình khởi động được bảo vệ và kết hợp với phần mềm chống mã độc trong Windows để nâng cao tính toàn vẹn của hệ thống sử dụng BIOS truyền thống. Mũi tên trong hình cho biết quá trình khởi động trao quyền điều khiển từ bộ phận này sang bộ phận khác.

Thành phần kích hoạt đầu tiên là cơ sở tin cậy gốc CRTM và sau đó là bộ phận firmware khởi động chuỗi tính toán cơ sở tin cậy. Tiếp theo là phần kiểm tra khi bật máy POST (*Power On Self-Test*). Các thanh ghi an toàn PCR của TPM được thể hiện trong ngoặc vuông. Phần chương trình khởi động có thể sử dụng các dữ liệu về cấu hình (config data) hoặc đoạn mã ROM do các nhà sản xuất thiết bị (bên thứ 3) cung cấp. Đoạn mã khởi động hệ điều hành (bootstrap) được chạy thông qua ngắt 19H và nó tiến hành tìm kiếm đoạn mã khởi động trong bản ghi khởi động chính MBR (*Master Boot Record*) được lưu trong thiết bị lưu trữ của hệ điều hành. Các chương trình thực hiện quá trình

trên do nhà sản xuất phần cứng máy tính cung cấp và mô-đun TPM thực hiện việc tính toán các giá trị tin cậy của chúng.



Hình 3-9. Khởi động được bảo vệ trong Windows với BIOS truyền thống

Phần hệ điều hành OS được bắt đầu kể từ khi đoạn mã của nó được lựa chọn và trao quyền điều khiển từ đoạn mã quản lý khởi động (*Boot Manager*), ngay sau khi phần nhân của hệ điều hành được khởi động nhờ Winload và tính từ PCR[8] đến [12] như trong hình vẽ. Phần nhân nạp các trình điều khiển và kiểm tra các chữ ký (*signature*) của các trình điều khiển này. Các trình điều khiển này bao gồm phần kiểm soát mã độc sớm ELAM (*Early Lauch AntiMalware*) ở pha 1, trình điều khiển TPM – pha 2, và trình điều khiển khởi động khác – pha 3. Khi hoạt động, phần mềm chống mã độc AM, mà thực thi các chính sách cho quá trình khởi động được bảo vệ, phải được chạy như phần điều khiển khởi động tối quan trọng vì nó được kích hoạt sau. Mặt khác các giá trị cơ sở tin cậy lưu trong TPM không hoàn toàn thể hiện trạng thái của hệ điều hành khi phần mềm AM thực thi. Thực tế, các trình điều khiển thông thường sẽ được khởi động và hệ điều hành tiếp tục phần khởi động của mình. Như vậy, phần AM có thể tiếp tục tính toán giá trị cơ sở tin cậy bổ sung cho các trình điều khiển này tại pha 4.

Khởi động được bảo vệ đảm bảo phần mềm chống mã độc AM bằng cách cung cấp bản log tin cậy của toàn bộ các thành phần (chương trình) tham gia vào việc khởi động trước khi AM có quyền kiểm soát máy tính. Phần AM có thể sử dụng log này để xác định

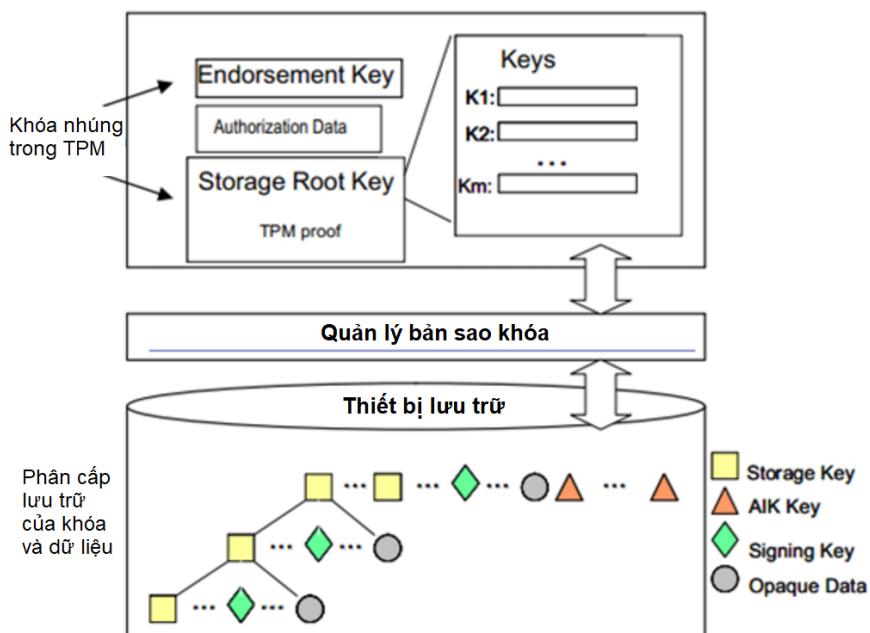
liệu có thành phần chạy trước nào là tin cậy chống lại thành phần bị nhiễm mã độc. Phần mềm kiểm soát mã độc sớm ELAM được hoạt động trước các thành phần do các nhà sản xuất thứ 3 cung cấp. Các trình điều khiển của AM được khởi động trước tiên và kiểm soát việc khởi tạo của các trình điều khiển khởi động và ngăn cản các trình điều khiển khởi động không rõ nguồn gốc.

3.3.2 Lưu trữ an toàn

a. Cơ chế lưu trữ an toàn TPM

Việc lưu trữ an toàn hay tin cậy thường sử dụng biện pháp mã hóa một phần hoặc toàn bộ thiết bị lưu trữ nhằm đảm bảo tính bí mật của dữ liệu lưu trên thiết bị. Một cách tiếp cận phổ biến là mã hóa dựa vào dịch vụ hệ thống file của hệ điều hành. Khi này TPM cung cấp khóa mã cho thiết bị được mã hóa và đóng dấu (*seal*) sao cho khóa mã chỉ được cung cấp khi có cấu hình hợp lệ. Điều này đảm bảo các thiết bị bị mất trộm không thể được giải mã do cấu hình (ngữ cảnh) chính xác để thực hiện việc này không tồn tại trong hệ thống của người tấn công. Máy tính vẫn khởi động song người tấn công cần các thông tin đăng nhập hợp lệ mới có thể truy nhập vào dữ liệu của thiết bị. Bitlocker là một chức năng tiêu biểu của Microsoft cho việc bảo vệ dữ liệu của người dùng. Quá trình mã hóa được hỗ trợ từ TPM minh họa trong hình vẽ dưới đây.

Khóa lưu trữ gốc SRK (*Storage Root Key*) được tạo ra bởi TPM và luôn nằm trong thiết bị này. Khóa này chỉ có thể truy nhập được khi cung cấp được dữ liệu bí mật còn được gọi là dữ liệu cấp phép SRK. Dữ liệu bí mật này tương tự như dữ liệu cấp phép chủ sở hữu và được nạp vào TPM cùng lúc trong quá trình xác lập quyền sở hữu. Cùng với dữ liệu cấp phép sở hữu, dữ liệu cấp phép SRK được mã hóa bởi khóa chứng thực EK (*Endorsement Key*) trước khi được gửi tới TPM. Khóa SRK tạo nên gốc cho việc phân cấp các khóa như trong hình.



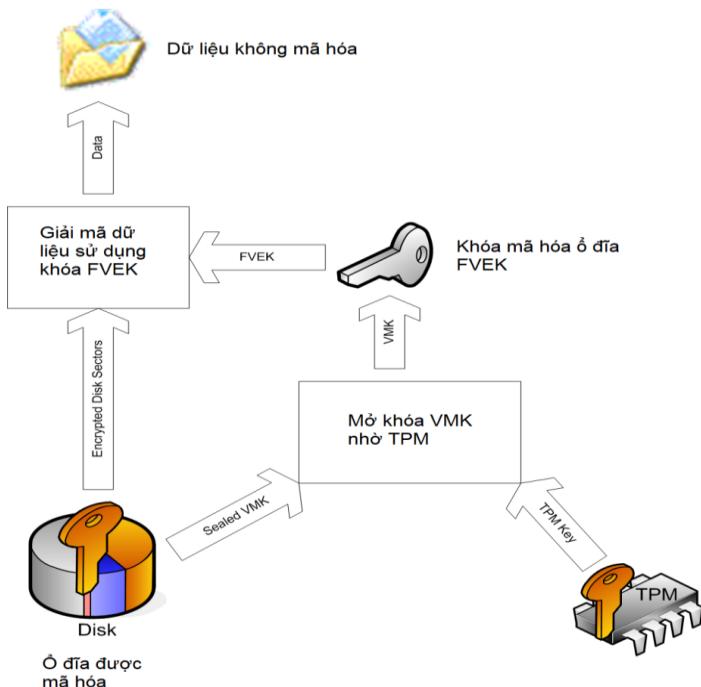
Hình 3-10. Bảo vệ và sử dụng khóa mã bằng TPM

Cơ chế phân cấp khóa này cho phép dữ liệu hay các khóa được mã hóa sao cho chúng chỉ có thể được giải mã thông qua TPM. Trong hình vẽ, các dữ liệu được mã hóa sử dụng khóa lưu trữ SK cụ thể. Khóa SK này cũng được mã hóa và lưu bên ngoài TPM. Để truy nhập dữ liệu, khóa mã lưu trữ SK được nạp vào trong TPM thông qua bộ quản lý bản sao khóa (*Key cache manager*) và giải mã bằng khóa lưu trữ gốc SRK. Do khóa SRK luôn bên trong TPM và TPM được gắn cứng vào máy tính nên dữ liệu mã hóa chỉ có thể được giải mã từ máy tính đó.

TPM cung cấp hai cơ chế khác cho việc lưu trữ an toàn là gắn (*binding*) và đóng dấu (*sealing*). Thao tác gắn mã hóa dữ liệu sử dụng khóa được quản lý như cách nêu trên. Quá trình đóng dấu mở rộng thêm bằng cách chỉ cho phép giải mã được tiến hành khi máy tính ở trạng thái xác định. Trạng thái này được thể hiện thông qua các dữ liệu lưu trong các thanh ghi PCR. Như vậy, khi dữ liệu được đóng dấu, không chỉ máy tính phải cùng cấu hình mà máy tính còn phải ở trạng thái xác định trước trước khi dữ liệu có thể được giải mã.

b. Mã hóa ổ đĩa Bitlocker

Microsoft cung cấp dịch vụ mã hóa ổ cứng kết hợp với TPM để tăng khả năng bảo vệ dưới tên gọi Bitlocker. Điều này đảm bảo không chỉ dữ liệu người dùng mà còn chống lại việc xâm nhập máy tính khi hệ thống không hoạt động. Bitlocker có khả năng kiểm chứng tính toàn vẹn của file cho khởi động và hệ thống trước khi giải mã vùng lưu trữ được bảo vệ. Cách thức bảo vệ dữ liệu trên ổ đĩa sử dụng TPM diễn ra theo các bước như dưới đây.



Hình 3-11. Quá trình giải mã ổ đĩa BitLocker

Khi Bitlocker được kích hoạt, toàn bộ nội dung của ổ đĩa dành cho hệ điều hành được mã hóa bằng khóa mã toàn bộ ổ đĩa FVMK (*full-volume encryption key*) và khóa này lại được bảo vệ bằng khóa mã chính VMK (*volume master key*). Việc bảo vệ khóa chính VMK sẽ giúp bảo vệ gián tiếp tính an toàn của dữ liệu trên ổ đĩa. Với sự hỗ trợ từ TPM, khóa VMK được đóng dấu và bảo vệ bằng phần cứng bên trong TPM. Việc truy nhập tới các dữ liệu bên trong ổ đĩa chỉ được phép khi TPM kiểm chứng thành công tính toàn vẹn của các phần tham gia vào quá trình khởi động máy tính. Cách kiểm chứng mặc định của TPM bảo vệ khóa VMK chống lại các thay đổi trong đoạn mã MBR, sector khởi động NTFS, và phần quản lý khởi động và các thành phần quan trọng khác như đĩa thẻ hiện trong quá trình khởi động bảo vệ.

Tất cả các khóa mã sử dụng trong quá trình khởi tạo của Bitlocker không truy nhập được từ phía người dùng cũng như không thể được thay đổi, sao chép hay rút lại.

Như vậy, TPM cung cấp cơ chế cho phép lưu trữ các khóa sử dụng cho BitLocker một cách an toàn chống lại việc sao chép và xâm nhập. Hệ thống file mã hóa có thể được dùng cùng với BitLocker để bảo vệ hệ điều hành khi chạy. Việc bảo vệ các file khởi các chương trình và người dùng trong hệ điều hành chỉ có thể được thực hiện bằng phần mềm mã hóa hoạt động bên trong hệ điều hành.

3.4 Phân tích an toàn các dịch vụ cơ bản hệ điều hành Windows và Unix/Linux

Mục tiêu của việc phân tích các đặc trưng an toàn của Windows và Unix/Linux nhằm làm sáng tỏ việc thiết kế hệ điều hành nhằm thực thi các chính sách an ninh là không đủ. Việc đảm bảo an ninh cần phải đầy đủ, bắt buộc, và kiểm chứng được. Việc đánh giá các hệ điều hành phổ biến có đạt được và đảm bảo các yêu cầu an ninh rất cần thiết. Hai cơ chế cơ bản cho việc đảm bảo an ninh cho các dịch vụ cũng như các chương trình người dùng đó là cơ chế xác thực và cơ chế bảo vệ truy nhập (cấp quyền truy nhập). Việc xâm phạm hay vô hiệu hóa hai cơ chế cốt lõi dẫn đến hậu quả rất tai hại cho hệ thống.

Bên cạnh việc phân tích và đánh giá các cơ chế đảm bảo an toàn cho Windows và Unix/Linux, phần này giới thiệu một cách tiếp cận khác trong việc phát triển hệ điều hành an toàn. Đó chính là phần nhân SCOMP.

3.4.1 Microsoft Windows

a. Giới thiệu

Windows bắt nguồn từ MS-DOS, một hệ điều hành rất hạn chế không hỗ trợ đa nhiệm và không khai thác hết các tính năng của kiến trúc x86. Kể từ 2000, Windows phát triển một cách độc lập với DOS và dựa vào nhân hệ điều hành mới hoàn toàn. Việc thiếu quan tâm đến vấn đề an toàn khiến cho hệ điều hành Windows (kể từ XP trở về trước) gặp phải những vấn đề nghiêm trọng về an toàn.

Windows ban đầu được thiết kế hướng tới người dùng máy vi tính PC riêng lẻ, không kết nối với mạng và vấn đề an toàn không được đặt ra với hệ thống như vậy. Mặt

khác, người dùng tự quản trị hệ thống của mình, việc cài đặt và chạy phần mềm theo nhu cầu của người dùng. Sự ra đời của dịch vụ Web khiến cho việc kết nối các máy tính Windows trở thành dịch vụ cơ bản và các dịch vụ mạng người dùng sử dụng như thư điện tử, duyệt Web, tải phần mềm dễ dàng làm lộ ra các điểm yếu mà Windows không được thiết kế để đối phó lại.

Windows sử dụng mô hình mở, mềm dẻo, cho phép người dùng tự quản trị khiến cho hệ điều hành này trở thành mục tiêu dễ dàng của người tấn công. Hơn thế, Microsoft khá chậm chạp trong việc xử lý các mối đe dọa này. Thực tế, Microsoft đã tập trung và có thành công bước đầu trong việc giảm thiểu lỗ hổng nhờ vào các qui định xây dựng mã chương trình tốt hơn, các công cụ phân tích mã và cấu hình hệ thống an toàn hơn. Tuy nhiên việc cải thiện các tính năng an toàn trên Windows trở nên kém hiệu quả. Các hệ thống kiểm soát truy nhập dựa trên Windows 2000 phức tạp và hầu như không sử dụng. Nền tảng tính toán tin cậy của Window quá lớn với hơn 50 triệu dòng lệnh chỉ trong phần hệ điều hành và các tính năng tăng cường của Vista không đủ để bảo vệ.

b. *Các cơ chế đảm bảo an toàn*

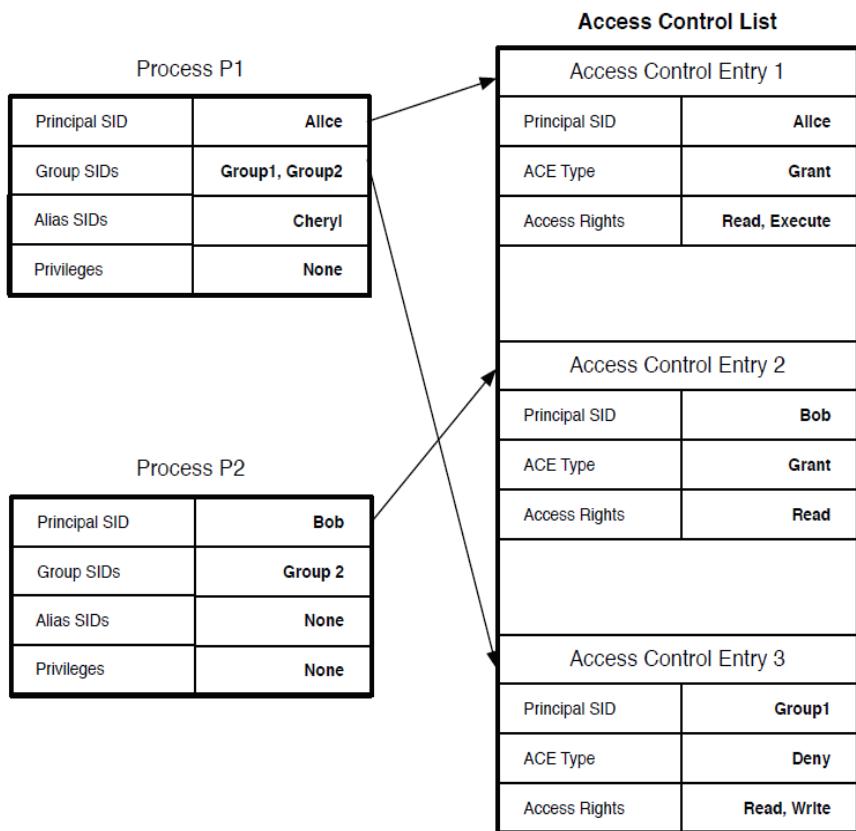
Cơ chế bảo vệ của Windows NT cung cấp mô hình kiểm soát truy nhập theo kiểu tùy chọn để quản lý các trạng thái bảo vệ, dán nhãn các đối tượng và dịch chuyển miền bảo vệ. Hệ thống bảo vệ của Windows có khả năng mở rộng và dễ biểu diễn các quyền cũng như là người dùng của hệ thống. Thực tế cho thấy, các cải thiện về tính mở rộng và biểu diễn có ảnh hưởng tiêu cực đến tính an toàn của hệ thống.

Các chủ thể của Windows cũng tương tự như Unix. Mỗi tiến trình được gán một thẻ (*token*) mô tả định danh của tiến trình. Định danh bao gồm định danh an ninh SID (*Security Identifier Descriptor*) của người dùng, SID nhóm, bí danh SID (Alias) để hoạt động bằng định danh SID khác, danh sách các quyền. Định danh Windows liên kết với người dùng song thẻ của tiến trình cho người dùng đó có thể bất kỳ tổ hợp quyền nào.

Các đối tượng cần kiểm soát trong Windows có thể là nhiều kiểu dữ liệu khác nhau bên cạnh các file. Các kiểu đối tượng mới được chương trình người dùng định nghĩa và thêm vào trong thư mục đóng (*active directory*). Từ góc độ kiểm soát truy nhập, các kiểu đối tượng được xác định thông qua các tập thao tác của chúng. Mô hình Windows hỗ trợ cách nhìn tổng quát lên các thao tác mà đối tượng có thể sở hữu. Windows mô tả tới hơn 30 thao tác với mỗi kiểu đối tượng bao gồm một số thao tác riêng biệt cho kiểu dữ liệu. Thậm chí cho các đối tượng file, Windows định nghĩa nhiều hơn các thao tác như truy nhập thuộc tính file, đồng bộ các thao tác. Ngoài ra, các ứng dụng có thể thêm các kiểu đối tượng mới và tự định nghĩa các thao tác lên đối tượng đó.

Windows sử dụng danh sách kiểm soát truy nhập ACL để đảm bảo an toàn. ACL bao gồm tập các mục kiểm soát truy nhập ACE (*Access control entry*). Mỗi mục mô tả các thao tác mà một SID có thể thực hiện trên đối tượng đó. Các hoạt động trong ACE được diễn giải dựa trên kiểu đối tượng. Trong Windows, các ACE có thể cho phép cũng như từ chối bất kỳ truy nhập nào.

Hình dưới đây biểu diễn ví dụ về ACL cho các tiến trình P1 và P2. Các mục ACE trong ACL cho biết các thao tác được cấp phép cũng như bị từ chối. Điểm chú ý là người dùng Alice thuộc về 2 nhóm khác nhau.



Hình 3-12. Danh sách kiểm soát truy nhập và thẻ của các tiến trình

c. Cơ chế xác thực

Các yêu cầu xác thực được xử lý bởi Bộ tham chiếu an toàn (*Security Reference Monitor – SRM*). SRM là phần mềm chạy trong nhân và nhận các tham số đầu vào thẻ tiến trình, SID của đối tượng và tập thao tác, trả về kết quả của yêu cầu truy nhập (chấp nhận/từ chối) trên cơ sở ACL mà nó tìm thấy. Do sử dụng quyền từ chối, cách thức SRM xử lý các truy vấn xác thực phức tạp hơn so với Unix. Sự khác biệt chủ yếu là thứ tự của các mục ACE. SRM tìm kiếm trong danh mục ACE cho đến khi tìm thấy ACE chấp thuận truy nhập hay ACE từ chối. Nếu ACE chấp thuận thao tác cần thiết thì yêu cầu được cho phép. Tuy nhiên nếu tìm thấy ACE từ chối mà chưa có thao tác được yêu cầu thì toàn bộ yêu cầu bị từ chối. Như trong ví dụ trên, tiến trình P1 sẽ chỉ được phép và thực thi đọc mà không được ghi vì yêu cầu nằm trong mục bị từ chối.

Bộ quản lý đối tượng đảm bảo việc đúng trung gian (ngăn chặn) cho các yêu cầu truy nhập. Các bộ quản lý đối tượng chạy trong nhân song các bộ phận này là các thực thể độc lập với nhau. Điều này mang lại lợi thế cho việc mô-đun hóa, song lại làm nảy sinh thách thức cho hệ thống trong việc ngăn chặn các yêu cầu. Điều cần thiết là các bộ quản lý đối

tượng cần ngăn chặn mọi thao tác và xác định một cách chính xác quyền cho những thao tác này. Không có quá trình nào đảm bảo điều này trong Windows.

Trong Windows, nền tảng tính toán tin cậy bao gồm toàn bộ các dịch vụ hệ thống và các tiến trình hoạt động sử dụng định danh người dùng tin cậy như *Administrator*. Windows cung cấp cơ chế tương tự như *setuid* để gọi các dịch vụ Windows để chạy với đặc quyền định trước, tối thiểu đủ để hỗ trợ tất cả người dùng. Như vậy, lỗ hổng trong các dịch vụ như vậy có thể dẫn đến hệ thống bị chọc thủng. Hơn thế, việc dễ dàng cài đặt phần mềm và độ phức tạp của mô hình kiểm soát truy nhập tùy chọn Windows thường khiến cho người dùng sử dụng tài khoản có đặc quyền, *Administrator*. Với các phiên bản sau này, mô hình của Windows được mở rộng để ngăn cản các chương trình tải về từ Internet được tự động cập nhập vào hệ thống bất kể định danh người dùng là gì. Mặc dù điều này đảm bảo mức độ toàn vẹn nhất định song không hoàn toàn bảo vệ tính toàn vẹn của hệ thống. Cải tiến này không ngăn cản việc gọi, các yêu cầu có mục đích xấu hay giả mạo các đoạn mã có độ an toàn *cao nhúng* trong file có độ toàn vẹn thấp.

Windows cung cấp cách thức hạn chế quyền cho các tiến trình một cách mềm dẻo còn được gọi là ngữ cảnh hạn chế (*restricted context*). Quyền để tiến trình hoạt động được là giao của ngữ cảnh hạn chế và các quyền bình thường của tiến trình. Do ngữ cảnh hạn chế có thể được gán một tập bất kỳ các quyền nên cơ chế này linh hoạt hơn Unix rất nhiều. Đồng thời, ngữ cảnh hạn chế được xây dựng vào trong hệ thống kiểm soát truy nhập nên nó ít bị mắc lỗi hơn. Tuy nhiên, các ngữ cảnh hạn chế khó cho người quản trị khi định nghĩa một cách chính xác nên chúng không được dùng rộng rãi.

d. *Đánh giá*

Mặc dù mô hình kiểm soát truy nhập của Windows mềm dẻo và có khả năng biểu diễn tốt, mô hình này không *thảo mãn* các yêu cầu của hệ điều hành an toàn. Tính mềm dẻo và khả năng biểu diễn khiến cho việc quản trị trở nên khó khăn hơn và không an toàn so với Unix/Linux.

Trong Windows, việc ngăn chặn (đứng trung gian) được các bộ quản lý đối tượng thực hiện song việc không có mã nguồn khiến cho rất khó để biết việc ngăn chặn được thực hiện ở đâu. Hơn thế, các bộ quản lý này có thể được mở rộng có thể tạo thêm các bộ phận không an toàn. Không có việc đánh giá chính tắc trong việc định nghĩa những thao tác và làm thế nào để an toàn với các bộ quản lý đối tượng này, sẽ không thể chắc chắn về việc ngăn chặn toàn bộ các yêu cầu truy nhập tới đối tượng.

Do Windows sử dụng cơ chế tùy chọn nên khi xét vấn đề chống xâm nhập, mô hình Windows chịu cùng điểm yếu của cơ chế này. Các tiến trình không tin cậy của người dùng có thể sửa đổi quyền tới các dữ liệu của người dùng một cách tùy ý. Như vậy, việc đặt mục tiêu an toàn với dữ liệu người dùng là không khả thi. Do người dùng thường *chạy bằng* tài khoản *Administrator* để thuận tiện cho việc quản trị, nên bất kỳ khía cạnh nào của hệ thống bảo vệ cũng có thể bị sửa đổi.

Thêm vào đó, các biện pháp bảo vệ nhân cũng hạn chế. Nhân của Windows có thể bị sửa đổi thông qua các mô-đun của nhân. Quá trình đóng dấu đoạn mã có thể dùng để chứng thực mô-đun nhân. Như vậy, người đóng dấu không nhất thiết phải là người viết mã. Người quản trị phải chịu trách nhiệm về tính tin cậy của người đóng dấu. Thủ tục an toàn phụ thuộc vào quyết định chủ quan của người dùng thường dễ bị lỗi do người dùng thường thiếu hiểu biết về những vấn đề như vậy. Mặt khác, nhân Windows không xác định các cách bảo vệ lời gọi hệ thống.

Nền tảng tính toán tin cậy của Windows không thực sự chặt chẽ. Gần như bất kỳ chương trình nào có thể nằm trong nền tảng tin cậy và bất kỳ tiến trình chạy chương trình này có thể sửa đổi các chương trình tin cậy khác làm mất hiệu lực của nền tảng tin cậy. Cũng giống Unix, bất cứ tiến trình tin cậy nào bị vô hiệu hóa có thể sửa đổi cơ chế bảo vệ làm mất hiệu lực các mục tiêu an toàn hệ thống và sửa đổi bản thân nhân của Windows qua các giao tiếp được cấp cho các tiến trình nằm trong cơ sở tính toán tin cậy để truy nhập tới các trạng thái của nhân.

Không giống Unix, Windows cung cấp giao diện lập trình cho phép xâm nhập các tiến trình khác như các hàm *CreateRemoteThread*, *OpenProcess* hay *WriteProcessMemory*. Các hàm thư viện này cho phép khởi tạo các luồng trong tiến trình khác hay chèn mã vào tiến trình trước khi khởi tạo luồng. Mặc dù các hàm này cần có đặc quyền để được sử dụng và phục vụ mục đích phát hiện lỗi của tiến trình. Song cần phải đảm bảo các đặc quyền này không bị lợi dụng để bảo đảm cho các cơ chế chống xâm nhập của cơ sở tính toán tin cậy.

Các cơ sở của tính đúng đắn trong Windows là không chính tắc và Windows có cơ sở tính toán tin cậy không bị giới hạn (kích cỡ) cũng như hệ thống nhân có khả năng mở rộng. Điều này khiến cho việc đánh giá chính tắc khó có kết quả. Mô hình khái quát của cơ chế bảo vệ của Windows cho phép mô tả các tổ hợp quyền nhưng lại không có bất cứ mục tiêu (yêu cầu) an toàn cụ thể được xác định trong hệ thống. Vì vậy, không thể nói được liệu hệ thống có an toàn hay không. Do mô hình Windows phức tạp hơn Unix và có thể mở rộng tùy ý nên việc kiểm chứng tính an toàn thậm chí còn khó khăn hơn.

e. Các lỗ hổng

Danh mục đăng ký (*Windows registry*) là cơ sở dữ liệu phân cấp toàn cục dùng để lưu dữ liệu cho toàn bộ các chương trình. Khi chương trình mới được nạp, nó có thể cập nhật danh mục đăng ký theo yêu cầu cụ thể của chương trình gồm các thông tin nhạy cảm như đường dẫn, thư viện. Trong khi mỗi mục đăng ký có thể được gắn với các ngữ cảnh an ninh và hạn chế truy nhập, song các hạn chế này không được sử dụng một cách hiệu quả.

Việc chỉ định rõ ràng tên các thư viện cần thiết cho ứng dụng không phải là cách mà các hãng phần mềm thường sử dụng một cách rộng rãi. Một cách tự nhiên, người dùng không muốn phần mềm mà họ mới mua lại không thể chạy một cách đúng đắn vì nó không có được thư viện cần thiết. Cũng như vậy, hãng sản xuất phần mềm không biết

được cách thức quản trị Windows của người dùng nên giải pháp tình thế là mở các quyền để chắn chắn phần mềm sẽ chạy bất kể người dùng quản trị thế nào. Như vậy, nếu mục đăng ký này sau này được sử dụng bởi người tấn công để vô hiệu hóa Windows, thì không hẳn là do vấn đề của phần mềm của nhà cung cấp.

Người dùng quản trị. Trong Windows người dùng thông thường chạy dưới định danh quản trị hay với các đặc quyền được chấp thuận. Điều này là vì người dùng cần quyền truy nhập rộng hơn để có thể sử dụng các chức năng cần thiết cho phép hệ thống hoạt động. Nếu người dùng tải về phần mềm trò chơi, người dùng có thể cần đặc quyền để cài đặt và chắc chắn cần các đặc quyền khi chạy phần mềm trò chơi sử dụng nhiều tài nguyên (phần cứng). Cuối cùng, người dùng có thể muốn kiểm chứng lý do phần mềm trò chơi không chạy và cho phép tất cả các đặc quyền để xử lý vấn đề. Unix thường được dùng bởi người dùng có nhiều kinh nghiệm hơn và hiểu rõ sự khác biệt giữa cài đặt và các hoạt động thông thường của máy tính. Như vậy, việc vận dụng đặc quyền hợp lý hơn người dùng Windows.

Cho phép ngầm định là cách triển khai Windows cho phép các quyền cũng như các chương trình được hoạt động một cách đầy đủ. Điều này tạo ra sâu Code Red nổi tiếng tấn công vào phần mềm chủ SQL hoạt động trong máy chủ Web IIS của Microsoft. Rất nhiều người chạy IIS mà không biết tính năng được bật.

3.4.2 Unix và Linux

Unix được viết bằng ngôn ngữ C giúp cho nó trở thành hệ điều hành đầu tiên có tính khả chuyển (chạy được trên nhiều phần cứng khác nhau) khởi nguồn từ Bell Labs của AT&T từ những năm 70 của thế kỷ 20 và thu hút được cộng đồng phát triển đông đảo. Bên cạnh đó, Unix có giao diện chương trình (API) thuận tiện cho người phát triển.

Unix hướng đến chương trình căn bản nhỏ gọi là nhân (kernel) với giao diện chuẩn để đơn giản hóa việc phát triển ứng dụng. Mục tiêu thiết kế của Unix là phát triển nền tảng chung có thể chia sẻ dễ dàng giữa các người dùng với nhau. Như vậy, mục tiêu an toàn của Unix là bảo vệ dữ liệu người dùng khỏi các lỗi vô tình trong chương trình người dùng. Tuy nhiên, việc bảo vệ này không đảm bảo yêu cầu về tính bí mật và toàn vẹn. Cơ chế an toàn Unix nhằm bảo vệ người dùng với nhau và hạ tầng tính toán tin cậy của hệ thống khỏi toàn bộ các người dùng.

Hệ thống Unix bao gồm nhân hệ điều hành và các tiến trình. Lớp bảo vệ được sử dụng để ngăn cách phần nhân với các tiến trình khác. Mỗi tiến trình có không gian địa chỉ riêng xác định các địa chỉ bộ nhớ được phép truy nhập. Các hệ thống Unix hiện đại coi các không gian địa chỉ như là các trang nhớ. Unix sử dụng khái niệm file cho tất cả các đối tượng lưu trữ hệ thống như lưu trữ thứ cấp (ổ đĩa), thiết bị vào/ra, mạng và liên lạc giữa các tiến trình. Mỗi tiến trình Unix được liên kết với một định danh căn cứ vào người dùng sử dụng tiến trình đó và việc truy nhập tới file bị hạn chế bởi định danh tiến trình.

Mục tiêu an toàn của Unix là bảo vệ người dùng với nhau và bảo vệ các chương trình tin cậy của hệ thống với người dùng. Các chương trình tin cậy trong Unix là các chương trình chạy bằng định danh *root* hay *superuser*. Các tiến trình *root* hay nhân có toàn quyền truy nhập hệ thống.

Linux có thể coi là biến thể của UNIX mà thu hút được sự chú ý trong thời gian vừa qua từ cách thiết bị nhỏ như điện thoại di động đến các máy chủ hay siêu máy tính. Linux rất giống với bất kỳ hệ thống Unix nào khác. Thực tế, mục tiêu thiết kế quan trọng của Linux là tương thích với Unix. Các tính năng an toàn và cơ chế bảo vệ của Lunix cùng sử dụng nguyên tắc như trong Unix vì vậy trong phần dưới đây không phân tách riêng hai hệ thống.

a. *Hệ thống bảo vệ*

Hệ thống bảo vệ: sử dụng cơ chế kiểm soát truy nhập tùy chọn. Một cách khái quát, hệ thống bảo vệ mô tả các trạng thái bảo vệ và các thao tác cho phép các tiến trình sửa đổi các trạng thái đó. Tuy nhiên, hệ thống bảo vệ cho phép mô tả việc chuyển trạng thái tức là cho phép tiến trình thay đổi giữa các miền bảo vệ (từ mức có đặc quyền sang mức không đặc quyền và ngược lại). Các tính năng bảo vệ này không đủ thỏa mãn các yêu cầu chặt chẽ về hệ điều hành an toàn.

Trạng thái bảo vệ cho biết các thao tác mà chủ thể của hệ thống có thể thực hiện lên các đối tượng. Các trạng thái bảo vệ Unix/Linux liên kết với định danh tiến trình (chủ thể) với các truy nhập tới file (đối tượng). Mỗi tiến trình bao gồm định danh người dùng, nhóm, và các nhóm phụ. Tất cả các tài nguyên trong hệ thống được biểu diễn như các file. Các trạng thái bảo vệ mà chủ thể có thể thực hiện là đọc, ghi và thực thi lên file. Các file cũng được gắn với định danh người dùng và nhóm của chủ sở hữu mà nó cho phép các đặc quyền của tiến trình khi truy nhập vào file đó. Tiến trình với định danh chủ sở hữu có thể sửa đổi bất cứ khía cạnh nào của các trạng thái bảo vệ với file đó. Tập hạn chế các đối tượng và thao tác sử dụng danh sách kiểm soát truy nhập ngắn gọi là bít chế độ. Các bít chế độ xác định các quyền của 3 nhóm chủ thể: file chủ sở hữu UID, file trong nhóm GID với chủ sở hữu, và nhóm còn lại.

Việc kiểm soát truy nhập được thực hiện thông qua các bít chế độ. Nếu định danh của người dùng trùng với UID thì các bít chế độ UID xác định quyền truy nhập. Cũng tương tự khi định danh nhóm của tiến trình hay nhóm còn lại phù hợp thì bít chế độ tương ứng giúp xác lập quyền truy nhập.

| Name | Owner | Group | Mode Bits |
|------|---------|----------|-----------|
| foo | alice | faculty | rwxr--r-- |
| bar | bob | students | rw-rw-r-- |
| baz | charlie | faculty | rwxrwxrwx |

Hình 3-13. Cơ chế bảo vệ bằng bít chế độ

Hình trên đây minh họa cho việc sử dụng các bít chế độ. Dòng đầu tiên trong cột bít chế độ rwxr--r--—cho thấy *alice* có đầy đủ các quyền đọc, ghi, và thực thi lên đối

tượng *foo*. Trong khi đó *nhóm faculty* và các nhóm khác chỉ có quyền đọc mà không được ghi và thực thi lên file.

Hệ thống bảo vệ theo kiểu này là hệ thống kiểm soát truy nhập tùy chọn. Các bít chế độ của người sở hữu, nhóm, và nhóm còn lại có thể bị thay đổi bởi bất cứ tiến trình nào đang chạy sử dụng định danh người dùng. Điều này sẽ gây rủi ro khi chương trình người dùng sử dụng không đáng tin cậy như chương trình tải về từ Internet.

Các bít chế độ cũng chứa đựng cách thức dịch chuyển miền bảo vệ được gọi là *bit setuid*. Khi bít này được thiết lập trên 1 file, bất cứ tiến trình nào được thực thi file đó được chuyển một cách tự động thành chủ sở hữu file và nhóm sở hữu.

b. Hệ thống xác thực

Hệ thống xác thực kiểm soát việc truy nhập của các tiến trình tới các file và thực hiện việc dịch chuyển miền bảo vệ cho phép người dùng thay đổi định danh. Hệ thống này nằm ở trong nhân song phụ thuộc vào các tiến trình bên ngoài (hệ thống hay người dùng) để xác định các yêu cầu xác thực và trạng thái bảo vệ. Quá trình xác thực diễn ra mỗi khi có yêu cầu truy nhập file và thao tác được phép trên file đó sẽ được thẩm tra. Tiến trình yêu cầu cung cấp tên file và thao tác cần được thực hiện tại thời điểm lời gọi hệ thống được thực hiện. Nếu được chấp thuận, hệ thống tạo ra thẻ mô tả file biểu diễn truy nhập được phép của tiến trình để thực hiện các thao tác trong tương lai lên file đó. Các thẻ mô tả file được lưu trong phần nhân và chỉ có chỉ mục được trả về cho tiến trình. Tiến trình người dùng cung cấp chỉ mục này cho nhân hệ thống mỗi khi cần truy nhập file.

Hệ thống xác thực kiểm soát các thao tác file bằng cách ngăn chặn thao tác mở file cho các quyền đọc, ghi, và thực thi. Như vậy, việc sử dụng các quyền này không phải lúc nào cũng cho phép việc kiểm soát phù hợp và không phải đối tượng nào cũng có thể biểu diễn dưới dạng file như việc truy nhập tới các dữ liệu *meta* hay các liên lạc mạng.

Mặt khác, hệ thống xác thực dựa vào các dịch vụ xác thực mức người dùng để xác định định danh của tiến trình. Khi người dùng đăng nhập vào hệ thống, các tiến trình được gán định danh phù hợp của người dùng. Tất cả các tiến trình sau đây đều thừa kế định danh đăng nhập của người dùng trừ phi có sự thay đổi miền bảo vệ. Những dịch vụ mức người dùng này cũng cần đặc quyền *root* để thay đổi định danh tiến trình cũng như thực thi một số công việc của chúng. Trong khi đó, phần nhân tin cậy lại phải bao gồm tất cả tiến trình *root*. Điều này tạo ra lỗ hổng cho các phần mềm an toàn quan trọng cũng như là nhân.

Hệ thống có thể cung cấp người dùng đặc biệt *nobody* mà không sở hữu file nào cũng như thuộc về nhóm nào cả. Như vậy, các tiến trình có thể bị hạn chế bởi người dùng đặc biệt này. Tuy nhiên, chủ thẻ *nobody* vẫn có các quyền khác và các bát cẩn trong việc sử dụng đặc quyền như lệnh *chroot* đều có thể làm rò rỉ hay phát sinh thêm quyền cho chủ thẻ *nobody*.

c. *Đánh giá*

Giao tiếp bộ giám sát tham chiếu bao gồm các mốc (*hook*) để kiểm tra truy nhập tới file tại một số lời gọi hệ thống. Vấn đề ở chỗ tập hạn chế các thao tác (đọc, ghi, và thực thi) không đủ khả năng biểu diễn để kiểm soát các truy nhập thông tin. Mặt khác, cơ chế bảo vệ cho phép sửa đổi file mà không cần quyền ghi.

Cơ chế xác thực không đảm bảo ngăn chặn toàn bộ tới tất cả tài nguyên hệ thống. Với một số đối tượng như mạng, không có cơ chế xác thực nào được triển khai. Do giao tiếp bộ giám sát tham chiếu được đặt tại nơi mà các thao tác nhạy cảm với an ninh được thực hiện, rất khó có thể biết liệu toàn bộ các thao tác được xác định và toàn bộ các đường dẫn được ngăn chặn. Không có cách tiếp cận nào được sử dụng để thẩm tra việc này.

Cho dù các cơ chế bảo vệ và bộ giám sát truy nhập được đặt vào phần nhân nhưng điều này không đảm bảo chống lại việc xâm nhập. Trước hết, cơ chế bảo vệ là tùy chọn nên cơ chế này có thể bị xâm nhập bởi bất kỳ tiến trình nào. Các tiến trình không tin cậy của người dùng có thể sửa đổi quyền tới các dữ liệu của họ một cách tùy ý nên việc bắt tuân thủ các mục tiêu an toàn với dữ liệu người dùng là không thể. Mặc dù sử dụng lớp bảo vệ, song các nhân Unix thường sử dụng các thủ tục để kiểm tra các tham số của lời gọi hệ thống, những thủ tục như vậy có thể bị đặt không đúng chỗ. Các tiến trình người dùng có nhiều kiểu giao tiếp để truy nhập và sửa đổi nhân phía trên và ngoài lời gọi hệ thống từ khả năng cài đặt các mô-đun của nhân đến các hệ thống file đặc biệt như */proc* hay *sysfs* để truy nhập trực tiếp vào bộ nhớ của nhân. Việc đảm bảo các giao tiếp này chỉ được truy nhập bởi các đoạn mã tin cậy là không khả thi.

Bên cạnh phần nhân, cơ sở tính toán tin cậy của Unix/Linux bao gồm toàn bộ các tiến trình chạy với định danh *root* kể cả những tiến trình người dùng được đăng nhập bằng người dùng *root*. Như vậy các tiến trình này có thể chạy bất kỳ chương trình nào nên việc khẳng định chống xâm nhập của cơ sở tính toán tin cậy là không thể. Ngoài ra, phần tính toán tin cậy quá lớn và đối mặt với quá nhiều mối đe dọa để chống lại việc xâm nhập. Nếu có một trong số các tiến trình bị phá vỡ, hệ thống Unix/Linux rốt cuộc bị phá hủy vì không biện pháp bảo vệ nào với các tiến trình *root*.

Các cơ sở cho tính đầy đủ của hệ thống Unix/Linux là không chính xác. Kích cỡ thực sự của cơ sở tính toán tin cậy không cho phép thực hiện bất kỳ việc kiểm nghiệm chính xác hiệu quả. Hơn thế, kích cỡ và khả năng mở rộng của nhân càng làm cho việc kiểm chứng tính đúng đắn không khả thi.

d. *Các lỗ hổng tiêu biểu*

Unix có một vài tiến trình *root* làm nhiệm vụ duy trì các cổng mạng mở cho tất cả các bên ở xa như *ssh*, *ftp*, *sendmail* ... gọi là các dịch vụ mạng (*Network-facing deamons*). Để duy trì tính toàn vẹn của cơ sở tính toán tin cậy và đạt được tính đảm bảo cho giám sát tham chiếu, những tiến trình này cần tự bảo vệ khỏi những đầu vào như vậy nghĩa là các tiến trình không tin cậy. Tuy nhiên, một vài lỗ hổng đã được báo cáo

đặc biệt do vấn đề tràn bộ đệm cho phép người tấn công phá vỡ hệ thống tính toán tin cậy. Một vài dịch vụ đã được thiết kế lại để loại bỏ những lỗ hổng như Postfix để thay thế sendmail, nhưng việc đánh giá đầy đủ của các bảo vệ cho các dịch vụ này vẫn chưa được cung cấp. Như vậy, việc bảo vệ tính toàn vẹn của dịch vụ mạng không hoàn chỉnh và mang tính tình thế.

Rootkit. Hệ thống Unix/Linux hiện đại cho phép mở rộng nhân qua các mô-đun mà có thể được nạp một cách linh hoạt vào nhân. Tuy nhiên, các mô-đun bị lỗi hay xấu có thể cho phép người tấn công thực thi mã bên trong nhân với đặc quyền đầy đủ của hệ thống. Một loạt các gói phần mềm độc hại, gọi là *rootkit*, đã được tạo ra để tận dụng ưu thế của mô-đun nhân hay các giao tiếp khác tới các tiến trình *root*. Các rootkit như vậy cho phép thực thi các hàm của người tấn công và cách thức che dấu việc phát hiện. Cho dù có nhiều cố gắng phát hiện mã độc trong nhân song những rootkit này khó phát hiện.

Các biến môi trường có sẵn cho các tiến trình để chuyển tải trạng thái qua các chương trình khác nhau. Như biến LIBPATH cho phép xác định trật tự tìm kiếm thư viện liên kết động. Lỗ hổng phổ biến là người tấn công có thể sửa đổi LIBPATH để nạp các file do mình tạo ra như là một bộ phận của thư viện động. Do các biến này được thửa hưỡng lại khi tiến trình con sinh ra nên một tiến trình không tin cậy có thể gọi được chương trình tin cậy. Nếu tiến trình tin cậy dựa vào biến môi trường mà không riêng cho mình LIBPATH thì nó có thể bị rủi ro chạy phải các đoạn mã độc.

Tài nguyên chia sẻ giữa tiến trình tin cậy với chương trình không tin cậy có thể tạo ra lỗ hổng để tấn công. Vấn đề thường thấy với thư mục tạm */tmp*. Vì bất kỳ tiến trình nào có thể tạo file trong thư mục này, tiến trình không tin cậy có thể tạo file và cấp quyền truy nhập cho các chương trình khác, đặc biệt tiến trình tin cậy, truy nhập tới các file này. Sau đó, tự nó có quyền truy nhập tới các file của các chương trình tin cậy. Các tiến trình tin cậy cần phải thận trọng khi sử dụng bất kỳ tài nguyên được chia sẻ bởi các tiến trình không tin cậy.

Thời điểm kiểm tra tới thời điểm sử dụng TOCTTOU (Time-of-Check-to-Time-of-Use) là tình trạng các tiến trình không tin cậy có thể thay đổi trạng thái của hệ thống giữa thời điểm của một thao tác được phép tới thời điểm mà thao tác đó được thực hiện. Nếu việc thay đổi như vậy cho phép tiến trình không tin cậy truy nhập tới một file mà nó đáng lẽ không được cho phép thì việc này làm xuất hiện một lỗ hổng. Ví dụ cổ điển nhất tiến trình *root* dùng lời gọi hệ thống *access* để xác định liệu người dùng mà chạy tiến trình có truy nhập tới một file cụ thể như */tmp/X*. Tuy nhiên, sau khi lời gọi hệ thống *access* cho phép truy nhập file và trước khi file được mở, người dùng có thể thay đổi việc gắn tên file và đổi tượng file cụ thể được truy nhập (*inode*). Việc này được hoàn tất bằng cách thay đổi file */tmp/X* thành liên kết tới file đích như là */etc/shadows*. Kết quả, Unix thêm một cờ để yêu cầu *open* có thể ngăn chặn việc duyệt nội dung thư mục qua các liên kết. Tuy nhiên, hệ thống file vẫn còn tồn tại nguy cơ bị tấn công kiểu này do việc ánh xạ giữa tên file và đối tượng file (lưu trữ) có thể bị điều chỉnh bởi các tiến trình không an toàn.

Do sử dụng có chế bảo vệ tùy chọn, kích cỡ của cơ sở tính toán tin cậy và các lỗ hổng như kể trên việc chuyển đổi Unix sang thành hệ điều hành an toàn thực sự là thách thức lớn.

3.4.3 Mô-đun an toàn cho Linux

a. Giới thiệu

Vào những năm đầu 2000, mức độ nghiêm trọng của sâu, mã độc và tấn công qua mạng ở mức báo động. Cộng đồng phát triển Linux nhận thấy cần phải có cơ chế đảm bảo an toàn chặt chẽ và chắc chắn cho Linux. Cuối cùng, thiết kế an toàn cho phép lựa chọn nhiều mô-đun khác nhau qua một giao tiếp thống nhất được lựa chọn. Đó chính là bộ khung các mô-đun an toàn cho Linux LSM (*Linux Security Modules*). Về cơ bản khung LSM là hệ thống giám sát tham chiếu cho nhân Linux và bao gồm hai phần: giao tiếp bộ giám sát tham chiếu được tích hợp vào nhân cơ bản của Linux và mô-đun giám sát tham chiếu thực hiện các chức năng của bộ giám sát như xác thực, kho chính sách phía sau giao tiếp LSM.

Mục tiêu thiết kế của khung LSM như sau:

- Giao tiếp với bộ giám sát tham chiếu phải thực sự khai quát để cho việc sử dụng các mô hình an toàn khác nhau chỉ thuận túy là việc nạp các nhân khác nhau.
- Giao tiếp cần phải đơn giản về khái niệm, tối giản và hiệu quả
- Cần hỗ trợ chuẩn POSIX.1e về kiểm soát truy nhập dựa trên năng lực

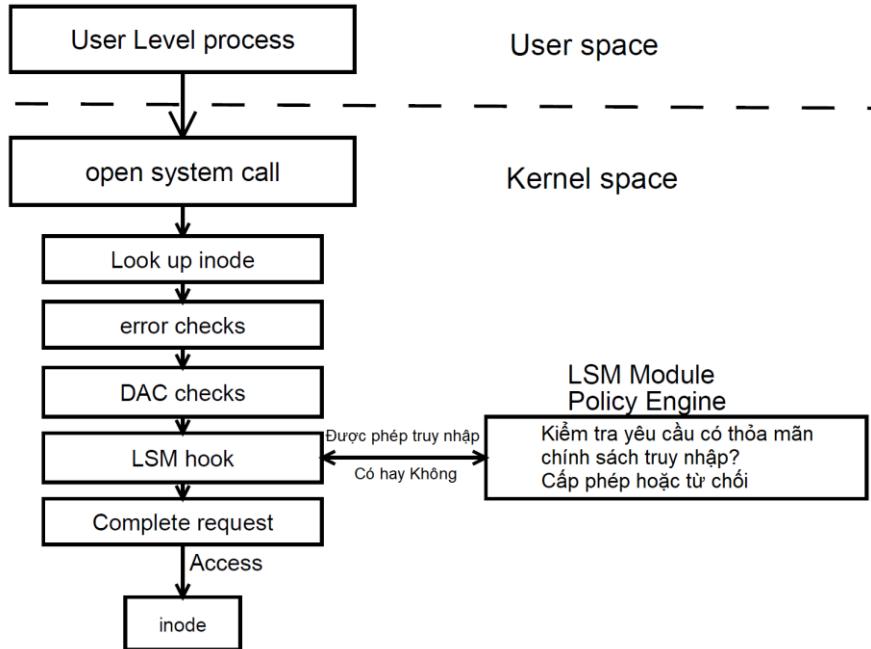
Hai yêu cầu đầu tiên bắt nguồn từ việc kết hợp tất cả các truy vấn xác thực từ các vấn đề an ninh Linux Từ trước sao cho toàn bộ các mô-đun có thể được hỗ trợ nhưng hạn chế số lượng các truy vấn xác thực nhiều nhất có thể được để ngăn chặn các truy vấn dư thừa mà chúng làm tăng độ phức tạp và ảnh hưởng đến hiệu năng. Thiết kế giao tiếp LSM được xây dựng thủ công và kiểm chứng nhờ công cụ phân tích mã nguồn về tính hoàn chỉnh và nhất quán. Việc đánh giá hiệu năng của các mô-đun cũng được thực hiện nhằm chắc chắn về hiệu năng chung của hệ thống.

Khung LSM được chính thức thêm vào nhân Linux từ phiên bản 2.6 và biến đổi Linux cũng như Unix phần nào thành hệ thống thỏa mãn các yêu cầu của bộ giám sát tham chiếu. Dù vậy đây vẫn là công việc còn nhiều khó khăn.

b. Triển khai

Việc triển khai bộ khung LSM gồm ba phần: định nghĩa giao tiếp bộ giám sát tham chiếu; việc bố trí giao tiếp này; và việc triển khai cụ thể của bộ giám sát tham chiếu. Về cơ bản, giao tiếp LSM là bảng các hàm, mà được gắn vào các lời gọi một cách ngầm định, thực hiện các phép kiểm tra truy nhập tùy chọn truyền thông DAC. Người viết mô-đun chịu trách nhiệm triển khai các hàm được quan tâm. Như vậy mô-đun LSM được tổ chức theo kiểu ngăn xếp mà các lớp trước cung cấp giao tiếp để mô-đun LSM phía sau

được nạp. Mô-đun đầu tiên chịu trách nhiệm soạn ra các quyết định truy nhập mà các quyết định này dựa vào mô-đun ở phía sau.



Hình 3-14. Kiến trúc giao tiếp LSM

Định nghĩa giao tiếp LSM mô tả cách thức nhân Linux có thể gọi bộ giám sát tham chiếu LSM. Đây là tập con trỏ tới các hàm (*function pointer*) mà dùng để gọi các hàm khi LSM được nạp. Các con trỏ hàm được gọi các mốc LSM (*LSM hook*). Về cơ bản các mốc LSM này tương ứng với các truy vấn cấp phép, tuy nhiên giao tiếp LSM cũng chứa các mốc cho các nhiệm vụ khác như gán nhãn an ninh, dịch chuyển hay duy trì các nhãn an ninh. Có khoảng hơn 150 các mốc LSM dùng cho các công việc an ninh.

Bố trí giao tiếp của bộ tham chiếu LSM. Thách thức chủ yếu trong thiết kế bộ khung LSM là vị trí đặt các mốc LSM. Hầu hết các mốc LSM liên kết với lời gọi hệ thống cụ thể, như vậy với các mốc LSM này được đặt tại đầu vào của lời gọi hệ thống. Tuy nhiên, một số mốc LSM không thể đặt tại vị trí như vậy. Ví dụ như trong hình về kiến trúc giao tiếp LSM, lời gọi hệ thống chuyển đường dẫn của file thành thẻ mô tả file mà cho phép truy nhập tới file đó. Xác định vị trí file cụ thể được mô tả bởi đường dẫn cần cho phép truy nhập tới các thư mục theo đường dẫn, bất kỳ các liên kết file, và cuối cùng là cho phép truy nhập vào file với thao tác cụ thể. Do các thành phần này được trích ra từ đường dẫn tạo các điểm khác nhau trong quá trình mở file *open*, vậy vị trí đặt các mốc LSM không đơn giản.

Trong khi có một số phép kiểm tra tùy chọn giúp cho việc xác định vị trí đặt các mốc LSM cho các thao tác *open*, quá trình đặt các mốc LSM nói chung mang tính tình thế. Với những chương trình chưa từng thực hiện việc cấp phép truy nhập tùy chọn thì phải thực hiện việc đặt các mốc LSM thủ công. Mốc LSM được chèn vào đoạn mã theo kiểu

trực tiếp (*inline*) và được liên kết tại thời điểm biên dịch. Việc này giúp cho các đoạn mã dễ theo dõi hơn.

Triển khai bộ giám sát tham chiếu. Thực tế, LSM bao gồm AppArmor (*Application Armor*), hệ thống phát hiện xâm nhập IDS (*Intrusion detection system*), SELinux (*Security-Enhanced Linux*) và danh sách năng lực POSIX. Mỗi một mô-đun LSM cung cấp cách thức khác nhau với việc kiểm soát truy nhập bắt buộc ngoại trừ danh sách năng lực POSIX là cơ chế kiểm soát tùy chọn đã có trong Linux. Việc chuyển POSIX thành mô-đun nhằm cho phép phát triển độc lập với nhân cơ bản và vì một số mô-đun LSM triển khai việc kiểm soát năng lực theo cách khác.

AppArmor là hệ thống kiểm soát truy nhập bắt buộc mà mô hình đe dọa tập trung vào môi trường Internet. Nếu hệ thống được cấu hình một cách đúng đắn thì Internet là cách thức duy nhất mà các đầu vào có mục đích xấu có thể chạm đến hệ thống. Một mối đe dọa chính là các dịch vụ mạng, chúng có nguy cơ nhận các dữ liệu đầu vào không đảm bảo như dữ liệu tràn bộ đệm. AppArmor sử dụng các chính sách hạn chế cho các dịch vụ mạng như vậy để ngăn chặn các dịch vụ bị xâm hại làm vô hiệu cả hệ thống.

Hệ thống phát hiện xâm nhập IDS hướng tới việc ngăn chặn xâm nhập dưới dạng hệ thống kiểm soát truy nhập. Hệ thống này thực hiện việc quản lý truy nhập dựa vào các chính sách mô tả file nào mà chương trình có thể được truy nhập.

SELinux triển khai việc kiểm soát truy nhập bắt buộc dựa trên kiến trúc kiểm soát truy nhập linh hoạt Flask bao gồm kho chính sách và phần nhân cho việc giám sát truy nhập. SELinux hỗ trợ mô hình an toàn dựa theo vai trò RBAC (*Role-Based Access Control*) hay an toàn nhiều mức. SELinux có thể hạn chế các chương trình người dùng theo nguyên tắc ít đặc quyền nhất, bảo vệ tính toàn vẹn, hay tính bí mật của chương trình và dữ liệu, cũng như các nhu cầu an ninh của chương trình. Tính khái quát và đầy đủ của SELinux góp phần thúc đẩy các yêu cầu đối với LSM.

Danh sách năng lực POSIX được triển khai trong nhân cơ bản của Linux, song LSM tách biệt rạch ròi chức năng này vào trong mô-đun an ninh. Việc này cho phép người dùng không cần chức năng này có thể loại bỏ nó khỏi nhân và cho phép việc phát triển kiểm soát theo năng lực độc lập với phần nhân cơ bản.

c. **Đánh giá**

Về tính ngăn chặn đầy đủ, giao tiếp với bộ giám sát tham chiếu của khung LSM được thiết kế để cấp phép truy nhập tới đối tượng (tài nguyên) cụ thể được dùng bởi nhân trong các thao tác nhạy cảm với an ninh để ngăn cản các lỗ hổng. Ngoài ra, khung LSM đứng ngăn chặn các thao tác được xác định bởi cộng đồng phát triển LSM tới các thao tác nhạy cảm. Cơ chế trung gian (ngăn chặn) thực tế là kết hợp toàn bộ các mẫu giám sát tham chiếu được xây dựng.

Do các giao tiếp LSM được thiết kế theo kiểu không chính tắc, việc kiểm chứng với việc ngăn chặn đầy đủ là cần thiết. Các công cụ phân tích mã nguồn được xây dựng để

kiểm chứng các cấu trúc dữ liệu nhân nhạy cảm với an ninh được ngăn chặn theo cách nhất quán. Các lỗi được phát hiện được sửa. Tuy nhiên, các công cụ này chỉ là xấp xỉ gần đúng các yêu cầu về ngăn chặn đầy đủ và chúng không được sử dụng thường xuyên. Tuy nhiên, không có lỗi nào trong phân bố trí giao tiếp của bộ giám sát truy nhập được phát hiện thêm kể từ khi được triển khai.

Bộ giám sát truy nhập như SELinux được hoạt động trong lớp bảo vệ mức quản trị (*supervisor*) nên chúng được bảo vệ như là nhân hệ thống. Mặc dù, các mô-đun LSM có giao tiếp nhưng chúng được biên dịch vào trong nhân và được kích hoạt tại thời điểm khởi động. Nhân Linux có thể truy nhập qua lời gọi hệ thống như hệ thống file đặc biệt, các file thiết bị vậy việc truy nhập tới những có chế như vậy cần phải chắc chắn chống xâm nhập. Mặc dù Linux không cung cấp cơ chế cổng (*gate*) cho phép lọc dữ liệu đầu vào, song Linux đã thực hiện việc phân tích mã nguồn với việc xử lý dữ liệu đầu vào. Hơn thế, hệ thống Linux cung cấp các thao tác khác cho phép truy nhập tới bộ nhớ nhân. Mặt khác SELinux có thể được cấu hình để hạn chế các truy nhập tới các tiến trình tin cậy.

Nói chung việc kiểm chứng tính đúng đắn của cá biện pháp thực thi an ninh là việc khó khăn nhất. Với mã nguồn lớn, được viết bằng ngôn ngữ không an toàn và bởi nhiều người, việc kiểm chứng là không thể hoàn thành trên thực tế. Linux đã được mức đánh giá theo tiêu chuẩn phổ quát EAL4. Mức này đòi hỏi việc lập tài liệu thiết kế mức thấp của nhân. Dù vậy việc kiểm chứng các đoạn mã tuân thủ các mô hình thiết kế có lẽ là việc không thực tế.

Các chính sách SELinux định nghĩa các yêu cầu bắt buộc và chính xác các thao tác được phép trong hệ thống. Như vậy, hoàn toàn có thể xây dựng luồng thông tin từ các chính sách này và thậm chí cả các trạng thái dịch chuyển. Cũng như vậy, các chính sách an ninh nhiều mức đảm bảo các luồng thông tin thỏa mãn các thuộc tính an toàn cơ bản và nâng cao. Như vậy, cách tiếp cận SELinux cho phép hệ thống được an toàn, song người phát triển hệ thống cần phải quản lý việc sử dụng đoạn mã tin cậy một cách cẩn thận để chắc chắn các mục tiêu an toàn thực sự đạt được.

3.4.4 Nhân SCOMP

SCOMP (*Secure communications Processor*) là hệ thống dựa trên nhân an toàn được thiết kế để triển hai các yêu cầu về an ninh theo nhiều mức. Với yêu cầu về hiệu năng và vấn đề an toàn, người thiết kế SCOMP không chỉ xây dựng nhân an toàn mà còn xây dựng các cơ chế phần cứng và giao tiếp ứng dụng mới để lập trình với nhân.

Phần tính toán tin cậy của SCOMP bao gồm 3 bộ phận hoạt động ở lớp 0,1 và 2. Hệ điều hành tin cậy bao gồm nhân an toàn hoạt động ở lớp 0 và các phần mềm tin cậy ở lớp 1. Các chức năng của gói giao tiếp nhân hoạt động ở lớp 2. Các ứng dụng truy nhập các tài nguyên được bảo vệ bởi nền tảng tính toán tin cậy sử dụng bộ thư viện giao tiếp hoạt động trong không gian địa chỉ người dùng.

| | | |
|-------------------------------------|--|-----------------------|
| Ứng dụng | Gói giao tiếp nhân SCOMP (Thư viện) | Lớp 3 (không tin cậy) |
| Nền tảng tính toán tin cậy SCOMP | Gói giao tiếp nhân SCOMP (Các hàm tin cậy) | Lớp 2 (tin cậy) |
| | Hệ điều hành tin cậy SCOMP (Phần mềm tin cậy SCOMP) | Lớp 1 (tin cậy) |
| | Hệ điều hành tin cậy SCOMP (Nhân an toàn) | Lớp 0 (tin cậy) |
| Phần cứng SCOMP | | |

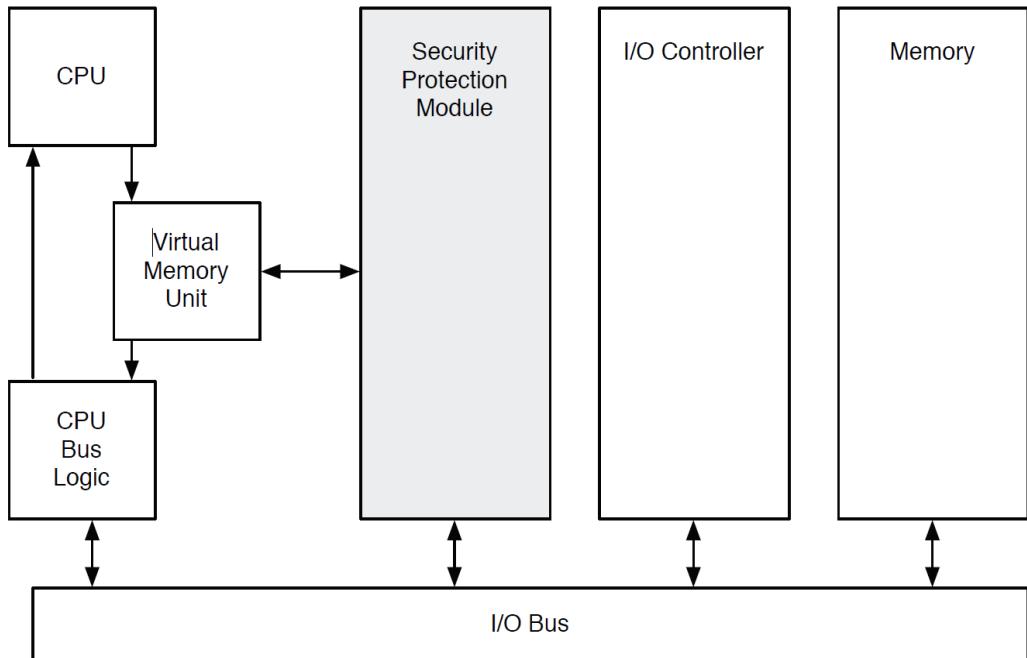
Hình 3-15. Kiến trúc SCOMP

Nhân an toàn của SCOMP ngăn chặn toàn bộ các truy nhập tới các tài nguyên sử dụng chính sách an toàn nhiều lớp. Khi ứng dụng cần sử dụng các tài nguyên được bảo vệ như bộ nhớ hay thiết bị vào/ra nó cần phải yêu cầu nhân an toàn cấp các thẻ mô tả phần cứng (*hardware descriptor*) phù hợp để sử dụng tài nguyên đó. Khi nhân an toàn chấp thuận yêu cầu của ứng dụng, nhân sẽ tạo ra thẻ mô tả phần cứng cho thao tác truy nhập này và lưu thẻ trong bộ nhớ. Tham chiếu tới thẻ mô tả phần cứng được trả lại cho ứng dụng cho các lần sử dụng sau. Thẻ mô tả phần cứng bao gồm tham chiếu tới đối tượng và các quyền truy nhập cho tiến trình.

Cách ly, chống xâm nhập được triển khai sử dụng cơ chế bảo vệ theo lớp. Cơ chế lớp truy nhập kiểm soát việc đoạn ở lớp này được chấp thuận để yêu cầu dịch vụ từ lớp khác. Các lớp và dịch chuyển lớp được thực thi nhờ phần cứng. Trong quá trình xây dựng hệ điều hành, việc kiểm chứng mô hình an toàn SCOMP và việc triển khai các chính sách an toàn được đặt ra hàng đầu cũng như các phần mềm tin cậy của hệ thống đều được thẩm tra. Dựa trên các tính năng thực thi phần cứng, phần mềm trong nền tảng tính toán tin cậy (chống xâm nhập) và thẩm tra chính xác nhân an toàn và các phần mềm tin cậy khác, SCOMP chứng tỏ việc xây dựng hệ thống an toàn thỏa mãn các tiêu chí của bộ giám sát tham chiếu. Quy trình này trở thành tiêu chuẩn cho việc đánh giá cấp độ an toàn cao nhất của Bộ quốc phòng Mỹ.

a. *Phần cứng*

SCOMP sử dụng cơ chế bảo vệ gồm 4 lớp. Nhân an toàn chạy ở lớp có đặc quyền cao nhất lớp 0 và ứng dụng người dùng ở lớp có đặc quyền ít nhất. Bên cạnh đó, SCOMP sử dụng cơ chế gọi hàm bằng cách sử dụng địa chỉ các tham số bên gọi cung cấp, việc này cho phép ngăn cản các đoạn mã truy nhập ra ngoài không gian nhớ này.



Hình 3-16. Phần cứng của SCOMP

Mô-đun bảo vệ an toàn SPM (*Secure protection module*) chịu trách nhiệm ngăn chặn toàn bộ các truy nhập bộ nhớ và vào/ra. Mỗi chương trình có địa chỉ cơ sở gốc trả về bộ nhớ và các thẻ vào/ra. Mỗi truy nhập bộ nhớ đều phải đi qua SPM và phải qua phép kiểm tra truy nhập. Do phần nhân cung cấp các thẻ và việc kiểm tra được thực hiện bằng phần cứng nên có thể cho phép chạy các câu lệnh vào/ra ở mức không cần đặc quyền.

Các kiến trúc phần cứng x86 áp dụng kiến trúc tương tự với phân lớp bảo vệ song không đảm bảo việc ngăn chặn các truy nhập tới thiết bị vào/ra qua SPM nên không thực sự an toàn.

b. Nền tảng tính toán an toàn

Nhân an toàn hoạt động ở lớp 0 cung cấp các chức năng cơ bản của hệ thống như quản lý bộ nhớ, điều độ tiến trình, quản lý ngắn, kiểm toán và giám sát tham chiếu. Các chức năng của nhân được giữ tối thiểu và chỉ chứa 10K dòng lệnh hầu hết được viết bằng ngôn ngữ Pascal. Các đối tượng trong nhân gồm có các tiến trình, các đoạn nhớ và các thiết bị được định danh bằng các thẻ 64bit. Nhân duy trì các thông tin kiểm soát truy nhập và trạng thái cho mỗi đối tượng. Mô hình kiểm soát truy nhập sử dụng mô hình Bell-La Padula mở rộng với phân phân loại, cách chính sách lớp bảo vệ và chính sách tùy chọn. Nhân an toàn định nghĩa 38 cổng để các tiến trình nằm ngoài nhân có thể gọi các dịch vụ nhân. Cổng tương tự như lời gọi hệ thống cung cấp chức năng tạo đối tượng, ánh xạ các đoạn nhớ và gắn bộ nhớ vật lý với bộ nhớ ảo.

Phần mềm tin cậy chạy các dịch vụ không cần mức 0 nhưng cung cấp các chức năng cần sự tin cậy để thực thi việc kiểm soát các ứng dụng người dùng một cách thích đáng. Có hai loại phần mềm tin cậy. Loại thứ nhất được tin cậy không vi phạm tính bí mật và

toàn vẹn của hệ thống như phần mềm nạp tiến trình. Loại thứ 2 được tin cậy để duy trì các chính sách an toàn một cách đúng đắn như các dịch vụ sửa đổi dữ liệu xác thực người dùng. Phần này có 23 tiến trình chứa 11k dòng lệnh viết bằng C.

Các phần mềm tin cậy được gọi thông qua các kênh thông tin tin cậy từ người dùng. Việc sử dụng các kênh này ngăn chặn các phần mềm xâm nhập giả mạo người dùng hợp lệ. Hơn thế, người dùng biết việc tương tác trực tiếp với phần mềm an toàn khi kênh an toàn được kích hoạt. Chỉ nhân có thể nhận được tín hiệu ngắn, như vậy người dùng cơ thể chắc chắn các đáp ứng bắt nguồn từ phần mềm tin cậy.

c. **Đánh giá**

SCOMP thực hiện việc ngăn chặn ở mức phần cứng như vậy đảm bảo bộ giám sát truy nhập kiểm soát được tất cả các yêu cầu của hệ thống tới các tài nguyên của hệ thống. Các tài nguyên của hệ thống như các đoạn bộ nhớ, bộ nhớ và vào/ra và tất cả các lệnh truy nhập vào các đoạn, phần ngăn chặn dựa trên phần cứng bẫy toàn bộ các truy nhập nhạy cảm.

Hệ thống file SCOMP ở lớp 2 kiểm soát việc truy nhập tới các file, mức cao, nơi các yêu cầu (chính sách truy nhập) được soạn thảo. Tuy nhiên, các truy nhập ban đầu tới file dữ liệu tùy thuộc vào truy nhập tới thiết bị vào/ra. Hệ thống file phải được tin cậy để ngăn chặn các truy nhập trái phép tới dữ liệu của người dùng bởi chương trình khác. Mặt khác, cơ chế ngăn chặn của SCOMP được kiểm tra và đánh giá dựa trên phần cứng.

SCOMP áp dụng lớp bảo vệ để bảo vệ nhân an ninh khỏi các sửa đổi trái phép. Nhân an ninh hoạt động ở lớp 0 và chỉ có 38 cổng cho phép truy nhập tới nhân từ các lớp bảo vệ khác. SCOMP sử dụng mô hình kiểm soát toàn vẹn phức tạp để biểu diễn truy nhập tới lớp 0 (lớp có nhiều đặc quyền nhất). Do nhân có thể cần phải cập nhật như hệ thống file, có một số các đối tượng (tài nguyên) và tiến trình có thể sửa đổi nhân bao gồm cả cơ chế bảo vệ và bộ giám sát tham chiếu.

Ngoài ra, SCOMP cũng sử dụng lớp bảo vệ và mô hình truy nhập (như đã trình bày trong cơ chế phần cứng) để bảo vệ tính toàn vẹn của các phần còn lại trong cơ sở tính toán tin cậy. Cơ sở tính toán tin cậy hoạt động ở lớp 0, 1, và 2. Các mã không tin cậy không được hoạt động ở những lớp này. Các tiến trình không tin cậy ở lớp 3 có thể gọi các đoạn mã trong phần tính toán tin cậy. Khi này các giao tiếp và các cổng được triển khai để bảo vệ cơ sở tính toán tin cậy không rõ.

Tuy vậy, tính đúng đắn của phần cơ sở tính toán tin cậy giữa phần thiết kế và triển khai được kiểm chứng bằng công cụ phân tích chính tắc. Thêm vào đó các mục tiêu an toàn được tuân thủ một cách bắt buộc sử dụng chính sách an ninh nhiều lớp kiểu bắt buộc. Các hệ thống phát triển kế tiếp được kiểm chứng về tính đúng đắn của thiết kế các chính sách hệ thống.

Như vậy, SCOMP đáp ứng một cách thuyết phục nhất cho câu hỏi về tính an toàn cũng như tính đúng đắn của các biện pháp đảm bảo an ninh và an toàn cho hệ thống. Mặc

dù vẫn còn một số nhỏ các điểm nguy hiểm như độ phức tạp trong giao tiếp với phần tính toán tin cậy cũng như tính không đầy đủ trong việc kiểm chứng hệ thống, SCOMP và các nhân an toàn khác tiềm gầm nhất đến hệ điều hành an toàn.

3.5 Kết luận

Kiến trúc máy tính x86 là kiến trúc được sử dụng phổ biến và rộng rãi cho các máy PC, máy trạm cũng như là máy chủ hiện thời. Các cơ chế an toàn với việc quản lý tiến trình cũng như là bộ nhớ đã được triển khai như cơ chế bảo vệ theo lớp đặc quyền, cơ chế quản lý bộ nhớ thông qua việc hạn chế truy nhập tới các thẻ quản lý thông tin cấp phát bộ nhớ hay thiết bị vào/ra. Intel và AMD là hai công ty lớn cung cấp bộ xử lý x86 đã triển khai các cơ chế bảo vệ cho việc quản lý các tiến trình cũng như bộ nhớ. Các chương trình không tin cậy và các chương trình hoạt động ở mức hệ thống được cách ly và việc truy nhập vào các thông tin bảo vệ được kiểm soát.

Mặt khác để bảo vệ tính toàn vẹn và bí mật của dữ liệu lưu trữ trên các thiết bị một cách tin cậy và bảo đảm, người ta sử dụng các kỹ thuật mã hóa và băm dữ liệu trên cơ sở các chức năng này được đóng gói trong các thiết bị phần cứng riêng biệt. Đây là động lực dẫn đến sự ra đời của các mô-đun tính toán tin cậy. Khi này dữ liệu không chỉ được bảo vệ chống lại việc mất cắp mà còn kiểm chứng được tính an toàn (tồn vẹn) của quá trình thực thi. Điều này đặc biệt quan trọng khi các chương trình tương tác với nhau qua môi trường mạng. Chương trình ở xa có thể chứng minh tính tin cậy thông qua việc cung cấp bằng chứng (chuỗi thực thi tin cậy) và bên tương tác có thể kiểm chứng được bằng chứng này.

Windows và Unix/Linux là hai nhóm hệ điều hành tiêu biểu sử dụng và triển khai các cơ chế bảo vệ được cung cấp từ kiến trúc x86. Việc đảm bảo an toàn được triển khai thông qua cơ chế xác thực người dùng và kiểm soát truy nhập tới các tài nguyên của hệ thống. Cơ chế kiểm soát đặc quyền là tiền đề cơ bản cho việc triển khai thành công và an toàn việc kiểm tra và xác thực người dùng. Với cách tiếp cận khác nhau, hai hệ điều hành này có cách cơ chế an toàn với mức độ phức tạp khác nhau cũng như là các điểm yếu có hổng. Hiện thời cả hai hệ điều hành đều hỗ trợ cơ chế bảo vệ các file dựa trên mô-đun tính toán tin cậy như khởi động có bảo vệ hay bảo mật dữ liệu.

Để cải thiện các cơ chế bảo vệ nhằm nâng cao độ tin cậy và khả năng chống xâm nhập, Linux cung cấp bộ khung các mô-đun an ninh cho phép kiểm soát việc truy nhập thông qua các chính sách bắt buộc. Yếu tố quan trọng trong việc sử dụng chính sách là khả năng kiểm chứng các yêu cầu an toàn và an ninh của hệ thống có bị vi phạm hay không. Tuy vậy, cách tiếp cận này vẫn không thể chắc chắn (chứng minh được) về tính tin cậy của hệ thống vì vẫn dựa trên các cơ chế bảo vệ truyền thống của x86 và cách chương trình nhân của Linux. Cách tiếp cận triệt để hơn là sử dụng phần cứng và phần mềm như trong việc xây dựng nhân SCOMP. Khi này tính an toàn và khả năng chống

xâm phạm sẽ được đảm bảo nhờ mô-đun kiểm soát bằng phần cứng và phần mềm được kiểm soát và đánh giá tính an toàn nhờ các công cụ phân tích mã nguồn.

3.6 Câu hỏi ôn tập

- 1) Cách thức triển khai lớp bảo vệ với câu lệnh cần đặc quyền trong kiến trúc x86?
- 2) Cách thức bảo vệ truy nhập không gian nhớ được bảo vệ trong kiến trúc x86?
- 3) Giải thích cơ chế cấm thực thi dữ liệu?
- 4) Quá trình khởi động được bảo vệ (measured boot)?
- 5) Giải thích khởi động được bảo vệ (measured boot) giúp người dùng chống lại mã độc như thế nào?
- 6) Giải thích cơ chế lưu trữ an toàn sử dụng TPM (Trusted Platform Module)?
- 7) Các đặc trưng của cơ chế an toàn trong hệ điều hành Windows và Unix/Linux?
- 8) Các đặc trưng an toàn của hệ thống SELinux?
- 9) Các đặc trưng an toàn của nhân an toàn SCOMP?

CHƯƠNG 4. CÁC MÔ HÌNH AN TOÀN

Một khái niệm quan trọng trong thiết kế và phân tích an toàn của hệ thống là mô hình an toàn do các mô hình này tích hợp chính sách an toàn hay các mục tiêu cần phải được thực thi và đảm bảo trong hệ thống. Nói cách khác, các yêu cầu an toàn đối với mô hình thể hiện một cách tường minh trong chính sách an toàn. Mô hình là biểu diễn dạng ký hiệu (*symbolic representation*) của các chính sách và ánh xạ mong muốn của người đề ra chính sách thành lập các luật mà phải được tuân thủ trong hệ thống máy tính. Chính sách là thuật ngữ trừu tượng mô tả mục tiêu và các kết quả mà hệ thống phải đáp ứng và hoàn thành theo cách an toàn và chấp nhận được.

Mô hình an toàn ánh xạ các mục tiêu khái quát và trừu tượng của chính sách vào các bộ phận của hệ thống máy tính bằng cách mô tả các cấu trúc dữ liệu và kỹ thuật cụ thể để thực thi chính sách an toàn. Thông thường, mô hình an toàn được biểu diễn bằng các ký hiệu toán học, các ý tưởng được phân tích, mà chúng được chuyển thành các đặc tả của hệ thống, và sau đây được phát triển thành các đoạn mã chương trình.

Một số mô hình an toàn thực thi các quy định và luật nhằm bảo vệ tính bí mật, một số khác hướng tới việc bảo vệ tính toàn vẹn. Các mô hình chính tắc (*formal model*) thường được dùng nhằm đảm bảo an ninh ở mức độ cao như Bell-LaPadula. Các mô hình phi chính tắc (*informal model*) như Clark-Wilson thường sử dụng như cấu trúc khung cho biết cách thức các chính sách an toàn được biểu diễn và thực thi.

4.1 Vai trò và đặc trưng của mô hình an toàn

Thành công trong việc đạt được mức độ an toàn cao trong hệ thống tùy thuộc vào mức độ cẩn thận trong quá trình thiết kế và triển khai các biện pháp kiểm soát an ninh. Mục tiêu của mô hình an ninh là biểu diễn các yêu cầu về an toàn của hệ thống một cách chính xác và kiểm chứng được.

4.1.1 Các đặc trưng

Mô hình an toàn có các đặc trưng cơ bản sau:

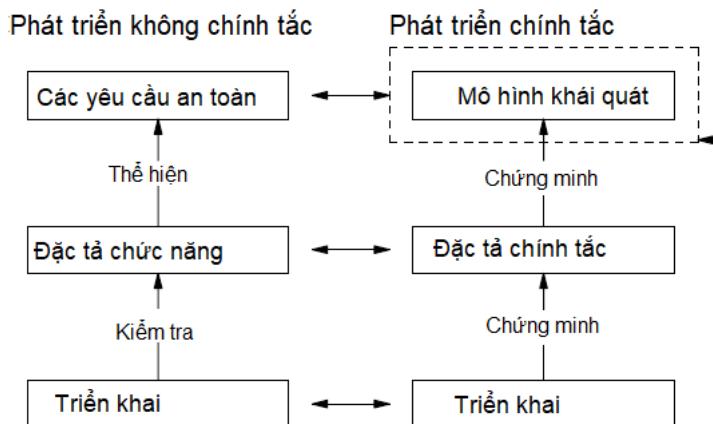
- Chính xác và không mơ hồ. Mô hình an toàn biểu diễn và thực thi chính sách an toàn của hệ thống chính vì vậy thuộc tính đầu tiên là tính chính xác và rõ ràng để mô tả một cách đầy đủ và trọn vẹn chính sách cần thực thi của hệ thống. Với các hệ thống an toàn cao, mô hình được diễn giải bằng các ký hiệu toán học. Tuy vậy, các khái niệm của việc lập mô hình hệ thống không nhất thiết cần các công cụ toán học đặc biệt là khi sử dụng lại mô hình sẵn có. Khi này, biểu diễn mô hình bằng ngôn ngữ thông thường hoàn toàn có thể đủ thỏa mãn thuộc tính này.

- Đơn giản, khái quát và do vậy dễ hiểu. Thuộc tính này giúp cho mô hình có thể được nắm bắt và triển khai một cách nhanh chóng và đầy đủ không chỉ với người thiết kế hay triển khai mà cả với người dùng cuối của hệ thống. Rõ ràng, nếu không ai có thể hiểu được yêu cầu an toàn thì không công cụ toán học nào có thể chứng minh sự phù hợp của mô hình an toàn.
- Căn bản: xử lý các thuộc tính an toàn và không hạn chế một cách quá đáng (không thích đáng) các chức năng khác hay việc triển khai của hệ thống
- Thể hiện rõ ràng chính sách an toàn. Mô hình an toàn cần chứa đựng đầy đủ và rõ ràng các mong muốn cũng như yêu cầu thiết yếu về việc vận hành và hoạt động hệ thống.

4.1.2 Vai trò

Một trong những vấn đề khiến cho hệ thống mất an toàn chính là người thiết kế không xác định được một cách chính xác và đúng đắn yêu cầu về an toàn. Vấn đề này một phần liên quan đến độ tin cậy của phần mềm và có thể khắc phục bằng kỹ thuật xây dựng phần mềm (*software engineering*) tốt kết hợp với các nguyên tắc và kỹ thuật riêng biệt cho vấn đề an toàn.

Vấn đề khác là xác định các chức năng hay hành vi của hệ thống xét về yếu tố an toàn. Việc này khá khó khăn khi xét yêu cầu an toàn vì các mô tả về chức năng hay hành vi này cần phải chính xác hơn rất nhiều. Khi này, mô hình an toàn khái quát (*abstract model*) đóng vai trò then chốt trong cách thức phát triển hệ thống một cách chính xác như trong Hình 4-1 dưới đây.



Hình 4-1. Tương quan giữa các bước phát triển mô hình an toàn

Mục tiêu phát triển hệ thống nhằm đảm bảo với mức độ chắc chắn nhất định rằng việc triển khai hệ thống phù hợp với mô hình an toàn lựa chọn. Mức độ khác biệt về chi tiết giữa mô hình và triển khai thường rất lớn nên cần thêm bước trung gian nhằm đảm bảo sự tương ứng giữa yêu cầu an toàn và triển khai thực tế.

Cách xây dựng không chính tắc giả định hệ thống không sử dụng công cụ chính tắc ngoại trừ việc định nghĩa mô hình an toàn. Các đặc tả trong cách phát triển này tập trung vào các chức năng của hệ thống và không chú trọng vào các yêu cầu về an ninh hay an toàn của hệ thống.

Với cách xây dựng chính tắc, người thiết kế cần tới các đặc tả và chứng minh chính tắc và cần có mô hình an toàn chính tắc. Về mức độ chi tiết các đặc tả chính tắc (*formal specification*) tương đương với các đặc tả chức năng song có độ chính xác và rõ ràng cao hơn nhiều. Tính chính tắc của các đặc tả cung cấp cơ sở toán học cho việc chứng minh toán học về sự tương ứng giữa đặc tả và mô hình an toàn đề ra.

Mô hình an toàn có thể dùng như các đặc tả về an ninh cho hệ thống. Việc này giúp hạn chế các lỗ hổng an toàn khi người thiết kế quá tập trung vào chức năng. Trên thực tế, việc lập mô hình an toàn chính tắc tiêu tốn nhiều công sức và nhân lực mà không phải ai cũng đủ để làm tất cả. Khi này, các mô hình an toàn đóng vai trò gợi ý cho người thiết kế để phát triển hệ thống an toàn một cách phù hợp.

4.2 Mô hình máy trạng thái

Mô hình máy trạng thái thường được sử dụng vì mô hình này biểu diễn hệ thống máy tính gần giống cách thức thực hiện của hệ điều hành. Một trạng thái là khái niệm trừu tượng của bit hay *byte* trong hệ thống thay đổi khi hệ thống hoạt động. Các ô nhớ trên bộ nhớ hay trên đĩa cứng hay thanh ghi của bộ xử lý đều là các trạng thái. Các hàm chuyển dịch trạng thái là cách khái quát của các lời gọi hàm hệ thống tới hệ điều hành và cho biết chính xác trạng thái có thể hay không thể thay đổi.

Mô hình an toàn không sử dụng tất cả các trạng thái và các chức năng của hệ thống mà người thiết kế mô hình lựa chọn các biến liên quan đến vấn đề an ninh để lập mô hình. Việc xây dựng mô hình an toàn máy trạng thái liên quan đến việc xác định các thành phần của mô hình bao gồm các biến, các chức năng, các quy định,... cùng với trạng thái an toàn khởi đầu. Khi đã xác định được tính an toàn của trạng thái khởi đầu và các chức năng khai là an toàn, việc suy diễn toán học cho phép xác định tình trạng an toàn của hệ thống bắt kể các chức năng được sử dụng như thế nào.

Các bước cụ thể để xây dựng mô hình máy trạng thái như sau:

1. Xác định các biến trạng thái liên quan

- Các biến mô tả các chủ thể và đối tượng bên trong hệ thống, các thuộc tính an toàn của chúng cũng như quyền truy nhập giữa chủ thể và đối tượng

2. Xác định trạng thái an toàn

- Mô tả một bất biến (*invariant*) biểu diễn quan hệ giữa các giá trị của biến mà luôn được đảm bảo trong khi thay đổi trạng thái

3. Xác định hàm chuyển dịch trạng thái

- Các hàm này mô tả các thay đổi tới các biến trạng thái còn gọi là các nguyên tắc hoạt động (*rule of operation*). Mục tiêu của các hàm là hạn chế các thay đổi mà hệ thống có thể thực hiện.
4. Chứng minh các hàm đảm bảo trạng thái an toàn
 - Để đảm bảo mô hình nhất quán với các mô tả về trạng thái an toàn, cần chứng minh với mỗi hàm hệ thống ở trạng thái an toàn trước và sau mỗi thao tác
 5. Xác định trạng thái khởi tạo. Lựa chọn các giá trị cho các biến trạng thái mà hệ thống bắt đầu ở trạng thái an toàn
 6. Chứng minh trạng thái khởi tạo an toàn theo các mô tả về trạng thái an toàn

Phần dưới đây sử dụng ví dụ minh họa cho các bước trình bày ở trên.

❖ Chính sách:

- Người dùng có thể đọc tài liệu khi và chỉ khi quyền (*clearance*) có được lớn hơn hoặc bằng phân loại (*classification*) của tài liệu

Đây là một dạng yêu cầu truy nhập tiêu biểu với cơ quan có áp dụng cơ chế đảm bảo an toàn thông tin trong cơ quan. Mục tiêu là xác định mô hình sử dụng cho hệ thống máy tính nhằm thực thi chính sách trên. Áp dụng cách thức mô tả về kiểm soát truy nhập ở phần trước, chính sách nêu trên được biểu diễn như mô tả chi tiết sau:

❖ Mô tả

- *Chủ thẻ có đọc đối tượng khi và chỉ khi lớp truy nhập của chủ thẻ lớn hơn hoặc bằng lớp truy nhập của đối tượng*
- *Chủ thẻ có ghi vào đối tượng khi và chỉ khi lớp truy nhập của chủ thẻ lớn hơn hoặc bằng lớp truy nhập của đối tượng*

❖ Mô tả các biến trạng thái

$S = \text{Tập các chủ thẻ}$

$O = \text{Tập các đối tượng}$

$sclass(s) = \text{lớp truy nhập của chủ thẻ } s$

$oclass(o) = \text{lớp truy nhập của đối tượng } o$

$A(s,o) = \text{Tập các chế độ truy nhập, nhận các giá trị sau}$

$\{r\}$ Nếu chủ thẻ s có thể đọc đối tượng o

$\{w\}$ Nếu chủ thẻ s có thể ghi lên đối tượng o

$\{r,w\}$ Nếu cả đọc và ghi

\emptyset Nếu không được đọc cũng như ghi

$contents(o) = \text{nội dung của đối tượng } o$

$subj = \text{chủ thẻ hoạt động}$

- ❖ Trạng thái hệ thống tại bất kỳ thời điểm nào được biểu diễn bằng tập giá trị của tất cả các biến trạng thái

$\{S, O, sclass, oclass, A, contents, subj\}$

- ❖ Trạng thái an toàn chính là biểu diễn toán học của các mô tả thể hiện chính sách truy nhập mong muốn. Trạng thái này thể hiện bất biến (*invariant*) của hệ thống.

Hệ thống an toàn khi và chỉ khi $\forall s \in S, o \in O,$

Nếu $r \in A(s, o)$, thì $sclass(s) \geq oclass(o),$

Nếu $w \in A(s, o)$, thì $oclass(o) \geq sclass(s).$

Mặc dù trông đơn giản song, không thể chứng minh được các định nghĩa và diễn giải nêu trên thể hiện chính xác chính sách truy nhập nguyên thủy ban đầu. Vậy nên, điều vô cùng quan trọng là các thuộc tính của mô hình cần đơn giản và rõ ràng để cho người dùng có thể thấy được sự tương ứng với chính sách thực tế.

- ❖ Các hàm chuyển đổi trạng thái có thể coi như các lời gọi hàm hay thủ tục tới các tiện ích của hệ thống mà các tiện ích này làm thay đổi các biến trạng thái. Sau đây là một số hàm tiêu biểu

1. **Create_object** (o, c) Tạo đối tượng o với lớp truy nhập $c.$
2. **Set_access** ($s, o, modes$) Thiết lập chế độ truy nhập cho chủ thể s tới đối tượng $o.$
3. **Create / Change_object** (o, c) Thiết lập lớp truy nhập của o tới c và tạo mới.
4. **Write_object** (o, d) Ghi dữ liệu d vào $contents(o).$
5. **Copy_object** ($from, to$) Sao chép $contents(from)$ tới $contents(to).$
6. **Append_data** (o, d) Thêm dữ liệu d tới $contents(o)$

Nội dung của hai hàm khởi tạo và thiết lập được mô tả như sau

Hàm 1: Create_object (o, c)

Nếu $o \notin O$

thì ' $O = O \cup \{o\}$ ' và

' $oclass(o) = c$ ' và

Với mọi $s \in S, A(s, o) = \emptyset.$

Hàm 2. Set_access ($s, o, modes$)

Nếu $s \in S$ và $o \in O$

và nếu $\{[r \in modes \text{ and } sclass(s) \geq oclass(o)] \text{ hoặc } r \notin modes\}$ và

$\{[w \in modes \text{ và } oclass(o) \geq sclass(s)] \text{ hoặc } w \notin modes\}$

thì ' $A(s, o) = modes.$ '

Với mỗi hàm cần chứng minh định lý sau:

bất biến (thuộc tính an toàn) \cap hàm chuyển dịch \Rightarrow bất biến

Điều này có nghĩa là khi áp dụng hàm chuyển dịch, bất biến vẫn đúng với trạng thái mới.

- ❖ Trạng thái ban đầu thể hiện giá trị khởi tạo của các biến trong hệ thống

$\{S_0, O_0, sclass_0, oclass_0, contents_0, subj_0\}$ hay có thể biểu diễn như sau:

$$\forall s \in S_0, o \in O_0$$

$$sclass_0(s) = c_0$$

$$oclass_0(o) = c_0$$

$$A_0(s, o) = \{\mathbf{r}, \mathbf{w}\}$$

4.3 Mô hình Harrison-Ruzzo-Ullman

4.3.1 Định nghĩa

Mô hình Harrison-Ruzzo-Ullman (HRU) hướng tới xử lý quyền truy nhập của các chủ thể và tính toàn vẹn của các quyền này. Mô hình này cho phép quyền truy nhập thay đổi và xác định chủ thể và đối tượng cần được tạo và xóa thẻ nào. Cuối cùng, mô hình này cho phép kiểm chứng các thay đổi về quyền có tác động như thế nào đến các yêu cầu về an toàn đặt ra. Nói cách khác, mô hình cho phép đánh giá tính an toàn của các thao tác thay đổi quyền này.

Mô hình HRU được định nghĩa như sau:

- Tập chủ thể S
- Tập đối tượng O
- Tập quyền truy nhập R
- Ma trận truy nhập $M \mid M = (M_{so})_{s \in S, o \in O} \mid M_{so} \in R$

- Tập các câu lệnh C với mỗi câu lệnh c dưới dạng

$c(x_1, \dots, x_k)$

if r_1 in $M_{s1,o1}$ and

if r_2 in $M_{s2,o2}$ and

...

if r_m in $M_{sm,om}$

then

op₁

op₂

....

op_n

end

Các thao tác op_i có thể là một trong các thao tác nguyên thủy như sau

1. ***enter r into (xs; xo)*** Thêm r vào $M[x_s, x_o]$
2. ***delete r from (xs; xo)*** Loại bỏ r khỏi $M[x_s, x_o]$
3. ***create subject x_s*** Tạo hàng mới và cột mới trong M
4. ***create object x_o*** Tao cột mới trong M
5. ***destroy subject xs*** Loại bỏ hàng và cột tương ứng với x_s
6. ***destroy object x_o*** Loại bỏ cột tương ứng với x_o

Ví dụ dưới đây minh họa cho việc biểu diễn các chức năng của hệ thống sử dụng cách mô tả của mô hình HRU.

- ❖ Chủ thẻ s tạo file f (có quyền sở hữu o file này) và có quyền đọc r , ghi w với file

command create_files(s,f)

create f

enter o into M_{s,f}

enter r into M_{s,f}

enter w into M_{s,f}

end

- ❖ Chủ sở hữu s của f cấp quyền truy nhập r cho chủ thẻ p

command grant_read(s,p,f)

if o in M_{s,f}

then enter r into M_{p,f}

end

Vấn đề đặt ra là các câu lệnh nêu trên có đảm bảo được thuộc tính an toàn của hệ thống hay không.

4.3.2 Các đặc trưng của mô hình HRU

Mô hình HRU có các đặc điểm như sau:

- ❖ Các câu lệnh làm thay đổi quyền truy nhập đối tượng được lưu lại thông qua sự thay đổi của ma trận truy nhập. Như vậy, ma trận truy nhập thể hiện trạng thái của hệ thống. Nói cách khác mô hình HRU sử dụng cách kiểm soát thông qua ma trận truy nhập.
- ❖ Mô hình HRU biểu diễn các chính sách an toàn thông qua việc điều chỉnh cấp quyền truy nhập. Để kiểm tra hệ thống tuân thủ chính sách an toàn, cần chứng minh không tồn tại cách cấp quyền truy nhập không mong muốn.
 - Ma trận M coi là rò rỉ quyền r nếu tồn tại thao tác c thêm quyền r vào một vị trí của M mà trước đó không chứa r .
 - Ma trận M là an toàn với quyền r nếu không có chuỗi lệnh c nào có thể chuyển M sang trạng thái rò rỉ r .
- ❖ Trạng thái an toàn của hệ thống được mô hình HRU diễn giải không chính xác như sau:
 - Truy nhập tài nguyên của hệ thống mà không có sự đồng ý của chủ sở hữu là không thể.
 - Người dùng cần có khả năng xác định liệu việc họ định làm có thể dẫn đến việc rò rỉ quyền tới các chủ thể không được phép.

4.3.3 Các kết quả của HRU

Mục tiêu mà mô hình HRU hướng tới là xây dựng mô hình đơn giản có thể áp dụng được cho nhiều hệ thống an toàn khác nhau. Mô hình này đã chứng tỏ:

- Với ma trận truy nhập M và quyền r , việc kiểm chứng tính an toàn của M với r là không xác định được. Bài toán an toàn không giải quyết được trong trường hợp tổng quát đầy đủ. Với mô hình hạn chế hơn thì có thể giải quyết được.
- Với hệ thống mà các lệnh chỉ chứa 1 thao tác (*toán tử*), với ma trận truy nhập M và quyền r , việc kiểm chứng tính an toàn của M là xác định được. Với hệ thống lệnh chứa 2 thao tác, việc kiểm chứng là không xác định được.
- Bài toán an toàn cho hệ thống xác thực bất kỳ là xác định được nếu số lượng các chủ thể là hữu hạn.

4.4 Các mô hình khác

Phần này trình bày các mô hình an toàn thông tin tiêu biểu khác bao gồm mô hình luồng thông tin khái quát và các mô hình hướng tới tính bí mật và toàn vẹn của hệ thống.

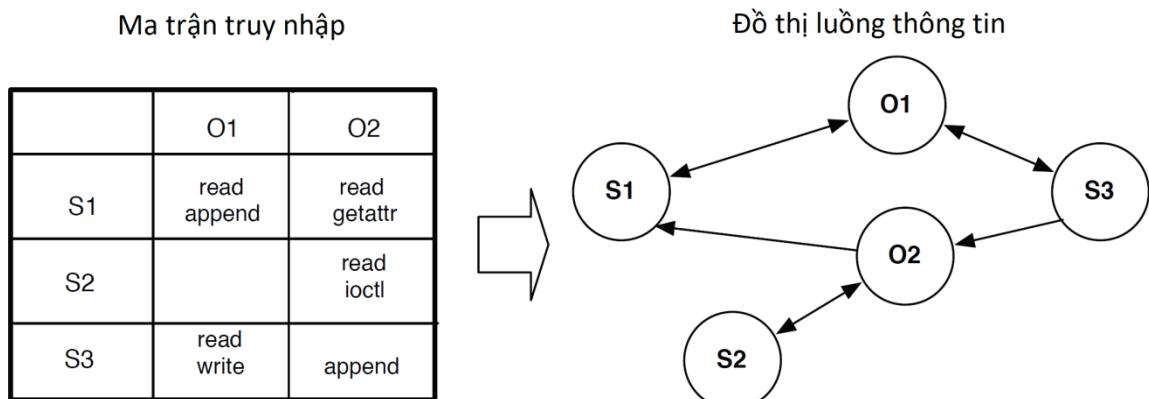
4.4.1 Mô hình luồng thông tin

Một trong những hạn chế của kỹ thuật dựa trên máy trạng thái là sự thiếu mô tả về luồng thông tin hơn là thiếu sót mô tả các ràng buộc hay bất biến của các thuộc tính an ninh của các đối tượng và chủ thể. Ở khía cạnh khác, vấn đề an toàn liên quan tới việc luân chuyển của dữ liệu thì cần được xử lý bằng các ràng buộc hay hạn chế lên luân chuyển này. Đó là lý do cần có mô hình luồng thông tin.

Mô hình luồng thông tin biểu diễn cách thức dữ liệu di chuyển giữa đối tượng và chủ thể trong hệ thống. Khi chủ thể (*chương trình*) đọc từ một đối tượng (*file*), dữ liệu từ đối tượng di chuyển vào bộ nhớ của chủ thể. Nếu có bí mật trong đối tượng thì bí mật này chuyển tới bộ nhớ của chủ thể khi đọc. Như vậy, bí mật có thể bị lộ khi chủ thể ghi bí mật này ra đối tượng.

Với mỗi thao tác trên một đối tượng thì thao tác này có thể là đọc luồng thông tin (tức là lấy dữ liệu ra khỏi chủ thể) hay là ghi luồng thông tin (tức là cập nhật đối tượng với dữ liệu mới) hay kết hợp cả hai.

Đồ thị luồng thông tin gồm các đỉnh là các chủ thể và đối tượng, các cung biểu diễn các thao tác giữa các chủ thể và đối tượng, chiều thể hiện hướng đi của dữ liệu tới đối tượng hay vào bộ nhớ của chủ thể. Hình dưới đây thể hiện cách thức xây dựng đồ thị luồng thông tin từ ma trận truy nhập của hệ thống.



Hình 4-2. Xây dựng luồng thông tin

Luồng thông tin được sử dụng để mô tả các mục tiêu an toàn của hệ thống (thuộc tính về bí mật và toàn vẹn) bằng cách phân tích các cung trong đồ thị luồng thông tin.

- ❖ **Tính bí mật.** Các cung trong đồ thị biểu diễn toàn bộ các đường dẫn mà dữ liệu có thể bị rò rỉ qua đó.

Chúng ta có thể dùng đồ thị để xác định liệu có một đối tượng bí mật o rò rỉ tới chủ thể không được phép s . Nếu tồn tại một đường dẫn từ o tới s thì tính bí mật của hệ thống bị xâm phạm

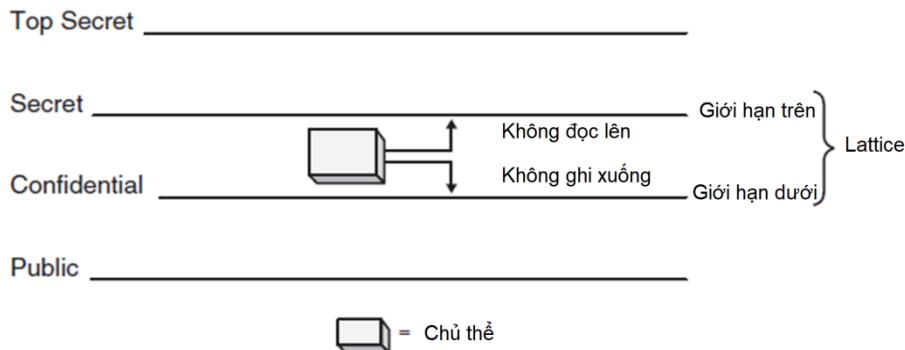
- **Tính toàn vẹn.** Không một chủ thể với mức độ toàn vẹn cao lê thuộc vào bất cứ chủ thể hay đối tượng nào có mức toàn vẹn thấp.

Chúng ta dùng đồ thị để xác định liệu chủ thẻ s_1 có nhận đầu vào từ chủ thẻ s_2 mà có mức độ toàn vẹn thấp hơn không. Nếu có tồn tại một đường dẫn từ s_2 tới s_1 thì không đảm bảo độ toàn vẹn

4.4.2 Mô hình đảm bảo tính bí mật - Bell-La Padula

Mô hình Bell-La Padula là mô hình luồng thông tin phổ biến nhất hướng tới việc bảo vệ tính bí mật. Để thực hiện việc kiểm soát truy nhập, mô hình này định nghĩa mức độ an toàn cho các đối tượng dữ liệu. Các đặc trưng của mô hình này như sau:

- Quyền truy nhập được định nghĩa thông qua ma trận truy nhập và thứ tự mức an toàn.
- Các chính sách an toàn ngăn chặn luồng thông tin đi xuống từ mức an toàn cao xuống mức thấp
- Mô hình này chỉ xem xét luồng thông tin xảy ra khi có sự thay đổi hay quan sát một đối tượng



Hình 4-3. Nguyên tắc an toàn trong Bell-La Padula

Như trong hình trên, các nhãn an toàn trong mô hình trên là “Top Secret”, “Secret”, “Confidential”, và “Public” với mức độ an toàn giảm dần. Nói cách khác, các dữ liệu với nhãn “*Top secret*” có mức bảo mật cao nhất và “*Public*” là thấp nhất. Các nhãn này cũng được gán cho các chủ thẻ thực hiện các thao tác lên các đối tượng. Nguyên tắc đảm bảo an toàn cho hệ thống được diễn giải đơn giản là không đọc lên và không ghi xuống. Nghĩa là, chủ thẻ chỉ được phép đọc dữ liệu mà nhãn an toàn của dữ liệu thấp hơn nhãn an toàn của chủ thẻ và chỉ được phép ghi khi nhãn an toàn của chủ thẻ không lớn hơn nhãn an toàn của dữ liệu.

Nguyên tắc an toàn của mô hình Bell-La Padula được biểu diễn như sau:

$SC(o)$ và $SC(s)$ là các nhãn an toàn của dữ liệu o và chủ thẻ s . Các nhãn này tuân thủ trật tự nhất định, khi này các thao tác đọc ghi được phép của chủ thẻ s lên đối tượng o khi và chỉ khi

- đọc: $SC(s) \geq SC(o)$
- ghi: $SC(s) \leq SC(o)$

Mô hình được phát triển tập trung cho việc đảm bảo tính bí mật của các đối tượng (dữ liệu). Mô hình không đề cập đến việc thay đổi quyền truy nhập của các người dùng (chủ thẻ) của hệ thống. Mô hình này chứa kênh ngầm (*covert channel*) thể hiện qua việc đối tượng mức thấp có thể phát hiện sự tồn tại của đối tượng mức cao khi bị từ chối truy nhập. Vấn đề này xâm phạm trực tiếp đến tính bí mật của đối tượng.

4.4.3 Mô hình đảm bảo tính toàn vẹn

a. Mô hình Biba

Mô hình này đảm bảo tính toàn vẹn theo cách thức tính toán vẹn của dữ liệu bị đe dọa khi chủ thẻ ở mức toàn vẹn thấp có khả năng ghi vào đối tượng (dữ liệu) có mức toàn vẹn cao hơn và khi chủ thẻ có thể đọc dữ liệu ở mức thấp. Mô hình Biba áp dụng hai quy tắc:

- *Không ghi lên*: Chủ thẻ có thể không thể ghi dữ liệu vào đối tượng có mức toàn vẹn cao hơn
- *Không đọc xuống*: Chủ thẻ không thể đọc dữ liệu từ mức toàn vẹn thấp hơn

Như vậy, tương tự như mô hình Bell-LaPadula các nhãn an ninh được sử dụng để biểu diễn mức độ an toàn mong muốn và kiểm soát các thao tác của chủ thẻ lên đối tượng. Tuy nhiên, luồng thông tin trong mô hình Biba được kiểm soát ngược với mô hình Bell-LaPadula. Các đối tượng có mức độ an ninh thấp có nghĩa là độ toàn vẹn thấp và chủ thẻ có mức độ an ninh cao không được sử dụng thông tin từ các đối tượng này. Nói cách khác chủ thẻ có yêu cầu an ninh cao không được sử dụng thông tin từ nguồn có độ tin cậy thấp. Tình huống này có thể thấy trong việc tiếp nhận dữ liệu đầu vào của các máy chủ Web, cơ sở dữ liệu hay dịch vụ hệ thống từ các nguồn không tin cậy.

b. Mô hình Clark-Winson

Mô hình cung cấp cách tiếp cận khác cho vấn đề toàn vẹn dữ liệu. Mô hình này tập trung vào việc ngăn chặn người dùng không hợp lệ sửa đổi trái phép dữ liệu. Trong mô hình này, người dùng không thao tác trực tiếp với các đối tượng mà thông qua một chương trình. Chương trình này hạn chế các thao tác người dùng được thực hiện lên đối tượng và như vậy bảo vệ tính toàn vẹn của đối tượng

Tính toàn vẹn được dựa trên nguyên tắc các công việc (thủ tục) được định nghĩa tường minh và việc tách biệt trách nhiệm (*separation of duty*). Nói cách khác, mô hình dựa trên cơ sở qui trình công việc được xây dựng một cách rõ ràng và nguyên tắc tách biệt trách nhiệm của người dùng tham gia vào qui trình xử lý công việc.

Mô hình Clark-Winson được mô tả như sau:

- ❖ Chủ thẻ và đối tượng được dán nhãn theo chương trình
- ❖ Chương trình đóng vai trò như lớp trung gian giữa chủ thẻ và đối tượng
- ❖ Việc kiểm soát truy nhập được thực hiện nhờ

- Định nghĩa các thao tác truy nhập có thể được thực hiện lên từng mục dữ liệu
- Định nghĩa các thao tác truy nhập có thể được thực hiện bởi chủ thể
- ❖ Các thuộc tính an toàn được mô tả qua các luật chứng thực và cần kiểm tra để đảm bảo các chính sách an ninh nhất quán với yêu cầu của chương trình

Các dữ liệu có mức độ toàn vẹn cao, gọi là các mục dữ liệu hạn chế CDI (*Constrained Data Items*), phải được kiểm chứng nhờ các chương trình đặc biệt có mức độ toàn vẹn tương đương. Các chương trình này gọi là các thủ tục kiểm chứng toàn vẹn IVP (*Integrity Verification Procedure*). Các dữ liệu này cũng chỉ được sửa đổi qua các thủ tục chuyển đổi TP (*Transformation Procedure*) có mức toàn vẹn tương đương. Các chương trình IVP đảm bảo các dữ liệu CDI thỏa mãn các yêu cầu nhất định để hệ thống đảm bảo được mức độ toàn vẹn mong muốn khi khởi động. Các thủ tục chuyển đổi có vai trò tương tự như các chủ thể trong mô hình Biba ở chỗ chúng chỉ có thể sửa đổi các dữ liệu có mức toàn vẹn cao.

Mô hình Clark-Winson xây dựng các yêu cầu chứng thực và các luật để đảm bảo tính toàn vẹn của hệ thống như sau:

Quy định chứng thực

- Thủ tục kiểm chứng toàn vẹn IVP phải đảm bảo các mục dữ liệu hạn chế CDI ở trạng thái hợp lệ khi IVP chạy
- Thủ tục chuyển đổi TP phải được chứng thực là hợp lệ tức là CDI bắt buộc phải chuyển đổi thành CDI hợp lệ
- Các luật truy nhập này phải thỏa mãn bất kỳ yêu cầu về việc tách biệt trách nhiệm
- Tất cả các thủ tục TP phải ghi vào log chỉ ghi thêm
- Bất kỳ TP có đầu vào dữ liệu không hạn chế UDI (*unconstrained data items*) thì phải chuyển đổi sang dạng CDI hoặc loại bỏ UDI đó và không thực hiện việc chuyển đổi nào

Quy định thực thi (Enforcement rules)

- Hệ thống phải duy trì và bảo vệ danh sách các mục {TP,CDI_i,CDI_j,...} cho phép TP được xác thực truy nhập tới các CDI
- Hệ thống phải duy trì và bảo vệ danh sách {UserID,TP_i;CDI_i,CDI_j,...} chỉ định các TP mà người dùng được chạy
- Hệ thống phải xác thực từng người dùng khi yêu cầu thực hiện TP
- Chỉ có chủ thể xác thực qui định truy nhập TP mới có thể sửa đổi mục tương ứng trong danh sách. Chủ thể này phải không có quyền thực thi trên TP đó.

Các qui định chứng thực và thực thi cho thấy mô hình Clark-Winson yêu cầu cả ba thành phần xác thực, kiểm toán, và quản trị. Vấn đề xác thực thể hiện rõ ràng trong qui

định thực thi. Vấn đề kiểm toán thể hiện ở việc các TP phải cung cấp đủ thông tin về việc thực hiện để có thể mô tả lại các thao tác sửa đổi dữ liệu hạn chế CDI. Yêu cầu về quản trị thực hiện qua việc hạn chế các người dùng được quyền chứng thực không được phép chạy các chương trình thay đổi dữ liệu. Hơn thế nữa, mô hình này cũng hạn chế người dùng được phép thực thi tất cả các thao tác của một công việc.

4.5 Kết luận

Chương này giới thiệu cách thức lập mô hình an toàn để đánh giá và kiểm tra khả năng đáp ứng các yêu cầu an toàn với hệ thống. Các mô hình này có gắng biểu diễn các yêu cầu an toàn của hệ thống thành dang có thể đóng đếm được. Thông qua phép chứng minh hay phân tích kiểm nghiệm xem liêu các yêu cầu an toàn này có bị phá vỡ hay không. Các thuộc tính an toàn của hệ thống có thể được biểu diễn thành các yêu cầu về tính toàn vẹn hay tính bí mật như trong mô hình Biba hay Bell-La Padula. Chương này chỉ hạn chế ở việc giới thiệu các kết quả chứng minh về mặt toán học của các mô hình mà không giới thiệu chi tiết về các chứng minh đó.

4.6 Câu hỏi ôn tập

- 1) Nêu các đặc trưng cơ bản của mô hình an toàn?
- 2) Giải thích vai trò của mô hình an toàn?
- 3) Trình bày các bước lập mô hình máy trạng thái?
- 4) Ví dụ máy trạng thái
- 5) Trình bày cách lập mô hình HRU? Ví dụ?
- 6) Các thuộc tính an toàn của mô hình HRU?
- 7) Cách lập mô hình luồng thông tin? Cách thức xác định các thuộc tính an toàn và toàn vẹn? Cho ví dụ?
- 8) Cách lập mô hình Bell-LaPadula? Đánh giá về thuộc tính an toàn của mô hình?
- 9) Cho ví dụ giải thích mô hình Biba?
- 10) Trình bày đặc trưng và cách xây dựng mô hình Clark-Winson?

CHƯƠNG 5. ĐÁNH GIÁ AN TOÀN

Vấn đề đánh giá an toàn có liên quan chặt chẽ đến việc đảm bảo chất lượng phần mềm theo nghĩa phần mềm được xây dựng xong phải thỏa mãn các đặc tả về an toàn của phần mềm. Như vậy, các thiếu sót trong quá trình phát triển và triển khai của phần mềm so với các yêu cầu sẽ dẫn đến các điểm yếu hay lỗ hổng mà có thể bị khai thác một cách vô tình hay có chủ đích. Bản thân hệ điều hành là tập hợp các phần mềm mà mỗi phần mềm cung cấp một số các chức năng của hệ điều hành. Như vậy an toàn của hệ điều hành lệ thuộc vào mức độ an toàn của các phần mềm chức năng.

Phần mở đầu chương giới thiệu tóm tắt các đặc trưng của đặc tả an toàn trong việc xây dựng các phần mềm hay hệ thống nói chung. Phần tiếp theo trình bày về các kỹ thuật giúp kiểm chứng hệ thống có đáp ứng các yêu cầu về an toàn hay không. Nhờ đó có thể phát hiện ra và đánh giá các vấn đề an toàn trong quá trình xây dựng các đặc tả an toàn cũng như trong quá trình phát triển hệ thống. Một vấn đề khác với người dùng cũng như phát triển hệ thống đó là sản phẩm phần mềm hay đơn giản các đoạn mã có thỏa mãn các yêu cầu về an toàn hay không. Các kỹ thuật kiểm chứng mã chương trình cho phép xác định các lỗi về an ninh trong việc xây dựng các đoạn mã, như vậy góp phần đánh giá mức độ an của phần mềm và hệ thống.

5.1 Các đặc trưng của đặc tả an toàn

Giữa đặc tả an toàn và mô hình an toàn có sự khác biệt. Các đặc tả an toàn chỉ hữu ích cho hệ thống cần phải đảm bảo mức độ an ninh cao nhất còn mô hình an toàn có khả năng ứng dụng rộng rãi hơn. Sử dụng mô hình an toàn giúp cho việc xây dựng các đặc tả an toàn được thuận tiện và dễ dàng hơn, tuy nhiên điều ngược lại không chính xác.

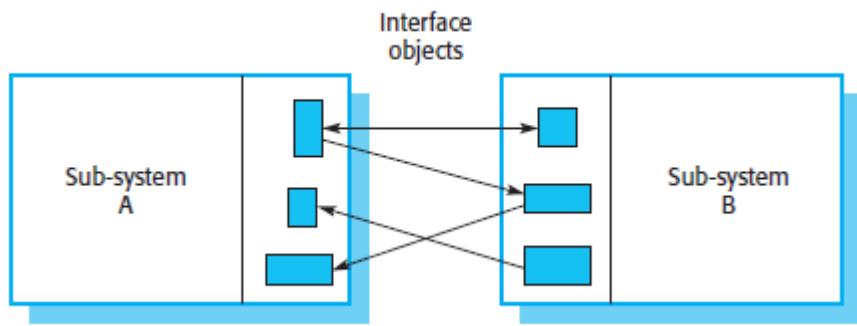
Mục đích của đặc tả an toàn là diễn tả các hành vi chức năng của hệ thống theo cách thức chính xác, không mơ hồ và phù hợp với việc xử lý của máy tính. Yêu cầu phù hợp với xử lý máy tính là để giảm thiểu lỗi do con người gây ra và giúp cho việc phân tích hệ thống được xây dựng có thỏa mãn các đặc tả hay không.

Các đặc tả chính xác có thể dùng để chứng minh các thuộc tính về thiết kế của hệ thống, đặc biệt là việc hệ thống xây dựng phù hợp với các đặc tả của mô hình an toàn. Công việc kiểm chứng chứng minh việc triển khai tuân thủ hay phù hợp với các đặc tả chính xác. Việc chứng minh một cách chính xác và đầy đủ cho hệ thống lớn thực sự là thách thức cho dù về mặt lý thuyết chứng minh đã được nghiên cứu rõ ràng. Dù vậy, việc thực hiện kiểm chứng chính xác một phần giúp người dùng đánh giá mức độ tương ứng giữa đặc tả và triển khai.

Các đặc tả chính xác trông giống như chương trình máy tính thông thường với biểu thức lô-gic và tính toán. Tuy nhiên, các ký hiệu có ngữ nghĩa phong phú hơn ngôn ngữ

máy tính cho phép biểu diễn các phép toán lô-gíc và quan hệ. Các đặc tả chính tắc bao gồm đặc tả giao tiếp và hành vi:

- *Đặc tả giao tiếp*: giúp cho việc phân rã các hệ thống lớn thành các hệ thống con. Các giao tiếp thường được mô tả bằng tập các đối tượng hay thành phần cho biết dữ liệu và các thao tác được truy nhập thông qua giao tiếp
- *Đặc tả hành vi*: mô tả các trạng thái có thể của hệ thống và các thao tác làm thay đổi trạng thái. Nói cách khác các hành vi của hệ thống có thể được bằng cách xây dựng cách thức các hành vi này làm thay đổi trạng thái của hệ thống như thế nào.



Hình 5-1. Giao tiếp giữa hai hệ thống

Hình vẽ dưới đây thể hiện các nguyên tắc an toàn của mô hình Bell-La Padula và các biểu diễn các nguyên tắc này dưới dạng đặc tả chính tắc.

MODEL

Invariant: The system is secure if and only if, for all $s \in S, o \in O$,
if $r \in A(s, o)$ then $sclass(s) \geq oclass(o)$,
if $w \in A(s, o)$ then $oclass(o) \geq sclass(s)$.

Constraint 1: For all $o \in O$,
 $'oclass(o) > oclass(o)$.

Constraint 2: For all $o \in O$,
if $r \notin A(subj, o)$
then for all $a \in S$, $'A(s, o) = A(s, o)$.

SPECIFICATION

```
INVARIANT
  FOR_ALL (p:process,f:file) SUCH_THAT (file_exists(f) AND
proc_exists(p))
    (IF "r" IN access (p,f)
     THEN proc_class (p) >= file-class (f))
    & (IF "w" IN access (p,f)
     THEN file_class (f) >= proc_class (p))

CONSTRAINTS
  FOR_ALL f:file SUCH_THAT file_exists (f)
    'file_class (f) >= file_class (f)
  FOR_ALL f:file SUCH_THAT file_exists (f)
    IF NOT ("r" IN access (cur_proc, f))
    THEN FOR_ALL p:process SUCH_THAT proc_exists (p)
      'access (p, f) = access (p, f)
```

Hình 5-2. Tương quan giữa mô hình và đặc tả an toàn

5.2 Các kỹ thuật kiểm chứng đặc tả an toàn

5.2.1 Kiểm chứng đặc tả

Chứng minh các đặc tả rất phức tạp và việc chứng minh thủ công có thể có lỗi đến mức không ai có thể tin tưởng do vậy cần có các công cụ tự động. Có hai cách tiếp cận là sử dụng công cụ chứng minh định lý (*theorem prover*) và kiểm chứng mô hình (*model checker*).

- ❖ *Chứng minh định lý.* Các công cụ này có thể có các mức độ tinh vi và phức tạp khác nhau từ việc kiểm tra các chứng minh các bước thủ công cho đến sử dụng các kỹ thuật của trí tuệ nhân tạo. Các hệ thống chứng minh và tích hợp đặc tả cho phép tạo ra một cách tự động các định lý dựa trên các tiên đề, hàm, bất biến, các hạn chế và các thành phần khác của đặc tả.

Các công cụ hiện thời cho phép chứng minh các bất biến và các ràng buộc của các đặc tả chứa hàng nghìn dòng với mức độ tin cậy thỏa đáng. Thông thường, các công cụ này cần có sự trợ giúp từ phía người dùng trong việc sinh ra các tiên đề, ràng buộc hay bất biến.

- ❖ *Kiểm chứng dựa trên mô hình.* Kỹ thuật này dựa trên việc mô tả các hành vi hệ thống có thể theo cách thức chính xác và rõ ràng về mặt toán học. Tiếp theo đó,

các mô hình hệ thống được kiểm nghiệm tất cả các trạng thái có thể mà thỏa mãn các mô tả ở trên bằng thuật toán. Việc này cho phép phát hiện sớm các lỗi như thiếu đầy đủ, mơ hồ, không nhất quán trong giai đoạn phân tích thiết kế.

Kỹ thuật này là cơ sở từ kiểm nghiệm toàn bộ (*kiểm chứng mô hình – model checking*) hay kiểm nghiệm các tình huống giới hạn (mô phỏng) hay kiểm nghiệm thực tế (*test*). Đáng chú ý nhất là kiểm chứng mô hình. Đây là kỹ thuật xem xét tất cả các trạng thái hệ thống có thể theo kiểu vết can. Nhờ vậy có thể chứng minh được mô hình hệ thống cho trước thực sự thỏa mãn một thuộc tính nhất định được mô tả trong phần đặc tả.

❖ *Các công cụ.* Phần dưới đây giới thiệu sơ bộ các công cụ hỗ trợ cho việc xây dựng và kiểm chứng hệ thống nói chung và ứng dụng cho phần mềm nói riêng.

- *Spin* : được dùng để lập mô hình phần mềm song song hay tiến trình dị bộ. Được phát triển bởi Bell Labs, Spin chủ yếu nhắm đến kiểm chứng chính xác các thuật toán máy tính.
- *Uppaal* : được dùng để lập mô hình hệ thống thời gian thực. Các chức năng cũng tương tự như Spin. Uppaal sử dụng ngôn ngữ riêng cho việc mô tả các mô hình cũng như các thuộc tính.
- *SMV, NuSMV* : được dùng để lập mô hình phản ứng (lô-gíc số) tuy nhiên cũng có thể dùng được cho lĩnh vực khác.
- *FDR* : được dùng để lập mô hình hệ thống dị bộ được phát triển bởi Trường Oxford. FDR sử dụng ngôn ngữ mô tả dành cho các tiến trình song song. Các hệ thống được lập mô hình dựa trên các sự kiện đồng bộ. FDR có nhiều ảnh hưởng lên các công cụ khác như SPIN.
- *Alloy* : được dùng để phân tích tính nhất quán của các cấu trúc dữ liệu dựa trên lý thuyết tập hợp do Trường MIT phát triển. Alloy sử dụng lô-gíc bậc nhất để chuyển các đặc tả thành các biểu thức Boolean và phân tích dựa trên bộ phân tích SAT.
- *Simulink Design Verifier* : được dùng để kiểm chứng mô hình được sinh ra từ Simulink, một công cụ mô phỏng dựa trên luồng dữ liệu và máy trạng thái. Công cụ này được các kỹ sư sử dụng rộng rãi. Điểm khác biệt là Simulink cho phép sinh ra mã C từ các mô hình. Vì vậy, công cụ này cũng có thể coi là công cụ triển khai.

5.2.2 Kiểm chứng bằng Alloy

Phần dưới đây giới thiệu chi tiết cách thức sử dụng Alloy cho việc mô tả và kiểm chứng các yêu cầu của hệ thống máy tính và ứng dụng cho việc mô tả các yêu cầu của hệ thống file trong máy tính do Trường MIT cung cấp.

Trước hết, Alloy là ngôn ngữ rút gọn dùng để mô tả các thuộc tính có cấu trúc. Alloy hỗ trợ việc mô tả các cấu trúc cơ sở (dạng đồ họa hay các khai báo dạng văn bản), cũng như các ràng buộc phức tạp và các thao tác diễn tả sự thay đổi cấu trúc động (cả hai được

biểu diễn bằng các công thức lô-gíc). Như vậy, Alloy không chỉ tích hợp mô hình đối tượng mà cả mô hình hoạt động theo phương pháp Fusion. Ngôn ngữ này có thể so sánh được với ngôn ngữ ràng buộc đối tượng OCL (*Object Constraint Language*) của ngôn ngữ UML.

Alloy có khả năng phân tích ngữ nghĩa hoàn toàn tự động mà có thể cho phép kiểm tra kết quả và tính nhất quán và việc thực hiện mô phỏng. Để có được khả năng thực thi, Alloy hy sinh việc trừu tượng hóa, mà có thể tạo ra các chuyển dịch mẫu của một thao tác được mô tả ngầm sử dụng phép phủ định và phép giao.

Ngôn ngữ này đã được sử dụng để lập mô hình và phân tích nhiều vấn đề như giao thức Internet di động, mô hình an toàn máy tính và mạng...

a. *Lập mô hình*

Mục tiêu của việc viết một mô hình là để mô tả một vài khía cạnh của hệ thống, hạn chế nó để loại trừ các trường hợp không đúng, và kiểm tra các thuộc tính về hệ thống. Ví dụ, có thể mô tả các thủ tục của công ty sử dụng cho việc chuyển hướng thư nội bộ, bổ sung một số ràng buộc về việc người chuyển thư cần phải xử lý như thế nào, và sau đó kiểm tra liệu các phần nào của thư hoặc tới đúng nơi nhận hay trả lại người gửi. Công cụ lập mô hình có thể trả lời “yêu cầu này luôn đúng với bài toán với kích cỡ là X” hoặc “yêu cầu này không đáp ứng được và đây là phản ví dụ”. Mô hình có thể coi là cách thức diễn giải các đặc tả của hệ thống mong muốn và chính là cách thức hệ thống được triển khai.

Có hai dạng vấn đề có thể nảy sinh mô hình được xây dựng:

- Thiếu sót trong bản thân mô hình. Điều này có thể xuất phát từ việc xây dựng thừa và thiếu các ràng buộc của mô hình.
- Lỗi trong đối tượng được lập mô hình. Cần phải kiểm tra các khẳng định (*assertion*) của hệ thống bằng cách lần theo vết khẳng định bị lỗi.

Alloy giống với các kỹ thuật lập mô hình và các ngôn ngữ khác nhưng có một số khác biệt quan trọng:

- Kiểm tra hữu hạn. Khi phân tích mô hình, cần xây dựng phạm vi hữu hạn của mô hình. Việc phân tích đảm bảo điều kiện cần nhưng không đủ. Dù vậy, nó đủ với phạm vi hữu hạn của mô hình và không bao giờ thiếu phần ví dụ.
- Mô hình vô hạn. Mô hình viết bằng Alloy không phản ánh thực tế là việc phân tích là hữu hạn. Tức là, người dùng mô tả các bộ phận của hệ thống và các tương tác của chúng nhưng không chỉ rõ có bao nhiêu bộ phận có thể có.
- Mô tả. Người lập mô hình mô tả trả lời câu hỏi “hệ thống nhận biết được X xảy ra như thế nào” ngược với người lập mô hình thực thi yêu cầu “Làm sao để hoàn thành X”.

- Phân tích tự động. Khác với ngôn ngữ mô tả đặc tả khác như Z hay UML, Alloy có thể tự động phân tích. Cụ thể, người dùng có thể tự sinh ra các ví dụ hay phản ví dụ về các yêu cầu của hệ thống được lập mô hình.
- Dữ liệu có cấu trúc. Alloy hỗ trợ các cấu trúc dữ liệu phức tạp như cây, như vây mang lại cách thức biểu diễn trạng thái phong phú.

b. Mô hình hệ thống file

Phần này giới thiệu sử dụng Alloy để lập mô hình hệ thống file và các ràng buộc đơn giản. Các đối tượng của hệ thống file có thể là file hay thư mục. Mỗi đối tượng file này có một gốc (thư mục chứa nó), các thư mục chứa nội dung (file hay thư mục mục con). Ngoài ra có thư mục gốc là đối tượng nằm trên đỉnh của hệ thống file.

Hệ thống file có thể có các ràng buộc như mỗi đối tượng của hệ thống file hoặc là file hoặc là thư mục, hệ thống file được kết nối, thư mục gốc không có cấp cao hơn,... Để khảo sát đặc tính động của hệ thống file cần bổ sung thêm thao tác di chuyển. Thao tác cho phép thay đổi cấu trúc của hệ thống file.

Định nghĩa các thành phần cơ bản

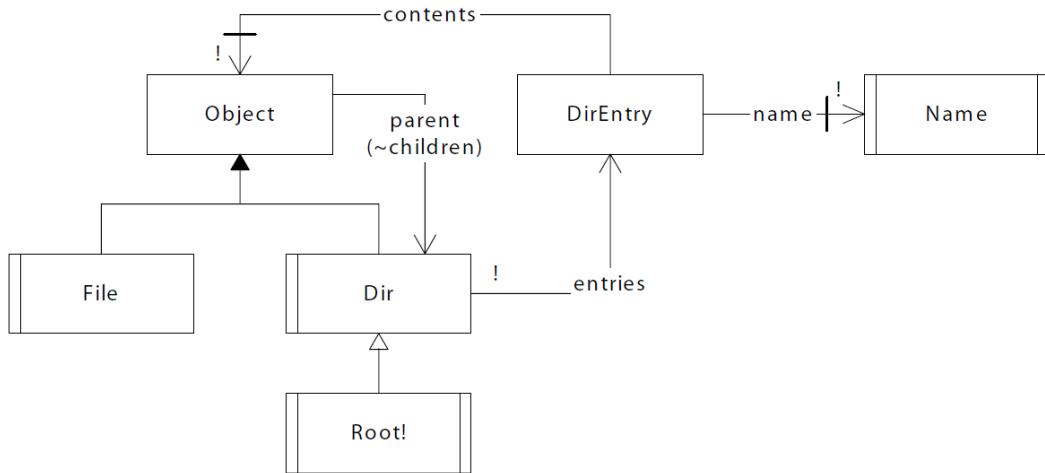
Tập các đối tượng của hệ thống file *FSObject* được định nghĩa giống như một lớp trong ngôn ngữ lập trình hướng đối tượng.

```
sig FSObject {
    parent: lone Dir
}
```

Các quan hệ của *FSObject* được mô tả giống như một trường của đối tượng. Ở đây *FSObject* có quan hệ với thư mục khác chứa bản thân nó gọi là *parent*. Từ khóa *lone* cho thấy quan hệ này là quan hệ 0 hoặc 1. Nghĩa là *FSObject* có thể nằm trong thư mục khác hoặc không. Tương tự như vậy, có thể định nghĩa hai đối tượng khác của hệ thống file là thư mục *Dir* và file *File*.

```
// Thư mục
sig Dir extends FSObject {
    contents: set FSObject
}
// file
sig File extends FSObject {}
```

Từ khóa *extends* cho thấy các file *File* và thư mục *Dir* là tập con của *FSObject* và hai tập *File* và *Dir* không có thành phần chung. Tất cả các thuộc tính của *FSObject* đều được *File* và *Dir* thừa kế.



Hình 5-3. Quan hệ giữa các đối tượng trong hệ thống file

Hệ thống file có một số ràng buộc cơ bản để chắc chắn các đối tượng hoạt động theo cách người dùng kỳ vọng. Ví dụ như quan hệ cấp trên (*parent*) và nội dung (*content*) cần không mang bất cứ quan hệ gì với nhau. Ngoài ra, có quan hệ ràng buộc tường minh là một thư mục là cấp trên với các thư bên chứa trong nó.

```
// Thư mục là cấp trên của các đối tượng chứa bên trong nó
fact {
    all d: Dir, o: d.contents | o.parent = d
}
```

Biểu diễn của ràng buộc trên như sau với $\forall d \in Dir, o \in d.contents \mid o.parent = d$. Hay với bất kỳ thư mục d nào và bất kỳ o thuộc về nội dung *contents* của d thì cấp trên của o chính là d . Nếu không có ràng buộc này thì sẽ có tình huống, *File* có cấp trên là thư mục gốc *Root* hay một thư mục có cấp trên chính là nó.

Phát biểu *fact* thể hiện một ràng buộc tường minh của mô hình. Khi Alloy tìm kiếm các ví dụ (trạng thái hệ thống thỏa mãn ràng buộc) thì các ví dụ vi phạm ràng buộc bị loại bỏ. Nếu ràng buộc bị lỗi thì sẽ không thể tìm thấy ví dụ nào cả.

Khi khai báo *Dir* và *File* thừa kế *FSObject* nghĩa là không có đối tượng *FSObject* có thể vừa là *File* vừa là thư mục *Dir*. Nhưng cần có ràng buộc không có *FSObject* nào không thuộc về một trong hai loại đối tượng trên.

```
// Tất cả các đối tượng hệ thống file phải hoặc là File hoặc là Dir
fact {
    File + Dir = FSObject
}
```

Hệ thống chỉ có duy nhất một thư mục gốc *Root* và được định nghĩa như dưới đây

```
// Tồn tại một thư mục gốc Root
one sig Root extends Dir { } { no parent }
```

Để chắc chắn hệ thống kết nối cần định nghĩa ràng buộc

```
// Hệ thống file kết nối
fact {
    FSObject in Root.*contents
}
```

Toán tử “*in*” được xem là ký hiệu “tập con của” còn toán tử * biểu diễn tập đóng phản xạ và bắc cầu. Ràng buộc này cho biết tập các đối tượng của hệ thống file là tập con của bát cứ đối tượng nào truy nhập được từ thư mục gốc *Root* bằng cách duyệt theo quan hệ *contents*. Bản thân *Root* không chứa chính nó trực tiếp hay gián tiếp.

Để kiểm tra các thuộc tính của hệ thống cần xây dựng các khai báo *assert* và dùng lệnh *check* để yêu cầu Alloy kiểm tra các phản ví dụ cho các mệnh đề khẳng định.

Khai báo *fact* bắt buộc mệnh đề phải đúng còn *assert* cho rằng một số thứ phải đúng do hành vi của mô hình. Sẽ không có thư mục nào chứa chính nó nghĩa là quan hệ giữa các thư mục không quay vòng. Như vậy quan hệ này được thể hiện như là tập đóng bắc cầu và khai báo thông qua toán tử ^

```
// Đường dẫn trong hệ thống file là không tuần hoàn
assert acyclic {
    no d: Dir | d in d.^contents
}
```

Để kiểm tra mô hình với tối đa 5 đối tượng *FSObject* sử dụng câu lệnh

check acyclic for 5

Khi kiểm tra, Alloy sẽ xem xét các trường hợp mà mức cao nhất của các đối tượng có tối đa là 5. Có thể có hai kết quả:

- “*no solution found*” nghĩa là không có phản ví dụ đối với câu khẳng định của mô hình với phạm vi kiểm tra hiện thời.
- “*solution found*” nghĩa là có phản ví dụ. Ví dụ này có thể được biểu diễn dưới dạng đồ họa.

Người dùng có thể kiểm tra có 1 thư mục gốc và với mỗi đối tượng file có một chỗ trong hệ thống file.

```
// hệ thống file có 1 thư mục gốc
assert oneRoot {
    one d: Dir | no d.parent
}

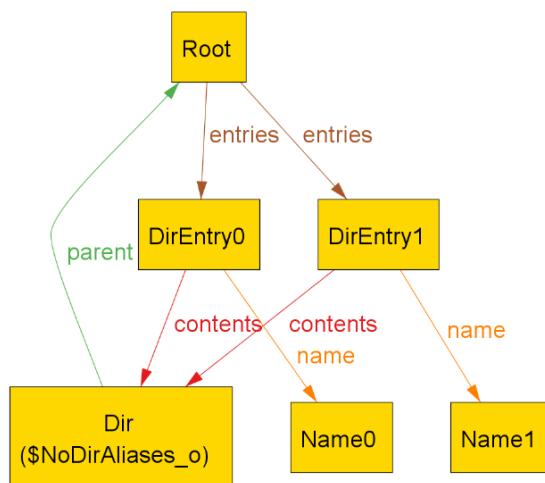
// Mỗi đối tượng có 1 chỗ
assert oneLocation {
    all o: FSObject | lone d: Dir | o in d.contents
}
```

Tùy khóa *one* và *lone* giúp định lượng các phần tử trong đó *one* có nghĩa là chỉ một còn *lone* có nghĩa là ít hơn hoặc bằng 1.

Giống như trong ngôn ngữ mô tả hành động, Alloy cho phép mô tả một thao tác/hành vi của hệ thống. Thao tác *move* di chuyển một đối tượng file hay thư mục từ chỗ này sang chỗ khác và cập nhật lại cấu trúc của hệ thống file. Alloy sử dụng tất cả các tham số này như làm tham số của hàm kề cả đầu ra. Kết quả của hàm là đúng nếu đầu ra của hàm là hợp lệ và sai khi ngược lại.

```
pred move [fs, fs': FileSystem, x: FSObject, d: Dir] {
    (x + d) in fs.objects
    fs'.parent = fs.parent - x->(x.(fs.parent)) + x->d
}
```

Mệnh đề *move* đúng nếu hệ thống *file fs'* là kết quả của việc chuyển đổi đối tượng *x* tới thư mục *d* trong hệ thống file *fs*. Cũng có thể *coi fs* là trạng thái ban đầu còn *fs'* là trạng thái sau thực hiện thao tác.



Hình 5-4. Phản chứng cho thấy lỗi với thư mục tuần hoàn

Ví dụ trên đây trình bày cách thức lập mô hình và kỹ thuật xây dựng các yêu cầu để đảm bảo hệ thống file hoạt động đúng như mong muốn của người thiết kế (sử dụng Alloy 4.2). Ở mức cao, cách thức xây dựng mô hình rất gần với cách thức phân tích thiết kế hướng đối tượng được sử dụng phổ biến trong công nghệ phần mềm hiện nay. Phần mô tả các ràng buộc hay các thuộc tính mà mô hình cần phải thỏa mãn hay đảm bảo rất gần với ngôn ngữ lô-gíc bậc nhất. Tuy nhiên, cốt lõi của biểu diễn là các phép biểu diễn tập hợp. Sau khi xây dựng xong, Alloy cho phép người thiết kế kiểm tra lại mô hình có đảm bảo được các thuộc tính (yêu cầu) đề ra hay không.

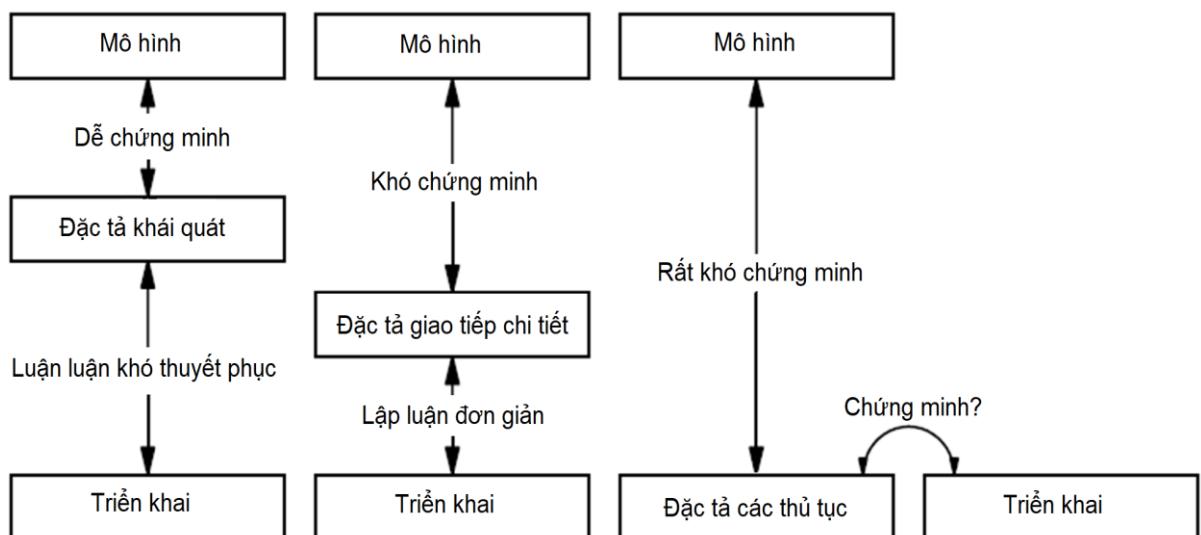
5.3 Các phương pháp phân rã dữ liệu và chương trình

Một mặt chúng ta mong muốn các đặc tả rất khái quát và gần với mô hình an toàn lựa chọn. Như vậy, chúng ta phải đổi mặt với khó khăn là diễn giải và chứng minh chi tiết một cách thuyết phục các đoạn mã sinh ra phù hợp với các đặc tả. Ở mức độ chi tiết hơn, các đặc tả gần với các thao tác thấy được ở giao tiếp của hệ thống. Đặc tả như vậy sẽ rất phức tạp và khó hiểu và việc chứng minh tương ứng giữa mô hình và triển khai sẽ không

thực tế. Mặt khác, các đặc tả biểu diễn các thủ tục/hàm bên trong của hệ thống hơn là các giao tiếp có thể thấy được. Khi này việc chứng minh mô hình có thể còn khó khăn hơn.

Hình 5-5 cho thấy các cách tiếp cận khác nhau với mức độ chi tiết của các đặc tả. Như vậy, công việc đặc tả cung cấp thông tin ở hai mức:

- Mức trừu tượng gần giống với việc lập mô hình
- Mức chi tiết mô tả các thao tác hay hoạt động của hệ thống như mô tả các giao tiếp



Hình 5-5. Các mức độ chi tiết của việc đặc tả giữa mô hình và việc triển khai

Chứng minh việc triển khai phù hợp mô hình đề ra có thể vô cùng khó khăn tuy nhiên việc chứng minh mã chương trình phù hợp với đặc tả đủ sát thì có thể được chấp nhận như việc chứng minh một phần.

Phản tiếp theo giới thiệu hai kỹ thuật cơ bản sử dụng cho việc xây dựng các đặc tả chính xác bao gồm phân rã cấu trúc dữ liệu và thuật toán.

5.3.1 Phân rã cấu trúc dữ liệu

Kỹ thuật phân rã dữ liệu (*data structure refinement*) sử dụng nhiều mức trừu tượng với mức độ chi tiết khác nhau. Mỗi lớp đặc tả là máy trạng thái mô tả hoàn chỉnh hệ thống. Vai trò các lớp như sau:

- Lớp trên cùng trừu tượng nhất và kết hợp nhiều kiểu dữ liệu, biến và các hàm vào trong một vài hàm đơn giản
- Các lớp kế tiếp bổ sung các chi tiết bằng các phân rã các hàm khái quát thành các đối tượng và hàm cụ thể. Các lớp sau là các mô tả cụ thể hơn của hệ thống và thỏa mãn các thuộc tính an toàn giống như lớp trên.
- Sau khi hoàn thành lớp sau thì lớp đặc tả trước đây hết vai trò. Lớp dưới cùng rất gần với các biến và hàm trong các đoạn mã của hệ thống. Như vậy, lớp này đảm

bảo các mô tả chi tiết và chính xác giao tiếp của hệ thống và giúp cho người thiết kế có thể triển khai được.

Kỹ thuật này không cho biết cách thức thiết kế hệ thống bên trong. Việc kiểm chứng cần sử dụng các kỹ thuật công nghệ phần mềm truyền thống như kiểm tra mã nguồn và kiểm thử.

5.3.2 Phân rã thuật toán

Kỹ thuật phân rã thuật toán (*Algorithm refinement*) cho phép mô tả một phần cấu trúc nội tại của hệ thống. Kỹ thuật này coi hệ thống như một chuỗi các máy trạng thái có phân lớp.

- Mỗi máy trạng thái sử dụng các chức năng do lớp trên cung cấp
- Việc triển khai các chức năng (*function*) bao gồm một chương trình khái quát sử dụng các chức năng có trong máy trạng thái ở bên dưới.
- Lớp thấp nhất cung cấp các chức năng nguyên thủy nhất cả hệ thống mà không thể phân rã thêm được nữa.

Hình 5-6 minh họa cho các bước khái quát nêu trên. Mỗi một bước ứng với một lớp người thiết kế mô tả máy trạng thái giống như trong kỹ thuật phân rã dữ liệu, thêm vào đó, bổ sung chương trình khái quát cho từng chức năng của máy trạng thái. Phần bổ sung này mô tả về thuật toán của các chức năng (hàm) theo dạng lời gọi hàm.

| Lớp | Các đặc tả chính tắc | Các chương trình khái quát | | |
|---------|--|---|---|---|
| | Giao tiếp với hệ thống ↓ | | | |
| N | Máy trạng thái mức cao Đặc tả giao tiếp func A func B | proc A_N call A_{N-1} call C_{N-1} return | proc B_N call B_{N-1} call A_{N-1} return | |
| $N - 1$ | Máy trạng thái trung gian func A func B func C | proc A_{N-1} call A_{N-2} call C_{N-2} return | proc B_{N-1} call B_{N-2} call A_{N-2} call A_{N-2} return | proc C_{N-1} call C_{N-2} return |
| $N - 2$ | Máy trạng thái trung gian | proc A_{N-1} | proc B_{N-1} | proc C_{N-1} |
| . | . | . | . | . |
| . | . | . | . | . |
| 1 | Máy trạng thái gốc | proc A_1 | proc B_1 | |
| 0 | Máy trạng thái gốc | proc A_0 | proc B_0 | |

Hình 5-6. Phân rã thuật toán theo các lớp

Việc chứng minh đặc tả sử dụng kỹ thuật này trước hết cần chứng minh các đặc tả mức cao nhất tương ứng với mô hình xây dựng. Tiếp theo, cũng tương tự chứng minh chương trình khái quát của lớp cao nhất phù hợp với đặc tả của nó khi biết các đặc tả các chức năng của lớp kế tiếp. Nhược điểm chính của kỹ thuật này là việc khó thực hiện các chứng minh thuật toán khái quát. Một số bài toán chứng minh thuộc lớp bài toán khó (*intractable*). Vấn đề khác đó là các đặc tả ở mức cao khá phức tạp do nó biểu diễn giao tiếp thực sự của hệ thống. Như vậy việc chứng minh mô hình mà hệ thống áp dụng với việc triển khai thực tế sẽ có thể khó khăn. Tuy vậy, việc này không hạn chế người dùng kết hợp các hai kỹ thuật phân rã dữ liệu và thuật toán để đạt được yêu cầu về đảm bảo tính phù hợp giữa mô hình lựa chọn và việc triển khai thực sự.

Ví dụ trong Hình 5-7 cho thấy cách áp dụng kỹ thuật phân rã thuật toán vào việc mô tả hệ thống file. Ở mức cao nhất (mức 2) người dùng sẽ chỉ quan tâm tới file và thư mục (*directories*). Các chức năng căn bản là tạo, xóa file và thư mục cũng như là các chức năng giám sát truy nhập. Mức tiếp theo (mức 1) mô tả các thao tác với file và mô tả file. Nói cách khác mức này đề cập tới cách thức xử lý cấu trúc lưu trữ file trên các thiết bị lưu trữ như làm việc với các thẻ file, các đơn vị cấp phát file. Các chức năng ở mức này vẫn không bị lệ thuộc vào các thiết bị lưu trữ vật lý cụ thể nào. Ở mức thấp nhất (mức 0), cách thức làm việc (chức năng) với thiết bị lưu trữ vật lý cụ thể được mô tả. Khi này, các chức năng sẽ thao tác lên các dữ liệu là các khối cụ thể trên thiết bị lưu trữ.

| Máy trạng thái khái quát | Các cấu trúc dữ liệu | Các hàm |
|--------------------------|---------------------------|---|
| Machine 2 | Files Directories | Create/delete files/directories Read/write files Access control functions |
| Machine 1 | Files File descriptors | Create/delete files Read/write files |
| Machine 0 | Disk blocks | Read/write disk blocks |

Hình 5-7. Phân rã thuật toán cho thao tác hệ thống file

5.4 Các kỹ thuật kiểm chứng mã chương trình

Tất cả các dự án phần mềm ít nhất đều hướng tới một kết quả cuối là các đoạn mã. Ở mức độ đoạn mã, mục tiêu trọng tâm là hạn chế và giảm thiểu các lỗi trong việc triển khai, nhất là những lỗi hay lỗ hổng phổ biến có thể phát hiện ra được nhờ công cụ quét mã nguồn. Các lỗi triển khai vừa nhiều vừa phổ biến và gồm cả những lỗi nội tiếng như tràn bộ đệm (*buffer-overflow*). Việc xác định các lỗi trong các đoạn mã của phần mềm cho phép xác định mức độ an toàn của phần mềm với các lỗi cũng như khả năng đáp ứng của phần mềm với các yêu cầu về an toàn với các đoạn mã trong quá trình xây dựng (viết mã).

Quá trình đánh giá mã nguồn cả thủ công hay tự động đều nhằm mục đích xác định các lỗi liên quan đến an toàn trước khi phần mềm được xuất xưởng. Tất nhiên, không có phương thuốc trị bách bệnh. Việc đánh giá mã nguồn là cần thiết nhưng chưa đủ để đạt được phần mềm an ninh vì các yếu tố sau:

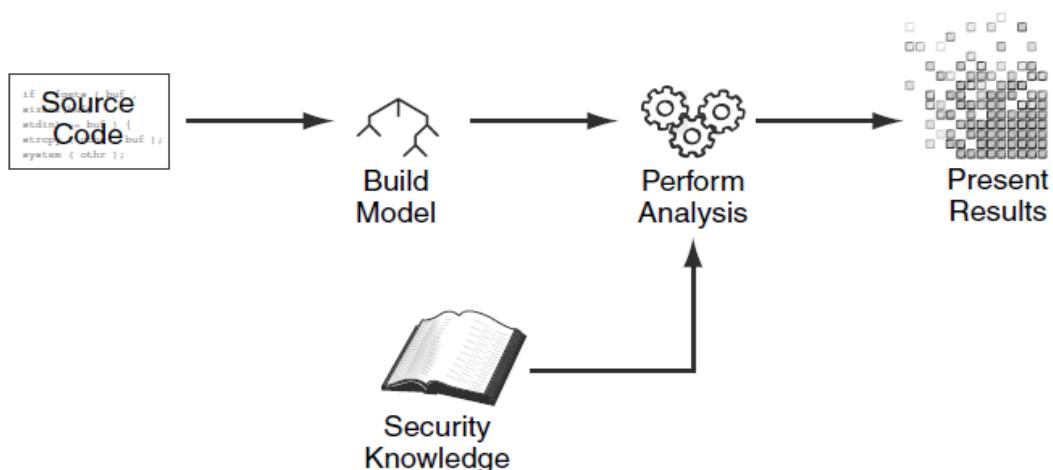
- Các lỗi an ninh là các vấn đề hiển hiện (cần phải xử lý) song các lỗ hổng thiết kế còn là vấn đề nghiêm trọng hơn. Các lỗi thiết kế hầu như không thể phát hiện thông qua việc đánh giá mã nguồn. Cách tiếp cận đầy đủ để đạt được phần mềm an toàn là kết hợp hài hòa giữa đánh giá mã nguồn và áp dụng quy trình phân tích thiết kế an toàn.
- Việc đánh giá mã nguồn hiển nhiên cần những tri thức về lập trình. Việc hiểu biết các cơ chế an toàn hay an toàn mạng không có ích nhiều với việc đánh giá mã nguồn. Nói cách khác, việc đánh giá mã nguồn tốt nhất là bắt đầu từ người viết ra chúng chứ không phải là các chuyên gia về an toàn.

Phản tiếp theo trình bày hai kỹ thuật phân tích chủ yếu là phân tích tĩnh và phân tích động. Phân tích tĩnh không cần thực thi chương trình hay đoạn mã cần đánh giá, ngược lại phân tích động sẽ thực thi các chương trình để đánh giá các hành vi của chương trình.

5.4.1 Phân tích tĩnh

Phân tích tĩnh để cập đến các kỹ thuật đánh giá mã nguồn nhằm cảnh báo các lỗ hổng an toàn tiềm tàng mà không thực thi chúng.

- Một cách lý tưởng các công cụ tự động có thể tìm kiếm các lỗi an ninh với mức độ đảm bảo nhất định về các lỗi này song việc này vượt quá khả năng của rất nhiều công cụ hiện thời.
- Các kỹ thuật kiểm tra phần mềm (*testing*) thông thường nhằm kiểm tra các hành vi (chức năng) với người dùng thông thường trong điều kiện thông thường nên rất khó để phát hiện ra các lỗi liên quan đến vấn đề an ninh và an toàn.



Hình 5-8. Mô hình phân tích tĩnh đoạn mã

Việc đầu tiên công cụ phân tích tĩnh cần làm là chuyển mã chương trình thành *mô hình chương trình (program model)* như trong Hình 5-8. Mô hình chương trình thực chất là cấu trúc dữ liệu biểu diễn đoạn mã cần phân tích. Việc phân tích được tiến hành kết hợp với các tri thức về vấn đề an toàn biết trước hoặc dựa trên kinh nghiệm. Bước cuối cùng là biểu diễn kết quả phân tích theo yêu cầu an toàn đề ra. Dạng cơ bản là cách cảnh báo, các công cụ cao cấp có thể cung cấp các phản ví dụ (khi áp dụng các công cụ dựa trên lô-gíc hay kiểm chứng mô hình).

Các kỹ thuật xây dựng mô hình phân tích bao gồm:

- Phân tích từ vựng (*lexical analysis*)
- Phân tích câu (*parsing*)
- Cú pháp khái quát (*abstract syntax*)
- Phân tích ngữ nghĩa (*semantic analysis*)
- Phân tích luồng điều khiển (*control flow*)
- Phân tích luồng dữ liệu (*data flow*)
- Phân tích lan truyền lỗi (*Taint propagation*)

a. *Phân tích từ vựng*

Chuyển đoạn mã thành chuỗi các thẻ (*token*) nhằm loại bỏ những thành phần không quan trọng trong đoạn mã chương trình. Các thẻ có thể chứa định danh (biến, tên hàm, ...) và vị trí của chúng trong đoạn mã. Đoạn mã dưới đây minh họa cho việc chuyển đổi dòng lệnh ra các thẻ.

```
if (ret) // probably true
mat[x][y] = END_VAL;

IF LPAREN ID(ret) RPAREN ID(mat) LBRACKET ID(x) RBRACKET LBRACKET
ID(y) RBRACKET EQUAL ID(END_VAL) SEMI
```

b. *Phân tích câu*

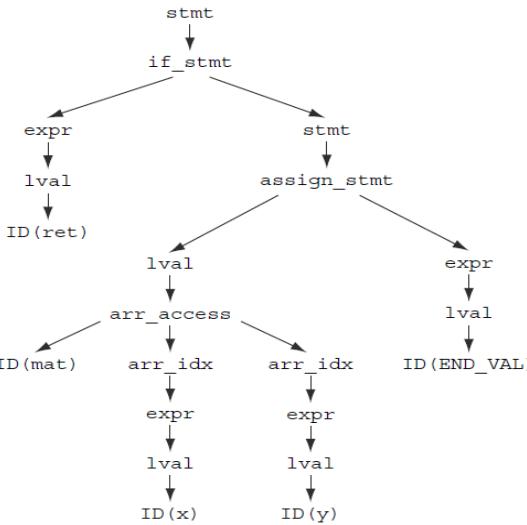
Bộ phân tích câu sử dụng ngữ pháp phi ngữ cảnh (*Context-free Grammar-CFG*) để đối sánh các chuỗi từ hay thẻ. Bộ ngữ pháp sử dụng các luật sinh để diễn tả các ký hiệu của ngôn ngữ (lập trình).

```

stmt := if_stmt | assign_stmt
if_stmt := IF LPAREN expr RPAREN stmt
expr := lval
assign_stmt := lval EQUAL expr SEMI
lval = ID | arr_access
arr_access := ID arr_index+
arr_idx := LBRACKET expr RBRACKET
```

Hình 5-9. Bộ luật sinh cây phân tích

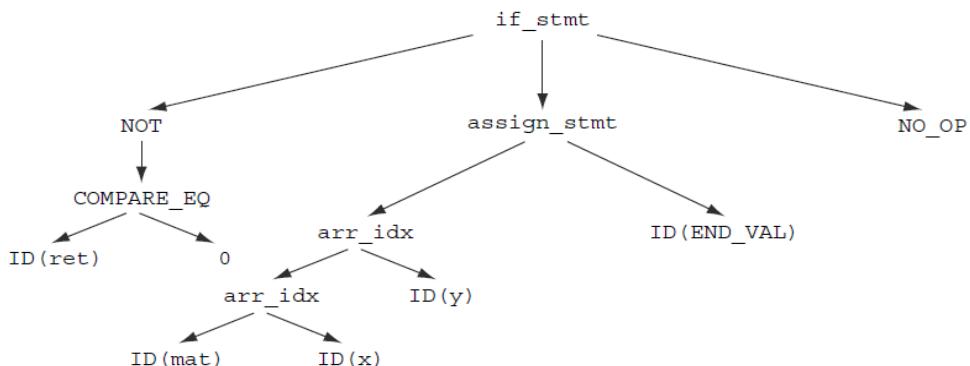
Bộ phân tích câu thực hiện việc suy diễn bằng cách đối sánh các ký hiệu (thẻ) với các luật sinh để tạo ra các cây phân tích. Hiện nay các bộ phân tích câu cho các ngôn ngữ như C, C++, hay Java được cung cấp dưới dạng mã nguồn mở hay cách dịch vụ cho phép người phát triển tùy biến theo yêu cầu của mình.



Hình 5-10. Cây cú pháp của câu lệnh

c. Cú pháp khái quát

Việc phân tích phức tạp có thể không phù hợp trên cây phân tích do mục tiêu của cây phân tích chỉ nhắm vào việc tách từ. Vì thế cây cú pháp khái quát cung cấp cấu trúc dữ liệu phù hợp với việc phân tích tiếp theo bằng cách loại bỏ các ký hiệu (thẻ) không phù hợp. Các ký hiệu trong cú pháp khái quát có thể ít hơn so với ngôn ngữ lập trình ban đầu.



Hình 5-11. Cây cú pháp khái quát

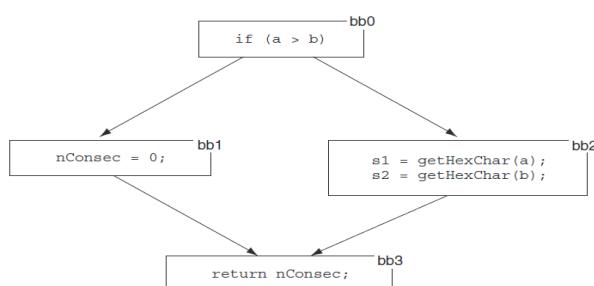
d. Phân tích ngữ nghĩa

Việc phân tích ngữ nghĩa có thể được bắt đầu bằng việc phân rã các ký hiệu (tên biến hay hàm) và kiểm tra kiểu dữ liệu. Việc phân tích này cho phép lập cấu trúc đoạn mã thông qua việc kiểm tra các lớp của đối tượng được sử dụng, đặc biệt hữu ích với lập trình hướng đối tượng. Chức năng phân tích ngữ nghĩa được sử dụng nhiều trong các bộ biên dịch (*compiler*) của các môi trường phát triển IDE (*Integrated Development*

Environment) vì nó cho phép kiểm tra các kiểu dữ liệu, phân rã các tên (định danh) trong chương trình mô tả các hằng, biến và hàm.

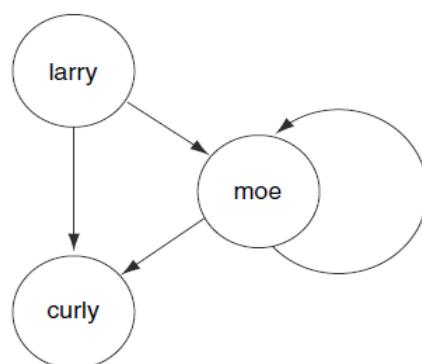
e. Phân tích luồng điều khiển

Mục tiêu của kỹ thuật này là để theo dõi các tình huống thực thi khác nhau của đoạn mã và thường được thực hiện bằng cách xây dựng đồ thị luồng điều khiển như trong ví dụ dưới đây. Việc xây dựng luồng điều khiển giống như việc xây dựng lưu đồ của chương trình từ đoạn mã, như vậy trái ngược với quá trình phát triển phần mềm thông thường (bắt đầu từ xây dựng lưu đồ rồi mới viết đoạn mã). Việc tái tạo lại luồng điều khiển cho người phân tích hình dung các khối cơ bản của đoạn mã và cách thức vận hành các khối này.



Hình 5-12. Luồng điều khiển

Đồ thị gọi hàm biểu diễn luồng điều khiển giữa các hàm hay phương thức và được xây dựng dựa trên đồ thị có hướng. Về cơ bản, đồ thị này thể hiện trạng thái hoạt động của chương trình thông qua việc hàm nào được kích hoạt (*gọi*). Kỹ thuật này thường kết hợp với việc phân tích luồng dữ liệu.



Hình 5-13. Đồ thị gọi hàm của ba phương thức larry, moe, curly

f. Phân tích luồng dữ liệu

Kỹ thuật này cho phép kiểm chứng cách dữ liệu di chuyển trong đoạn mã. Việc phân tích này thường kết hợp với luồng điều khiển để xác định vị trí bắt đầu và kết thúc của dữ liệu. Việc phân tích luồng có thể phát hiện những tình huống như sử dụng mật khẩu hay khóa cố định (*hard-coded*) trong đoạn mã.

g. Phân tích lan truyền lỗi

Kỹ thuật này được sử dụng để tìm hiểu các giá trị bên trong đoạn mã mà người tấn công có khả năng kiểm soát bằng cách sử dụng luồng dữ liệu. Việc này cần thông tin về việc biến chứa lỗi xuất hiện ở đâu trong chương trình và cách thức di chuyển trong chương trình. Mục đích của việc phân tích này là để hiểu cách thức các con trỏ có thể tham chiếu đến cùng vị trí nhó. Việc phân tích này rất quan trọng với việc phân tích lan truyền lỗi.

Việc phân tích tĩnh được sử dụng một cách rộng rãi hơn là mọi người biết. Điều này một phần vì có nhiều kiểu công cụ phân tích tĩnh hướng tới các mục tiêu khác nhau. Hình dưới đây liệt kê một số công cụ phân tích tĩnh hướng tới các mục tiêu khác nhau.

| Loại công cụ | Địa chỉ |
|--|--|
| <i>Kiểm tra kiểu lập trình</i> PMD Parasoft | http://pmd.sourceforge.net http://www.parasoft.com |
| <i>Kiểm chứng chương trình</i> Praxis High Integrity Systems Escher Technologies | http://www.praxis-his.com http://www.eschertech.com |
| <i>Kiểm tra thuộc tính</i> Polyspace Grammatech | http://www.polyspace.com http://www.gramatech.com |
| <i>Tìm lỗi</i> FindBugs Visual Studio 2005 \analyze | http://www.findbugs.org http://msdn.microsoft.com/vstudio/ |
| <i>Dánh giá an ninh</i> Fortify Software Ounce Labs | http://www.fortify.com http://www.ouncelabs.com |

Hình 5-14. Các công cụ phân tích tĩnh

Một dạng ứng dụng phân tích tĩnh được nhiều người biết tới đó chính là kiểm tra kiểu dữ liệu sử dụng trong chương trình. Các trình soạn thảo sẽ cảnh báo cho người lập trình về các tình huống không tương hợp về kiểu. Điều này có thể dẫn đến việc mất mát dữ liệu và thậm chí gây mất an toàn cho chương trình như trong trường hợp kiểm tra độ dài chuỗi. Một dạng khác là kiểm tra kiểu lập trình (*style checking*) mà cụ thể yêu cầu người lập trình tuân thủ đầy đủ các qui ước viết mã, đặt tên cũng như sử dụng các cấu trúc điều khiển.

Kiểm tra thuộc tính và kiểm chứng chương trình là ứng dụng phức tạp hơn của việc phân tích tĩnh. Mục tiêu là kiểm chứng đoạn mã xây dựng có phù hợp với các đặc tả của chương trình hay các qui định và ràng buộc đề ra (hay thuộc tính an toàn) với đoạn mã.

Nếu các đặc tả về chương trình bao trùm toàn bộ các chức năng của chương trình, các công cụ kiểm chứng có thể thực hiện việc kiểm chứng tương đương để chắc chắn đoạn mã và chương trình tương ứng chính xác với nhau.

Thông thường, các đặc tả (chính tắc) chỉ áp dụng cho một số chức năng thiết yếu của chương trình (hệ thống) nên chỉ áp dụng việc kiểm chứng một phần. Đôi khi còn được gọi là kiểm tra thuộc tính. Đây là các điều kiện mà chương trình phải tuân theo. Các công cụ phức tạp cho phép phân tích cả về mặt thời gian hay cung cấp các ví dụ phản chứng giúp cho người lập trình hình dung tốt hơn về những vi phạm với yêu cầu đề ra. Tìm lỗi cũng là một ứng dụng quan trọng của phân tích tĩnh. Mục tiêu là chỉ ra tình huống mà chương trình có thể hoạt động không như người lập trình dự định. Công cụ tìm lỗi lý tưởng là chứng minh đủ và cần cung cấp phản ví dụ. Điều này cho biết tình huống có thể khi xảy ra lỗi.

Đánh giá an ninh là mục tiêu thu hút được nhiều người chú ý tới các công cụ phân tích tĩnh. Các công cụ thời kỳ đầu tập trung vào việc quét các hàm hay bị lạm dụng như `strcpy` và cần tiến hành việc đánh giá an ninh thủ công. Điều này đôi khi là cho người dùng coi các lỗi phát hiện được như là các lỗi lập trình hơn là các vấn đề an ninh cần được chú ý. Các công cụ mới thường kết hợp việc tìm lỗi và kiểm tra thuộc tính. Việc kiểm tra tràn bộ đệm có thể được diễn giải như là yêu cầu lập trình sao cho chương trình không truy nhập ngoài không gian nhớ được cấp phát.

Thách thức lớn nhất cho việc phân tích tĩnh là lựa chọn cách thức biểu diễn chương trình phù hợp mà nó cho phép cân đối giữa mức độ chính xác, khả năng mở rộng và kỹ thuật phân tích nhằm phát hiện vấn đề cần quan tâm. Bài toán cơ bản cho phân tích tĩnh thường sử dụng các kỹ thuật áp dụng trong trình dịch bao gồm việc phân tích luồng điều khiển và dữ liệu. Các công cụ phân tích tĩnh tốt thường sử dụng kiểu luật. Điều này cho phép các công cụ phân tích lập mô hình về môi trường hoạt động và kết quả của các lời gọi hệ thống và thư viện lập trình. Luật cũng cho phép định nghĩa các thuộc tính an toàn một cách thuận tiện và các công cụ có thể kiểm chứng được.

Như vậy công cụ phân tích tĩnh tốt cần dễ sử dụng ngay cả với những người không có chuyên môn về vấn đề an toàn. Điều này nghĩa là các kết quả phân tích cần dễ hiểu với người lập trình thông thường, có thể không hiểu biết sâu về an toàn, và hướng dẫn người dùng về thói quen lập trình an toàn. Đặc điểm quan trọng khác là dạng tri thức (tập các luật) công cụ hỗ trợ. Tập luật tốt trợ giúp người dùng đắc lực trong quá trình phân tích đoạn mã. Cuối cùng các công cụ tốt cho phép phát hiện triệt để các lỗi an toàn phổ biến.

5.4.2 Phân tích động

Quá trình phân tích tĩnh đánh giá chương trình giống như trình biên dịch đọc chương trình này. Cách phân tích khác kiểm tra chương trình bằng cách giống như môi trường hoạt động sử dụng các chương trình này bằng cách xem xét mã thực thi hay `bytecode` (với ngôn ngữ Java). Quá trình phân tích động phần mềm được thực hiện bằng cách chạy

các đoạn mã trên bộ xử lý vật lý hay ảo. Các đoạn mã hiện nay ngày càng sử dụng thư viện liên kết động hay theo yêu cầu, việc phân tích tĩnh không thể hiện đầy đủ các khía cạnh của đoạn mã. Tuy nhiên, việc phân tích động nên được thực hiện sau khi hoàn thành việc phân tích tĩnh do việc thực thi các đoạn mã có thể làm tổn hại đến hệ thống.

Trên thực tế, để phân tích và đánh giá tính an toàn của hệ thống, có thể cần thực hiện thêm:

- Quét lỗ hổng
- Kiểm thử xâm nhập

Việc phân tích động có thể được thực hiện thông qua phần mềm sửa lỗi debugger. Sửa lỗi có thể hoạt động ở mức cao, mức mã nguồn hay mức thấp hơn như mã máy. Công cụ sửa lỗi cung cấp các cơ chế bẫy như chạy từng bước, chạy qua hay lấy thông tin về chương trình tại thời điểm chạy cho phép người dùng kiểm tra hệ quả của từng bộ phận trong chương trình được phân tích.

Việc phân tích mã chương trình sau khi biên dịch có ưu điểm:

- Người dùng không phải dự đoán xem trình biên dịch sẽ diễn giải các câu lệnh như thế nào do các câu lệnh của chương trình đã được dịch, Như vậy, việc này loại trừ tính mơ hồ.
- Mã nguồn chương trình không phải lúc nào cũng có được trong khi mã thực thi hay mã bytecode thì lại có sẵn.

Tuy nhiên việc phân tích động cũng có nhược điểm:

- Hiểu được các đoạn mã đã được biên dịch có thể rất khó khăn và một số dang mã máy rất khó dịch ngược như tập lệnh có độ rộng lệnh thay đổi như Intel x86. Điều này là vì ý nghĩa của đoạn mã thay đổi khi thay đổi vị trí dịch ngược của chuỗi byte biểu diễn mã lệnh.
- Việc sử dụng mã đã biên dịch khiến cho việc hiểu được ngữ nghĩa (ý định ban đầu của người lập trình) khó khăn hơn nhiều. Đặc biệt khi trình biên dịch thực hiện việc tối ưu hóa đoạn mã thực thi dẫn đến đoạn mã rất khác so với mã nguồn.

Trong tương lai việc phân tích động tập trung vào các nghiên cứu:

- Khai thác hiệu quả thông tin chạy chương trình (run-time) để tối ưu hóa chương trình.
- Trợ giúp người lập trình hiểu, sửa chữa và nâng cấp phần mềm đặc biệt về khía cạnh an toàn.
- Cải thiện cách thức thu thập thông tin chạy trong quá trình phân tích động bằng cách kết hợp phần cứng và phần mềm nhằm giảm chi phí. Với một số phần mềm độc hại có khả năng chống lại việc thu thập thông tin dạng debug thông thường thì cơ chế giải pháp kết hợp có giá trị rất lớn trong việc phân tích hành vi của phần mềm.

5.5 Kết luận

Từ khía cạnh thiết kế, việc đánh giá an toàn cho biết khả năng và mức độ đáp ứng của phần mềm với các yêu cầu về an toàn toàn đề ra. Các kỹ thuật kiểm chứng giúp phát hiện và xác định các thiếu sót trong quá trình phát triển và triển khai của phần mềm so với các yêu cầu về an toàn, như vậy hạn chế việc tồn tại các điểm yếu hay lỗ hổng mà có thể bị khai thác một cách vô tình hay có chủ đích.

Việc đánh giá an toàn cũng cần được thực hiện với các đoạn mã ngay cả khi không có các thông tin đặc tả về đoạn mã đó bằng cách áp dụng các kỹ thuật phân tích tĩnh và động. Việc này không chỉ có tác dụng kiểm chứng tính an toàn của đoạn mã mà còn có tác dụng nâng cao chất lượng của phần mềm và cải thiện thói quen lập trình an toàn.

Việc đảm bảo an toàn hệ thống (*system assurance*) hướng đến kiểm chứng hệ thống tuân thủ các mục tiêu an toàn mong muốn. Các kỹ thuật và công cụ trình bày trong các phần trước của chương hỗ trợ cho việc đảm bảo và đánh giá an toàn hệ thống. Cách thức mô tả các yêu cầu về an toàn và các đặc tả hệ thống giúp cho việc kiểm chứng cũng đã được giới thiệu. Qua đó giúp đánh giá mức độ đáp ứng các yêu cầu về an toàn của hệ thống.

Trên thực tế, các tổ chức quốc gia và quốc tế nỗ lực phát triển các tiêu chuẩn cho các mô tả yêu cầu an toàn/an ninh và cách thức đánh giá để kiểm chứng việc triển khai các yêu cầu này. Các tiêu chuẩn này bao phủ từ các chức năng xác thực mật khẩu cho đến giám truy nhập và cấu hình hệ thống. Các bộ tiêu chuẩn tiêu biểu liên quan đến an toàn hệ điều hành có thể kể đến là Tiêu chuẩn đánh giá hệ thống máy tính tin cậy TCSEC (*Trusted Computer System Evaluation Criteria*) hay Các tiêu chuẩn phổ quát (*Common Criteria*). Việc áp dụng các bộ tiêu chuẩn giúp cho cả người dùng và người thiết kế nắm được mức độ an toàn của hệ thống dự định sử dụng hay triển khai.

5.6 Câu hỏi ôn tập

- 1) Trình bày các đặc trưng của đặc tả an toàn?
- 2) Trình bày khái niệm máy chứng minh (theorem prover)?
- 3) Trình bày khái niệm kiểm chứng mô hình (model checking)?
- 4) Trình bày khái niệm phân rã cấu trúc dữ liệu?
- 5) Trình bày khái niệm phân rã chương trình?
- 6) Trình bày sự cần thiết áp dụng kiểm chứng mã chương trình?
- 7) Khái niệm phân tích tĩnh?
- 8) Khái niệm phân tích động?

Tài liệu tham khảo

- [1] Andrew S. Tanenbaum, Herbert Bos, *Modern Operating Systems 4th Edition*, Pearson Education, Inc 2015
- [2] Abraham Silberschatz, Peter B. Galvin, Greg Gagne, *Operating System Concepts Essentials*, John Wiley & Sons Inc., 2014.
- [3] Daniel Jackson, “Alloy: a lightweight object modelling notation,” ACM Transactionson Software Engineering and Methodology (TOSEM), vol. 11, no. 2, pp. 256–290, 2002.
- [4] Gustavo Duarte, *CPU Rings, Privilege, and Protection*, 2008
- [5] Intel Co., *Intel x64 and IA-32 Architectures Software Developer’s Manual*, Intel Co. 2016
- [6] Morrie Gasser, *Building a secure computer system*, Library of congress, ISBN 0-442-23022-2
- [7] Mehedi Al Mamun, *Operating Systems Security: Linux*, LAP Lambert Acad. Publishing, 2011.
- [8] Seymour Bosworth. M.E. Kabay, Eric Whyne, *Computer Security Handbook 6th Edition*, John Wiley & Sons, 2014.
- [9] Trent Jaeger, *Operating System Security*, Morgan & Claypool Publishers, 2008.
- [10] Will Arthur & David Challener, A Practical Guide to TPM 2.0: Using the New *Trusted Platform Module in the New Age of Security*, © 2015 by Apress Media