การพัฒนาระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่าย

Development of Security Policy Testing System for Network Control Device

โดย

นายชินวัฒน์ ติวิธมไหศูรย์

รหัส 43067055

อาจารย์ที่ปรึกษา

อาจารย์อัครินทร์ คุณกิตติ

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 2 ปีการศึกษา 2544

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนาระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่าย
นักศึกษา	นายชินวัฒน์ ติวิธมไหศูรย์
อาจารย์ที่ปรึกษา	อาจารย์อัครินทร์ คุณกิตติ
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2544

บทคัดย่อ

เราเตอร์ (router) เป็นอุปกรณ์หนึ่งที่สามารถนำมาใช้ในการควบคุมเครือข่ายได้ สิ่งที่เราเตอร์ใช้ในการควบคุมนั้นคือการคอยตรวจสอบและควบคุมการจราจรที่เข้าออกเครือข่ายทีละแพคเกตตามกฎ (ACL - Access Control List) ที่กำหนดไว้ตามนโยบายความปลอดภัยของระบบโดยผู้บริหารระบบ โครงการพัฒนาระบบงานนี้ได้ทำการศึกษาหลักการของนโยบายความปลอดภัย การทำงานและการปรับเปลี่ยนการกรองแพคเกตบนเราเตอร์ พบว่าการที่จะทดสอบนโยบายความปลอดภัยนี้ว่าสามารถทำงานได้จริง มีประสิทธิภาพ และมีข้อบกพร่องอย่างไรบ้างจะต้องอาศัยระบบที่สามารถสร้างและดักจับแพคเกตในเครือข่ายตามรูปแบบการจราจรตามรูปแบบที่ได้กำหนดไว้ได้ จึงได้ทำการออกแบบและพัฒนาระบบงานที่ช่วยในการทดสอบอุปกรณ์ควบคุมเครือข่ายที่มีพื้นฐานจากการตรวจสอบแพคเกต ที่มีความสามารถในการวัดผล และวิเคราะห์ผลลัพธ์ที่ได้ ซึ่งสามารถทำได้ง่ายผ่านทางการควบคุมรวม การออกแบบระบบทดสอบนโยบายความปลอดภัยนี้มีการทำงานแบ่งเป็น 5 ส่วนด้วยกันคือ ส่วนที่จัดการนโยบายความปลอดภัยและรูปแบบการจราจรสำหรับการทดสอบ ส่วนที่ตรวจสอบแพคเกตที่ส่งเข้ามา ส่วนที่จัดการ Syslog ส่วนที่วิเคราะห์และรายงานผล และส่วนที่บริหารควบคุมระบบทั้งหมด ในด้านการพัฒนานั้นครอบคลุมส่วนแรกที่จัดการนโยบายความปลอดภัยและรูปแบบการจราจรสำหรับการทดสอบ โดยโปรแกรมทั้งหมดพัฒนาด้วยภาษา C โดยใช้คอมไพเลอร์ GCC บนระบบปฏิบัติการ Linux ในส่วนของที่ติดต่อกับผู้ใช้นั้นรับคำสั่งผ่านบรรทัดควบคุม (command line) เป็นหลัก

การทำงานของส่วนที่แปลงนโยบายความปลอดภัยนั้นใช้หลักการของการแปลงภาษา โดยทำการแปลงนโยบายความปลอดภัยที่ใช้งานบนอุปกรณ์ที่เป็นแบบ ACL ให้อยู่ในรูปแบบกลางแบบ SNORT เพื่อใช้ในการสร้างรูปแบบการจราจรสำหรับการทดสอบ การสร้างจราจรสำหรับการ

ทดสอบนั้นอาศัยการติดต่อกับเครือข่ายในระดับต่ำที่ต้องเตรียมโครงสร้างของแพคเกตเองทำให้สามารถกำหนดค่าต่างๆ ในแพคเกตได้ตามที่ต้องการ ในการทดสอบนั้นได้ทำการทดสอบทั้งแบบแยก และแบบโดยรวมกับเราเตอร์จริงๆ ผลที่ได้นั้นได้รับการตรวจสอบแล้วพบว่ามีความถูกต้องและเป็นไปตามรูปแบบการจราจรที่กำหนดไว้

จากการออกแบบ การพัฒนา และการทดสอบที่ได้ทำ สามารถสรุปได้ว่าโปรแกรมที่พัฒนาขึ้นมานี้สามารถนำไปใช้ในการทดสอบอุปกรณ์ควบคุมเครือข่ายได้ตามจุดประสงค์ที่ตั้งไว้สำหรับการพัฒนาต่อเนื่องนั้นสามารถนำไปพัฒนาต่อให้ครบทั้งระบบได้ตามที่ได้ออกแบบไว้

Title	Development of Security Policy Testing System for Network Control Device
Student	Mr. Chinnawat Tivitmahaisoon
Advisor	Mr. Akharin Khunkitti
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2001

ABSTRACT

Router, one of device that can be used to control the network, uses Access Control List (ACL) to determine and control incoming and outgoing packet of the traffic on its interface. The system administrator defines this ACL according to the system security policy. This project studies concept and methodology of security policy, working mechanism and configuration of packet filtering on the router. This project covers design and development of a Security Policy Testing System that can generate, capture network traffic pattern, monitor and analyze the result against security policy to test the functionality and efficiency of security policy for network control device. Tester can manage and control this testing system from central management console via command line. Designed Security Policy Testing System consists of 5 parts; security policy and traffic management, traffic capturing, syslog management, analyzer and report generator, and system management. Development covers the first part for security policy and traffic management which are developed in C and compiled using GCC on Linux platform. System Administrator can manage Security Policy Testing System primary from command line.

Security policy translation is working based on the concept of compilation. Security policy in implementation level or in ACL format will be compiled to medium security policy format in SNORT format. This medium SNORT format will be used to generate traffic pattern. To generating packet from traffic pattern, sender program needs to access network using low level API to construct packet structure according to the the traffic pattern and send the packet out to the network control device. Functional verification of the program is done on module and

integrated test with router in test environment. The result confirms the correctness of the algorithm. Output packets are matched to the defined traffic pattern.

Conclusion for design, development and testing is that the programs delivered from this project can be used to test the network control device as stated in the objective. Continue development can be done according to the design.

กิตติกรรมประกาศ

ผลงานโครงการพัฒนาระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์เครือข่ายที่ได้จัดทำขึ้นนี้ ข้าพเจ้าได้รับการสนับสนุน ช่วยเหลือ และคำแนะนำจากบุคคลหลายท่าน จึงเป็นผลให้โครงการนี้สามารถสำเร็จลุล่วงไปได้ด้วยดี ข้าพเจ้าจึงขอขอบพระคุณบุคคลต่างๆ ดังต่อไปนี้

- บิดา มารดา และน้องสาว ที่คอยให้ความดูแล คอยให้ความช่วยเหลือและกำลังใจ
- อาจารย์อัครินทร์ คุณกิตติ อาจารย์ที่ปรึกษาโครงการ ที่คอยให้คำแนะนำ คอยให้คำปรึกษา และชี้แนะแนวทางต่างๆ เป็นอย่างดี
- อาจารย์ทุกๆ ท่านที่ได้มอบความรู้ทางวิชาการต่างๆ ซึ่งสามารถนำมาใช้ในการพัฒนาโครงการ รวมทั้งทางคณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่เปิดโอกาสให้ได้มีการพัฒนาโครงการระบบงานนี้ขึ้น
- บริษัท เอสโซ่ (ประเทศไทย) จำกัด มหาชน ที่ให้การสนับสนุนอุปกรณ์เครือข่ายที่ใช้ในการพัฒนาและทดสอบระบบ
- เพื่อนๆ IS 9 สมทบ ที่คอยเป็นกำลังใจและให้คำปรึกษา

ชินวัฒน์ ติวิธมไหศูรย์
ผู้จัดทำ

สารบัญ

หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญภาพ

หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

ระบบสารสนเทศส่วนใหญ่ที่มีอยู่ในปัจจุบันนั้นมักจะเป็นระบบเปิด (open system) ซึ่งมีการติดต่อสื่อสารระหว่างสิ่งต่างๆ (entity) ทั้งภายในระบบด้วยกันเองหรือกับสิ่งที่อยู่ภายนอกระบบ เพื่อทำการแลกเปลี่ยนข้อมูลสารสนเทศเพื่อทำให้ระบบนั้นทำงานได้ตามที่ต้องการ เครือข่ายจึงได้ถูกสร้างขึ้นเพื่อเป็นโครงสร้างพื้นฐานสำคัญในการเป็นสื่อกลางในการสื่อสารและเชื่อมโยงส่วนต่างๆ ในระบบเข้าด้วยกัน ระบบเครือข่ายที่เชื่อมต่อกันอย่างทั่วถึงทั้งภายในและภายนอกระบบนี้เป็นช่องทางหนึ่งที่ทำให้เกิดการคุกคามจากการบุกรุกทั้งภายในและภายนอกได้ ภัยที่เกิดจากการรุกรานต่อระบบสารสนเทศไม่ว่าจะเป็นการเข้าถึงโดยไม่ได้รับอนุญาต การแก้ไขเปลี่ยนแปลงข้อมูล หรือการลักลอบนำข้อมูลไปใช้ ย่อมส่งผลให้เกิดความเสียหายต่อระบบได้ ผู้บริหารควบคุมระบบจึงต้องกำหนดมาตรการและนโยบายความปลอดภัยที่สามารถป้องกัน ตรวจสอบภัยรุกรานเหล่านี้ให้ได้อย่างทันท่วงที

เราเตอร์เป็นอุปกรณ์เครือข่ายที่ทำงานอยู่ในระดับชั้นที่ 3 หรือชั้นเครือข่ายตามมาตรฐานOSI โดยทำหน้าที่ส่งต่อแพคเกต (packet) ข้อมูลจากเครือข่ายหนึ่งไปอีกเครือข่ายหนึ่ง เนื่องจากมันทำงานตรวจสอบต้นทางและปลายทางทีละแพตเกต ดังนั้นเราเตอร์จึงสามารถนำมาใช้ในเป็นอุปกรณ์ที่ใช้ในการควบคุมเครือข่ายได้อีกด้วย การทำงานของเราเตอร์ในการควบคุมเครือข่ายนี้อาศัยการกลั่นกรองจราจรในเครือข่ายตามกฎในการควบคุมที่ได้กำหนดไว้ตามนโยบายความปลอดภัยของระบบ กฎที่ใช้ในการควบคุมเราเตอร์นี้เรียกว่า ACL (Access Control List) ซึ่งมีลักษณะเป็นการกำหนดเงื่อนไขในการตรวจสอบแพคเกตทีละข้อในแต่ละบรรทัด รูปแบบของการเขียน ACL นั้นจะมีความแตกต่างกันไปขึ้นอยู่กับซอฟต์แวร์ที่ใช้ในควบคุมเราเตอร์ เนื่องจากเราเตอร์ส่วนใหญ่ที่มีการใช้งานกับอย่างแพร่หลายนั้นเป็นของบริษัท Cisco System ดังนั้น ACL ที่ใช้กันจึงเป็นไปตามรูปแบบที่ใช้ในระบบปฏิบัติการควบคุมเราเตอร์ที่เรียกว่า IOS (Internetwokring Operating System) ของทาง Cisco ด้วยไปโดยปริยาย

การทดสอบ ACL ที่กำหนดขึ้นตามนโยบายความปลอดภัยทางเครือข่ายนั้นนับว่าเป็นงานที่ต้องอาศัยเวลาและทรัพยากรมากเนื่องจากต้องมีการเตรียมอุปกรณ์เครือข่ายและเครื่องคอมพิวเตอร์จำนวนหนึ่งที่กำหนดหมายเลข I P ให้ครอบคลุมเพียงพอที่จะใช้ในการตรวจสอบประเมินเงื่อนไข

ต่างๆ ใน ACL ที่จะทดสอบ อุปกรณ์เครือข่ายสำหรับสร้างแพคเกตส่วนใหญ่ที่มีอยู่นั้นก็ไม่สามารถสร้างแพตเกตตามนโยบายความปลอดภัยที่กำหนดไว้ได้ ตัวภาษาในการกำหนด ACL และนโยบายความปลอดภัยเองก็มีหลากหลายและความแตกต่างกันในระดับการนำไปใช้งานจริง นอกจากนี้การสร้างการจราจรซึ่งเป็นภาระงาน (workload) สำหรับการทดสอบในแต่ละครั้งสำหรับการวัดผลนั้นก็ไม่สามารถทำซ้ำได้เหมือนเดิม ภาระงานที่แตกต่างกันนี้อาจจะส่งผลให้ผลลัพธ์มีความที่ได้นั้นมีความแม่นยำน้อยลง

1.2 วัตถุประสงค์ของการพัฒนาระบบงาน

1.2.1 ช่วยอำนวยความสะดวกให้ผู้บริหารควบคุมระบบเครือข่าย และผู้ทดสอบอุปกรณ์ควบคุมเครือข่าย ในการทดสอบนโยบายความปลอดภัยที่อยู่ในรูปของ ACL ที่ทำงานบนเราเตอร์ได้โดยไม่ต้องอาศัยเครื่องคอมพิวเตอร์หลายๆ เครื่อง สามารถสร้างภาระงานจาก ACL ได้ทันที

1.2.2 สามารถตรวจสอบประสิทธิภาพ รวมทั้งข้อจำกัดในการทำงานของเราเตอร์ในการกลั่นกรองแพตเกตได้

1.2.3 สามารถสร้างภาระงานในรูปของการจราจรในเครือข่ายแบบแพคเกต ที่สามารถกำหนดโครงสร้างแพคเกตนั้นได้ในระดับต่ำ รวมทั้งสามารถกำหนด IP address ของต้นทางและปลายได้อย่างอิสระไม่ขึ้นกับ IP address จริงของเครื่องต้นทาง หรือที่เรียกว่าการปลอมแปลง IP (IP Spoofing)

1.2.4 สามารถสร้างภาระงานโดยไม่ขึ้นกับรูปแบบของนโยบายความปลอดภัยที่ใช้ในระบบที่จะถูกทดสอบ โดยจะมีส่วนที่ทำหน้าที่แปลงภาษาในการกำหนดนโยบายความปลอดภัยในระดับใช้งานจริงนั้น ให้อยู่ในรูปแบบของกฎที่ใช้ภายในระบบทดสอบก่อน

1.2.5 ระบบทดสอบนี้สามารถสร้างภาระงานทดสอบ สำหรับอุปกรณ์ควบคุมเครือข่ายแบบอื่นๆ อย่างเช่น ไฟล์วอล (Firewall) หรืออุปกรณ์ตรวจสอบผู้รุกราน (Intrusion Detection System) เป็นต้น

1.3 เป้าหมายของการพัฒนาระบบงาน

โครงการพัฒนาระบบงานนี้มีเป้าหมายเพื่ออำนวยความสะดวกแก่ผู้บริหารระบบและผู้ทดสอบอุปกรณ์ควบคุมเครือข่าย ในการทดสอบอุปกรณ์ควบคุมเครือข่ายแบบกลั่นกรองที่สามารถทำได้ง่าย ใช้ทรัพยากรระบบไม่มากนัก รวมทั้งใช้ได้อย่างกว้างขวาง แต่ก็ขึ้นกับรูปแบบของภาษาที่

ใช้ในการกำหนดนโยบายควบคุมความปลอดภัยในระบบเครือข่ายซึ่งจะถูกแปลงให้อยู่ในรูปของภาษากลาง ดังในโครงการนี้ซึ่งใช้ภาษาในรูปแบบของ ACL ที่ใช้สำหรับเราเตอร์ของทาง Cisco

1.4 ขอบเขตของการศึกษาและพัฒนาระบบงาน

โครงการพัฒนาระบบงานซอฟต์แวร์ที่ใช้ในการทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่ายนี้ จะทำการศึกษาและพัฒนาระบบงานเพื่อสร้างภาระงานในการทดสอบระบบ วัดผล และวิเคราะห์ผลลัพธ์โดยอาศัยวิธีการแบบการตรวจวัด (measurement) โดยทำงานร่วมกับอุปกรณ์ควบคุมเครือข่ายแบบเราเตอร์ของทาง Cisco โดยการศึกษาจะมุ่งเน้นไปในการสร้างและรับแพคเกตในระดับต่ำที่เป็นส่วนประกอบของภาระงาน และการสร้างรูปแบบของการจราจรจากนโยบายความปลอดภัยที่ได้กำหนดไว้สำหรับการทดสอบ

ในด้านของการออกแบบระบบนั้น ระบบทดสอบนี้จะแบ่งออกเป็น 5 ส่วนด้วยกันคือ ส่วนที่จัดการนโยบายความปลอดภัยและการจราจรสำหรับการทดสอบ (Manage Security Policy and Traffic) ส่วนที่ตรวจสอบแพคเกตที่ส่งเข้ามา (Capture Traffic) ส่วนที่จัดการ Syslog (Manage Syslog) ส่วนที่วิเคราะห์และรายงานผล (Analyze and Generate Report) และส่วนที่บริหารควบคุมระบบทั้งหมด (Manage and Control System) โดยจะมีการแยกส่วนการทำงานออกไปยังเครื่องคอมพิวเตอร์อย่างน้อยสองเครื่อง คือเครื่องที่จะภายในเครือข่ายที่ถูกควบคุม และเครื่องที่อยู่ภายนอกเครือข่ายที่ถูกควบคุม โดยมีเราเตอร์เป็นอุปกรณ์ในการควบคุมเครือข่ายอยู่ระหว่างเครือข่ายทั้งสองนี้ การจราจรสำหรับการทดสอบที่สร้างขึ้นนั้นจะถูกส่งผ่านเราเตอร์ไปยังเครื่องที่ทำหน้าที่เป็น ส่วนตรวจวัดอยู่ โดยส่วนที่พัฒนาสำหรับการทดสอบและใช้งานในโครงงานนี้ประกอบไปด้วยเฉพาะส่วนที่จัดการนโยบายความปลอดภัยและการจราจรสำหรับการทดสอบเท่านั้น

1.5 ขั้นตอนการศึกษาและพัฒนาระบบ

ในการศึกษาและพัฒนาระบบนี้ได้มีการกำหนดขั้นตอนไว้ดังต่อไปนี้

ขั้นตอนที่ 1 ศึกษาหลักการของความปลอดภัยและภาษาในการกำหนดนโยบายความปลอดภัย รวมทั้งขั้นตอนในการกำหนดความนโยบายความปลอดภัย และทำความเข้าใจกับภาษาในการกำหนดนโยบายความปลอดภัยแบบต่างๆ นอกจากนี้ยังได้เลือกรูปแบบของภาษาในการกำหนดนโยบายความปลอดภัยที่ใช้ในระบบ

ขั้นตอนที่ 2 ศึกษาระบบที่จะใช้การทดสอบนโยบายความปลอดภัย กำหนดเป้าหมาย ขอบเขตของระบบ และทำความเข้าใจหลักการทำงานของระบบที่จะทดสอบ ซึ่งในที่นี้คือระบบการควบคุมเครือข่ายที่มีการใช้เราเตอร์เป็นส่วนประกอบหลัก

ขั้นตอนที่ 3 วิเคราะห์และออกแบบระบบจากที่ได้ทำการศึกษาในขั้นตอนที่ผ่านมา ศึกษาความเป็นไปได้ในการพัฒนา และสภาพแวดล้อมและเครื่องมือในการพัฒนา ทำการสร้างต้นแบบ (prototype) ของส่วนที่สำคัญเพื่อใช้ในการพัฒนาโปรแกรมในขั้นตอนต่อไป ภายใต้ขอบเขตของการศึกษาและพัฒนาระบบที่ได้กำหนดไว้

ขั้นตอนที่ 4 พัฒนาและทดสอบโปรแกรมในแต่ละส่วนตามที่ได้พัฒนาต้นแบบและออกแบบไว้

ขั้นตอนที่ 5 ทดสอบและตรวจสอบข้อผิดพลาดโดยรวมทั้งหมดในสภาพแวดล้อมควบคุม โดยนำผลลัพธ์ที่ได้ในใช้ในการปรับแต่งและแก้ไขเพื่อพร้อมจะนำไปใช้งานจริงต่อไป

ขั้นตอนที่ 6 ทดสอบการใช้งานในสภาพแวดล้อมจริง ศึกษาผลกระทบ สรุปผลการทดสอบและประเมินผลการพัฒนาระบบ

1.6 รายละเอียดของเครื่องคอมพิวเตอร์และเครื่องมือที่ใช้ในการพัฒนาระบบ

เนื่องจากโปรแกรมที่เป็นส่วนประกอบของระบบงานนี้ต้องมีการติดต่อกับระบบเครือข่ายในระดับต่ำ เพื่อให้มีสภาพแวดล้อมในการพัฒนาที่เหมาะสม สามารถควบคุมตัวแปรระบบ และไม่มีผลกระทบต่อการใช้งานจริง จึงได้ทำการพัฒนาระบบในสภาพแวดล้อมที่แยกออกมาเป็นเอกเทศ โดยมีเครื่องคอมพิวเตอร์และเครื่องมือในการพัฒนาดังต่อไปนี้

1.6.1 เครื่องคอมพิวเตอร์สำหรับควบคุมจากส่วนกลาง สร้างภาระงาน และรายงานผล

1.6.1.1 คุณสมบัติด้านฮาร์ดแวร์ (Hardware)

CPU : Intel Pentium III Mobile Processor 700MHz

RAM : SDRAM PC133 128MB

Harddisk : IDE 10GB UDMA4

NIC : 3COM 10/100 Mbps

1.6.1.2 ระบบปฏิบัติการ (Operating System)

Linux Redhat 7.2 kernel vesion 2.4.7-10

1.6.2 เครื่องคอมพิวเตอร์สำหรับตรวจสอบภาระงาน

1.6.2.1 คุณสมบัติด้านฮาร์ดแวร์ (Hardware)

CPU : Intel Pentium III Mobile Processor 600MHz

RAM : SDRAM PC133 128MB

Harddisk : IDE 15GB UDMA4

NIC : 3COM 10/100 Mbps

1.6.2.2 ระบบปฏิบัติการ (Operating System)

Linux Redhat 7.2 kernel vesion 2.4.7-10

1.6.3 เครื่องคอมพิวเตอร์สำหรับตรวจสอบโปรแกรมและแพคเกต

1.6.3.1 คุณสมบัติด้านฮาร์ดแวร์ (Hardware)

CPU : Intel Pentium III Processor 700MHz

RAM : SDRAM PC133 512MB

Harddisk : IDE 15GB UDMA4, 2 x 30GB UDMA5

NIC : Intel Pro/100+

1.6.3.2 ระบบปฏิบัติการ (Operating System)

Windows NT 4.0 SP4 server และ Linux Redhat 7.2 kernel vesion 2.4.7-10

1.6.4 ระบบเครือข่ายข้อมูล (Network)

1.6.4.1 Router: Cisco 2612, 2 Ethernet interfaces, 1 Tokenring interface 32MB RAM, 16MB Flash IOS 12.1(1)T IP Plus

1.6.4.2 Switch: Cisco Catalyst 1924EN, 24 x 10BaseTX, 2 x 100BaseTX

1.6.5 เครื่องมือที่ใช้ในการพัฒนา

- GNU C (GCC) for Linux version 2.96
- libpcap version 0.4

1.6.6 โปรแกรมสำเร็จรูปที่นำมาใช้ประกอบการพัฒนาและการทำงาน

- NAI LAN Sniffer Pro 4.5.04
- Netscape Navigator version 4.7 for Windows and Linux
- Microsoft Internet Explorer version 5.5/6.0 for Windows

1.7 รายละเอียดของแต่ละบท

สำหรับเนื้อหาของเอกสารประกอบการพัฒนาระบบถัดไปจากบทนี้ จะมีการแบ่งเนื้อหาหลักออกเป็น 5 หัวข้อ โดยเริ่มจาก

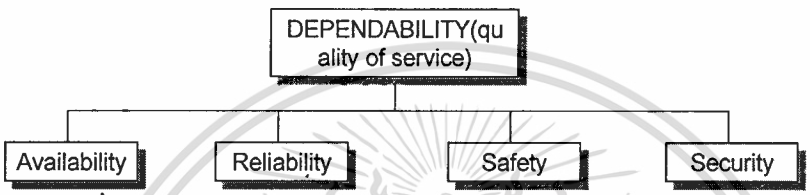
บทที่ 2 จะกล่าวถึงเราเตอร์ในด้านของการนำมาใช้ในการควบคุมเครือข่าย และทฤษฎีที่เกี่ยวข้อง หลักการของความปลอดภัย ความหมายของเราเตอร์ การทำงานของเราเตอร์ในการกลั่นกรองแพคเกตว่าเกี่ยวข้องกับนโยบายความปลอดภัยในระบบเครือข่ายอย่างไร รวมทั้งวิธีการกำหนดและทดสอบการกลั่นกรองแพคเกตบนเราเตอร์ รวมทั้งวิธีการบริหารและจัดการ ACL

บทที่ 3 กล่าวถึงการวิเคราะห์ออกแบบจากปัญหาและความต้องการ จนได้ส่วนประกอบของระบบที่จะพัฒนา การออกแบบโครงสร้างและรูปแบบข้อมูลสารสนเทศที่ไหลเวียนในระบบ รูปแบบและโปรโตคอลในการติดต่อระหว่างส่วนต่างๆ ของระบบ การนำเสนอและการติดต่อกับผู้ใช้ (user interface)

ในส่วนของบทที่ 4 จะเป็นขั้นตอนการพัฒนาระบบงานตามที่ได้ออกแบบไว้ในบทที่ 3 การทดสอบ และวิธีการทำงานของโปรแกรม ส่วนหัวข้อสุดท้าย บทที่ 5 จะเป็นการสรุปผลการพัฒนาระบบงาน ประโยชน์ที่ได้รับ อุปสรรคในการพัฒนา แนวทางการนำไปใช้หรือพัฒนาต่อเนื่อง และข้อเสนอแนะเพื่อเป็นประโยชน์ในการทำงานต่อไป

บทที่ 2
ความปลอดภัยของเครือข่าย

2.1 เกริ่นนำ



ภาพที่ 2.1 ความสัมพันธ์ระหว่างความไว้วางใจและความปลอดภัย

ความปลอดภัย (Security) นั้นมีความเกี่ยวข้องอย่างใกล้ชิดกับความไว้วางใจได้ (Dependability) ของระบบ ซึ่งสามารถดูได้จากคุณภาพของบริการที่ระบบนั้นให้ โดยประกอบไปด้วยปัจจัย 4 ประการตามรูปที่ 2.1 คือ

2.1.1 *ความพร้อม (Availability)* ความพร้อมของระบบนั้นนิยามโดยฟังก์ชัน A(t) ที่แสดงถึงความสามารถของระบบที่จะทำงานได้ ณ เวลา t กล่าวก็คือเป็นเปอร์เซนต์ที่ระบบสามารถปฏิบัติงานได้ตามหน้าที่ การสูญเสียความพร้อมนั้นมักจะเรียกว่าเกิดความบกพร่องในการให้บริการ (denial of service)

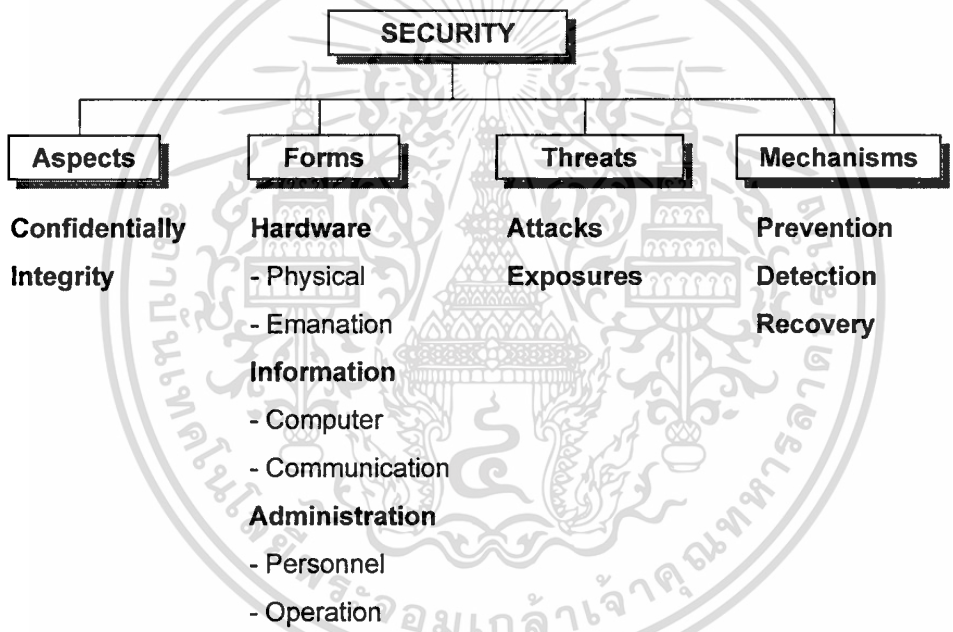
2.1.2 *ความเชื่อถือ (Reliability)* R(t) มีความเกี่ยวข้องอย่างมากกับความพร้อม แต่ความเชื่อถือนั้นเป็นความสามารถที่ระบบจะปฏิบัติหน้าที่ในช่วงเวลาหนึ่ง $[t_0, t]$ เมื่อระบบทำงานที่เวลา t_0 ความเชื่อถือจะเป็นการวัดความต่อเนื่องของบริการตลอดช่วงเวลานั้น

2.1.3 *ความปลอดภัย (Safety)* S(t) เป็นความสามารถของระบบที่จะปฏิบัติงานที่ควรจะเป็นอย่างถูกต้อง หรือเกิดความผิดพลาดเกิดขึ้นโดยที่ไม่มีความหายนะตามมา (fail-safe operation) ความปลอดภัยนี้มีความสำคัญอย่างยิ่งต่อระบบที่มีปฏิสัมพันธ์กับระบบอื่นที่มีโอกาสผิดพลาด หรือในแอพพลิเคชันที่ถ้าไม่ได้ควบคุมความผิดพลาดแล้ว หายนะที่เกิดขึ้นทำความเสียหายอย่างใหญ่หลวง

2.1.4 *ความปลอดภัย (Security)* ฟังก์ชัน Sec(t) แสดงถึงความของระบบที่จะปกป้อง objects โดยคำนึงถึงความบูรณภาพและความมั่นคงตลอดช่วงเวลาที่กำหนด [t_0, t] ซึ่ง objects นี้จะหมายถึงทรัพยากรต่าง ๆ ในระบบ

2.2 นิยามของความปลอดภัย

ความหมายของความปลอดภัยนั้นสามารถนิยามได้หลายลักษณะขึ้นอยู่กับปัจจัยที่ใช้ในการพิจารณา ในรายงานนี้จะพิจารณานิยามของความปลอดภัยจากใน 4 แง่มุม คือ จากมุมมองรูปแบบ การคุกคาม และกลไกความปลอดภัย ดังที่แสดงไว้ในรูปที่ 2.2



ภาพที่ 2.2 ความสัมพันธ์ของความปลอดภัย

2.2.1 **มุมมองของความปลอดภัย** สามารถพิจารณาได้สองแง่มุมคือ การเป็นความลับ (confidentiality) และความบูรณภาพ (integrity) การเป็นความลับนั้นจะดูว่าทำอย่างไรจึงจะปกป้อง objects ไม่ว่าจะเป็นการใช้ทรัพยากรระบบหรือนำข้อมูลสารสนเทศไปใช้โดยไม่ได้รับอนุญาต ส่วนความบูรณภาพนั้นจะเป็นเรื่องของการรักษาความสมบูรณ์ของ objects ไม่ให้ถูกแก้ไขเปลี่ยนแปลงโดยไม่ได้รับอนุญาต

2.2.2 **รูปแบบ** รูปแบบความปลอดภัยนั้นสามารถจำแนกได้เป็น 3 กลุ่มตามลักษณะ หรือคุณสมบัติทางความปลอดภัยที่คล้ายกันในแต่ละกลุ่มคือ

2.2.2.1 *ความปลอดภัยทางฮาร์ดแวร์* เกี่ยวข้องกับการป้องกัน objects จากความไม่มั่นคงเปราะบาง อันเนื่องมาจากการจัดการควบคุมฮาร์ดแวร์นั้น ซึ่งแบ่งเป็นสองลักษณะคือ ทางกายภาพที่เป็นภัยทางกายภาพภายนอก และทางการแพร่กระจายที่เกิดจากการแผ่คลื่นสัญญาณจากฮาร์ดแวร์ในระบบ

2.2.2.2 *ความปลอดภัยทางสารสนเทศ* เป็นการป้องกัน objects จากความไม่มั่นคงที่มีอยู่ในสถาปัตยกรรมของระบบ แบ่งเป็นสองแบบคือ ความปลอดภัยทางคอมพิวเตอร์ซึ่งเน้นในด้านสถาปัตยกรรมระบบ และความปลอดภัยในการสื่อสารซึ่งเกี่ยวข้องกับการป้องการสารสนเทศและ objects ระหว่างที่มีการสื่อสารเกิดขึ้น

2.2.2.3 *ความปลอดภัยทางการบริหาร* เป็นการป้องการ objects จากความไม่มั่นคงที่เกิดจากผู้ใช้ และภัยคุกคามต่อการบริหารงานขององค์กร แบ่งเป็น 2 แบบ คือ ความปลอดภัยทางบุคลากร และความปลอดภัยในการปฏิบัติงาน

2.2.3 **การคุกคาม** การคุกคามคือโอกาสที่ความปลอดภัยนั้นถูกละเมิดอันเนื่องมาจากความบกพร่อง หรือความอ่อนแอในระบบ ซึ่งแบ่งออกได้ 2 ประเภทคือการคุกคามที่เกิดโดยอุบัติเหตุ และการคุกคามที่เกิดโดยเจตนา การคุกคามที่เป็นอุบัติเหตุนั้นคือการที่ objects นั้นมีจุดอ่อนที่ทำให้ถูกแก้ไขเปลี่ยนแปลงจนเสียความบูรณภาพในตนเองไป ส่วนการคุกคามโดยเจตนานั้นหมายถึงการโจมตีจาก entity ใด ๆ โดยตั้งใจที่จะละเมิดความปลอดภัย ได้แก่การแก้ไข ขัดจังหวะ ทำลาย ปลอมแปลง หรือดักจับเป็นต้น ซึ่งเป็นได้ทั้งการโจมตีโดยทางตรง และทางอ้อม

2.2.4 **มาตรการความปลอดภัย** มาตรการในการรักษาความปลอดภัยที่ใช้ปฏิบัติกับ objects ซึ่งหมายถึงทรัพย์สินทั่วไปที่รวมถึงระบบคอมพิวเตอร์นั้นมีอยู่สามประการด้วยกันคือ

2.2.4.1 *การป้องกัน (Prevention)* เป็นมาตรการในการป้องกันobjects จากการถูกทำให้เสียหาย ถูกทำลาย หรือเปลี่ยนแปลงสภาพ

2.2.4.2 *การตรวจสอบ (Detection)* มาตรการที่ทำให้สามารถตรวจสอบได้ว่าเมื่อใดที่ objects มีความเสียหายเกิดขึ้น เกิดได้อย่างไร และใครเป็นผู้กระทำความเสียหายนั้น

2.2.4.3 *การแก้ไข (Reaction/Recover)* เป็นมาตรการในการฟื้นฟู objects ที่ได้รับความเสียหาย หรือแก้ไขความเสียหายที่เกิดขึ้นต่อ objects นั้น

2.3 นโยบายความปลอดภัย

นโยบายความปลอดภัยคือกลุ่มของกฎ (set of rules) ที่ใช้แสดงว่าการกระทำอะไรบ้างที่อนุญาตและอะไรบ้างที่ห้ามไม่ให้กระทำต่อ objects ภายในระบบ ณ ช่วงเวลาปฏิบัติงานปกติ ความรับผิดชอบในการกำหนดนโยบายความปลอดภัยที่เหมาะสมนั้นเป็นหน้าที่ของการตัดสินใจทางนโยบายของส่วนผู้บริหารในองค์กร

นโยบายความปลอดภัยที่ดีนั้นจะต้องอธิบายถึงการป้องกันระบบอย่างมีความสมดุลย์ระหว่างแต่ละ object ในระบบ รวมถึงคุ้มค่ากับค่าใช้จ่ายที่ต้องใช้ไปอีกด้วย ขั้นตอนแรกที่ต้องทำก่อนกำหนดนโยบายความปลอดภัยก็คือการกำหนดขอบเขต และวิเคราะห์ความเสี่ยงหรือการคุกคามต่อระบบภายใต้ขอบเขตนั้น เพื่อใช้ในการตัดสินใจว่าจะวางมาตรการความปลอดภัยอย่างไรให้เหมาะสมสำหรับระบบ

ภาพที่ 2.3 ขั้นตอนการกำหนดนโยบายความปลอดภัย

2.4 นโยบายความปลอดภัยบนอุปกรณ์ควบคุมเครือข่าย

นโยบายความปลอดภัยบนอุปกรณ์ควบคุมเครือข่ายในระบบทดสอบนี้จะครอบคลุมถึงการกลั่นกรองจราจรในเครือข่าย (traffic filtering) ซึ่งเป็นความสามารถหนึ่งบนเราเตอร์เป็นหลัก จากบทความสัมมนาเรื่องการวิเคราะห์เปรียบเทียบนโยบายความปลอดภัยระหว่างแบบที่เป็นทั้ง policy-based และ rule-based ในการพัฒนาระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่ายนี้จะใช้นโยบายแบบกฎในการทำงานเพื่อความสอดคล้องกับนโยบายที่กำหนดบนตัวเราเตอร์ที่เรียกว่า Access Control List (ACL) ซึ่งจะมีการกล่าวถึงโดยละเอียดต่อไป

นอกเหนือจาก ACL แล้ว ยังได้มีการศึกษาภาษาที่ใช้ในการกำหนดนโยบายความปลอดภัยแบบอื่นด้วย อย่างเช่นรูปแบบของกฎที่ใช้ใน SPSL (Security Policy Specification Language) ของ IETF สำหรับการกำหนดนโยบายของระบบเพื่อการทดสอบที่เป็น ACL นั้นจะถูกแปลงให้อยู่ในรูปของกฎที่ใช้เฉพาะภายในระบบที่เป็น SNORT อีกทีหนึ่งโดยตัวแปลงภาษา

2.5 Access Control List

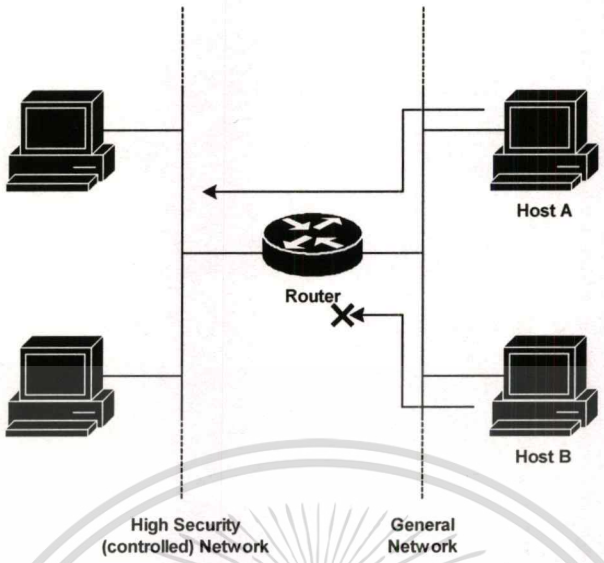
Access Control List หรือที่เรียกอย่างสั้นว่า ACL นั้น เป็นบริการหนึ่งของซอฟต์แวร์ที่ควบคุมและดูแลการทำงานของเราเตอร์ด้านการกรองจราจรในเครือข่ายข้อมูล ในบทความสัมมนานี้จะใช้

และอ้างอิงถึง ACL ที่มีการใช้งานใน IOS (Internetworking Operating System) ซึ่งเป็นระบบปฏิบัติการซอฟต์แวร์ควบคุมดูแลการทำงานของเราเตอร์จากทางบริษัท Cisco ที่มีการใช้งานกันทั่วไปอย่างกว้างขวางแพร่หลาย ส่วนกฎที่ใช้ในไฟร์วอลนั้นแม้จะต่างกันแต่ก็มีหลักการในการทำงานแบบเดียวกันกับ ACL

ACL กลั่นกรองการจราจรที่เข้าหรือออกเครือข่ายโดยการตรวจสอบแพคเกตข้อมูลที่ขาเชื่อมต่อหรืออินเตอร์เฟส (interface) ระหว่างเราเตอร์กับเครือข่ายนั้นว่าจะกั้น (block) ด้วยการปล่อยแพคเกตเหล่านั้นทิ้งไป หรือปล่อยผ่านให้ไป (forward) ตามเงื่อนไขที่กำหนดไว้ใน ACL ที่กำหนดโดยผู้ควบคุมระบบตามนโยบายความปลอดภัย ซึ่งสามารถระบุจากหมายเลขหรือแอดเดรส (address) ของต้นทาง ปลายทาง และโปรโตคอลที่อยู่ระดับเหนือขึ้นไป รวมทั้งข้อมูลอื่นๆ ซึ่งบางโปรโตคอลนั้นจะใช้คำว่า filter แทนคำว่า Access-list

2.5.1 การกรองแพคเกตโดยใช้เราเตอร์

สาเหตุที่ต้องมีการนำ ACL มาใช้นั้นก็เกิดจากความจำเป็นหลายประการ ไม่ว่าจะเป็นการควบคุมการเปลี่ยนแปลงข้อมูลเส้นทาง (route update) ระหว่างเราเตอร์หรือควบคุมการไหลของจราจร แต่ที่สำคัญที่สุดนั่นก็คือการนำมาใช้เพื่อรักษาความปลอดภัยในเครือข่าย วิธีการเบื้องต้นก็คือการใช้ ACL กลั่นกรองแพคเกตที่ผ่านเข้าออกแต่ละอินเตอร์เฟส โดยปกติถ้าไม่มีการกำหนด ACL แล้ว แพคเกตที่ผ่านเข้ามายังเราเตอร์นั้นได้รับการอนุญาตให้ไปยังส่วนใดๆ ของเครือข่ายก็ได้ ตัวอย่างดังรูปที่ 2.4 ACL ในเราเตอร์ปล่อยให้เครื่อง A สามารถเข้าถึงเครือข่ายได้อย่างอิสระ ในขณะที่ไม่อนุญาตให้เครื่อง B เข้าถึงทรัพยากรในเครือข่ายที่ควบคุมไว้ นอกจากนี้ยังสามารถระบุตามประเภทของโปรโตคอลว่าจะให้ผ่านหรือกันไว้ได้อีกด้วยเช่นอนุญาตให้ email ผ่านได้แต่กัน Telnet ไว้เป็นต้น ซึ่งเป็นการควบคุมโดยการตรวจสอบจากหมายเลขพอร์ตซึ่งต่างกัน



ภาพที่ 2.4 การใช้ Traffic Filter เพื่อการควบคุมการจราจรที่ผ่านเข้าไปในเครือข่าย

2.5.2 หลักการของการกำหนด ACL

เราเตอร์จะทำการตรวจสอบแพคเกตกับ ACL ตามลำดับ และทิศทางที่กำหนดไว้สำหรับอินเตอร์เฟส บางโปรโตคอลอย่างเช่น IP นั้นสามารถระบุได้สอง ACL สำหรับแพคเกตที่ผ่านเข้ามา (inbound) และออกไปจากอินเตอร์เฟสนั้น (outbound) โดยแต่ละ ACL นั้นจะต้องมีหมายเลขกำกับไว้ในการกำหนดให้กับอินเตอร์เฟสที่ต้องการ ซึ่งหมายเลขนี้จะใช้บอกประเภทและโปรโตคอลด้วย สำหรับ IP นั้นจะมีอยู่สองช่วงคือ 1-99 กับ 1300-1999 และ 100-199 กับ 2000-2999 สำหรับแบบ Extended ซึ่งเป็น ACL ของ IP แบบที่มีการระบุถึงพอร์ตของโปรโตคอลด้วย กฎที่ประกอบกันเป็น ACL นั้นจะทำงานตามลำดับ การเพิ่มกฎใหม่เข้าไปจะต่อท้ายเสมอ ดังนั้นถ้าต้องการแทรกจะต้องทำการลบ ACL ออกทั้งกลุ่มแล้วจึงป้อนเข้าไปใหม่ทั้งหมด ถ้าไม่ระบุไว้ กฎสุดท้ายใน ACL นั้นคือไม่ให้แพคเกตผ่านไป (denied all traffic)

ตารางที่ 2.1 ช่วงของเลขที่ใช้ในการกำหนด ACL ใน Cisco IOS

โปรโตคอล	ช่วงหมายเลขสำหรับ ACL
IP	1 to 99 and 1300 to 1999
Extended IP	100 to 199 and 2000 to 2699
Ethernet type code	200 to 299
Ethernet address	700 to 799
Transparent bridging (protocol type)	200 to 299
Transparent bridging (vendor code)	700 to 799
Extended transparent bridging	1100 to 1199
DECnet and extended DECnet	300 to 399
XNS	400 to 499
Extended XNS	500 to 599
AppleTalk	600 to 699
Source-route bridging (protocol type)	200 to 299
Source-route bridging (vendor code)	700 to 799
IPX	800 to 899
Extended IPX	900 to 999
IPX SAP	1000 to 1099
Standard VINES	1 to 100
Extended VINES	101 to 200
Simple VINES	201 to 300

2.5.3 รูปแบบของการกำหนด ACL

```
access-list access-list-number {deny | permit} protocol source source-
wildcard destination destination-wildcard [precedence precedence] [tos
tos] [established] [log]
```

แบบนี้เป็น ACL แบบ Extended ตามเงื่อนไข โดยใช้คำสั่ง log เพื่อให้มีการบันทึกเหตุการณ์ที่เกี่ยวข้องกับ ACLนั้นไว้ด้วย

```
access-list access-list-number {deny | permit} protocol any any
```

แบบนี้ใช้ any เพื่อระบุต้นทางและปลายทางเป็นใดใดก็ได้ ใช้แทนความหมาย wildcard ที่เป็น 0.0.0.0 255.255.255.255

```
access-list access-list-number {deny | permit} protocol host source
host destination
```

แบบนี้ระบุถึงต้นทางและปลายทางที่เป็นเครื่องโดดๆ มีความหมายเหมือนกัน wildcard ที่เป็น 0.0.0.0

2.5.4 ขั้นตอนการกำหนด ACL

- ออกแบบและเขียน ACL ตามนโยบายความปลอดภัยของระบบที่อยู่ในการพิจารณา
- นำ ACL สู่เราเตอร์ สามารถทำได้หลายวิธีคือ การป้อนผ่านบรรทัดคำสั่ง การใช้ TFTP
- นำ ACL ที่กำหนดขึ้นนั้นไปใช้กับอินเตอร์เฟสของเราเตอร์ที่ต้องการ

2.6 ภาษาที่ใช้ในการกำหนดนโยบายความปลอดภัย

การกำหนดนโยบายความปลอดภัยในระดับสูงนั้นสามารถกำหนดตาม SPSL (Security Policy Specification Language) ซึ่งถูกออกแบบเพื่อใช้สนับสนุน PCIM (Policy Core Information Model) ซึ่งเป็นส่วนขยายของ CIM (Common Information Model) โดยมีคณะทำงานทางด้านนโยบาย และ IPSec ของ IETF (Internet Engineering Task Force) เป็นผู้ร่างขึ้นมา ผู้ออกแบบหลักนั้นมาจากบริษัท BBN โดยมีเป้าหมายในการพัฒนาเพื่อใช้ใน PBSM (Policy Based System Management) ในรายละเอียดนั้นค่อนข้างจะมีการกำหนด class เป็นประเภทต่างๆ ที่ครอบคลุมในเรื่องของระบบเครือข่ายและ attribute ของ class ไว้อย่างครบถ้วน ใช้ data-type มาตรฐานแบบเดียวกับ CIM ส่วนโมเดลที่ใช้เป็น Object Oriented/UML

วากยสัมพันธ์ของ SPSL นั้นได้รับแบบอย่างมาจาก RPSL (Routing Policy Specification Langauge) แต่กฎการประมวลผลของ SPSL แตกต่างออกไปเนื่องจากได้รับการออกแบบเพื่อการกำหนดนโยบายทาง IP Sec และ ISAKMP (Internet Security Association and Key Management Protocol) ซึ่งใช้ในอุปกรณ์เครือข่ายที่เกี่ยวข้องกับการกรองแพคเกต (packet filtering)

2.6.1 ข้อกำหนดของภาษา

ใน SPSL นั้น นโยบายถูกนิยามด้วยการผูกพันกัน (bind) ระหว่างเซตของเงื่อนไขในการการติดต่อสื่อสารและเซตของการกระทำด้านความปลอดภัยที่ตอบสนองต่อเงื่อนไขนั้น เซตเงื่อนไขการติดต่อสื่อสารนี้ถูกกำหนดโดย tuple ของค่า selector ซึ่งเป็นค่าที่กำหนดตามหลักการของ IPSec ถ้าการติดต่อสื่อสารที่เกิดขึ้นนั้นตรงกับหนึ่งในเงื่อนไขที่กำหนดไว้ ตัวบังคับนโยบายก็จะกระทำตาม action อย่างใดอย่างหนึ่งเช่น

- อาจเป็นการกระทำที่เกี่ยวข้องกับการกรองแพคเกต เช่นการระงับ การส่งผ่าน หรือการส่งต่อ (tunneling) ไปยัง network entity อื่นต่อไป
- อาจจะเป็นการกำหนดข้อแนะนำด้านความปลอดภัย สำหรับปกป้องการแลกเปลี่ยน ISAKMP
- อาจจะเป็นการกำหนด IPSec tunnel หรือการช่องทางส่งสำหรับส่งต่อแพคเกต

2.6.2 โครงสร้างของภาษา

SPSL ใช้แนวคิดแบบ object โดยการนิยามเซตของ class ซึ่งสามารถนำไปใช้สร้าง instance เป็น object เพื่อใช้ในการกำหนดนโยบาย แต่อย่างไรก็ตาม SPSL นั้นเอื้ออำนวยให้มีการสร้าง class ใหม่ได้ตามไวยากรณ์ของภาษา object-oriented โดยแต่ละ object ใน SPSL นั้นจะมีค่าเฉพาะที่ใช้แยกแยะและอ้างอิงถึงจาก attribute แรกที่เรียกว่า key attribute ในภาษา SPSL มีการแบ่งclass ออกเป็น 4 หมวดคือ

2.6.2.1 Primitive Data ประกอบไปด้วยข้อมูลพื้นฐานที่ใช้ในการกำหนดนโยบายเช่น object-name, ipv4-address หรือ date เป็นต้น

2.6.2.2 *Management Agents* ประกอบไปด้วยสารสนเทศที่เกี่ยวเนื่องกับ management entity ซึ่งมี สอง class คือ maintainer (mntner) และ certificate (cert)

2.6.2.3 *Network Entities* เป็นส่วนประกอบทางเครือข่ายที่เกี่ยวข้องกับนโยบาย ได้แก่ class ต่าง ๆ เช่น node, node-set, gateway, gateway-set, polserver และ domain

2.6.2.4 *Policies* ประกอบไปด้วยนโยบายที่กำหนดไว้ ซึ่งมีสอง class ในตอนนี้คือ class ที่กำหนดนโยบายสำหรับกฎในการกรองแพคเกตและ class สำหรับนโยบายของ IPSec ที่กำหนด selector และ action

รูปแบบของภาษาในส่วนที่ใช้กำหนดนโยบายใน policy class ที่นำมาใช้ในการกำหนดนโยบายความปลอดภัยสำหรับเราเตอร์เป็นดังนี้

```
policy: dst * | any | list of [not]
    | list of [not]
  [port * | opaque | any | list of [not] |
    list of [not] [dynamic []]]
  [src * | any | list of [not] | list of [not]
  [port * | opaque | any | list of [not]
    | list of [not] [dynamic []]]]
  [xport-proto * | opaque | any | list of [not]
    | list of [not] ]
  [direction inbound | outbound [, symmetric]]
  permit [, forward ] | deny [, forward ]
    | forward
```

ทั้งในการกำหนดต้นทางและปลายทางนั้นใช้การระบุหมายเลข IP โดยแต่ละหมายเลขเป็นช่วงจากค่าต่ำสุดไปยังสูงสุด (inclusive) หรืออาจจะระบุว่าเป็นหมายเลขใดใดก็ได้โดยการใช้ any หรือ * แทน ในกรณีที่ใช้ not เพื่อบอกไว้ต้องไม่ใช่หมายเลขที่ระบุ (exclusive) ส่วน port นั้นจะกำหนดหรือไม่ก็ได้ ถ้าไม่ระบุก็จะหมายถึงทุกพอร์ต ส่วน xport-proto นั้นจะหมายถึงประเภทของโปรโตคอลที่ใช้ในการสื่อสาร (transportation protocol) ตามที่กำหนดไว้ในโปรโตคอล IP อย่างเช่น ถ้าเป็น TCP ก็ใช้หมายเลข 6 เป็นต้น ส่วนสุดท้ายจะเป็นการกำหนดทิศทางของการจราจร และการกระทำต่อแพคเกตนั้นตามตามเงื่อนไขว่าจะส่งต่อหรือไม่

2.6.3 ตัวอย่างของการนิยาม object โดยการใช้ SPSL

การนิยาม node object

```
node:     SG-FOO-FIREWALL:COTTON
name:     cotton.foo.com
ifaddr:   172.16.5.196
```

การนิยาม policy object แบบย่อ

```
policy-name: foo
association: sg-bar
policy: dst 172.16.0.0-172.16.255.255
        src 192.168.100.0-192.168.100.255
        xport-proto 6 permit
policy: dst 172.16.0.0-172.16.255.255 deny
```

2.7 ภาษาที่ใช้ในการสร้างการจราจร

รูปแบบของแบบการจราจรที่ใช้ในระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่ายนี้จะใช้รูปแบบที่ใช้ในโปรแกรม IDS ชื่อ SNORT เป็นระบบที่ใช้ตรวจสอบการรุกราน (Intrusion Detection System) ขนาดย่อมซึ่งมีความสามารถในการวิเคราะห์การจราจรในเครือข่ายแบบ real-time ภาษาในการอธิบายนโยบายที่ SNORT ใช้นั้นอยู่รูปแบบของกฎ (rule-based) ที่มีความยืดหยุ่นสูงและมีกฎที่กำหนดไว้แล้วสามารถนำมาใช้งานได้ทันที

2.7.1 รูปแบบของกฎตามแบบ SNORT

รูปแบบข้อกำหนดสำคัญของกฎแต่ละข้อที่ใช้ใน SNORT นั้นคือต้องจบภายในบรรทัดเดียว ประกอบไปด้วยรูปแบบดังนี้

```
[Rule Action ] [Protocol] [IP address/Mask] [Port] [Direction
Operator] [IP address/Mask] [Port] ([Rule options])
```

ส่วนที่อยู่ด้านหน้าวงเล็บที่เป็นส่วนที่เรียกว่า Rule header ซึ่งประกอบไปด้วย Rule Action, Protocol และหมายเลข IP กับพอร์ตของต้นทางและปลายทาง

- *Rule Action* ประกอบไปด้วยการกระทำสามแบบคือ alert ซึ่งเป็นการเตือนตามรูปแบบที่ได้กำหนดไว้พร้อมกับบันทึกแพคเกตนั้นไว้ log เพื่อบันทึกแพคเกตนั้น และ pass คือทิ้งแพคเกตนั้นไป
- *Protocol* สนับสนุนโปรโตคอล tcp, udp และ icmp
- *IP Address และ Port* ส่วนนี้สามารถใช้คำว่า any เพื่อกำหนดว่าเป็น IP address อะไรก็ได้ สำหรับ IP address นั้นจะต้องมีการระบุ netmask ในรูปจำนวนบิตด้วย เครื่องหมาย ! นั้นใช้บอกว่าเป็น IP address ที่ยกเว้น ส่วน port นั้นสามารถใช้เครื่องหมาย : บอกช่วงได้
- *Direction Operator* ใช้เครื่องหมาย -> บอกทิศทางของแพคเกต โดยด้านซ้ายแสดงถึงต้นทางและด้านขวาเป็น ปลายทาง ส่วนเครื่องหมาย <> ใช้บอกว่าเป็นการพิจารณาแบบสองทิศทาง
- *Rule options* แต่ละ rule option ในวงเล็บจะต้องลงท้ายด้วยเครื่องหมาย ; และใช้เครื่องหมาย : คั่นระหว่าง Rule option keyword กับ argument ของมัน

2.7.2 ตัวอย่างกฎตามแบบของ SNORT

```
log udp any any -> 192.168.1.0/24 1:1024
```

บันทึก UDP traffic ที่มาจาก port ใด ๆ ไปยัง port 1 ถึง 1024 ในเครือข่าย 192.168.1.0

```
alert tcp any any -> 192.168.1.0/24 111(content:"|00 01 86 a5|"; msg:
  "mountd access";)
```

เตือนเมื่อมี TCP traffic จาก port ใด ๆ ไปยัง port 111 ในเครือข่าย 192.168.1.0 โดยประกอบไปด้วยข้อมูลตามรูปแบบที่กำหนด โดยให้บันทึกข้อความว่า mountd access ลงไปด้วย

2.8 สรุป

ในการออกแบบนโยบายทางความปลอดภัยที่ดีนั้น นอกจากผู้ออกแบบต้องมีความเข้าใจจุดอ่อนและความเสี่ยงต่อการสูญเสียที่เกิดขึ้นเป็นอย่างดีแล้ว ภาษาที่ใช้ในการกำหนดนโยบายความปลอดภัยก็นับว่ามีความสำคัญเป็นอย่างยิ่งในการสื่อเพื่อการทำความเข้าใจนโยบายความปลอดภัยระหว่างผู้ออกแบบนโยบาย ผู้ออกแบบระบบและผู้บริหารควบคุมระบบ ไม่ว่าจะเลือกใช้ภาษาในก็ตามในการกำหนดนโยบาย ขั้นตอนที่ต้องให้ความสำคัญเป็นอันดับแรกคือการกำหนดขอบเขตของระบบและเป้าหมายของนโยบายนั้นให้สอดคล้องกับนโยบายขององค์กร

ในบทนี้ได้มีการศึกษาหลักการของความปลอดภัยและภาษาในการกำหนดนโยบายแบบต่างๆ ซึ่งสามารถนำมาประยุกต์ใช้ร่วมกันเพื่อนำไปใช้พัฒนาระบบจำลองและทดสอบนโยบายความปลอดภัยในสภาพแวดล้อมการใช้งานจริงซึ่งจะมีการกล่าวโดยละเอียดเพิ่มเติมในบทต่อไปในส่วนของการวิเคราะห์ออกแบบ และการพัฒนาระบบ

บทที่ 3
การวิเคราะห์และออกแบบ

3.1 ปัญหาของระบบในปัจจุบัน

การทำงานของเราเตอร์ในการควบคุมเครือข่ายนั้นอาศัยการกลั่นกรองจราจรในเครือข่ายตามกฎในการควบคุมที่ได้กำหนดไว้ตามนโยบายความปลอดภัยของระบบ กฎที่ใช้ในการควบคุมการกรองแพคเกตโดยเราเตอร์นั้นคือ ACL ที่เป็นการกำหนดเงื่อนไขในการตรวจสอบแพคเกตทีละข้อในแต่ละบรรทัด การที่รูปแบบของการเขียน ACL นั้นจะมีความแตกต่างกันไปขึ้นอยู่กับซอฟต์แวร์ที่ใช้ในควบคุมเราเตอร์ ทำให้การทดสอบ ACL ที่กำหนดขึ้นตามนโยบายความปลอดภัยทางเครือข่ายนั้นนับว่าเป็นงานที่ต้องอาศัยเวลาและทรัพยากร และต้องอาศัยความเข้าใจเป็นอย่างมาก เนื่องจากต้องมีการเตรียมอุปกรณ์เครือข่ายและเครื่องคอมพิวเตอร์จำนวนหนึ่งที่กำหนดหมายเลข IP ให้ครอบคลุมเพียงพอที่จะใช้ในการตรวจสอบประเมินเงื่อนไขต่างๆ ใน ACL ที่จะทดสอบ อีกทั้งอุปกรณ์เครือข่ายสำหรับสร้างแพคเกตส่วนใหญ่ที่มีอยู่นั้นก็ไม่สามารถสร้างแพตเกตตามนโยบายความปลอดภัยที่กำหนดไว้ได้ นอกจากนี้การสร้างการจราจรซึ่งเป็นภาระงาน (workload) สำหรับการทดสอบในแต่ละครั้งสำหรับการวัดผลนั้นก็ไม่สามารถทำซ้ำได้เหมือนเดิม ภาระงานที่แตกต่างกันนี้อาจจะส่งผลให้ผลลัพธ์ที่ได้นั้นมีความแม่นยำน้อยลง

3.2 ความต้องการของระบบ

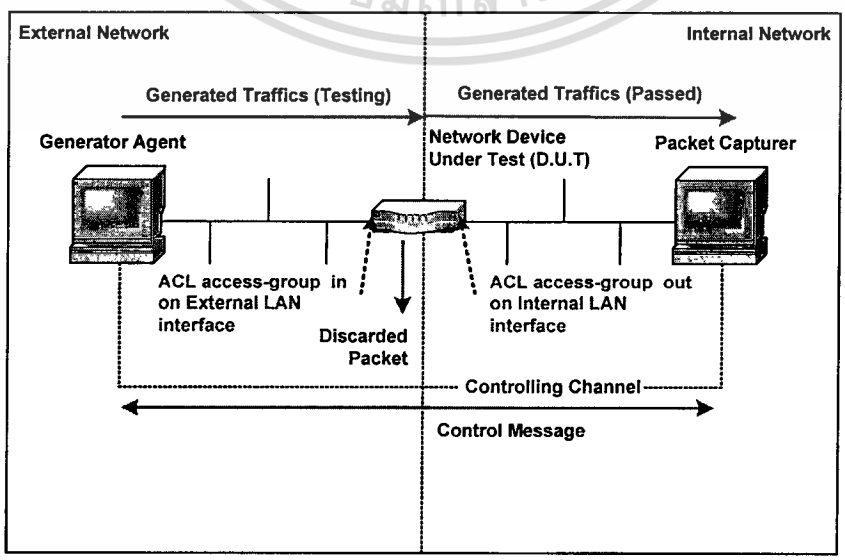
จุดมุ่งหมายของระบบทดสอบนโยบายอุปกรณ์ควบคุมเครือข่ายนี้จะเน้นหนักไปในการทดสอบนโยบายที่อยู่ในรูปของ ACL ที่ใช้ควบคุมการไหลผ่านเข้าออกของการจราจรในเครือข่ายที่ควบคุมโดยเราเตอร์ หน้าที่หลักของระบบทดสอบดังกล่าวนี้จะประกอบไปด้วยการสร้างรูปแบบของการจราจรจากนโยบายความปลอดภัยที่จะทดสอบ การจราจรจะถูกส่งจากระบบทดสอบส่วนแรกเข้าสู่เครือข่ายที่ได้รับการควบคุมดูแลโดยเราเตอร์ที่มีนโยบายความปลอดภัยนั้นทำงานอยู่ ระบบทดสอบจึงต้องมีส่วนประกอบอีกส่วนหนึ่งที่อยู่ภายในเครือข่ายที่ควบคุมนั้นเพื่อทำการดักจับแพตเกตข้อมูลที่ถูกส่งมาจากภายนอก แล้วทำการวิเคราะห์เปรียบเทียบกับนโยบายความปลอดภัยที่ทดสอบภายหลัง เพื่อให้ได้ประสิทธิภาพและสามารถสร้างการจราจรที่เหมือนจริงที่สุด ระบบทดสอบส่วนที่ทำหน้าที่ส่งข้อมูลนั้นจึงต้องมีความสามารถในการระบุหมายเลข IP ต้นทางตามที่กำหนดไว้ได้ด้วย (IP Spoofing) ในกรณีนี้จะพิจารณาการทดสอบอุปกรณ์ควบคุมเครือข่ายในแบบ

การจราจรที่เข้ามาในเครือข่ายควบคุม (inbound) เท่านั้น คือสามารถทดสอบ ACL ที่อยู่บนอินเตอร์เฟสที่เชื่อมต่อกับเครือข่ายภายนอกแบบ in หรือ ACL ที่อยู่บนอินเตอร์เฟสที่เชื่อมต่อกับเครือข่ายภายในแบบ out

3.3 ขอบข่ายและส่วนประกอบของระบบ

ดังที่กล่าวไว้แล้วในความต้องการของระบบทดสอบนั้นจะต้องมีการสร้างการจราจรซึ่งเป็นชุดของแพตเกตจากนโยบายความปลอดภัยที่อยู่ในรูปของ ACL ที่ทำงานอยู่บนเราเตอร์ที่ทำหน้าที่เป็นอุปกรณ์ควบคุมเครือข่ายภายใน โดยจะต้องมีส่วนที่ทำหน้าที่วิเคราะห์ผลลัพธ์ที่ได้ประกอบกับนโยบายความปลอดภัยด้วยว่ามีกฎใดนโยบายนั้นที่ปล่อยให้แพคเกตสามารถผ่านเข้าไปได้ ซึ่งอาจจะเกิดจากเงื่อนไขที่กำหนดหรือลำดับของ ACL นั้นไม่เหมาะสมก็เป็นไปได้

ในการทดสอบนั้นจะพิจารณาตามโดเมนของต้นทางและปลายทาง โดยจะสร้างการจราจรทั้งหมดที่เป็นความสัมพันธ์ทั้งหมดจากสมาชิกของโดเมนต้นทางไปยังปลายทาง และกำหนดรูปแบบของโปรโตคอลและพอร์ตกำกับไว้กับแพคเกตนั้นอีกทีหนึ่ง ในด้านของลำดับของแพคเกตนั้นถือว่าไม่มีนัยสำคัญต่อการทดสอบนี้เนื่องจาก ACL ไม่สามารถใช้ในการตรวจสอบชุดลำดับของแพคเกตได้ดังเช่นใน IDS (Intrusion Detection System) อย่างไรก็ตามระบบทดสอบนี้จะได้พิจารณาเรื่องลำดับของแพคเกตไว้ด้วยในการพัฒนา นั่นคือในส่วนของโมดูลที่ส่งแพคเกตเพื่อทำการทดสอบนั้นจะสามารถสร้างรูปแบบการจราจรตามแบบแผนของการจราจรที่กำหนดไว้ได้โดยผู้ทำการทดสอบหรือสร้างจากนโยบายความปลอดภัยในลักษณะสุ่ม และอาจจะนำเอากฎที่มีอยู่แล้วใน IDS อย่าง SNORT มาใช้อีกด้วย



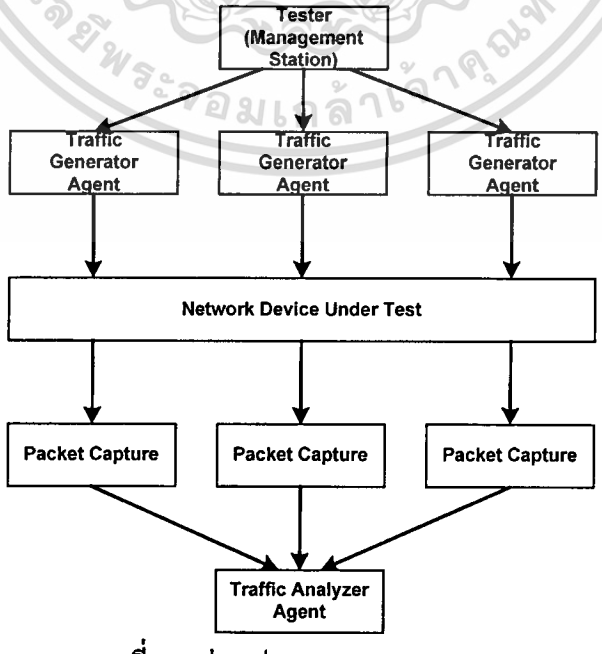
ภาพที่ 3.1. การทำงานของระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่าย

ในส่วนของการวิเคราะห์ผลลัพธ์ที่ได้นั้น จะพิจารณาจากแพคเกตที่เก็บไว้ในเครือข่ายควบคุมโดยตัว Packet Capturer นโยบายความปลอดภัย รูปแบบของการจราจรที่สร้างขึ้น และข้อความที่ส่งมาจากเราเตอร์ผ่านทางบริการ Syslog แต่แบบหลังนี้จะต้องอาศัยความร่วมมือจากเราเตอร์ด้วยเนื่องจากต้องมีการกำหนดใน ACL ว่าให้ทำการ log เหตุการณ์ที่เกิดจากกฎของ ACL แต่ละข้อไว้ ในการทำงานส่วนนี้ในขั้นตอนพัฒนาและทดสอบจะใช้ไฟล์ Syslog ที่เก็บจากโปรแกรม Syslog server ที่เป็นโปรแกรมสำเร็จรูป หรือที่เป็นบริการทางเครือข่ายที่มีอยู่แล้วในระบบ UNIX ก็ได้ โดยจะทำการศึกษารูปแบบของไฟล์ Syslog และรูปแบบของข้อความโดยทำการกรองเอาเฉพาะเหตุการณ์ที่เกี่ยวข้องกับ ACL เท่านั้น ข้อมูลเข้าสำหรับการวิเคราะห์โดยระบบย่อยส่วนนี้จึงทำงานเกี่ยวข้องกับไฟล์ข้อมูลเป็นส่วนใหญ่ในลักษณะแบบ text แบบ online

สิ่งสำคัญคือเราเตอร์ที่เป็นอุปกรณ์เครือข่ายสำหรับการทดสอบนั้นจะต้องมีอินเตอร์เฟสสำหรับ LAN (Local Area Network) อย่างน้อยสองอินเตอร์เฟสสำหรับเครือข่ายสองด้านคือเครือข่ายภายนอก และเครือข่ายภายใน

3.4 กระบวนการทำงานของระบบ

ในแง่ของสถาปัตยกรรมภายในระบบทดสอบความปลอดภัยนี้สามารถแบ่งส่วนประกอบหลักออกเป็นสามส่วนด้วยกันตามที่แสดงไว้ในรูปคือ Traffic Generator Agent, Packet Capture, Traffic Analyzer Agent และ Management Station ที่ใช้โดยผู้ทดสอบระบบ



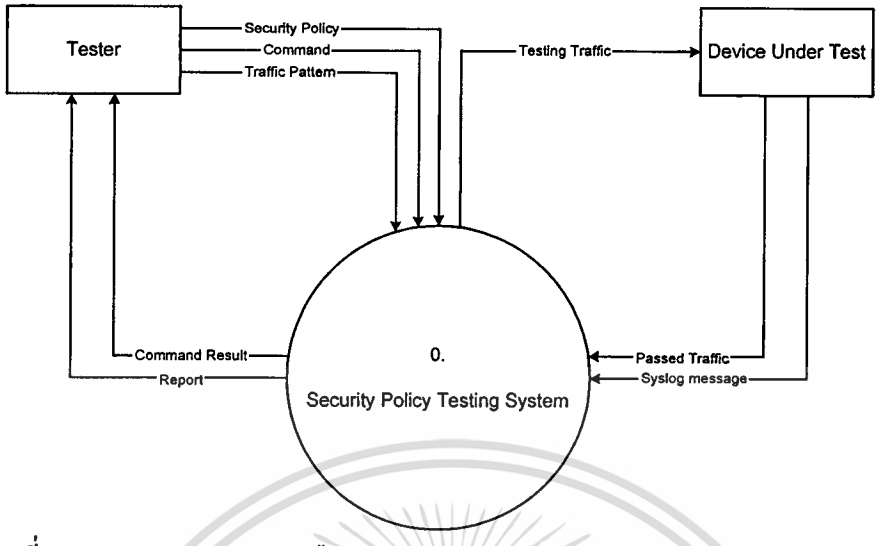
ภาพที่ 3.2 ส่วนประกอบของระบบ

3.4.1 **Management Station** เป็นส่วนที่ผู้ทดสอบทำการสั่งงานส่วนต่างๆ ในระบบทดสอบ กำหนดนโยบายความปลอดภัย สร้างรูปแบบการจราจรสำหรับทดสอบทั้งโดยตรงและจากนโยบายความปลอดภัย นอกจากนี้ยังเป็นส่วนที่ทำหน้าที่วิเคราะห์และสร้างรายงานผลการทดสอบอีกด้วย ส่วนนี้จะเป็นส่วนหลักที่ติดต่อกับผู้ใช้ (user interface) ทั้งในด้านการรับคำสั่งและแสดงผล เพื่อความคล่องตัวการติดต่อกับผู้ใช้นั้นทำได้ด้วยการป้อนคำสั่งผ่านบรรทัดคำสั่ง หรือผ่านระบบติดต่อกับผู้ใช้แบบกราฟฟิค (GUI - Graphics User Interface) ผ่านทาง web

3.4.2 **Traffic Generator Agent (Workload Generator)** มีหน้าที่ในการสร้างการจราจรเครือข่ายเพื่อเป็นภาระงานสำหรับการทดสอบอุปกรณ์ควบคุมเครือข่าย ในส่วนนี้ประกอบไปด้วยส่วนที่ทำหน้าที่ส่งแพคเกตเข้าไปในเครือข่ายตามรูปแบบการจราจรที่ได้จากการแปลงจากนโยบายความปลอดภัย หรือจากรูปแบบการจราจรที่ได้รับการกำหนดจากผู้ทดสอบ

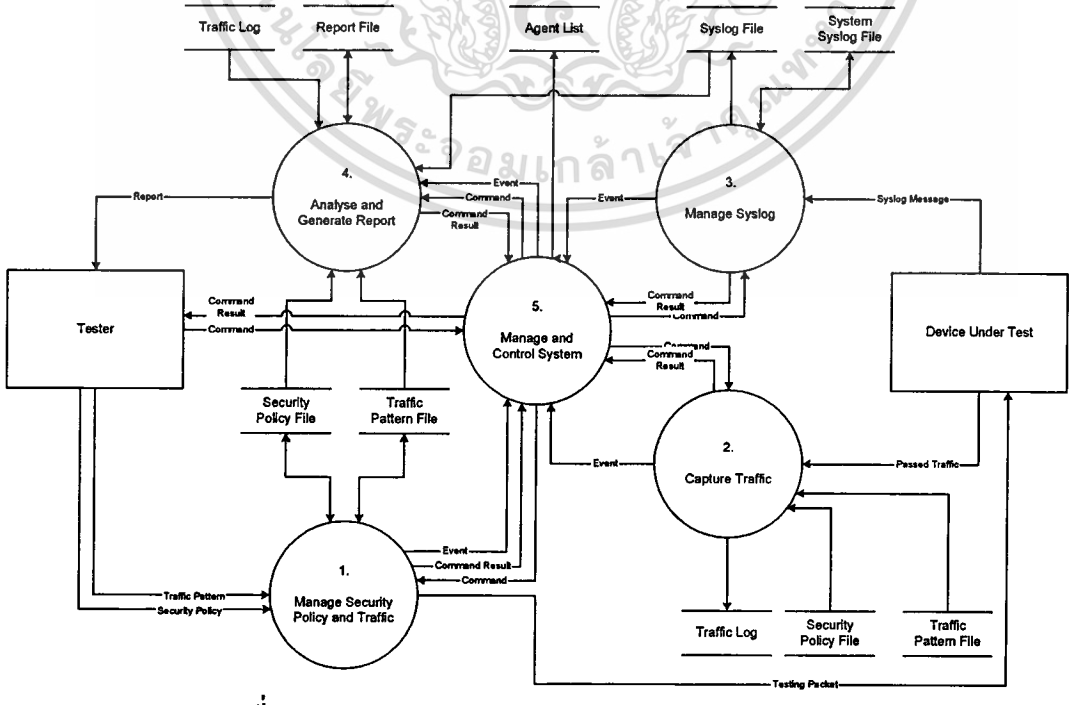
3.4.3 **Packet Capture และ Traffic Analyzer Agent (Observer)** เป็นส่วนที่ทำงานอยู่ในเครื่องคอมพิวเตอร์ที่ต่อเข้ากับเครือข่ายภายใน (internal network) ที่ได้รับการควบคุมการจราจรเข้าออกโดยเราเตอร์ ส่วนนี้จะคอยรับแพคเกตที่ส่งมาจาก Traffic Generator Agent อีกทีหนึ่งแล้วบันทึกแพคเกตที่ได้รับนั้นเพื่อทำการวิเคราะห์โดยส่วนที่ทำหน้าที่วิเคราะห์และรายงานผลต่อไป

จากที่ได้ทำการวิเคราะห์ระบบและออกแบบไว้ สามารถนำมาเขียนเป็นแผนภาพอธิบายระบบ (Context Diagram) หรือแผนภาพแสดงการไหลเวียนของข้อมูลในระดับ 0 (Data Flow Diagram - DFD) ได้ดังภาพที่ 3.3 แผนภาพนี้โดยเป็นแผนภาพที่อธิบายระบบโดยรวมว่าระบบมีการเชื่อมต่อกับสิ่งภายนอกอะไรบ้าง อย่างไร ระบบทดสอบนโยบายความปลอดภัยที่ได้ออกแบบขึ้นนี้มีความเกี่ยวข้องกับสิ่งภายนอก (external entity) ในอยู่สองตัว คือผู้ทดสอบและอุปกรณ์ควบคุมเครือข่าย (ซึ่งในกรณีนี้คือเราเตอร์ซึ่งเป็นอุปกรณ์ที่อยู่ภายใต้การทดสอบ โดยมีข้อมูลสารสนเทศต่างๆ ไหลเวียนระหว่างภายนอกและภายในดังที่ได้แสดงไว้ดังภาพที่ 3.3



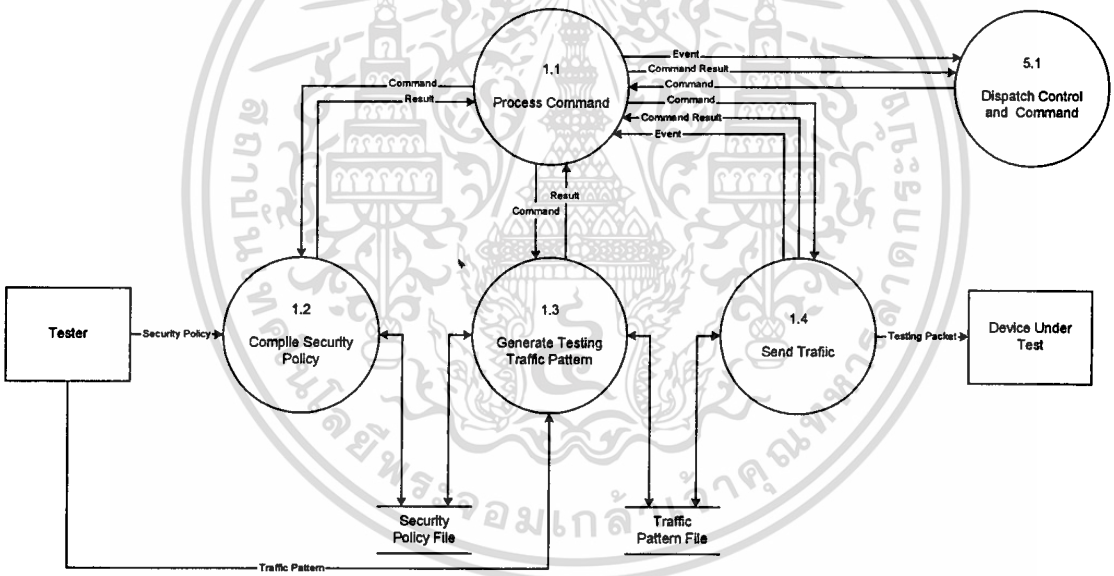
ภาพที่ 3.3 Context Diagram หรือ DFD level 0 ของ Security Policy Testing System

ในแผนภาพแสดงการไหลเวียนโดยมีกระบวนการย่อยอีก 5 กระบวนการภายในที่กระจายการทำงานแยกออกไปตามหน้าที่สามส่วนตามสถาปัตยกรรมของระบบคือ Management Station, Traffic Generator Agent และ Traffic Analyzer Agent ดังที่แสดงไว้ในสถาปัตยกรรมของระบบตามรูปที่ 3.2 ซึ่ง Management Station และ Traffic Generator Agent นั้นสามารถทำงานอยู่บนเครื่องเดียวกันได้ ดังนั้นในการใช้งานนี้จึงต้องมีเครื่องคอมพิวเตอร์อย่างน้อย 2 เครื่อง



ภาพที่ 3.4 DFD level 1 ของ Security Policy Testing System

จากแผนภาพการไหลของข้อมูลระดับที่ 1 ระบบนี้สามารถแบ่งการทำงานออกเป็น 5 ส่วน คือ กระบวนการที่ 1 เป็นการประมวลผลนโยบายความปลอดภัยและรูปแบบการจราจร รวมทั้งการสร้างการจราจรสำหรับเครือข่ายทีละแพคเกต กระบวนการที่ 2 เป็นการดักจับแพคเกตที่ผ่านเราเตอร์เข้ายังเครือข่ายที่ควบคุม ตรวจสอบกับรูปแบบการจราจร และเก็บผลที่ได้ไว้ กระบวนการที่ 3 จะเกี่ยวข้องกับการจัดเก็บ Syslog ที่ได้รับคอยบันทึกเหตุการณ์ที่ส่งมาจากเราเตอร์ไว้ ซึ่งเหตุการณ์เหล่านี้จะมีเหตุการณ์ที่เกิดจาก ACL ด้วย กระบวนการที่ 4 เป็นส่วนที่นำบันทึกต่างๆ มาวิเคราะห์ตรวจสอบกับนโยบายความปลอดภัย เพื่อนำเสนอรายงานผลลัพธ์แก่ผู้ทดสอบ และสุดท้ายกระบวนการที่ 5 นั้นจะทำหน้าที่รับคำสั่งควบคุมการทำงานของส่วนประกอบต่างๆ ในระบบจากผู้ทดสอบก่อนที่ส่งคำสั่งควบคุมที่เหมาะสมไปยังแต่ละส่วนประกอบต่อไป นอกจากนี้ยังทำหน้าที่ประสานการทำงานของส่วนประกอบทั้งหมดด้วย



ภาพที่ 3.5 DFD level 2 ของส่วน Manage Security Policy and Traffic

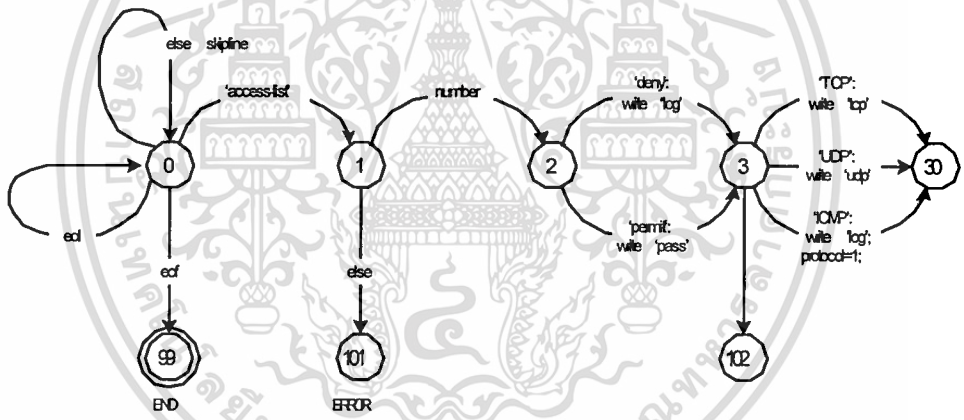
3.4.4 กระบวนการที่ 1 Manage Security Policy and Traffic

3.4.4.1 กระบวนการย่อย 1.1: Process Command

เป็นส่วนที่รับคำสั่งในการควบคุมการทำงานจากส่วนควบคุมกลาง (management station) แล้วกระจายการทำงานไปยังส่วนย่อยต่างๆ อีกทีหนึ่งโดยส่งผ่านตัวแปร (parameter) ต่างๆ ตามรูปแบบที่ได้กำหนดไว้แล้วทั้งผ่านระบบเครือข่ายหรือผ่านบรรทัดคำสั่ง

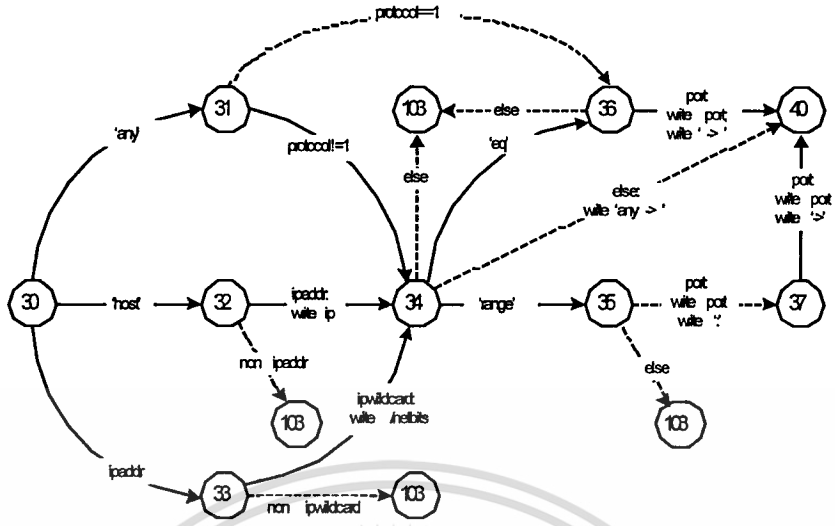
3.4.4.2 กระบวนการย่อย 1.2: Compile Security Policy

ระบบย่อยส่วนนี้จะเป็นส่วนโมดูลที่ทำหน้าที่บริหารและจัดการนโยบายและกฎความปลอดภัยที่ใช้ในการทดสอบโดยจะทำการแปลนโยบายความปลอดภัยที่กำหนดขึ้นโดยใช้รูปแบบของกฎตาม Cisco ACL เพื่อให้อยู่ในรูปแบบของกฎที่ใช้ภายในโปรแกรมที่นำรูปแบบจาก SNORT มาใช้ ซึ่งผลลัพธ์ที่ได้จากการแปลนี้จะถูกนำไปใช้ในการสร้างรูปแบบการจราจรสำหรับการทดสอบและนำไปใช้วิเคราะห์โดยระบบย่อยส่วนที่ 4 สำหรับการวิเคราะห์และรายงานผล ในส่วนของการแปลงกฎที่อยู่ในรูปแบบของ ACL ให้เป็น SNORT นี้สามารถแสดงออกมาเป็นแผนผังสถานะของระบบ (state diagram) ได้ดังรูปที่ 3.6 - 3.9 สำหรับไวยากรณ์ของกฎที่ใช้ใน ACL และ SNORT นั้นสามารถดูรายละเอียดได้ในหัวข้อ 2.5 และ 2.7 ตามลำดับ



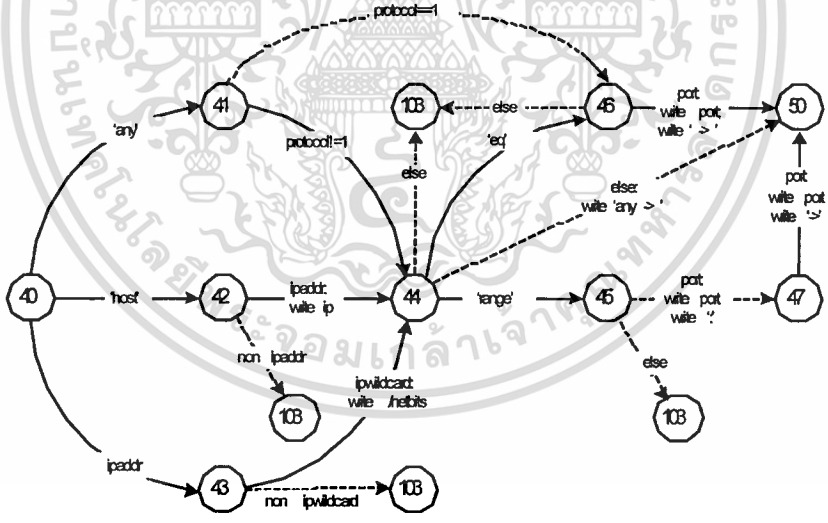
ภาพที่ 3.6 แผนผังสถานะของการแปลงกฎจากรูปแบบ ACL เป็น SNORT (สถานะ 0 ถึง 30)

ในการแปลงกฎในระยะแรกตั้งแต่สถานะที่ 0 ถึง 30 นี้เป็นส่วนที่พิจารณา access-list จาก ACL โดยทำการแปลงเงื่อนไขสำหรับการไม่ส่งผ่านแพคเกต deny ใน ACL เป็น log และการส่งผ่านแพคเกต permit เป็น pass นอกจากนี้ยังทำการตรวจสอบโปรโตคอลที่สนับสนุนทั้ง 4 แบบด้วยคือ TCP, UDP, ICMP และ IP พร้อมกับบันทึกโปรโตคอลนั้นๆ ไว้ สำหรับโปรโตคอล ICMP นั้นจะแยกพิจารณาเป็นพิเศษต่างจาก TCP และ UDP เนื่องจากใน Cisco ACL นั้น ICMP จะไม่ใช้คำสั่ง eq (Equal) ในการระบุหมายเลข port ในระยะที่สองตั้งแต่สถานะที่ 30 ถึง 40 จะเป็นการตรวจสอบ IP address และ port ของทางต้นทาง ตัวแปลกฎนี้จะทำการพิจารณาIP wildcard แล้วแปลงให้เป็นรูปแบบของ mask bits ที่ใช้ใน SNORT ในระยะนี้จะมีการเขียนทิศทางของการจราจรด้วย โดยสนับสนุนทิศทางเดียวจากต้นทางไปปลายทาง (->) เท่านั้น



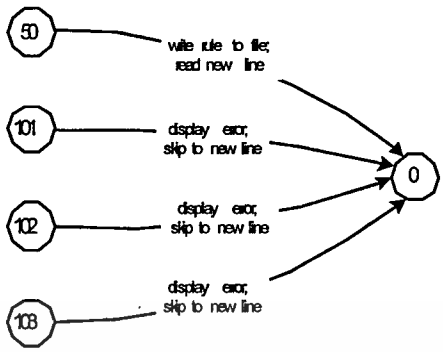
ภาพที่ 3.7 แผนผังสถานะของการแปลงกฎจากรูปแบบ ACL เป็น SNORT (สถานะ 30 ถึง 40)

ในระยะที่สามตั้งแต่สถานะที่ 40 ถึง 50 จะคล้ายกับระยะที่สองนั่นคือพิจารณา IP address และ port เหมือนกันแต่ในส่วนนี้จะเป็นของทางปลายทาง



ภาพที่ 3.8 แผนผังสถานะของการแปลงกฎจากรูปแบบ ACL เป็น SNORT (สถานะ 40 ถึง 50)

สำหรับในส่วนสุดท้ายที่สถานะ 50 นั้นจะทำการเขียนรูปแบบของกฎแบบ SNORT ที่ได้เป็นไฟล์เพื่อการประมวลผลต่อไป จากนั้นก็จะเริ่มแปลงกฎในบรรทัดต่อไปเรื่อยๆ จนกระทั่งหมดไฟล์ของกฎที่อยู่ในรูป ACL ในส่วนของสถานะที่ 101 ถึง 103 นั้นจะเป็นสถานะที่บอกถึงความผิดพลาดที่เกิดจากรูปแบบของกฎ ACL นั้นไม่ถูกต้องผิดจากรูปแบบไป บรรทัดที่มีความผิดพลาดเกิดขึ้นนี้จะถูกข้ามไป



ภาพที่ 3.9 แผนผังสถานะของการแปลงกฎจากรูปแบบ ACL เป็น SNORT (สถานะ 50 ถึง 0 สำหรับเริ่มการแปลงกฎในบรรทัดใหม่ต่อไปจนกว่าจะจบไฟล์)

3.4.4.3 กระบวนการย่อย 1.3: Generate Testing Traffic Pattern

ระบบย่อยส่วนนี้ทำหน้าที่สร้างแพคเกตตามรูปแบบของการจราจรจากนโยบายความปลอดภัยที่ได้จากระบบย่อย 1.2 โดยจะสร้างจากหมายเลข IP ตามโดเมนของต้นทางและปลายทางจากกฎต่างๆ ของนโยบายความปลอดภัยในลักษณะความสัมพันธ์ทั้งหมดที่เป็นไปได้แบบสุ่ม ส่วนโปรโตคอลและพอร์ตนั้นจะเป็นอีกพารามิเตอร์หนึ่งที่ประกอบไปกับคู่ของต้นทางและปลายทาง รูปแบบการจราจรที่ได้นั้นจะแยกออกเป็นอย่างน้อยสองประเภทตามการกระทำของเราเตอร์คือ permit หรือ deny ดังนั้นรูปแบบของการจราจรที่ได้จึงอาจจะแยกเป็นสองส่วนตามการกระทำของเราเตอร์ด้วย ในกรณีที่กฎข้อนั้นๆ มีการระบุ IP address เป็นหมายเลขใดๆ ก็ได้ (any) กระบวนการย่อย 1.3 นี้จะนำ IP address จากไฟล์มาใช้ สำหรับไฟล์ที่เก็บหมายเลข IP นี้จะมีสองไฟล์สำหรับเครือข่ายภายนอกและภายใน ผลลัพธ์ที่ได้นี้จะอยู่ในรูปของกฎในแบบของ SNORT ซึ่งจะนำไปใช้โดยส่วนอื่นๆ โดยเฉพาะระบบย่อย 1.4 ที่ทำหน้าที่ส่งแพคเกตตามกฎแต่ละข้อเพื่อทดสอบเราเตอร์หรืออุปกรณ์ควบคุมเครือข่ายที่เป็นเป้าหมายต่อไป ในขั้นตอนนี้จะมีการสร้างไฟล์ index สำหรับเก็บจุดเริ่มต้นในไฟล์ของกฎแต่ละข้อไว้ได้วยเพื่อที่จะได้การส่งแพคเกตในกระบวนการย่อยที่ 1.4 นั้นไม่ต้องเสียเวลาในการหากฎแต่ละข้ออีกรอบหนึ่ง

3.4.4.4 กระบวนการย่อย 1.4: Send Traffic

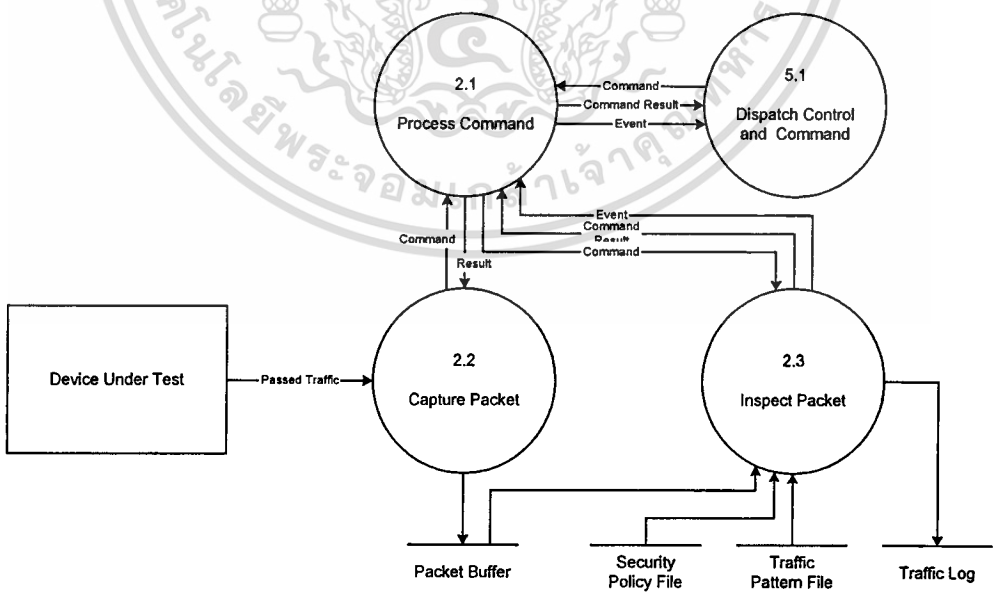
ระบบย่อยส่วนนี้ทำหน้าที่อ่านไฟล์ที่ประกอบไปด้วยรูปแบบการจราจรที่ได้จากระบบย่อย 1.3 หรือจากผู้ทดสอบที่จัดเตรียมขึ้นมาเองแล้วสร้างแพคเกตตามกฎแต่ละข้อเพื่อส่งออกไปยังเครือข่ายปลายทางที่มีเราเตอร์เป็นอุปกรณ์เครือข่ายที่ควบคุมการจราจรเข้าออกแบบกรองแพคเกตโดยใช้ ACL เนื่องจากต้องมีการติดต่อกับเครือข่ายในระดับล่างจึงต้องมีการใช้คำสั่งเกี่ยวกับ socket จาก sys/socket.h ด้วย โดยจะมีการระบุประเภทว่าเป็นการติดต่อแบบ raw socket ในส่วนของการกำหนด

โครงสร้างข้อมูลที่จะใช้นั้นประกอบไปด้วยโครงสร้างของข้อมูลตามโปรโตคอล ICMP TCP และ UDP บน IP ดังที่แสดงไว้ด้านล่างนี้

ในส่วนของการใช้งาน socket ในระดับล่างนั้นจะใช้คำสั่ง socket เพื่อเริ่มการใช้งาน ในตัวอย่างนี้ที่เป็นการใช้ socket สำหรับโปรโตคอล ICMP ที่เห็นตัวแปร ssock นั่นก็คือ file descriptor ที่ใช้ในการอ้างถึง socket ตัวแปรนี้จะใช้ในคำสั่ง sendto() เพื่อการส่งข้อมูลผ่าน socket รวมทั้งเมื่อใช้งานเสร็จแล้วในคำสั่ง close() เพื่อบอกว่าสิ้นสุดการใช้งาน socket นั้น

```
int sendto(int sockfd, const void *msg, int len, unsigned int flags,
const struct sockaddr *to, int tolen);
...
ssock = socket (AF_INET, SOCK_RAW, ICMP);
...
sendto(ssock, buffer, buff_len, 0, (struct
  sockaddr *) &sock_in, sizeof(sock_in);
...
close(ssock);
```

การทำงานของระบบย่อยนี้นอกจากจะเกี่ยวข้องกับการโปรแกรมด้านเครือข่ายแล้ว จะมีส่วนที่ต้องอ่านข้อมูลที่เป็นการจราจรจากไฟล์ขึ้นมาเพื่อทำการประมวลผล และสร้างแพคเกตเพื่อส่งออกไปตามรูปแบบของการจราจรที่กำหนดไว้ตามรูปแบบการจราจรที่สร้างมาจากกระบวนการย่อยที่ 1.3



ภาพที่ 3.10 DFD level 2 ของส่วน Capture Traffic

3.4.5 กระบวนการที่ 2: Capture Traffic

กระบวนการนี้ทำหน้าที่ในการรับแพคเกตที่ผ่านเราเตอร์เข้ามาในเครือข่ายที่ควบคุมไว้โดย จะเก็บไว้เป็นไฟล์ ทำการเทียบกับรูปแบบการจราจรทดสอบ และเก็บผลไว้เพื่อนำไปใช้ในการวิเคราะห์โดยส่วนที่ทำหน้าที่วิเคราะห์และรายงานผลต่อไป นอกเหนือจากการรับแพคเกตแล้วเก็บไว้ ส่วนนี้ยังอาจจะส่งผลลัพธ์ที่ได้ในแบบ online ได้เช่นกัน ในการควบคุมนั้นจะรับคำสั่งในการควบคุมจากผู้ทดสอบผ่านทางกระบวนการที่ 5 อีกต่อหนึ่ง

3.4.5.1 กระบวนการย่อย 2.1: Process Command

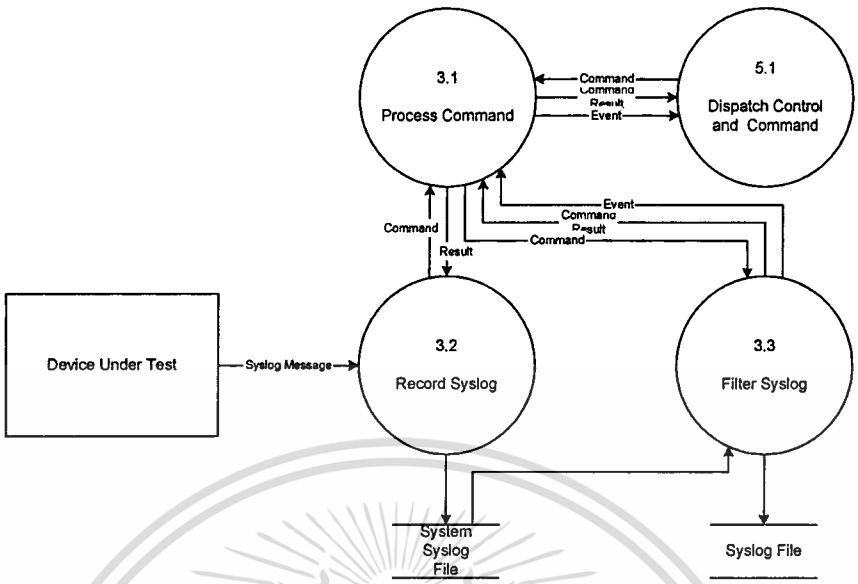
เป็นส่วนที่รับคำสั่งในการควบคุมการทำงานจากส่วนควบคุมกลาง (management station) แล้วกระจายการทำงานไปยังส่วนย่อยต่างๆ ที่อยู่ในระบบ Capture Traffic อีกทีหนึ่งโดยส่งผ่านตัวแปร (parameter) ต่างๆ ตามรูปแบบที่ได้กำหนดไว้แล้วทั้งผ่านระบบเครือข่ายหรือผ่านบรรทัดคำสั่ง ซึ่งถูกส่งต่อมาจากกระบวนการย่อยที่ 5.1

3.4.5.2 กระบวนการย่อย 2.2: Capture Packet

ระบบย่อยส่วนนี้จะทำการเก็บแพคเกตที่มีการส่งผ่านเข้ามาทางอินเตอร์เฟสของเครื่องที่เชื่อมต่ออยู่กับเครือข่ายภายในที่มีอุปกรณ์ควบคุมเครือข่ายหรือเราเตอร์เป็นตัวกลางกั้นระหว่างเครือข่ายภายในกับเครือข่ายภายนอก ในการทำงานนั้นอาศัยไลบรารีในการดักจับแพคเกต libpcap ซึ่งจะคอยตรวจสอบแพคเกตที่ได้รับเข้ามาก่อนที่จะบันทึกแพคเกตที่ได้รับเพื่อใช้ในการตรวจสอบและหาประสิทธิภาพของนโยบายความปลอดภัยของระบบ ในการทำงานนั้นอาศัยฟังค์ชันต่างๆ ที่มีอยู่ใน libpcap โดยจะต้องมีการตรวจสอบแพคเกตในระดับ IP เอง เนื่องจากแพคเกตที่ได้นั้นจะอยู่ในระดับต่ำ (Data Link Layer)

3.4.5.3 กระบวนการย่อย 2.3: Inspect Packet

เป็นระบบย่อยที่ทำหน้าที่ตรวจสอบและบันทึกแพคเกตตามรูปแบบการจราจรในเครือข่ายที่ส่งมาจากระบบย่อย 1.4 Send Traffic การตรวจสอบนั้นจะอาศัยการเปรียบเทียบกับไฟล์รูปแบบการจราจรที่ใช้ในการสร้างแพคเกต และอาศัยการตรวจสอบจากแพคเกตที่ได้รับจากระบบย่อย 2.2 Capture Packet



ภาพที่ 3.11 DFD level 2 ของส่วน Manage Syslog

3.4.6 กระบวนการที่ 3: Manage Syslog

กระบวนการนี้ทำหน้าที่ในการประมวลบันทึกข้อความเหตุการณ์ที่ส่งมาจากเราเตอร์ผ่านเข้ามาทางพอร์ต 514 ของโปรโตคอล UDP โดยจะต้องมีการระบุหมายเลข IP ของ Syslog server ที่เราเตอร์และบอกให้ส่งข้อความเมื่อมีแพคเกตตรงกับเงื่อนไขในกฎโดยการใช้คำสั่งลงท้ายว่า log ใน ACL ด้วย การเลือกให้มีการส่งข้อความอยู่ในรูปแบบดังนี้

```
logging <Syslog IP address>
...
access-list 100 deny tcp any any log
```

การทำงานของส่วนนี้จะใช้โปรแกรม Syslog ภายนอก โดยจะนำเฉพาะไฟล์ Syslog ที่บันทึกไว้มาใช้เท่านั้น การพัฒนาจึงจะเน้นไปในการประมวลผลตามรูปแบบของเหตุการณ์ที่บันทึกไว้ในไฟล์ Syslog

3.4.6.1 กระบวนการย่อย 3.1: Process Command

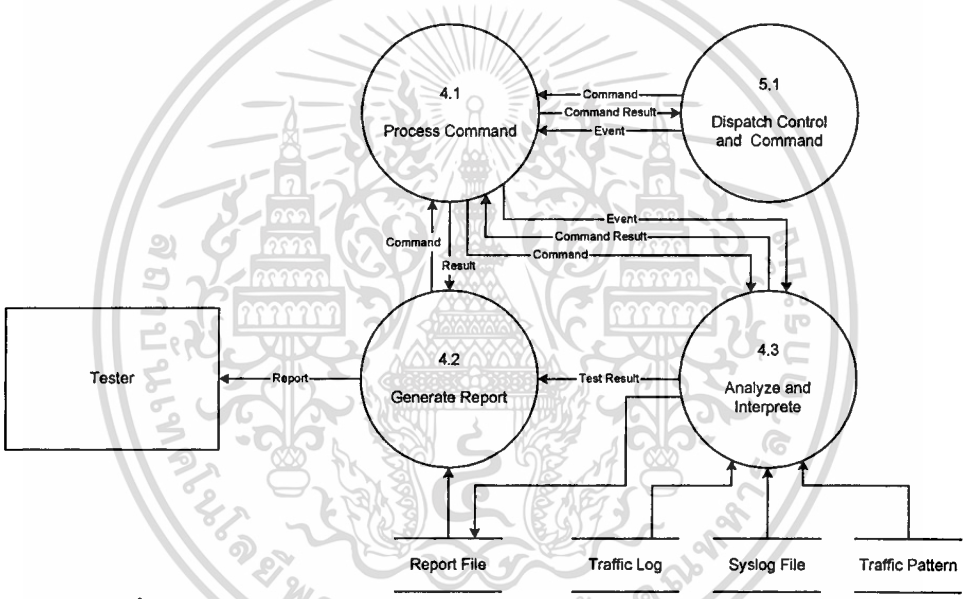
เป็นส่วนที่รับคำสั่งในการควบคุมการทำงานจากส่วนควบคุมกลาง (management station) แล้วกระจายการทำงานไปยังส่วนย่อยต่างๆ ที่อยู่ในระบบที่เกี่ยวข้องกับการจัดการ Syslog อีกทีหนึ่ง โดยส่งผ่านตัวแปร (parameter) ต่างๆ ตามรูปแบบที่ได้กำหนดไว้แล้วทั้งผ่านระบบเครือข่ายหรือผ่านบรรทัดคำสั่ง

3.4.6.2 กระบวนการย่อย 3.2: Record Syslog

ส่วนที่ทำหน้าที่ในการรับข้อความที่ส่งมาจากเราเตอร์แล้วบันทึกไว้เป็นไฟล์ syslog นี้อาศัยบริการ Syslog ของระบบปฏิบัติการ โดยทำการตั้งค่า syslog สำหรับการบันทึกข้อความที่ส่งมาจากเราเตอร์แยกออกเป็นไฟล์ syslog ต่างหากเพื่อทำการประมวลผลต่อไป

3.4.6.3 กระบวนการย่อย 3.3: Filter Syslog

เนื่องจากข้อความที่ส่งมาจากเราเตอร์นั้นอาจจะมีมาจากอุปกรณ์หลายตัว รวมทั้งไม่ได้แยกประเภทของข้อความ ดังนั้นกระบวนการย่อย 3.3 นี้จะทำการพิจารณาแยกเอาเฉพาะข้อความที่เกี่ยวข้องกับการทำงานของ ACL ที่กำหนดไว้เท่านั้น แล้วบันทึกเป็นไฟล์เพื่อใช้ในการวิเคราะห์ต่อไป



ภาพที่ 3.12 DFD level 2 ของส่วน Analyze and Generate Report

3.4.7 กระบวนการที่ 4: Analyze and Generate Report

กระบวนการนี้เป็นส่วนที่ทำหน้าที่วิเคราะห์ผลลัพธ์ที่เกิดขึ้นภายหลังการทดสอบส่งแพคเกตผ่านอุปกรณ์ควบคุมเครือข่าย โดยพิจารณาจากนโยบายความปลอดภัยที่กำหนดไว้ และแพคเกตที่ได้รับโดยกระบวนการที่ 2 รวมถึง Syslog ที่ส่งจากอุปกรณ์ควบคุมเครือข่าย หลักการวิเคราะห์โดยเบื้องต้นจะพิจารณาจากนโยบายที่ตั้งไว้ว่าสามารถครอบคลุมเงื่อนไขต่างๆ ได้หรือไม่ มีแพคเกตใดบ้างที่ตรงตามเงื่อนไขและถูกกั้นไว้ (สามารถตรวจสอบได้จาก Syslog หรือพิจารณาจาก time-out ที่ฝั่งรับแพคเกต) มีแพคเกตใดบ้างที่ผ่านอุปกรณ์ควบคุมเครือข่ายไปได้ (ตรวจสอบจากฝั่งรับแพคเกต) จัดทำเป็นรายงานเพื่อนำเสนอต่อผู้ทดสอบ

3.4.7.1 กระบวนการย่อย 4.1: Process Command

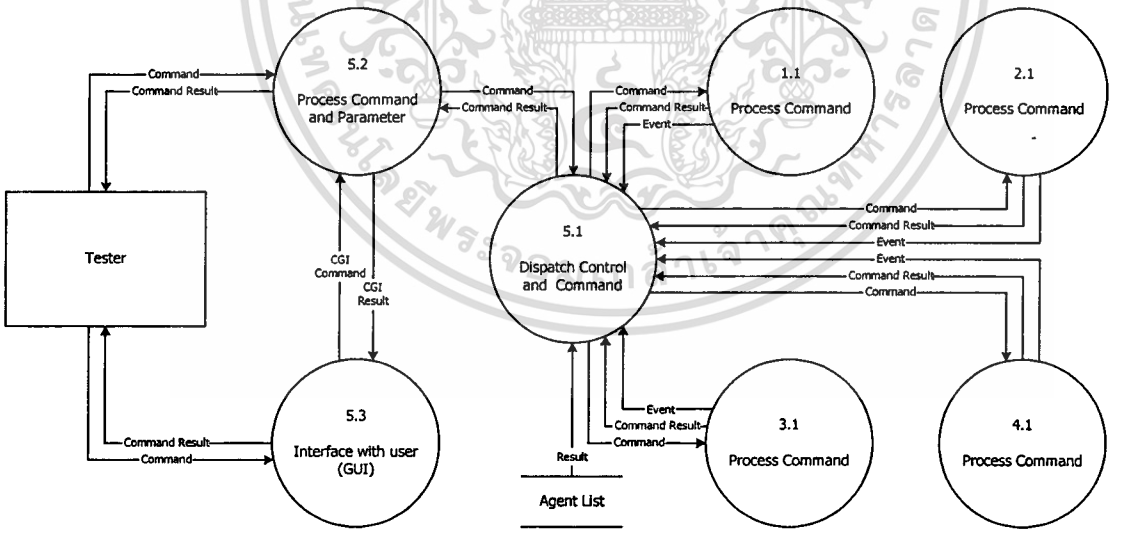
เป็นส่วนที่รับคำสั่งในการควบคุมการทำงานจากส่วนควบคุมกลาง (management station) แล้วกระจายการทำงานไปยังส่วนย่อยต่างๆ ที่อยู่ในระบบที่เกี่ยวข้องกับการวิเคราะห์และทำรายงานอีกทีหนึ่งโดยส่งผ่านตัวแปร (parameter) ต่างๆ ตามรูปแบบที่ได้กำหนดไว้แล้วทั้งผ่านระบบเครือข่ายหรือผ่านบรรทัดคำสั่ง

3.4.7.2 กระบวนการย่อย 4.2: Generate Report

ในส่วนนี้จะนำผลลัพธ์ที่ได้จากการวิเคราะห์โดยกระบวนการย่อยที่ 4.3 มาสร้างเป็นรายงานเพื่อนำเสนอต่อผู้ทดสอบ

3.4.7.3 กระบวนการย่อย 4.3: Analyze and Interprete

เป็นกระบวนการย่อยที่วิเคราะห์ผลลัพธ์ที่ได้จากการบันทึกแพคเกตจากกระบวนการย่อย Capture Packet รวมทั้งรูปแบบของการจราจร และ Syslog ของ ACL ที่ใช้ในการทดสอบด้วย โดยพิจารณาเทียบว่าแพคเกตที่ได้รับหรือถูกกั้นไว้นั้นตรงกับกฎในบรรทัดใด และตรวจสอบว่าอุปกรณ์ควบคุมเครือข่ายนั้นสามารถกลั่นกรองได้ตาม ACL ที่กำหนดไว้หรือไม่ ข้อมูลจากการวิเคราะห์นี้จะถูกจัดเก็บไว้เป็นข้อมูลสำหรับการทำรายงานโดยกระบวนการย่อย 4.2 ต่อไป



ภาพที่ 3.13 DFD level 2 ของส่วน Manage and Control System

3.4.8 กระบวนการที่ 5: Manage and Control System

กระบวนการนี้ทำหน้าที่ควบคุมและประสานการทำงานระหว่างกระบวนการต่างที่เป็นส่วนประกอบของระบบที่ทำงานอยู่บน Traffic Generator Agent กับ Traffic Analyzer Agent โดยมีการกำหนดรูปแบบของข้อความคำสั่งและโปรโตคอลในการติดต่อและคำสั่งที่ใช้นี้ตามที่ออกแบบไว้ใน

เรื่องข้อกำหนดสำหรับการติดต่อกับส่วนต่างๆ ของระบบ นอกเหนือจากนี้ยังเป็นส่วนที่รับคำสั่งจากผู้ทำการทดสอบ ส่งต่อคำสั่งนั้นไปยังส่วนประกอบของระบบที่มีหน้าที่ตอบสนองต่อคำสั่ง และยังทำหน้าที่ดูแลให้การทำงานในส่วนต่างๆ มีจังหวะสอดคล้องกันด้วย

ส่วนนี้ยังดูแลการลงทะเบียนของ Agent ต่างๆ ที่ทำงานอยู่ในระบบเครือข่ายอีกด้วย

3.4.8.1 กระบวนการย่อย 5.1: Dispatch Control and Command

กระบวนการย่อยส่วนนี้เป็นส่วนที่รับคำสั่งและการควบคุมจากผู้ใช้ ผ่านทางส่วนติดต่อกับผู้ใช้แล้วส่งต่อไปให้ยัง Agent ที่ทำงานอยู่ผ่านทางระบบเครือข่าย การรับส่งข้อมูลจาก Agent แต่ละตัวนั้นสามารถทำได้โดยอาศัยข้อมูลที่ได้มีการลงทะเบียนไว้โดยเฉพาะ IP address และประเภทของ Agent นั้นว่าเป็น Traffic Generator หรือ Packet Capture Agent

3.4.8.2 กระบวนการย่อย 5.2: Process Command and Parameter

เป็นส่วนที่ติดต่อกับผู้ใช้ทั้งในการรับคำสั่ง และการแสดงผลลัพธ์จากการทำงานแบบตัวอักษร โดยมีรูปแบบของการติดต่อกับผู้ใช้เป็นแบบการรับคำสั่งผ่านบรรทัดคำสั่ง

3.4.8.3 กระบวนการย่อย 5.3: Interface with user

เป็นส่วนที่ติดต่อกับผู้ใช้แบบกราฟฟิค โดยอาจจะมีทางเลือกในแบบ web ผ่านทาง browser เป็นทางเลือกเสริมนอกจากส่วนติดต่อกับผู้ใช้แบบตัวอักษร การรับส่งข้อมูลกับกระบวนการอื่นๆ นั้นสามารถทำได้โดยอาศัยการติดต่อกับโปรแกรมแบบ CGI

3.5 ข้อกำหนดในการติดต่อกับส่วนต่างๆ ของระบบ

3.5.1 คำสั่ง (Command) ในการทำงานและควบคุม

รูปแบบของคำสั่งนั้นอยู่ในลักษณะของคำสั่งที่เป็นข้อความ (command string) โดยที่แต่ละคำสั่งนั้นจะมีพารามิเตอร์ประกอบแตกต่างกันไปตามที่กำหนดไว้สำหรับแต่ละส่วนประกอบย่อยในระบบ โดยมีรูปแบบทั่วไปของคำสั่งตามภาพที่ 3.14 ซึ่งคำสั่งนั้นจะจบภายในหนึ่งบรรทัดโดยลงท้ายด้วย อักขระ 0 (NULL) เสมอ

Command	Parameter1 Parameter2 ...	NULL

ภาพที่ 3.14 รูปแบบของคำสั่งในการทำงานและควบคุมภายในระบบทดสอบนโยบายความปลอดภัยอุปกรณ์ควบคุมเครือข่าย

3.6 รูปแบบแฟ้มข้อมูล

3.6.1 แฟ้มข้อมูลเก็บ IP address สำหรับเครือข่ายภายในและภายนอก

รูปแบบของไฟล์ที่ใช้เก็บ IP address นี้อยู่ในรูปแบบเดียวกับที่ใช้ใน SNORT โดยใช้การระบุจำนวนของบิตเครือข่ายแทนการใช้ subnet mask โดยมีรูปแบบดังนี้

```
a.b.c.d/netbits
```

ตัวอย่างของการกำหนดเครือข่ายที่ใช้ในการสร้างการจราจรสำหรับการทดสอบ เครือข่ายนี้จะถูกใช้ในการสร้างไฟล์ที่เก็บการจราจรสำหรับการทดสอบในขั้นตอนการแทนค่า any

```
172.19.0.0/16
198.162.255/24
172.30.0.0/16
159.129.141.0/24
```

3.6.2 แฟ้มข้อมูลนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่ายในแบบ ACL

รูปแบบของของไฟล์นโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครอข่ายที่ใช้นั้นเป็นรูปแบบตาม Cisco ACL ที่ใช้ในเราเตอร์ของทาง Cisco ซึ่งมีรูปแบบตามที่ได้อธิบายไว้แล้วในหัวข้อที่ 2.5 โดยโปรแกรมที่พัฒนาขึ้นมานี้สามารถแปลงให้อยู่ในรูปแบบของกฎที่ใช้ใน SNORT เพื่อใช้ในการสร้างรูปแบบการจราจรสำหรับการทดสอบ

ตัวอย่างของนโยบายความปลอดภัยของเราเตอร์แบบ ACL ต้นฉบับ

```
access-list 101 permit icmp any any echo
access-list 101 permit icmp any any echo-reply
access-list 101 permit icmp any any traceroute
access-list 101 deny   udp any eq 2000 any log
access-list 101 deny   tcp any eq 2000 any log
access-list 101 deny   tcp any any range 5000 5004 log
access-list 101 permit tcp host 1.10.10.10 eq telnet 172.29.46.0 0.0.1.255
access-list 101 permit tcp host 1.10.10.10 eq ftp 172.29.46.0 0.0.1.255
access-list 101 permit tcp host 1.10.10.10 eq ftp-data 172.29.46.0 0.0.1.255
```

3.6.3 แฟ้มข้อมูลนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่ายในแบบ SNORT

รูปแบบของไฟล์นโยบายความปลอดภัยที่ใช้ในการสร้างรูปแบบของการจราจรนั้นใช้รูปแบบของกฎที่ใช้ SNORT ซึ่งได้อธิบายไว้ในหัวข้อ 2.7

ตัวอย่างของนโยบายความปลอดภัยในรูปแบบ SNORT ที่แปลงมาจาก ACL

```
pass icmp any any -> any 8
pass icmp any any -> any 0
pass icmp any any -> any 11
log udp any 2000 -> any any
log tcp any 2000 -> any any
log tcp any any -> any 5000:5004
pass tcp 1.10.10.10/32 23 -> 172.29.46.0/23 any
pass tcp 1.10.10.10/32 21 -> 172.29.46.0/23 any
pass tcp 1.10.10.10/32 20 -> 172.29.46.0/23 any
```

3.6.4 แฟ้มข้อมูลเก็บรูปแบบของการจราจรเพื่อการทดสอบอุปกรณ์ควบคุมเครือข่าย

แฟ้มข้อมูลที่เก็บรูปแบบของการจราจรนี้อยู่ในรูปแบบของกฎใน SNORT โดยถูกสร้างขึ้นจากแฟ้มข้อมูลที่เก็บนโยบายความปลอดภัยในรูปแบบ SNORT อีกทีหนึ่ง โดยทำการแทนค่าของ IP address ในกรณีที่เป็นเครือข่ายด้วย IP address เดี่ยวๆ และแทนค่าของ IP address แบบ any ด้วย IP address ที่มีการกำหนดไว้เป็นแฟ้มข้อมูลต่างหากสำหรับเครือข่ายภายในและภายนอก หมายเลขพอร์ตก็จะมีการแทนค่าเช่นกัน โดยการแทนค่านั้นจะอาศัยการสุ่มตามช่วงที่กำหนดไว้โดยนโยบายแต่ละข้อ

ตัวอย่างของรูปแบบการจราจรสำหรับการทดสอบอุปกรณ์ควบคุมเครือข่าย

```
pass icmp 40.29.74.128/32 51567 -> 159.129.141.51/32 8
pass icmp 40.29.74.128/32 51567 -> 198.162.255.187/32 8
pass icmp 40.29.74.128/32 51567 -> 172.19.179.212/32 8
pass icmp 40.29.74.128/32 51567 -> 198.162.255.162/32 8
pass icmp 177.43.235.68/32 55769 -> 172.19.188.158/32 0
pass icmp 177.43.235.68/32 55769 -> 159.129.141.102/32 0
pass icmp 177.43.235.68/32 55769 -> 198.162.255.74/32 0
pass icmp 177.43.235.68/32 55769 -> 159.129.141.136/32 0
pass icmp 59.240.243.209/32 39029 -> 172.30.2.159/32 11
pass icmp 59.240.243.209/32 39029 -> 172.30.45.76/32 11
pass icmp 59.240.243.209/32 39029 -> 172.19.3.246/32 11
pass icmp 59.240.243.209/32 39029 -> 172.19.135.154/32 11
```

บทที่ 4
การพัฒนาระบบงาน

ขั้นตอนต่อไปซึ่งเป็นผลสืบเนื่องมากจากการออกแบบที่ได้จากการศึกษาและวิเคราะห์ไว้ นำมาพัฒนาเป็นโปรแกรมใช้งานจริง โดยมีรายละเอียดดังนี้

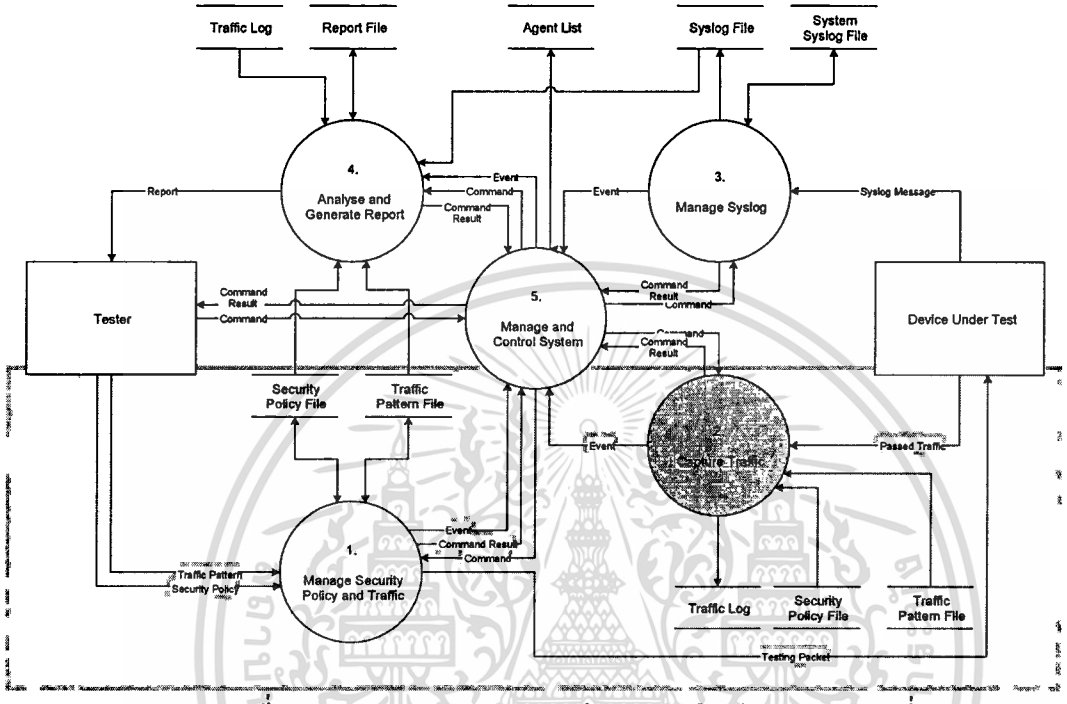
4.1 สภาพแวดล้อมในการการพัฒนาโปรแกรม

เพื่อความคล่องตัว การสนับสนุนโปรแกรมทางเครือข่ายในระดับล่าง และสามารถโยกย้ายสภาพแวดล้อมในการพัฒนาและการใช้งานได้ การพัฒนาระบบทดสอบนโยบายนี้จึงเลือกใช้การพัฒนาโดยใช้ภาษาซีบนสภาพแวดล้อมของ UNIX ผ่านทางซีคอมไพเลอร์ GCC เป็นหลัก โดยจะทำการพัฒนาต้นแบบบนระบบปฏิบัติการวินโดวส์บนพื้นฐานของสภาพแวดล้อม CYGWIN อีกทีหนึ่ง CYGWIN นี้จะจำลอง API (Application Program Interface) ของระบบปฏิบัติการ UNIX ตามมาตรฐาน POSIX 1/90 สนับสนุนการเรียกใช้ system call และบริการต่างๆ ของ BSD และ SVR4 รวมถึง Berkeley socket ผ่านทาง DLL ของระบบ Win32 ด้วย แต่หลังจากที่ได้มีการทดสอบไปในระดับหนึ่งแล้วพบว่า CYGWIN (POSIX Emulation DLL API เวอร์ชัน 0.39 เวอร์ชันไฟล์ 1.32) ที่ใช้ในมีปัญหาไม่สามารถใช้คำสั่ง setsockopt() ที่ใช้ในการเลือกควบคุมส่วนหัวของแพคเกตของ IP ได้ โดยเกิดข้อความแสดงความผิดพลาด (General Projection Fault) ขึ้น จึงต้องพัฒนาโปรแกรมบนระบบปฏิบัติการ Linux เพียงอย่างเดียว แต่อย่างไรก็ตามในส่วนการทำงานอื่นๆ ก็ยังสามารถทดสอบการทำงานบน CYGWIN ได้เป็นอย่างดี อย่างไรก็ตามโปรแกรมสามารถย้ายให้ไปทำงานบนระบบปฏิบัติการวินโดวส์ได้โดยต้องมีการแก้ไขในส่วนของการติดต่อกับระบบเครือข่ายและการทำงานกับโปรเซส

4.2 ขอบข่ายในการพัฒนาโปรแกรม

จากระบบทดสอบนโยบายความปลอดภัยสำหรับการควบคุมเครือข่ายที่ได้รับการออกแบบและอธิบายไว้แล้วในบทที่ 3 สู่การพัฒนาเพื่อการทดสอบใช้งานจริงนั้นจะพัฒนาเฉพาะส่วนประกอบย่อยที่ 1 ซึ่งเกี่ยวข้องกับการจัดการนโยบายความปลอดภัย การสร้างรูปแบบจราจรสำหรับการทดสอบ และการส่งแพคเกตจากรูปแบบการจราจรก่อน ส่วนประกอบย่อยที่ 2 ที่ใช้ในการดักจับแพคเกตที่สร้างขึ้นมานั้นใช้เครื่องมือและไลบรารีที่มีใช้งานอยู่แล้ว เพื่อใช้ในการยืนยันความถูกต้อง

ในการทำงานของส่วนประกอบย่อยที่ 1 ส่วนระบบย่อยทั้ง 1 และ 2 ที่พัฒนานี้ได้แสดงไว้ใน DFD ดังภาพที่ 4.1



ภาพที่ 4.1 ขอบเขตของการพัฒนาโปรแกรมในส่วนระบบย่อยที่ 1

4.3 การโปรแกรมด้านเครือข่ายผ่านทาง socket

ในการเขียนโปรแกรมเพื่อติดต่อกับเครือข่ายเพื่อการรับส่งข้อมูล โปรแกรมจะติดต่อผ่านทาง socket ซึ่งเป็นช่องทางในการสื่อสารจากต้นทางไปยังปลายทาง ซึ่งมีอยู่หลายประเภทตามการใช้งาน คือแบบ connection-oriented ที่สามารถรับส่งข้อมูลไปกลับได้ตามที่ต้องการ และแบบ connectionless หรือที่เรียกว่า datagram ซึ่งสามารถส่งได้ทีละข้อความเท่านั้น นอกจากนี้ในระบบ UNIX มีตระกูลของ socket อยู่หลายแบบ แต่มี 2 แบบที่ใช้กันทั่วไปคือ AF_INET สำหรับการเชื่อมต่อแบบอินเตอร์เนต และ AF_UNIX สำหรับการติดต่อสื่อสารกันระหว่างโปรเซส (Interprocess communication - IPC) รูปแบบของการเรียนใช้ socket เป็นดังนี้

```
socket_descriptor = socket(domain, type, protocol)
int socket_descriptor, domain, type, protocol;
```

ดังที่กล่าวมาแล้วส่วนที่เป็น domain นั้นจะเป็นตัวที่ใช้กำหนดรูปแบบของโครงสร้างข้อมูลที่จะใช้ตามตระกูลของ socket ตัวอย่างเช่นโครงสร้างของ address เป็นต้น การกำหนดรูปแบบ

ต่างของแบบข้อมูลนี้อยู่ในไฟล์ <sys/socket.h> ซึ่งต้องมีการประกาศไว้ในส่วนต้นของโปรแกรมเสมอ ในกรณีนี้จะเป็นการใช้งานที่เกี่ยวข้องกับโดเมนแบบ AF_INET เป็นหลัก การระบุแบบของข้อมูล address ของโดเมนนี้ประกอบไปด้วยค่า 2 ค่าคือส่วนที่เป็น host address ที่เป็นหมายเลข IP และหมายเลขพอร์ต โครงสร้างข้อมูล sockaddr_in นี้ถูกกำหนดไว้ในไฟล์ <netinet/in.h> ดังนี้

```
struct sockaddr_in {
  short sin_family;         /* AF_INET */
  u_short sin_port;         /* port number */
  struct in_addr sin_addr; /* see struct below */
  char sin_zero[8];         /* not used */
}
struct in_addr {
  union {
    struct { u_char s_b1,s_b2,s_b3,s_b4; }
      S_un_b;
    struct { u_short s_w1,s_w2; } S_un_w;
      u_long S_addr;
  } S_un;
#define s_addr S_un.S_addr /* can be used for most tcp & ip code */
}
```

4.4 การติดตั้งเครื่องมือต่างๆ ที่ใช้

เครื่องมือที่ใช้เป็นหลักในการพัฒนาโปรแกรมนี้คือ GNU C Compiler หรือ GCC ซึ่งถูกเลือกติดตั้งอยู่ในระบบปฏิบัติการ Linux อยู่แล้ว ในการติดตั้ง Linux Redhat นั้นจะต้องเลือกให้มีการติดตั้งเครื่องมือในการพัฒนา (Development Tool) ด้วย สำหรับ GCC ที่ใช้ในการพัฒนานี้เป็นเวอร์ชัน 2.96 บนไมโครโปรเซสเซอร์แบบ i386

4.4.1 การสร้าง makefile สำหรับ GCC

เพื่อให้สามารถคอมไพล์และลิงค์โปรแกรมซึ่งมีหลายไฟล์แบบไลบรารีได้สะดวกขึ้นจึงได้สร้าง makefile ขึ้นเป็นสคริปต์สำหรับ GCC สำหรับตัวอย่างนี้เป็น makefile สำหรับโปรแกรม sender ซึ่งสามารถใช้ได้หลายระบบปฏิบัติการ รูปแบบเป็นดังนี้

```
# Security Policy Testing System
# by Chinnawat Tivitmahaisoon
# Generic Makefile

# Linux / *BSD* / Others
CC = gcc
CFLAGS = -Wall -O3
CLIBS =

# Solaris (IRIX / AIX / HPUX ?)
#CC = gcc
#CFLAGS = -Wall -O3
#CLIBS = -lnsl -lsocket

# Win32 (cygwin)
#CC = gcc
#CFLAGS = -Wall -DWINDOZE -O2
#CLIBS =

SENDER_OBJ = ../ip.o ../netlib.o ../genlib.o sender.o

all: sender.o

clean:
        @echo removing junk...
        @rm -f sender *.exe *.o *~ *.out

sender:
        ${SENDER_OBJ}
        ${CC} ${CFLAGS} ${CLIBS} ${SENDER_OBJ} -o sender
```

แฟ้มข้อมูลที่ 4.1 makefile สำหรับโปรแกรม sender

4.4.2 การติดตั้งไลบรารี libpcap

ไลบรารี libpcap (Packet Capture) นี้เป็นไลบรารีที่รวบรวมคำสั่งสำหรับใช้งานฟังค์ชันของส่วนดักจับแพคเกตโดยไม่ขึ้นกับระบบปฏิบัติการโดยสนับสนุนรูปแบบของการดักจับแพคเกตแบบต่างๆ เช่น BPF (BSD Packet Filter) สำหรับระบบปฏิบัติการ UNIX BSD, DLPI (Data Link Provider Interface) ที่ทำงานอยู่ภายใต้ Solaris 2.X, NIT ของ SunOS และ SOCK_PACKET สำหรับ Linux สำหรับ libpcap นี้สามารถดาวน์โหลดได้จาก

`ftp.de.lbl.gov/libpcap.tar.z`

ในการพัฒนาส่วนดักจับแพคเกตของระบบทดสอบนโยบายความปลอดภัยของอุปกรณ์ควบคุมเครือข่ายนั้นก็ได้ใช้ libpcap ด้วย ขั้นตอนการติดตั้ง libpcap มีดังนี้

1. ขยายไฟล์ที่ถูกรวมย่อไว้โดยใช้คำสั่ง gunzip libpcap.tar.z และ tar -xvf libpcap.tar (libpcap ที่ใช้นี้เป็นเวอร์ชัน 0.4)

2. ทำการสร้างไฟล์สำหรับการติดตั้งโดยเรียก ./configure โปรแกรมจะสร้าง Makefile จาก Makefile.in ตามระบบที่ทำการติดตั้ง
3. เรียกโปรแกรม make เสร็จแล้วใช้คำสั่ง su เพื่อเปลี่ยนผู้ใช้เป็น root
4. ป้อนคำสั่ง make install แล้วตามด้วย make install-incl และ make install-man ในกรณีที่ไม่มีไดเรคทอรีปลายทางสำหรับ include และ man จะต้องสร้างก่อน

4.5 การทดสอบโปรแกรม

ในส่วนของการทดสอบโปรแกรมที่ได้พัฒนาขึ้นนั้นจะประกอบไปด้วยการทดสอบต้นแบบโปรแกรมและฟังค์ชันสำหรับการส่งแพตเกตในระดับต่ำ และการทดสอบโปรแกรมโดยรวมซึ่งครอบคลุมถึงขั้นตอนการแปลงนโยบายความปลอดภัยให้เป็นรูปแบบการจราจรสำหรับการทดสอบและการส่งแพคเกตไปยังเราเตอร์เพื่อทดสอบนโยบายความปลอดภัยตามที่ได้กำหนดไว้

4.5.1 ฟังค์ชันสำหรับการส่งแพคเกต

โปรแกรม Sender นี้เป็นส่วนประกอบหนึ่งของกระบวนการย่อย 1.5 โดยทำหน้าที่อ่านรูปแบบการจราจรจากไฟล์ (ในกรณีนี้ใช้ไฟล์ตัวอย่าง test.rules) ฟังก์ชันที่ใช้นั้นมีการกำหนดไว้ในไลบรารีสองไฟล์คือ netlib ซึ่งประกอบไปด้วยฟังก์ชันที่เกี่ยวกับเครือข่าย และ genlib ซึ่งประกอบไปด้วยฟังก์ชันทั่วไป จึงต้องมีการประกาศ include header ไฟล์ในส่วนเริ่มต้นของโปรแกรม ฟังก์ชันหลักที่มีการเรียกใช้ในโปรแกรมนี้คือฟังก์ชัน rawtransmit() ซึ่งใช้ในการส่งข้อมูลแพคเกตสู่เครือข่าย โดยสามารถเลือกโปรโตคอล TCP, UDP หรือ ICMP กำหนด IP address และพอร์ตของต้นทางและปลายทาง และข้อมูลที่ต้องการจะส่ง

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "netlib.h"
#include "genlib.h"

int debug=1;

int main(int argc, char *argv[])
{
  int len, iProto;
  FILE *fp;
  char linebuff[255], szSrc[20], szDst[20], szPro[8], szSrcport[8],
    szDstport[8];
  unsigned long srcaddr, dstaddr;
  unsigned int uiSrcport, uiDstport;

  fp = fopen("test.rules", "rt");
  if (fp) {
    while ((len=filegetline(fp, linebuff, 254))>0) {
      if (len>1) {
        sscanf(linebuff,"%s %s %s %s %s",szPro,szSrc,szSrcport,
          szDst, szDstport);
        srcaddr = (unsigned long)inet_addr(szSrc);
        dstaddr = (unsigned long)inet_addr(szDst);
        uiSrcport = atoi(szSrcport);
        uiDstport = atoi(szDstport);
        iProto = str2proto(szPro);
        rawtransmit(srcaddr,uiSrcport,dstaddr,uiDstport,
          iProto,"hello",5);
        }
      }
    }
  fclose(fp);
  return 0;
}
```

แฟ้มข้อมูลที่ 4.2 โปรแกรมต้นฉบับ sender.c สำหรับทดสอบฟังก์ชัน rawtransit()

```
tcp 1.1.1.1 1024 1.1.1.2 1024
udp 10.10.10.10 2048 10.10.10.11 2048
icmp 100.100.100.100 0 100.100.100.101 0
```

แฟ้มข้อมูลที่ 4.3 test.rules เก็บรูปแบบการจราจรในการทดสอบ

ผลการทดสอบการทำงานของโปรแกรมนี้ใช้โปรแกรม LAN Sniffer Pro 4.5 ในการตรวจสอบแพคเกตที่ส่งออกมาจากเครื่องทดสอบ ผลลัพธ์จากการถอดความหมาย (decode) นั้นแสดงให้เห็นว่าข้อมูลในแต่ละแพคเกตที่ส่งออกมานั้นถูกต้องเป็นไปตามที่กำหนดไว้ ไม่ว่าจะเป็นหมายเลข IP โปรโตคอล หมายเลขพอร์ต ข้อมูล และ checksum รายละเอียดของผลลัพธ์นั้นเป็นดังนี้

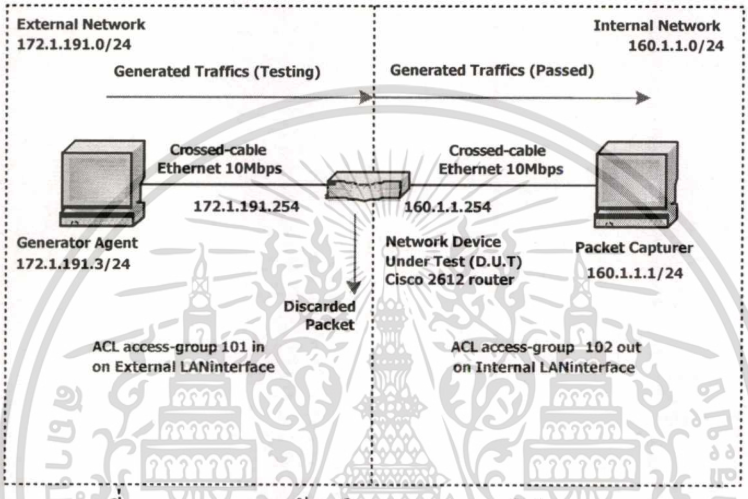
```
- - - - - - - - - - - - - - - - - - - - - - Frame 6 - - - - - - - - - - - - - - - - - -
   Frame Status Source Address    Dest. Address      Size Rel. Time     Delta Time
       6        [1.1.1.1]         [1.1.1.2]           60 0:00:00.486   0.035.657
  DLC:  ----- DLC Header -----
        DLC:
        DLC:  Frame 6 arrived at  02:18:38.2980; frame size is 60 (003C hex) bytes.
        DLC:  Destination = Station Intel EA1A3D
        DLC:  Source      = Station 0004764492D4
        DLC:  Ethertype   = 0800 (IP)
        DLC:
  IP: ----- IP Header -----
        IP:
        IP: Version = 4, header length = 20 bytes
        IP: Type of service = 00
        IP:       000. ....   = routine
        IP:       ...0 .... = normal delay
        IP:       .... 0... = normal throughput
        IP:       .... .0.. = normal reliability
        IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
        IP:       .... ...0 = CE bit - no congestion
        IP: Total length    = 45 bytes
        IP: Identification  = 65531
        IP: Flags           = 0X
        IP:       .0.. .... = may fragment
        IP:       ..0. .... = last fragment
        IP: Fragment offset = 0 bytes
        IP: Time to live    = 230 seconds/hops
        IP: Protocol        = 6 (TCP)
        IP: Header checksum = D0CA (correct)
        IP: Source address      = [1.1.1.1]
        IP: Destination address = [1.1.1.2]
        IP: No options
        IP:
  TCP: ----- TCP header -----
        TCP:
        TCP: Source port                = 1024
        TCP: Destination port           = 1024
        TCP: Initial sequence number = 3714867
        TCP: Next expected Seq number= 3714873
        TCP: Acknowledgment number   = 0
        TCP: Data offset             = 20 bytes
        TCP: Flags                   = 12
        TCP:               ..0. .... = (No urgent pointer)
        TCP:               ...1 .... = Acknowledgment
        TCP:               .... 0... = (No push)
        TCP:               .... .0.. = (No reset)
        TCP:               .... ..1. = SYN
        TCP:               .... ...0 = (No FIN)
        TCP: Window                  = 0
        TCP: Checksum                = B08B (correct)
        TCP: No TCP options
        TCP: [5 Bytes of data]
        TCP:
  ADDR  HEX                                                 ASCII
  0000: 00 a0 c9 ea 1a 3d 00 04 76 44 92 d4 08 00 45 00 | .....=..vD....E
  0010: 00 2d ff fb 00 00 e6 06 d0 ca 01 01 01 01 01 01 | .-..............
  0020: 01 02 04 00 04 00 00 38 af 33 00 00 00 00 50 12 | .......8.3....P.
```

```
0030: 00 00 b0 8b 00 00 68 65 6c 6c 6f 00             | ......hello.

- - - - - - - - - - - - - - - - - - - - Frame 7 - - - - - - - - - - - - - - - - - -
 Frame Status Source Address    Dest. Address      Size Rel. Time     Delta Time
     7        [10.10.10.10]     [10.10.10.11]        60 0:00:00.487   0.000.491
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 7 arrived at  02:18:38.2984; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station Intel EA1A3D
      DLC:  Source      = Station 0004764492D4
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
      IP:       .... ...0 = CE bit - no congestion
      IP: Total length    = 33 bytes
      IP: Identification  = 23559
      IP: Flags           = 0X
      IP:       .0.. .... = may fragment
      IP:       ..0. .... = last fragment
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 219 seconds/hops
      IP: Protocol        = 17 (UDP)
      IP: Header checksum = 5B9C (correct)
      IP: Source address      = [10.10.10.10]
      IP: Destination address = [10.10.10.11]
      IP: No options
      IP:
UDP: ----- UDP Header -----
      UDP:
      UDP: Source port      = 2048
      UDP: Destination port = 2048
      UDP: Length           = 13
      UDP: Checksum         = 83D9 (correct)
      UDP: [5 byte(s) of data]
      UDP:
ADDR  HEX                                                ASCII
0000: 00 a0 c9 ea 1a 3d 00 04 76 44 92 d4 08 00 45 00 | .....=..vD....E.
0010: 00 21 5c 07 00 00 db 11 5b 9c 0a 0a 0a 0a 0a 0a | .!\.....[.......
0020: 0a 0b 08 00 08 00 00 0d 83 d9 68 65 6c 6c 6f 00 | ..........hello.
0030: 00 00 00 00 00 00 00 00 00 00 00 00             | ............

- - - - - - - - - - - - - - - - - - - - Frame 8 - - - - - - - - - - - - - - - - - -
 Frame Status Source Address    Dest. Address      Size Rel. Time     Delta Time
     8        [100.100.100.100] [100.100.100.101]    60 0:00:00.487   0.000.007
DLC:  ----- DLC Header -----
      DLC:
      DLC:  Frame 8 arrived at  02:18:38.2985; frame size is 60 (003C hex) bytes.
      DLC:  Destination = Station Intel EA1A3D
      DLC:  Source      = Station 0004764492D4
```

```
      DLC:  Ethertype   = 0800 (IP)
      DLC:
IP: ----- IP Header -----
      IP:
      IP: Version = 4, header length = 20 bytes
      IP: Type of service = 00
      IP:       000. ....   = routine
      IP:       ...0 .... = normal delay
      IP:       .... 0... = normal throughput
      IP:       .... .0.. = normal reliability
      IP:       .... ..0. = ECT bit - transport protocol will ignore the CE bit
      IP:       .... ...0 = CE bit - no congestion
      IP: Total length    = 33 bytes
      IP: Identification  = 4083
      IP: Flags           = 0X
      IP:       .0.. .... = may fragment
      IP:       ..0. .... = last fragment
      IP: Fragment offset = 0 bytes
      IP: Time to live    = 216 seconds/hops
      IP: Protocol        = 1 (ICMP)
      IP: Header checksum = 4157 (correct)
      IP: Source address      = [100.100.100.100]
      IP: Destination address = [100.100.100.101]
      IP: No options
      IP:
ICMP: ----- ICMP header -----
      ICMP:
      ICMP: Type = 0 (Echo reply)
      ICMP: Code = 0
      ICMP: Checksum = 54EC (correct)
      ICMP: Identifier = 0
      ICMP: Sequence number = 54738
      ICMP: [5 bytes of data]
      ICMP:
      ICMP: [Normal end of "ICMP header".]
      ICMP:
ADDR  HEX                                             ASCII
0000: 00 a0 c9 ea 1a 3d 00 04 76 44 92 d4 08 00 45 00 | .....=..vD....E.
0010: 00 21 0f f3 00 00 d8 01 41 57 64 64 64 64 64 64 | .!......AWdddddd
0020: 64 65 00 00 54 ec 00 00 d5 d2 68 65 6c 6c 6f 00 | de..T.....hello.
0030: 00 00 00 00 00 00 00 00 00 00 00 00             | ............
```

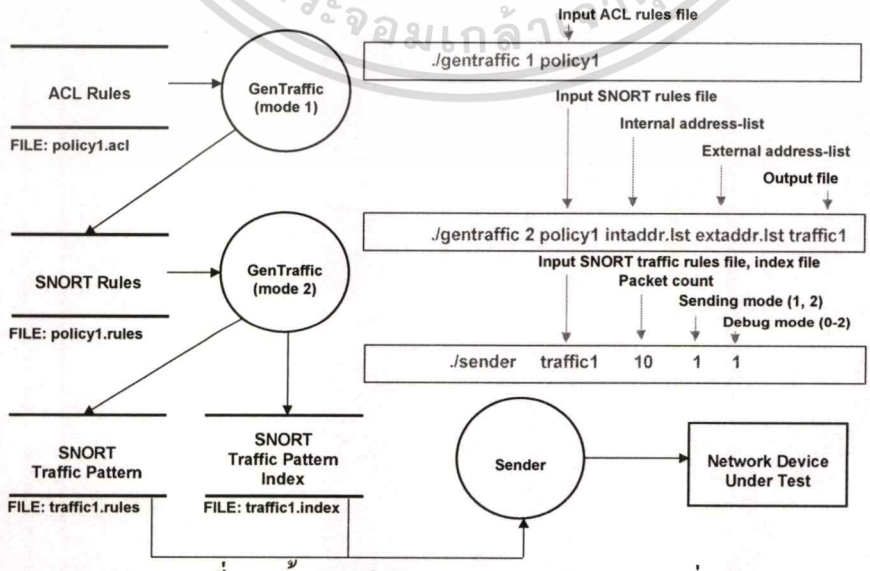
4.5.2 การทดสอบระบบทดสอบนโยบายความปลอดภัย

ในการทดสอบโปรแกรมในสภาพแวดล้อมการใช้งานจริงนี้ได้ทำการติดตั้งเราเตอร์และเครื่องคอมพิวเตอร์สำหรับการทดสอบโดยจำลองเครือข่ายภายนอกและเครือข่ายภายในภายใต้นโยบายความปลอดภัยที่ได้กำหนดขึ้น แผนผังการเชื่อมต่อของสิ่งต่างๆ ในระบบที่ทำการทดสอบนี้แสดงไว้ดังภาพที่ 4.2



ภาพที่ 4.2 สภาพแวดล้อมในการทดสอบโปรแกรมระบบ

ขั้นตอนการทดสอบนั้นจะเหมือนกันกับการใช้งานจริงทุกประการโดยประกอบไปด้วยลำดับขั้นในการทำงานตามภาพที่ 4.3



ภาพที่ 4.3 ขั้นตอนในการทำงานของระบบย่อยที่ 1

1. กำหนดนโยบายความปลอดภัยสำหรับระบบและอุปกรณ์ควบคุมเครือข่าย (เราเตอร์) ที่จะทำการทดสอบซึ่งอยู่ในรูปแบบของกฎแบบ ACL ในการทดสอบนี้จะกำหนดให้เป็นไฟล์ที่มีนามสกุลเป็น .acl (policy1.acl) ในส่วนของค่าติดตั้งสำหรับเราเตอร์นั้นเป็นดังนี้

```
!
interface Ethernet0/0
 description EXTERNAL NETWORK IP=172.1.191.254 255.255.255.0
 ip address 172.1.191.254 255.255.255.0
 ip access-group 101 in
 no ip redirects
 no ip route-cache
 no ip mroute-cache
 full-duplex
!
interface Ethernet1/0
 description INTERNAL NETWORK IP=160.1.1.254 255.255.255.0
 ip address 160.1.1.254 255.255.255.0
 ip access-group 102 out
 no ip redirects
 no ip unreachables
 no ip proxy-arp
 no ip route-cache
 no ip mroute-cache
 full-duplex
!
ip classless
ip route 0.0.0.0 0.0.0.0 Ethernet1/0
no ip http server
!
access-list 101 permit ip any any log
access-list 102 permit ip any any log
!
```

2. เรียกโปรแกรม gentraffic ในการทำงานแบบที่ 1 ซึ่งใช้ในการแปลงนโยบายความปลอดภัยแบบ ACL ให้เป็นแบบ SNORT โดยโปรแกรมจะสร้างไฟล์ผลลัพธ์นามสกุล .rules (policy1.rules)

```
[root@positron root]# more test2.acl
access-list 101 permit icmp host 172.1.191.3 host 160.1.1.1
access-list 101 permit tcp host 172.1.191.3 host 160.1.1.1 eq telnet
access-list 101 permit udp host 172.1.191.3 eq 2000 host 160.1.1.1
[root@positron root]# ./gentraffic 1 test2

GenTraffic: Compile ACL to SNORT rules
  Input ACL rule file    = test2.acl
  Output SNORT rule file = test2.rules
Compiling...
1: pass icmp 172.1.191.3/32 any -> 160.1.1.1/32 any
2: pass tcp 172.1.191.3/32 any -> 160.1.1.1/32 23
3: pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 any
Done.
```

3. เรียกโปรแกรม gentraffic ในการทำงานแบบที่ 2 ซึ่งจะสร้างรูปแบบการจราจรสำหรับการทดสอบจากนโยบายความปลอดภัยที่อยู่ในรูปแบบของ SNORT (traffic1.rules) และไฟล์ที่เก็บหมายเลข IP address ของเครือข่ายภายในและเครือข่ายภายนอก

```
[root@positron root]# more intaddr.lst
160.1.1.0/24
160.1.2.0/24
160.1.3.0/24
160.1.4.0/24
[root@positron root]# more extaddr.lst
172.1.0.0/16
172.2.0.0/16

[root@positron root]# ./gentraffic 2 test2 intaddr.lst extaddr.lst traffic2

GenTraffic: Generate Traffic rules from SNORT rules
  Input SNORT rule file  = test2.rules
  Internal address list  = intaddr.lst
  External address list  = extaddr.lst
  Output traffic file    = traffic2.rules
  Output traffic index   = traffic2.index
Generating traffic pattern ...
Done.

[root@positron root]# more traffic2.rules
pass icmp 172.1.191.3/32 63939 -> 160.1.1.1/32 52584
pass icmp 172.1.191.3/32 63939 -> 160.1.1.1/32 44892
pass icmp 172.1.191.3/32 63939 -> 160.1.1.1/32 63248
pass icmp 172.1.191.3/32 63939 -> 160.1.1.1/32 53614
pass icmp 172.1.191.3/32 34317 -> 160.1.1.1/32 39512
pass icmp 172.1.191.3/32 34317 -> 160.1.1.1/32 45914
pass icmp 172.1.191.3/32 34317 -> 160.1.1.1/32 17964
pass icmp 172.1.191.3/32 34317 -> 160.1.1.1/32 44822
pass tcp 172.1.191.3/32 33707 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 33707 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 33707 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 33707 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 6202 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 6202 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 6202 -> 160.1.1.1/32 23
pass tcp 172.1.191.3/32 6202 -> 160.1.1.1/32 23
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 13525
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 50
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 37414
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 44270
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 31459
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 9005
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 8004
pass udp 172.1.191.3/32 2000 -> 160.1.1.1/32 18223
```

4. ไฟล์ผลลัพธ์ที่ได้จากการสร้างในขั้นตอนที่ 3 จะถูกนำไปใช้โดยโปรแกรม sender ซึ่งจะสร้างแพคเกตตามรูปแบบการจราจรแล้วส่งออกไปยังระบบเครือข่ายสู่เราเตอร์เป้าหมาย โดยเครื่องคอมพิวเตอร์ที่ใช้ส่งแพตเกตสำหรับการทดสอบนี้จะต้องตั้ง default gateway เป็นเราเตอร์ที่จะทำการทดสอบด้วย

```
[root@positron root]# ./sender
Usage: sender <input file> <count> <sending mode> <debug mode>
  input file       = name of input file .rules and .index
  count            = packet counts, 0 for continue loop
  sending mode: 1 = random
                2 = sequential
  debug mode:   0 = off
                1 = all
                2 = error
[root@positron root]# ./sender traffic2 5 1 1
sending: packet# 0 index# 8
rawtransmit() src: 172.1.191.3
rawtransmit() dst: 160.1.1.1
rawtransmit() buf: 45002E00B7570000CB060000AC01BF03A001010183AB001700740BAA00000
000500200054CFAD000068656C6C6F21
sending: packet# 1 index# 10
rawtransmit() src: 172.1.191.3
rawtransmit() dst: 160.1.1.1
rawtransmit() buf: 45002E00A1B60000CC060000AC01BF03A001010183AB00170000000000000
000500120000DC0F000068656C6C6F21
sending: packet# 2 index# 14
rawtransmit() src: 172.1.191.3
rawtransmit() dst: 160.1.1.1
rawtransmit() buf: 45002E00B6A60000D9060000AC01BF03A0010101183A001700FAF1D400000
000501202D051E2000068656C6C6F21
sending: packet# 3 index# 22
rawtransmit() src: 172.1.191.3
rawtransmit() dst: 160.1.1.1
rawtransmit() buf: 4500220041E50000C9110000AC01BF03A001010107D01F44000E88C368656
C6C6F21
sending: packet# 4 index# 19
rawtransmit() src: 172.1.191.3
rawtransmit() dst: 160.1.1.1
rawtransmit() buf: 4500220086000000DF110000AC01BF03A001010107D0ACEE000EFB1868656
C6C6F21
```

5. ตรวจสอบแพคเกตที่ได้รับโดย Packet Capture (ใช้โปรแกรม NAI LAN Sniffer Pro 4.5, TCPDUMP) ทั้งในด้านของความถูกต้องในโครงสร้างของแพตเกต IP ตามโปรโตคอล นอกจากนี้ยังตรวจสอบ LOG บนตัวเราเตอร์ที่ทำการทดสอบด้วย

ภาพที่ 4.4 ผลลัพธ์ที่ได้จาก LAN Sniffer Pro 4.5 ที่ใช้ในการดักจับแพคเกตจากเครือข่ายภายใน

4.6 สรุปผลการทดสอบ

จากผลลัพธ์ที่ได้จากการทดสอบการทำงานของแต่ละฟังค์ชัน โปรแกรม และการทดสอบโดยรวมสามารถสรุปได้ดังนี้

- นโยบายความปลอดภัยในรูปแบบ SNORT ที่แปลงจาก ACL นั้นมีความถูกต้อง และเป็นไปตามที่ได้กำหนดไว้
- แพคเกตที่สร้างขึ้นตามนโยบายความปลอดภัยและรูปแบบการจราจรนั้นมีความถูกต้องตามที่ต้องการ
- แพคเกตที่ส่งผ่านเราเตอร์นั้นสามารถตรวจพบได้ทั้งจากใน LOG ของเราเตอร์และจากโปรแกรมที่ใช้ดักจับแพคเกต

บทที่ 5
สรุปการพัฒนาระบบงาน

5.1 ผลจากการพัฒนาระบบ

การพัฒนาซอฟต์แวร์ที่ช่วยในการทดสอบอุปกรณ์ควบคุมเครือข่ายนั้นสามารถสรุปได้ดังนี้

5.1.1 การศึกษาการทำงานของเราเตอร์ในการควบคุมเครือข่าย

ผลการศึกษาพบว่าเราเตอร์นั้นสามารถทำงานในการควบคุมเครือข่ายโดยการใช้การกรองแพคเกตได้เป็นอย่างดี แม้ว่าจะไม่สามารถตรวจสอบเงื่อนไขในระดับแอพพลิเคชันหรือในตัวเนื้อข้อมูลอย่างไพล์วอลได้ก็ตาม แต่เราเตอร์ก็สามารถกลั่นกรองแพคเกตได้ในระดับหนึ่งอย่างมีประสิทธิภาพ และมี overhead ไม่มากนัก เงื่อนไขในการตรวจสอบแพคเกตหรือ ACL นั้นเป็นรูปหนึ่งของการกำหนดนโยบายความปลอดภัย ซึ่งสามารถทำการทดสอบได้แต่ไม่สะดวกและใช้ทรัพยากรมาก และไม่มีเครื่องมือในการทดสอบอย่างเป็นระบบ

5.1.2 การวิเคราะห์และออกแบบ

ในส่วนของการออกแบบระบบทดสอบ ACL สำหรับเราเตอร์นี้ได้ออกแบบโดยอาศัยหลักการทำงานแบบกระจายแต่ผู้ทดสอบสามารถควบคุมการงานได้จากส่วนกลาง โดยจะมีโปรแกรม Traffic Generator Agent ทำงานอยู่ภายนอกสำหรับการสร้างและส่งภาระงานสู่ภายในเครือข่ายที่ทำการทดสอบซึ่งจะมี Traffic Analyzer Agent คอยรับแพคเกตที่ผ่านเข้ามาได้ Agent เหล่านี้รับคำสั่งจาก Management Station ส่วนการตรวจสอบประสิทธิภาพนั้นอาศัยการวิเคราะห์ Traffic Log, Syslog เทียบกับ Traffic Pattern และประมวลผลสร้างรายงานนำเสนอต่อผู้ทดสอบ

5.1.3 การพัฒนาระบบงาน

ได้ทำการพัฒนาโปรแกรมสำหรับการแปลงนโยบายความปลอดภัยที่สามารถสร้างรูปแบบการจราจรจากนโยบายความปลอดภัย และโปรแกรมสำหรับสร้างแพตเกตจากรูปแบบการจราจร เพื่อส่งไปทดสอบอุปกรณ์ควบคุมเครือข่าย โดยใช้ Packet Capture ที่มีอยู่แล้วเพื่อใช้ในการตรวจจับแพคเกตที่ส่งผ่านเข้ามาในเครือข่ายที่ควบคุม ในส่วนของการควบคุมการทำงานนั้นทำผ่านบรรทัดคำสั่ง

5.2 ประโยชน์ที่ได้รับ

ผลที่ได้รับโดยตรงจากพัฒนาระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์ควบคุมเครือข่ายนี้คือ ผู้ทดสอบมีเครื่องมือที่สามารถสร้างภาระงานที่เป็นรูปแบบการจราจรสำหรับการ

ทดสอบ ACL บนเราเตอร์ได้จากนโยบายความปลอดภัยหรือจาก ACL นั้นๆ โดยสามารถทำซ้ำ และกำหนดเวลาในการทดสอบได้โดยสามารถสร้างภาระงานจากเครื่องคอมพิวเตอร์หลายๆ เครื่องได้โดยอาศัยการทำงานร่วมกันแบบกระจาย ผลลัพธ์ที่ได้นั้นสามารถนำไปใช้ในการปรับปรุงหรือแก้ไขข้อบกพร่องของนโยบายความปลอดภัยได้ นอกจากนี้ยังสามารถนำไปใช้ทดสอบอุปกรณ์ควบคุมเครือข่ายแบบอื่นๆ อย่างไฟล์วอล หรือระบบตรวจสอบผู้รุกราน และการประเมินหาสมรรถนะของอุปกรณ์ควบคุมเครือข่ายก็ได้เช่นกัน

5.3 อุปสรรคและข้อจำกัดในการพัฒนาระบบ

5.3.1 การทำงานของโปรแกรมนี้ ออกแบบให้สามารถทำงานได้กับเครื่องคอมพิวเตอร์หลายเครื่อง แต่ในการพัฒนานี้มีเครื่อง Linux สำหรับการทดสอบเพียง 2 เครื่องเท่านั้น

5.3.2 สภาพแวดล้อมในการพัฒนา UNIX บนวินโดวส์ CYGWIN มีข้อผิดพลาดในการทำงานกับเครือข่ายระดับต่ำ โปรแกรมจึงไม่สามารถทำงานบนระบบปฏิบัติการวินโดวส์ได้โดยตรง ทำให้ต้องย้ายโปรแกรมไปพัฒนาบน Linux เพียงอย่างเดียว

5.3.3 เนื่องจากเป็นการยุ่งยากที่จะแยกช่องทางการสื่อสารในการควบคุม Agent ต่างออกจากระบบเครือข่ายที่ใช้ในการทดสอบอย่างแท้จริง ดังนั้นจึงต้องอาศัยความสามารถของเราเตอร์ในการกำหนดให้มีสองเครือข่าย IP อยู่บนเครือข่ายกายภาพเดียวกันแทน หรือการกำหนดให้ ACL นั้นปล่อยให้การติดต่อกับ Agent ผ่านไปได้

5.3.4 การติดต่อกับเครือข่ายในระดับต่ำผ่านทาง socket นั้นมีความยุ่งยากกว่าปกติ เนื่องจากต้องมีการจัดทำโครงสร้างของแพคเกตเองทั้งหมด ทำให้ต้องเสียเวลาไปอย่างมากในการเขียนฟังก์ชันคำนวณ checksum ของแต่ละโปรโตคอล ทั้งในการรับและการส่ง

5.3.5 การใช้ raw socket นั้นต้องใช้สิทธิพิเศษแบบ super user หรือ root เท่านั้น จึงไม่สามารถทำการทดสอบกับเครื่อง UNIX ของทางสถาบันฯ ได้

5.4 ข้อเสนอแนะ

5.4.1 ควรมีการจัดเก็บแฟ้มข้อมูลรูปแบบการจราจรสำหรับการทดสอบที่สร้างไว้ ในกรณีที่ต้องการทดสอบกับระบบเดิมอีกรอบหนึ่ง เนื่องจากในการสร้างรูปแบบการจราจรจากนโยบายความปลอดภัยในแต่ละครั้งจะได้รูปแบบไม่เหมือนกัน อันเป็นผลมาจากการสุ่มจากช่วงของระบบเครือข่ายที่กำหนดไว้

5.4.2 ควรมีการกำหนดชื่อแฟ้มข้อมูลต่างๆ ให้เป็นระบบ เพื่อป้องความสับสนในการนำไปใช้งาน เช่น กำหนดนามสกุล .acl สำหรับนโยบายความปลอดภัยแบบ ACL นามสกุล .rules สำหรับนโยบายความปลอดภัยแบบ SNORT เป็นต้น

5.4.3 เนื่องจากโปรแกรมสร้างการจราจรทดสอบนี้สามารถสร้างแพคเกต ในลักษณะที่อาจจะก่อให้เกิดความผิดพลาดในการทำงานของอุปกรณ์ควบคุมเครือข่ายได้ ดังนั้นจึงไม่ควรนำไปใช้ในสภาพแวดล้อมการทำงานจริงในช่วงที่มีภาระงานสำคัญในระบบ

5.4.4 เพื่อให้สามารถทดสอบได้ครบถ้วน จะต้องมีการกำหนดให้อุปกรณ์ควบคุมเครือข่ายที่จะทดสอบนั้นเป็น default gateway ของเครื่องที่โปรแกรม Traffic Generator ทำงานอยู่ด้วย แพคเกตที่สร้างขึ้นทั้งหมดจึงจะถูกส่งไปยังอุปกรณ์นั้น

บรรณานุกรม

ชินวัฒน์ ติวิธมไหศูรย์. 2543. "การวิเคราะห์เปรียบเทียบการอธิบายนโยบายความปลอดภัยโดยการใช้วิธีการแบบการกำหนดนโยบายและแบบการใช้กฎ." รายงานสัมมนา 1 วิทยาศาสตร์มหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

ชินวัฒน์ ติวิธมไหศูรย์. 2544. "การออกแบบระบบทดสอบนโยบายความปลอดภัยสำหรับอุปกรณ์เครือข่าย." รายงานสัมมนา 2 วิทยาศาสตร์มหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ บัณฑิตวิทยาลัย, สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง.

Cisco System Inc. 2000. **Cisco IOS Release 12.0 Security Configuration Guide.** [Online]. Available: http://www.cisco.com/univercd/cc/td/doc/product/software/ios120/12cgcr/secur_c/

Cisco System Inc. 2001. **Characterizing and Tracing Packet Floods Using Cisco Routers.** [Online]. Available: http://www.cisco.com/warp/public/707/22.html

Coulouris G., Dollimore J. and Kindberg T. 1994. **Distributed Systems Concept and Design.** 2nd edition. New York : Addison-Wesley

Condell, M., Lynn, C. and Zao, J. 1998. "**Security Policy Specification Language**", IETF Network Working Group IPSEC.

Gollmann, D. 1999. **Computer Security.** New York : John Wiley & Sons.

Jain, R. n.p. **The Art of Computer Systems Performance Analysis.** New York : John Wiley & Sons.

Olovsson, T. 1992. "**A Structured Approach to Computer Security**" Technical Report No 122. Sweden Chalmers : University of Technology.

Roesch, M. 1999. **Snort – Lightweigth Intrusion Detection for Networks.** [Online]. Seattle : Standford Telecommunications. Available : http://www.snort.org

Steven, W. R. 1999. **UNIX Network Programming Network Volume 1.** 2nd edition. New York : Addison-Wesley.

ภาคผนวก

ภาคผนวก ก.
วิธีการติดตั้งโปรแกรม

1. Unzip ต้นฉบับโปรแกรมและไลบรารีทั้งหมด จะมีไฟล์และโครงสร้างไดเรกทอรีเป็นดังนี้

```
ip.c
ip.h
ip.o
genlib.c
genlib.h
genlib.o
netlib.c
netlib.h
netlib.o
GenTraffic/gentraffic.c
GenTraffic/gentraffic
GenTraffic/makefile
Sender/sender.c
Sender/sender
Sender/makefile
```

2. สามารถเรียกใช้โปรแกรม ./GenTraffic/gentraffic หรือ ./Sender/sender ได้ถ้าใช้งานบนระบบปฏิบัติการ Linux บนสภาพแวดล้อม i386
3. ในกรณีที่จำเป็นต้อง compile และ link โปรแกรม gentraffic และ sender ใหม่เมื่อต้องการใช้โปรแกรมในสภาพแวดล้อมที่ต่างออกไป ให้ใช้คำสั่งดังนี้

```
cd GenTraffic
make clean
make gentraffic

cd ../Sender
make clean
make sender
```

4. หลังจากนี้ก็อาจจะสำเนาเอาเฉพาะโปรแกรมที่เรียกใช้งานได้ไปยังไดเรคทอรีปลายทางที่ต้องการเพื่อความสะดวกในการใช้งาน จะได้ไม่ต้องทำการเปลี่ยนไดเรคทอรีเพื่อเข้าไปเรียกใช้โปรแกรมอีก การสำเนาเพื่อนำไปใช้งานต้องการเฉพาะไฟล์ GenTraffic/gentraffic กับ Sender/sender เท่านั้น

ภาคผนวก ข.

วิธีการใช้งานโปรแกรม

1. การใช้งานโปรแกรม gentraffic

เมื่อเรียกโปรแกรม gentraffic โดยไม่ระบุพารามิเตอร์ โปรแกรมจะแสดงคำอธิบายการใช้งานและพารามิเตอร์ต่างๆ ขึ้นมาดังนี้

```
[root@positron root]# ./gentraffic
Usage  gentraffic 1 <input file>
       gentraffic 2 <input file> <in addr-list> <ext addr-list> <output file>
Mode:  1 = Complile input ACL to SNORT rules file
           Output file will have .rules extension
       2 = Generate traffic pattern/index file from SNORT rules file
           Output file will have .rules and .index extension
Note:  in addr-list  = Internal address list, required for mode 2
       ext addr-list = External address list, required for mode 2
```

การทำงานของโปรแกรมแยกออกเป็นสองส่วนคือ

1.1 แปลงนโยบายความปลอดภัยในรูปแบบ ACL ให้เป็น SNORT

ในการทำงานแบบนี้ จะต้องระบุโหมดการทำงานในพารามิเตอร์แรกเป็น 1 ส่วนพารามิเตอร์ถัดไปนั้นจะเป็นชื่อไฟล์ที่เก็บนโยบายความปลอดภัยในแบบ ACL การป้อนไม่ต้องระบุนามสกุล โปรแกรมจะเติมนามสกุล .acl ให้ ดังนั้นชื่อไฟล์จริงๆ จะต้องมีนามสกุลเป็น .acl เท่านั้น ไฟล์ผลลัพธ์ที่ได้นั้นจะมีชื่อเดียวกันกับไฟล์ input แต่จะมีนามสกุลเป็น .rules

1.2 สร้างรูปแบบการจราจรจากนโยบายความปลอดภัยในรูปแบบ SNORT

การทำงานในโหมดนี้พารามิเตอร์แรกจะต้องมีค่าเป็น 2 แล้วตามด้วยชื่อของไฟล์ที่เก็บนโยบายความปลอดภัยในรูปแบบ SNORT ที่ได้จาก 1.1 พารามิเตอร์ถัดมาจะเป็นชื่อของไฟล์ที่เก็บ IP address สำหรับเครือข่ายภายในและภายนอก และลงท้ายด้วยพารามิเตอร์ที่กำหนดชื่อไฟล์ผลลัพธ์ ซึ่งจะได้ออกมาเป็นสองไฟล์คือ .rules และ .index ทั้งสองไฟล์นี้จะเป็นไฟล์ input สำหรับโปรแกรม sender ที่ใช้ในการส่งการจราจรสำหรับทดสอบเราเตอร์

2. การใช้งานโปรแกรม sender

เมื่อเรียกโปรแกรม sender โดยไม่ระบุพารามิเตอร์ โปรแกรมจะแสดงคำอธิบายการใช้งานและพารามิเตอร์ต่างๆ ขึ้นมาดังนี้

```
[root@positron root]# ./sender
Usage: sender <input file> <count> <sending mode> <debug mode>
  input file       = name of input file .rules and .index
  count            = packet counts, 0 for continue loop
  sending mode: 1 = random
                2 = sequential
  debug mode:   0 = off
                1 = all
                2 = error
```

พารามิเตอร์ที่ต้องระบุมีดังนี้

- ชื่อของไฟล์ที่เก็บรูปแบบการจราจรในแบบ SNORT และไฟล์ index
- จำนวนของแพคเกตที่ต้องการส่ง ถ้ากำหนดเป็น 0 โปรแกรมก็จะส่งแพตเกตออกมาเรื่อยๆ จนกว่าจะยกเลิกการทำงานของโปรแกรมด้วยการกด Control-break
- รูปแบบของการส่งแพคเกตว่าจะเป็นการส่งแบบสุ่ม หรือส่งตามลำดับตามรูปแบบการจราจร
- แบบของการหาข้อผิดพลาด (Debug) ว่าต้องการให้แสดงข้อความระหว่างการทำงานของโปรแกรมมากน้อยแค่ใด ถ้าระบุเป็น 1 ก็จะแสดงข้อความทั้งหมดรวมทั้งข้อความที่เป็นสารสนเทศธรรมดา แต่ถ้าระบุเป็น 2 ก็จะแสดงเฉพาะข้อความผิดพลาดที่เกี่ยวข้องกับการทำงานเท่านั้น

ภาคผนวก ค.

Source Code ของโปรแกรม

โปรแกรมต้นฉบับ IP.H

```
#ifndef _IP_H
#define _IP_H
#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>

#ifndef INADDR_ANY
#define INADDR_ANY ((unsigned) 0x00000000)
#endif

#ifndef IP_HDRINCL
#define IP_HDRINCL 3
#endif
#ifndef PF_INET
#define PF_INET 2
#endif
#ifndef AF_INET
#define AF_INET PF_INET
#endif

typedef char s8;
typedef unsigned char u8;
typedef short int s16;
typedef unsigned short int u16;
typedef int s32;
typedef unsigned int u32;

#define ICMP_ECHOREPLY  0
#define ICMP_ECHO   8

#ifndef htons

#if __BYTE_ORDER == __BIG_ENDIAN
#define ntohl(x)        (x)
#define ntohs(x)        (x)
#define htonl(x)        (x)
#define htons(x)        (x)
#else
unsigned long int htonl (unsigned long int hostlong);
unsigned short int htons (unsigned short int
hostshort);
unsigned long int ntohl (unsigned long int netlong);
unsigned short int ntohs (unsigned short int
netshort);
#endif
#endif

#define IP 0
#define ICMP 1
#define IGMP 2
#define TCP 6
#define UDP 17
#define RAW 255

struct sa {
    u16 fam, dp;
    u32 add;
    u8 zero[8];
  };

struct su {
    u16 fam;
    char path[108];
  };

struct ip  {
#if __BYTE_ORDER == __LITTLE_ENDIAN
    u8 ihl:4, ver:4;
#else
    u8 ver:4, ihl:4;
#endif
    u8 tos;
    u16 tl, id, off;
    u8 ttl, pro;
    u16 sum;
    u32 src, dst;
  };

struct tcp  {
    u16 src, dst;
    u32 seq, ack;
#if __BYTE_ORDER == __LITTLE_ENDIAN
    u8 x2:4, off:4;
#else
    u8 off:4, x2:4;
#endif
    u8 flg;                                 /* flag1 |
flag2 */
#define FIN  0x01
#define SYN  0x02
#define RST  0x04
#define PUSH 0x08
#define ACK  0x10
#define URG  0x20
    u16 win, sum, urp;
  };
struct udp  {
    u16 src, dst, len, sum;
  };

struct icmp  {
    u8 type, code;
    u16 sum;
    u16 id, seq;
  };
char *inet_ntoa (struct in_addr);
// unsigned long int inet_addr (const char *cp);
u16 cksum (u16 *, int);
unsigned short ip_sum (unsigned short *, int);
int isip (char *);
unsigned long resolve (char *);
unsigned short udpchecksum(struct ip *ip, struct udp
*udp);
unsigned short tcpchecksum(struct ip *ip, u16 *,
int);
u16 tcp_sum_calc(struct ip *ip,int padding, u16
*buff,u16 len_tcp);

#endif
```

โปรแกรมต้นฉบับ IP.C

```
#include <string.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "ip.h"

unsigned long resolve (char *host)
{
  struct hostent *he;
  struct sa tmp;
  if (isip (host))
    return (inet_addr (host));
  he = gethostbyname (host);
  if (he) {
    memcpy ((caddr_t) & tmp.add, he->h_addr, he->
h_length);
    }
  else
    return (0);
  return (tmp.add);
}
int isip (char *ip)
{
  int a, b, c, d;
  sscanf (ip, "%d.%d.%d.%d", &a, &b, &c, &d);
  if (a < 0)
    return 0;
  if (a > 255)
    return 0;
  if (b < 0)
    return 0;
  if (b > 255)
    return 0;
  if (c < 0)
    return 0;
  if (c > 255)
    return 0;
  if (d < 0)
    return 0;
  if (d > 255)
    return 0;
  return 1;
}

u16 cksum (u16 * buf, int nwords)
{
  unsigned short i;
  int count;
  unsigned long sum;
  sum = 0;
  for (count = nwords >> 1; count > 0; count--) {
    sum += *buf++;
    }

  if (nwords%2 == 1) {
    i = *buf;
    sum += (i << 8);
    }

  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);

  return ~sum;
}

unsigned short ip_sum (addr, len)
  unsigned short *addr;
  int len;
{
  register int nleft = len;
  register unsigned short *w = addr;
  register int sum = 0;
  unsigned short answer = 0;

  while (nleft > 1) {
    sum += *w++;
    nleft -= 2;
    }
  if (nleft == 1) {
    *(unsigned char *) (&answer) = *(unsigned char *)
w;
    sum += answer;
    }
  sum = (sum >> 16) + (sum & 0xffff);
  sum += (sum >> 16);
  answer = ~sum;
  return (answer);
}

unsigned short udpchecksum(struct ip *ip, struct udp
*udp)
{
  unsigned long sum;
  u16 *s;
  int size;
  u32 addr;
  sum = 0;

  addr = ip->src;
  sum += addr >> 16;
  sum += addr & 0xffff;
  addr = ip->dst;
  sum += addr >> 16;
  sum += addr & 0xffff;
  sum += ip->pro << 8;   /* endian swap */
  size = udp->len;
  sum += size;

  size = ntohs(size);
  s = (u16 *)udp;
  while (size > 1) {
    sum += *s;
    s++;
    size -= 2;
    }
  if (size)
    sum += *(u8 *)s;

  sum  = (sum & 0xffff) + (sum >> 16);    /* add
overflow counts */
  sum  = (sum & 0xffff) + (sum >> 16);    /* once
again */

  return ~sum;
}

unsigned short tcpchecksum(struct ip *ip, u16 *tcp,
int tcplen)
{
  unsigned long sum;
  u16 *s, size;
  u32 addr;

  sum = 0;
  addr = ip->src;
  sum += htonl(addr);
```

```
addr = ip->dst;
sum += htonl(addr);
sum += ip->pro;
sum += tcplen;

s = tcp;
size = tcplen;
while (size > 1) {
  sum += htons(*s);
  s++;
  size -= 2;
  }
if (size) {
  sum += htons(*(u8 *)s);
  }

sum = htonl(sum);
sum = (sum >> 16) + (sum & 0xffff);
sum += (sum >> 16);

if (!(sum >> 16))
  return (~sum)-0x0100;
else
  return (~sum)-0x0001;
}
```

โปรแกรมต้นฉบับ GENLIB.H

```
#include <stdio.h>
#include <string.h>

#ifndef STRLEN_ADDRLIST
#define STRLEN_ADDRLIST 24
#endif

int filegetline(FILE *fp,char *buff, int len);
int filegettoken(FILE *fp, char *buff, int len);
int fileskipline(FILE *fp, int len);
long getfilesize(FILE *filepoint);
int extension_over(char *tempstr,char *filename,char *ext);

int token_isnumber(char *token);
int token_portlookup(char *token);
int token_isip(char *ip);
int token_wildcardtosubnet(char *ip, char *ipout);
int token_subnettonetbit(char *ip);

int rule_acl2snort(char *infile, char *outfile);

int traffic_port(char *szOut,const char *szPort);
int traffic_address(char *szOut, const char *szAddr, char *addrlist);
int traffic_gen(int mode, char *infile, char *idxfile, char *outfile, char *extaddr, char *intaddr);

int address_getfromfile(char *addrlist, char *infile, int maxline);
int address_print(char *addrlist);
```

โปรแกรมต้นฉบับ GENLIB.C

```
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "genlib.h"

#ifndef STRLEN_ADDRLIST
#define STRLEN_ADDRLIST 24
#endif

extern int gendebug;
char token_port[2048] = "ECHO 0 ECHO-REPLY 8
TRACEROUTE 1 CHARGEN 40 DOMAIN 3 UUCP,4 SUNRPC 5
TELNET 23 FTP-DATA 20 FTP 21 EXEC 59 SMTP 159 HTTP 80
LPD 505 .";

int filegetline(FILE *fp, char *buff, int len)
// Purpose: read text stream from the file line by
line, advance file pointer
// Parameter:
//   fp = File pointer of input file (text/read)
//   buff = pointer to output string
//   len  = size (maximum) of output string
// Return:
//   size of line (characters), return -1 if fail
{
  int count=0;
          char *cp,ctemp;
  if (fp == 0) {
            return -1;
          }
          cp = buff;
  ctemp = 0;
  while (!feof(fp) && count<len && ctemp!=10) {
            ctemp = getc(fp);
          *cp = ctemp;
                      cp++;
           count++;
                    }
                    *cp = 0;
  return count;
}
int fileskipline(FILE *fp,int len)
{
  int count=0;
          char ctemp;
  if (fp == 0) {
            return -1;
          }

  ctemp = 0;
  while (!feof(fp) && count<len && ctemp!=10) {
            ctemp = getc(fp);
    count++;
          }

  return count;
}
int filegettoken(FILE *fp, char *buff, int len)
// Purpose: read text stream from the file by token,
advance file pointer
// Parameter:
//   fp = File pointer of input file (text/read)
//   buff = pointer to output string
//   len  = size (maximum) of output string
// Return:
//   size of line (characters), return -1 if fail
{
  int count=0, state=0;
          char *cp,ctemp;
  if (fp == 0) {
            return -1;
```

```
        }
        cp = buff;
  ctemp = 0;

  while (state!=99 && !feof(fp)) {
    ctemp = getc(fp);
    switch (state) {
      case 0:
        if (ctemp==' ' || ctemp==9)
          state = 0;
        else if (ctemp == 10) {
          *cp = ctemp;
          cp++;
          count++;
          *cp = 0;
          state = 99;
          }
        else {
          state = 1;
          *cp = toupper(ctemp);
          cp++;
          count++;
          }
        break;
      case 1:
        if (ctemp==' ' || ctemp==9) {
          *cp = 0;
          state = 99;
          }
        else if (ctemp==10) {
          *cp = 0;
          state = 99;
          fseek(fp, -1, SEEK_CUR);
          }
        else {
          *cp = toupper(ctemp);
          cp++;
          count++;
          }
        break;
      }
    }

  return count;
}
int token_isnumber(char *token)
{
  char *ptoken;
  int isNumber = 1;

  ptoken = token;
  while (*ptoken!=0 && isNumber==1) {
    if (!isdigit(*ptoken))
      isNumber = 0;
    ptoken++;
    }
  return isNumber;
}
int token_portlookup(char *token)
{
  char *ptoken, *ptoken2, portnum[8], temptoken[255];

  strcpy(temptoken,token);
  strcat(temptoken," ");

  ptoken = strstr(token_port, temptoken);
  if (ptoken <= 0)
    return -1;

  ptoken2 = ptoken;

  while (*ptoken2!=' ') ptoken2++;
  if (*ptoken2==' ') ptoken2++;
  ptoken = ptoken2;

  while (*ptoken2!=' ') ptoken2++;
  memcpy(portnum,ptoken,ptoken2-ptoken);
  portnum[ptoken2-ptoken] = 0;

  return atoi(portnum);
}
int token_isip (char *ip)
{
  int a, b, c, d;
  sscanf (ip, "%d.%d.%d.%d", &a, &b, &c, &d);
  if (a < 0)
    return 0;
  if (a > 255)
    return 0;
  if (b < 0)
    return 0;
  if (b > 255)
    return 0;
  if (c < 0)
    return 0;
  if (c > 255)
    return 0;
  if (d < 0)
    return 0;
  if (d > 255)
    return 0;
  return 1;
}
int token_wildcardtosubnet(char *ip, char *ipout)
{
  int a[4], b[4], count, i;

  sscanf (ip, "%d.%d.%d.%d", &a[0], &a[1], &a[2], &a
[3]);
  count = 0;
  do {
    b[count] = 255-a[count];
    count++;
    } while (a[count-1]==0 && count < 4);
  i = 4-count;
  while (i > 0) {
    b[count] = 0;
    count++;
    i--;
    }
  sprintf (ipout, "%d.%d.%d.%d", b[0],b[1],b[2],b
[3]);
  return 1;
}
int token_subnettonetbit(char *ip)
{
  int a[4], netbits = 0, i;
  sscanf (ip, "%d.%d.%d.%d", &a[0], &a[1], &a[2], &a
[3]);

  for (i=0;i<4;i++) {
    switch (a[i]) {
      case 255: netbits+=8; break;
      case 254: netbits+=7; break;
      case 252: netbits+=6; break;
      case 248: netbits+=5; break;
      case 240: netbits+=4; break;
      case 224: netbits+=3; break;
      case 192: netbits+=2; break;
      case 128: netbits+=1; break;
      }
    }
  return netbits;
}
```

```
int rule_acl2snort(char *infile, char *outfile)
// Purpose: compile Cisco ACL rule to SNORT rule
// Parameter:
//   infile = name of input file
//   outfile = name of output file
// Return:
//   Status (1 = success, <0 = error)

{
  FILE *infp, *outfp;
  char token[255], buff[1024], *pbuff=0, strtemp[80],
action;
  long linecount = 1;
  int state=0, tokenlen=0, newtoken=1, netbits=32,
netport=0, protocol=0;
  infp = fopen(infile, "rt");
  outfp = fopen(outfile, "wt");

  if (!infp) {
    printf("Error: rule_acl2snort() can't open input
file for read\n");
    return -1;
    }
  if (!outfp) {
    printf("Error: rule_acl2snort() can't create
output file for write\n");
    return -2;
    }

  do {
    if (newtoken) {
      tokenlen = filegettoken(infp,token,255);
      }
    newtoken = 1;
    if (tokenlen==0)
      state = 99;
    switch (state) {
      case 0:
        pbuff = buff;
        protocol = 0;
        action = 0;
        if (strcmp(token,"ACCESS-LIST")==0)
          state = 1;
        else if (strchr(token,10))
          linecount++;
        break;
      case 1:
        if (token_isnumber(token))
          state = 2;
        else {
          newtoken = 0;
          state = 101;
          }
        break;
      case 2:
        if (strcmp(token,"DENY")==0) {
          strcpy(buff,"log ");
          pbuff+=4;
          state = 3;
          action = 0;
          }
        else if (strcmp(token,"PERMIT")==0) {
          strcpy(buff,"pass ");
          pbuff+=5;
          state = 3;
          action = 1;
          }
        break;
      case 3:
        if (strcmp(token,"TCP")==0) {
          strcpy(pbuff,"tcp ");
          pbuff+=4;
          state = 30;
          }
        else if (strcmp(token,"UDP")==0) {
          strcpy(pbuff,"udp ");
          pbuff+=4;
          state = 30;
          }
        else if (strcmp(token,"ICMP")==0) {
          strcpy(pbuff,"icmp ");
          pbuff+=5;
          protocol = 1;
          state = 30;
          }
        else if (strcmp(token,"IP")==0) {
          strcpy(pbuff,"ip ");
          pbuff+=3;
          protocol = 1;
          state = 30;
          }
        else {
          newtoken = 0;
          state = 102;
          }
        break;
      case 30:
        if (strcmp(token,"ANY")==0) {
          strcpy(pbuff,"any ");
          pbuff+=4;
          state = 31;
          }
        else if (strcmp(token,"HOST")==0) {
          state = 32;
          }
        else if (token_isip(token)) {
          strcpy(pbuff,token);
          pbuff+=tokenlen;
          state = 33;
          }
        else {
          newtoken = 0;
          state = 103;
          }
        break;
      case 31:
        if (protocol!=1)
          state = 34;
        else
          state = 36;
        newtoken = 0;
        break;
      case 32:
        if (token_isip(token)) {
          strcpy(pbuff,token);
          pbuff+=tokenlen;
          strcpy(pbuff,"/32 ");
          pbuff+=4;
          state = 34;
          }
        else {
          newtoken = 0;
          state = 103;
          }
        break;
      case 33:
        if (token_isip(token)) {
          token_wildcardtosubnet(token,strtemp);
          netbits = token_subnettonetbit(strtemp);
          strcpy(pbuff,"/");
          pbuff++;
          sprintf(strtemp,"%d ",netbits);
          strcpy(pbuff,strtemp);
          pbuff+=strlen(strtemp);
```

```
      state = 34;
      }
    else {
      newtoken = 0;
      state = 103;
      }
    break;
  case 34:
    if (strcmp(token,"EQ")==0)
      state = 36;
    else if (strcmp(token,"RANGE")==0)
      state = 35;
    else {
      strcpy(pbuff,"any -> ");
      pbuff+=7;
      newtoken = 0;
      state = 40;
      }
    break;
  case 35:
    if (token_isnumber(token)==0) {
      netport = token_portlookup(token);
      if (netport < 0) {
        newtoken = 0;
        state = 103;
        }
      else {
        sprintf(strtemp,"%d:",netport);
        strcpy(pbuff,strtemp);
        pbuff+=strlen(strtemp);
        state = 37;
        }
      }
    else {
      strcpy(pbuff, token);
      pbuff+=tokenlen;
      strcpy(pbuff, ":");
      pbuff+=1;
      }
    state = 37;
    break;
  case 37:
    if (token_isnumber(token)==0) {
      netport = token_portlookup(token);
      if (netport < 0) {
        newtoken = 0;
        state = 103;
        }
      else {
        sprintf(strtemp,"%d -> ",netport);
        strcpy(pbuff,strtemp);
        pbuff+=strlen(strtemp);
        state = 40;
        }
      }
    else {
      strcpy(pbuff, token);
      pbuff+=tokenlen;
      strcpy(pbuff, " -> ");
      pbuff+=4;
      }
    state = 40;
    break;
  case 36:
    if (token_isnumber(token)==0) {
      netport = token_portlookup(token);
      if (netport>0) {
        sprintf(strtemp, "%d -> ",netport);
        strcpy(pbuff, strtemp);
        pbuff+=strlen(strtemp);
        }
      else {
        strcpy(pbuff,"any -> ");
        pbuff+=7;
        }
      if (protocol==1)
        newtoken = 0;
      }
    else {
      strcpy(pbuff, token);
      pbuff+=tokenlen;
      strcpy(pbuff, " -> ");
      pbuff+=4;
      }
    state = 40;
    break;
  case 40:
    if (strcmp(token,"ANY")==0) {
      strcpy(pbuff,"any ");
      pbuff+=4;
      state = 41;
      }
    else if (strcmp(token,"HOST")==0) {
      state = 42;
      }
    else if (token_isip(token)) {
      strcpy(pbuff,token);
      pbuff+=tokenlen;
      state = 43;
      }
    else {
      newtoken = 0;
      state = 103;
      }
    break;
  case 41:
    if (protocol!=1)
      state = 44;
    else
      state = 46;
    newtoken = 0;
    break;
  case 42:
    if (token_isip(token)) {
      strcpy(pbuff,token);
      pbuff+=tokenlen;
      strcpy(pbuff,"/32 ");
      pbuff+=4;
      state = 44;
      }
    else {
      newtoken = 0;
      state = 103;
      }
    break;
  case 43:
    if (token_isip(token)) {
      token_wildcardtosubnet(token,strtemp);
      netbits = token_subnettonetbit(strtemp);
      strcpy(pbuff,"/");
      pbuff++;
      sprintf(strtemp,"%d ",netbits);
      strcpy(pbuff,strtemp);
      pbuff+=strlen(strtemp);
      state = 44;
      }
    else {
      newtoken = 0;
      state = 103;
      }
    break;
  case 44:
    if (strcmp(token,"EQ")==0)
      state = 46;
```

```
    else if (strcmp(token,"RANGE")==0)
      state = 45;
    else {
      strcpy(pbuff,"any ");
      pbuff+=4;
      *pbuff = 0;
      newtoken = 0;
      state = 50;
      }
    break;
  case 45:
    if (token_isnumber(token)==0) {
      netport = token_portlookup(token);
      if (netport < 0) {
        newtoken = 0;
        state = 103;
        }
      else {
        sprintf(strtemp,"%d:",netport);
        strcpy(pbuff,strtemp);
        pbuff+=strlen(strtemp);
        state = 47;
        }
      }
    else {
      strcpy(pbuff, token);
      pbuff+=tokenlen;
      strcpy(pbuff, ":");
      pbuff+=1;
      }
    state = 47;
    break;
  case 47:
    if (token_isnumber(token)==0) {
      netport = token_portlookup(token);
      if (netport < 0) {
        newtoken = 0;
        state = 103;
        }
      else {
        sprintf(strtemp,"%d ",netport);
        strcpy(pbuff,strtemp);
        pbuff+=strlen(strtemp);
        state = 50;
        }
      }
    else {
      strcpy(pbuff, token);
      pbuff+=tokenlen;
      strcpy(pbuff, ":");
      pbuff+=1;
      }
    state = 50;
    break;
  case 46:
    if (token_isnumber(token)==0) {
      netport = token_portlookup(token);
      if (netport>0) {
        sprintf(strtemp, "%d ", netport);
        strcpy(pbuff, strtemp);
        pbuff+=strlen(strtemp);
        }
      else {
        strcpy(pbuff,"any ");
        pbuff+=4;
        }
      if (protocol==1)
        newtoken = 0;
      }
    else {
      strcpy(pbuff, token);
      pbuff+=tokenlen;
      strcpy(pbuff, " ");
      pbuff+=1;
      }
    *pbuff = 0;
    state = 50;
    break;
  case 50:
    if (gendebug) {
      printf("%ld: %s\n",linecount, buff);
      }
    fprintf(outfp,"%s\n", buff);
    linecount++;
    while (strchr(token,10)==0) {
      tokenlen = filegettoken(infp,token,255);
      };
    state = 0;
    break;
  case 101:
    if (gendebug) {
      printf("Warning: rule_acl2snort() rule
error, skip line#%ld\n",linecount);
      }
    fileskipline(infp,1024);
    linecount++;
    buff[0] = 0;
    pbuff = buff;
    state = 0;
    break;
  case 102:
    if (gendebug) {
      printf("Warning: rule_acl2snort() unsupport
protocol, skip line#%ld\n",linecount);
      }
    fileskipline(infp,1024);
    linecount++;
    buff[0] = 0;
    pbuff = buff;
    state = 0;
    break;
  case 103:
    if (gendebug) {
      printf("Warning: rule_acl2snort() invalid
source/destination format, skip
line#%ld\n",linecount);
      }
    fileskipline(infp,1024);
    linecount++;
    buff[0] = 0;
    pbuff = buff;
    state = 0;
    break;
  default:
    break;
  }

 } while (state!=99);
fclose(infp);
fclose(outfp);
return linecount;
}
int traffic_port(char *szOut,const char *szPort)
{
  char *pPort;
  int min,max,c;

  strcpy(szOut,szPort);
  pPort = strchr(szOut,':');
  if (pPort!=0) {
    *pPort = 0;
    min = atoi(szOut);
    max = atoi(pPort+1);
```

```
    c = ((rand() % (int) (((max) + 1) - (min))) +
(min));
    sprintf(szOut,"%d",c);
    }
  else if (strcmp(szOut,"any")==0) {
    c = (rand() % 65535) + 1;
    sprintf(szOut,"%d",c);
    }
  return 0;
}
int traffic_address(char *szOut, const char *szAddr,
char *addrlist)
{
  char *pBits;
  int netbits,a[4],b[4],c[4],i,min,max;

  strcpy(szOut,szAddr);
  pBits = strchr(szOut,'/');
  if (pBits==0) {
    if (addrlist==NULL) {
      min = 1;
      max = 223;
      c[0] =  ((rand() % (int) (((max) + 1) - (min)))
+ (min));
      max = 255;
      for (i=1;i<4;i++)
        c[i] =  ( (rand() % (int) (((max) + 1) -
(min))) + (min) );
      }
    else {
      max = atoi(addrlist);
      i = ((rand() % (int) (((max) + 1) - (1))) +
(1));
      pBits =
strchr(addrlist+(i*STRLEN_ADDRLIST),'/');
      if (pBits==0)
        netbits = 32;
      else {
        pBits++;
        netbits = atoi(pBits);
        }

sscanf(addrlist+(i*STRLEN_ADDRLIST),"%d.%d.%d.%d",&a
[0],&a[1],&a[2],&a[3]);
        switch (netbits) {
          case 0 : b[0] = 255; b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 1 : b[0] = 127; b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 2 : b[0] = 63;  b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 3 : b[0] = 31;  b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 4 : b[0] = 15;  b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 5 : b[0] = 7;   b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 6 : b[0] = 3;   b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 7 : b[0] = 1;   b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 8 : b[0] = 0;   b[1] = 255; b[2] =
255; b[3] = 255; break;
          case 9 : b[0] = 0;   b[1] = 127; b[2] =
255; b[3] = 255; break;
          case 10: b[0] = 0;   b[1] = 63;  b[2] =
255; b[3] = 255; break;
          case 11: b[0] = 0;   b[1] = 31;  b[2] =
255; b[3] = 255; break;
          case 12: b[0] = 0;   b[1] = 15;  b[2] =
255; b[3] = 255; break;
          case 13: b[0] = 0;   b[1] = 7;   b[2] =
255; b[3] = 255; break;
          case 14: b[0] = 0;   b[1] = 3;   b[2] =
255; b[3] = 255; break;
          case 15: b[0] = 0;   b[1] = 1;   b[2] =
255; b[3] = 255; break;
          case 16: b[0] = 0;   b[1] = 0;   b[2] =
255; b[3] = 255; break;
          case 17: b[0] = 0;   b[1] = 0;   b[2] =
127; b[3] = 255; break;
          case 18: b[0] = 0;   b[1] = 0;   b[2] = 63;
b[3] = 255; break;
          case 19: b[0] = 0;   b[1] = 0;   b[2] = 31;
b[3] = 255; break;
          case 20: b[0] = 0;   b[1] = 0;   b[2] = 15;
b[3] = 255; break;
          case 21: b[0] = 0;   b[1] = 0;   b[2] = 7;
b[3] = 255; break;
          case 22: b[0] = 0;   b[1] = 0;   b[2] = 3;
b[3] = 255; break;
          case 23: b[0] = 0;   b[1] = 0;   b[2] = 1;
b[3] = 255; break;
          case 24: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 255; break;
          case 25: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 127; break;
          case 26: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 63;  break;
          case 27: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 31;  break;
          case 28: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 15;  break;
          case 29: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 7;   break;
          case 30: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 3;   break;
          case 31: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 1;   break;
          case 32: b[0] = 0;   b[1] = 0;   b[2] = 0;
b[3] = 0;   break;
          }
        for (i=0;i<4;i++) {
          if (b[i]==0) {
            c[i] = a[i];
            }
          else if (b[i]==255) {
            c[i] = ((rand() % (int) (((254) + 1) -
(1))) + (1));
            }
          else {
            min = a[i] % b[i];
            if (min == 0) {
              min = a[i];
              }
            else
              min = a[i] - min;
            max = min + b[i];
            c[i] = ((rand() % (int) (((max) + 1) -
(min))) + (min));
            }
          }

      }
    }
  else {
    pBits++;
    netbits = atoi(pBits);
    if (netbits==32)
      return 0;

    sscanf(szOut,"%d.%d.%d.%d",&a[0],&a[1],&a[2],&a
[3]);
```

```
    switch (netbits) {
      case 0 : b[0] = 255; b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 1 : b[0] = 127; b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 2 : b[0] = 63;  b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 3 : b[0] = 31;  b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 4 : b[0] = 15;  b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 5 : b[0] = 7;   b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 6 : b[0] = 3;   b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 7 : b[0] = 1;   b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 8 : b[0] = 0;   b[1] = 255; b[2] = 255; b
[3] = 255; break;
      case 9 : b[0] = 0;   b[1] = 127; b[2] = 255; b
[3] = 255; break;
      case 10: b[0] = 0;   b[1] = 63;  b[2] = 255; b
[3] = 255; break;
      case 11: b[0] = 0;   b[1] = 31;  b[2] = 255; b
[3] = 255; break;
      case 12: b[0] = 0;   b[1] = 15;  b[2] = 255; b
[3] = 255; break;
      case 13: b[0] = 0;   b[1] = 7;   b[2] = 255; b
[3] = 255; break;
      case 14: b[0] = 0;   b[1] = 3;   b[2] = 255; b
[3] = 255; break;
      case 15: b[0] = 0;   b[1] = 1;   b[2] = 255; b
[3] = 255; break;
      case 16: b[0] = 0;   b[1] = 0;   b[2] = 255; b
[3] = 255; break;
      case 17: b[0] = 0;   b[1] = 0;   b[2] = 127; b
[3] = 255; break;
      case 18: b[0] = 0;   b[1] = 0;   b[2] = 63;  b
[3] = 255; break;
      case 19: b[0] = 0;   b[1] = 0;   b[2] = 31;  b
[3] = 255; break;
      case 20: b[0] = 0;   b[1] = 0;   b[2] = 15;  b
[3] = 255; break;
      case 21: b[0] = 0;   b[1] = 0;   b[2] = 7;   b
[3] = 255; break;
      case 22: b[0] = 0;   b[1] = 0;   b[2] = 3;   b
[3] = 255; break;
      case 23: b[0] = 0;   b[1] = 0;   b[2] = 1;   b
[3] = 255; break;
      case 24: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 255; break;
      case 25: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 127; break;
      case 26: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 63;  break;
      case 27: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 31;  break;
      case 28: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 15;  break;
      case 29: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 7;   break;
      case 30: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 3;   break;
      case 31: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 1;   break;
      case 32: b[0] = 0;   b[1] = 0;   b[2] = 0;   b
[3] = 0;   break;
      }

    for (i=0;i<4;i++) {
      if (b[i]==0) {
        c[i] = a[i];
        }
      else if (b[i]==255) {
        c[i] = ((rand() % (int) (((254) + 1) - (1)))
+ (1));
        }
      else {
        min = a[i] % b[i];
        if (min == 0) {
          min = a[i];
          }
        else
          min = a[i] - min;
        max = min + b[i];
        c[i] = ((rand() % (int) (((max) + 1) -
(min))) + (min));
        }
      }
    }
    sprintf(szOut,"%d.%d.%d.%d/32",c[0],c[1],c[2],c
[3]);
    return 0;
}
int traffic_gen(int mode, char *infile, char
*idxfile, char *outfile, char *extaddr, char
*intaddr)
{
    FILE *infp, *outfp, *idxfp;
    char szOpt[5], szPro[5], szSrc[50], szSrcport[30],
szDst[50], szDstport[3],
      linebuff[1024], szTemp1[50],szTemp2[50],szTemp3
[50],szTemp4[50];
    int len,i,j,ei,ej;
    long bytecount = 0, temp1;

    infp = fopen(infile, "rt");
    outfp = fopen(outfile, "wt");
    idxfp = fopen(idxfile, "wb");

    if (!infp) {
      printf("Error: traffic_gen() can't open input
file for read\n");
      return -1;
      }
    if (!outfp) {
      printf("Error: traffic_gen() can't create output
file for write\n");
      return -2;
      }
    if (!idxfp) {
      printf("Error: traffic_gen() can't create index
file for write\n");
      return -3;
      }

    switch (mode) {
      case 1:
        while (!feof(infp)) {
          len=filegetline(infp, linebuff, 1024);
          if (len>1) {
                                        sscanf(linebuff,"%s
%s %s %s -> %s %s",
szOpt,szPro,szSrc,szSrcport,szDst,szDstport);
            traffic_address(szTemp1,szSrc,NULL);
            traffic_port(szTemp2,szSrcport);
            traffic_address(szTemp3,szDst,NULL);
            traffic_port(szTemp4,szDstport);
            fprintf(outfp,"%s %s %s %s -> %s %s\n",
szOpt,szPro,szTemp1,szTemp2,szTemp3,szTemp4);
          }
        }
        break;
```

```
    case 2:
      while (!feof(infp)) {
        len=filegetline(infp, linebuff, 1024);
        if (len>1) {
                                    sscanf(linebuff,"%s
%s %s %s -> %s %s",
szOpt,szPro,szSrc,szSrcport,szDst,szDstport);
          if (extaddr==NULL)
            ei = 1;
          else
            ei = atoi(extaddr);
          if (intaddr==NULL)
            ej = 1;
          else
            ej = atoi(intaddr);

          for (i=0;i<ei;i++) {
            traffic_address(szTemp1,szSrc,extaddr);
            traffic_port(szTemp2,szSrcport);
            for (j=0;j<ej;j++) {
              traffic_address(szTemp3,szDst,intaddr);
              traffic_port(szTemp4,szDstport);
              sprintf(linebuff,"%s %s %s %s -> %s
%s\n", szOpt,szPro,szTemp1,szTemp2,szTemp3,szTemp4);
              fprintf(outfp,"%s",linebuff);
              templ=htonl(bytecount);
              fwrite(&templ,sizeof(long),1,idxfp);
              bytecount+=strlen(linebuff);
              }
            }
          }
        }
      break;
    }
  fclose(infp);
  fclose(outfp);
  fclose(idxfp);
  return 0;
}
int address_getfromfile(char *addrlist, char *infile,
int maxline)
{
  FILE *infp;
  int len,a[4],i=1,isip=1;
  char buff[255];
  infp = fopen(infile, "rt");

  if (!infp) {
    printf("Error: address_getfromfile() can't open
input file %s for read\n",infile);
    return -1;
    }

  while (!feof(infp) && i<=maxline) {
    len=filegetline(infp, buff, 255);
    buff[len-1]=0;
    isip = 1;
    if (len>1) {
      sscanf(buff,"%d.%d.%d.%d",&a[0],&a[1],&a[2],&a
[3]);
      if (a[0]>255 || a[0]<0)
        isip = 0;
      if (a[1]>255 || a[1]<0)
        isip = 0;
      if (a[2]>255 || a[2]<0)
        isip = 0;
      if (a[3]>255 || a[3]<0)
        isip = 0;
      if (isip==1) {
        strcpy(addrlist+(i*STRLEN_ADDRLIST),buff);
        i++;
        }
      }
    }
  sprintf(addrlist,"%d",i-1);
  fclose(infp);
  return i;
}
int address_print(char *addrlist)
{
  int i,ei;

  ei = atoi(addrlist)+1;
  if (addrlist!=NULL) {
    for (i=1;i<ei;i++) {
      printf("%s\n",addrlist+(i*STRLEN_ADDRLIST));
      }
    return 0;
    }
  else {
    return -1;
  }
}
long getfilesize(FILE *filepoint)
{
  fpos_t curpos;
  long lsize;

  fgetpos(filepoint,&curpos);
  fseek(filepoint,0,SEEK_END);
  lsize = ftell(filepoint);
  fsetpos(filepoint,&curpos);
  return lsize;
}
int extension_over(char *tempstr,char *filename,char
*ext)
{
  char *dotpoint;
  int dotposit,namesize;
  dotpoint=strchr(filename,'.');
  if (dotpoint) {
    dotposit=dotpoint-filename;
    memcpy(tempstr,filename,dotposit);
    tempstr[dotposit]='.';
    strcpy(tempstr+dotposit+1,ext);
    }
  else {
    namesize=strlen(filename);
    strcpy(tempstr,filename);
    tempstr[namesize]='.';
    strcpy(tempstr+namesize+1,ext);
    }
  return strlen(tempstr);
}
```

โปรแกรมต้นฉบับ NETLIB.H

```
#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "ip.h"

#define BS 4096

void printhex(unsigned char *ucIn, unsigned int uiLen);
int rawtransmit(unsigned long srcaddr, unsigned int srcport, unsigned long dstaddr,
  unsigned int dstport, int proto, char *srcdata, unsigned int datalen);
int str2proto(char *proto);
char *str2upper(char *in);
```

โปรแกรมต้นฉบับ NETLIB.C

```
#include <stdio.h>
#include <ctype.h>
#include <unistd.h>
#include <fcntl.h>
#include <signal.h>
#include <string.h>
#include <stdlib.h>
#include <sys/time.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <netdb.h>
#include <errno.h>
#include <netinet/in.h>
#include <arpa/inet.h>
#include "ip.h"
#include "netlib.h"

int rcounter = 0;
extern int debug;
char rseed[65535];

void random_init (void);
inline long getrandom (int min, int max);

int rawtransmit(unsigned long srcaddr, unsigned int
srcport, unsigned long dstaddr,
  unsigned int dstport, int proto, char *srcdata,
unsigned int datalen)
{
  struct sockaddr_in dst_addr, src_addr;
  char buf[BS], *p;
  struct ip *ih = (struct ip *) buf;
  struct icmp *ich = (struct icmp *) (buf + sizeof
(struct ip));
  struct udp *udh = (struct udp *) (buf + sizeof
(struct ip));
  struct tcp *tch = (struct tcp *) (buf + sizeof
(struct ip));
  int tot_len = sizeof (struct ip), ssock=-1, i=1,
error=0;

  if ( ( tot_len+sizeof(struct tcp)+datalen ) > BS )
  {
    if (debug) {
      printf("Error: rawtransmit() data larger than
buffer size\n");
    }
    error = 0x08;
    return (error);
  }
  src_addr.sin_family = AF_INET;        // host byte
order
  memcpy(&(src_addr.sin_addr),&srcaddr,4);
  memset(&(src_addr.sin_zero),'\0', 8); // zero the
rest of the struct

  dst_addr.sin_family = AF_INET;        // host byte
order
  memcpy(&(dst_addr.sin_addr),&dstaddr,4);
  memset(&(dst_addr.sin_zero),'\0', 8); // zero the
rest of the struct

  memset (buf, 0, BS);

  ih->ver = 4;
  ih->ihl = 5;
  ih->tos = 0x00;
  ih->tl = 0;
  ih->id = htons (getrandom (1024, 65535));
  ih->off = 0;
  ih->ttl = getrandom (200, 255);
  ih->sum = 0;
  ih->src = src_addr.sin_addr.s_addr;
  ih->dst = dst_addr.sin_addr.s_addr;

  if (debug) {
    printf("rawtransmit() src: %s\n",inet_ntoa
(src_addr.sin_addr));
    printf("rawtransmit() dst: %s\n",inet_ntoa
(dst_addr.sin_addr));
    }
```

```
  switch (proto) {
  case ICMP:
    tot_len += sizeof (struct icmp);
    ih->pro = ICMP;
    ssock = socket (AF_INET, SOCK_RAW, ICMP);
    ich->type = (unsigned short)srcport;
    ich->code = (unsigned short)dstport;
    ich->id = getrandom (0, 1) ? getrandom (0, 65535)
: 0;
    ich->seq = getrandom (0, 1) ? getrandom (0,
65535) : 0;
    ich->sum = 0;

    p = buf + tot_len;
    memcpy(p, srcdata, datalen);
    tot_len += datalen;

    ich->sum = cksum ((u16 *) ich, tot_len - sizeof
(struct ip));
    ih->tl = tot_len;
    dst_addr.sin_port = htons(1024);
    break;
  case UDP:
    tot_len += sizeof (struct udp);
    ih->pro = UDP;
    ssock = socket (AF_INET, SOCK_RAW, UDP);

    udh->src = htons(srcport);
    udh->dst = htons(dstport);
    udh->sum = 0;

    p = buf + tot_len;
    memcpy(p, srcdata, datalen);
    tot_len += datalen;

    udh->len = htons (tot_len - sizeof(struct ip));
    udh->sum = udpchecksum(ih, udh);
    ih->tl = tot_len;
    dst_addr.sin_port = htons(1024);
    break;
  case TCP:
    tot_len += sizeof (struct tcp);
    ih->pro = TCP;
    ssock = socket (AF_INET, SOCK_RAW, TCP);
    tch->src = htons(srcport);
    tch->dst = htons(dstport);
    tch->off = 5;
    tch->seq = getrandom (0, 1) ? htonl (getrandom
(0, 65535) + (getrandom (0, 65535) << 8)) : 0;
    tch->ack = getrandom (0, 1) ? htonl (getrandom
(0, 65535) + (getrandom (0, 65535) << 8)) : 0;
    tch->flg = getrandom (0, 1) ? (getrandom (0, 1) ?
SYN : ACK) : SYN | ACK;
    tch->win = getrandom (0, 1) ? htons (getrandom
(0, 1024)) : 0;
    tch->urp = 0;
    tch->sum = 0;

    p = buf + tot_len;
    memcpy(p, srcdata, datalen);
    tot_len += datalen;

    i = tot_len-sizeof(struct ip);
    tch->sum = tcpchecksum(ih, (u16 *) tch, i);
    ih->tl = tot_len;
    dst_addr.sin_port = htons(1024);
    break;
    }

  if (debug) {
    printf("rawtransmit() buf: ");
    printhex(buf,tot_len);
    printf("\n");
                }
  if (ssock==-1) {
    if (debug) {
      printf("Error: rawtransmit() create socket\n");
      }
    error = 0x01;
    }
  if (setsockopt (ssock, IPPROTO_IP, IP_HDRINCL, &i,
sizeof(int)) < 0 ) {
    if (debug) {
      printf("Error: rawtransmit() set socket
option\n");
      }
    error = error | 0x02;
    }
  if (sendto (ssock, buf, tot_len, 0, (struct
sockaddr_in *) &dst_addr, sizeof (dst_addr)) < 0) {
    if (debug) {
      printf("Error: sendto() in rawtransmit() ");
      perror ("sendto");
      printf("\n");
      }
      error = error | 0x04;
    }

  close (ssock);
  return (error);
}

void random_init (void)
{
  int rfd = open ("/dev/urandom", O_RDONLY);
  if (rfd < 0)
    rfd = open ("/dev/random", O_RDONLY);
  rcounter = read (rfd, rseed, 65535);
  close (rfd);
}

inline long getrandom (int min, int max)
{
  if (rcounter < 2)
    random_init ();
  srand (rseed[rcounter] + (rseed[rcounter - 1] <<
8));
  rcounter -= 2;
  return ((random () % (int) (((max) + 1) - (min))) +
(min));
}
void printhex(unsigned char *ucIn, unsigned int
uiLen)
{
  unsigned char *ucPoint;
  unsigned int uiCount;

  ucPoint = ucIn;
  for (uiCount=0;uiCount<uiLen;uiCount++) {
    printf("%02X",*ucPoint);
    ucPoint++;
    }
}
int str2proto(char *proto)
{
  str2upper(proto);
  if (!strcmp(proto,"ICMP"))
    return ICMP;
  else if (!strcmp(proto,"TCP"))
    return TCP;
  else if (!strcmp(proto,"UDP"))
    return UDP;
  else if (!strcmp(proto,"RAW"))
    return RAW;
```

```
  else if (!strcmp(proto,"IP"))
    return TCP;
  else
    return RAW;
}
char *str2upper(char *in)
{
  char *cp;
  cp = in;
  while (*cp!=0) {
    *cp = toupper(*cp);
    cp++;
    }
  return in;
}
```

โปรแกรมต้นฉบับ GENTRAFFIC.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <time.h>
#include "../genlib.h"
#define VERSION 1

int debug=1;
int gendebug=1;

void app_usage(void);

int main(int argc, char *argv[])
{
  int mode;
  char intaddr[1024], extaddr[1024], infile[254], outfile[254], outfile2[254];

  if (argc<2) {
    app_usage();
    exit(1);
    }

  srand( (unsigned)time(NULL) );
  mode = atoi(argv[1]);

  switch (mode) {
    case 1:
      if (argc<3) {
        app_usage();
        exit(1);
        }
      extension_over(infile,argv[2],"acl");
      extension_over(outfile,argv[2],"rules");
      printf("\nGenTraffic: Compile ACL to SNORT rules\n");
      printf("  Input ACL rule file    = %s\n",infile);
      printf("  Output SNORT rule file = %s\n",outfile);
      printf("Compiling...\n");
      rule_acl2snort(infile,outfile);
      printf("Done.\n");
      break;
    case 2:
      if (argc<6) {
        app_usage();
        exit(1);
        }
      extension_over(infile,argv[2],"rules");
      extension_over(outfile,argv[5],"rules");
      extension_over(outfile2,argv[5],"index");
      printf("\nGenTraffic: Generate Traffic rules from SNORT rules\n");
      printf("  Input SNORT rule file  = %s\n",infile);
      printf("  Internal address list  = %s\n",argv[3]);
      printf("  External address list  = %s\n",argv[4]);
      printf("  Output traffic file    = %s\n",outfile);
      printf("  Output traffic index   = %s\n",outfile2);
      printf("Generating traffic pattern ...\n");
      address_getfromfile(intaddr, argv[3], 1024/STRLEN_ADDRLIST);
      address_getfromfile(extaddr, argv[4], 1024/STRLEN_ADDRLIST);
      traffic_gen(2, infile, outfile2, outfile, extaddr, intaddr);
      printf("Done.\n");
      break;
```

```
    }

  return 0;
}
void app_usage(void)
{
  printf("Usage  gentraffic 1 <input file>\n");
  printf("       gentraffic 2 <input file> <in addr-list> <ext addr-list> <output file>\n");
  printf("Mode:  1 = Complile input ACL to SNORT rules file\n");
  printf("           Output file will have .rules extension\n");
  printf("       2 = Generate traffic pattern/index file from SNORT rules file\n");
  printf("           Output file will have .rules and .index extension\n");
  printf("Note:  in addr-list  = Internal address list, required for mode 2\n");
  printf("       ext addr-list = External address list, required for mode 2\n");
}
```

โปรแกรมต้นฉบับ SENDER.C

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include "../netlib.h"
#include "../genlib.h"

int debug=1;
int gendebug=1;
int continuesend=1;

int sender_usage(const char *msg);
int sender_cleanup();
int traffic_sendfromfile(int mode, char *infile, char *idxfile, unsigned long packetnum, int interval);

int main(int argc, char *argv[])
{
  int mode, count, debugmode;
  char infile[254], indexfile[254];

  if (argc<4) {
    sender_usage("");
    exit(1);
    }
  if (argc==5) {
    debugmode = atoi(argv[4]);
    switch (debugmode) {
      case 0: debug=0; gendebug=0; break;
      case 1: debug=1; gendebug=1; break;
      case 2: debug=0; gendebug=1; break;
      }
    }
  count = atoi(argv[2]);
  mode = atoi(argv[3]);
  if (mode<1 || mode>3) {
    sender_usage("");
    exit(1);
    }
  if (count!=0)
    continuesend=0;
  extension_over(infile,argv[1],"rules");
  extension_over(indexfile,argv[1],"index");
  traffic_sendfromfile(mode, infile, indexfile, count, 1);
  return 0;
}
int sender_usage(const char *msg)
{
  if (strlen(msg)!=0)
    printf("%s\n",msg);
  else {
    printf("Usage: sender <input file> <count> <sending mode> <debug mode>\n");
    printf("  input file      = name of input file .rules and .index\n");
    printf("  count           = packet counts, 0 for continue loop\n");
    printf("  sending mode: 1 = random\n");
```

```
    printf("                2 = sequential\n");
    printf("  debug mode:   0 = off\n");
    printf("                1 = all\n");
    printf("                2 = error\n");
    }
  return 0;
}
int traffic_sendfromfile(int mode, char *infile, char *idxfile, unsigned long packetnum, int delay)
{
  FILE *infp, *idxfp;
  unsigned long ulCount, ulPos;
  long lIndex, max;
  char buff[255], szAct[8], szSrc[20], szDst[20], szPro[8], szSrcport[8], szDstport[8], szData[200], *pSz;
  unsigned long srcaddr, dstaddr;
  unsigned int uiSrcport, uiDstport;
  int iLen, iProto;

  infp = fopen(infile, "rt");
  idxfp = fopen(idxfile, "rb");

  if (!infp) {
    printf("Error: traffic_sendfromfile() can't open input file for read\n");
    return -1;
    }
  if (!idxfp) {
    printf("Error: traffic_sendfromfile() can't open index file for read\n");
    return -2;
    }
  max = getfilesize(idxfp);
  max = max >> 2;
  switch (mode) {
    case 1:
      for (ulCount=0;ulCount<packetnum || continuesend==1 ;ulCount++) {
        if (ulCount==packetnum)
          ulCount=0;
        ulPos = (rand() % (max + 1));
        fseek(idxfp,ulPos*4,SEEK_SET);
        fread(&lIndex, sizeof(long), 1, idxfp);
        lIndex = ntohl(lIndex);
        fseek(infp,lIndex,SEEK_SET);
        if (gendebug==1) {
          printf("sending: packet# %lu index# %lu \n",ulCount,ulPos);
          }
        iLen = filegetline(infp,buff,254);
        if (iLen>1) {
            sscanf(buff,"%s %s %s %s -> %s %s",szAct,szPro,szSrc,szSrcport,szDst,szDstport);
          pSz = strchr(szSrc,'/');
          if (pSz!=NULL) *pSz = 0;
          pSz = strchr(szDst,'/');
          if (pSz!=NULL) *pSz = 0;
          srcaddr = (unsigned long)inet_addr(szSrc);
          dstaddr = (unsigned long)inet_addr(szDst);
          uiSrcport = atoi(szSrcport);
          uiDstport = atoi(szDstport);
          iProto = str2proto(szPro);
          rawtransmit(srcaddr,uiSrcport,dstaddr,uiDstport,iProto,"hello!",6);
          }
        }
      break;
    case 2:
      for (ulCount=0;ulCount<packetnum || continuesend==1;ulCount++) {
        if (ulCount%max==0) {
          fseek(idxfp,0,SEEK_SET);
          printf("Rewind index file ...\n");
          }
        fread(&lIndex, sizeof(long), 1, idxfp);
        lIndex = ntohl(lIndex);
        fseek(infp,lIndex,SEEK_SET);
        iLen = filegetline(infp,buff,254);
        if (gendebug==1) {
          printf("sending: packet# %lu\n",ulCount);
          }
```

```
        if (iLen>1) {
            sscanf(buff,"%s %s %s %s -> %s %s",szAct,szPro,szSrc,szSrcport,szDst,szDstport);
          pSz = strchr(szSrc,'/');
          if (pSz!=NULL) *pSz = 0;
          pSz = strchr(szDst,'/');
          if (pSz!=NULL) *pSz = 0;
          srcaddr = (unsigned long)inet_addr(szSrc);
          dstaddr = (unsigned long)inet_addr(szDst);
          uiSrcport = atoi(szSrcport);
          uiDstport = atoi(szDstport);
          iProto = str2proto(szPro);
          rawtransmit(srcaddr,uiSrcport,dstaddr,uiDstport,iProto,"hello!",6);
          }
        }
      break;
return 0;
}
```

ประวัติผู้เขียน

ชื่อผู้เขียน	นายชินวัฒน์ ติวิธมไหศูรย์
วันเกิด	26 พฤษภาคม 2517
สถานที่เกิด	กรุงเทพมหานคร
วุฒิการศึกษาระดับปริญญาตรี	วศ.บ. (คอมพิวเตอร์) คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ตำแหน่งหน้าที่	วิศวกรระบบเครือข่าย
สถานที่ทำงาน	บริษัท เอสโซ่ (ประเทศไทย) จำกัด มหาชน