การค้นหาโดยใช้ตัวแบ่งระหว่างคำเพื่อเพิ่มความเร็วในการตรวจสอบกฎสำหรับโปรแกรมช่วยรักษาความปลอดภัยในระบบคอมพิวเตอร์

DELIMITER SEARCH FOR LOCATING RULE PATTERN IN SECURITY APPLICATION

ธนัญชัย ตรีภาค
THANUNCHAI THREEPAK

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2546

ISBN 974-324-522-7

การค้นหาโดยใช้ตัวแบ่งระหว่างคำเพื่อเพิ่มความเร็วในการตรวจสอบกฎสำหรับโปรแกรมช่วยรักษาความปลอดภัยในระบบคอมพิวเตอร์

DELIMITER SEARCH FOR LOCATING RULE PATTERN IN SECURITY APPLICATION

ธนัญชัย ตรีภาค

THANUNCHAI THREEPAK

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิศวกรรมคอมพิวเตอร์
บัณฑิตวิทยาลัย
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2546

ISBN 974-324-522-7

DELIMITER SEARCH FOR LOCATING RULE PATTERN IN SECURITY APPLICATION

THANUNCHAI THREEPAK

A THESIS SUBMITTED IN PARTIAL FULFILLMENT

OF THE REQUIREMENT FOR THE DEGREE OF

MASTER OF ENGINEERING IN COMPUTER ENGINEERING

SCHOOL OF GRADUATE STUDIES

KING MONGKUT'S INSTITUTE OF TECHNOLOGY LADKRABANG

2003

ISBN 974-324-522-7

บัณฑิตวิทยาลัย

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ใบรับรองวิทยานิพนธ์

หัวข้อวิทยานิพนธ์ การค้นหาโดยใช้ตัวแบ่งระหว่างคำเพื่อเพิ่มความเร็วในการตรวจสอบกฎสำหรับโปรแกรมช่วยรักษาความปลอดภัยในระบบคอมพิวเตอร์

DELIMITER SEARCH FOR LOCATING RULE PATTERN IN SECURITY APPLICATION

ชื่อนักศึกษา นายธนัญชัย ตรีภาค

รหัสประจำตัว 43061609

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.ดร.บุญธีร์ เครือตราชู

คณะกรรมการสอบวิทยานิพนธ์		ลายมือชื่อ
ดร.วรวัฒน์	ลิ้มโภคา	
ดร.วิศิษฏ์	หิรัญกิตติ	
รศ.ดร.เอื้อน	ปิ่นเงิน	
รศ.ดร.บุญวัฒน์	อัตชู	
รศ.ดร.บุญธีร์	เครือตราชู	

วัน/เดือน/ปี ที่สอบ 28 เมษายน 2546 เวลา 10.30-12.30 น.

สถานที่สอบ ณ อาคาร 12 ชั้น ชั้น 4 (ห้อง E12-403)

บัณฑิตวิทยาลัยรับรองแล้ว

(รศ.ดร.บุญวัฒน์ อัตชู)

คณบดีบัณฑิตวิทยาลัย

วันที่ 30 **เดือน** พฤษภาคม **พ.ศ.** 2546

หัวข้อวิทยานิพนธ์ การค้นหาโดยใช้ตัวแบ่งระหว่างคำเพื่อเพิ่มความเร็วในการตรวจสอบกฎสำหรับโปรแกรมช่วยรักษาความปลอดภัยในระบบคอมพิวเตอร์

นักศึกษา นายธนัญชัย ตรีภาค

รหัสนักศึกษา 43061609

ปริญญา วิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชา วิศวกรรมคอมพิวเตอร์

พ.ศ. 2546

อาจารย์ผู้ควบคุมวิทยานิพนธ์ รศ.ดร. บุญธีร์ เครือตราชู

บทคัดย่อ

งานวิจัยชิ้นนี้มีจุดประสงค์เพื่อปรับปรุงประสิทธิภาพในการทำงานของระบบตรวจจับผู้บุกรุกทางเครือข่ายหรือ Network Intrusion Detection System (NIDS) เพื่อให้ระบบดังกล่าวสามารถวิเคราะห์ข้อมูลได้เร็วขึ้น โดยการใช้วิธีการค้นหาโดยใช้ตัวแบ่งระหว่างคำหรือ delimiter search และเปลี่ยนแปลงโครงสร้างของกฎ ให้อยู่ในรูปโครงสร้างของ burst tries ในการทดสอบสมมุติฐานนั้นใช้วิธีการนับจำนวนการเปรียบเทียบตัวอักษรในกฎ ซึ่งเปรียบเทียบผลลัพธ์ของโครงสร้างกฎ 3 โครงสร้างคือ โครงสร้างกฎดั้งเดิมของโปรแกรม snort ซึ่งเป็น NIDS ที่ใช้กันอย่างแพร่หลาย , โครงสร้างกฎแบบ burst tries และโครงสร้างกฎใหม่ที่รวมเอา burst tries กับ delimiter search เข้าด้วยกัน ซึ่งจากการทดลองพบว่า โครงสร้างกฎที่รวมเอา delimiter กับ burst tries เข้าด้วยกันจะมีการเปรียบเทียบตัวอักษรในการตรวจสอบกฎน้อยที่สุดคือประมาณ 4.96 เปอร์เซ็นต์ของโครงสร้างกฎแบบดั้งเดิมเท่านั้น ซึ่งผลลัพธ์ดังกล่าวขึ้นอยู่กับชุดของข้อมูล ชุดของกฎและเซตของตัวแบ่งระหว่างคำที่นำมาใช้ด้วย

Thesis Title	Delimiter Search for Locating Rule Pattern in Security Application
Student	Mr. Thanunchai Threepak
Student ID.	43061609
Degree	Master of Engineering
Programme	Computer Engineering
Year	2003
Thesis Advisor	Assoc. Prof. Dr. Boontee Kruatrachue

ABSTRACT

The goal of this research is to improve the run time performance of Network Intrusion Detection Systems (NIDS). The system performs intrusion detection by matching specific intrusion patterns with the incoming network packet data. Since the numbers of patterns is large (up to 2000) and keep increasing leading to huge search time. In this paper, we proposed a search time reduction by using delimiter search and implementing intrusion patterns into a dictionary of patterns with burst tries structure. The search time is compared between the snort Boyer-Moore algorithm and proposed method. The result of the proposed method on average take only 4.96 percent of time of original snort.And the result is upto data , rules and delimiter set.

กิตติกรรมประกาศ

วิทยานิพนธ์นี้สำเร็จได้ด้วยความอนุเคราะห์จาก รศ.ดร.บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษาวิทยานิพนธ์ และ อาจารย์ธนา หงษ์สุวรรณ หัวหน้าห้องวิจัย ISAG ที่ได้กรุณาให้คำแนะนำ ให้ความช่วยเหลือ ให้กำลังใจ และช่วยตรวจสอบ แก้ไข จนวิทยานิพนธ์นี้สำเร็จได้อย่างสมบูรณ์ ผู้วิจัยรู้สึกซาบซึ้งในความกรุณา และขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณอาจารย์ทุกท่าน ที่ได้ประสิทธิ์ประสาทความรู้ ตลอดจนข้อคิดต่างๆ อันก่อให้เกิดประโยชน์ต่อการศึกษาค้นคว้า และเป็นแนวทางในการจัดทำวิทยานิพนธ์จนประสบความสำเร็จ

ขอขอบพระคุณ คุณพ่อ และคุณแม่ ผู้เป็นที่เคารพรักยิ่ง ให้การสนับสนุนและช่วยเหลือทุกด้าน ตลอดจนกำลังใจที่ไม่เคยหมดสิ้น

ขอขอบคุณพี่ๆ น้องๆ ห้องวิจัย ISAG ที่ช่วยเหลือด้านข้อมูล และอุปกรณ์ในการทำงานวิจัย รวมถึง เพื่อนๆ และบุคคลที่ผู้วิจัยไม่ได้กล่าวไว้ในที่นี้ ที่ให้การสนับสนุน และความช่วยเหลือในด้านต่างๆ ตลอดจนเป็นกำลังใจให้ผู้วิจัยมาโดยตลอด

คุณค่า และประโยชน์ใดๆ ที่เป็นผลมาจากวิทยานิพนธ์นี้ ผู้วิจัยขอมอบแด่ คุณพ่อ คุณแม่ และ ครู-อาจารย์ ทุกท่าน ด้วยความเคารพยิ่ง

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1 ความเป็นมาและความสำคัญของปัญหา

เทคโนโลยีอินเตอร์เน็ต และเครือข่ายได้พัฒนาขึ้นทุกที องค์กรต่างๆ มีการใช้งานคอมพิวเตอร์และเครือข่ายเพิ่มมากขึ้น ปัญหาที่ตามมาก็คือ ปัญหาความปลอดภัยในระบบคอมพิวเตอร์ ทั้งปัญหาความปลอดภัยของผู้ใช้งาน และปัญหาความปลอดภัยของตัวระบบคอมพิวเตอร์เอง ซึ่งต่อมามีการพัฒนาระบบตรวจจับผู้บุกรุกขึ้น เพื่อรับผิดชอบการตรวจจับความผิดปกติต่างๆ ในระบบคอมพิวเตอร์ และเครือข่าย แต่การทำงานของระบบดังกล่าวยังมีปัญหาในสองประเด็นคือยังมีความสามารถไม่พอในการดูแลระบบเครือข่ายที่มีข้อมูลปริมาณมาก และมี traffic สูงๆ ได้ และอีกประเด็นหนึ่งก็คือ ระบบดังกล่าวกลายเป็นเป้าหมายในการโจมตีเสียเอง โดยการโจมตีอยู่ในรูปของการสร้าง bandwidth ปริมาณมากเข้าสู่ระบบ ทำให้ระบบทำงานไม่ทัน และมีปัญหาในที่สุด

แนวทางในการแก้ไขปัญหาดังกล่าวคือการพัฒนาประสิทธิภาพให้ระบบตรวจจับผู้บุกรุกทำงานได้เร็วขึ้น เพื่อให้สามารถรองรับการทำงานในเครือข่ายที่มี traffic สูงๆ และทนทานต่อการโจมตีของผู้ไม่หวังดีได้ ถ้าพัฒนาประสิทธิภาพให้ระบบตรวจจับผู้บุกรุกทำงานได้เร็วขึ้นตามที่ออกแบบไว้ ปัญหาที่ไม่สามารถตรวจสอบสัญญาณการบุกรุกได้ทันในเครือข่ายที่มี traffic สูงๆ หรือปัญหาที่ไม่สามารถตรวจจับสัญญาณการบุกรุกได้ขณะที่ระบบถูกโจมตีนั้นก็จะหมดไป ทำให้ระบบมีความปลอดภัยมากขึ้น

1.2 ความมุ่งหมายและวัตถุประสงค์ของการศึกษา

ความมุ่งหมายและวัตถุประสงค์ของการศึกษานี้ คือการปรับปรุงประสิทธิภาพการทำงานของระบบตรวจจับผู้บุกรุก โดยประยุกต์ใช้การค้นหาโดยใช้ตัวแบ่งระหว่างคำ(delimiter search) และการเปลี่ยนแปลงโครงสร้างการทำงานให้เหมาะสมกับปัญหาของระบบตรวจจับผู้บุกรุกเพื่อลด ขนาดของข้อมูลที่จะต้องค้นหา แล้วทำการทดสอบให้เห็นจริงว่าการประยุกต์ใช้การค้นหาโดยใช้ตัวแบ่งระหว่างคำ และการแก้ไขโครงสร้างให้เหมาะสมนั้น สามารถเพิ่มประสิทธิภาพการทำงานของโปรแกรมตรวจจับการบุกรุกได้

1.3 สมมุติฐานของการศึกษา

สมมุติฐานของงานวิจัยคือ ถ้าสามารถลดขนาดของข้อมูลด้วยวิธีการของการค้นหาโดยใช้ตัวแบ่งระหว่างคำและเปลี่ยนแปลงโครงสร้างกฎให้สอดคล้องกับการทำงานจะทำให้การทำงานของระบบทำได้เร็วขึ้น ในกรณีของโปรแกรม SNORT ถ้าเปลี่ยนแปลงโครงสร้างกฎให้เป็นแบบ trie และใช้ลักษณะการทำงานของการค้นหาโดยใช้ตัวแบ่งระหว่างคำมาประยุกต์ใช้ จะทำให้การทำงานของโปรแกรม SNORT ทำงานได้เร็วขึ้น

1.4 ทฤษฎีหรือแนวความคิดที่ใช้ในการวิจัย

แนวคิดที่ใช้ในงานวิจัยนี้คือการลดขนาดของข้อมูลในการค้นหา หรือ search space ของปัญหาเพื่อให้ระยะเวลาในการค้นหาคำตอบลดลง ในการศึกษาครั้งนี้ใช้วิธีการเปลี่ยนรูปแบบของคำสำคัญ (keyword) และข้อมูลที่ต้องค้นหาให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งระหว่างคำเพื่อลดขนาดของข้อมูลและขนาดของคำสำคัญให้เล็กลง

1.5 ขอบเขตการวิจัย

ขอบเขตของการวิจัยศึกษาและทดลองคือหาโครงสร้างข้อมูลที่เหมาะสมและความสัมพันธ์ของเซตของตัวแบ่งระหว่างคำ (delimiter set) สำหรับปรับปรุงโปรแกรม SNORT ซึ่งเป็นระบบตรวจจับผู้บุกรุกทางเครือข่ายที่ใช้งานกันอย่างแพร่หลาย ให้สามารถทำงานให้เร็วขึ้น โดยนำเอาเซตของตัวแบ่งระหว่างคำที่ได้มาลดขนาดของข้อมูล แล้วทดสอบเพื่อเปรียบเทียบการทำงานระหว่างรูปแบบการค้นหาโดยโครงสร้างข้อมูลแบบเก่ากับการค้นหาโดยโครงสร้างข้อมูลแบบใหม่

1.6 ขั้นตอนการศึกษา

1. ศึกษาการทำงานของโปรแกรม SNORT
2. ศึกษารูปแบบกฎและโครงสร้างของกฎในโปรแกรม SNORT
3. หาทางเปลี่ยนแปลงรูปแบบของข้อมูลในแพ็กเก็ต และข้อมูลในกฎของโปรแกรม SNORT เพื่อลดขนาดของ search space
4. หาทางเปลี่ยนแปลงโครงสร้างกฎของโปรแกรม SNORT ให้อยู่ในรูปที่ไม่มีความซ้ำซ้อนในการทำงาน
5. ทดสอบประสิทธิภาพการทำงานของระบบใหม่ เมื่อเทียบกับระบบเก่า
6. วิเคราะห์การทำงานของกระบวนการตรวจสอบกฎแต่ละแบบ

บทที่ 2
เอกสารและงานวิจัยที่เกี่ยวข้อง

2.1 Libpcap

ในการวิเคราะห์ข้อมูลในเครือข่ายนั้น จำเป็นต้องใช้ข้อมูลที่ผ่านไปมาในเครือข่าย เพื่อนำรายละเอียดของข้อมูลและโพรโตคอลต่างๆ รวมถึงความหนาแน่นของข้อมูลมาใช้ในการวิเคราะห์ โดยปกติแล้วการทำงานของเน็ตเวิร์คการ์ดจะรับข้อมูลเฉพาะข้อมูลที่จะส่งมายังการ์ดใบนั้นเท่านั้น จะไม่ยอมรับข้อมูลอื่นๆ เลย ในการวิเคราะห์ข้อมูลในเครือข่ายนั้นจึงต้องเปลี่ยนโหมดการทำงานปกติของเน็ตเวิร์คการ์ดให้กลายเป็น Promiscuous mode เสียก่อน โดย Promiscous mode เป็นโหมดการทำงานหนึ่งของเน็ตเวิร์คการ์ดที่แตกต่างการโหมดการทำงานปกติคือ จะรับแพ็กเก็ตทุกๆ แพ็กเก็ตที่ผ่านไปมาในเครือข่าย โดยไม่คำนึงว่าเป็นแพ็กเก็ตที่ส่งมายังตัวมันเองหรือเปล่า ในการทำงานในโหมดนี้มีความจำเป็นอย่างยิ่งในการวิเคราะห์เครือข่ายเพราะช่วยให้ผู้วิเคราะห์ได้ข้อมูลทุกๆ อย่างที่ผ่านไปมาในเครือข่าย

เดิมการเขียนโปรแกรมเพื่อเปลี่ยนโหมดการทำงานของเน็ตเวิร์คการ์ดต้องเขียนโปรแกรมในระดับฮาร์ดแวร์ซึ่งมีความยุ่งยากมากในการอ่านและเขียนข้อมูลในเมมโมรี และต้องการการควบคุมช่วงเวลาการทำงานในการรับข้อมูลด้วย จึงมีผู้คิดค้นไลบรารี่ที่ช่วยให้การเขียนโปรแกรมเพื่อทำงานเกี่ยวกับการวิเคราะห์ข้อมูลเครือข่ายขึ้น เพื่อให้ใช้งานได้ง่าย และมีประสิทธิภาพ โดยใช้ชื่อว่า Libpcap (Jacobson, V. et. al.:2002.)

Libpcap เป็นไลบรารี่ที่สร้างโดย Van Jacobson, Craig Leres และ Steven McCanne ทั้งสามทำงานอยู่ที่ Lawrence Berkeley National Laboratory , University of California จุดประสงค์ในการสร้างก็เพื่อเป็นไลบรารี่ที่ช่วยในการเขียนโปรแกรมสำหรับการดึงข้อมูลจากเครือข่าย โดยที่ผู้เขียนโปรแกรมไม่จำเป็นต้องยุ่งเกี่ยวกับการตั้งค่าฮาร์ดแวร์เลย ตัวไลบรารีจะจัดการตั้งค่าต่างๆ รวมทั้งควบคุมช่วงเวลาการทำงานในส่วนของฮาร์ดแวร์ให้โดยอัตโนมัติ ผู้ใช้เพียงเรียกใช้ API ที่ไลบรารี่ดังกล่าวสร้างมาให้ถูกลำดับเท่านั้น

ในการทำงานปกติ กระบวนการในการทำงานกับข้อมูลเครือข่ายแต่ละแพ็กเก็ตต้องมีการอ่านข้อมูลในหน่อยความจำหลักของคอมพิวเตอร์ ซึ่งเมื่อขนาดของแพ็กเก็ตแต่ละแพ็กเก็ตไม่เท่ากันทำให้การอ่านข้อมูลทำได้ยาก ไลบรารี่ libpcap ได้ช่วยแก้ปัญหาตรงส่วนนี้ให้หมดไปได้ โดยการให้โครงสร้างข้อมูลมาเพื่อเก็บข้อมูลโดยเฉพาะ ผู้ใช้งานไม่จำเป็นต้องวุ่นวายกับการจัดการกับข้อมูลแพ็กเก็ตที่มีขนาดไม่เท่ากันตลอดเวลา

2.1.1 ความสามารถของ libpcap

ความสามารถของไลบรารี่ libpcap นั้นมีความสามารถหลักๆ ดังนี้คือ

1. สามารถหาตำแหน่งแอดเดรสของเน็ตเวิร์คการ์ดได้โดยผู้ใช้เพียงแต่ทราบชื่อของอุปกรณ์เท่านั้น
2. สามารถหาค่าหมายเลขเครือข่าย และซับเน็ตมาส์ก (subnet mask) ของอุปกรณ์ได้ สามารถตั้งค่าเน็ตเวิร์คการ์ดที่ต้องการใช้ได้ โดยสามารถตั้งค่าได้สองค่าคือโหมดการทำงานปกติ และ promiscuous mode
3. สามารถวนลูปเพื่ออ่านค่าข้อมูลที่ผ่านไปมาในเน็ตเวิร์คการ์ดได้
4. สามารถทำงานกับโปรแกรมสำหรับกรองข้อมูลได้
5. สามารถเขียนข้อมูลที่อ่านมาได้ลงในไฟล์ข้อมูลที่กำหนดให้ได้
6. ฯลฯ

2.1.2 ข้อจำกัดของ libpacp

ไลบรารี่ libpcap มีข้อจำกัดคือยังไม่สามารถทำงานกับระบบปฏิบัติการทุกๆ ระบบได้ ขณะนี้สามารถทำงานได้กับระบบปฏิบัติการบางระบบเช่น LINUX (2.2.x,2.4.x) ,AIX ,DEC OSF/1 เป็นต้น ซึ่งในระบบปฏิบัติการวินโดวส์นั้นมีไลบรารี่ที่ทำงานลักษณะเดียวกันในชื่อ Winpacp ซึ่งเป็นการนำ libpcap ไปพัฒนาต่อให้ใช้กับระบบปฏิบัติการวินโดวส์ได้

2.2 Boyer-moore algorithm

อัลกอริทึมบอยเออร์-มัวร์ (Boyer, R. S. and Moore, J. S. :1977.) เป็นอัลกอริทึมที่คิดค้นโดย BOYER R.S., MOORE J.S. ตีพิมพ์ใน Communications of the ACM. เมื่อปี 1977 โดยมีชื่องานวิจัยว่า "A fast string searching algorithm" โดยอัลกอริทึมนี้ทำให้การค้นหาคำในข้อมูลขนาดใหญ่ทำได้เร็วขึ้น โดยการสร้างตารางในการ shift ขึ้นมาสองตาราง ทำงานเปรียบเทียบตัวอักษรทีละตัว ถ้าตัวอักษรไม่เหมือนกัน จึงทำงานตามเงื่อนไขสองข้อคือ bad-character shift และ good-suffix shift เพื่อตัดสินใจว่าจะทำการ shift จากจุดเดิมไปกี่ตัวอักษร ข้อดีของอัลกอริทึมนี้คือในกระบวนการเปรียบเทียบไม่จำเป็นต้องเปรียบเทียบตัวอักษรทั้งหมดทุกตัว และมีกระบวนการ shift ทำให้การค้นหาทำได้เร็วขึ้น

2.2.1 การทำงานเด่นของอัลกอริทึม

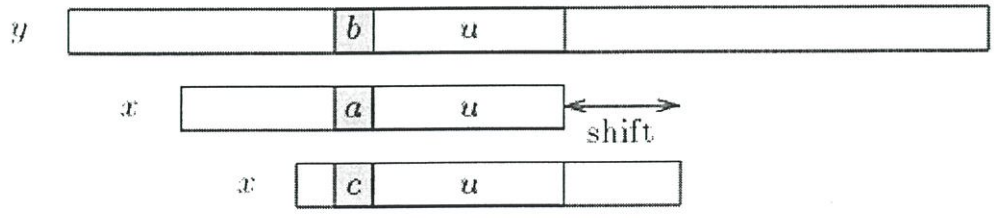
จากการค้นหาคำ x ในข้อมูล y โดยคำ x มีความยาวเท่ากับ m ตัวอักษร และข้อมูล y มีความยาวเท่ากับ n ตัวอักษร สำหรับทั้ง x และ y จะมีตัวอักษรอยู่ในเซตของตัวอักษรที่มีขนาดเท่ากับ σ ตัวอักษร จากการทดสอบเพื่อสรุปผลจากปัญหาดังกล่าว จะได้ข้อสรุปการทำงานของอัลกอริทึมบอยเออร์-มัวร์ดังนี้

1. มีการเปรียบเทียบข้อมูลจากขวามาซ้าย
2. กระบวนการก่อนการทำงานจะใช้เวลาและเนื้อที่เป็นออเดอร์ $O(m + \sigma)$
3. กระบวนการในการค้นหาคำจะใช้เวลาเป็นออเดอร์ $O(m \times n)$
4. ในกรณีที่แย่ที่สุดในการค้นหาข้อมูลจะใช้จำนวนการเปรียบเทียบทั้งหมดเป็น 3n เมื่อเปรียบเทียบรูปแบบที่ไม่มีความซ้ำซ้อนกันอยู่ภายในคำเลย
5. ในกรณีที่ค้นหาได้ดีที่สุดจะใช้เวลาเป็น O(m/n)

2.2.2 รายละเอียดการทำงานของอัลกอริทึม

อัลกอริทึมบอยเออร์-มัวร์ (boyer-moore algorithm) เป็นอัลกอริทึมที่มีประสิทธิภาพสูงสุดในการทำงานโดยทั่วๆ ไป ซึ่งได้มีการนำเอาอัลกอริทึมนี้หรือการประยุกต์อื่นๆ ของอัลกอริทึมนี้มาใช้เป็นส่วนหนึ่งของการทำงานของคำสั่ง “Search” หรือคำสั่ง “Substitute” ในโปรแกรมต่างๆ ที่ใช้งานกันอยู่โดยทั่วไปด้วย

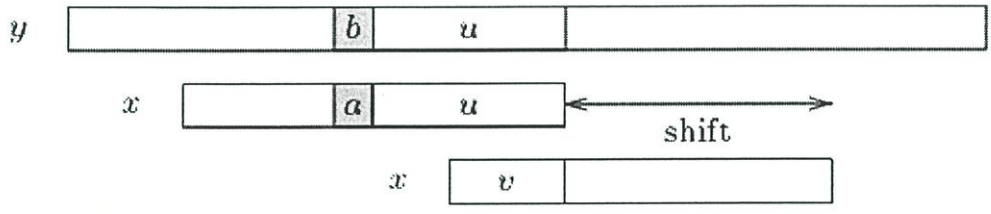
โดยการทำงานของอัลกอริทึมนี้จะทำการหาตัวอักษรของรูปแบบที่ต้องการหาจากขวาไปซ้ายโดยเริ่มหาจากตัวอักษรขวามือสุดก่อน ในกรณีที่เปรียบเทียบแล้วไม่เหมือนกัน (หรือในกรณีที่เหมือนกันหมดทั้งคำ) อัลกอริทึมนี้จะใช้ฟังชั่นสองฟังก์ชั่นที่ตัวอัลกอริทึมสร้างขึ้นในการกระโดดข้าม (Shift) ตัวอักษรเพื่อไปเปรียบเทียบในตัวอักษรถัดไป โดยฟังก์ชันทั้งสองนี้เรียกว่า good-suffix shift (หรือ matching shift) และอีกฟังก์ชันหนึ่งคือ bad-character shift (หรือเรียกว่า occurrence shift)



รูปที่ 2.1 การ shift ใน good-suffix shift ที่มีชุดตัวอักษรซ้ำกันในคำสำคัญ

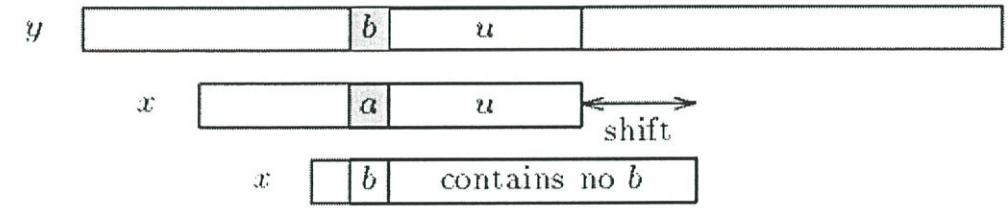
รูปที่ 2.1 เป็นเหตุการณ์ที่เกิดขึ้นในกรณีที่ค้นหาคำ x ในข้อมูล y ในกรณีนี้เป็นส่วนการทำงานที่มีการเปรียบเทียบตัวอักษรที่อินเด็กซ์ชี้อยู่ในคำแล้วไม่เหมือนกัน ที่ x[i] = a แต่ตัวอักษร

y[i+j] = b ส่วนตัวอักษรอื่นๆ ที่ตามหลังนั้นเหมือนกันทั้ง x และ y นั่นคือ x[i+1..m-1] = y[i+j+1..j+m+1] = u และ x[i] ≠ y[i+j] โดย i คือตัวชี้อันดับของตัวอักษรในคำ x และ j คือตัวชี้อันดับของตัวอักษรในข้อมูล y ในกรณีที่เกิดเหตุการณ์เช่นนี้ขึ้น จะมีการทำงานในส่วนของ good-suffix shift โดยฟังก์ชันจะทำการคำนวณหาชุดตัวอักษรที่มีลำดับเหมือนกับ x[i+1..m-1] ที่เกิดขึ้นอีกใน x แล้วทำการ shift ต่อไปเพื่อเปรียบเทียบตัวอักษรตัวหน้าสุดอีกครั้ง



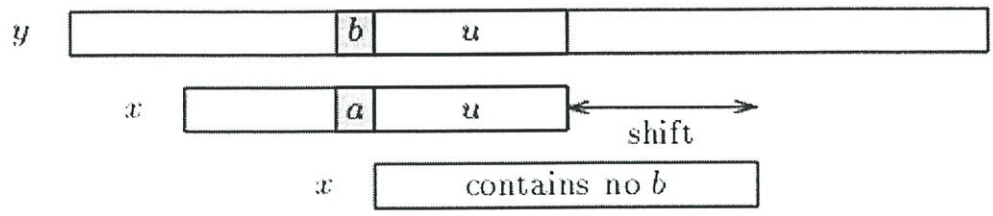
รูปที่ 2.2 การ shift ใน good-suffix shift ในกรณีที่มีชุดของตัวอักษรซ้ำกันเพียงบางส่วนในคำสำคัญ

ถ้าในกรณีที่ไม่มีส่วนของ x[i+1..m-1] เกิดขึ้นซ้ำอีกใน x ก็จะ shift ไปยังกลุ่มของตัวอักษรที่มีความยาวมากที่สุดที่ทำให้ได้กลุ่มคำ v ที่มีความยาวคำมากที่สุด ใน y[i+j+1..j+m-1] ที่เหมือนกับกลุ่มตัวอักษร v ที่อาจจะเป็นกลุ่มตัวอักษรที่อยู่หน้าสุดในคำ x ดังรูปที่ 2.2



รูปที่ 2.3 การทำ bad-character shift ในกรณีที่คำสำคัญมีตัวอักษรที่เหมือนกันกับตัวอักษร

สำหรับการทำ bad-character shift นั้นเกิดขึ้นในกรณีที่เหมือนกับ good-suffix shift คือเกิดขึ้นที่ x[i+1..m-1] = y[i+j+1..j+m+1] = u และ x[I] ≠ y[i+j] แต่มีหลักการ shift ที่ต่างกันคือ ในกรณีของ bad-character shift จะ shift ไปยังตัวอักษรใน y ที่เหมือนกับ x[i] ดังรูปที่ 2.3 ซึ่งในการทำงานของ bad-character shift อาจเกิดกรณีที่ไม่มีตัวอักษรใน y เลยที่เหมือนกับ x[i] ในกรณีนี้ก็จะสามารถ shift ไปได้เท่ากับความยาวคำที่เหลือของ x ดังรูปที่ 2.4



รูปที่ 2.4 การทำ bad-character shift ในกรณีที่คำสำคัญไม่มีตัวอักษรที่เหมือนกับตัวอักษรที่อินเด็กซ์ของข้อมูลชี้อยู่

จากลักษณะของ bad-character shift ในการทำงานจริงสามารถ shift เป็นค่าติดลบได้ ซึ่งทำให้เกิดปัญหาในการทำงานขึ้น อัลกอริทึมนี้แก้ปัญหาดังกล่าวด้วยการหาค่ามากที่สุดจากผลลัพธ์ของฟังก์ชั่น good-suffix-shift หรือ bad-character shift แล้ว shift ตามค่าที่ได้ สำหรับรายละเอียดของแต่ละฟังก์ชั่นมีรายละเอียดดังนี้คือ

2.2.2.1 good-suffix shift

เป็นฟังก์ชั่นที่เก็บอยู่ในตารางที่มีขนาดมากกว่าความยาวของคำสำคัญที่ใช้ค้นหาอยู่ 1 เช่น จะค้นหาคำสำคัญคือคำว่า “tables” ในเอกสารขนาดใหญ่ ตารางของ good-suffix shift จะมีขนาด 7 ช่อง ในตารางจะมีเงื่อนไขอยู่ 2 เงื่อนไขคือ

$$CS(i, s) : \text{for each } k \text{ such that } i < k < m,\ s \geq k \text{ or } x[k - s] = x[k],$$

$$Co(i, s) : \text{if } s < i \text{ then } x[i - s] \neq x[i]$$

สำหรับ ทุกๆ ค่า i ที่ $0 \leq i < m$:

$$BmGs[i+1] = \min \{ s>0 : Cs(i, s) \text{ and } Co(i, s) \text{ hold} \}$$

สำหรับค่าใน bmGs[0] จะเก็บค่าความยาวของ x หรือคำสำคัญนั้นๆ ส่วนการคำนวณค่าในตาราง bmGs นั้นจะใช้ตาราง suff มาช่วยในการคำนวณด้วย โดยค่าในตารางจะคำนวณได้โดย

$$\text{For } 1 \leq i < m,\ suff[i] = \max \{k : k[i - k + 1..i] = x[m - k, m - 1]\}$$

2.2.2.2 bad-character shift

สำหรับฟังก์ชัน bad-character shift จะเก็บในตารางของ bmBc ที่มีขนาด σ สำหรับทุกๆ $c \in \Sigma$ โดย

$$BmBc[c] = \min \{ I: 1 \leq I < m-1 \text{ and } x[m-1-I] = c \} \text{ if } x \text{ occors in } x, m \text{ otherwise}$$

สำหรับตาราง bmBc และ bmGs สามารถสร้างโดยใช้เวลาเป็นออเดอ $O(m + \sigma)$ ก่อนเริ่มทำการค้นหา และต้องการเนื้อที่ในการเก็บข้อมูลเป็นออเดอ $O(m + \sigma)$ ด้วยเช่นกัน ส่วนเวลาที่ใช้ในการค้นหาจะเป็นแบบ quadratic แต่จะใช้จำนวนการเปรียบเทียบตัวอักษรมากที่สุดคือ 3n ครั้ง เมื่อค้นหารูปแบบที่เป็น non periodic โดย n คือความยาวของข้อมูล

สำหรับเซตของคำที่ใหญ่มาก ซึ่งจะมีผลจากความยาวของคำที่ยาว อัลกอริทึมนี้จะมีการทำงานที่เร็วมาก ในกรณีที่ค้นหาคำว่า $a^{m-1}b$ ใน a^n อัลกอริทึมนี้จะใช้เพียง O(n/m) เท่านั้น

2.2.2.3 ตัวอย่างโค้ดภาษา C เพื่ออธิบายการทำงานของอัลกอริทึม

```
void preBmBc(char *x, int m, int bmBc[]) {
        int i;
        for (i = 0; i < ASIZE; ++i) bmBc[i] = m;
        for (i = 0; i < m - 1; ++i) bmBc[x[i]] = m - i - 1;
}
void suffixes(char *x, int m, int *suff) {
  int f, g, i;
  suff[m - 1] = m;
  g = m - 1;
  for (i = m - 2; i >= 0; --i) {
       if (i > g && suff[i + m - 1 - f] < i - g) suff[i] = suff[i + m - 1 - f];a
    } else {
      if (i < g)    g = i;
      f = i;
      while (g >= 0 && x[g] == x[g + m - 1 - f])   --g;
      suff[i] = f - g;
    }
  }
}
 void preBmGs(char *x, int m, int bmGs[]) {
        int i, j, suff[XSIZE];
        suffixes(x, m, suff);
        for (i = 0; i < m; ++i)   bmGs[i] = m;
        j = 0;
        for (i = m - 1; i >= -1; --i)
                if (i == -1 || suff[i] == i + 1)
                        for (; j < m - 1 - i; ++j)   if (bmGs[j] == m)  bmGs[j] = m - 1 - i;
  for (i = 0; i <= m - 2; ++i)     bmGs[m - 1 - suff[i]] = m - 1 - i;
```

```
}
void BM(char *x, int m, char *y, int n) {
  int i, j, bmGs[XSIZE], bmBc[ASIZE];

  /* Preprocessing */
  preBmGs(x, m, bmGs);
  preBmBc(x, m, bmBc);

  /* Searching */
  j = 0;
  while (j <= n - m) {
    for (i = m - 1; i >= 0 && x[i] == y[i + j]; --i);
    if (i < 0) {
      OUTPUT(j);
      j += bmGs[0];
    }
    else        j += MAX(bmGs[i], bmBc[y[i + j]] - m + 1 + i);
  }
}
```

2.2.2.4 ตัวอย่างการทำงานของอัลกอริทึม

ตัวอย่างการทำงานจะค้นหาคำในข้อมูลคือ "GCATCGCAGAGAGTATACAGTACG" โดยจะใช้คำสำคัญคือ "GCAGAGAG" โดยการทำงานในช่วงการเตรียมข้อมูลนั้น จะต้องเตรียมตารางของ bmGs และ bmBc ก่อน โดยตารางที่ได้จะมีลักษณะดังรูปที่ 2.5

c	A	C	G	T
$bmBc[c]$	1	6	2	8

i	0	1	2	3	4	5	6	7
$x[i]$	G	C	A	G	A	G	A	G
$suff[i]$	1	0	0	2	0	4	0	8
$bmGs[i]$	7	7	7	2	7	4	7	1

รูปที่ 2.5 ตาราง bmBc และ bmGs ที่ได้ในช่วงของการเตรียมข้อมูล

ในช่วงของการค้นหาคำสำคัญในข้อมูล มีกระบวนการสองส่วนคือการเปรียบเทียบตัวอักษร และการ shift ในการเปรียบเทียบตัวอักษรถ้าเปรียบเทียบแล้วเหมือนกันจะเปรียบเทียบตัวอักษรถัดไปเรื่อยๆ ส่วนการ shift นั้น จะเกิดขึ้นทุกครั้งที่มีการเปรียบเทียบตัวอักษรแล้วไม่เหมือนกัน ในการค้นหาครั้งนี้จะมีการ shift ทั้งหมด 5 ครั้ง โดยมีรายละเอียดดังนี้คือ

ครั้งที่ 1 ที่การเปรียบเทียบกับตัวอักษร A ซึ่งเมื่อดูจากตารางทั้งสองและอัลกอริทึมจะได้ว่า bmGs[7] = 1 และ bmBc[A] – 7 + 7 = 1 เมื่อหาตัวที่มากที่สุดจะได้การ shift ไป 1 ตัวอักษร ดังรูปที่ 2.6

```
G C A T C G C A G A G A G T A T A C A G T A C G
            1
G C A G A G A G
```

รูปที่ 2.6 ตัวอย่างการ shift ครั้งที่ 1

ครั้งที่ 2 ที่การเปรียบเทียบกับตัวอักษร C ซึ่งเมื่อดูจากตารางทั้งสองและ อัลกอริทึมจะได้ว่า bmGs[5] = 4 และ bmBc[C] – 7 + 5 = 4 เมื่อหาตัวที่มากที่สุดจะได้การ shift ไป 4 ตัวอักษร ดังรูปที่ 2.7

```
G C A T C G C A G A G A G T A T A C A G T A C G
          3 2 1
  G C A G A G A G
```

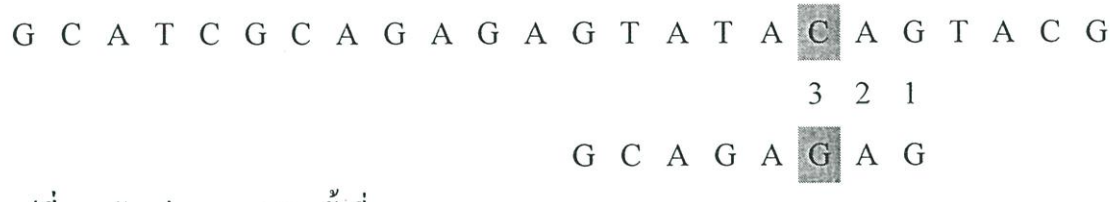
รูปที่ 2.7 ตัวอย่างการ shift ครั้งที่ 2

ครั้งที่ 3 เป็นกรณีที่ match กันทั้งหมด จึงส่งผลลัพธ์ของการค้นหาคือตำแหน่งของตัวอักษรในข้อมูล แล้วทำการ shift ไปเท่ากับค่า bmGs[0] = 7 ตัวอักษร ดังรูปที่ 2.8.

```
G C A T C G C A G A G A G T A T A C A G T A C G
        8 7 6 5 4 3 2 1
        G C A G A G A G
```

รูปที่ 2.8 ตัวอย่างการ shift ครั้งที่ 3

ครั้งที่ 4 ที่การเปรียบเทียบกับตัวอักษร C ซึ่งเมื่อดูจากตารางทั้งสองและ อัลกอริทึมจะได้ว่า bmGs[5] = 4 และ bmBc[C] – 7 + 5 = 4 เมื่อหาตัวที่มากที่สุดจะได้การ shift ไป 4 ตัวอักษร ดังรูปที่ 2.9



รูปที่ 2.9 ตัวอย่างการ shift ครั้งที่ 4

ครั้งที่ 5 ที่การเปรียบเทียบกับตัวอักษร C ซึ่งเมื่อดูจากตารางทั้งสองและ อัลกอริทึมจะได้ว่า bmGs[6] = 7 และ bmBc[C] – 7 + 6 = 5 เมื่อหาตัวที่มากที่สุดจะได้การ shift ไป 7 ตัวอักษร ดังรูปที่ 2.10 แต่เนื่องจากสิ้นสุดคำแล้วจึงจบการทำงานของอัลกอริทึม

รูปที่ 2.10 ตัวอย่างการ shift ครั้งที่ 5

จากการใช้อัลกอริทึมบอยเออร์-มัวร์ในการค้นหาข้อมูลครั้งนี้ จะมีการเปรียบเทียบตัวอักษรทั้งหมด 17 ครั้ง ในข้อมูลทั้งหมด 24 ตัวอักษร และคำสำคัญ 8 ตัวอักษร

2.3 Toward Faster String Matching for Intrusion Detection

เอกสารงานวิจัยชื่อ "Toward Faster String Matching for Intrusion Detection" (Jason, C. C. : 2001.) เป็นงานวิจัยของ C. Jason Coit , Stuart Staniford และ Joseph McAlerney. จาก Silicon Defense (www.silicondefense.com) เป็นงานวิจัยที่เป็นจุดเริ่มต้นของงานวิจัยนี้ โดยให้ข้อคิดเกี่ยวกับการพัฒนาระบบตรวจจับผู้บุกรุกทางให้มีประสิทธิภาพการทำงานที่เร็วขึ้น โดยงานวิจัยนี้สนใจเพียงการปรับเปลี่ยนโครงสร้างข้อมูลที่นำมาตรวจสอบกฎ โดยไม่สนใจการลด search space

ในงานวิจัยนี้ได้ศึกษาโครงสร้างข้อมูลที่ใช้สำหรับตรวจสอบกฎในโปรแกรม SNORT โดยผู้วิจัยทราบว่าโครงสร้างกฎมีลักษณะเป็นลิงค์ลิสต์สองมิติ ทำการค้นหาคำสำคัญโดยใช้อัลกอริทึม Boyer-moore และพบความซ้ำซ้อนในการทำงานของโปรแกรม SNORT ผู้วิจัยจึงแก้ไข

ความซ้ำซ้อนดังกล่าวโดยมีการนำเอาอัลกอริทึมชื่อว่า Aho-Corassick มาผสมกับอัลกอริทึม Boyer-moore แล้วนำมาใช้ในปัญหาของโปรแกรม SNORT

จากการทดสอบของผู้วิจัยโดยการรันโปรแกรม SNORT ที่ใช้อัลกอริทึม Boyer-moore กับโปรแกรม SNORT ที่มีการเปลี่ยนแปลงอัลกอริทึมในการค้นหา และโครงสร้างกฎบางส่วนนั้น ผลปรากฏว่าโปรแกรม SNORT ที่ใช้อัลกอริทึม Aho-Corassick ผสมกับ Boyer-moore มีการทำงานเร็วกว่าโปรแกรม SNORT เดิมถึง 3.22 เท่า และใช้พื้นที่หน่วยความจำเพิ่มขึ้นถึง 3 เท่าเช่นกัน

งานวิจัยดังกล่าวเป็นงานวิจัยที่เป็นจุดเริ่มต้นของงานวิจัยนี้ เนื่องจากเมื่อศึกษาการทำงานของอัลกอริทึมที่รวมเอา Aho-corassick และ Boyer-moore เข้าด้วยกันแล้ว พบว่าความซ้ำซ้อนในการคำนวณหาค่ากฎ ลดน้อยลง แต่ความซับซ้อนในส่วนของอัลกอริทึมกลับเพิ่มขึ้น จึงมีความสนใจว่า ผลลัพธ์ที่เร็วขึ้นนั้นควรจะมาจากโครงสร้างข้อมูลที่เปลี่ยนไปมากกว่าอัลกอริทึมในการค้นหา ในงานวิจัยนี้จึงนำเอาโครงสร้างกฎลักษณะเดียวกัน แต่มีประสิทธิภาพในการค้นหาได้ดีกว่ามาใช้ และจากการศึกษางานวิจัยนี้ ตอนท้ายของงานวิจัยยังให้บทสรุปเกี่ยวกับผลการทำงาน ว่าผลการทำงานที่เร็วขึ้นนั้น จะมีผลจากขนาดและจำนวนของกฎที่ใช้ด้วย จึงน่าจะมีวิธีการเพื่อลดขนาดของกฎ และข้อมูล เพื่อเพิ่มความเร็วในการทำงานได้เช่นกัน

2.4 Performance of Data Structures for Small Sets of Strings

เอกสารงานวิจัยชื่อ "Performance of Data Structures for Small Sets of Strings" (Heinz, S. and Zobel, J. 2002.) เป็นงานวิจัยของ Steffen Heinz และ Justin Zobel จาก School of Computer Science and Information Technology , RMIT University ประเทศออสเตรเลีย เป็นงานวิจัยเพื่อทดสอบประสิทธิภาพของโครงสร้างกฎแบบต่างๆ โดยโครงสร้างกฎที่นำมาทดสอบคือ binary search tree , splay tree , trie , burst trie , hash table ปัญหาที่นำมาใช้ในการทดสอบคือการทดสอบหาข้อมูลของเซตของคำเล็กๆ คล้ายๆ กับการสร้างอินเด็กซ์ในการทำ document processing ผลการทดสอบพบว่า burst trie เป็นโครงสร้างข้อมูลที่มีประสิทธิภาพสูงสุดสำหรับปัญหานี้

จากผลลัพธ์ของงานวิจัยนี้ ซึ่งเป็นการทดสอบปัญหาการค้นหาข้อมูลกับเซตของคำเล็กๆ ซึ่งเหมือนกับปัญหาของโปรแกรม SNORT จึงเลือกใช้ burst tries เป็นโครงสร้างข้อมูลของกฎแทนโครงสร้างกฎแบบเดิม ซึ่งเชื่อว่าโครงสร้างกฎแบบ burst tries นี้จะทำให้การทำงานของโปรแกรม SNORT ทำได้เร็วขึ้น

2.5 Burst Tries: A Fast, Efficient Data Structure for String Keys

เอกสารงานวิจัยชื่อ "Burst Tries: A Fast, Efficient Data Structure for String Keys" (Heinz, S. 2001.) เป็นงานวิจัยของ Steffen Heinz , Justin Zobel และ Hugh E. Williams จาก

School of Computer Science and Information Technology , RMIT University โดยงานวิจัยนี้เป็นการนำเสนอโครงสร้างข้อมูลแบบใหม่ที่เรียกว่า burst tries ที่ปรับปรุงจากโครงสร้างข้อมูลแบบ trie ซึ่งมีลักษณะโครงสร้างประกอบด้วย 3 ส่วนหลักๆ คือ

1. record

เป็นส่วนของโครงสร้างข้อมูลแบบ burst tries ที่เก็บข้อมูลสำหรับใช้ในการค้นหา ซึ่งแล้วแต่ว่าจะนำ burst trie ไปใช้กับงานใดโดย string ที่เก็บอยู่จะต้องไม่ซ้ำกัน ในส่วนโครงสร้างนี้จะมีองค์ประกอบอีกส่วนหนึ่งคือพอยเตอร์ที่ชี้ไปยัง container ที่เก็บ record ซึ่งอาจจะมีหรือไม่มีก็ได้ แล้วแต่ว่าจำเป็นจะต้องใช้งานหรือไม่

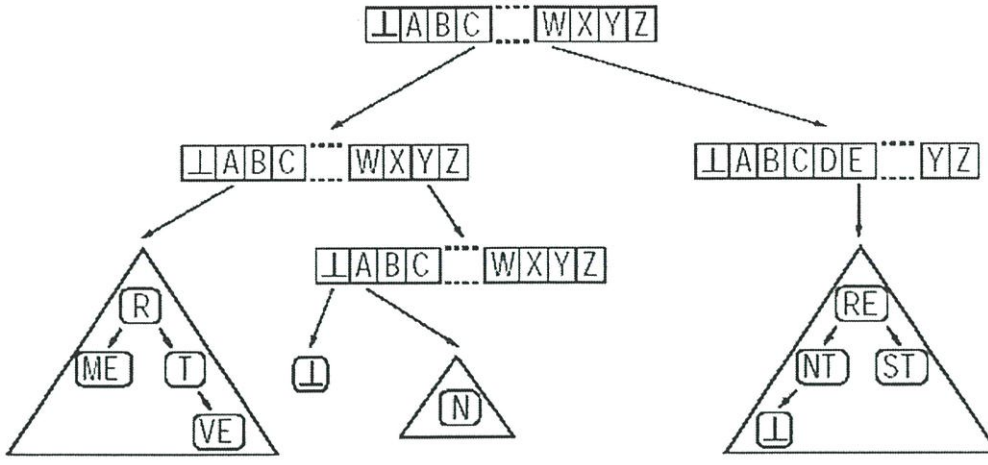
2. container

เป็นเซตของ record ซึ่งมีลักษณะคล้ายกับลิงค์ลิสต์หรือไบนารี่ทรี สำหรับ container ที่มีความลึก K นั้น หมายถึงความยาวของ string จะต้องยาวอย่างน้อย K ตัวอักษร และตัวอักษร K ตัวแรกจะต้องไม่ซ้ำกัน แต่ละ container จะมี header สำหรับเก็บสถิติการใช้งานสำหรับการทำงานในลักษณะ heauristics for bursting ด้วย ดังนั้น container ที่มีความลึกเป็น 3 นั้นจะสามารถเก็บสตริงค์ “auther” ,” automated” และ “autopsy” ได้ แต่ไม่สามารถเก็บสตริงค์ “auger” ได้

3. access trie

acces trie ซึ่งเป็นส่วนของโครงสร้างที่มีลักษณะคล้ายกับ standard trie มีหน้าที่สำหรับเป็นโครงข่ายหลักในการเชื่อมต่อกับโหนดปลายสุดของโครงสร้างข้อมูล แต่ละโหนดประกอบด้วยอาเรย์ของพอยเตอร์ที่มีขนาดเท่าๆ กับจำนวนตัวอักษรทั้งหมด พอยเตอร์แต่ละพอยเตอร์จะชี้ไปที่โหนดอื่นๆ ของ trie และจะมีพอยเตอร์พิเศษอีกหนึ่งอันคือ empty-string พอยเตอร์ ที่จะชี้ไปยังเรคอร์ด

รูปที่ 2.11 เป็นตัวอย่างของโครงสร้างแบบ burst trie ที่เก็บเรคอร์ด 10 เรคอร์ดซึ่งมีข้อมูลคือ “came” , “car” , “cat” , “cave” , “cy” , “cyan” , “we” , “went” , “were” และ “west” ในตัวอย่างนี้เซตของตัวอักษรคือตัวอักษรตั้งแต่ A ถึง Z และ empty-string (⊥) สำหรับโครงสร้างใน container นั้นใช้ Binary Search Tree ในรูปนี้ access trie มีโหนด 4 โหนด และความลึกที่มากที่สุดคือ 3 ชั้น container ที่อยู่ซ้ายมือสุดมี 4 เรคอร์ด สำหรับเก็บสตริงค์ “came” , “car” , “cat” และ “cave” มีสตริงค์ที่อยู่ใน container ทางด้านขวาที่มีสัญลักษณ์เป็น “⊥” หมายถึงสตริงค์ “we” สำหรับ string “cy” นั้นจะถูกเก็บอยู่ใน access trie ซึ่งในโครงสร้างจะถูกแสดงอยู่ในรูปของพอยเตอร์ที่ช่อง empty-string ชี้ไปยังเรคอร์ดที่ถูกอินเด็กซ์โดยสัญลักษณ์ของ empty-string



รูปที่ 2.11 ตัวอย่าง burst trie ที่มี container เป็นไบนารีทรี

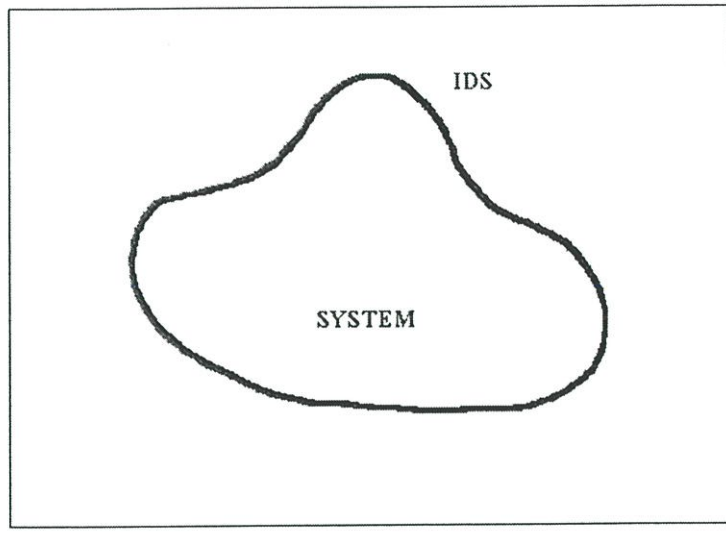
บทที่ 3

Intrusion Detection System

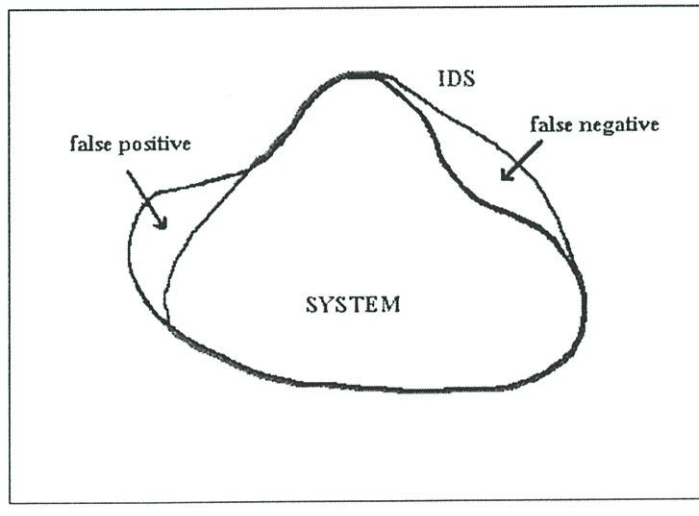
ปัจจุบันมีการใช้งานระบบคอมพิวเตอร์อย่างแพร่หลาย หลายๆ บริษัท หรือหน่วยงานต่างๆ ได้นำระบบคอมพิวเตอร์มาใช้เพื่อพัฒนาศักยภาพการทำงานของตนให้มากขึ้น การทำงานของระบบคอมพิวเตอร์จะทำงานอย่างต่อเนื่อง และมีการออนไลน์ให้คนอื่นๆ เข้ามาใช้งานระบบบางส่วนด้วย ซึ่งปัญหาที่เกิดขึ้นตามมาก็คือปัญหาผู้บุกรุกเข้ามาสร้างความเสียหายให้กับระบบ และอาจเข้ามาเพื่อขโมยข้อมูลที่สำคัญไป ปัญหาดังกล่าวจะเป็นปัญหากับผู้ดูแลระบบอย่างมาก เนื่องจากผู้ดูแลระบบต้องคอยป้องกันและแก้ไขปัญหาต่างๆ อยู่เสมอๆ การทำงานของผู้ดูแลระบบนี้จะหนักมากหรือน้อยก็ขึ้นอยู่กับขนาดของระบบว่ามีขนาดใหญ่ และซับซ้อนมากน้อยเพียงใด ยิ่งระบบที่มีความซับซ้อนสูง มีความยุ่งยากในการดูแลมาก มีการออนไลน์ใช้งานอยู่ตลอดเวลา ผู้ดูแลระบบก็คงต้องอยู่ดูแลระบบตลอดเวลาซึ่งในทางปฏิบัติแล้วเป็นไปไม่ได้แน่นอน อีกทั้งปัญหาบางปัญหาผู้ดูแลระบบไม่สามารถตรวจสอบด้วยตาของตัวเองได้เลย ปัญหาการดูแลระบบจึงยุ่งยากขึ้นทุกวัน ทางออกที่จะช่วยแก้ปัญหาเหล่านี้ ก็คือการหาผู้ช่วยมาช่วยดูแลระบบตลอดเวลา สามารถมองเห็นในสิ่งที่ผู้ดูแลระบบมองไม่เห็น คอยแจ้งเตือนให้กับผู้ดูแลระบบยามมีเหตุการณ์ผิดปกติใดๆ และในบางครั้งก็สามารถแก้ไขปัญหาง่ายๆ ได้ด้วยตัวเองด้วย ผู้ช่วยที่สามารถช่วยแบ่งเบาภาระของผู้ดูแลระบบได้อย่างมากก็คือ ระบบตรวจจับผู้บุกรุก หรือ Intrusion Detection System (Bace, R. G. 2000. : ธนัญชัย ตรีภาค. 2545.)

3.1 ความหมาย Intrusion Detection System

Intrusion Detection System คือ ระบบตรวจจับสัญญาณของความผิดปกติต่างๆ ที่เกิดขึ้นในระบบที่อยู่ในขอบเขตที่ระบบนี้มีหน้าที่ตรวจสอบ โดยทั่วไปจะหมายถึงระบบที่ตรวจจับความผิดปกติในบริษัท เขตทหาร โรงงาน หรือในเขตที่พักอาศัย แต่ในที่นี้เราจะหมายถึงโปรแกรมที่ใช้สำหรับตรวจจับความผิดปกติในระบบคอมพิวเตอร์ และระบบเครือข่าย เท่านั้น โดยตัวโปรแกรมจะมีความสามารถในการตรวจจับสัญญาณของความผิดปกติทุกๆ อย่างที่เกิดขึ้นในระบบ ไม่ว่าจะเป็นภายในระบบคอมพิวเตอร์ ระบบปฏิบัติการ โปรแกรมที่รันอยู่ในเครื่อง การทำงานกับฐานข้อมูล หรือแม้แต่ข้อมูลที่วิ่งผ่านไปมาในเครือข่ายด้วย



รูปที่ 3.1 ระบบ Intrusion Detection System



รูปที่ 3.2 ขอบเขตที่เหลื่อมกันของ IDS กับระบบ ทำให้เกิด false positive และ false negative

จากรูปที่ 3.1 ถ้าเรามองระบบเป็นเซตของการทำงานเซตหนึ่ง ระบบตรวจจับผู้บุกรุกที่แท้จริง (Ideal IDS) ต้องทราบของเขตของเซตของระบบ ต้องทราบว่าระบบทำงานอะไรบ้าง การทำงานไหนปกติและผิดปกติ โดยระบบตรวจจับผู้บุกรุกที่มีประสิทธิภาพ จะทราบขอบเขตของระบบโดยไม่มีการเหลื่อมล้ำเข้าไปในระบบ หรือเหลื่อมล้ำออกนอกระบบ อย่างเด็ดขาด

แต่ในระบบที่ใช้งานในโลกของความเป็นจริง กรอบของระบบที่ IDS รับรู้อาจมีความเหลื่อมล้ำกันระบบที่ IDS ต้องตรวจสอบ ทำให้เกิดความผิดพลาดในการตรวจสอบได้ ซึ่งเราสามารถแบ่งความผิดพลาดในการตรวจสอบได้เป็นสองลักษณะคือ False Positive และ False Negative ดังรูปที่ 3.2 ซึ่งความผิดพลาดทั้งสองแบบมีรายละเอียดคือ

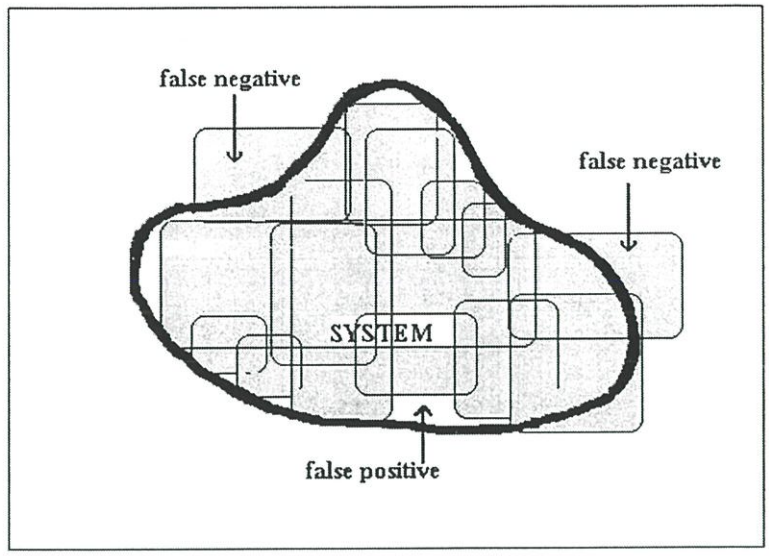
1. False Positive คือความผิดพลาดอันเนื่องมาจากการเกิดเหตุการณ์ซึ่งเป็นเหตุการณ์ปกติในระบบ แต่เป็นเหตุการณ์ที่ไม่ได้อยู่ในกรอบของระบบที่ IDS รับรู้ว่าเป็นปกติ จึงทำให้ IDS คิดว่าเกิดเหตุการณ์ผิดปกติเกิดขึ้น ผลลัพธ์ก็คือ IDS จะแจ้งเตือนผู้ดูแลระบบว่าเกิดเหตุการณ์นั้นๆ ขึ้น แต่ก็เป็นการแจ้งเตือนเหตุการณ์ที่เป็นปกติ
2. False Negative คือความผิดพลาดอันเนื่องมาจากการเกิดเหตุการณ์ที่ผิดปกติเกิดขึ้น แต่เป็นเหตุการณ์ที่อยู่ในกรอบของระบบที่ IDS รับรู้ว่าเป็นเหตุการณ์ปกติในระบบ จึงทำให้ IDS ไม่แจ้งเตือนความผิดปกติเกิดขึ้น ถึงแม้ว่าจะเป็นเหตุการณ์ที่ผิดปกติก็ตาม

โดยการออกแบบระบบ IDS นั้นจะพยายามออกแบบ และใช้วิธีการต่างๆ มากมายที่ทำให้ False Positive และ False Negative มีน้อยที่สุดซึ่งก็มีงานวิจัยหลายๆ ชิ้นที่ทำเพื่อลดพื้นที่ตรงส่วนนี้ ยกตัวอย่างเช่น

1. An Immunological Model of Distributed Detection and Its Application to Computer Security. (Hofmeyr, S. A. : 1999.)
2. A Data Mining and CIDF Based Approach for Detecting Novel and Distributed Intrusions. (Lee, W. et. al. : 2000.)
3. A Sense of Self for Unix Processes. (Forrest, S. et. al. : 1996.)
4. Data mining approaches for intrusion detection. (Lee, W. and Stolfo, S. : 1998.)
5. Artificial Neural Networks for Misuse Detection. (Cannady, J. : 1998.)
6. Network Security Monitor. (Heberlein, L. et. al. 1990 : Heberlein, L.T. 2001.)
7. Bro: A System for Detecting Network Intruders in Real-Time. (Paxson, V. : 1998)
8. Intrusion Detection with Neural Networks (Ryan, J. et. al. : 1998.)

3.2 ประเภทของระบบตรวจจับผู้บุกรุก

จากการออกแบบระบบตรวจจับผู้บุกรุกนั้น เราสามารถแบ่งระบบตรวจจับผู้บุกรุกได้สองรูปแบบคือ ระบบที่ตรวจตราหาการทำงานที่ผิดไปจากการทำงานปกติระบบเรียกว่า Anormaly Detection ซึ่งเป็นเหมือนกับการตรวจจับคนที่ไม่มีสิทธิทำงานอยู่ในระบบ อีกรูปแบบหนึ่งของ IDS คือระบบที่ตรวจหาการทำงานที่ไม่ควรเกิดขึ้นในระบบเรียกว่า Misuse Detection ในที่นี้ก็เปรียบเสมือนเป็นบุคคลที่มีสิทธิในระบบ สามารถเข้าออกในระบบได้ แต่เป็นผู้ที่ทำในสิ่งที่ระบบไม่อนุญาตให้ทำ หรือทำการใดๆ ที่อยู่นอกเหนือสิทธิของตนในระบบ



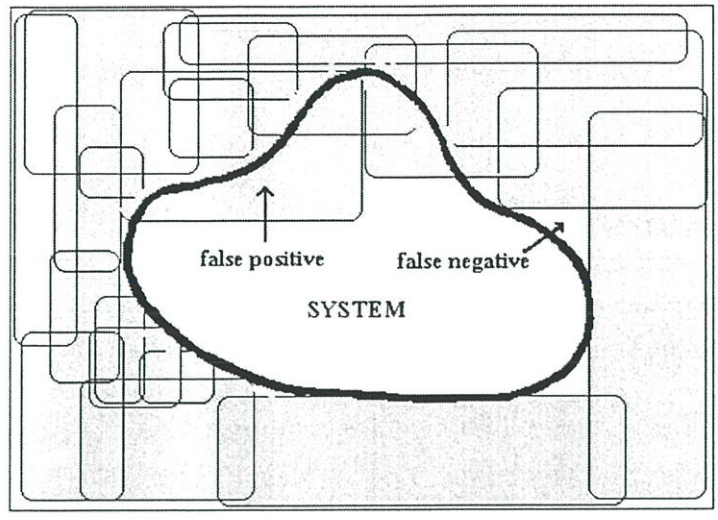
รูปที่ 3.3 Anomaly Detection Model

3.2.1 Anormaly Detection

แนวความคิดของการทำ Anomaly Detection คือการหาเซตของการทำงานที่เป็นปกติย่อยๆ ขึ้นมาแล้วนำมารวมกันเพื่อให้ระบบ IDS ทราบข้อมูลของเซตการทำงานที่เป็นปกติทั้งหมดในระบบ หลังจากนั้นเมื่อให้ระบบ IDS ทำงาน ถ้าเกิดกรณีที่ IDS ตรวจจับการทำงานที่ไม่ได้อยู่ในเซตของการทำงานที่เป็นปกติ ระบบ IDS จะแจ้งเตือนต่อผู้ดูแลระบบทันที สำหรับการสร้างขอบเขตของระบบนั้น อาจสร้างได้โดยการหาข้อมูลการทำงานที่เป็นปกติในระบบขึ้นมา โดยเอาข้อมูลการทำงานของผู้ใช้งานแต่ละคน เวลาที่มีการใช้งาน ทรัพยากรที่ผู้ใช้งานคนนั้นๆ มักจะใช้บ่อยๆ หรือแม้กระทั่งข้อมูลในระบบ หรือในเครือข่ายก็สามารถนำมาสร้างเป็น เซตของระบบได้เช่นกัน ดังตัวอย่างรูปที่ 3.3 ในการหาเซตของการทำงานที่เป็นปกติทั้งหมด อาจเกิดการผิดพลาดขึ้นมา ทำให้เกิดลักษณะของ false positive และ false negative ขึ้นมาได้เช่นกัน

3.2.2 Misuse Detection

เป็นแนวความคิดที่ตรงข้ามกับ Anomaly Detection คือ การทำ IDS รูปแบบนี้จะใช้ข้อมูลของการทำงานที่ผิดปกติต่างๆที่เคยเกิดขึ้นมาแล้ว สร้างเป็นฐานข้อมูลของการทำงานที่ผิดปกติให้ระบบ IDS จดจำไว้ ในการทำงานของ IDS ที่มีการทำงานแบบ Misuse จะนำข้อมูลที่อยู่ในระบบมาค้นหาในฐานข้อมูลว่ามีอยู่หรือไม่ ถ้าระบบ IDS มีข้อมูลของการทำงานรูปแบบนั้นๆ อยู่ ก็แสดงว่าเกิดความผิดปกติขึ้นแล้ว ดังรูปที่ 3.4 มีการเก็บเซตของการทำงานที่ผิดปกติแล้วสร้างเป็นขอบเขตของการทำงานที่ผิดปกติ แต่ในการรวบรวมนี้อาจรวมเอาการทำงานที่เป็นปกติเข้าไปด้วย ทำให้เกิด false positive หรือในบางกรณีที่ไม่ได้เก็บข้อมูลความผิดปกติไว้ก็ทำให้เกิดกรณีของ false negative ได้เช่นกัน ซึ่งในการทำงานของ Misuse Detection นี้ จะมีข้อเสียคือจะไม่สามารถตรวจจับการบุกรุกชนิดใหม่ๆ ได้ เนื่องจากต้องมีข้อมูลของการบุกรุกอยู่ก่อน จึงจะตรวจจับได้

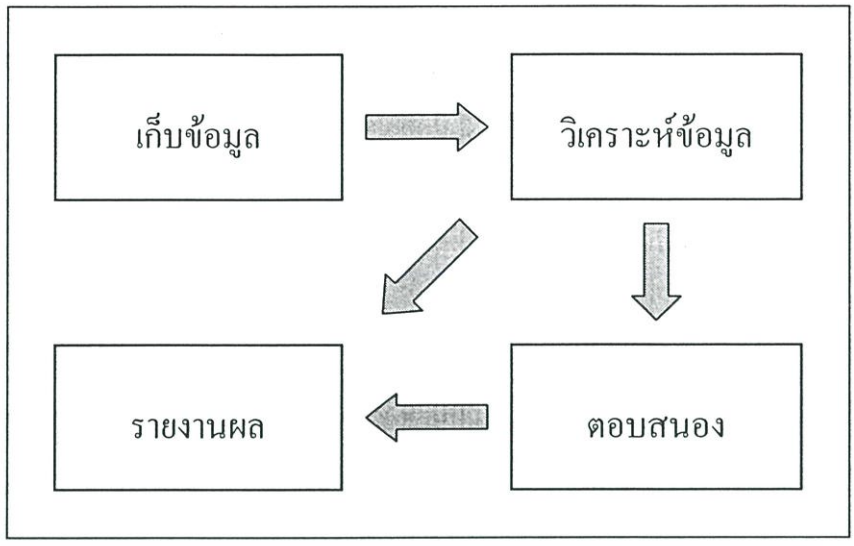


รูปที่ 3.4 Misuse Detection Model

3.3 การทำงานของ Intrusion Detection System

ระบบตรวจจับผู้บุกรุกแต่ละแบบจะมีหน้าที่การทำงานที่แตกต่างกันออกไป บางตัวจะตรวจจับความผิดปกติในระบบเครือข่าย บางตัวจะตรวจจับความผิดปกติในระบบฐานข้อมูล แต่โดยการทำงานทั้งหมดแล้วเราสามารถสรุปการทำงานของระบบตรวจจับผู้บุกรุกได้ 3 ขั้นตอนคือ การเก็บข้อมูลระบบ การวิเคราะห์ข้อมูลที่เก็บได้ และการรายงานผลการทำงานให้ผู้ดูแลระบบหรือผู้ที่เกี่ยวข้องทราบ

ในการทำงานหลักๆ ของระบบตรวจจับผู้บุกรุก อาจมีขั้นตอนเสริมอยู่ขั้นตอนหนึ่ง คือการตอบสนองต่อการบุกรุกนั้นๆ การทำงานในขั้นตอนนี้จะใช้ในกรณีที่การบุกรุกเป็นรูปแบบการบุกรุกที่ระบบตรวจจับผู้บุกรุกสามารถแก้ไขด้วยตัวเองได้ ซึ่งในบางระบบอาจไม่มีการทำงานในส่วนนี้ ระบบที่ไม่มีการตอบสนองต่อการบุกรุก ส่วนใหญ่จะเป็นระบบที่ไม่ได้ทำงานแบบตอบสนองทันที (realtime) คือจะเก็บข้อมูลของระบบไว้ก่อน แล้วจึงวิเคราะห์ข้อมูลภายหลัง เมื่อทำการวิเคราะห์ข้อมูลแล้วพบว่ามีการบุกรุกเข้าสู่ระบบก็จะทำการแจ้งเตือนในขั้นตอนการทำรายงานผลการทำงาน ระบบตรวจจับผู้บุกรุกที่ไม่มีการตอบสนองต่อการบุกรุก ก็มักใช้ในงานที่ไม่มีความสำคัญมากนัก แต่ต้องการความถูกต้องสูง โดยลำดับการทำงานของระบบตรวจจับผู้บุกรุกสามารถมองเป็นขั้นตอนต่างๆ ได้ดังรูปที่ 3.5



รูปที่ 3.5 การทำงานของ intrusion detection system

3.3.1 การเก็บข้อมูลในระบบ

จากที่ได้กล่าวมาแล้วว่าระบบตรวจจับผู้บุกรุกจะมีการทำงานที่แตกต่างกันไป หน้าที่ในการเก็บข้อมูลของระบบที่ต้องการตรวจสอบก็แตกต่างกันไปตามหน้าที่ของระบบตรวจจับผู้บุกรุกด้วย โดยเราสามารถแบ่งการเก็บข้อมูลของระบบที่ต้องการตรวจสอบออกเป็นกลุ่มต่างๆ ได้ 4 กลุ่มด้วยกันคือ มีการเก็บข้อมูลในชั้นแอปพลิเคชั่น (Application-based Approach) เพื่อนำมาตรวจสอบการทำงานของแอปพลิเคชั่นต่างๆ ว่าผิดปกติหรือไม่, การเก็บข้อมูลของการทำงานของเครื่อง (Host-based Approach) เพื่อนำมาตรวจสอบการทำงานของระบบของเครื่องที่ใช้งานอยู่, การเก็บข้อมูลการเปลี่ยนแปลงข้อมูลในระบบ (Target-based Approach) เพื่อนำมาตรวจสอบว่าข้อมูลมีการเปลี่ยนแปลงอย่างไร และการเก็บข้อมูลเครือข่าย (Network-based Approach) เพื่อนำมาตรวจสอบว่ามีการบุกรุกทางระบบเครือข่ายหรือไม่ อย่างไร

3.3.1.1 การเก็บข้อมูลในชั้นแอปพลิเคชั่น

การเก็บข้อมูลในชั้นแอปพลิเคชั่นนั้น เป็นการเก็บข้อมูลที่โปรแกรมต่างๆ สร้างขึ้นมาเพื่อรายงานผลการทำงานของโปรแกรมนั้นๆ เช่น log file หรือ error message ต่างๆ ของเว็บเซิร์ฟเวอร์, ไฟร์วอลล์ หรือโปรแกรมบริหารฐานข้อมูล รวมถึงข้อมูลของการทำงานตอบสนองกันระหว่างผู้ใช้งานโปรแกรม และข้อมูลที่เกี่ยวข้อง ในการเก็บข้อมูลในลักษณะนี้ นอกจากเป็นการทำงานด้านการรักษาความปลอดภัยในระบบแล้ว ยังช่วยในการวิเคราะห์ระบบและปรับปรุงระบบเนื่องจากผลจากการวิเคราะห์ข้อมูลที่ได้ทำให้ทราบว่าการใช้งานโปรแกรมไหนในระบบมีมากน้อยอย่างไร และควรให้ความสำคัญกับการทำงานตรงส่วนไหน แต่การทำงานในส่วนนี้ก็ยังมีข้อเสีย ในกรณีที่มีผู้บุกรุกแล้วทำการเปลี่ยนแปลงข้อมูลดังกล่าว ทำให้การตรวจจับทำไม่ได้ ดังนั้นจึงควรเก็บข้อมูลดังกล่าวไว้ในที่ๆ ปลอดภัยด้วย

3.3.1.2 การเก็บข้อมูลของการทำงานของเครื่อง

สำหรับการเก็บข้อมูลของการทำงานของเครื่อง จะเน้นไปในการเก็บข้อมูลของระบบปฏิบัติการเป็นหลัก ข้อมูลที่เก็บได้จะอยู่ในรูปของการแจ้งเตือนในระบบเช่นการตั้งค่าบางอย่างไม่สมบูรณ์ การทำงานของโพรเซสบางโพรเซสมีปัญหาหรือปัญหาของฮาร์ดแวร์เป็นต้น หรืออาจอยู่ในรูปข้อมูลของการทำงานโดยปกติของระบบปฏิบัติการนั้นๆ เช่นข้อมูลการใช้งานของยูสเซอร์แต่ละคน ใคร ทำอะไร เมื่อเวลาเท่าไร ซึ่งเมื่อนำข้อมูลเหล่านี้ไปวิเคราะห์แล้ว จะได้ผลการวิเคราะห์ในลักษณะ มีการใช้งานอย่างไม่ถูกต้องหรือไม่ ถ้ามี ใครเป็นผู้ใช้งานนั้นๆ เมื่อเวลาเท่าไร จากที่ไหน ข้อดีอีกข้อหนึ่งก็คือการเก็บข้อมูลในลักษณะนี้สามารถเก็บข้อมูลที่ถูกเข้ารหัสได้ด้วย ส่วนข้อเสียของการเก็บข้อมูลการทำงานของเครื่องก็คือข้อมูลที่ได้มักจะมีขนาดใหญ่ ระบบที่ทำการเก็บข้อมูลลักษณะนี้จะมี overhead สูงขึ้น นอกจากนี้โปรแกรมที่ทำการเก็บข้อมูลและวิเคราะห์ข้อมูลยังขึ้นอยู่กับ platform และมีราคาสูงมากด้วย

3.3.1.3 การเก็บข้อมูลการเปลี่ยนแปลงข้อมูลในระบบ

การเก็บข้อมูลการเปลี่ยนแปลงข้อมูลในระบบจะใช้หลักการของ integrity analysis ในการตรวจสอบการเปลี่ยนแปลงข้อมูลต่างๆ ในระบบ ในระบบตรวจจับผู้บุกรุกบางระบบจะใช้ checksum เป็นตัวบ่งบอกการเปลี่ยนแปลงในระบบ การวิเคราะห์ลักษณะนี้จะเริ่มจากการสร้างฐานข้อมูล signature ของไฟล์ต่างๆ ในระบบปกติไว้ เมื่อระบบมีทำงานก็จะทำการตรวจสอบค่า signature นี้ไปเรื่อยๆ อาจเป็นวันละครั้ง สองครั้ง หรือบ่อยกว่านั้นแล้วความสำคัญของระบบ เมื่อมีข้อมูลไฟล์ไหนมีการเปลี่ยนแปลงก็จะทราบได้ ข้อดีของการทำ integrity analysis ลักษณะนี้ช่วยให้การตรวจจับการบุกรุกที่มีการเปลี่ยนแปลงระบบเช่น ทำการเจาะระบบแล้วทำการวางโทรจัน หรือ Back Door ไว้ได้ สำหรับการแก้ไขเมื่อทราบว่ามีผู้บุกรุกมาเปลี่ยนแปลงไฟล์ข้อมูลในระบบก็ทำการแก้ไขเฉพาะไฟล์ที่ถูกแก้ไขเท่านั้น ไม่จำเป็นต้องทำการติดตั้งระบบใหม่ แต่ระบบนี้ก็มีข้อเสียในกรณีที่เมื่อมีไฟล์ในระบบเยอะมาก การเก็บข้อมูล signature ของไฟล์ต่างๆและการวิเคราะห์ข้อมูลก็จะใช้เวลานาน ระบบนี้จึงไม่เหมาะในการทำงาน real time เพราะทำให้เกิด overhead ในระบบสูงมาก

3.3.1.4 การเก็บข้อมูลเครือข่าย

การเก็บข้อมูลเครือข่ายนั้นนับวันจะมีความสำคัญขึ้นเรื่อยๆ เพราะการบุกรุกทางเครือข่ายมีมากขึ้นเรื่อยๆ การเก็บข้อมูลแบบนี้จะใช้การดักจับข้อมูลที่ผ่านไปมาในเครือข่าย โดยการทำให้เน็ตเวิร์คการ์ดอยู่ใน promiscuous mode เมื่อเน็ตเวิร์คการ์ดอยู่ในโหมดดังกล่าว จะสามารถรับข้อมูลทุกอย่างที่อยู่ในเครือข่ายได้ การเก็บข้อมูลเครือข่ายในลักษณะนี้สามารถตรวจจับการโจมตีทางเครือข่ายได้เช่นการทำ SYN flood การทำ port scan หรือการส่งแพ็กเก็ตปริมาณมากมารบกวนในระบบ แต่เนื่องจากการเก็บข้อมูลเครือข่ายนี้ใช้ลักษณะการทำสนิฟเฟอร์เป็นหลัก จึงไม่สามารถ

ทำงานในเครือข่ายที่เป็นเครือข่ายสวิชชิ่ง ไม่สามารถทำงานในระบบเครือข่ายที่เข้ารหัสข้อมูล หรือไม่สามารถเก็บข้อมูลในเครือข่ายที่มีข้อมูลหนาแน่นได้เพราะการทำงานในการเก็บข้อมูลอาจไม่เร็วพอที่จะเก็บข้อมูลทั้งหมดที่ผ่านไปมาในระบบได้ ข้อเสียอีกข้อหนึ่งของการเก็บข้อมูลนี้ก็คือข้อมูลที่เก็บมีขนาดใหญ่มาก โดยเฉพาะอย่างยิ่งในระบบเครือข่ายที่มีการรับส่งแพ็กเก็ตปริมาณมากอยู่ตลอดเวลา

นอกจากนี้ยังมีการเก็บข้อมูลโดยทำการเก็บข้อมูลทั้ง Application-based , Host-based และ Network-based ร่วมกันด้วย เพื่อให้ได้ข้อมูลระบบอย่างครบถ้วน และใช้ข้อมูลจากทั้งสามแหล่งมาประกอบกันในการวิเคราะห์ความผิดปกติที่เกิดขึ้นในระบบด้วย การเก็บข้อมูลในลักษณะนี้เรารวมเรียกว่า "Integrated – based"

3.3.2 การวิเคราะห์ข้อมูลระบบ

เมื่อได้ข้อมูลของระบบที่จำเป็นแล้ว ในขั้นตอนต่อมาเราก็จะนำเอาข้อมูลที่ได้มาวิเคราะห์ว่าระบบของเรามีความผิดปกติเกิดขึ้นหรือไม่ การวิเคราะห์ข้อมูลเราสามารถแบ่งการทำงานตามรูปแบบการวิเคราะห์ข้อมูลได้ 2 รูปแบบคือ ทำการวิเคราะห์ในขณะที่เก็บข้อมูล(Real Time) หรือจะเก็บข้อมูลทั้งหมดไว้ก่อน แล้วจึงวิเคราะห์ข้อมูลนั้นๆ ภายหลัง(Batch) ในการวิเคราะห์ข้อมูลทั้งสองรูปแบบก็มีข้อดีข้อเสียแตกต่างกันไป

3.3.2.1 การวิเคราะห์ในขณะที่เก็บข้อมูล(Real Time)

ในการวิเคราะห์ข้อมูลที่ได้ในขณะที่เก็บข้อมูล หรือแบบ Real Time นั้น ระบบจะจัดเก็บข้อมูล วิเคราะห์ข้อมูล และรายงานผลการวิเคราะห์ ในช่วงเวลาเดียวกัน เมื่อเกิดข้อผิดพลาดขึ้นสามารถตอบสนองได้ทันท่วงที ระบบที่ทำงานแบบ Real Time มีการแจ้งเตือนหลายๆ แบบ เช่น E-mail หรือ Instant Messaging ให้กับผู้ดูแลระบบได้ในช่วงเวลาที่มีการบุกรุกได้ ในการตรวจสอบระบบแบบ Real Time ทำให้ระบบสามารถตรวจสอบข้อผิดพลาดได้อย่างรวดเร็ว แต่ก็ขึ้นอยู่กับความเร็วในการวิเคราะห์ข้อมูล ด้วย ถ้าข้อมูลมีความซับซ้อนมากๆ ก็จะใช้เวลามากตามเช่นเดียวกัน เมื่อระบบทำการตรวจสอบการบุกรุกได้ในขณะที่เพิ่งเกิดการบุกรุกขึ้น ผู้ดูแลระบบ หรือ ระบบตรวจจับผู้บุกรุกเองสามารถแก้ไขปัญหาที่เกิดขึ้นได้ทันที แต่ทั้งนี้ก็ขึ้นอยู่ความเร็วในการวิเคราะห์ข้อมูล และชนิดของปัญหาว่ามีความยุ่งยากในการแก้ปัญหามากน้อยเพียงไรด้วย

ระบบ Real Time ทำงานได้อย่างรวดเร็ว แต่การทำงานที่รวดเร็วดังกล่าวก็ต้องแลกกับการใช้หน่อยความจำปริมาณมาก และการประมวลผลที่รวดเร็วมากด้วย อีกทั้งการตอบสนองต่อการบุกรุกโดยอัตโนมัติ อาจทำให้เกิดความเสียหายกับระบบมากกว่าเดิม เพราะว่าในบางครั้ง การทำงานที่รวดเร็วเกินไปของระบบนี้ ทำให้เกิดเกิดความผิดพลาดในการวิเคราะห์ข้อมูลจนประมวลผลการทำงานปกติ กลายเป็นการทำงานที่ผิดปกติ แล้วทำการแก้ไขตามข้อมูลที่มีอยู่ ก็ยิ่งทำให้ระบบมีความเสียหายมากกว่าเดิม ระบบตรวจจับผู้บุกรุกที่ทำงานแบบ Real Time จึงเหมาะ

กับระบบที่มีข้อมูลที่ต้องพิจารณาน้อย ต้องการการรายงานอย่างรวดเร็วเมื่อผิดปกติ และข้อมูลที่ต้องนำมาวิเคราะห์ไม่ซับซ้อนมากนัก

3.3.2.2 การวิเคราะห์ข้อมูลภายหลังจากที่เก็บข้อมูล(Batch)

อีกรูปแบบหนึ่งในการวิเคราะห์ระบบที่ใช้กันคือ การวิเคราะห์ข้อมูลภายหลังจากที่เก็บข้อมูลไว้แล้วหรือการทำงานแบบ Batch การทำงานในแบบนี้เหมาะกับงานที่ไม่จำเป็นต้องตอบสนองทันทีเมื่อเกิดความผิดปกติขึ้น แต่ให้มีการบันทึก และรายงานว่าเกิดความผิดปกติขึ้น การทำงานจะใช้หน่วยความจำ และการประมวลผลน้อยกว่าแบบแรก แต่ก็ใช้เนื้อที่ในการเก็บข้อมูลมากกว่าแบบแรกแน่นอน เหมาะกับองค์กรที่มีบุคลากรจำกัด ข้อเสียของการทำงานแบบ Batch คือมักแก้ปัญหาที่เกิดขึ้นไม่ทัน เพราะกว่าจะทราบว่าเกิดปัญหาขึ้น ปัญหานั้นก็เกิดขึ้นนานมาก ความเสียหายที่เกิดขึ้นก็แก้ไขได้ยาก

ไม่ว่าจะเป็นการวิเคราะห์ระบบจะเป็นแบบ Real Time หรือเป็นแบบ Batch ก็จะมีวิธีการวิเคราะห์ระบบที่เหมือนกัน คือ ทำการหารูปแบบของการโจมตีในข้อมูลที่ได้รับมา (Signature Analysis) , วิธีการวิเคราะห์ทางสถิติ (Statistical Analysis) และวิธีการตรวจสอบการเปลี่ยนแปลงของระบบ (Integrity Analysis)

1) การหารูปแบบของการโจมตี (Signature Analysis)

ในวิธีการวิเคราะห์ระบบแบบ Signature Analysis เป็นการวิเคราะห์ข้อมูลโดยการหาสัญญาณของการโจมตี (Attack Signature) การทำงานจะทำโดยการเปรียบเทียบรูปแบบของข้อมูลกับรูปแบบของการโจมตีในฐานข้อมูล ว่ามีความคล้ายคลึงกันหรือไม่ ถ้ามีความคล้ายคลึงกันก็แสดงว่ามีการโจมตีเกิดขึ้นแล้ว ในการเปรียบเทียบอาจเป็นแบบอย่างง่ายคือการเปรียบเทียบข้อมูลว่ามีความเข้ากันได้กับข้อมูลของการโจมตีเพียงใด หรือเป็นแบบที่มีความสลับซับซ้อนขึ้นอีกเช่น การทำ state transition เป็นต้น

สำหรับโปรแกรมตรวจจับผู้บุกรุกที่มีจำหน่ายในท้องตลาด ส่วนใหญ่จะทำงานในลักษณะของการเปรียบเทียบรูปแบบของระบบกับการโจมตีในฐานข้อมูล ซึ่งบริษัทผู้ขายจะให้ฐานข้อมูลของการโจมตีไว้ด้วย ผู้ใช้งานสามารถป้อนข้อมูลการโจมตีหรือกฎในการตรวจจับได้ด้วย ซึ่งโปรแกรมในลักษณะนี้มักจะมีการอัพเดทข้อมูลในฐานข้อมูลบ่อยๆ เพื่อเพิ่มความสามารถในการวิเคราะห์ข้อมูลในระบบ การวิเคราะห์ระบบด้วยวิธี signature analysis นี้จะมี overhead ไม่มากนัก เพราะเป็นเพียงการเปรียบเทียบข้อมูลกับข้อมูลในฐานข้อมูลเท่านั้น และยังเพิ่มความรวดเร็วในการทำงานโดยรวมมากขึ้น เพราะสามารถนำข้อมูลในฐานข้อมูลมาเป็นกฎในการกรองข้อมูลที่จะเก็บให้น้อยลงได้ด้วย แต่วิธีการนี้ก็มีข้อเสียเพราะฐานข้อมูลจะมีขนาดใหญ่ขึ้นเรื่อยๆ ต้องมีการอัพเดทฐานข้อมูลบ่อยๆ

2) วิธีการวิเคราะห์ทางสถิติ (Statistical Analysis)

วิธีการวิเคราะห์ทางสถิติ (Statistical Analysis) เป็นวิธีการวิเคราะห์ข้อมูลอีกแบบหนึ่งที่มีแนวคิดตรงข้ามกับวิธีการแรก คือจะหารูปแบบของการทำงานที่เป็นปกติ แล้วสร้างเป็นโพรไฟล์(Profile) เก็บไว้ก่อน ในการวิเคราะห์ข้อมูล จะเปรียบเทียบข้อมูลกับโพรไฟล์ที่สร้างไว้ ถ้าไม่เข้ากันก็แสดงว่ามีความผิดปกติเกิดขึ้นแล้ว สำหรับโพรไฟล์นั้นอาจแยกเป็นโพรไฟล์สำหรับออปเจ็กต์ต่างๆ ในระบบเช่นยูสเซอร์,ไฟล์,ไดเร็กทอรีและอุปกรณ์ต่างๆ โดยรายละเอียดที่เก็บอยู่ในโพรไฟล์จะมีข้อมูลของจำนวนครั้งที่เข้าสู่ระบบ, จำนวนครั้งที่เข้าสู่ระบบผิดพลาด , เวลา และข้อมูลอื่นๆ ที่จำเป็น ค่าแต่ละค่าที่เก็บจะเป็นค่าของการใช้งานที่เป็นปกติ การตรวจจับว่าเกิดความผิดปกติขึ้นแล้ว จะดูจากค่าที่ไม่เข้ากับโพรไฟล์ เป็นต้น ยกตัวอย่างเช่น ในการทำงานปกติ ผู้ใช้งานฐานข้อมูลจะมีการเข้าใช้ข้อมูลในฐานข้อมูลตั้งแต่เวลา 8 โมงเช้า ถึง 6 โมงเย็นเท่านั้น แต่ข้อมูลที่ตรวจจับได้มีการเข้าใช้ฐานข้อมูลตอนตีสอง ซึ่งก็สามารถบอกได้ว่ามีการบุกรุกเกิดขึ้นแล้ว

เนื่องจากวิธีการวิเคราะห์ข้อมูลแบบ Statistical Analysis เป็นการตรวจจับโดยใช้หลักการของการอนุญาตให้ใช้งานในการทำงานทั่วๆ ไปและไม่อนุญาตให้ใช้งานนอกเหนือจากที่เคยใช้โดยทั่วไปเท่านั้น การตรวจจับในลักษณะนี้จึงสามารถตรวจจับการบุกรุกที่ไม่เคยเจอมาก่อนได้ และสามารถตรวจจับการบุกรุกในรูปแบบที่ซับซ้อนได้ด้วย เพราะเราถือว่าการทำงานที่สลับซับซ้อนมักจะไม่เหมือนกับการทำงานโดยปกติ แต่วิธีการวิเคราะห์ข้อมูลแบบนี้ก็มีข้อเสียเนื่องจากเป็นการเก็บข้อมูลการทำงานที่เป็นปกติไว้เพื่อเปรียบเทียบกับการทำงานที่ผิดปกติ เมื่อทำการบุกรุกในลักษณะเดิมๆ เป็นเวลานาน ก็จะทำให้โพรไฟล์มีการเปลี่ยนแปลง ระบบตรวจจับก็จะเห็นว่าการโจมตีในลักษณะนั้นเป็นการทำงานที่เป็นปกติแทน และไม่สามารถตรวจจับการทำงานที่ผิดปกติในลักษณะนั้นได้อีกต่อไป และไม่เหมาะกับองค์กรที่มีการเปลี่ยนแปลงการทำงานบ่อยๆ เพราะทำให้โพรไฟล์มีขนาดใหญ่ ทำให้ระบบตรวจจับรวน และมีความผิดพลาดในการวิเคราะห์สูง

3) วิธีการตรวจสอบการเปลี่ยนแปลงของระบบ (Integrity Analysis)

วิธีสุดท้ายที่นิยมใช้กันในการวิเคราะห์ข้อมูลคือ วิธีการตรวจสอบการเปลี่ยนแปลงของระบบ (Integrity Analysis) ลักษณะการทำงานของวิธีการนี้คือการหาว่ามีการเปลี่ยนแปลงเกิดขึ้นในระบบหรือไม่ เช่นมีไฟล์ไหนมีการเปลี่ยนแปลง หรือมี ออปเจ็กต์อะไรที่มีการเปลี่ยนแปลงคุณสมบัติบ้าง แล้วทำการแจ้งเตือนกับผู้ดูแลระบบ ในการวิเคราะห์ลักษณะนี้จะใช้แฮชอัลกอริทึม (hash algorithm) เพื่อสร้างแมสเซชไดเจส(message digest) ของข้อมูล แล้วทำการเปรียบเทียบแมสเซชไดเจส ของข้อมูลในช่วงเวลาต่างๆ ว่าเหมือน หรือต่างกันหรือไม่ ถ้าแมสเซชไดเจสต่างกันก็แสดงว่าข้อมูลมีการเปลี่ยนแปลง Integrity Analysis สามารถตรวจจับการบุกรุก ที่เข้ามาเปลี่ยนแปลงข้อมูลในระบบ หรือมีการติดตั้งโปรแกรมเช่น sniffer , rootkit ต่างๆ ในระบบได้ แต่ก็มีข้อเสียคือการวิเคราะห์ระบบในลักษณะนี้จะทำงานเป็นแบบ batch เท่านั้น ไม่เหมาะกับการทำ real time อย่างยิ่งเพราะจะทำให้เปลืองทรัพยากรมาก

3.3.3 การตอบสนอง

เมื่อมีการตรวจพบว่ามีการบุกรุกเกิดขึ้นในระบบ สำหรับระบบตรวจจับที่ทำงานแบบ real time จะมีการตอบสนองต่อการบุกรุกเพื่อไม่ให้เกิดความเสียหาย หรือบรรเทาความเสียหายที่เกิดขึ้น สำหรับระบบที่ทำงานเป็นแบบ batch การตอบสนองอาจทำได้ไม่มากนัก เพราะการบุกรุกนั้นเกิดไปแล้ว ความเสียหายก็เกิดขึ้นแล้ว การตอบสนองอาจอยู่ในรูปการบรรเทาไม่ให้ความเสียหายมีเพิ่มมากขึ้นเท่านั้น การตอบสนองต่อการบุกรุกนั้นแบ่งออกได้เป็นสามแบบด้วยกันคือการเปลี่ยนแปลงสภาพของระบบ การแก้ไขความผิดพลาดให้ถูก และการแจ้งเตือนผู้ดูแลระบบเมื่อถูกบุกรุก

3.3.3.1 การเปลี่ยนแปลงสภาพของระบบ

สำหรับการตอบสนองต่อการบุกรุกโดยการเปลี่ยนแปลงสภาพของระบบที่ถูกโจมตีก็เพื่อแก้ปัญหาหรือลดความเสียหายที่จะเกิดขึ้นเช่นตัดการเชื่อมต่อระหว่างระบบกับผู้บุกรุกออกจากกัน การตั้งค่าอุปกรณ์เครือข่ายหรือไฟร์วอลล์ไม่ให้มีการติดต่อกับระบบของผู้บุกรุกอีกต่อไป และการหาข้อมูลเกี่ยวกับการโจมตีโดยอัตโนมัติเพื่อตรวจหาผู้บุกรุกต่อไป

3.3.3.2 การแก้ไขความผิดพลาดให้ถูก

การแก้ไขระบบ เป็นการตอบสนองต่อปัญหาที่เกิดขึ้นแล้วในระบบ โดยปกติแล้วผู้บุกรุกมักเปลี่ยนแปลงค่าต่างๆ ในระบบ โดยเฉพาะเข้ามาทำการเปลี่ยนแปลงข้อมูลในระบบตรวจจับผู้บุกรุกเพื่อไม่ให้สามารถตรวจจับการบุกรุกได้ การแก้ไขระบบก็เพื่อให้ระบบดังกล่าวสามารถทำงานได้อย่างเป็นปกติ

3.3.3.3 การแจ้งเตือนผู้ดูแลระบบ

สุดท้ายเป็นการแจ้งเตือนผู้ดูแลระบบ โดยปกติมักแจ้งเตือนผู้ดูแลทันทีเมื่อทำการวิเคราะห์ได้ว่ามีความผิดปกติเกิดขึ้น เพื่อให้ผู้ดูแลระบบรับรู้และสามารถแก้ไขระบบได้ทันท่วงที สำหรับการแจ้งเตือนนี้ ผู้ดูแลระบบสามารถเลือกได้ว่าจะแจ้งเตือนใครบ้าง และทำการแจ้งเตือนในรูปแบบไหนอาจเป็น E-mail , Pager หรือ Instant Messaging ต่างๆ

3.3.4 การรายงานผลการทำงาน

เมื่อระบบตรวจจับผู้บุกรุกทำการวิเคราะห์ระบบ และตรวจพบความผิดปกติในระบบ อาจมีการตอบสนองต่อความผิดปกตินั้นถ้าทำได้ จากนั้นระบบตรวจจับผู้บุกรุกต้องมีการรายงานผลให้กับผู้ดูแลระบบทราบในรูปแบบต่างๆ โดยรายละเอียดของการรายงานผลนั้น จะบอกถึงช่องโหว่ในระบบ การแก้ไขปัญหาคร่าวๆ บางครั้งอาจมีรายละเอียดของความรู้พื้นฐานบางอย่างของระบบ ที่ทำให้เกิดการบุกรุกลักษณะนั้นๆ ได้ การรายงานผลการทำงาน นอกจากเป็นการรายงานต่อผู้ดูแล

ระบบเพื่อให้ทราบการทำงาน หรือจุดอ่อนในระบบแล้ว ยังเป็นประโยชน์ต่อการวิเคราะห์สถานะของระบบ และการวิเคราะห์ความปลอดภัยในระบบอีกด้วย

3.4. ความสำคัญของ IDS

เมื่อได้ทราบถึงการทำงานคร่าวๆ ของระบบตรวจจับผู้บุกรุกแล้ว อาจคิดว่าระบบตรวจจับผู้บุกรุกนั้นไม่มีความสำคัญเพราะในเมื่อมีการใช้งานไฟร์วอลอยู่แล้ว แต่ความเป็นจริงแล้วถึงแม้ว่าระบบจะมีไฟร์วอลอยู่แล้วก็ยังจำเป็นต้องใช้ระบบตรวจจับผู้บุกรุกด้วยเพราะในงานบางอย่างไฟร์วอลก็ไม่สามารถช่วยได้

จุดประสงค์ของการใช้งานไฟร์วอลล์นั้น สร้างขึ้นเพื่อเป็นเสมือนตัวป้องกันระบบให้แยกตัวออกมาจากเครือข่ายที่ไม่ปลอดภัย เป็นเมืองหน้าด่านของระบบ เป็นผู้ป้องกันการบุกรุกจากภายนอก แต่ระบบตรวจจับผู้บุกรุกนั้นมีจุดประสงค์ที่แตกต่างไป โดยเป็นผู้เฝ้าดูระบบ และเป็นผู้เตือนเมื่อเกิดความผิดปกติเกิดขึ้น ยกตัวอย่างในอาคารใหญ่ๆ จะมี คนคอยดูแลอยู่ภายนอกกันคนที่ไม่ควรเข้ามาในอาคารให้อยู่ภายนอก แต่ภายในตัวอาคารก็จะมีกล้องวีดีโอคอยตรวจตราอยู่ภายในมีกริ่งสัญญาณเตือนเมื่อเกิดความผิดปกติเกิดขึ้น ซึ่งก็ช่วยให้แก้ปัญหาได้ทันท่วงที และในกรณีที่มีความผิดปกติเกิดขึ้น แต่ไม่สามารถตรวจจับได้ในขณะนั้น ระบบตรวจจับผู้บุกรุกก็มีการจัดเก็บข้อมูลการใช้ระบบไว้ จึงสามารถนำข้อมูลดังกล่าวมาวิเคราะห์หาความผิดปกติได้ภายหลัง โดยจุดประสงค์ในการสร้างระบบตรวจจับผู้บุกรุกและไฟร์วอลล์ จึงต่างกันโดยสิ้นเชิง แต่ถึงแม้ว่าจุดประสงค์การทำงานของตรวจจับผู้บุกรุกและ ไฟร์วอลล์ จะแตกต่างกัน แต่ทั้งสองก็สามารถทำงานร่วมกันและทำให้ประสิทธิภาพการรักษาความปลอดภัยในระบบดีขึ้นด้วย

3.5 สรุป

ระบบตรวจจับผู้บุกรุกเป็นผู้ช่วยที่ดีสำหรับผู้ดูแลระบบ หน้าที่ของตรวจจับผู้บุกรุกนั้นจะรวบรวมข้อมูล วิเคราะห์ข้อมูลและทำการแจ้งผลการวิเคราะห์ข้อมูลต่างๆ ในระบบให้กับผู้ดูแลระบบ ซึ่งช่วยให้การดูแลระบบทำได้อย่างมีประสิทธิภาพ แต่ระบบตรวจจับผู้บุกรุกก็ยังไม่ใช่สิ่งที่จะมาแก้ปัญหาความปลอดภัยในระบบโดยสิ้นเชิงได้ เนื่องจากการที่จะทำให้ระบบปลอดภัยต้องอาศัยความร่วมมือจากหลายๆ ฝ่าย ไม่เพียงแต่เฉพาะการดูแลของผู้ดูแลระบบคนเดียว สิ่งที่สำคัญที่สุดที่ผู้เขียนเห็นว่าจะสร้างความปลอดภัยให้กับระบบในระยะยาวก็คือ การปลุกจิตสำนึกด้านความปลอดภัยในการใช้งานให้กับผู้ใช้ระบบแต่ละคนด้วย

บทที่ 4

SNORT

SNORT (Roesch, M. : 1999.) คือโปรแกรมตรวจจับผู้บุกรุกทางเครือข่ายที่มีการใช้อย่างแพร่หลาย เนื่องจากเป็นระบบตรวจจับผู้บุกรุกที่ใช้งานง่าย แต่มีประสิทธิภาพสูง สามารถตรวจจับการบุกรุกทางเครือข่ายได้หลากหลาย โดยผู้ใช้งานเพียงแค่สร้างกฎขึ้นมาให้มีรูปแบบของกฎตามที่โปรแกรม SNORT ได้ให้ไว้ ตัวแปรในกฎมีหลากหลายทำให้สามารถสร้างกฎที่ซับซ้อนได้ อีกทั้งยังมีการแบ่งปันกฎที่ใช้ในการตรวจสอบการบุกรุกแบบต่างๆ ในอินเตอร์เน็ต ผู้ใช้งานสามารถดาวน์โหลดมาใช้งานได้ทันที นอกจากนี้ในส่วนของการพัฒนาตัวโปรแกรมนั้นก็มีการพัฒนาอย่างรวดเร็วเพราะเป็นโปรแกรมที่เปิดเผยซอร์สโค้ดทำให้มีผู้ร่วมพัฒนามากมาย

4.1 โครงสร้างของ SNORT

การออกแบบโปรแกรม SNORT นั้นออกแบบโดยคำนึงถึงประสิทธิภาพในการทำงานและการใช้งานง่ายเป็นหลัก โดยมีส่วนการทำงานหลักๆ อยู่สามส่วนด้วยกันคือ ส่วนของการแปลความหมายข้อมูลแพ็กเก็ต(packet decoder) , ส่วนของการตรวจสอบกฎ (detection engine) และการบันทึกผลการทำงานและแจ้งเตือน(logging and alerting subsystem) โดยมีการทำงานร่วมกับไลบรารี่ Libcap ซึ่งเป็นไลบรารี่ในการอ่านข้อมูลของแพ็กเก็ตที่ผ่านไปมาในเครือข่าย โดยการทำงานแต่ละส่วนจะมีรายละเอียดดังนี้

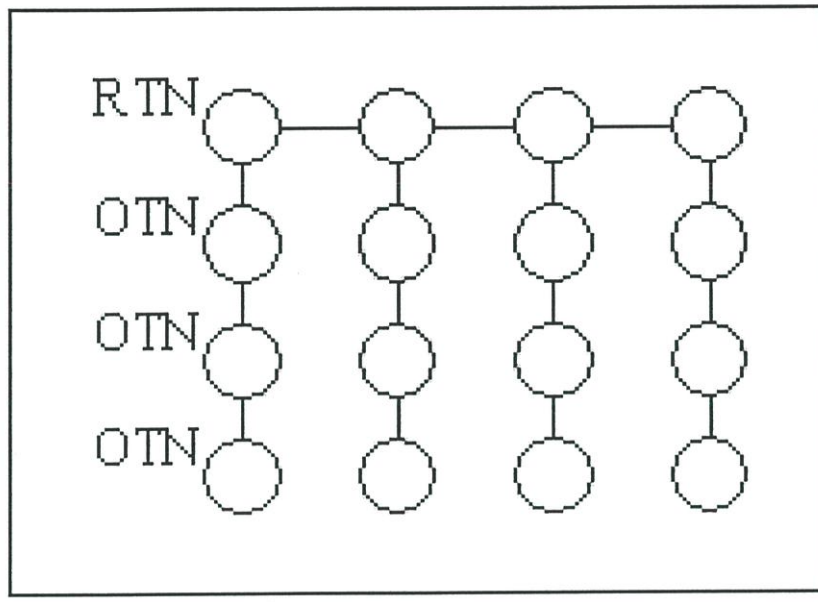
4.1.1 ส่วนของการแปลความหมายข้อมูลแพ็กเก็ต (Packet Decoder)

ในส่วนของการแปลความหมายแพ็กเก็ตนั้นจะทำงานอยู่บริเวณโพรโตคอลสแตก ในชั้นดาตาลิงค์เลเยอร์ และ ทีซีพี/ไอพี ซึ่งจะถูกแบ่งเป็นโปรแกรมย่อยต่างๆ แต่ละโปรแกรมย่อยของการแปลความหมายแพ็กเก็ตจะทำงาน โดยการนำเอาข้อมูลของแพ็กเก็ต มาเทียบค่ากับโครงสร้างข้อมูลของแพ็กเก็ตแบบต่างๆ ส่วนของโปรแกรมที่แปลความหมายแพ็กเก็ตจะถูกเรียกให้ทำงานเมื่อโปรแกรม SNORT ได้รับแพ็กเก็ตจากเครือข่าย โดยจะแปลความหมายของโพรโตคอลตั้งแต่ชั้นดาตาลิงค์ , ชั้นเน็ตเวิร์ค ,ชั้นทรานสพอร์ต ขึ้นไปเรื่อยๆ จนถึงชั้นแอพพลิเคชั่น

การทำงานในส่วนนี้ต้องทำงานด้วยความเร็วสูงมาก ซึ่งโปรแกรม SNORT ใช้วิธีการสร้างพอยเตอร์หลายๆ ตัว เพื่อชี้ในส่วนต้นของเฮดเดอร์ ในโพรโตคอลชั้นต่างๆ ทำให้การแปลความหมายข้อมูลทำได้อย่างรวดเร็ว สำหรับการตีความแพ็กเก็ตนั้นโปรแกรม SNORT สามารถตีความแพ็กเก็ตในโพรโตคอลได้หลายๆ โพรโตคอล เช่น Ethernet , SLIP , raw (PPP) datalink protocol และโพรโตคอลอื่นๆ ที่มีการใช้งานกันอย่างแพร่หลาย

4.1.2 ส่วนของการตรวจสอบกฎ (detection engine)

สำหรับโครงสร้างที่เป็นลิงค์ลิสท์ 2 มิตินั้นประกอบด้วยโหนด 2 รูปแบบด้วยกันคือ Rule Tree Node (RTN) และ Option Tree Node (OTN) ดังรูปที่ 4.1 โดย RTN จะเป็นโหนดที่บรรจุส่วนของกฎที่รูปแบบการบุกรุกหลายๆ แบบจะต้องมี เช่น ไอพีแอดเดรสต้นทาง , ไอพีแอดเดรสปลายทาง , พอร์ตต้นทาง , พอร์ตปลายทางและโพรโตคอลที่ใช้ในแพ็กเก็ตนั้นๆ เช่น ทีซีพี ยูดีพี หรือ ไอซีเอ็มพี เป็นต้น ส่วน OTN ประกอบด้วยส่วนเฉพาะของกฎ ที่เป็นส่วนที่แตกต่างกันในการโจมตีแต่ละแบบเช่น โค้ดของไอซีเอ็มพีแพ็กเก็ต ทีซีพีแฟล็ก และ ข้อมูลในแพ็กเก็ตที่ใช้บุกรุก เป็นต้น



รูปที่ 4.1 โครงสร้างข้อมูลแบบลิงค์ลิสต์สองมิติ

สำหรับจุดประสงค์ในการสร้างโครงสร้างกฎในลักษณะนี้คือหลีกเลี่ยงการตรวจสอบกฎทุกๆ กฎ โดยพยายามแบ่งกลุ่มของกฎออกเป็นกลุ่มๆ แทน ถ้าตรวจสอบข้อมูลในแพ็กเก็ตกับ RTN แล้วไม่เหมือนก็ไม่จำเป็นต้องตรวจสอบกฎกับ OTN ที่ติดกับ RTN นั้นอีก สำหรับ SNORT ในกรณีที่ตรวจสอบกับเงื่อนไขใน RTN แล้วไม่เหมือน ก็จะไปตรวจสอบกับข้อมูลใน RTN อื่นๆ ต่อไปจนหมด

ในกรณีที่ตรวจสอบกับข้อมูลใน RTN แต่ละส่วนแล้วตรงกับเงื่อนไข จะต้องตรวจสอบกับเงื่อนไขใน OTN อีกครั้งหนึ่ง ในส่วนของการเปรียบเทียบ content ของกฎนั้น โปรแกรม SNORT ใช้อัลกอริทึมในการตรวจสอบกฎโดยใช้ อัลกอริทึมของบอยเออร์-มัวร์ ถ้าตรวจสอบใน OTN แล้วไม่พบความผิดปกติ ก็จะตรวจสอบแพ็กเก็ตดังกล่าวกับ OTN ถัดไปจนหมดรายการของ OTN กระบวนการดังกล่าวเป็นกระบวนการที่เสียเวลามาก ถึงแม้ว่าบอยเออร์-มัวร์ จะช่วยให้การค้นหา

และตรวจสอบทำได้เร็วขึ้นก็ตามเพราะการทำงานในการตรวจสอบกฎก็ต้องทำทีละกฎไปเรื่อยๆ จนครบทุกกฎ และทุกครั้งของการตรวจสอบจะต้องอ่านแพ็กเก็ตข้อมูลจนหมดด้วย

4.1.3 การบันทึกผลการทำงานและแจ้งเตือน(logging and alerting subsystem)

ส่วนการบันทึกผลการทำงานและแจ้งเตือนต่อผู้ดูแลระบบ ผู้ใช้งานโปรแกรม SNORT สามารถเลือกได้จากการป้อนพารามิเตอร์ ขณะรันโปรแกรม SNORT ในการทำงานในส่วนนี้สามารถเลือกการบันทึกผลการทำงานได้ 3 รูปแบบและสามารถเลือกการแจ้งเตือนได้ 5 รูปแบบ สำหรับการทำงานอื่นๆ ที่เป็นตัวเลือกในการเก็บข้อมูลแพ็กเก็ต สามารถเลือกได้ว่าจะเก็บข้อมูลไว้ในรูปแบบอย่างไร โดยสามารถเลือกรูปแบบได้เป็น แพ็กเก็ตที่ถูกแปลความหมายแล้ว , แพ็กเก็ตที่อยู่ในรูปแบบที่สามารถอ่านค่าต่างๆ ได้ (human readable) หรืออยู่ในรูปแบบของโปรแกรม tcpdump (tcpdump binary format) เก็บไว้ในไฟล์ข้อมูลเพียงไฟล์เดียว

โดยการบันทึกแพ็กเก็ตที่ถูกแปลความหมายแล้วจะสามารถนำเอาข้อมูลไปวิเคราะห์เร็วกว่าการเก็บข้อมูลที่ยังไม่ได้แปลความหมาย ส่วนการเก็บข้อมูลในรูปแบบของ tcpdump จะสามารถเก็บข้อมูลได้เร็วกว่า ซึ่งเหมาะกับการใช้งานในกรณีที่ต้องการประสิทธิภาพการทำงานที่สูงกว่า ในบางครั้งการบันทึกการทำงานนั้นอาจไม่จำเป็นต้องทำ โดยผู้ดูแลระบบจะใช้เพียงการแจ้งเตือนเท่านั้นเพื่อให้โปรแกรมทำงานน้อยลง และได้ความเร็วในการทำงานเพิ่มมากขึ้น

สำหรับการแจ้งเตือนผู้ดูแลระบบ โปรแกรม SNORT สามารถส่งผลลัพธ์ไปยังระบบการเก็บข้อมูลการทำงานของระบบปฏิบัติการหรือ syslog ได้ โดยการทำงานนี้สามารถบันทึกผลลัพธ์ได้ในลักษณะของเท็กซ์ไฟล์หรือสามารถส่งการแจ้งเตือนในลักษณะ Winpopup โดยใช้โปรแกรม Samba client ส่งการแจ้งเตือนไปยังหน้าต่างของ Microsoft Windows การส่งข้อมูลเพื่อแจ้งเตือนผู้ดูแลระบบโดยส่งไปเก็บไว้ในเท็กซ์ไฟล์นั้นสามารถทำได้ 2 รูปแบบคือแบบเต็มรูปแบบ(Full Alert) และแบบแจ้งเตือนแบบรวดเร็ว (Fast Alert)

ในการแจ้งเตือนแบบเต็มรูปแบบจะเขียนข้อมูลที่แจ้งเตือน และข้อมูลตั้งแต่ เฮดเดอร์ของไอพี แพ็กเก็ตจนถึงข้อมูลในชั้นทรานส์พอร์ต ไว้ในเท็กซ์ไฟล์ ส่วนการแจ้งเตือนแบบรวดเร็วจะเขียนข้อมูลเพียงบางส่วนของเฮดเดอร์เก็บไว้ในไฟล์ เพื่อให้การทำงานรวดเร็วขึ้น ส่วนการทำงานรูปแบบสุดท้ายที่สามารถเลือกได้ในการแจ้งเตือนนั้นคือการ disable alerting ใช้ในกรณีที่ไม่ต้องการให้มีการแจ้งเตือนใดๆ เกิดขึ้น เช่นในกรณีที่ทำการทดสอบระบบเครือข่ายเป็นต้น

4.2 การทำงานของ SNORT

การทำงานของโปรแกรม SNORT สามารถแบ่งเป็น 3 ขั้นตอนหลักๆ คือ ส่วนการทำงานก่อนการตรวจจับ , ส่วนการทำงานขณะทำการตรวจจับ และการทำงานหลังการตรวจจับ

4.2.1 การทำงานก่อนการตรวจจับ

การทำงานก่อนการตรวจจับ จะเป็นกระบวนการเตรียมสิ่งแวดล้อมให้พร้อมต่อการทำงาน โดยสิ่งที่ต้องเตรียมคือโครงสร้างข้อมูลของกฎที่จำเป็นต้องใช้ในการตรวจสอบ กระบวนการนี้เริ่มต้นเมื่อมีการรันโปรแกรม SNORT โปรแกรมจะอ่านค่าพารามิเตอร์ต่างๆ เช่น อ่านไฟล์ที่เป็นกฎจากไฟล์ใด อ่านแพ็กเก็ตจากไฟล์หรือจากเครือข่าย ให้แสดงผลแบบใด ทำงานเป็นเดมอนหรือไม่ ฯลฯ โดยเฉพาะไฟล์ของกฎที่จะให้ตรวจสอบนั้น จะต้องถูกอ่านในกระบวนการนี้เมื่อ SNORT อ่านไฟล์ของกฎแล้ว จะสร้างโครงสร้างข้อมูลของกฎเพื่อใช้ในการวิเคราะห์ข้อมูล โดยโครงสร้างข้อมูลของกฎมีลักษณะเป็นลิงค์ลิสต์ 2 มิติ แล้วจึงเปลี่ยนโหมดของเน็ตเวิร์คการ์ดให้เป็น promiscuous mode เพื่อใช้ในการอ่านข้อมูลในแพ็กเก็ตที่ผ่านไปมาในเครือข่าย

4.2.2 ส่วนการทำงานขณะทำการตรวจจับ

การทำงานในส่วนของการตรวจจับนั้นเริ่มจาก โปรแกรม SNORT รับแพ็กเก็ตข้อมูลจากเครือข่าย ซึ่งใช้ฟังก์ชันในไลบรารี่ Libpcap เพื่อดึงข้อมูลจากเครือข่าย หลังจากนั้นโปรแกรมจะตีความส่วนต่างๆ ของข้อมูลที่ได้รับเข้ามาว่าเป็นข้อมูลของโพรโตคอลใดประกอบกันบ้าง แล้วจึงนำมาเปรียบเทียบกับกฎซึ่งได้สร้างเป็นโครงสร้างข้อมูลแบบลิงค์ลิสต์ 2 มิติไว้แล้วในส่วนการทำงานแรก เมื่อเปรียบเทียบกฎต่างๆ เรียบร้อยแล้ว ถ้าแพ็กเก็ตไหนที่ตรงตามกฎ จะตอบสนองตามที่กฎได้ตั้งค่าไว้ เช่นแจ้งเตือนผู้ดูแลระบบหรือบันทึกการทำงานในการตรวจจับ ซึ่งการทำงานของโปรแกรมจะทำงานไปเรื่อยๆ จนกว่าจะมีการส่งสัญญาณต่างๆ ไปหยุดโพรเซส เช่น SIGHUB หรือ SIGKILL เป็นต้น

4.2.3 ส่วนการทำงานหลังการตรวจจับเสร็จสิ้น

ในส่วนการทำงานนี้จะเป็นการสรุปผลการรับข้อมูลว่าข้อมูลที่ได้รับนั้นเป็นแพ็กเก็ตข้อมูลโพรโตคอลอะไรบ้าง เป็นจำนวนเท่าไหร่ และคิดเป็นกี่เปอร์เซ็นต์ของแพ็กเก็ตที่รับเข้ามาทั้งหมด ดังตัวอย่างในรูปที่ 4.2

```
Breakdown by protocol:
TCP: 25          (18.248%)
UDP: 28          (20.438%)
ICMP: 6          (4.380%)
FRAGS: 0         (0.000%)
ARP: 61          (44.526%)
IPv6: 0          (0.000%)
IPX: 0           (0.000%)
OTHER: 17        (12.409%)
```

รูปที่ 4.2 ตัวอย่างการทำงานของโปรแกรม SNORT เมื่อสิ้นสุดการตรวจจับ

จากรูปตัวอย่างหมายถึง ตลอดระยะเวลาทำงานของโปรแกรม SNORT ได้รับแพ็กเก็ต TCP 25 แพ็กเก็ต คิดเป็น 18.248 เปอร์เซ็นต์ของทั้งหมด แพ็กเก็ต UDP 28 แพ็กเก็ต คิดเป็น 20.438 เปอร์เซ็นต์ของทั้งหมด แพ็กเก็ต ICMP 6 แพ็กเก็ต คิดเป็น 4.380 เปอร์เซ็นต์ของทั้งหมด แพ็กเก็ต ARP 61 แพ็กเก็ต คิดเป็น 44.526 เปอร์เซ็นต์ของทั้งหมด แพ็กเก็ตที่ใช้โพรโตคอลอื่นๆ อีก 17 แพ็กเก็ตคิดเป็น 12.409 เปอร์เซ็นต์ของแพ็กเก็ตทั้งหมด ในการทำงานครั้งนี้ไม่มีแพ็กเก็ตที่เป็น IPV6 , IPX และแพ็กเก็ตที่เป็นแฟรกเมนต์เตชั่น (fragmentation)

4.4 กฎและโครงสร้างของกฎที่ใช้ใน SNORT

กฎในโปรแกรม SNORT นั้นคือประโยคที่เป็นเงื่อนไขในการทำงานของโปรแกรม SNORT โดยจะเป็นประโยคที่บอกว่า แพ็กเก็ตลักษณะไหนที่เป็นแพ็กเก็ตที่ผิดปกติ แล้วโปรแกรม SNORT จะต้องทำงานอย่างไรกับกฎนั้น ในช่วงการทำงานของโปรแกรม SNORT กฎจะถูกอ่านในช่วงของการทำงานก่อนการตรวจจับ เพื่อสร้างโครงสร้างของกฎในรูปลิงค์ลิส 2 มิติ แล้วนำไปใช้ในช่วงการทำงานขณะทำการตรวจจับ

สำหรับกฎที่ SNORT ใช้นั้นมีไวยกรณ์ของกฎประกอบด้วย ส่วนที่อธิบายการตอบสนองของ SNORT ต่อแพ็กเก็ตผิดปกติ , ส่วนเงื่อนไขของเฮดเดอร์ของข้อมูลและพอร์ต และส่วน Option field ยกตัวอย่างในรูปที่ 4.3

```
Alert tcp any any -> any 80 (msg:"IIS-cmd?";flags:PA;content:".cmd?&"; nocase;)
```

รูปที่ 4.3 ตัวอย่างกฎของ SNORT

จากรูปกฎตัวอย่างหมายถึงโปรแกรมจะแจ้งเตือนต่อผู้ดูแลระบบถ้าแพ็กเก็ตที่รับเข้ามาเป็นแพ็กเก็ตที่ทำงานกับโพรโตคอลทีซีพี โดยส่งจากไอพีและพอร์ตใดๆ ไปยังไอพีปลายทางใดๆ ที่พอร์ต 80 โดยในทีซีพีเฮดเดอร์จะเซตแฟลก PSH และ ACK และในแพ็กเก็ตมีคำว่า ".cmd?&" อยู่ในแพ็กเก็ต ในการตรวจสอบข้อมูลในแพ็กเก็ตจะไม่คำนึงถึงนัยสำคัญของตัวอักษรตัวพิมพ์เล็กและตัวพิมพ์ใหญ่

โดยรายละเอียดของกฎ ในส่วนของการตอบสนองของโปรแกรม SNORT ในกรณีที่ข้อมูลในแพ็กเก็ตตรงตามกฎ จะมีการตอบสนอง 3 รูปแบบด้วยกันคือ แจ้งเตือนผู้ดูแลระบบ (alert) , เก็บข้อมูลแพ็กเก็ต (log) และไม่สนใจแพ็กเก็ตนั้น (pass) ในส่วนถัดมาเป็นส่วนเงื่อนไขของเฮดเดอร์ข้อมูล สามารถตั้งค่าได้สามส่วนคือไอพีแอดเดรส , พอร์ต และ direction operator

1. ไอพีแอดเดรส

ในส่วนนี้จะเป็นรายละเอียดเกี่ยวกับหมายเลขไอพีแอดเดรสที่อยู่ในแพ็กเก็ต โดยมีค่าพิเศษคือ any ซึ่งจะหมายถึงไอพีแอดเดรสใดๆ ในส่วนนี้โปรแกรม SNORT จะไม่สามารถป้อนค่าเป็นชื่อโฮสได้เนื่องจากไม่มีกระบวนการในการเปลี่ยนชื่อโฮสให้กลายเป็นไอพีแอดเดรส สำหรับค่าหมายเลขไอพีแอดเดรสที่ใช้ได้นั้น สามารถเป็นหมายเลขไอพีแอดเดรสโดดหรือหมายเลขไอพีแอดเดรสตามด้วย CIDR block ก็ได้ โดย CIDR block จะเป็นการบอก netmask ของแพ็กเก็ต โดย CIDR block /24 จะหมายถึงเครือข่ายคลาสซี , /16 หมายถึงเครือข่ายคลาสบี และ /32 จะหมายถึงหมายเลขแอดเดรสของเครื่องนั้นๆ ยกตัวอย่างเช่น 192.168.1.0/24 จะหมายถึงช่วงของแอดเดรสตั้งแต่ 192.168.1.1 ถึง 192.168.1.255

นอกจากนี้ยังมีโอเปอเรเตอร์อีกตัวหนึ่งที่ใช้งานควบคู่กับหมายเลขไอพีแอดเดรสได้คือ Negation "!" โดยโอเปอเรเตอร์ดังกล่าวจะหมายถึงให้โปรแกรม SNORT ตรวจสอบหมายเลขไอพีแอดเดรสทุกๆ หมายเลขยกเว้นหมายเลขไอพีแอดเดรสนั้น ยกตัวอย่างเช่น การตั้งกฎเพื่อให้โปรแกรม SNORT แจ้งเตือนเมื่อมีข้อมูลจากภายนอกส่งเข้ามาในเครือข่าย สามารถทำได้โดยตั้งกฎดังตัวอย่างรูปที่ 4.4

```
alert tcp !192.168.1.0/24 any -> 192.168.1.0/24 any …
```

รูปที่ 4.4 ตัวอย่างกฎที่ใช้ Negation กับไอพีแอดเดรส

จากตัวอย่างมีความหมายว่าให้โปรแกรม SNORT แจ้งเตือนเมื่อมีแพ็กเก็ตที่เป็นโพรโตคอล TCP โดยมีหมายเลขไอพีต้นทางไม่ได้อยู่ในเครือข่าย แต่หมายเลขไอพีปลายทางอยู่ในเครือข่าย

2. พอร์ต

จะเป็นรายละเอียดเกี่ยวกับหมายเลขพอร์ตที่ต้องการดักจับ ซึ่งในกรณีที่เป็นพอร์ตเดี่ยวๆ สามารถใช้เป็นตัวเลขได้ หรือสามารถดักจับในช่วงโดยใช้เครื่องหมาย ":" คั่นระหว่างช่วงพอร์ต เช่น 1:1024 และในกรณีที่ต้องการดักทุกๆ พอร์ตจะให้คำว่า "any" ยกตัวอย่างในรูปที่ 4.5 เป็นตัวอย่างการตั้งกฎโดยในกฎแรกจะให้โปแกรม SNORT บันทึก UDP แพ็กเก็ตที่มาจากทุกๆ ไอพี และทุกๆ พอร์ต และมีแอดเดรสปลายทางอยู่ในเครือข่าย 192.168.1.0/24 ในช่วงพอร์ต 1 ถึง 1024 ในกฎที่สองจะให้โปรแกรม SNORT บันทึก TCP แพ็กเก็ตที่ส่งมาจากไอพีแอดเดรสใดๆ และพอร์ตใดๆ ที่ส่งมายังในเครือข่าย 192.168.1.0/24 ที่พอร์ตน้อยกว่าหรือเท่ากับ 6000 ส่วนกฎที่สามเป็นกฎที่ให้โปรแกรม SNORT บันทึก TCP แพ็กเก็ตที่ส่งมาจากไอพีแอดเดรสใดๆ ที่พอร์ตน้อยกว่าหรือเท่ากับ 1024 ส่งมายังเครื่องในเครือข่าย 192.168.1.0/24 ที่พอร์ตมากกว่าหรือเท่ากับ 500

```
log udp any any -> 192.168.1.0/24 1:1024
log tcp any any -> 192.168.1.0/24 :6000
log tcp any :1024 -> 192.168.1.0/24 500:
```

รูปที่ 4.5 ตัวอย่างการใช้กฎที่มีการใช้พอร์ตรูปแบบต่างๆ

นอกจากนี้ยังสามารถใช้ Negation "!" เพื่อให้มีความหมายตรงข้ามได้ เช่นในรูปที่ 4.6 ตัวอย่างการใช้กฎที่มีการใช้ Negation กับพอร์ต โดยกฎนี้จะให้โปรแกรม SNORT บันทึกแพ็กเก็ตทุกๆ แพ็กเก็ตที่ส่งเข้ามาในเครือข่าย 192.168.1.0/24 ที่ไม่ใช่พอร์ตของ X Windows (พอร์ต 6000 ถึง 6010) แต่การใช้ Negation นี้จะไม่สามารถใช้กับ "any" ได้

```
log tcp any any -> 192.168.1.0/24 !6000:6010
```

รูปที่ 4.6 ตัวอย่างการใช้กฎที่มีการใช้ Negation กับพอร์ต

3. direction operator

Direction Operator หรือเครื่องหมาย "->" เป็นเครื่องหมายในการบอกทิศทางของข้อมูลที่กฎจะนำไปใช้ โดยหมายเลขไอพีแอดเดรสและหมายเลขพอร์ตที่อยู่ด้านซ้ายมือของเครื่องหมายเป็นการบอกที่มาของแพ็กเก็ตว่าต้นทางคือหมายเลขไอพีแอดเดรสอะไร และส่งมาจากพอร์ตไหน ส่วนหมายเลขไอพีแอดเดรสและหมายเลขพอร์ตที่อยู่ด้านขวามือของเครื่องหมายจะหมายถึงหมายเลขไอพีแอดเดรสและพอร์ตของเครื่องปลายทาง

สำหรับ Direction Operator นี้จะมีอีกรูปแบบหนึ่งคือ bi-directional operator ซึ่งเป็นเครื่องหมาย "<>" ซึ่งหมายถึงให้ โปรแกรม SNORT พิจารณาคู่ของหมายเลขไอพีแอดเดรสและพอร์ต ไม่ว่าจะเป็นต้นทางหรือปลายทาง ซึ่งเหมาะกับการวิเคราะห์ข้อมูลทั้งไปและกลับเช่น เซสชั่นของ telnet และ POP3 รูปที่ 4.7 เป็นตัวอย่างของกฎที่ใช้ bi-direectional operator โดยกฎนี้จะให้โปรแกรม SNORT บันทึกแพ็กเก็ตที่ส่งไปมาระหว่างภายนอกเครือข่าย 192.168.1.0/24 ที่พอร์ตใดๆ กับ ในเครือข่าย 192.168.1.0/24 ที่พอร์ต 23 ซึ่งก็คือการเก็บข้อมูลการทำงานต่างๆ ในเซสชั่น telnet นั่นเอง

```
log !192.168.1.0/24 any <> 192.168.1.0/24 23
```

รูปที่ 4.7 ตัวอย่างของกฎที่ใช้ bi-directional operator

ในส่วนสุดท้ายเป็นส่วนของ Option field ซึ่งเป็นส่วนของการตั้งกฎให้โปรแกรม SNORT ตรวจสอบค่าปลีกย่อยต่างๆ ในแพ็กเก็ต ซึ่งใน SNORT version 1.2.1 จะมี 14 option field คือ

1. content: เป็นการตั้งค่ารูปแบบที่ต้องการให้โปรแกรมค้นหาใน payload ของแพ็กเก็ต
2. flags: เป็นการตั้งค่าให้โปรแกรม SNORT ตรวจสอบค่าแฟลกของแพ็กเก็ต
3. ttl: ตรวจสอบค่า TTL ในไอพีเฮดเดอร์ของแพ็กเก็ต
4. itype: ตรวจสอบค่า ICMP type
5. icode: ตรวจสอบค่า ICMP code
6. minfrag: ตั้งค่าขนาดของ fragment size ของไอพีแพ็กเก็ต
7. id: ทดสอบค่าไอพีเฮดเดอร์ตามค่าที่กำหนดไว้
8. ack: ตรวจสอบค่าหมายเลข acknowledgement number ของทีซีพีเฮดเดอร์
9. seq: ตรวจสอบค่าหมายเลข sequence number ของทีซีพีเฮดเดอร์
10. logto: เป็นชื่อของไฟล์ที่ต้องการเก็บแพ็กเก็ตที่ตรงกับกฎ
11. dsize: ตรวจสอบขนาดของ payload ในแพ็กเก็ต
12. offset: เป็น option เสริมของค่า content option โดยจะตั้งค่าเพื่อเป็น offset ในแพ็กเก็ตในการบอกจุดเริ่มในการค้นหาข้อมูล
13. depth: เป็น option เสริมของค่า content option โดยจะตั้งค่าเพื่อเป็นขอบเขตในการค้นหาในแพ็กเก็ต
14. msg: เป็นการตั้งค่าประโยคที่จะไปปรากฏในการแจ้งเตือนหรือการบันทึก เมื่อพบกฎนั้นในแพ็กเก็ต

4.5 ตัวอย่างกฎของโปรแกรม SNORT

จากตัวอย่างกฎที่ใช้งานจริงในโปรแกรม SNORT version 1.6.3 นั้น เราจะเห็นได้ว่ามีการใช้งานหลายๆ รูปแบบเพื่อดักจับการบุกรุกหลายๆ แบบได้ เมื่อพิจารณาดู content ของแต่ละกฎเราพบว่ามีรูปแบบอยู่คือจะเป็นคำคั่นด้วยตัวแบ่งระหว่างคำนั้นเป็นส่วนใหญ่ ดังตัวอย่าง

กฎที่ 1 : alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth- IIS search97 access attempt"; content:"/search97.vts"; nocase; flags: PA;)

กฎที่ 2 : alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth- BigConf access attempt"; content:"/bigconf.cgi"; nocase; flags: PA;)

กฎที่ 3 : alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth- Shopping cart access attempt"; content:"/quikstore.cfg"; nocase; flags: PA;)

กฎที่ 4 : alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth Mode 8- Order log access attempt"; content:"/admin_files\\order.log"; nocase; flags: PA;)

กฎที่ 5 :. alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth- WS_FTP.INI access attempt "; content:"/ws_ftp.ini"; nocase; flags: PA;)

กฎที่ 6 : alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth- Order log access attempt"; content:"/admin_files/order.log"; nocase; flags: PA;)

กฎที่ 7 : alert tcp !any any -> any 80 (msg:"SCAN - Whisker Stealth Mode 8- wrap CGI access attempt"; content:"/cgi-bin\\wrap"; nocase; flags: PA;)

กฎที่ 8 : log tcp !any any -> any 80 (msg:"ColdFusion-display";flags:PA; content:"cfdocs/expeval/displayopenedfile.cfm"; nocase;)

กฎที่ 9 : log tcp !any any -> any 80 (msg:"ColdFusion-evaluate";flags:PA; content:"cfdocs/snippets/evaluate.cfm"; nocase;)

กฎที่ 10 : log tcp !any any -> any 80 (msg:"ColdFusion-Example-beaninfo";flags:PA; content:"cfdocs/examples/cvbeans/beaninfo.cfm"; nocase;)

กฎที่ 11 : log tcp !any any -> any 80 (msg:"ColdFusion-Example-cfappman";flags:PA; content:"/cfappman/index.cfm"; nocase;)

กฎที่ 12 : log tcp !any any -> any 80 (msg:"ColdFusion-Example-parks";flags:PA; content:"cfdocs/examples/parks/detail.cfm"; nocase;)

กฎที่ 13 : log tcp !any any -> any 80 (msg:"CVE-1999-0455 - ColdFusion-exprcalc";flags:PA; content:"cfdocs/expeval/exprcalc.cfm"; nocase;)

จากตัวอย่างทั้ง 13 กฎนี้เมื่อพิจารณาดู content ของทุกๆ กฎพบว่ามีตัวอักษร "/" และ "." กระจายอยู่ในกฎแทบทุกกฎ ซึ่งเราอาจนำเอานัยสำคัญของคุณลักษณะนี้มาช่วยในการค้นหาคำหรือลด search space ได้

4.6 โครงสร้างข้อมูลของกฎเป็นอย่างไร

SNORT เป็นโปรแกรมที่ใช้สำหรับตรวจจับผู้บุกรุกทางเครือข่าย การทำงานของโปรแกรมนี้คือการดักจับข้อมูลในเครือข่าย และเปรียบเทียบข้อมูลนั้นกับรูปแบบของการบุกรุก โดยโครงสร้างข้อมูลสำหรับตรวจสอบรูปแบบการบุกรุกจะอยู่ในรูปของลิงค์ลิสท์ 2 มิติ (Two Dimensional Linked List) ดังรูปที่ 4.1 ซึ่งจะถูกสร้างในการทำงานก่อนการตรวจสอบกฎ ซึ่งถ้าในกระบวนการตรวจสอบกฎ ถ้ามีข้อมูลภายในแพ็กเก็ตที่มีข้อมูลตรงตามเงื่อนไขกับข้อมูลในกฎ โปรแกรม SNORT จะแจ้งเตือนให้ผู้ดูแลระบบทันที กระบวนการทำงานเช่นนี้ทำให้ผู้ดูแลระบบทราบได้ว่ามีการบุกรุกและสามารถป้องกันความเสียหายได้ทันท่วงที

โครงสร้างกฎที่เป็นลิงค์ลิสท์ 2 มิติประกอบด้วยโหนด 2 แบบด้วยกันคือ Rule Tree Node (RTN) และ Option Tree Node (OTN) โดย RTN เป็นโหนดที่บรรจุส่วนของกฎที่รูปแบบการบุกรุกหลายๆ แบบจะต้องมี เช่น ไอพีแอดเดรสต้นทาง , ไอพีแอดเดรสปลายทาง , พอร์ตต้นทาง , พอร์ตปลายทางและโพรโตคอลที่ใช้ในแพ็กเก็ตนั้นๆ เช่น ทีซีพี ยูดีพี หรือ ไอซีเอ็มพี เป็นต้น โดยโครงสร้างกฎของ RTN จะเป็นดังรูปที่ 4.8

```
typedef struct _RuleTreeNode
{
  RuleFpList *rule_func;
  int head_node_number;
    int type;
    u_long sip;         /* src IP */
    u_long smask;       /* src netmask */
    u_long dip;         /* dest IP */
    u_long dmask;       /* dest netmask */
    int not_sp_flag;    /* not implemented yet... */
    u_short hsp;        /* hi src port */
    u_short lsp;        /* lo src port */
    int not_dp_flag;    /* not implemented yet... */
    u_short hdp;        /* hi dest port */
    u_short ldp;        /* lo dest port */
    u_char flags;       /* control flags */
    struct _RuleTreeNode *right;
    OptTreeNode *down;  /* list of rule options to associate with this rule node */
} RuleTreeNode;
```

รูปที่ 4.8 โครงสร้างของ Rule Tree Node

ส่วนโครงสร้างใน OTN ประกอบด้วยส่วนเฉพาะของกฎที่เป็นส่วนที่แตกต่างกันในการโจมตีแต่ละแบบเช่น โค้ดของไอซีเอ็มพีแพ็กเก็ต ทีซีพีแฟล็ก และ ข้อมูลในแพ็กเก็ตที่ใช้บุกรุก เป็นต้น โดยจะมีโครงสร้างข้อมูลดังรูปที่ 4.9

```
typedef struct _OptTreeNode
{
   /* plugin/detection functions go here */
   OptFpList *opt_func;
   /* the ds_list is absolutely essential for the plugin system to work,
     it allows the plugin authors to associate "dynamic" data structures
     with the rule system, letting them link anything they can come up
     with to the rules list */
   void *ds_list[512];  /* list of plugin data struct pointers */
   int chain_node_number;
   int type;          /* alert, log, or pass */
   int proto;         /* protocol, added for integrity checks
                      during rule parsing */
                      int session_flag;   /* record session data */
   char *logto;       /* log file in which to write packets which
                      match this rule*/
   char *message;     /* alert message */
   int response_flag;  /* flexible response on alert */
   struct _OptTreeNode *next;
} OptTreeNode;
```

รูปที่ 4.9 โครงสร้างของ Opt Tree Node

สำหรับจุดประสงค์ในการสร้างโครงสร้างกฎในลักษณะนี้คือหลีกเลี่ยงการตรวจสอบกฎทุกๆ กฎ โดยพยายามแบ่งกลุ่มของกฎออกเป็นกลุ่มๆ แทน ถ้าตรวจสอบข้อมูลในแพ็กเก็ตกับ RTN แล้วไม่เหมือนก็ไม่จำเป็นต้องตรวจสอบกฎกับ OTN ที่ติดกับ RTN นั้นอีก สำหรับ SNORT ในกรณีที่ตรวจสอบกับเงื่อนไขใน RTN แล้วไม่เหมือน ก็จะไปตรวจสอบกับข้อมูลใน RTN อื่นๆ ต่อไปจนหมด

4.7 การวิเคราะห์แพ็กเก็ต

เมื่อได้รับแพ็กเก็ตจากเครือข่ายโดยข้อมูลจะถูกดักจับมาโดยไลบรารี Libcap แล้วโปรแกรมจะตรวจสอบเฮดเดอร์ของข้อมูลในแพ็กเก็ตกับข้อมูลใน RTN ก่อน โดยจะตรวจสอบค่าของไอพีแอดเดรสต้นทาง ไอพีแอดเดรสปลายทาง พอร์ตต้นทาง และพอร์ตปลายทางของแพ็กเก็ตแต่ละแพ็กเก็ต กับข้อมูลที่อยู่ใน RTN ไปเรื่อยๆ จนครบทุกโหนด ในกรณีที่เจอแพ็กเก็ตที่มีข้อมูลตรงกับข้อมูลใน RTN พอดี จะทำการเปรียบเทียบข้อมูลในส่วนของ OTN ที่เชื่อมต่อกับ RTN โหนดนั้นต่อไป ในขั้นตอนการเปรียบเทียบ OTN นี้จะมีการทำงาน option และ plugin ต่างๆ ซึ่งถ้าข้อมูลในแพ็กเก็ตเกิดตรงกับข้อมูลใน OTN โหนดใดโหนดหนึ่ง จะทำการแจ้งเตือนให้กับผู้ดูแลระบบในรูปแบบที่ได้ตั้งค่าไว้ในกฎ

ในการเปรียบเทียบค่าต่างๆ ใน OTN และ RTN นั้นจะมีการเปรียบเทียบค่าตัวแปรที่มีขอบเขตชัดเจน ยกเว้นการตรวจสอบ content field ใน OTN เท่านั้นที่มีการเปรียบเทียบที่แตกต่างจากการเปรียบเทียบอื่นๆ เนื่องจากโปรแกรม SNORT จะใช้วิธีการตรวจสอบข้อมูลในแพ็กเก็ตที่ผ่านไปมาในเครือข่ายว่ามีข้อมูลตรงตามที่กฎได้ตั้งไว้หรือไม่ การค้นหาจึงต้องใช้วิธีการค้นหาคำในข้อมูลยาวๆ ซึ่ง โปรแกรม SNORT จะใช้อัลกอริทึมบอยเออร์-มัวร์ในการค้นหาคำ ถ้าตรวจสอบใน OTN แล้วไม่พบความผิดปกติ จะตรวจสอบแพ็กเก็ตดังกล่าวกับ OTN ถัดไปจนหมดรายการของ OTN กระบวนการดังกล่าวเป็นกระบวนการที่เสียเวลามาก ถึงแม้ว่าบอยเออร์-มัวร์จะช่วยให้การค้นหาและตรวจสอบทำได้เร็วขึ้นก็ตามเพราะ การทำงานในการตรวจสอบกฎก็ต้องทำทีละกฎไปเรื่อยๆ จนครบทุกกฎ และทุกครั้งของการตรวจสอบจะต้องอ่านแพ็กเก็ตข้อมูลจนหมดด้วย ซึ่งเมื่อดูตามการทำงานแล้วจะเห็นว่ากระบวนการตรงนี้เป็นกระบวนการที่ซ้ำซ้อนและเสียเวลามาก

บทที่ 5
การปรับปรุง SNORT

การสร้างระบบ NIDS ต้องคำนึงถึงการการใช้งานเป็นหลัก เนื่องจาก NIDS จะต้องสามารถทำงานแบบ near-realtime ได้คือต้องเป็นระบบที่สามารถตรวจจับการบุกรุกได้เกือบทันทีที่มีการบุกรุกเกิดขึ้น ถ้าสร้าง NIDS แล้วไม่สามารถตรวจจับสัญญาณของการบุกรุกได้ทัน จะเป็นระบบที่ไม่สามารถทำงานได้จริง สำหรับปัญหาที่เกิดขึ้นในการใช้งาน NIDS คือในบางครั้งระบบ NIDS ถูกรบกวนโดยใช้เทคนิค Insertion , Evasion และ Denial of Service (Ptacek, T. and Newsham, T. : 1998.) ทำให้ระบบ NIDS ไม่สามารถตรวจจับการบุกรุกระบบที่แท้จริงได้

เพื่อให้การรบกวนระบบ NIDS มีผลต่อการทำงานน้อยที่สุด และทำให้การทำงานของ NIDS เป็น near-realtime มากที่สุด เราจึงพยายามปรับปรุงระบบ NIDS ให้ทำงานได้เร็วขึ้น โดยเทคนิคที่ใช้ในการปรับปรุงก็คือ การเปลี่ยนแปลงโครงสร้างข้อมูลของกฎ และเพิ่มกระบวนการเปลี่ยนรูปแบบกฎและข้อมูลในแพ็กเก็ตเสียก่อนเพื่อลดขนาดของข้อมูลที่จะเปรียบเทียบ โดยเปลี่ยนรูปของกฎและข้อมูลในแพ็กเก็ตให้เหลือเพียง ตัวแบ่งและระยะห่างระหว่างตัวแบ่งภายในกฎเท่านั้น แล้วจึงวิเคราะห์หารูปแบบของการบุกรุก การทำเช่นนี้ทำให้ข้อมูลที่จะต้องเปรียบเทียบลดลง การทำงานโดยรวมของระบบจึงเร็วขึ้น สำหรับในงานวิจัยนี้เราได้ทดลองเพื่อพิสูจน์สมมุติฐานโดยเปลี่ยนแปลงโครงสร้างข้อมูล และวิธีการเปรียบเทียบกฎของโปรแกรม SNORT ซึ่งเป็นโปรแกรมตรวจจับผู้บุกรุกทางเครือข่ายที่ใช้งานกันอย่างแพร่หลาย แล้วทดลองเปรียบเทียบผลกับข้อมูลเครือข่ายหลายๆ รูปแบบ

ในการทดลองเพื่อพิสูจน์สมมุติฐาน ได้เลือกใช้โปรแกรม SNORT ซึ่งเป็นโปรแกรมตรวจจับผู้บุกรุกที่เปิดเผยโค้ดของโปรแกรม ทำให้ง่ายในการสร้างและทดสอบโมเดลต่างๆ ลำดับการทำงานในการปรับปรุงโปรแกรม SNORT นั้นมีขั้นตอนดังนี้คือ

1. เริ่มจากการศึกษารายละเอียดการทำงานและโครงสร้างของโปรแกรม SNORT
2. วิเคราะห์การทำงานเพื่อหาจุดที่สามารถเพิ่มความเร็วให้โปรแกรม SNORT ได้
3. ศึกษาและทดลองใช้โครงสร้างที่เหมาะสมกับโปรแกรม SNORT
4. ปรับปรุงโครงสร้างของโปรแกรม SNORT
5. ทดสอบเพื่อพิสูจน์สมมุติฐาน

ในการวิจัยครั้งนี้ ในส่วนของการศึกษารายละเอียดการทำงานและโครงสร้างของโปรแกรม SNORT ได้อธิบายไปแล้วในบทที่ 4 และส่วนของการทดสอบเพื่อพิสูจน์สมมุติฐาน จะมีรายละเอียดในบทที่ 6

5.1 การวิเคราะห์การทำงานเพื่อหาจุดที่สามารถเพิ่มความเร็วให้โปรแกรม SNORT

จากการทำงานของโปรแกรม SNORT ซึ่งมีรายละเอียดในบทที่ 4 จะเห็นการทำงานสองข้อที่ยังไม่สัมพันธ์กันและมีการทำงานบางอย่างที่สามารถเพิ่มประสิทธิภาพการทำงานของโปรแกรม SNORT ได้คือ

5.1.1 โครงสร้างกฎยังไม่มีความสัมพันธ์กับการทำงานของ SNORT

เนื่องจากโครงสร้างกฎของโปรแกรม SNORT จะมีลักษณะเป็นลิงค์ลิสต์ 2 มิติ ซึ่งเป็นการออกแบบเพื่อลดความซ้ำซ้อนของการตรวจสอบกฎ จากรูปจะเห็นได้ว่าถ้าโปรแกรม SNORT รับแพ็กเก็ตข้อมูลมาแล้ว จะนำส่วนเฮดเดอร์ของแพ็กเก็ตมาตรวจสอบกับข้อมูลในแต่ละโหนดใน RTN ไปเรื่อยๆ จนครบทุกๆ RTN

ในกรณีที่มีข้อมูลใน RTN โหนดไหนที่เหมือนกับข้อมูลในเฮดเดอร์ของแพ็กเก็ตแล้ว โปรแกรม SNORT จะตรวจสอบข้อมูลภายในแพ็กเก็ตกับข้อมูลใน OTN ที่ต่อเข้ากับ RTN นั้นๆ ต่ออีก ในการตรวจสอบข้อมูลใน OTN จะใช้อัลกอริทึมชื่อว่าบอยเออร์-มัวร์เพื่อช่วยในการทำ string matching ให้เร็วขึ้น ซึ่งการตรวจสอบนี้จะทำไปเรื่อยๆ จนกว่าจะหมดลิสต์ของ OTN ซึ่งถ้ามีแพ็กเก็ตไหนที่มีข้อมูลของ OTN นั้นๆ อยู่ โปรแกรม SNORT ก็จะแจ้งเตือนผู้ดูแลระบบตามกฎที่ตั้งไว้

ในการทำงานทั้งหมด โครงสร้างลิงค์ลิสต์แบบสองมิติมีความเหมาะสมแล้วเนื่องจากการแยกข้อมูลที่เป็นข้อมูลทั่วๆ ไปรวมไว้เป็นลิสต์ของโหนดชื่อว่า RTN และแยกข้อมูลที่มีความแตกต่างกันไว้เป็นลิสต์ของโหนดชื่อว่า OTN ทำให้ตรวจสอบเฮดเดอร์ ทำได้เร็วขึ้นเพราะไม่จำเป็นต้องทำงานซ้ำซ้อน แต่ในกรณีของการตรวจสอบข้อมูลใน OTN การตรวจสอบคำทีละคำ ในข้อมูลชุดเดียวกัน ยังไม่ใช่การทำงานที่ดีนัก เนื่องจากถ้ามีกฎที่ต้องตรวจสอบมากขึ้น จำนวนครั้งที่ต้องอ่านข้อมูลเพื่อใช้ในการเปรียบเทียบคำก็ต้องใช้มากขึ้นตามจำนวนกฎ ถ้ากฎมากขึ้นการทำงานของโปรแกรมจึงช้าลงเรื่อยๆ

5.1.2 กฎและข้อมูลที่ใช้มีรูปแบบสามารถนำมาใช้ลด search space ได้

จากกฎดังรูปที่ 5.1 เป็นส่วนของ content ในกฎซึ่งเป็นข้อมูลที่ต้องนำมาเป็นคำสำคัญในการค้นหาในข้อมูลของแพ็กเก็ตต่างๆ มีรูปแบบที่คล้ายๆ กันอยู่ก็คือ มีลักษณะเป็น คำต่างๆ คั่นด้วยตัวแบ่ง ซึ่งเราสามารถเอาคุณสมบัติข้อนี้มาใช้เพื่อลด search space ทำให้จำนวนการเปรียบเทียบตัวอักษรลดลง ความเร็วในการทำงานเพิ่มขึ้นได้

```
"/bin/shA-cA/usr/openwin"
"scripts/tools/newdsn.exe"
"iisadmpwd/aexp4.htr"
"_vti_pvt/service.cnf"
"_private/orders.txt"
"cfdocs/expeval/sendmail.cfm"
```

รูปที่ 5.1 ตัวอย่าง content ในกฎ

ในการค้นหาคำสำคัญในข้อมูลของแพ็กเก็ตนั้น ถ้าใช้การเปรียบเทียบคำสำคัญกับข้อมูลในแพ็กเก็ตทั้งหมดจะมีจำนวนการเปรียบเทียบตัวอักษรปริมาณมาก โดยต้องเปรียบเทียบตัวอักษรทีละตัวอักษร กับข้อมูลทั้งหมดในแพ็กเก็ตด้วย แต่จากรูปแบบของกฎที่มีลักษณะเป็นความยาวของคำคั่นด้วยตัวแบ่งระหว่างคำ ทำให้เราสามารถลดขนาดของข้อมูลและคำสำคัญได้

5.2 วิธีการปรับปรุงการทำงานของโปรแกรม SNORT

จากการวิเคราะห์ลักษณะการทำงานของโปรแกรม SNORT ซึ่งสามารถปรับปรุงการทำงานให้มีความเร็วในการทำงานมากขึ้นทำให้ได้คุณลักษณะ 2 ข้อที่สามารถปรับปรุงให้มีการทำงานเร็วขึ้นนั่นคือการเปลี่ยนแปลงโครงสร้างข้อมูล และการลด search space ด้วยตัวแบ่งระหว่างคำในข้อมูล

5.2.1 เปลี่ยนแปลงโครงสร้างข้อมูล ให้เป็น burst trie

ในการเปลี่ยนแปลงโครงสร้างของข้อมูลนั้นเมื่อดูจากการทำงานของ SNORT ซึ่งต้องมีการค้นหาคำสำคัญหลายๆ คำในข้อมูลชุดเดียวกัน โครงสร้างที่เหมาะสมที่ใช้ในปัญหานี้ควรเป็นโครงสร้างที่มีลักษณะเป็น tree หรือ tries แต่เพื่อความเร็วในการใช้งานควรใช้โครงสร้างข้อมูลแบบ tries มากกว่า โดย burst tries เป็นการปรับปรุงจากโครงสร้างข้อมูลแบบ tries ซึ่งมีลักษณะโครงสร้างประกอบด้วย 3 ส่วนหลักๆ คือ record , container และ access tries

ข้อดีของโครงสร้างกฎแบบ tries คือมีความเร็วในการตรวจสอบกฎมาก เนื่องจากการเปรียบเทียบข้อมูลใน tries จะไม่จำเป็นต้องเปรียบเทียบข้อมูลกับข้อมูลเหมือน tree แต่จะใช้วิธีการเปรียบเทียบโดยใช้ index ของข้อมูลนั้นแทน เมื่อเทียบความเร็วในการทำงานของโครงสร้างกฎแบบต่างๆ ที่มี binary tree, hash , tries , burst tries พบว่าโครงสร้างกฎแบบ hash จะมีความเร็วมากที่สุด รองลงมาคือ burst tries แต่ที่ไม่เลือกใช้ hash กับโครงสร้างกฎนี้เนื่องจากไม่มีความเหมาะสมในการประยุกต์ใช้กับงาน

ข้อเสียของโครงสร้างกฎแบบ tries คือใช้เนื้อที่ในการเก็บข้อมูลเยอะ เนื่องจากการออกแบบพยายามให้โครงสร้างมีการทำงานที่รวดเร็วที่สุด สามารถตรวจสอบข้อมูลได้โดยใช้ index ของข้อมูลได้เลย จึงต้องมีการจองเนื้อที่ไว้เป็นจำนวนมาก และในเนื้อที่ที่จองไว้บางส่วนจะไม่ได้ใช้งาน ทำให้สิ้นเปลืองมาก

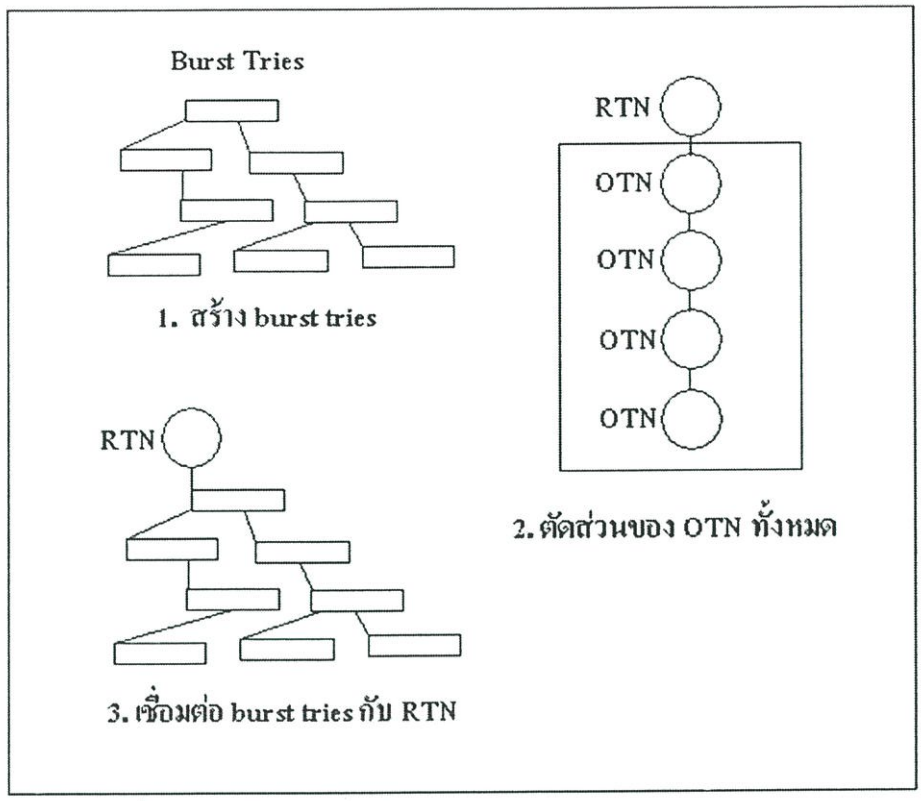
5.2.2 ความเหมาะสมของโครงสร้าง burst tries กับโครงสร้างกฎของ SNORT

เหตุผลที่ต้องใช้โครงสร้างกฎแบบ burst tries แทนโครงสร้างกฎของ SNORT มีเหตุผล 3 ข้อคือ

1. การเปลี่ยนแปลงโครงสร้างกฎทำได้ง่าย และไม่จำเป็นต้องเปลี่ยนแปลงโครงสร้างกฎเดิมของโปรแกรม SNORT มากนัก
2. ต้องการความเร็วในการทำงานจึงจำเป็นต้องใช้โครงสร้างกฎที่มีความเร็วในการทำงานสูง ถึงแม้ว่าในการทดสอบแล้วโครงสร้างกฎแบบ hash จะมีความเร็วมากกว่า แต่ก็เป็นโครงสร้างกฎที่ไม่เข้ากับโครงสร้างกฎของ SNORT และไม่สามารถลดความซ้ำซ้อนในการทำงานได้
3. โดยตัวโครงสร้างเมื่อนำมาใช้ในการทำงานในลักษณะการค้นหาคำสำคัญหลายๆ คำในชุดของข้อมูล trie จะเป็นโครงสร้างที่สามารถลดความซ้ำซ้อนในการทำงานได้มากที่สุด

5.2.3 ขั้นตอนการเปลี่ยนแปลงโครงสร้างกฎให้อยู่ในรูปของ burst tries

จากเดิมที่โครงสร้างกฎอยู่ในรูปของลิงค์ลิสต์สองมิติ โดยมี RTN และ OTN เชื่อมต่อกันอยู่ โดย RTN จะมีการเชื่อมต่อกันเป็นสาย และในแต่ละโหนดของ RTN จะเชื่อมต่อเข้ากับลิงค์ของ OTN ซึ่งโครงสร้างกฎแบบนี้มีความเหมาะสมในการตรวจสอบเฮดเดอร์ของข้อมูลในแพ็กเก็ต แต่เมื่อมองการทำงานก็ยังมีลักษณะมีการตรวจสอบแบบลำดับซึ่งถือว่าช้า ซึ่งยังไม่น่าเป็นปัญหามากนักเนื่องจากการเปรียบเทียบค่าใน RTN มีการเปรียบเทียบเพียงครั้งเดียว แล้วไปตรวจสอบข้อมูลใน RTN ใน 1 แพ็กเก็ตจึงมีการอ่านค่าข้อมูลเฮดเดอร์ของแพ็กเก็ตเพียง 1 ครั้งและมีการอ่านค่าข้อมูลใน RTN ในทุกๆ โหนด โหนดละ 1 ครั้ง ซึ่งยังถือว่ารวดเร็ว ต่างจากการตรวจสอบข้อมูลใน OTN ซึ่งทุกๆ ครั้งที่ทำงานกับ OTN 1 โหนด ต้องอ่านข้อมูลในแพ็กเก็ตซึ่งมีขนาดใหญ่ ทำให้เสียเวลามากกว่า และการทำงานกับ OTN หลายๆ โหนดย่อมเสียเวลามากกว่าการทำงานในส่วนของ RTN หลายเท่า



รูปที่ 5.2 ขั้นตอนการเปลี่ยนแปลงโครงสร้างให้อยู่ในรูปของ burst trie

ส่วนโครงสร้างกฎในส่วนของ OTN ยังทำให้การทำงานมีความซ้ำซ้อนในการเปรียบเทียบกฎอยู่ การทำงานในส่วนของ OTN ทำหน้าที่การตรวจสอบว่าข้อมูลในแพ็กเก็ตว่ามีข้อมูลตรงกับข้อมูลใน OTN หรือไม่ ถ้าไม่ตรงก็จะตรวจสอบกับโหนดถัดไป โดยการตรวจสอบนั้น ข้อมูลใน OTN จะเป็นคำ 1 คำ การตรวจสอบกฎคือการค้นหาคำๆ นี้ว่ามีอยู่ในแพ็กเก็ตหรือเปล่า ในการทำงานจึงต้องอ่านคำใน OTN 1 ครั้ง แล้วต้องอ่านข้อมูลในแพ็กเก็ตทั้งหมด 1 ครั้ง สำหรับ 1 โหนด การทำงานตรงส่วนนี้จึงมีความซ้ำซ้อนอย่างมาก ควรเปลี่ยนแปลงโครงสร้างส่วนนี้ไม่ให้มีความซ้ำซ้อน ถ้าสามารถเปรียบเทียบคำทั้งหมด โดยการอ่านข้อมูลในแพ็กเก็ตเพียง 1 ครั้งได้ ก็ยิ่งทำให้ประสิทธิภาพการทำงานเพิ่มขึ้นอย่างมาก

ในการเปลี่ยนแปลงโครงสร้างกฎนั้น เนื่องจากการวิเคราะห์ดูตามโครงสร้างแล้วเห็นว่าโครงสร้างกฎและการทำงานในส่วนของ RTN มีความเหมาะสมดีแล้ว แต่โครงสร้างกฎใน OTN นั้นยังมีความซ้ำซ้อนอยู่มาก จึงต้องเปลี่ยนแปลงโครงสร้างกฎตรงส่วน OTN จากการออกแบบจะเปลี่ยนแปลงโครงสร้างกฎในส่วนของ OTN ให้กลายเป็นโครงสร้างกฎแบบ burst tries โดยการเปลี่ยนแปลงทำได้โดยตัดส่วนโครงสร้างลิงค์ลิสต์ที่ต่ออยู่กับโหนด RTN ทุกๆ โหนด แล้วเชื่อมต่อด้วยโครงสร้างแบบ burst tries ที่สร้างขึ้นมาใหม่ ดังรูปที่ 5.2 โดยโครงสร้างของ burst tries จะมีลักษณะเป็นโครงสร้างต้นไม้ แต่ละโหนดของโครงสร้างต้นไม้จะเป็นอาเรย์ของพอยเตอร์ ไปยังโหนดอื่นๆ ต่อไป

5.2.4 การลด search space

จากรูปที่ 5.1 เราจะเห็นว่ากฎของโปรแกรม SNORT จะอยู่ในรูปของคำ คั่นด้วยตัวแบ่งระหว่างคำ ซึ่งเราสามารถนำลักษณะของกฎแบบนี้มาใช้เพื่อลดขนาดของ search space ได้โดยการเปลี่ยนรูปแบบของกฎให้อยู่ในรูปของความยาวคำ คั่นด้วยตัวแบ่งระหว่างคำ ดังรูปที่ 5.3

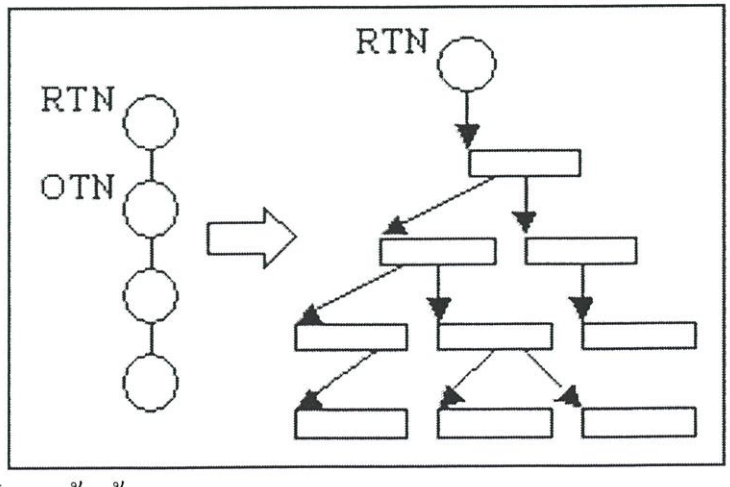
"_vti_pvt/service.cnf"	กลายเป็น	“8/7.3”
"_private/orders.txt"	กลายเป็น	“8/6.3”
"cfdocs/expeval/sendmail.cfm"	กลายเป็น	“6/7/8.3”
"/bin/shA-cA/usr/openwin"	กลายเป็น	“0/3/6/3/7”
"scripts/tools/newdsn.exe"	กลายเป็น	“7/5/6.3”
"iisadmpwd/aexp4.htr"	กลายเป็น	“9/5.3”

รูปที่ 5.3 ตัวอย่างการเปลี่ยนรูปแบบของข้อมูลโดยใช้กลุ่มของตัวแบ่งคือ “/.”

ในการลดขนาดของ search space ถ้าเปลี่ยนรูปแบบของคำสำคัญในการค้นหาให้อยู่ในรูปแบบของความยาวคำคั่นด้วยตัวแบ่งแล้ว ทำให้ขนาดของข้อมูล และคำสำคัญมีขนาดลดลงไป ซึ่งจะลดขนาดลงไปมากหรือน้อยนั้นจะขึ้นอยู่กับ ลักษณะของชุดของข้อมูล และชุดของคำสำคัญว่ามีตัวแบ่งที่ซ้ำซ้อนกันมากน้อยเพียงใดด้วย

เมื่อเรานำวิธีการปรับปรุงประสิทธิภาพของโปรแกรม SNORT ทั้งสองข้อมารวมกันทำให้เกิดโครงสร้างข้อมูลแบบ burst tries ที่ใช้การค้นหาข้อมูลแบบ delimiter search ได้ ขั้นตอนการนำกระบวนการทั้งสองมารวมกันคือ

1. สร้างโครงสร้างแบบ burst tries ขึ้น แล้วนำมาต่อกับ RTN แทน OTN ดังรูปที่ 5.4



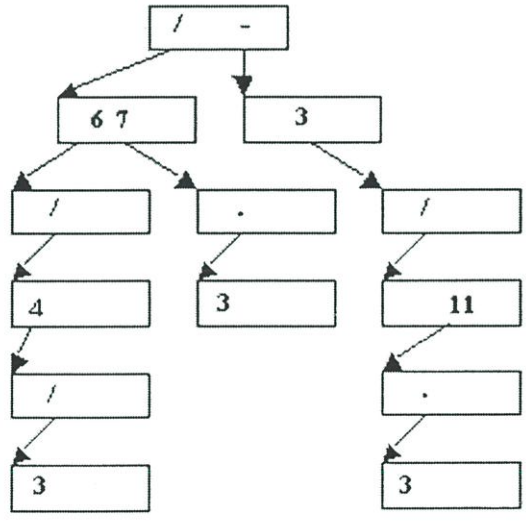
รูปที่ 5.4 การเปลี่ยนโครงสร้างข้อมูลของกฎ

2. นำข้อมูลภายในโหนดของ OTN มาเปลี่ยนรูปแบบให้อยู่ในรูปแบบของความยาวคำคั่นด้วยตัวแบ่งระหว่างคำดังรูปที่ 5.5

adsamples/config/site/csc	เปลี่ยนเป็น	9/6/4/3
admcgi/contents.htm	เปลี่ยนเป็น	6/7.3
cgi-win/wwwuploader.exe	เปลี่ยนเป็น	3-3/11.3

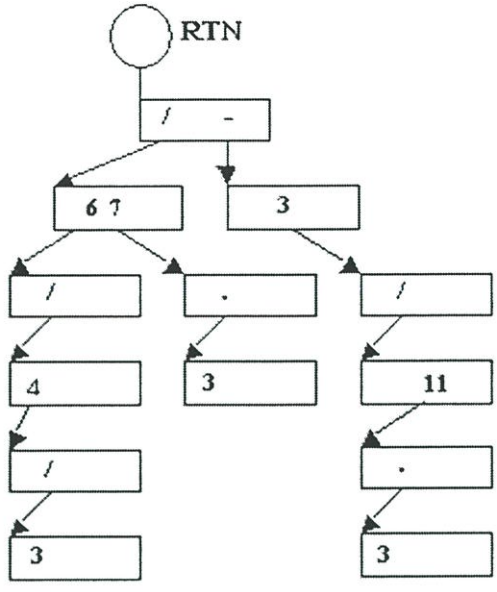
รูปที่ 5.5 การเปลี่ยนรูปแบบของคำในกฎโดยใช้กลุ่มตัวแบ่งคือ "/.-"

3. นำแต่ละตัวอักษรในคำที่เปลี่ยนรูปแล้วมาใส่ลงในโหนดของ burst tries โดยใส่ไว้ตัวอักษรละโหนด แล้วทำลิงค์ไปยังโหนดถัดๆ ไปดังตัวอย่างรูปที่ 5.6



รูปที่ 5.6 การนำตัวอักษรที่เปลี่ยนรูปแบบแล้วใส่ลงในโครงสร้าง burst tries

4. เมื่อสิ้นสุดกระบวนการทั้งหมดแล้วจะได้โครงสร้างกฎที่มีลักษณะเป็นโหนด RTN ต่อเข้ากับโครงสร้างกฎแบบ burst tries ที่มีการทำตัวชี้ตามข้อมูลของความยาวคำและตัวแบ่งดังรูปที่ 5.7



รูปที่ 5.7 โครงสร้างข้อมูลของกฎหลังจากเปลี่ยนรูปแบบแล้ว

บทที่ 6

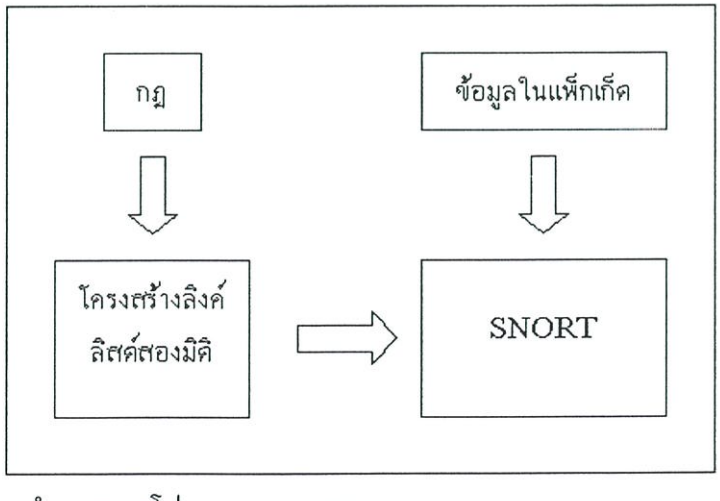
วิธีการทดลอง,ผลการทดลอง และการวิเคราะห์ข้อมูล

6.1 อุปกรณ์และโปรแกรมต่างๆ ที่ใช้ในการทดลอง

1. เครื่องคอมพิวเตอร์ Intel pentium pro 133 MHz หน่วยความจำขนาด 64 เมกกะไบต์ และเครื่องคอมพิวเตอร์ AMD K6-2 400 หน่วยความจำขนาด 196 เมกกะไบต์ สำหรับเขียนโปรแกรมเพื่อทดสอบสมมุติฐาน
2. เครื่องคอมพิวเตอร์ Celeron 500 MHz หน่วยความจำขนาด 256 เมกกะไบต์ สำหรับเก็บข้อมูลเครือข่ายและสร้างแพ็กเก็ตในการทดสอบต่างๆ
3. ฮับและอุปกรณ์เครือข่ายอื่นๆ
4. โปรแกรม SNORT 1.6.3 เป็นโปรแกรมต้นแบบ
5. โปรแกรม TCPDUMP สำหรับช่วยวิเคราะห์ข้อมูล
6. ไลบรารี่ LIBPCAP สำหรับใช้งานควบคู่กับ TCPDUMP และ SNORT 1.6.3
7. ไลบรารี่ LIBNET สำหรับเขียนโปรแกรมสร้างแพ็กเก็ตทดสอบ

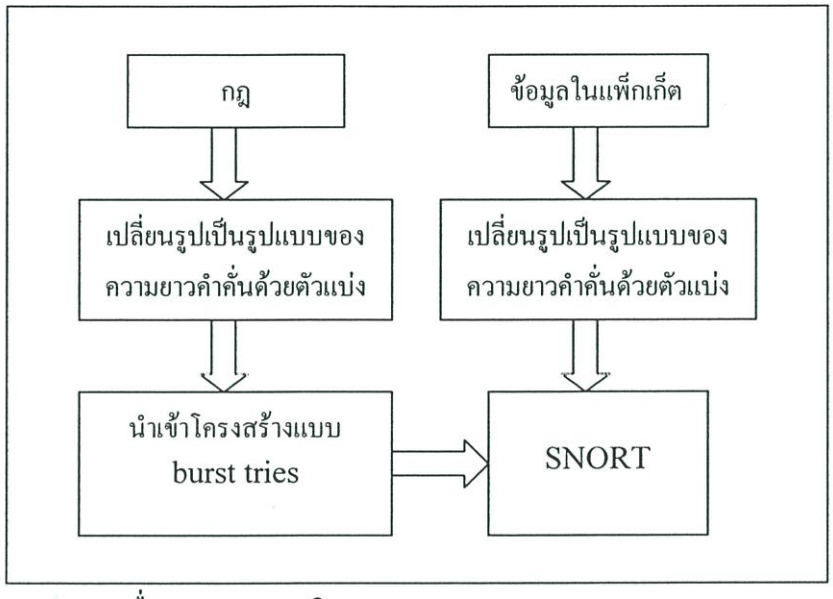
6.2 โมเดลการทำงานของโปรแกรมเพื่อทดสอบสมมุติฐาน

จากการทำงานโปรแกรม SNORT ที่มีรายละเอียดอยู่ในบทที่ 4 เราสามารถสร้างโมเดลการทำงานของโปรแกรม SNORT ได้ดังรูปที่ 6.1 โดยโมเดลการทำงานของโปรแกรมจะมีส่วนของอ่านค่ากฎในไฟล์ แล้วนำมาใส่ในโครงสร้างแบบลิงค์ลิสต์สองมิติ หลังจากนั้นจึงอ่านข้อมูลจากเครือข่ายแล้วนำมาเป็นอินพุทของระบบการตรวจสอบกฎ ถ้าพบแพ็กเก็ตที่มีลักษณะตรงตามที่กฎบอกไว้ จะแจ้งเตือนต่อผู้ดูแลระบบทันที



รูปที่ 6.1 โมเดลการทำงานของโปรแกรม SNORT

เพื่อทดสอบสมมุติฐาน จำเป็นต้องเพิ่มเติมส่วนของการทำงาน โดยเราจะต้องเพิ่มในสองส่วนหลักคือการทำงานเพื่อเปลี่ยนแปลงกฎให้อยู่ในรูปแบบของความยาวคำคั่นด้วยตัวแบ่งระหว่างคำ และส่วนของการเปลี่ยนแปลงข้อมูลในแพ็กเก็ตให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งระหว่างคำ



รูปที่ 6.2 โมเดลในการทำงานเพื่อทดสอบสมมุติฐาน

สำหรับรูปแบบโมเดลการทำงานที่เปลี่ยนแปลงไปนี้ เนื่องจากมีการเพิ่มกระบวนการในการเปลี่ยนแปลงรูปแบบแพ็กเก็ตเข้าไป จึงต้องเสียเวลาในการอ่านข้อมูลในแพ็กเก็ตก่อน 1 รอบเสมอ สำหรับการทำงานกับแพ็กเก็ต 1 แพ็กเก็ตซึ่งก็ถือว่าเป็นข้อจำกัดอย่างหนึ่งของโมเดลการทำงานแบบนี้

6.3 กฎและข้อมูลที่ใช้ในการทดลอง

สำหรับกฎทั้งหมดที่โปรแกรม SNORT ใช้ในเวอร์ชั่น 1.9.3 นั้นอยู่ในไฟล์ชื่อ snortrules-stable.tar.gz ซึ่งสามารถดาวน์โหลดได้จากเว็บไซต์ www.snort.org โดยมีกฎทั้งหมดดังตารางที่ 6.1

ตารางที่ 6.1 กฎทั้งหมดและจำนวนกฎที่มีอยู่ในกฎแต่ละแบบของโปรแกรม SNORT version 1.9.3

ชื่อกฎ	จำนวนกฎ
attack-responses.rules	28
backdoor.rules	130
bad-traffic.rules	21
chat.rules	29
ddos.rules	42
deleted.rules	77
dns.rules	28
dos.rules	25
experimental.rules	13
exploit.rules	46
finger.rules	21
ftp.rules	106
icmp-info.rules	108
icmp.rules	35
imap.rules	23
info.rules	14
local.rules	6
misc.rules	43
multimedia.rules	15
mysql.rules	15
netbios.rules	21
nntp.rules	9
oracle.rules	43
other-ids.rules	21
p2p.rules	23
policy.rules	26
pop2.rules	11
pop3.rules	20
porn.rules	36

ตารางที่ 6.1 (ต่อ)

rpc.rules	76
rservices.rules	18
scan.rules	38
shellcode.rules	33
smtp.rules	32
snmp.rules	24
sql.rules	46
telnet.rules	26
tftp.rules	22
virus.rules	97
web-attacks.rules	60
web-cgi.rules	307
web-client.rules	17
web-coldfusion.rules	43
web-frontpage.rules	41
web-iis.rules	122
web-misc.rules	351
web-php.rules	25
x11.rules	9
รวม	2422

จากกฎทั้งหมดจะเห็นได้ว่ามีกฎของเว็บมากที่สุดคือ 966 กฎจากทั้งหมด 2422 กฎ ซึ่งเป็นผลมาจากที่มีการใช้งานเว็บมาก และมีการโจมตีเข้าไปยังเว็บเซิร์ฟเวอร์มากที่สุด เราจึงเลือกกฎของเว็บมาเป็นกฎหลักในการทดสอบในงานวิจัยนี้ สำหรับข้อมูลนั้นเราแบ่งข้อมูลที่ใช้ทดสอบออกเป็น 3 กลุ่มข้อมูลคือ

1. ข้อมูลตัวอักษรคือข้อมูลของการใช้งานเว็บเพจต่างๆ โดยไม่รวมรูปภาพ หรืองานบริการอื่นๆ ที่มีการส่งข้อมูลเป็นตัวอักษรเช่น SMTP, POP3 เป็นต้น
2. ข้อมูลรูปภาพคือข้อมูลที่เป็นข้อมูลภาพต่างๆเช่นไฟล์ .jpg ,.bmp, .gif ฯลฯ
3. ข้อมูลไบนารีคือข้อมูลอื่นๆ ที่รับส่งในรูปแบบไบนารีเช่นการถ่ายโอนไฟล์ หรือการส่งข้อมูลที่มีข้อมูลไบนารีอื่นๆ

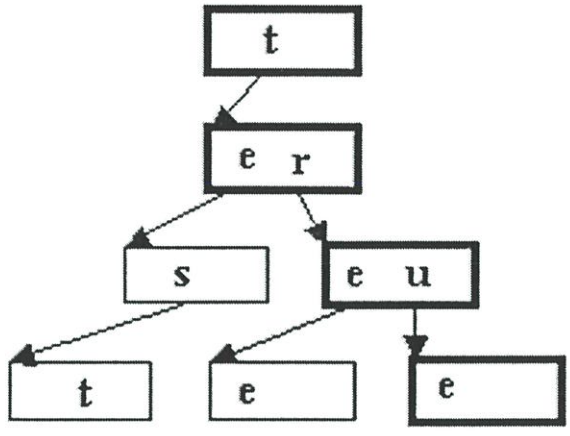
4. ข้อมูลรวมทั้งหมดเป็นข้อมูลเหมือนกับข้อมูลที่ผ่านไปมาปกติในเครือข่ายคือจะมีทั้งข้อมูลที่เป็นตัวอักษร รูปภาพ และข้อมูลไบนารี ในสัดส่วนที่ต่างๆ กันแล้วแต่การใช้งาน

6.4 เปรียบเทียบในแต่ละโครงสร้าง

ในการทดสอบสมมุติฐานจะเปรียบเทียบกระบวนการทำงานของการค้นหา 3 รูปแบบคือ SNORT ปกติ version 1.6.3 ซึ่งใช้อัลกอริทึมบอยเออร์-มัวร์ ในการเปรียบเทียบรูปแบบการบุกรุก ระบบที่สองคือ โปรแกรม SNORT ที่เปลี่ยนแปลงโครงสร้างเป็นแบบ burst tries ส่วนระบบสุดท้ายคือระบบที่เปลี่ยนแปลงจากระบบที่สองให้มีการเปลี่ยนแปลงรูปแบบของกฎและรูปแบบของข้อมูลในแพ็กเก็ตเป็นแบบตัวแบ่งและความยาวคำก่อนนำมาจัดเข้าเป็นโครงสร้างแบบ burst triesโดยในการทดสอบข้อสมมุติฐานนั้น งานวิจัยนี้ใช้วิธีนับจำนวนการเปรียบเทียบตัวอักษรในการทำงานทั้งสามแบบ โดยการนับจำนวนการเปรียบเทียบจะนับทุกๆครั้งที่เกิดการเปรียบเทียบตัวอักษรขึ้นในระบบ

สาเหตุที่ไม่นำเอาเวลาของการทำงานมาเปรียบเทียบนั้น เนื่องจาก ระบบปฏิบัติการที่เป็นระบบทดสอบนั้นมีการทำ context switch เกิดขึ้นในระหว่างการทดสอบซึ่งเป็นการทำงานของระบบซึ่งไม่สามารถควบคุมได้ ทำให้ค่าเวลาที่ได้จากการวัดด้วยฟังก์ชั่น time() นั้นคลาดเคลื่อนจากปกติมาก จนไม่สามารถนำมาเปรียบเทียบกันได้ จึงใช้วิธีการเปรียบเทียบจำนวนการเปรียบเทียบตัวอักษรในกฎแทน

วิธีการนับการเปรียบเทียบตัวอักษรในการทดลองนี้จะใช้วิธีนับทุกๆ การเปรียบเทียบตัวอักษรเช่นในรูปที่ 6.3 จะเปรียบเทียบข้อมูลคำว่า intrusion ใน tries ในการเปรียบเทียบนี้จะมีการเปรียบเทียบตัวอักษรคือ จะนำคำว่า intrusion เปรียบเทียบในโครงสร้างกฎก่อน ซึ่งเมื่อเปรียบเทียบตัว i แล้วไม่อยู่ในโครงสร้างกฎ จึงนำคำว่า ntrusion มาเปรียบเทียบในกฎต่อไป แต่ตัวอักษร n ก็ยังไม่ได้อยู่ในโครงสร้างกฎอีกเช่นเดียวกัน ในการเปรียบเทียบคำว่า intrusion และ ntrusion จึงมีการเปรียบเทียบตัวอักษรละหนึ่งครั้ง



รูปที่ 6.3 ตัวอย่างวิธีการนับการเปรียบเทียบตัวอักษรคำว่า intrusion

แต่เมื่อนำคำว่า trusion มาเปรียบเทียบในโครงสร้างกฎจะพบว่าชุดของตัวอักษร tru มีอยู่ในโครงสร้างกฎซึ่งจะต้องมีการเปรียบเทียบตัวอักษร 3 ครั้ง แต่ก็ยังไม่ match ทั้งหมดเนื่องจากไม่มีตัวอักษร s ในโหนดสุดท้าย จำนวนการเปรียบเทียบตัวอักษรสำหรับคำว่า trusion จึงเป็น 4 ครั้ง หลังจากนั้นจึงต้องเปรียบเทียบคำว่าคำว่า rusion , usion , sion , ion , on และ n ต่อไปตามลำดับ ซึ่งก็ไม่ match กับตัวอักษร t ในโหนดแรกของ tries เลยจึงมีการเปรียบเทียบคำละ 1 ครั้ง รวมจำนวนการเปรียบเทียบทั้งหมดของคำว่า intrusion คือ 12 ครั้ง

สำหรับกรณีของ delimiter search จะมีการนับการเปรียบเทียบตัวแปรในสองส่วนคือ นับจำนวนการอ่านตัวอักษรของข้อมูลในแพ็กเก็ตในขั้นตอนการเปลี่ยนรูปแบบของข้อมูลให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งระหว่างคำ และส่วนของการเปรียบเทียบตัวอักษรในคำที่ถูกย่อแล้ว ซึ่งกระบวนการที่สองจะคล้ายกับการนับจำนวนการเปรียบเทียบตัวอักษรในโครงสร้างกฎแบบ burst tries ข้างต้น โดยค่าที่ได้จากการนับในส่วนแรกจะมีค่าเท่ากับความยาวของข้อมูลในแพ็กเก็ตนั้นๆ พอดี

และกรณีของโครงสร้างกฎของโปรแกรม SNORT เดิมนั้น จะนับจำนวนการเปรียบเทียบตัวอักษรที่เกิดขึ้นการทำงานของฟังก์ชั่น good-suffix shift และ bad-character shift ทั้งหมดที่เกิดขึ้นในการตรวจสอบแพ็กเก็ต 1 แพ็กเก็ต ผลลัพธ์ที่ได้จะเป็นผลรวมของการนับจำนวนการเปรียบเทียบตัวอักษรในโหนดกับข้อมูลในแพ็กเก็ตจนครบทุกโหนด

6.4.1 ผลการนับในการทดสอบ

ในการทดสอบสมมุติฐานจะใช้การเปรียบเทียบผลลัพธ์ของการทำงานสามแบบด้วยกันคือ

1. โปรแกรม SNORT ปกติ ที่มีโครงสร้างข้อมูลแบบลิงค์ลิสต์สองมิติ ใช้อัลกอริทึมบอยเออร์-มัวร์ช่วยในการค้นหา

2. โปรแกรม SNORT ที่มีการปรับเปลี่ยนโครงสร้างข้อมูลของกฎให้อยู่ในรูปของ burst tries ซึ่งมีการค้นหาข้อมูลโดยการนำ trie เข้าไปค้นหาในข้อมูล
3. โปรแกรม SNORT ที่มีการปรับเปลี่ยนโครงสร้างข้อมูลของกฎให้อยู่ในรูปของ burst tries โดยมีการเปลี่ยนรูปแบบของกฎและข้อมูลให้อยู่ในรูปคำคั่นด้วยตัวแบ่งระหว่างคำก่อนที่จะตรวจสอบกฎ

ตารางที่ 6.2 ตารางเปรียบเทียบจำนวนการเปรียบเทียบตัวอักษรในโครงสร้าง 3 โครงสร้างคือ โครงสร้างของ SNORT ปกติ , burst tries และ delimiter burst tries

ความยาวของข้อมูลในแพ็กเก็ต	จำนวนการเปรียบเทียบใน SNORT	จำนวนการเปรียบเทียบใน burst tries	จำนวนการเปรียบเทียบใน Delimiter tries
103	2998	151	149
165	4714	233	220
204	5994	293	269
250	7267	353	322
302	8972	441	401
348	10413	498	437
400	11855	577	515
450	13200	660	584
500	14732	753	680
551	16118	844	786

หลังจากที่เขียนโปรแกรมเพื่อเปลี่ยนแปลงโครงสร้างข้อมูลให้กลายเป็นแบบ burst trie และสร้างโปรแกรมย่อยในการเปลี่ยนแปลงข้อมูลในแพ็กเก็ต และในกฎให้อยู่ในรูปของความยาวตัวอักษรคั่นด้วยตัวแบ่งระหว่างคำ พร้อมทั้งเพิ่มส่วนของโปรแกรมในการนับจำนวนการเปรียบเทียบตัวอักษรเพื่อให้ได้โปรแกรมทดสอบทั้งสามเรียบร้อยแล้ว จะรันโปรแกรมทั้งสามเพื่อเปรียบเทียบการทำงานโดยใช้กฎสำหรับตรวจสอบการบุกรุกที่พอร์ต 80 จำนวน 290 กฎ และใช้ข้อมูลตัวอย่างจากการทำงานเพื่อดึงข้อมูลทั้งหมดจากเว็บไซต์ www.kmitl.ac.th ทั้งหมด 1107 แพ็กเก็ต มีขนาดแพ็กเก็ตโดยเฉลี่ยคือ 364.27 ไบต์ โดยข้อมูลเป็นตัวอักษรและข้อมูลรูปภาพ เพื่อทดสอบการทำงานของโปรแกรมทั้งสาม แล้วนำมาเรียงลำดับตามขนาดของแพ็กเก็ต สำหรับชุด

ของตัวแบ่งที่ใช้ในการทดสอบจะใช้ชุดของตัวแบ่งคือ "-/,:?_AEIOUBCWT" ตัวอย่างของผลลัพธ์ที่ได้แสดงในตารางที่ 6.2

จากตารางที่ 6.2 เราจะเห็นได้ว่า ในการทดลองนี้จำนวนการเปรียบเทียบตัวอักษรของทุกๆ โครงสร้างข้อมูลจะแปรผันโดยตรงกับขนาดของข้อมูลในแพ็กเก็ต โดยจำนวนการเปรียบเทียบตัวอักษรในโครงสร้างกฎแบบลิงค์ลิสต์สองมิติของโปรแกรม SNORT นั้นมีการเปรียบเทียบตัวอักษรมากที่สุด รองลงมาคือโครงสร้างกฎแบบ burst trie ที่ไม่มีการย่อขนาดของข้อมูล และการทำงานที่มีการเปรียบเทียบตัวอักษรน้อยที่สุดคือ โครงสร้างกฎแบบ burst trie ที่มีการย่อขนาดของข้อมูลให้อยู่ในรูปแบบของความยาวคำ คั่นด้วยตัวแบ่งระหว่างคำ โดยจำนวนการเปรียบเทียบตัวอักษรในโครงสร้างแบบ burst tries นั้นจะมีจำนวนการเปรียบเทียบน้อยกว่าโครงสร้างเดิมของ SNORT คือจะมีจำนวนการเปรียบเทียบตัวอักษรเหลือเพียงแค่ 4.96 เปอร์เซ็นต์เท่านั้น ซึ่งในกรณีของโครงสร้างกฎแบบ burst trie ที่มีการย่อขนาดของข้อมูลให้อยู่ในรูปแบบของความยาวคำจะมีจำนวนการเปรียบเทียบตัวอักษรน้อยกว่า burst tries เล็กน้อย ประมาณ 9.29 เปอร์เซ็นต์

ผลจากการทำงานดังกล่าวพบว่า จากขนาดแพ็กเก็ตโดยเฉลี่ย 364.27 ไบต์ สำหรับโครงสร้างกฎเดิมของโปรแกรม SNORT จะมีการเปรียบเทียบตัวอักษรโดยเฉลี่ยคือ 10764.99 ครั้ง ส่วนโครงสร้างแบบ burst tries ที่ไม่ได้บีบอัดข้อมูลนั้นจะมีการเปรียบเทียบตัวอักษรเฉลี่ย 531.35 ครั้ง ส่วนโครงสร้างแบบ burst tries ที่มีการบีบอัดข้อมูลโดยใช้ตัวแบ่งนั้นมีการเปรียบเทียบตัวอักษรเฉลี่ย 478 ครั้ง

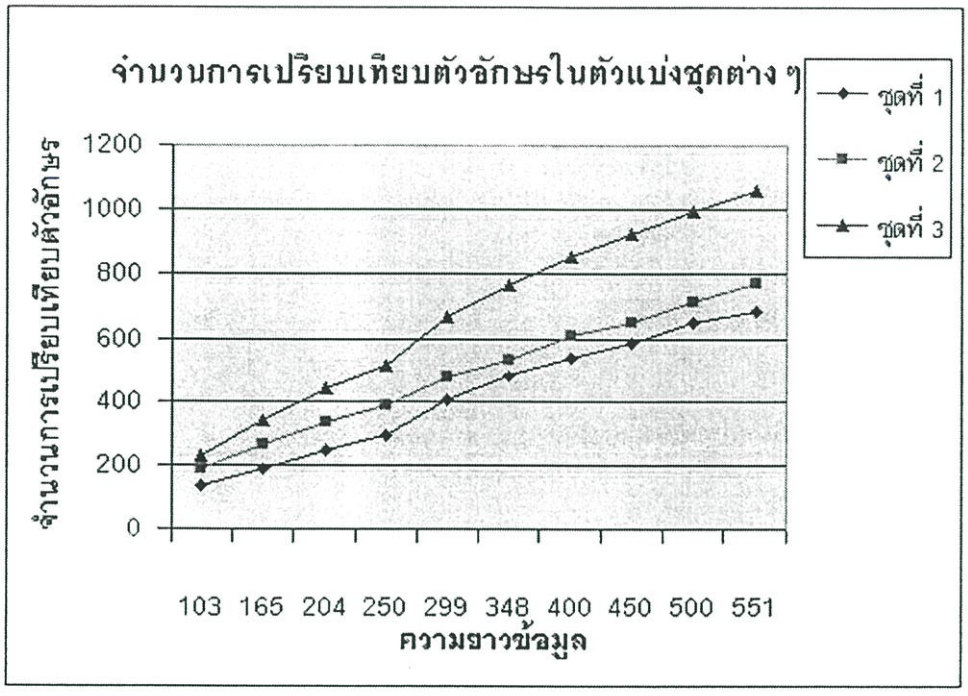
สำหรับการทำงานในโครงสร้างที่เป็น burst tries นั้น เมื่อลองพิจารณาตามอัลกอริทึม และการทำงานของโปรแกรมที่ส่วนใหญ่จะเป็นการทำงานที่ค้นหาแล้วไม่พบความผิดปกติใดๆ การเข้าไปค้นหาใน tries ในชั้นลึกๆ จึงเป็นผลให้สูญเสียเวลาในการเปรียบเทียบตัวอักษรมากกว่า ดังนั้นการทำงานที่เข้าไปค้นหาใน tries ในระดับต้นๆ จะมีผลให้การทำงานมีประสิทธิภาพมากกว่า จากการเปรียบเทียบการเข้าไปค้นหาใน tries โดยคิดเป็นเปอร์เซ็นต์ของจำนวนการค้นหาโดยเฉลี่ยในโปรแกรมทดสอบทั้งสองประเภทนั้นเราได้ผลลัพธ์ดังตารางที่ 6.3

ตารางที่ 6.3 เปอร์เซ็นต์การค้นหาเข้าไปใน trie โดยเฉลี่ยของโครงสร้างกฎแบบ burst trie

ลำดับที่ ใน trie	Burst tries (เปอร์เซ็นต์)	Delimiter Burst Tries (เปอร์เซ็นต์)
1	61.76	14.6
2	32.18	56.21
3	4.96	18.9
4	0.78	7.76
5	0.02	0.47
6	0.25	2
7	0.03	0.14
8	0	0
9	0	0
10	0	0

จากตารางเราสามารถสรุปได้ว่าการทำงานของโครงสร้างกฎแบบ burst tries ที่มีการบีบอัดข้อมูลก่อนมีการสูญเสียเวลาในการเปรียบเทียบตัวอักษรไปเปล่าๆ มากกว่าโครงสร้างกฎที่ไม่มีการบีบอัดข้อมูล โดยผลลัพธ์ที่ได้ในโปรแกรมทดสอบที่มีโครงสร้างเป็น burst trie ที่ไม่ผ่านการบีบอัดข้อมูลนั้น จะมีการตรวจสอบแพ็กเก็ตในระดับที่ 1 ของ burst tris แล้วพบว่าไม่เหมือนกันอยู่ 61.76 เปอร์เซ็นต์ โดยไม่ต้องเสียเวลาในการเปรียบเทียบตัวอักษรหลายๆ ระดับเลย ต่างกับโครงสร้างกฎแบบ burst tries ที่ผ่านการบีบอัดข้อมูลโดยจะมีการตรวจสอบข้อมูลแล้วพบว่าไม่เหมือนกันในครั้งแรกเพียง 14.6 เปอร์เซ็นต์เท่านั้น

ผลการทดสอบที่ได้นั้น ไม่ได้มีผลลัพธ์ดังที่ได้แสดงในตาราง 6.1 ในทุกๆ ชุดของตัวแบ่งจากการทดสอบเมื่อเลือกชุดของตัวแบ่งต่างชุดกัน มาทำงานเปรียบเทียบกันจะได้ผลลัพธ์ต่างกันเพื่อประสิทธิภาพสูงสุดในการทำงาน การลด search space ด้วยการเปลี่ยนรูปของข้อมูลโดยใช้ตัวแบ่งระหว่างคำนั้น จะต้องมาพิจารณาชุดของตัวแบ่งที่จะนำมาใช้ด้วยว่าจะใช้แบบไหน เนื่องจากถ้าใช้ชุดของตัวแบ่งที่ไม่เหมาะสม จะกลายเป็นการเพิ่ม search space แทน จากรูปที่ 6.4 เป็นผลการทดลองโดยใช้ตัวแบ่ง 3 ชุดคือชุดที่ 1 ใช้ชุดของตัวแบ่งคือ "-/,;?_" ชุดที่ 2 ใช้ชุดของตัวแบ่งคือ "-/,;?_AEIOU" และชุดที่ 3 ใช้ชุดของตัวแบ่งคือ "-/,;?_AEIOU " ใช้ข้อมูลทดสอบคือข้อมูลตัวอักษรจากการดึงข้อมูลทั้งหมดจาก www.kmitl.ac.th และ www.ce.kmitl.ac.th แล้วนำมาแยกแยะการทำงานตามความยาวของแพ็กเก็ต โดยผลลัพธ์ที่ได้นั้นชุดที่ 1 ให้ผลลัพธ์การทำงานดีที่สุด ส่วนชุดที่ 3 จะให้ผลลัพธ์การทำงานที่แย่ที่สุด



รูปที่ 6.4 จำนวนการเปรียบเทียบตัวอักษรของตัวแบ่งชุดต่างๆ

เมื่อผลการทดสอบสรุปได้ว่าชุดของตัวแบ่งมีผลอย่างมากต่อจำนวนการเปรียบเทียบตัวอักษรในการทำงานถ้าเลือกกลุ่มของตัวแบ่งดีก็จะมีจำนวนการเปรียบเทียบตัวอักษรน้อย ทำให้ทำงานเร็วขึ้น และถ้าเลือกกลุ่มของตัวแบ่งไม่ดีก็จะส่งผลให้จำนวนการเปรียบเทียบตัวอักษรมากขึ้นส่งผลให้การทำงานช้าลง เมื่อนำการค้นหาโดยใช้ตัวแบ่งระหว่างคำไปใช้งานจึงต้องมีการทดสอบเพื่อหาว่าตัวแบ่งชุดใดจะดีที่สุดสำหรับการทำงานหนึ่งๆ และมีหลักการเลือกตัวแบ่งอย่างไรในปัญหาแต่ละปัญหาด้วย

6.5 ทดลองหาชุดของตัวแบ่งที่ควรใช้

จากการวิเคราะห์ตัวอักษรที่จะนำมาใช้เป็นตัวแบ่ง สามารถใช้ได้ทั้งหมดในตาราง ASCII คือ 256 ตัวอักษรซึ่งเมื่อดูจากกฎและข้อมูลซึ่งเป็นข้อมูลที่นำมาแสดงผลได้นั้นจะสามารถใช้งานได้เพียง 95 ตัวอักษรคือตัวอักษรที่ 32 ถึงตัวอักษรที่ 126 ในตาราง ASCII เท่านั้น สำหรับกลุ่มของตัวแบ่งนั้นสามารถเป็นได้ทั้งหมดคือตั้งแต่ 1 ตัว จนถึง 95 ตัวอักษร โดยความเป็นไปได้ทั้งหมดของกลุ่มตัวแบ่งสามารถแยกแยะได้ตามจำนวนตัวอักษรที่ใช้ได้ดังตารางที่ 6.4

ตารางที่ 6.4 จำนวนความเป็นไปได้ทั้งหมดของชุดตัวแบ่งที่จำนวนตัวอักษรในกฎต่างๆ กัน

จำนวนตัวอักษรในกฎ	จำนวนความเป็นไปได้ทั้งหมดของชุดตัวแบ่ง
1	95
2	8930
3	830490
4	76405080
...	...
94	1.033×10^{148}
95	1.033×10^{148}
รวม	2.808×10^{148}

จากตารางความเป็นไปได้ทั้งหมดของชุดตัวแบ่งจะเห็นได้ว่ากรณีที่เกิดขึ้นทั้งหมดคือ 2.808×10^{148} ตัวอักษรซึ่งมีขนาดใหญ่มากถ้าจะต้องทดสอบในทุกๆ กรณีจึงจำเป็นจะต้องลดจำนวนตัวอักษรที่จะทดสอบลงให้มากที่สุดโดยพิจารณาลักษณะของกฎและข้อมูลประกอบกัน จากการทดสอบคุณสมบัติของตัวแบ่งพบว่าคุณสมบัติของตัวแบ่งที่ดีนั้นควรมีคุณสมบัติดังนี้คือ

6.5.1 ไม่ควรพบมากในกฎ

สำหรับการใช้งานตัวแบ่งนั้นเราจะใช้สำหรับการคั่นระหว่างความยาวของคำ ซึ่งถ้าพบตัวแบ่งมากๆ ในกฎแล้วแสดงว่าเมื่อผ่านกระบวนการแปลงกฎให้อยู่ในรูปแบบของความยาวคำคั่นด้วยตัวแบ่งแล้ว จะยังมีความยาวไม่น้อยลงเลย หรือความยาวน้อยลงไม่มาก ยกตัวอย่างกรณีที่แย่ที่สุดสำหรับการเลือกตัวอักษรก็คือ ถ้าเซตของตัวแบ่งคือเซตของตัวอักษรทั้งหมดในกฎ จะได้ความยาวกฎเพิ่มขึ้นเป็นสองเท่าของความยาวกฎเดิมเช่น

คำที่อยู่ในกฎคือ

"_vti_bin/shtml.exe"

เซตของตัวแบ่งคือ

“_./behilmnstvx”

หลังจากเปลี่ยนรูปแล้วจะได้รูปแบบของความยาวคำคั่นด้วยตัวแบ่งคือ

“0_0v0t0i0_0b0i0n0/0s0h0t0m0l0.0e0x0e0”

ซึ่งความยาวของคำสำคัญจะกลายเป็น 2n+1 เมื่อ n คือความยาวของคำในกฎ

6.5.2 ควรมีอย่างน้อย 2-3 ตัวอักษรใน 1 กฎ

ในการตรวจสอบกฎเมื่อเปลี่ยนรูปแบบของกฎไปเป็นความยาวคำคั่นด้วยตัวแบ่งระหว่างคำแล้ว รูปแบบที่จะแยกแยะความแตกต่างระหว่ากฎก็คือ คำที่คั่นอยู่ระหว่างตัวแบ่งนั่นเอง การที่มีตัวอักษรที่เป็นตัวแบ่งอยู่มากๆ ในกฎ ก็หมายถึงจำนวนความยาวคำก็จะมากขึ้นตาม

สิ่งที่จะมาแทนตัวอักษรในกฎทั้งกฎนั้นคือความยาวคำที่คั่นด้วยตัวแบ่ง ซึ่งก็หมายความว่าต้องมีตัวแบ่งอย่างน้อยสองตัวในกฎนั้น จึงจะทำให้เกิดความยาวคำนั้นๆ ได้ ตัวอย่างเช่น "_vti_bin/shtml.exe" จะมีตัวอักษรที่สามารถอยู่ในชุดของตัวแบ่งได้ทั้งหมด แต่เมื่อพิจารณาตามข้อ 1 แล้วยิ่งมีตัวแบ่งมากจะยิ่งไม่ดี สำหรับในตัวอย่างนี้ ตัวอักษรที่ควรนำมาใช้เป็นตัวแบ่งคือ "_" , "i" , "e" , "t" หรือตัวอักษรที่ทำการประกอบกันแล้วสามารถสร้างรูปแบบที่แทนที่คำทั้งหมดได้ เช่น "vb" , "sl" หรืออื่นๆ

6.5.3 การเลือกชุดของตัวแบ่งต้องคำนึงถึงกฎทั้งหมดด้วย

การเลือกตัวอักษรนั้นต้องคำนึงถึงกฎส่วนใหญ่ด้วย ไม่ใช่จะคำนึงเฉพาะคำในกฎเพียงกฎเดียว เพราะการเลือกตัวแบ่งแต่ละตัวจะมีผลต่อขนาดของข้อมูลที่แปลงรูปแล้ว การเพิ่มตัวแบ่งเข้าไปในเซตของตัวแบ่งเพียงตัวเดียวอาจทำให้ขนาดของข้อมูลเปลี่ยนไปได้มากทีเดียวเช่นในตัวอย่างกฎในรูปที่ 6.5

"_vti_inf.html"

"_private/registrations.txt"

"_vti_bin/shtml.exe"

"_vti_pvt/service.cnf"

"_private/form_results.htm"

รูปที่ 6.5 ตัวอย่าง content ในกฎ

จากรูป ถ้าเลือกเฉพาะ "_" เป็นตัวแบ่งจะไม่สามารถเปลี่ยนรูปกฎที่มี content เป็น "_private/registrations.txt" ได้ ถ้าเลือก "_/" หรือ ก็จะสามารถแก้ปัญหาตรงนี้ได้ ชุดของตัวอักษรที่ควรใช้จะกลายเป็น "_/" หรือ "_." เพราะสามารถใช้ได้กับกฎทุกๆ กฎ แต่ก็ต้องยอมแลกด้วยการลดขนาดของข้อมูลที่จะทำได้น้อยลงด้วย

จากคุณสมบัติของตัวแบ่ง ที่ขัดแย้งกันนั้น ทำให้เกิดปัญหาขึ้นในกรณีที่กฎมีมากขึ้น เช่น ในกรณีที่ชุดของกฎที่นำมาใช้คือ "_private/registrations.txt" , "_private/form_results.htm" , "_vti_bin/shtml.exe" และ "_vti_pvt/service.cnf" ซึ่งควรใช้ชุดของตัวแบ่งคือ "/." จะเหมาะสมมาก

เนื่องจากใช้ได้ทุกกฎ และมีไม่มากในกฎแต่ละข้อ เมื่อใช้แล้วข้อมูลจะถูกย่อลงให้กลายเป็น “8/13.3” , “8/5.3” , “8/7.3” และ “8/12.3”

แต่เมื่อเพิ่มกฎเข้ามาอีก 1 กฎคือ "_vti_inf.html" กลายเป็นชุดของกฎคือ "_vti_inf.html" , "_private/registrations.txt" , "_vti_bin/shtml.exe" , "_vti_pvt/service.cnf" และ "_private/form_results.htm" ตัวแบ่งที่เหมาะสมจะเป็น “_/.” ทันที เพราะใช้ได้กับทุกๆ กฎ ซึ่งทำให้ขนาดของข้อมูลที่ต้องเปรียบเทียบกลายเป็น “0_3_3.4” , “0_7/13.3” , “0_3_3/5.3” , “0_3_3/7.3” และ “0_7/4_7.3” ทันที ซึ่งเมื่อนับจำนวนตัวอักษรทั้งหมดแล้วจะมากกว่าในกรณีแรกมาก

เพื่อการเพิ่มความเร็วอย่างมีประสิทธิภาพ ผู้ใช้งานต้องเลือกชุดของตัวแบ่งที่สามารถเปลี่ยนแปลงรูปแบบและเหมาะสมกับกฎให้มากที่สุดด้วยคือ ควรกระจายอยู่มากในกฎทุกๆ กฎ แต่เมื่อนับตัวอักษรโดยรวมแล้วจะมีจำนวนน้อยที่สุด จากปัญหาความขัดแย้งกันในหลักการเลือกตัวแบ่ง ที่ไม่ควรพบตัวอักษรนั้นมากในกฎหนึ่งๆ และในแต่ละกฎควรมีอย่างน้อย 2-3 ตัวอักษรก็เพียงพอ สำหรับการเปลี่ยนแปลงรูปแบบกฎเพื่อแทนที่กฎใหญ่ๆ ทั้งกฎ แต่เมื่อกฎเพิ่มขึ้น ก็ต้องมีตัวอักษรที่อยู่ในเซตของตัวแบ่งมากขึ้นตาม จึงกลายเป็นปัญหา optimize ในที่สุด

6.5.4 ไม่ควรพบตัวแบ่งมากในข้อมูลที่จะค้นหา

ทำนองเดียวกันกับหลักการเลือกตัวแบ่งในข้อ 6.4.1 คือไม่ควรพบตัวแบ่งมากในกฎ ในกรณีของข้อมูลที่ต้องค้นหาก็เช่นเดียวกัน เนื่องจากในการทำงานทุกๆ ครั้งที่ได้รับแพ็กเก็ตเข้ามาเพื่อตรวจสอบแพ็กเก็ตที่ได้รับเข้ามานั้นต้องถูกเปลี่ยนรูปโดยใช้ชุดของตัวแบ่งนั้นๆ ด้วย ถ้าตัวแบ่งมีอยู่ในข้อมูลมากๆ แล้ว ข้อมูลจะถูกขยายขึ้นคล้ายกับตัวอย่างในข้อ 1 สำหรับการเลือกตัวอักษรที่ควรหลีกเลี่ยงนี้ ก็จะขึ้นอยู่กับข้อมูลด้วยว่าเป็นข้อมูลแบบไหน เนื่องจากจะมีความแตกต่างกันมาก

จากหลักการทั้งสี่ข้อเราจึงนำมาใช้หากลุ่มของตัวแบ่งที่ทำให้การทำงานเป็นไปได้อย่างมีประสิทธิภาพ เริ่มจาก

1. เขียนโปรแกรมเพื่อค้นหาตัวอักษรที่มีการกระจายมากที่สุดในกฎ

   จากการทดสอบเพื่อหาตัวอักษรที่มีการกระจายมากที่สุดเพื่อนำไปใช้เป็นตัวแบ่งโดยใช้ชุดของกฎทดสอบจำนวน 290 กฎในกฎได้ผลลัพธ์ดังตารางที่ 6.5

ตารางที่ 6.5 ผลการค้นหาการกระจายของตัวอักษรในกฎ 20 อันดับแรก

อันดับที่	ตัวอักษร	จำนวนกฎที่พบตัวอักษร
1	.	200
2	S	177
3	I	174
4	E	173
5	/	172
6	C	156
7	P	155
8	A	151
9	R	147
10	T	146
11	D	130
12	N	125
13	M	119
14	O	117
15	L	112
16	F	90
17	G	89
18	H	85
19	W	70
20	X	62

2. เขียนโปรแกรมเพื่อค้นหาตัวอักษรที่มีมากที่สุดในกฎ

จากการเขียนโปรแกรมเพื่อหาจำนวนตัวอักษรมากที่สุดในกฎทั้งหมด 290 กฎเพื่อให้ทราบว่าตัวอักษรใดที่มีจำนวนมากที่สุด ซึ่งถ้าพบตัวอักษรนั้นน้อยก็ไม่เหมาะที่จะนำมาเป็นตัวแบ่ง และถ้าพบตัวอักษรนั้นมากจนเกินไป ก็จะไม่ควรนำมาใช้เป็นตัวแบ่งเช่นกัน ในกรณีนี้ใช้กฎจำนวน 290 กฎ ถ้าพบตัวอักษรมากเกินสองเท่าของจำนวนกฎคือ 580 ตัวอักษร ก็จะไม่เหมาะสมในการใช้งาน ผลการทดลองได้ผลลัพธ์ดังตารางที่ 6.6

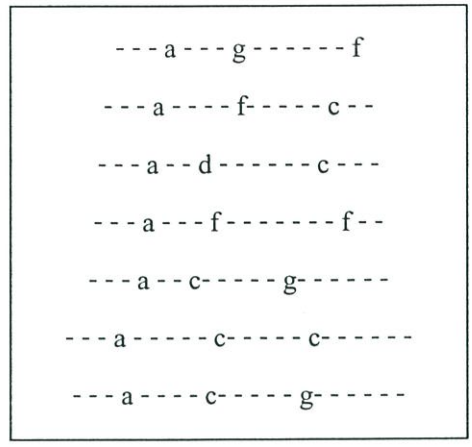
ตารางที่ 6.6 ตารางแสดงผลการนับตัวอักษรในกฎ จำนวน 20 ตัวอักษรเรียงจากมากไปน้อย

อันดับที่	ตัวอักษร	จำนวนตัวอักษรทั้งหมดในชุดกฎ
1	S	322
2	E	313
3	I	300
4	C	252
5	T	252
6	/	251
7	A	243
8	.	237
9	P	220
10	R	204
11	D	175
12	O	159
13	L	154
14	M	153
15	N	153
16	F	121
17	G	99
18	H	93
19	W	93
20	U	65

จากตารางที่ 6.5 และ 6.6 เราจะได้กลุ่มของตัวอักษรที่มีการกระจายใน 10 อันดับแรกคือ ".SIE/CPART" และเราจะได้กลุ่มตัวอักษรที่มีอยู่มากที่สุดในกฎ 10 อันดับแรกคือ "SEICT/A.PR" ซึ่งเมื่อวิเคราะห์ดูความซ้ำซ้อนจากตารางทั้งสองแล้วพบว่าโดยเฉลี่ยแล้วกฎจะมีตัวอักษรตัว "S", "E" และเครื่องหมาย "." กระจายอยู่ในกฎมากที่สุด และมีปริมาณในกฎมากด้วย ซึ่งมีความเหมาะสมในการนำมาใช้เป็นตัวแบ่งที่สุด

3. หา combination ที่ดีที่สุดในกฎ

สำหรับการหาชุดของตัวแบ่งนั้น ถือว่า combination มีความสำคัญมากที่สุดเนื่องจากถ้านำตัวแบ่งแต่ละตัวมาเข้าคู่กันแล้วเกิดความซ้ำซ้อนกันมากในกฎ ก็จะกลายเป็นชุดของกฎที่ไม่ดี ถึงแม้ว่าจะเป็นตัวอักษรที่มีการกระจายมากที่สุด และก็มีน้อยในกฎก็ตาม ยกตัวอย่างเช่น



รูปที่ 6.6 ตัวอย่างการเลือก combination ในชุดของตัวแบ่ง

จากตัวอย่างรูปที่ 6.6 จะเห็นได้ว่าจำนวนตัวอักษรเรียงจากมากไปน้อยคือ “a” , “c” , “f” , “g” และ “d” ตามลำดับ แต่เมื่อนำตัวอักษรคือ “ac” “af” หรือ “ag” เข้าชุดกันเพื่อใช้เป็นตัวแบ่งแล้วจะเห็นว่าไม่สามารถใช้งานชุดตัวแบ่งดังกล่าวได้เนื่องจากไม่สามารถใช้งานในบางกฎได้ แต่ชุดของตัวอักษรที่เหมาะสมมากกว่าที่จะใช้เป็นชุดตัวแบ่งคือ “cdfg” ซึ่งใช้ตัวอักษรที่มีการกระจายน้อยกว่า แต่สามารถนำมาใช้งานร่วมกันได้เป็นอย่างดี

4. เขียนโปรแกรมเพื่อค้นหาตัวอักษรที่มีมากที่สุดในข้อมูล

การทดสอบอีกอันหนึ่งเพื่อหาตัวอักษรที่เหมาะสมจะนำมาเป็นตัวแบ่งระหว่างคำในกฎคือการหาตัวอักษรที่มีมากที่สุดในข้อมูล เนื่องจากถ้ามีตัวอักษรนั้นๆ มากจนเกินไปก็จะไม่เหมาะในการนำมาเป็นตัวแบ่ง เพราะจะใช้เวลาในการสร้างข้อมูลในรูปของความยาวคำและตัวแบ่งระหว่างคำนานมาก และจะไม่สามารถย่อส่วนของข้อมูลได้มากนัก

ตารางที่ 6.7 ผลการเรียงลำดับจำนวนตัวอักษรที่พบในข้อมูลจากข้อมูลทดสอบจากมากไปน้อย จำนวน30 อันดับ

อันดับที่	ตัวอักษร	จำนวนที่ปรากฏในข้อมูล
1	[space bar]	3608719
2	T	1261186
3	E	826608
4	>	821977
5	<	815131
6	N	665626
7	I	659482
8	A	606220
9	D	595786
10	O	593026
11	R	574108
12	=	548002
13	F	545335
14	‘	533229
15	/	499368
16	“	438663
17	H	400668
18	C	395749
19	L	382311
20	S	375493
21	G	352556
22	B	294471
23	P	259850
24	0	251076
25	2	212658
26	M	200639

ตารางที่ 6.7 (ต่อ)

27	.	155854
28	1	143549
29	V	135765
30	W	134984

จากตารางที่ 6.7 จากการเปรียบเทียบตัวอักษรทั้งหมด 20034161 ตัวอักษร จากข้อมูลภายในเครือข่ายทดสอบ จะได้ตัวอักษรที่ปรากฏในข้อมูล 10 อันดับแรกคือ " TE><NIADO" ซึ่งไม่ควรนำมาใช้เป็นตัวแบ่งระหว่างคำ โดยเมื่อนำมาเปรียบเทียบกับตัวอักษรที่เกิดขึ้นทั้งหมด ถือว่าตัวอักษรที่มากที่สุดคือ " " มีจำนวนตัวอักษรคือ 3608719 ตัวอักษร คิดเป็น 18.01 เปอร์เซ็นต์ เป็นค่าที่สูงมาก ไม่ควรนำมาใช้เป็นตัวแบ่งระหว่างคำอย่างยิ่ง ถ้ามีการใช้เครื่องหมาย " " มาเป็นตัวแบ่งจะทำให้ลดขนาดข้อมูลลงได้ไม่มากนัก หรืออาจทำให้ข้อมูลขยายมากขึ้นกว่าเดิมด้วย

จากตารางที่ 6.5 6.6 และ 6.7 ทำให้พอสรุปได้ว่าชุดตัวอักษรที่เหมาะสมจะเป็นชุดของตัวแบ่งในการทำงานคือ "SC./PR" เพราะมีการกระจายดี มีจำนวนน้อยในข้อมูลและกฎ และตัวอักษรที่ควรหลีกเลี่ยงคือ " " , "T" และตัวอักษร "E"

6.6 การทดสอบเพื่อวิเคราะห์ข้อมูล

หลังจากนั้นจึงวิเคราะห์ว่าข้อมูลในเครือข่ายประเภทใดที่เหมาะสมสำหรับการใช้การค้นหาโดยใช้ตัวแบ่งระหว่างคำมากที่สุด โดยข้อมูลที่นำมาวิเคราะห์มี 3 รูปแบบคือ

1. ข้อมูลตัวอักษร ได้จากการดักจับข้อมูลในเครือข่ายขณะมีการดึงข้อมูลหน้าเว็บเพจต่างๆ ซึ่งข้อมูลมีจำนวน 66 แพ็กเก็ต มีขนาดแพ็กเก็ตที่เล็กที่สุดคือ 589 ไบต์ แพ็กเก็ตที่มีขนาดใหญ่ที่สุดมีขนาด 1442 ไบต์ และมีขนาดแพ็กเก็ตโดยเฉลี่ย 1381.21 ไบต์
2. ข้อมูลไบนารีไฟล์ได้จากการดักจับข้อมูลในเครือข่ายขณะที่มีการดึงข้อมูลไฟล์ต่างๆ ซึ่งข้อมูลมีจำนวนทั้งหมด 82 แพ็กเก็ต ข้อมูลในแพ็กเก็ตที่มีขนาดเล็กสุดคือ 346 ไบต์ และข้อมูลในแพ็กเก็ตที่มีขนาดใหญ่สุดคือ 1500 ไบต์ ซึ่งเฉลี่ยแล้วจะมีแพ็กเก็ตที่มีขนาด 1101.87 ไบต์
3. ข้อมูลรูปภาพ ได้จากการดักจับข้อมูลในเครือข่ายขณะที่มีการดึงข้อมูลรูปภาพ ซึ่งข้อมูลมีทั้งหมด 33 แพ็กเก็ต ข้อมูลในแพ็กเก็ตที่มีขนาดเล็กที่สุดคือ 496 ไบต์ และข้อมูลในแพ็กเก็ตที่มีขนาดใหญ่ที่สุดคือ 1370 ไบต์ โดยเฉลี่ยแล้วข้อมูลจะมีขนาด 1052.91 ไบต์

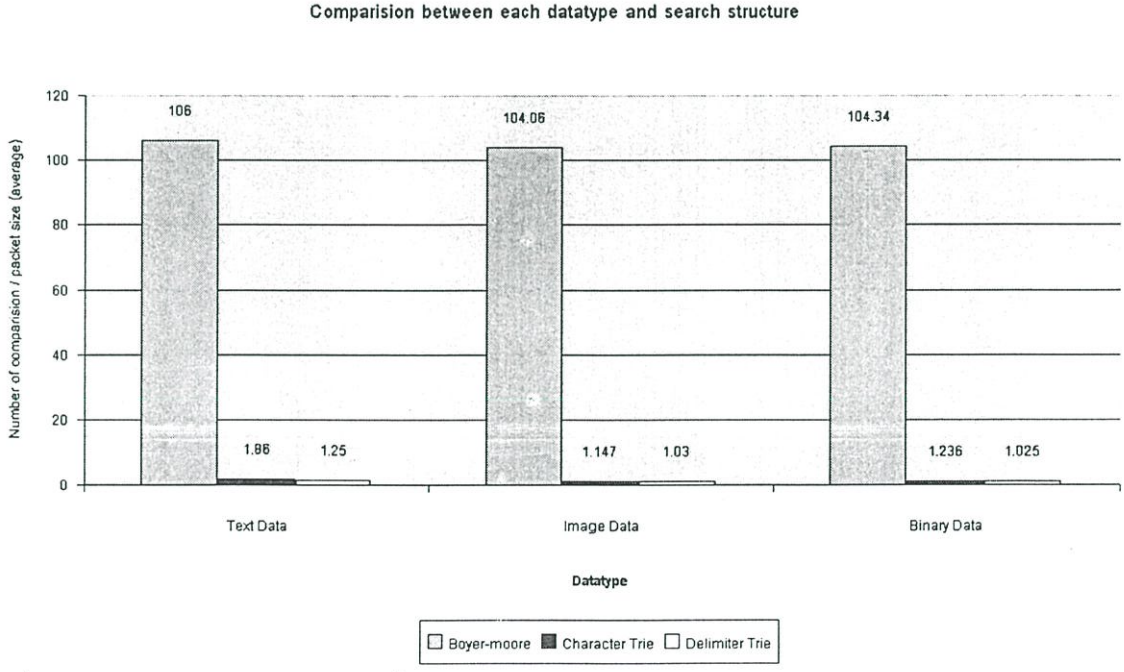
จากข้อมูลที่ดักจับได้ทั้งสามแบบ นำมาทดสอบกับกระบวนการทั้งสามแบบคือ การค้นหาของ SNORT , การค้นหาโดยใช้โครงสร้าง Trie ของตัวอักษร และการค้นหาโดยโครงสร้าง Trie

ของตัวแบ่ง โดยจุดประสงค์ในการทดสอบก็เพื่อหาว่ากระบวนการแต่ละแบบนั้น เหมาะสมกับข้อมูลประเภทใดบ้าง กระบวนการแบบใดที่ชนิดของข้อมูลไม่มีผลกระทบ การทดลองทำได้โดยมีการทดสอบย่อยๆ 3 การทดสอบคือ

การทดสอบที่ 1 นำข้อมูลทั้งสามชุด มาเข้ากระบวนการค้นหาโดยใช้วิธีการของ SNORT คือผ่านอัลกอริทึมบอยเออร์-มัวร์ แล้วนับจำนวนการเปรียบเทียบตัวอักษร แต่เนื่องจากข้อมูลที่นำมาทดสอบเป็นคนละชุดและขนาดของข้อมูลไม่เท่ากัน จึงใช้วิธีเปรียบเทียบจำนวนการเปรียบเทียบตัวอักษรต่อขนาดของแพ็กเก็ตแทน

การทดสอบที่ 2 นำข้อมูลทั้งสามชุดมาเข้ากระบวนการค้นหาโดยใช้โครงสร้าง Trie ของตัวอักษร แล้วนับจำนวนการเปรียบเทียบตัวอักษร และเปรียบเทียบจำนวนตัวอักษรต่อขนาดของข้อมูลแต่ละแบบ

การทดสอบที่ 3 นำข้อมูลทั้งสามชุดมาเข้ากระบวนการค้นหาโดยใช้โครงสร้าง Trie ของตัวแบ่ง โดยใช้เซตของตัวแบ่งคือ "./?;,_" แล้วนับจำนวนการเปรียบเทียบตัวอักษร และเปรียบเทียบจำนวนตัวอักษรต่อขนาดของข้อมูลแต่ละแบบ จากการทดสอบทั้งสามชุดเราสามารถสรุปความสัมพันธ์ได้ดังรูปที่ 6.7



รูปที่ 6.7 การเปรียบเทียบการทำงานเมื่อใช้ข้อมูลต่างกัน

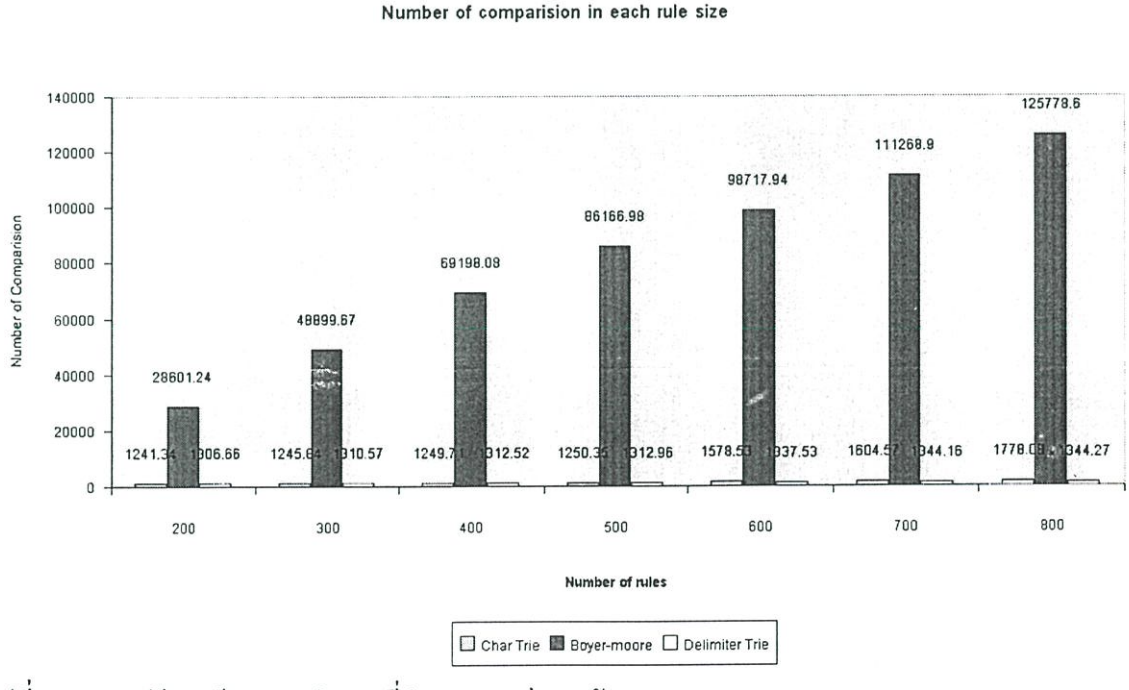
จากรูปเราสามารถสรุปได้ว่า การทำงานของกระบวนการตรวจสอบกฎโดยใช้อัลกอริทึมบอยเออร์-มัวร์นั้น มีการเปรียบเทียบตัวอักษรมากที่สุดในทุกๆ ชนิดของข้อมูล ส่วนการตรวจสอบโดยใช้โครงสร้างกฎแบบ Trie ของตัวอักษร และ Trie ของตัวแบ่งจะมีการเปรียบเทียบตัวอักษรในสัดส่วนที่ใกล้เคียงกันคือประมาณ 1-1.2 เท่าของขนาดแพ็กเก็ต และข้อมูลที่เหมาะสม

ที่สุดสำหรับการใช้งาน การค้นหาโดยใช้ตัวแบ่งระหว่างคำคือข้อมูลรูปภาพและไบนารีเนื่องจากจะใช้การเปรียบเทียบตัวอักษรเพียง 1.03 เท่าของขนาดแพ็กเก็ตเท่านั้น เมื่อเทียบกับข้อมูลที่เป็นตัวอักษรคือ 1.25 เท่าของขนาดแพ็กเก็ต

6.7 การทดสอบเพื่อวิเคราะห์ผลกระทบจากจำนวนกฎ

หลังจากที่ได้ทดสอบเพื่อหาว่าข้อมูลแต่ละแบบ มีผลต่อการทำงานของกระบวนการตรวจสอบกฎแต่ละแบบอย่างไรแล้วจึงทดลองเพื่อวิเคราะห์หาผลกระทบจากการเพิ่มจำนวนกฎ โดยทดลองโดยใช้ข้อมูลทั้งสามแบบคือข้อมูลตัวอักษร ข้อมูลไบนารี และข้อมูลรูปภาพมารวมกันแล้วทดสอบเปรียบเทียบจำนวนการเปรียบเทียบตัวอักษรในวิธีการแต่ละแบบ โดยจำนวนกฎที่ใช้มีตั้งแต่ 200 กฎ จนถึง 800 กฎ (เพิ่มขึ้นทีละ 100 กฎ)

ในการทดลองนี้จะมีแพ็กเก็ตรวมกันทั้งหมด 181 แพ็กเก็ต ข้อมูลในแพ็กเก็ตที่มีขนาดเล็กสุดคือ 346 ไบต์ ข้อมูลในแพ็กเก็ตที่มีขนาดใหญ่ที่สุดคือ 1500 ไบต์ และมีขนาดแพ็กเก็ตเฉลี่ยคือ 1195.59 ไบต์ เนื่องจากมีการทดสอบกับข้อมูลหลายๆ แพ็กเก็ต การนับจำนวนการเปรียบเทียบกฎจึงใช้วิธีการเฉลี่ยจำนวนการเปรียบเทียบตัวอักษรในการทำงานแต่ละครั้งแทน ซึ่งได้ผลลัพธ์ดังรูปที่ 6.8



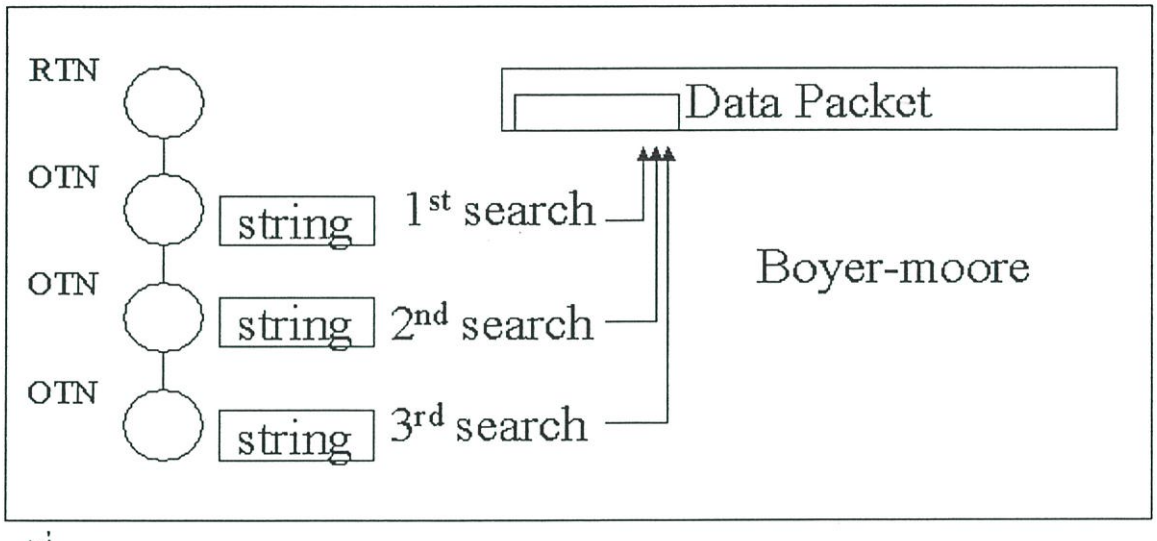
รูปที่ 6.8 การเปรียบเทียบการทำงานที่จำนวนกฎต่างๆ กัน

จากรูปจะเห็นได้ว่ากระบวนการเปรียบเทียบกฎโดยใช้อัลกอริทึมบอยเออร์-มัวร์นั้นจะเพิ่มจำนวนการเปรียบเทียบตัวอักษรตามการเปลี่ยนแปลงของจำนวนกฎ ส่วนกระบวนการตรวจสอบกฎโดยใช้โครงสร้าง Trie ของตัวอักษร และกระบวนการตรวจสอบกฎโดยใช้ Trie ของตัวแบ่งนั้นไม่มีผลกระทบจากการเพิ่มขึ้นของกฎมากนัก ซึ่งผลการทดลองที่ได้นี้จะมีการวิเคราะห์ผลในบทที่ 7 ต่อไป

บทที่ 7
บทวิเคราะห์

ในบทนี้เป็นการวิเคราะห์อัลกอริทึมและโครงสร้างกฎทั้งสามลักษณะ เพื่อพิสูจน์ให้เห็นว่ากระบวนการต่างๆ ส่งผลต่อการทำงานอย่างไรบ้าง ทำให้การทำงานมีการปรับปรุงอย่างไร และเพื่อสนับสนุนผลการทดลองในบทที่ 6

7.1 วิเคราะห์กระบวนการทำงานของ SNORT



รูปที่ 7.1 การตรวจสอบกฎของ SNORT

จากรูปที่ 7.1 การทำงานของโปรแกรม SNORT เมื่อโปรแกรมได้รับแพ็กเก็ตข้อมูลมาแล้ว จะนำเอาข้อมูลไปตรวจในชุดข้อมูลของกฎทีละกฎจนครบทุกกฎ ในการตรวจสอบกฎแต่ละกฎ คือ การค้นหาคำในกฎว่ามีอยู่ในข้อมูลนั้นหรือไม่ ซึ่งในการค้นหาจะใช้อัลกอริทึมบอยเออร์-มัวร์ ดังนั้นจากโครงสร้างกฎของโปรแกรม SNORT และกระบวนการในการตรวจสอบกฎของโปรแกรม SNORT จะได้กระบวนการทำงานดังนี้คือ

```
Loop (Rules[i:0..P])
{
   boyer-moore (data[0..M], rule[i][0..N])  // (O(M x N))
}
```

โดย

P คือจำนวนกฎทั้งหมด

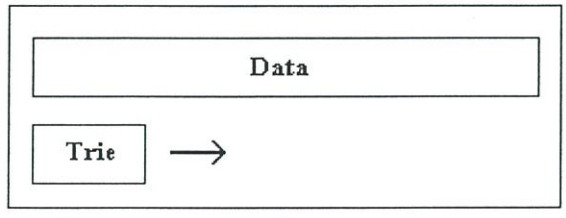
M คือความยาวของข้อมูล

N คือความยาวของกฎโดยเฉลี่ย

จากโค้ดที่เปรียบการทำงานของโปรแกรม SNORT มีการทำงานที่ซ้ำๆ กันอยู่ โดยมีการค้นหาคำโดยใช้อัลกอริทึมบอยเออร์-มัวร์ จะถูกเรียกให้ทำงานทุกๆ ครั้ง ซึ่งก็คือมีการอ่านข้อมูลในแพ็กเก็ตทุกๆ ครั้ง จากโค้ดดังกล่าวเราสามารถสรุปการทำงานของโปรแกรม SNORT ได้ โดยกรณีที่โปรแกรม SNORT ทำงานได้ดีที่สุดนั้น(Best Case) คือ O(P x M / N) ซึ่งเกิดขึ้นในกรณีที่การค้นหาคำในแพ็กเก็ตไม่มีคำนั้นๆ ปรากฏอยู่เลยหรือเป็นกรณีที่เกิดกรณีที่ดีที่สุดของอัลกอริทึมบอยเออร์-มัวร์ทุกๆครั้งที่มีการตรวจสอบกฎ และในกรณีที่แย่ที่สุด (Worst Case) จะได้ค่า complexity เป็น O(P x M x N) ซึ่งเกิดขึ้นในกรณีที่เกิดกรณีที่แย่ที่สุดของอัลกอริทึมบอยเออร์-มัวร์ในทุกๆ ครั้งของการค้นหาข้อมูลแต่ละกฎ

7.2 วิเคราะห์การทำงาน Trie ของตัวอักษร

จากการทำงานของโปรแกรม SNORT ซึ่งต้องอ่านข้อมูลแพ็กเก็ตเป็นจำนวนรอบเท่ากับจำนวนกฎ จึงทำให้จำนวนการเปรียบเทียบตัวอักษรในกฎ แปรผันตรงกับจำนวนกฎ และทำให้ความเร็วในการตรวจสอบกฎแปรผกผันกับจำนวนกฎ ถ้ากฎมีจำนวนมากขึ้น ความเร็วในการตรวจสอบกฎลดลง จึงเป็นปัญหาในกรณีที่มีกฎเพิ่มมากขึ้นเรื่อยๆ ภายหลังจากที่เราเปลี่ยนโครงสร้าง และกระบวนการตรวจสอบใหม่โดยเปลี่ยนแปลงโครงสร้างกฎให้อยู่ในรูปของ Burst Trie และเปลี่ยนการตรวจสอบกฎให้กลายเป็นนำ Trie ของกฎ ไปค้นหาภายในข้อมูลแพ็กเก็ตเพียงรอบเดียว ทำให้การตรวจสอบกฎทำได้เร็วขึ้น และไม่มีผลกระทบจากจำนวนกฎด้วย



รูปที่ 7.2 การตรวจสอบกฎของโครงสร้าง Trie

กระบวนการตรวจสอบกฎถูกเปลี่ยนจากการนำเอาข้อมูล ไปค้นหาในชุดของกฎ กลายเป็นการเปลี่ยนแปลงข้อมูลของกฎให้อยู่ในรูปของ Trie แล้วนำไปค้นหาในข้อมูล โดยโครงสร้าง Trie จะเปรียบเสมือนกับกฎทั้งหมด จากรูปที่ 7.2 ในการตรวจสอบกฎนั้นเราจะเลื่อน Trie จากจุดเริ่มต้น

ของข้อมูลไปเรื่อยๆ จนถึงท้ายข้อมูล ในการเลื่อนทุกๆ ครั้งจะเปรียบเทียบตัวอักษรในแต่ละชั้นของ Trie กับตัวอักษรตำแหน่งนั้นๆ ของข้อมูล ถ้าเหมือนกันจะเปรียบเทียบตัวอักษรตัวถัดไป กับตัวอักษรใน Trie ชั้นถัดไป แต่ถ้าเปรียบเทียบกันแล้วแตกต่างกัน จะเลื่อน Trie ไปเปรียบเทียบยังตัวอักษรตัวถัดไปของข้อมูล ในกรณีที่มีการเปรียบเทียบข้อมูลใน Trie จนถึงตัวอักษรตัวสุดท้าย แสดงว่าในข้อมูลมีคำในกฎอยู่ แต่ถ้าเลื่อน Trie ไปจนสุดข้อมูลแล้ว แสดงว่าที่ข้อมูลไม่มีคำใดๆ ที่อยู่ในกฎเลย

จากกระบวนการตรวจสอบกฎดังกล่าวเราสามารถเขียนโค้ดของการทำงาน เพื่อวิเคราะห์การทำงานได้คือ

```
Loop [data[i:0..M]]
{
    compare_string(data[i..i+N],trie[0..N]) // (O(N))
}
```

โดย

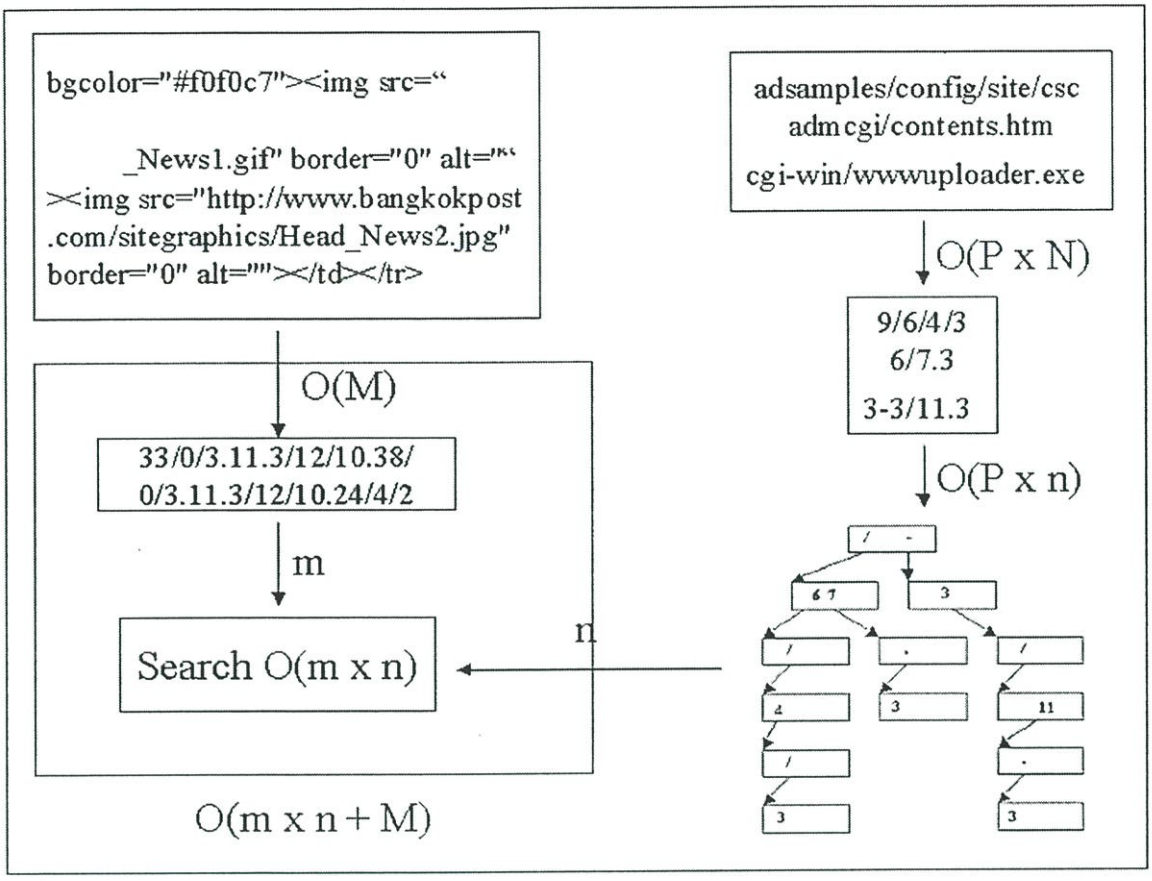
M คือความยาวของข้อมูล

N คือความยาวของกฎโดยเฉลี่ย

จากโค้ดการทำงานที่ได้เราสามารถสรุปได้ว่าการทำงานในกรณีที่ดีที่สุดคือ O(M) ซึ่งเป็นกรณีที่ตรวจสอบข้อมูลแล้วไม่พบตัวอักษรใดๆ ในแพ็กเก็ตที่เหมือนกับตัวอักษรตัวแรกใน Trie และการทำงานของโปรแกรมจะมี O(M x N) เนื่องจากในการตรวจสอบจะมีการอ่านข้อมูลในแพ็กเก็ตทีละตัวอักษร แล้วเปรียบเทียบกับข้อมูลใน Trie ถ้าตัวอักษรเหมือนกันแล้วก็จะเปรียบเทียบตัวอักษรถัดไป กับตัวอักษรใน Trie ในระดับที่ต่ำกว่า ถ้าไม่เหมือนกันจะเปลี่ยนเป็นเปรียบเทียบข้อมูลตัวอักษรตัวถัดไป กับตัวอักษรแรกใน Trie จากผลลัพธ์ดังกล่าวเราจะเห็นได้ว่าความเร็วในการทำงานของโปรแกรมจะไม่มีผลจากจำนวนกฎเลย ซึ่งแตกต่างกับการทำงานของโครงสร้างกฎแบบเดิมที่เป็น O(M x N x P)

7.3 วิเคราะห์การทำงาน Tries ของตัวแบ่งระหว่างคำ

หลังจากที่เราได้วิเคราะห์การทำงานของโครงสร้างกฎแบบ Trie ของตัวอักษรเราพบว่ามีการทำงานเป็น O(M x N) ซึ่งความเร็วในการทำงานจะไม่มีผลจากจำนวนกฎ แต่ก็ยังมีผลจากความยาวของข้อมูล และความยาวเฉลี่ยของกฎแต่ละกฎอยู่ ซึ่งถ้าเป็นการทำงานในลักษณะนี้ เมื่อมีการตรวจสอบกฎกับข้อมูลที่มีขนาดใหญ่มากๆ หรือถ้ากฎมีคำที่มีความยาวมากๆ ก็ทำให้การทำงานช้าลงได้เหมือนกัน จึงได้เปลี่ยนแปลงกระบวนการตรวจสอบใหม่ให้ผลกระทบจากขนาดของแพ็กเก็ต และความยาวของกฎไม่มีผลโดยตรง



รูปที่ 7.3 การทำงานของการค้นหาโดยใช้ตัวแบ่งระหว่างคำ

จากปัญหาดังกล่าวจึงเปลี่ยนแปลงกระบวนการตรวจสอบใหม่โดยการเปลี่ยนแปลงรูปแบบของข้อมูลและกฎ ให้อยู่ในรูปแบบอื่นๆ ที่มีขนาดเล็กลง โดยเปลี่ยนรูปแบบของข้อมูลให้อยู่ในรูปแบบของความยาวคำคั่นด้วยตัวแบ่ง และเปลี่ยนข้อมูลในกฎให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งเช่นกัน แล้วนำไปสร้างโครงสร้างกฎแบบ Trie ในกระบวนการตรวจสอบจะตรวจสอบข้อมูลในรูปของความยาวคำคั่นด้วยตัวแบ่ง กับ Trie ของกฎในรูปของความยาวคำคั่นด้วยตัวแบ่งก่อน แล้วถ้ามีรูปแบบที่เหมือนกันอยู่ จึงเปรียบเทียบข้อมูลกับคำในกฎนั้นๆ อีกรอบ ดังแสดงในรูปที่ 7.3 จากกระบวนการทำงานของการค้นหาโดยใช้ตัวแบ่งระหว่างคำ เราสามารถเขียนเป็นโค้ดของการทำงานได้เป็น

```
Convert data to delimiter form (data,Compressed_data); // O(M)
Loop [Compressed_data[j:0..m]] // O(m)
{
  if compare_string(Compressed_data[j..j+n],delimiter_trie[0..n])// (O(n))
  { exactly_matching(data,keyword) } // O(N)
}
```

โดย

M คือความยาวของข้อมูล

N คือความยาวของกฎโดยเฉลี่ย

m คือความยาวของข้อมูลที่ถูกเปลี่ยนให้อยู่ในรูปความยาวคำคั่นด้วยตัวแบ่งระหว่างคำ

n คือความลึกของ Delimiter Trie

จากโค้ดของการทำงานที่ได้ เราสามารถสรุปการทำงานโดยรวมได้เมื่อไม่มีตัวแบ่งตัวใดเลยที่อยู่ในแพ็กเก็ต ทำให้เมื่อผ่านกระบวนการเปลี่ยนรูปแล้วข้อมูลที่ได้มีความยาวเป็น 1 ทันที จึงไม่จำเป็นต้องผ่านกระบวนการตรวจสอบอีกครั้งเพราะแสดงว่าเป็นแพ็กเก็ตปกติแน่นอน จึงได้การทำงานที่เป็น O(M) ซึ่งเป็นการทำงานที่มีความเร็วสูงสุด ส่วนในกรณีที่มีตัวแบ่งระหว่างคำอยู่ในข้อมูลแพ็กเก็ตอยู่บ้าง ซึ่งในการทำงานจริงเป็นกรณีที่เกิดขึ้นเป็นส่วนใหญ่ถ้าเป็นข้อมูลแบบตัวอักษรทำให้ต้องมีการตรวจสอบการทำงานอีกครั้งหนึ่ง จะได้การทำงานที่เป็น O(m x n + M) เนื่องจากการทำงานต้องเปลี่ยนรูปแบบแพ็กเก็ตให้อยู่ในรูปความยาวคำคั่นด้วยตัวแบ่งก่อนรอบหนึ่ง ซึ่งต้องอ่านแพ็กเก็ตทั้งหมด 1 รอบ (M) แล้วหลังจากนั้นจึงค้นหาโดยเปรียบเทียบตัวอักษรต่างๆ ใน Trie ของตัวแบ่ง กับแพ็กเก็ตที่เปลี่ยนรูปแล้ว (m x n) และในกรณีที่ค้นหารูปแบบแล้วตรวจพบว่ามีรูปแบบที่เหมือนกันอยู่ใน Trie ของตัวแบ่งก็ต้องเปรียบเทียบตัวอักษรกับกฎจริงๆ อีกครั้งหนึ่ง ซึ่งต้องมีการอ่านข้อมูลในกฎอีกรอบ (N) จึงเป็นกรณีที่มีการทำงานช้าที่สุดคือ O(m x n + M + N)

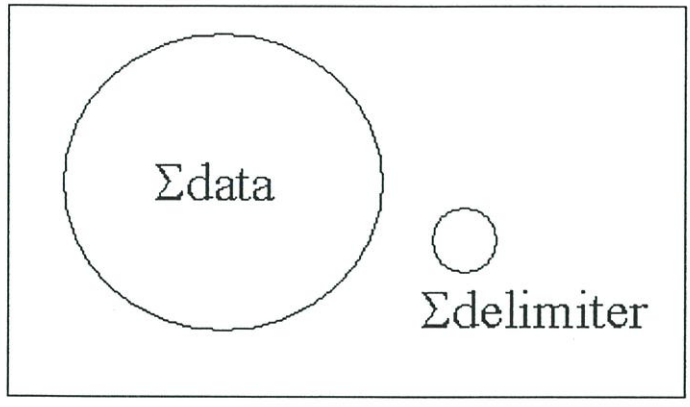
7.4 สมมุติฐานของตัวแปรต่างๆ

ในการทำงานของการค้นหาโดยใช้ตัวแบ่งระหว่างคำมีองค์ประกอบที่มีส่วนในการตัดสินความช้าเร็วของการทำงานอยู่ 3 ส่วนคือเซตของตัวอักษรในข้อมูล , เซตของตัวอักษรในกฎ และเซตของตัวแบ่ง ซึ่งเมื่อเราพิจารณาความสัมพันธ์ขององค์ประกอบดังกล่าวจะทำให้เราทราบถึงข้อดีข้อเสียของการค้นหาโดยใช้ตัวแบ่งระหว่างคำมากขึ้น

7.4.1 สมมุติฐานของความสัมพันธ์ระหว่างเซตของตัวอักษรในข้อมูลกับเซตตัวของตัวแบ่ง

สมมุติฐานของความสัมพันธ์ระหว่างเซตของตัวอักษรในข้อมูลกับเซตของตัวแบ่งเป็นการพิจารณาถึงผลที่เกิดขึ้นเมื่อความสัมพันธ์ระหว่างเซตของตัวแบ่งกับเซตของตัวอักษรในข้อมูลอยู่ในลักษณะต่างๆ กัน โดยจุดประสงค์ก็เพื่อวิเคราะห์ว่าผลลัพธ์ที่ได้จากการเปลี่ยนรูปจะมีขนาดของข้อมูลเท่าใด มีขนาดเล็กลงหรือใหญ่กว่าข้อมูลแพ็กเก็ต ซึ่งแบ่งออกเป็น 3 แบบคือ

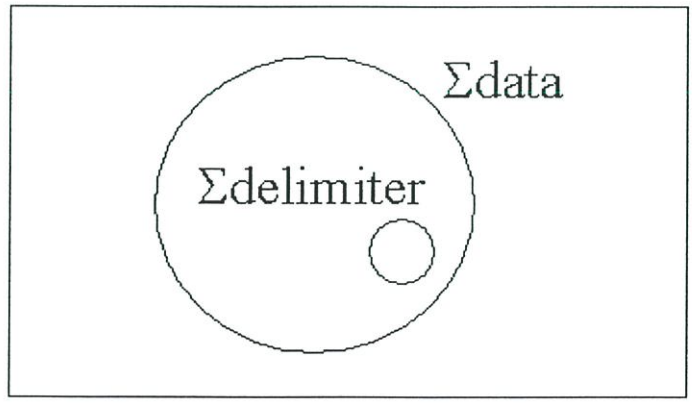
1. เซตของตัวแบ่ง ไม่เป็นซับเซตของ เซตตัวอักษรในข้อมูล



รูปที่ 7.4 เซตของตัวแบ่งไม่เป็นซับเซตของเซตตัวอักษรในข้อมูล

จากรูปที่ 7.4 ในกรณีที่เซตของตัวแบ่งระหว่างคำ ไม่เป็นซับเซตของตัวอักษรในข้อมูลเลย ทำให้การทำงานของการค้นหาโดยใช้ตัวแบ่งระหว่างคำ ทำได้เร็วที่สุดเนื่องจากเมื่อผ่านกระบวนการเปลี่ยนรูปข้อมูลให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งแล้ว จะได้ข้อมูลที่มีเฉพาะตัวเลขเท่านั้น ซึ่งความยาวคือ 1 ตัวอักษร(ความยาวข้อมูลทั้งหมด) หลังจากนั้นเมื่อเข้ากระบวนการตรวจสอบใน Trie จึงไม่จำเป็นต้องตรวจสอบกฎอะไรเลย

2. ตัวอักษรในเซตของตัวแบ่งทุกๆ ตัว อยู่ในเซตของตัวอักษรในข้อมูล และเซตของตัวแบ่งมีขนาดเล็ก

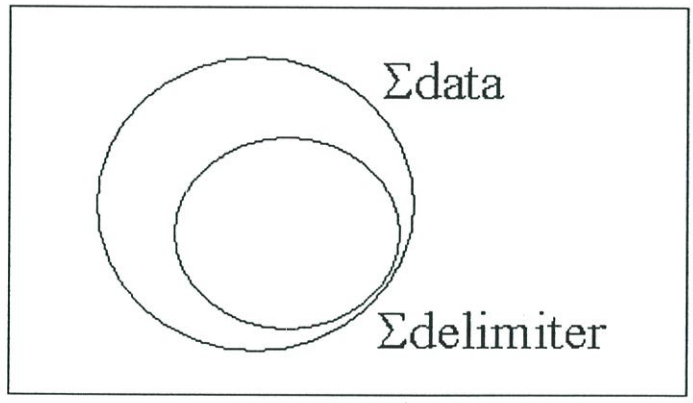


รูปที่ 7.5 เซตของตัวแบ่งขนาดเล็กอยู่ในเซตของตัวอักษรในข้อมูล

จากรูปที่ 7.5 ในกรณีที่เซตของตัวแบ่งทุกๆ ตัวอยู่ในเซตของตัวอักษรในข้อมูล และเซตของตัวแบ่งมีขนาดเล็กนั้น จะเกิดกรณีที่ข้อมูลถูกเปลี่ยนให้อยู่ในรูปความยาวคำคั่นด้วยตัวแบ่งระหว่างคำ ซึ่งเมื่อผ่านกระบวนการเปลี่ยนรูปแบบแล้วข้อมูลที่ได้จะมีขนาดเล็กลงหรือใหญ่ขึ้นเท่า

ใดนั้น ก็ขึ้นอยู่กับจำนวนตัวแบ่งที่พบในข้อมูล ถ้าในข้อมูลมีตัวแบ่งระหว่างคำน้อยๆ ข้อมูลที่ได้จากการเปลี่ยนรูปแบบแล้วก็จะเล็กลง แต่ถ้าพบจำนวนตัวอักษรปริมาณมากก็อาจทำให้ผลลัพธ์ที่ได้จากการเปลี่ยนรูปมีขนาดใหญ่ขึ้นกว่าข้อมูลเดิมก็เป็นได้ ในกรณีนี้เซตของตัวแบ่งมีขนาดเล็ก ทำให้โอกาสที่จะพบตัวแบ่งมีน้อยลงด้วย ผลลัพธ์ที่ได้จากการเปลี่ยนรูปจึงมีโอกาสที่จะมีขนาดเล็กกว่าข้อมูลก่อนเปลี่ยนรูปได้มาก

3. ตัวอักษรในเซตของตัวแบ่งทุกๆ ตัว อยู่ในเซตของตัวอักษรในข้อมูล และเซตของตัวแบ่งมีขนาดใหญ่



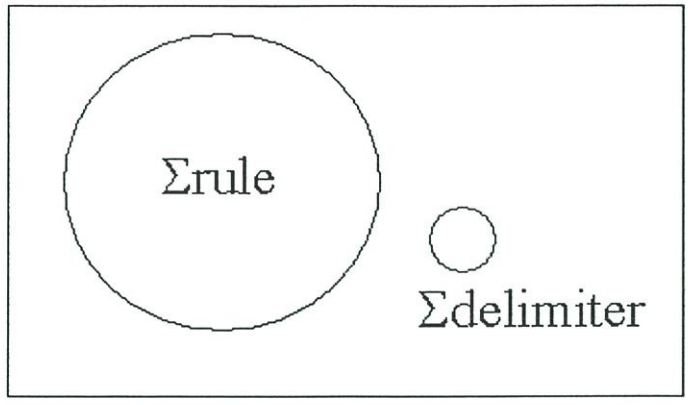
รูปที่ 7.6 เซตของตัวแบ่งขนาดใหญ่อยู่ในเซตของตัวอักษรในข้อมูล

จากรูปที่ 7.6 ในกรณีที่ตัวอักษรในเซตของตัวแบ่งทุกๆ ตัว อยู่ในเซตของตัวอักษรในข้อมูล และเซตของตัวแบ่งมีขนาดใหญ่ เป็นกรณีที่ทำให้เกิดการทำงานที่ช้ากว่าการทำงานในโครงสร้างกฎแบบ Trie ของตัวอักษร เมื่อเซตของตัวแบ่งมีขนาดใหญ่ ส่งผลให้ความน่าจะเป็นที่จะเจอตัวแบ่งในกระบวนการเปลี่ยนรูปแบบมีมากขึ้น ทำให้ข้อมูลที่ได้หลังเปลี่ยนรูปแบบแล้วมีขนาดใหญ่มาก ผลสืบเนื่องที่เกิดขึ้นคือกระบวนการตรวจสอบข้อมูลกับ Trie มีจำนวนการเปรียบเทียบมากขึ้น จนการทำงานช้ากว่ากระบวนการค้นหาข้อมูลโดย Trie ของตัวอักษร

7.4.2 สมมุติฐานของความสัมพันธ์ระหว่างเซตของตัวอักษรในกฎกับเซตของตัวแบ่ง

สมมุติฐานของความสัมพันธ์ระหว่างเซตของตัวอักษรในกฎกับเซตของตัวแบ่ง จะพิจารณาถึงผลลัพธ์ที่เกิดขึ้นเมื่อลักษณะของเซตของตัวอักษรในกฎ กับเซตของตัวแบ่ง มีความสัมพันธ์กันในลักษณะต่างๆ กัน ซึ่งผลลัพธ์ที่จะพิจารณา คือความลึกของ Trie ที่เกิดขึ้นภายหลังจากที่เปลี่ยนรูปของกฎและเปลี่ยนโครงสร้างให้อยู่ในรูป Trie เรียบร้อยแล้วซึ่งมีลักษณะความสัมพันธ์อยู่ 3 รูปแบบคือ

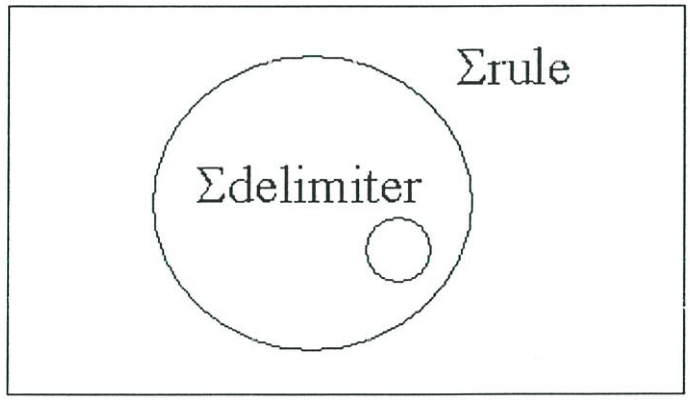
1. เซตของตัวแบ่ง ไม่เป็นซับเซตของเซตของตัวอักษรในกฎ



รูปที่ 7.7 เซตของตัวแบ่งไม่เป็นซับเซตของเซตตัวอักษรในกฎ

จากรูปที่ 7.7 เมื่อเซตของตัวแบ่งกับเซตของตัวอักษรในกฎอยู่แยกจากกัน ไม่มีส่วนไหนที่ซ้ำกันเลย เป็นกรณีที่ไม่ควรเกิดขึ้น เนื่องจากเมื่อเกิดกรณีนี้ขึ้นแล้วจะไม่สามารถสร้าง Trie ของกฎได้ เมื่อไม่มี Trie ของกฎก็ไม่สามารถนำไปตรวจสอบกฎความผิดปกติใดๆ ได้เลย

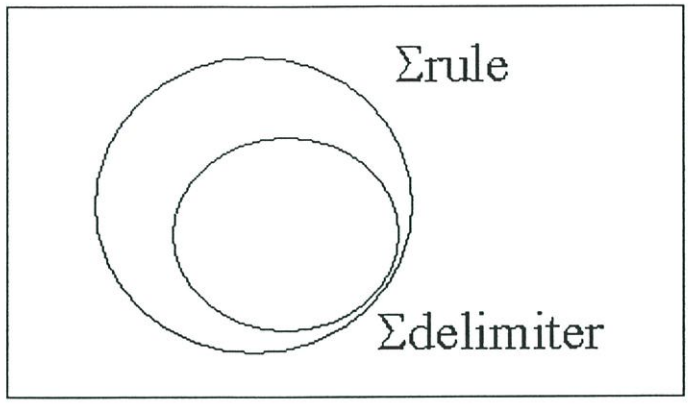
2. ตัวอักษรในเซตของตัวแบ่งทุกๆ ตัว อยู่ในเซตของตัวอักษรในกฎ และเซตของตัวแบ่งมีขนาดเล็ก



รูปที่ 7.8 เซตของตัวแบ่งขนาดเล็กอยู่ในเซตของตัวอักษรในกฎ

จากรูปที่ 7.8 เมื่อเซตของตัวแบ่งอยู่ในเซตของตัวอักษรในกฎ และเซตของตัวแบ่งมีขนาดเล็ก ทำให้มีโอกาสพบตัวแบ่งในกฎไม่มากนัก ในกรณีนี้สามารถสร้าง Trie ของกฎได้และจะได้ Trie ที่มีความลึกไม่มากนัก ส่งผลให้กระบวนการตรวจสอบกฎทำได้เร็วขึ้นด้วย

3. ตัวอักษรในเซตของตัวแบ่งทุกๆ ตัว อยู่ในเซตของตัวอักษรในกฎ และเซตของตัวแบ่งมีขนาดใหญ่



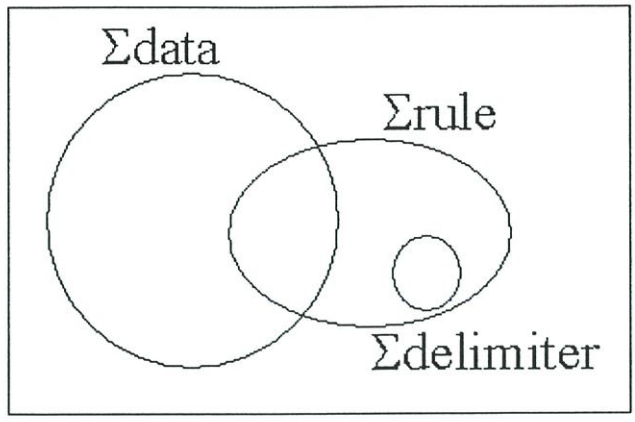
รูปที่ 7.9 เซตของตัวแบ่งขนาดใหญ่อยู่ในเซตของตัวอักษรในกฎ

จากรูปที่ 7.9 เมื่อเซตของตัวแบ่งทุกๆ ตัวอยู่ในเซตของกฎ และเซตของตัวแบ่งมีขนาดใหญ่ ทำให้โอกาสพบตัวแบ่งในกฎมีมาก ผลที่ตามมาคือจะได้ Trie ของกฎที่มีความลึกมาก และเมื่อนำไปใช้ในกระบวนการตรวจสอบกฎจะเสียเวลามากขึ้น

7.4.3 สมมุติฐานของความสัมพันธ์ระหว่างเซตของตัวอักษรในข้อมูล , เซตของตัวอักษรในกฎ และเซตของตัวแบ่ง

สมมุติฐานของความสัมพันธ์ระหว่างเซตของตัวอักษรในข้อมูล , เซตของตัวอักษรในกฎ และเซตของตัวแบ่งเป็นการพิจารณาถึงความสัมพันธ์โดยรวมของส่วนประกอบทั้งสามส่วน เพื่อวิเคราะห์ถึงผลลัพธ์ที่อาจจะเกิดขึ้นว่าความเร็วในการตรวจสอบกฎจะเพิ่มขึ้นหรือลดลงในกรณีที่องค์ประกอบทั้งสามมีความสัมพันธ์กันอยู่ในลักษณะใดบ้าง โดยพิจารณาใน 3 ลักษณะคือ

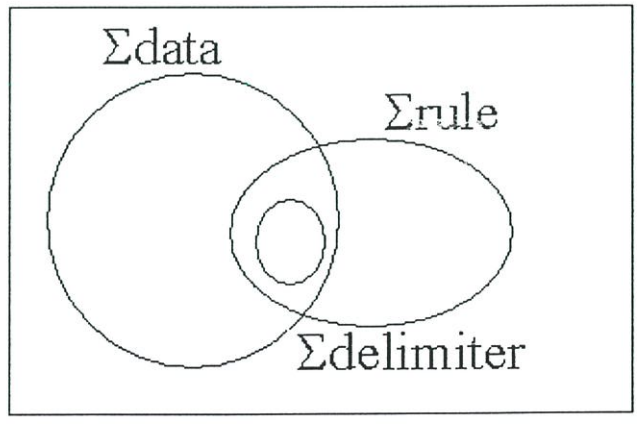
1. เซตของตัวอักษรในกฎมีบางส่วนอยู่ในเซตของตัวอักษรในข้อมูล และมีเซตของตัวแบ่งอยู่ในเซตของกฎเท่านั้น



รูปที่ 7.10 เซตของตัวอักษรในกฎมีบางส่วนอยู่ในเซตของตัวอักษรในข้อมูล และเซตของตัวแบ่งขนาดเล็กอยู่ในเซตของกฎเท่านั้น

จากรูปที่ 7.10 เมื่อเซตของตัวอักษรในกฎมีบางส่วนอยู่ในเซตของตัวอักษรในข้อมูล และมีเซตของตัวแบ่งอยู่ในเซตของกฎเท่านั้น ทำให้ผลลัพธ์ที่ได้จากการเปลี่ยนรูปข้อมูลในแพ็กเก็ตให้อยู่ในรูปความยาวคำ คั่นด้วยตัวแบ่งมีขนาดเป็น 1 คือความยาวของข้อมูลเท่านั้น และไม่จำเป็นต้องตรวจสอบข้อมูลหลังการเปลี่ยนรูปอีกครั้งหนึ่ง และลักษณะของ Trie ของกฎที่ได้จะมีความลึกไม่มากนัก ทำให้การตรวจสอบทำได้อย่างรวดเร็วมากที่สุดเพราะการตรวจสอบทั้งหมดมีขั้นตอนเพียงอ่านข้อมูลในแพ็กเก็ตเพียงครั้งเดียวเท่านั้น

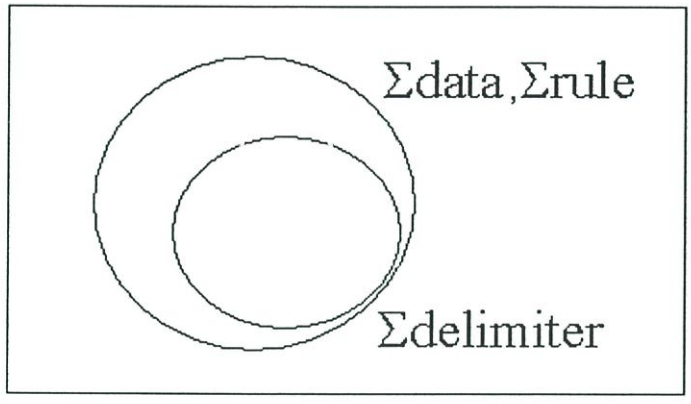
2. เซตของตัวอักษรในกฎมีบางส่วนอยู่ในเซตของตัวอักษรในข้อมูล และมีเซตของตัวแบ่งอยู่ในเซตของกฎและเซตของตัวอักษรในข้อมูล



รูปที่ 7.11 เซตของตัวอักษรในกฎมีบางส่วนอยู่ในเซตของตัวอักษรในข้อมูล และเซตของตัวแบ่งขนาดเล็กอยู่ในเซตทั้งสอง

จากรูปที่ 7.11 ในกรณีที่เซตของตัวอักษรในกฎมีบางส่วนอยู่ในเซตของตัวอักษรในข้อมูล และมีเซตของตัวแบ่งอยู่ในเซตของกฎและเซตของตัวอักษรในข้อมูล ผลลัพธ์ที่ได้คือเมื่อเปลี่ยนรูปข้อมูลในแพ็กเก็ตให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งแล้ว จะมีขนาดเล็กลงจากเดิม แต่ก็ขึ้นอยู่กับชุดของข้อมูลในแพ็กเก็ตด้วยว่าจะพบ หรือไม่พบตัวแบ่งมากน้อยเพียงใด ส่วน Trie ของกฎจะมีความลึกมากน้อยเพียงใดนั้นก็ขึ้นอยู่กับว่าพบตัวแบ่งในกฎมากเท่าใดด้วย

3. เซตของตัวอักษรในกฎและเซตของตัวอักษรในข้อมูลเป็นเซตเดียวกัน และมีเซตของตัวแบ่งขนาดใหญ่ อยู่ภายใน



รูปที่ 7.12 เซตของตัวอักษรในข้อมูลและกฎเป็นเซตเดียวกัน และมีเซตของตัวแบ่งขนาดใหญ่อยู่ภายใน

จากรูปที่ 7.12 ในกรณีนี้จะเป็นกรณีที่ให้ผลลัพธ์ที่แย่ที่สุดคือเนื่องจาก เซตของตัวแบ่งมีขนาดใหญ่และมีอยู่ในเซตของตัวอักษรในข้อมูล ซึ่งเมื่อเปลี่ยนรูปข้อมูลมาอยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่งแล้วจะได้ข้อมูลที่มีขนาดที่เล็กลงไม่มากนัก หรืออาจจะมีขนาดใหญ่ขึ้น และเนื่องจากมีเซตของตัวแบ่งขนาดใหญ่อยู่ในเซตของตัวอักษรในกฎ ทำให้เมื่อเปลี่ยนรูปและสร้างเป็น Trie แล้วจะได้ Trie ที่มีขนาดใหญ่มาก ทั้งสองปัจจัยมีส่วนอย่างมากที่ทำให้กระบวนการทำงานต่างๆ ทำได้ช้าลง

7.4.4 สมมุติฐานของลักษณะกฎที่ส่งผลต่อเซตของตัวแบ่ง

ในส่วนนี้เราจะมาพิจารณาลักษณะของชุดของกฎสองลักษณะ ในประเด็นเกี่ยวกับการมีอักขระร่วมมากหรือน้อย แล้วมีผลอย่างไรต่อชุดของตัวแบ่ง ซึ่งจะส่งผลกับการทำงานโดยรวมอย่างไรบ้าง

1. กรณีที่กฎทุกๆ กฎมีตัวอักขระร่วม

เมื่อกฎทุกๆ กฎมีตัวอักขระที่เป็นตัวอักขระร่วม ทำให้เราสามารถเลือกตัวอักขระร่วมให้อยู่ในเซตของตัวแบ่งได้เลย ถ้าลักษณะกฎเป็นเช่นนี้ทำให้เซตของตัวแบ่งมีขนาดเล็ก และได้ Trie ของกฎที่มีขนาดเล็ก แล้วยังส่งผลให้กระบวนการเปลี่ยนแปลงรูปแบบของข้อมูลในแพ็กเก็ตให้มีขนาดเล็กลงได้มากด้วย เมื่อใช้การค้นหาโดยใช้ตัวแบ่งระหว่างคำ กับชุดของกฎที่มีลักษณะนี้ จะทำให้การตรวจจับทำได้รวดเร็วขึ้นมาก

2. กรณีที่มีกฎบางกฎเท่านั้นที่มีอักขระร่วม

กระบวนการทำงานของการค้นหาโดยใช้ตัวแบ่งระหว่างคำจำเป็นต้องสร้าง Trie ของกฎขึ้นมาโดยจะต้องใช้ข้อมูลจากการเปลี่ยนรูปกฎแต่ละข้อให้อยู่ในรูปของความยาวคำคั่นด้วยตัวแบ่ง แล้วนำมาใส่ในโครงสร้าง Trie จึงจำเป็นที่ในกฎแต่ละกฎจะต้องมีตัวอักษรที่อยู่ในเซตของตัวแบ่ง ถ้ามีกฎบางกฎเท่านั้นที่มีอักขระร่วม ทำให้จำเป็นต้องดึงเอาตัวอักขระจากกฎอื่นๆ มาเพื่อให้ได้เซตของตัวแบ่งที่มีตัวอักษรอยู่ทุกๆ กฎ จึงทำให้เซตของตัวแบ่งมีขนาดใหญ่ ส่งผลให้ Trie และข้อมูลที่ผ่านกระบวนการเปลี่ยนรูปมีขนาดใหญ่และการทำงานช้าลงกว่าปกติได้

7.5 วิเคราะห์ข้อดีข้อเสียของการค้นหาโดยใช้ตัวแบ่งระหว่างคำ

การค้นหาโดยใช้ตัวแบ่งระหว่างคำ เป็นกระบวนการค้นหาข้อมูลที่เหมาะกับการค้นหาชุดของกฎ ในข้อมูลขนาดใหญ่ โดยกฎแต่ละกฎจะต้องมีตัวอักขระร่วมกันอยู่อย่างน้อย 1 ตัวอักษร ด้วยการค้นหาในลักษณะนี้เมื่อนำไปใช้กับงานโดยทั่วๆ ไปควรเป็นการค้นหาที่มีกฎจำนวนมากๆ และเป็นการค้นหาที่ไม่พบข้อมูลในกฎเป็นส่วนใหญ่ เนื่องจากโดยกระบวนการทำงานแล้ว เมื่อค้นหาแล้วไม่พบรูปแบบของกฎ กระบวนการค้นหาจะทำงานได้อย่างรวดเร็วมาก จึงเหมาะสมกับข้อมูลที่เป็นข้อมูลเว็บต่างๆ เนื่องจากข้อมูลจะมีทั้งข้อมูลตัวอักษร ข้อมูลรูปภาพ โดยลักษณะเซตของตัวแบ่งที่ดีนั้นมีคุณสมบัติสามข้อคือ

1. มีตัวอักขระภายในเซตของตัวแบ่งอยู่ในทุกๆ กฎ
2. ในแต่ละกฎมีอักขระในเซตของตัวแบ่งจำนวนน้อย
3. พบอักขระในเซตของตัวแบ่งในข้อมูลจำนวนน้อย

7.6 สรุปการวิเคราะห์

การค้นหาโดยใช้ตัวแบ่งระหว่างคำ ซึ่งใช้โครงสร้างกฎแบบ Trie นั้นสามารถทำงานได้ดีในกรณีที่มีลักษณะขององค์ประกอบต่างๆ ดังนี้คือ

1. มีกฎจำนวนมาก ซึ่งชุดของกฎมีตัวอักขระร่วมกันอยู่ในทุกๆ กฎ และแต่ละกฎมีอักขระในเซตของตัวแบ่งจำนวนน้อย
2. ข้อมูลในแพ็กเก็ตมีขนาดใหญ่ และมีอักขระในเซตของตัวแบ่งอยู่ในแพ็กเก็ตเป็นจำนวนน้อย หรือไม่มีเลย
3. เซตของตัวแบ่งมีขนาดเล็ก

บรรณานุกรม

ธนัญชัย ตรีภาค. 2545 "คิดให้หนักเมื่อต้องพบกับยามไซเบอร์." **ไมโครคอมพิวเตอร์.** ฉบับที่ 206, กันยายน. หน้า 181-186.

Bace, R. G. 2000. **Intrusion detection.** Macmillan Technical Publishing.

Jacobson, V. et. al. 2002. "The Libpcap project" [Online]. Available : http://www.tcpdump.org.

Boyer, R. S. and Moore, J. S. 1977. "A fast string searching algorithm." Communications of the ACM, 20(10):762--772,

Cannady, J. 1998. "Artificial Neural Networks for Misuse Detection." In Proceedings of the 1998.

Forrest, S. et. al. 1996. "A sense of self for unix processes." In Proceedings of the 1996 IEEE Symposium on Security and Privacy, pages 120--128, Los Alamitos, CA, 1996. IEEE Computer Society Press.

Heberlein, L.T. 2001. "Network Security Monitor (NSM) – Final Report." Lawrence Livermore National Laboratory project deliverable, [Online]. Available : http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf

Heberlein, L. et. al. 1990. "A Network Security Monitor." In Proceedings of the IEEE Symposium on Research in Security and Privacy, pages 296--304, May. URL http://seclab.cs.ucdavis.edu/papers/pdfs/th-gd-90.pdf.

Heinz, S. 2001. "Burst tries: a fast, efficient data structure for string keys." In submission 2001

Heinz, S. and Zobel, J. 2002. "Performance of data structure for small sets of strings." Twenty-Fifth Australasian Computer Science Conference (ACSC2002).

Hofmeyr, S. A. 1999. "An immunological model of distributed detection and its application to computer security." PhD thesis, University of New Mexico, Albuquerque, New Mexico, 1999.

Jason, C. C. 2001. "Towards Faster String Matcching for Intrusion Detection" DISCEX-II 2001 [Online]. Available : http://www.silicondefense.com/software/acbm/speed_of_snort_03_16_2001.pdf.

Lee, W. et. al. 2000. "A data mining and CIDF based approach for detecting novel and distributed intrusions." In Proceedings of the 3rd International Workshop on Recent Advances in Intrusion Detection (RAID 2000), October 2000.

Lee, W. and Stolfo, S. 1998. "Data mining approaches for intrusion detection." In Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, 1998.

Paxson, V. 1998. "Bro: A System for Detecting Network Intruders in Real-Time." Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998.

Ptacek, T. and Newsham, T. 1998. "Insertion, Evasion, and Denial of Service : Eluding the Network Intrusion Detection." Secure Networks, Inc., Http://www.aciri.org/vern/Ptacek-NewshamEvasion-98.ps , Jan. 1998.

Roesch, M. 1999. "Snort – Lightweight Intrusion Detection for Networks" USENIX LISA Conference November 1999.

Ryan, J. et. al. 1998. "Intrusion Detection with Neural Networks." In Advances in Neural Information Processing Systems 10, May 1998.

ภาคผนวก ก

ISSN 0125-1724

วิศวสาร ลาดกระบัง

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

LADKRABANG ENGINEERING JOURNAL

ปีที่ 19 ฉบับที่ 4 ธันวาคม 2545

Delimiter Search for Locating Rule Pattern in Intrusion Detection Systems

การค้นหาโดยใช้ตัวแบ่งระหว่างคำเพื่อเพิ่มความเร็วในการตรวจสอบกฎสำหรับโปรแกรมตรวจจับผู้บุกรุก

บุญธีร์ เครือคราชู , ธนัญชัย ดรีภาค และ ชนา หงษ์สุวรรณ
ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

บทคัดย่อ

งานวิจัยชิ้นนี้มีจุดประสงค์เพื่อปรับปรุงประสิทธิภาพในการทำงานของระบบตรวจจับผู้บุกรุกทางเครือข่าย หรือ Network Intrusion Detection System (NIDS) เพื่อให้ระบบดังกล่าวสามารถวิเคราะห์ข้อมูลได้เร็วขึ้น โดยการใช้วิธีการค้นหาโดยใช้ตัวแบ่งระหว่างคำหรือ delimiter search และเปลี่ยนแปลงโครงสร้างของกฎ ให้อยู่ในรูปโครงสร้างของ burst tries ในการทดสอบสมมุติฐานนั้นใช้วิธีการนับจำนวนการเปรียบเทียบตัวอักษรในกฎ ซึ่งเปรียบเทียบผลลัพธ์ของโครงสร้างกฎ 3 โครงสร้างคือ โครงสร้างกฎดั้งเดิมของโปรแกรม SNORT ซึ่งเป็น NIDS ที่ใช้กันอย่างแพร่หลาย , โครงสร้างกฎแบบ burst tries และโครงสร้างกฎใหม่ที่รวมเอา burst tries กับ delimiter search เข้าด้วยกัน จากการทดลองพบว่า โครงสร้างกฎที่รวมเอา delimiter search กับ burst tries เข้าด้วยกันจะมีการเปรียบเทียบตัวอักษรในการตรวจสอบกฎน้อยที่สุดคือประมาณ 4.49 เปอร์เซ็นต์ของโครงสร้างกฎแบบดั้งเดิมเท่านั้น

ABSTRACT

The goal of this research is to improve the run time performance of Network Intrusion Detection Systems (NIDS) by using delimiter search and change its rule's data structure to burst tries. In this paper we compare the number of character comparison on three cases, original SNORT , burst tries structure and delimiter search with burst tries structure. The result is delimiter search with burst tries structure has the least character comparison, about 4.49 percent of the number of character comparison in original SNORT .

1. บทนำ

ระบบตรวจจับผู้บุกรุกทางเครือข่ายหรือ Network Intrusion Detection System (NIDS) เป็นระบบที่ทำงานเพื่อตรวจจับรูปแบบของการบุกรุกโดยการดักจับข้อมูลในเครือข่ายแล้ววิเคราะห์ข้อมูล NIDS ในยุคแรกนั้นคือ UC Davis Network Security Monitor [1,2] ซึ่งวิเคราะห์ข้อมูลโดยใช้วิธีการเปรียบเทียบข้อมูลในเครือข่ายกับรูปแบบของการบุกรุก สำหรับในปัจจุบันก็มีการสร้าง NIDS ขึ้นมาอีกหลายๆ โปรแกรมเช่น SNORT [3] และ Bro[4] ซึ่งเทคนิคที่ใช้งานก็ยังคงเป็นการเปรียบเทียบข้อมูลในเครือข่ายกับข้อมูลในกฎเช่นเดิม

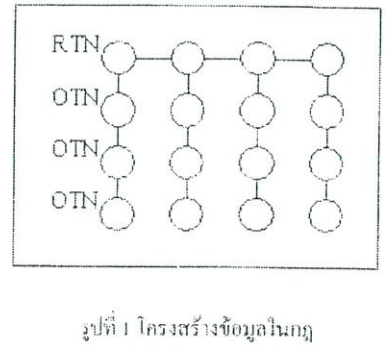
การสร้าง NIDS นั้นต้องคำนึงถึงการใช้งานเป็นหลัก เนื่องจาก NIDS ที่ต้องทำงานแบบ near-realtime ได้อย่าง SNORT ต้องเป็นระบบที่สามารถตรวจจับการบุกรุกได้เกือบทันทีที่มีการบุกรุกเกิดขึ้น ถ้าสร้าง NIDS แล้วไม่สามารถตรวจจับสัญญาณของการบุกรุกได้ทัน ก็จะเป็นระบบที่ไม่สามารถทำงานได้จริง สำหรับปัญหาที่เกิดขึ้นในการใช้งาน NIDS คือในบางครั้งระบบ NIDS จะถูกรบกวนโดยใช้เทคนิค Insertion , Evasion และ Denial of Service [5] ทำให้ระบบ NIDS ไม่สามารถตรวจจับการบุกรุกระบบที่แท้จริงได้

เพื่อให้การรบกวนระบบ NIDS มีผลต่อการทำงานน้อยที่สุด และทำให้การทำงานของ NIDS เป็น near-realtime มากที่สุด เราจึงพยายามปรับปรุงระบบ NIDS ให้ทำงานได้เร็วขึ้น โดยเทคนิคที่ใช้ในการปรับปรุงก็คือ การเปลี่ยนแปลงโครงสร้างข้อมูลของกฎ และเพิ่มกระบวนการเปลี่ยนรูปแบบกฎและข้อมูลในแพ็กเก็ตก่อน เพื่อลดขนาดของข้อมูลที่จะเปรียบเทียบ โดยจะเปลี่ยนรูปของกฎและข้อมูลในแพ็กเก็ตให้เหลือเพียงตัวแบ่งและระยะห่างระหว่างตัวแบ่งภายในกฎเท่านั้น แล้วจึงวิเคราะห์หารูปแบบของการบุกรุก การทำเช่นนี้ทำให้ข้อมูลที่จะต้องเปรียบเทียบลดลง การทำงานโดยรวมของระบบจึงเร็วขึ้น สำหรับในงานวิจัยนี้เราได้ทดลองเพื่อพิสูจน์สมมุติฐานโดยเปลี่ยนแปลงโครงสร้างข้อมูล และวิธีการเปรียบเทียบกฎของโปรแกรม SNORT ซึ่งเป็นโปรแกรมตรวจจับผู้บุกรุกทางเครือข่ายที่ใช้งานกันอย่างแพร่หลาย แล้วทดลองเปรียบเทียบผลกับข้อมูลเครือข่ายหลายๆ รูปแบบ

สำหรับรายละเอียดของงานวิจัยนั้น ในส่วนที่ 2 จะมีรายละเอียดโครงสร้างข้อมูลของโปรแกรม SNORT และวิธีการเปรียบเทียบกฎ ส่วนที่ 3 เป็นข้อมูลเกี่ยวกับกฎที่ใช้ในโปรแกรม SNORT ส่วนที่ 4 เป็นการเปลี่ยนแปลงรูปแบบของกฎ ส่วนที่ 5 และ 6 เป็นผลการทดลองและบทสรุปของงานวิจัย

2. SNORT

SNORT เป็นโปรแกรมที่ใช้สำหรับตรวจจับผู้บุกรุกทางเครือข่าย การทำงานของโปรแกรมนี้คือการดักจับข้อมูลในเครือข่าย และเปรียบเทียบข้อมูลนั้นกับรูปแบบของการบุกรุก โดยโครงสร้างข้อมูลสำหรับตรวจสอบรูปแบบการบุกรุกจะอยู่ในรูปของลิงค์ลิสท์ 2 มิติ (Two Dimensional Linked List) (ดังรูปที่ 1) ในกระบวนการตรวจสอบกฎนั้น ถ้ามีข้อมูลภายในแพ็กเก็ตที่เหมือนกับข้อมูลในกฎโปรแกรมนี้จะแจ้งเตือนให้ผู้ดูแลระบบทราบทันที กระบวนการทำงานเช่นนี้ทำให้ผู้ดูแลระบบทราบได้ว่ามีการบุกรุกและสามารถป้องกันความเสียหายได้ทันเวลา



รูปที่ 1 โครงสร้างข้อมูลในกฎ

สำหรับโครงสร้างที่เป็นลิงค์ลิสท์ 2 มิตินั้น ประกอบด้วยโหนด 2 รูปแบบด้วยกันคือ Rule Tree Node

(RTN) และ Option Tree Node (OTN) โดย RTN เป็นโหนดที่บรรจุส่วนของกฎที่รูปแบบการบุกรุกหลายๆ แบบจะต้องมี เช่น ไอพีแอดเดรสต้นทาง , ไอพีแอดเดรสปลายทาง , พอร์ตต้นทาง , พอร์ตปลายทางและโพรโทคอลที่ใช้ในแพ็กเก็ตเป็นต้น (ทีซีพี ยูดีพี หรือ ไอซีเอ็มพี เป็นต้น) ส่วน OTN นั้นประกอบด้วยส่วนเฉพาะของกฎ ที่เป็นส่วนที่แตกต่างกันในการโจมตีแต่ละแบบเช่น ไฮด์ของไอซีเอ็มพีแพ็กเก็ต ทีซีพีแฟล็ก และ ข้อมูลในแพ็กเก็ตที่ใช้บุกรุกเป็นต้น

สำหรับจุดประสงค์ในการสร้างโครงสร้างกฎในลักษณะนี้คือหลีกเลี่ยงการตรวจสอบกฎทุกๆกฎ โดยพยายามแบ่งกลุ่มของกฎออกเป็นกลุ่มๆ แทน ถ้าตรวจสอบข้อมูลในแพ็กเก็ตกับ RTN แล้วไม่เหมือนก็ไม่จำเป็นต้องตรวจสอบกฎกับ OTN ที่ติดกับ RTN นั้นอีก สำหรับ SNORT ในกรณีที่ตรวจสอบกับเงื่อนไขใน RTN แล้วไม่เหมือนก็จะไปตรวจสอบกับข้อมูลใน RTN ตัวอื่นๆ ต่อไปจนหมด

ในกรณีที่ ตรวจสอบกับข้อมูลใน RTN แต่ละส่วนแล้วตรงกับเงื่อนไข จะต้องมีการตรวจสอบกับเงื่อนไขใน OTN อีกครั้งหนึ่ง ในส่วนของการเปรียบเทียบ content ของกฎนั้น โปรแกรม SNORT ใช้อัลกอริทึมในการตรวจสอบกฎโดยใช้ อัลกอริทึมของ Boyer-Moore เข้าตรวจสอบใน OTN แล้วไม่พบความผิดปกติก็จะตรวจสอบแพ็กเก็ตตัวต่อไป OTN ตัวไปจนหมดทุกการของ OTN ตรวจสอบการเข้าล่วงเป็นระบบการที่เสียเวลามาก ถึงแม้ว่า Boyer-Moore จะช่วยให้การค้นหาและตรวจสอบทำได้เร็วขึ้นแต่ก็ยังมีการตรวจสอบกฎที่ต้องทำทีละกฎไปเรื่อยๆ จนครบทุกกฎ และทุกครั้งของการตรวจสอบจะต้องทำเช่นเดิมกับแพ็กเก็ตต่อไป

3. กฎของ SNORT

กฎของ SNORT มีโครงสร้างที่ยืดหยุ่นมากทำให้ผู้ใช้งานสามารถสร้างกฎของโปรแกรม SNORT รูปแบบของกฎประกอบด้วย ตัวแปรต่างๆ ในการตรวจสอบของ SNORT ต่อแพ็กเก็ตผิดปกติ , ส่วนเงื่อนไขของเฮดเดอร์ของข้อมูล และส่วนเงื่อนไขของ content ตัวอย่างเช่น

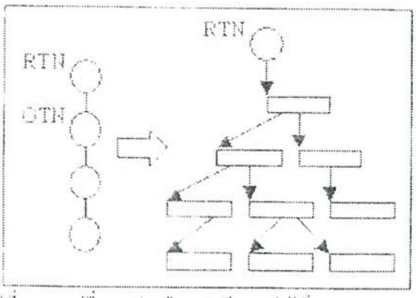
Alert tcp any any -> any 80 (msg:"IIS-cmd?";flags:PA;content:"cmd?&"; nocase;)

จากกฎตัวอย่าง หมายถึงโปรแกรมจะแจ้งเตือนผู้ดูแลระบบถ้าแพ็กเก็ตที่รับเข้ามาเป็นแพ็กเก็ตที่ทำงานกับโพรโทคอลทีซีพี โดยส่งจากไอพีและพอร์ตใดๆ ไปยังไอพีปลายทางใดๆ ที่พอร์ต 80 โดยในทีซีพีเฮดเดอร์จะมีแฟล็ก PSH และ ACK และในแพ็กเก็ตมีคำว่า "cmd?&" อยู่ในแพ็กเก็ต ในการตรวจสอบข้อมูลในแพ็กเก็ตจะไม่คำนึงถึงนัยสำคัญของตัวอักษรตัวพิมพ์เล็ก และตัวพิมพ์ใหญ่

4. การเปลี่ยนแปลงรูปแบบของกฎ

จากที่ได้อธิบายถึงรูปแบบการทำงานและโครงสร้างของ SNORT ในหัวข้อที่ผ่านมา งานวิจัยนี้จึงนำเสนอวิธีการปรับปรุงการทำงานเพื่อให้การทำงานเร็วขึ้น 2 แนวทางคือ แนวทางแรกจะต้องเปลี่ยนแปลงโครงสร้างข้อมูลในการตรวจสอบการบุกรุกให้เป็นแบบ burst trie[6] ดังรูปที่ 2 การปรับปรุงโครงสร้างให้เป็นแบบนี้จะสามารถลดความซ้ำซ้อนในการตรวจสอบรูปแบบของการบุกรุกได้ โดยตรวจสอบรูปแบบของคำในพิกัดชั้นมาเป็นแบบที่จะตรวจสอบรูปแบบของการบุกรุกในแพ็กเก็ตกับกฎทีละกฎ ซึ่งจะช้ามาก

การปรับเปลี่ยนโครงสร้างในลักษณะนี้เป็นเพียงส่วนเสริมเพื่อเพิ่มอย่างมาก ในการเพิ่มความเร็วในการตรวจสอบกฎ เพราะการตรวจสอบแพ็กเก็ตแต่ละ 1 ครั้งจะสามารถตรวจสอบกฎทั้งหมดได้ ซึ่งถ้าเป็นการตรวจสอบปกติของ SNORT จะต้องเปรียบเทียบแพ็กเก็ตกับกฎที่ละกฎซึ่งต้องทำเท่ากับจำนวนกฎ ถึงแม้ว่าจะมีการใช้ Boyer-Moore มาช่วยแล้วก็ยังช้ากว่าการทำงานด้วยโครงสร้างดังกล่าว



รูปที่ 2 การเปลี่ยนแปลงโครงสร้างกฎให้เป็น Burst tries

เนื่องจากแพ็กเก็ตที่โปรแกรมรับเข้ามาส่วนใหญ่ไม่ใช่แพ็กเก็ตของการบุกรุก การลดจำนวนการเปรียบเทียบตัวอักษรจึงสามารถเพิ่มความเร็วได้เช่นกัน แนวทางที่สองคือการเปลี่ยนแปลงรูปแบบของกฎและข้อมูลในแพ็กเก็ต จากการสังเกตข้อมูล content ของกฎหลายๆ กฎ พบว่าใน content จะประกอบด้วยคำหลายๆ คำ คั่นด้วยตัวแบ่งอยู่หลายๆ รูปแบบด้วยกัน (ดังตัวอย่างรูปที่ 3) เราจึงนำคุณสมบัติดังกล่าวมาเปลี่ยนแปลงรูปแบบของกฎและข้อมูลในแพ็กเก็ต ให้อยู่ในรูปแบบของตัวแบ่งดังรูปที่ 4 และนำมารวมกับโครงสร้างแบบ burst tries ดังรูปที่ 5

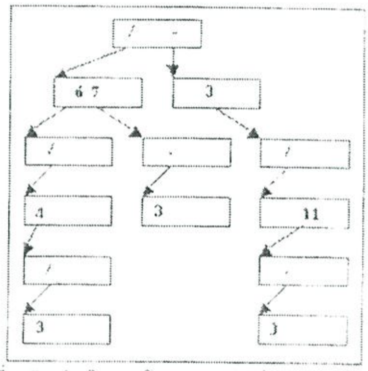
cgi-win/wwwuploader.exe
_vti_bin/_vti_aut/author.exe
admcgi/contents.htm
_vti_pvt/services.cnf
iissamples/sdk/asp/docs/codebrws.asp
scripts/iisadmin/ism.dll?http/serv
adsamples/config/site.csc

รูปที่ 3 ตัวอย่างข้อมูล content ในกฎที่จะต้องตรวจสอบ

เมื่อความยาวของกฎและข้อมูลในแพ็กเก็ตถูกย่อลง เวลาที่ใช้ในการเปรียบเทียบกฎก็จะลดลงไปด้วย แต่การทำเช่นนี้ก็มีข้อเสียอยู่คือจะต้องแปลงข้อมูลในแพ็กเก็ตอย่างน้อย 1 รอบเสมอ ก่อนการทำงานใดๆ เพื่อสร้างแพ็กเก็ตในรูปแบบของตัวแบ่งและความยาวคำระหว่างตัวแบ่ง แต่ก็ทำงานเพียง 1 ครั้งต่อการตรวจสอบกฎใน RTN ทั้งหมด ซึ่งถ้าใช้ตัวแบ่งชุดเดียวกันหมดทุก RTN ก็จะทำงานตรงส่วนนี้เพียงครั้งเดียวเท่านั้น

adsamples/config/site.csc	เปลี่ยนเป็น	9.6/4.3
admcgi/contents.htm	เปลี่ยนเป็น	6/7.3
cgi-win/wwwuploader.exe	เปลี่ยนเป็น	3-3/11.3

รูปที่ 4 ตัวอย่างการเปลี่ยนรูปโดยใช้ตัวแบ่งคือ - / และ .



รูปที่ 5 ตัวอย่างโครงสร้างของกฎโดยใช้ตัวแบ่งร่วมกับ burst tries

5. ผลการทดสอบ

ในการทดสอบสมมุติฐานจะเปรียบเทียบระบบ 3 ระบบคือ SNORT ปกติ version 1.6.3 ซึ่งใช้ Boyer-Moore ในการเปรียบเทียบรูปแบบการบุกรุก ระบบที่สองคือโปรแกรม SNORT ที่เปลี่ยนแปลงโครงสร้างเป็นแบบ burst tries สำหรับระบบสุดท้ายคือระบบที่เปลี่ยนแปลงจากระบบที่สองให้มีการเปลี่ยนแปลงรูปแบบของกฎและรูปแบบของข้อมูลในแพ็กเก็ตเป็นแบบตัวแบ่งและความยาวคำก่อนจัดเข้าเป็นโครงสร้างแบบ burst tries ในการทดสอบข้อสมมุติฐานนั้น งานวิจัยนี้ใช้วัดประสิทธิภาพจากจำนวนการเปรียบเทียบตัวอักษรในการทำงานทั้งสามแบบ โดยการนับจำนวนการเปรียบเทียบจะนับทุกๆ ครั้งที่มีการเปรียบเทียบตัวอักษรขึ้นในระบบ

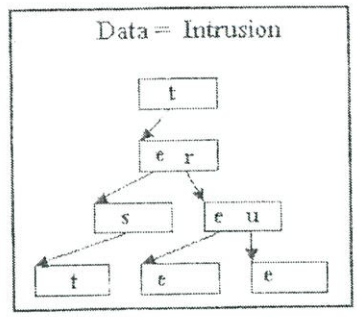
ใช้ตารางนับการเปรียบเทียบตัวอักษรในการทดลองนี้จะใช้วิธีนับทุกๆ การเปรียบเทียบตัวอักษรจนในรูปที่ 6 จะเปรียบเทียบข้อมูลคำว่า intrusion ในการเปรียบเทียบนี้จะมีการเปรียบเทียบตัวอักษรทีละตัว จะนำคำว่า intrusion เปรียบเทียบในโครงสร้างกฎก่อน ซึ่งเมื่อเปรียบเทียบตัว i แล้วไม่ตรงกับในโครงสร้างกฎ จำนับคำว่า intrusion มาเปรียบเทียบในกฎต่อไป แต่ตัวอักษร n ก็ยังไม่ได้อยู่ในโครงสร้างกฎเช่นเดียวกัน ในการเปรียบเทียบคำว่า intrusion นี้ และ intrusion จำนับการเปรียบเทียบตัวอักษรเป็นหนึ่งครั้ง

แต่เมื่อนำคำว่า trusion มาเปรียบเทียบในโครงสร้างกฎจะพบว่าชุดของตัวอักษร tru มีกฎในโครงสร้างกฎ ซึ่งจะต้องมีการเปรียบเทียบตัวอักษร 3 ครั้ง แต่ก็ยังไม่ match ทั้งหมดเนื่องจากไม่มีตัวอักษร s ในโหนดสุดท้าย จำนวนการเปรียบเทียบตัวอักษรสำหรับคำว่า trusion นี้จะเป็น 4 ครั้ง

หลังจากนั้นจึงต้องเปรียบเทียบคำว่า rusion , usion , sion , ion , on และ n ต่อไปตามลำดับ ซึ่งก็ไม่ match กับตัวอักษร i ในโหนดแรกของ tries เช่นกัน มีการเปรียบเทียบคำละ 1 ครั้ง รวมจำนวนการเปรียบเทียบทั้งหมดของคำว่า intrusion นี้ 12 ครั้ง

สำหรับการใช้งาน delimiter search นั้นจะใช้ลักษณะเดียวกัน โดยการใช้งานจากตัวอักษรแต่ละตัวของคำที่จะนำมาเปรียบเทียบ และความยาวของคำที่จะ search ของ SNORT จะนับจำนวนครั้งของการเปรียบเทียบตัวอักษรก่อนกลับ search ในการค้นหาของ Boyer-Moore

ผลการเปรียบเทียบจำนวนครั้งของการเปรียบเทียบตัวอักษรที่ได้จากการทดลองในรูปที่ 7 จะเห็นได้ว่าจำนวนการเปรียบเทียบตัวอักษรในโครงสร้างของ burst tries นั้นจะมีจำนวนการเปรียบเทียบน้อยกว่า SNORT เป็นอย่างมาก ที่จะมีจำนวนการเปรียบเทียบน้อยกว่าเฉลี่ยเพียงแค่ 4.96 เปอร์เซ็นต์เท่านั้น ซึ่งในกรณีของ delimiter burst tries จะมีจำนวนการเปรียบเทียบตัวอักษรน้อยกว่า burst tries เล็กน้อย ประมาณ 9.29 เปอร์เซ็นต์ หรือประมาณ 4.49 เปอร์เซ็นต์เมื่อเทียบกับ SNORT เดิม

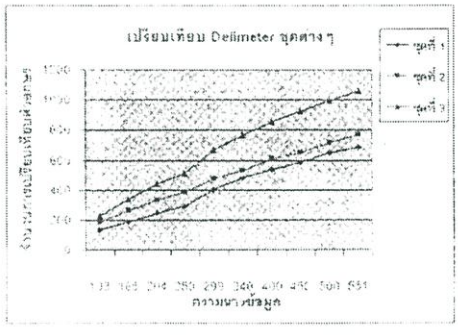


รูปที่ 6 ตัวอย่างการนับการเปรียบเทียบใน tries

ความยาวของข้อมูลในแพ็กเก็ต	จำนวนการเปรียบเทียบใน SNORT	จำนวนการเปรียบเทียบใน burst tries	จำนวนการเปรียบเทียบใน Delimiter tries
103	2998	151	149
165	4714	236	220
204	5994	293	269
250	7267	358	322
302	8972	441	401
348	10413	498	437
400	11855	577	515
450	13200	660	584
500	14732	738	689
551	16118	841	786

รูปที่ 7 ตารางเปรียบเทียบจำนวนการเปรียบเทียบตัวอักษรในโครงสร้าง 3 โครงสร้างคือ โครงสร้างของ SNORT ปกติ , burst tries และ delimiter burst tries

ผลลัพธ์ที่ได้นั้นเป็นการทดลองเปรียบเทียบการทำงานที่ delimiter ชุดหนึ่ง แต่เมื่อเปรียบเทียบการทำงานเมื่อเปลี่ยนชุดของ delimiter จะทำให้ผลลัพธ์เปลี่ยนไป โดยอาจเลือก delimiter ที่ไม่เหมาะสม การทำงานอาจช้าลงได้ ดังรูปที่ 8 เป็นกราฟเปรียบเทียบการทำงานของโปรแกรมเมื่อเลือก delimiter ชุดต่างๆ ซึ่งจะเห็นได้ว่าจำนวนการเปรียบเทียบตัวอักษรจะมีความแตกต่างกันมาก ดังนั้นวิธีการใช้ delimiter search นี้จะมีปัจจัยที่เกี่ยวข้องต่อการทำงานอยู่ก็คือ ขนาดของข้อมูล และความสัมพันธ์ระหว่าง delimiter กับข้อมูลในกฎ และข้อมูลในแพ็กเก็ตด้วย



รูปที่ 8 delimiter ชุดต่างๆ กัน ให้ผลการทำงานต่างกัน

6. สรุป

งานวิจัยฉบับนี้อธิบายถึงการปรับปรุงเทคนิคในการวิเคราะห์ข้อมูลของระบบตรวจจับผู้บุกรุกทางเครือข่าย ทำให้ระบบตรวจสอบความผิดปกติได้เร็วขึ้น งานวิจัยนี้ได้ลดความซ้ำซ้อนในการตรวจสอบกฎลงโดยการเปลี่ยนโครงสร้างกฎจาก อิงก์อิสร์ สองมิติ ให้กลายเป็นรูปแบบ burst tries และลดจำนวนข้อมูลในการค้นหาลง โดยการเปรียบรูปแบบของแพ็กเก็ตและกฎเพียงใหม่ การทดลองเพื่อเปรียบเทียบผลลัพธ์จะเห็นได้ว่าจากจำนวนกฎที่ถูกดังกล่าวสามารถลดจำนวนการเปรียบเทียบกฎให้เหลือเพียง 4.49 เปอร์เซ็นต์ของโครงสร้างกฎของโปรแกรม SNORT เดิมโดยผลลัพธ์ดังกล่าวจะขึ้นอยู่กับปัจจัยคือ จำนวนกฎที่ใช้ในการตรวจสอบ ขนาดของแพ็กเก็ตที่อยู่ในเครือข่าย และชุดของตัวแบ่งที่ใช้ด้วย จึงน่าจะมีการวิจัยต่อไปว่าเมื่อนำ delimiter search ไปใช้ในงานรูปแบบต่างๆ แล้ว ชุดของตัวแบ่งการมีลักษณะอย่างไร

7. เอกสารอ้างอิง

[1] L.T. Habergeon, G.V. Dias, K.N. Levitt, B. Mukherjee, "A Network Security Monitor". Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy. Oakland, CA, 7-9 May 1990, pp.296-304.

[2] L.T. Heberlein, "Network Security Monitor (NSM) – Final Report". Lawrence Livermore National Laboratory project deliverable,
http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf

[3] M. Roesch, "SNORT – Lightweight Intrusion Detection for Networks" USENIX LISA Conference November 1999

[4] V. Paxson, "Bro: A system for detecting network intruders in real-time," Computer Networks, vol. 31, no. 23-24, pp. 2435–2465, 1999.

[5] T. Ptacek and T. Newsham, "Insertion, Evasion, and Denial of Service : Eluding the Network Intrusion Detection," Secure Networks, Inc., http://www.aciri.org/vern/Ptacek-Newsham-Evasion-98.ps , Jan. 1998.

[6] S. Heinz, "Burst tries: a fast, efficient data structure for string keys." In submission 2001, http://citeseer.nj.nec.com/532638.html

ประวัติผู้เขียน

นายธนัญชัย ตรีภาค เกิดเมื่อวันที่ 31 มีนาคม 2522 ที่จังหวัดแพร่ สำเร็จการศึกษาวิศวกรรมศาสตรบัณฑิต จากสถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง ปีการศึกษา 2542 สาขาวิชาวิศวกรรมคอมพิวเตอร์ ผลงานทางวิชาการที่ได้รับการยอมรับคือ "ระบบตรวจจับผู้บุกรุกทางเครือข่ายคอมพิวเตอร์" การประชุมวิชาการประจำปี 2543 เทคโนโลยี ECTI กับเศรษฐกิจใหม่ โดยศูนย์เทคโนโลยีอิเล็กทรอนิกส์และคอมพิวเตอร์แห่งชาติ และ "การลดความรุนแรงของการโจมตีเพื่อปิดบริการโดยใช้เทคนิคแพ็กเก็ตฟลัดดิง" การประชุมวิชาการทางวิศวกรรมไฟฟ้าครั้งที่ 24 (24 th Electrical Engineering Conference (EECON-24))

ทุนการศึกษาที่เคยได้รับคือทุนอุดหนุนการศึกษา โครงการพัฒนาอาจารย์สาขาขาดแคลนเพื่อศึกษาในประเทศ สาขาวิศวกรรมคอมพิวเตอร์ ปีการศึกษา 2543