การทำแพทเทิร์นแมทชิ่งด้วยกรรมวิธีคัคคูแฮชชิ่ง
สำหรับระบบตรวจจับผู้บุกรุก

PATTERN MATCHING USING CUCKOO HASHING ALGORITHM FOR INTRUSION DETECTION SYSTEM

พรชัย กอประเสริฐทวร
PORNCHAI KORPRASERTTAWORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ. 2553

KMITL - 2010 - EN - M - 070 - 013

การทำแพทเทิร์นแมทชิ่งด้วยกรรมวิธีคุ๊กคูแฮทชิ่ง สำหรับระบบตรวจจับผู้บุกรุก

PATTERN MATCHING USING CUCKOO HASHING ALGORITHM FOR INTRUSION DETECTION SYSTEM

พรชัย กอประเสริฐถาวร

PORNCHAI KORPRASERTTAWORN

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2553

KMITL-2010-EN-M-070-013

หัวข้อวิทยานิพนธ์	การทำแพทเทิร์นแมทชิ่งด้วยกรรมวิธีคุ๊กคูแฮทชิ่งสำหรับระบบตรวจจับผู้บุกรุก
นักศึกษา	นายพรชัย กอประเสริฐถาวร
รหัสนักศึกษา	49060703
ปริญญา	วิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
พ.ศ.	2553
อาจารย์ที่ปรึกษาวิทยานิพนธ์	ผศ.ดร.สุรินทร์ กิตติธรกุล

บทคัดย่อ

ระบบตรวจจับผู้บุกรุกทางเครือข่าย (NIDS) เป็นระบบที่ใช้สำหรับการป้องกันการละเมิดความปลอดภัยทางเครือข่าย ด้วยเทคโนโลยีในปัจจุบันทำให้ความเร็วของเครือข่ายเพิ่มสูงขึ้น ปริมาณข้อมูลที่ส่งผ่านเครือข่ายมีมากขึ้น การทำงานของระบบตรวจจับผู้บุกรุกทางเครือข่ายจึงต้องมีประสิทธิภาพเพิ่มมากขึ้น ในการทำงานของระบบตรวจจับผู้บุกรุกทางเครือข่ายนั้นมีความจำเป็นที่จะต้องทำการตรวจสอบเนื้อหา (Pattern Matching) ของข้อมูล (Packet Payload) ที่ถูกส่งผ่านเครือข่ายเพื่อหารูปแบบ (Signature) ของการละเมิดความปลอดภัย ในการตรวจสอบดังกล่าวเป็นส่วนที่ระบบตรวจจับผู้บุกรุกทางเครือข่ายใช้ทรัพยากรในการประมวลผลมากที่สุด

งานวิจัยนี้ได้นำเสนอการเพิ่มประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุก โดยนำเทคนิคของอัลกอริธึม Cuckoo Hashing มาประยุกต์ใช้และพัฒนาบนโครงสร้างของอัลกอริธึม Piranha ซึ่งเป็นอัลกอริธึมที่ใช้ในระบบตรวจจับผู้บุกรุกและมีประสิทธิภาพการทำงานที่ดีทั้งในแง่ของเวลาที่ใช้ในการประมวลผลและปริมาณหน่วยความจำที่ใช้งาน

จากการทดลองอัลกอริธึมที่พัฒนาสามารถทำงานได้อย่างถูกต้องแม่นยำ และมีประสิทธิภาพที่ดีขึ้นเมื่อเปรียบเทียบกับอัลกอริธึมเดิม

Thesis Title	Pattern Matching Using Cuckoo Hashing Algorithm for Intrusion Detection System
Student	Mr. Pornchai Korpraserttaworn
Student ID.	49060703
Degree	Master of Engineering
Program	Computer Engineering
Year	2010
Thesis Advisor	Asst.Prof.Dr. Surin Kittitornkun

ABSTRACT

Network Intrusion Detection Systems (NIDSs) provide an important security function to detect and defend network attacks. As the network speeds and workloads increase, it is important for NIDSs to be highly efficient. Most NIDSs must check for thousands of known attack patterns in every incoming packet, making pattern matching the most expensive task of pattern/signature-based NIDSs in terms of processing time and memory resource.

This thesis proposes a new algorithm Piranha+Cuckoo Snort which combines filtering of Piranha and hashing of Cuckoo together. Particularly, the matching speed of Piranha+Cuckoo Snort is faster than that of Piranha-based Snort.

กิตติกรรมประกาศ

การจัดทำวิทยานิพนธ์ฉบับนี้จะสำเร็จลุล่วงไปเสียมิได้ หากไม่ได้รับความกรุณาเป็นอย่างสูงจากอาจารย์ผู้ควบคุมวิทยานิพนธ์ คือ ผศ.ดร.สุรินทร์ กิตติธรกุล ที่ช่วยให้คำปรึกษา ข้อชี้แนะ และความช่วยเหลือในหลายอย่างจนกระทั่งลุล่วงไปได้ด้วยดี ผู้วิจัยขอกราบขอบพระคุณเป็นอย่างสูงมา ณ ที่นี้

ขอขอบพระคุณอาจารย์ทุกท่าน ที่ได้ให้ความรู้ต่างๆ ตลอดจนนักวิจัยทุกท่านที่เอื้อเฟื้องานวิจัยด้วยดีเสมอมา

ขอบคุณและขอบใจเพื่อนๆ พี่ๆ และน้องๆ ที่ได้คอยให้กำลังใจ และให้ความช่วยเหลือในหลายๆ ด้านตลอดจนคำแนะนำแก่ข้าพเจ้า

ขอขอบคุณคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ให้การสนับสนุนในการทำวิจัย

ขอขอบพระคุณบิดา มารดา และครอบครัวของข้าพเจ้า ที่ได้ให้การสนับสนุนในเรื่องต่างๆ จนทำให้วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยดี

สุดท้ายนี้คุณความดีประโยชน์และคุณค่าที่เกิดจากวิทยานิพนธ์ฉบับนี้ ผู้วิจัยขอมอบแด่บิดา มารดา ครูอาจารย์ผู้ประสิทธิ์ประสาทความรู้ ตลอดจนผู้มีพระคุณทุกท่าน

พรชัย กอประเสริฐถาวร

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมา และความสำคัญของงานวิจัย

เนื่องจากทุกวันนี้เทคโนโลยีได้ก้าวหน้าไปมาก การสื่อสารข้อมูลจึงมีความเร็วมากขึ้น อุปกรณ์เครือข่ายได้รับการพัฒนาให้สามารถรองรับกับความต้องการในการส่งข้อมูลความเร็วสูง ในขณะที่เทคโนโลยีกำลังก้าวหน้าไปการละเมิดความปลอดภัยทางเครือข่ายจากผู้ไม่ประสงค์ดีก็เพิ่มขึ้นตามไปด้วย นำมาซึ่งความเสียหายต่อองค์กรหรือธุรกิจ การให้ความสำคัญกับกระบวนการในการปกป้องข้อมูลหรือระบบเครือข่ายในองค์กรต่างๆ มีมากขึ้น ระบบตรวจจับผู้บุกรุกทางเครือข่ายจึงเข้ามามีบทบาทสำคัญในปัจจุบัน มีการใช้งานอย่างแพร่หลายและมีการพัฒนาประสิทธิภาพอยู่ตลอดเวลาเพื่อให้รองรับกับการสื่อสารข้อมูลที่มีความเร็วสูง

ระบบตรวจจับผู้บุกรุกทางเครือข่ายทำงานโดยการพยายามตรวจจับกลไกที่นำไปสู่การละเมิดความปลอดภัยต่อการรักษาความลับของข้อมูล ความถูกต้องของข้อมูล และการใช้งานได้ของระบบ หรืออาจเป็นกลไกที่พยายามหลบหลีกระบบรักษาความปลอดภัยของเครือข่าย [5] การทำงานของระบบตรวจจับผู้บุกรุกทางเครือข่ายโดยทั่วไปมีพื้นฐานมาจาก Signature แต่ละ Signature จะบ่งบอกถึงลักษณะของการบุกรุกต่างๆ กัน ระบบตรวจจับผู้บุกรุกจะตรวจสอบ Packet จากเครือข่ายว่าตรงกับ Signature ใดหรือไม่ และหากตรวจพบว่าตรงกับ Signature ใดก็จะแจ้งเตือนถึงการบุกรุกนั้น

โปรแกรม Snort [1,12] เป็นโปรแกรมตรวจจับผู้บุกรุกทางเครือข่ายที่มีการใช้งานกันอย่างแพร่หลาย ในการทำงานของ Snort จำเป็นที่จะต้องตรวจสอบในส่วนของ Header และเนื้อหาของข้อมูลของแต่ละ Packet กับ Signature ซึ่งอาจมีความยาวมากถึงร้อยตัวอักษร การทำงานดังกล่าวจะใช้ทรัพยากรในการประมวลผลอย่างมาก

ในงานวิจัยนี้ได้พยายามพิจารณาวิธีการที่จะเพิ่มประสิทธิภาพของระบบตรวจจับผู้บุกรุกโดยมุ่งเน้นไปที่ Algorithm ของการทำ Pattern Matching ซึ่งเป็นส่วนที่ระบบตรวจจับผู้บุกรุกใช้ทรัพยากรในการประมวลผลมากที่สุด

1.2 วัตถุประสงค์ของงานวิจัย

1. ศึกษาเทคนิคและวิธีการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุกที่มีอยู่ในปัจจุบันเพื่อพิจารณาถึงข้อดีและข้อด้อยของแต่ละวิธีการ
2. ศึกษาเทคนิคและวิธีการอื่นๆ ที่จะนำมาช่วยพัฒนาประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุก
4. ศึกษาประสิทธิภาพและข้อจำกัดของวิธีการทำ Pattern Matching ที่นำเสนอพร้อมกับเปรียบเทียบกับวิธีอื่นๆ
5. เพื่อเป็นแหล่งข้อมูลอ้างอิงสำหรับผู้สนใจงานวิจัยด้านนี้ต่อไป

1.3 สมมุติฐานของงานวิจัย

เนื่องจากการทำ Pattern Matching ของอัลกอริธึม Piranha ซึ่งเป็นอัลกอริธึมที่ดีที่สุดในแง่ของหน่วยความจำที่ใช้และปริมาณการใช้งาน CPU มีโครงสร้างการทำงานที่มีข้อเสียในบางส่วน โดยเฉพาะในขั้นตอนที่ทำการเปรียบเทียบข้อมูลใน Packet Payload กับ Pattern ต่างๆ ขั้นตอนดังกล่าวนี้อัลกอริธึม Piranha ใช้โครงสร้างข้อมูลชนิด Link List ที่แก้ปัญญาการชนกันของข้อมูลด้วย Separate Chaining ทำให้ไม่สามารถคาดการณ์จำนวนครั้งของการ Search ได้

ดังนั้นในงานวิจัยนี้ ได้พัฒนาขั้นตอนการ Search ให้มีประสิทธิภาพที่ดีขึ้นโดยนำเทคนิคการ Hash แบบ Cuckoo Hashing [4] มาผนวกเข้ากับอัลกอริธึม Piranha ด้วยคุณสมบัติของ Cuckoo Hashing ที่รับประกันการ Search ไม่เกิน 2 ครั้ง จะช่วยลดปริมาณการ Search ลงและส่งผลต่อประสิทธิภาพโดยรวมของการทำ Pattern Matching

1.4 ทฤษฎีและหลักการที่ใช้ในงานวิจัย

การพัฒนาประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุกในงานวิจัยนี้จะอาศัยทฤษฎีและหลักการดังต่อไปนี้

1. Piranha
2. Cuckoo Hashing

1.5 ขอบเขตของงานวิจัย

วิทยานิพนธ์นี้ได้ทำการศึกษา และพัฒนาวิธีการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุก โดยกำหนดขอบเขตปัญหาให้เฉพาะเจาะจง ซึ่งจะมีขอบเขตการวิจัย ดังนี้

1. การพัฒนาจะอาศัย Frame Work ของอัลกอริธึม Piranha
2. ระบบตรวจจับผู้บุกรุกที่ใช้งานเป็นโปรแกรม Snort รุ่น 2.4.2

1.6 ขั้นตอนของการศึกษา

1. กำหนดวัตถุประสงค์ และขอบเขตของงานวิจัยตั้งสมมุติฐานการศึกษา ตลอดจนกำหนดแนวทางการศึกษาวิจัยเบื้องต้น
2. ศึกษาประสิทธิภาพของอัลกอริธึม Pattern Matching ต่างๆ ที่ใช้ในระบบตรวจจับผู้บุกรุก เพื่อเปรียบเทียบข้อดีและข้อด้อยของแต่ละอัลกอริธึม และเลือกอัลกอริธึมที่มีประสิทธิภาพดีมาเป็นพื้นฐานในการพัฒนา
3. นำทฤษฎีและหลักการที่เกี่ยวข้องมาพิสูจน์สมมุติฐานที่ได้กำหนดไว้ และกำหนดแนวทางในการทำงานวิจัยที่ครอบคลุมและชัดเจนยิ่งขึ้น
4. เตรียม Dataset ที่เป็น Packet ที่เก็บจากเครือข่ายเพื่อนำมาใช้ทดลอง
5. พัฒนาอัลกอริธึม Pattern Matching โดยใช้โครงสร้างของอัลกอริธึมที่เลือกมาพัฒนาโดยใช้เทคนิคที่ได้ทำการศึกษามา
6. ทำการทดลองกับ Dataset ที่ได้เตรียมไว้ โดยใช้อัลกอริธึม Pattern Matching ที่พัฒนาขึ้น และทำการเก็บข้อมูล และผลลัพธ์ของการทดลอง เพื่อการวิเคราะห์ และปรับปรุงต่อไป
7. ประเมินผลการทดลองโดยนำข้อมูลและผลลัพธ์ที่ได้จากการทดลองมา เปรียบเทียบกับงานวิจัยเดิมที่เคยมีผู้วิจัยไว้ก่อนหน้า ทั้งในด้านความถูกต้องและเวลาที่ใช้ในการประมวลผล สรุปและวิเคราะห์ผลการทดลอง

1.7 เครื่องมือและอุปกรณ์ที่ใช้ในงานวิจัย

เครื่องมือ และอุปกรณ์ที่ใช้ในงานวิจัยในครั้งนี้ ได้แก่

1. เครื่องคอมพิวเตอร์ CPU Pentium III 500 MHz หน่วยความจำ 512 MB จำนวน 1 เครื่อง
2. ระบบปฏิบัติการ Debian 3.1r5
3. โปรแกรม Snort 2.4.2
4. Rule set จำนวน 7,344 rule

1.8 โครงสร้างของวิทยานิพนธ์

วิทยานิพนธ์ฉบับนี้แบ่งเนื้อหาออกเป็น 6 บท แต่ละบทประกอบด้วยเนื้อหาดังนี้

บทที่ 1 กล่าวถึงความเป็นมา ความสำคัญ วัตถุประสงค์ สมมติฐานของงานวิจัย แนวคิดเบื้องต้นที่ใช้ในงานวิจัย ขอบเขตของงานวิจัย ขั้นตอนของการศึกษา รวมถึงเครื่องมือและอุปกรณ์ที่ใช้ในงานวิจัย

บทที่ 2 กล่าวถึงทฤษฎีและหลักการเบื้องต้นเกี่ยวกับการละเมิดความปลอดภัยทางเครือข่าย แนวความคิดเบื้องต้นของระบบตรวจจับผู้บุกรุก และงานวิจัยที่เกี่ยวข้อง

บทที่ 3 กล่าวถึงทฤษฎีพื้นฐานและงานวิจัยที่เกี่ยวข้อง ได้แก่ ระบบตรวจจับผู้บุกรุก Snort, ความหมายของ Pattern Matching, อัลกอริธึม Piranha และอัลกอริธึม Cuckoo Hashing

บทที่ 4 กล่าวถึงการเพิ่มประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุกด้วยวิธีการ Cuckoo Hashing โดยอธิบายรายละเอียดและแนวความคิดของการพัฒนา

บทที่ 5 กล่าวถึงวิธีการทดลองและผลการทดลองทั้งหมดด้วยวิธีการที่นำเสนอและวิธีการอื่นๆ เพื่อนำมาเปรียบเทียบผลลัพธ์ที่ได้ทั้งในเรื่องของความถูกต้องและเวลาที่ใช้ในการประมวลผล

บทที่ 6 กล่าวถึงภาพรวมของงานวิจัยที่ได้นำเสนอ วิเคราะห์ และสรุปผลการทดลอง พร้อมทั้งข้อเสนอแนะสำหรับแนวทางการพัฒนาต่อไปในอนาคต

ภาคผนวก แสดงรายละเอียดของผลการทดลองเพิ่มเติมในงานวิจัยนี้ และผลงานในระหว่างการศึกษาที่ได้รับการตีพิมพ์เผยแพร่

บทที่ 2
แนวคิดเบื้องต้นและงานวิจัยที่เกี่ยวข้อง

ปัจจุบันการพัฒนาประสิทธิภาพของการทำ Pattern Matching ได้เป็นไปอย่างต่อเนื่อง ส่งผลให้โปรแกรมระบบตรวจจับผู้บุกรุกทำงานได้อย่างมีประสิทธิภาพมากขึ้น รองรับกับการเพิ่มขึ้นของความเร็วในติดต่อสื่อสารผ่านเครือข่ายอินเตอร์เน็ต เนื้อหาโดยรวมในบทนี้จะกล่าวถึงแนวคิดเบื้องต้นของการละเมิดความปลอดภัยทางเครือข่าย แนวความคิดเบื้องต้นของระบบตรวจจับผู้บุกรุก และอัลกอริธึมการทำ Pattern Matching ประเภทต่างๆ ที่มีการนำมาประยุกต์ใช้กับระบบตรวจจับผู้บุกรุก ตลอดจนกล่าวถึงตัวอย่างงานวิจัยที่เป็นการพัฒนาประสิทธิภาพของอัลกอริธึม Pattern Matching ของระบบตรวจจับผู้บุกรุกและงานวิจัยที่สามารถนำเทคนิคมาประยุกต์ใช้ในการพัฒนา

2.1 แนวคิดเบื้องต้น

ในการใช้งานระบบตรวจจับผู้บุกรุกพบว่า กระบวนการทำ Pattern Matching เป็นกระบวนการที่ต้องใช้เวลาในการประมวลผล (Processing Time) มากถึง 30 เปอร์เซ็นต์ [1] และอาจมากถึง 80 เปอร์เซ็นต์ในกรณี Web-Intensive Traffic [2] หากนำระบบตรวจจับผู้บุกรุกไปใช้กับเครือข่ายที่มีความเร็วสูง การทำงานของระบบตรวจจับผู้บุกรุกจึงต้องมีประสิทธิภาพสูงขึ้นตามไปด้วย เพื่อให้รองรับกับความเร็วของเครือข่ายนั้น ประกอบกับการเพิ่มขึ้นของรูปแบบในการละเมิดความปลอดภัยทางเครือข่ายทำให้ Rule หรือ Signature ที่ใช้ในการตรวจหาการละเมิดความปลอดภัยหรือการบุกรุกมีเพิ่มมากขึ้น

การพัฒนาประสิทธิภาพของอัลกอริธึม Pattern Matching จึงเน้นไปที่การลดเวลาที่ต้องใช้ในการประมวลผล และการลดปริมาณหน่วยความจำที่ใช้งาน อัลกอริธึม Piranha [3] เป็นอัลกอริธึม Pattern Matching สำหรับระบบตรวจจับผู้บุกรุกอัลกอริธึมใหม่ที่มีประสิทธิภาพดีทั้งในแง่ของเวลาที่ใช้ในการประมวลผลและปริมาณหน่วยความจำที่ใช้เมื่อเปรียบเทียบกับอัลกอริธึมอื่นๆ ในปัจจุบัน และสามารถรองรับการใช้งานในอนาคต

จากการพิจารณาโครงสร้างการทำงานของอัลกอริธึม Piranha นั้นพบว่ายังมีข้อเสียบางประการที่สามารถพัฒนาเพื่อเพิ่มประสิทธิภาพของอัลกอริธึมได้ ในงานวิจัยนี้จึงนำอัลกอริธึม Piranha มาทำการพัฒนาโดยปรับปรุงในส่วนที่เป็นข้อเสียหรือปัญหาของอัลกอริธึม Piranha ด้วยเทคนิคของการทำ Hashing ที่เรียกว่า Cuckoo Hashing [4]

2.2 การละเมิดความปลอดภัยระบบคอมพิวเตอร์ [5]

2.2.1 ความหมายของการละเมิดความปลอดภัยระบบคอมพิวเตอร์

เหตุการณ์ละเมิดความปลอดภัย คือ การกระทำใดๆ ก็ตามที่เกี่ยวข้องกับความปลอดภัยระบบคอมพิวเตอร์ในด้านลบ เป็นการกระทำที่ละเมิดนโยบายความปลอดภัยมีหลากหลายรูปแบบและขนาดของความเสียหาย เหตุการณ์เหล่านี้สามารถมาจากทุกหนทุกแห่งบนอินเตอร์เน็ต ซึ่งรูปแบบการโจมตีทั่วไปก็ได้แก่การลักลอบใช้ชื่อบัญชีของผู้ใช้ การขโมยสิทธิพิเศษบนระบบการใช้ระบบของเหยื่อเป็นฐานทัพสำหรับโจมตีไซต์ (Site) ซึ่งสามารถทำได้โดยใช้เวลาเพียงไม่กี่วินาที และหากมีเครื่องมือที่สามารถสั่งให้ทำงานโดยอัตโนมัติก็ยิ่งใช้เวลาน้อยลงไป

โดยปกติการโจมตีระบบหรือเครือข่าย เมื่อถูกตรวจพบมักจะถูกแจ้งเตือนโดยมีข้อมูลพื้นฐานเกี่ยวกับการโจมตี ซึ่งมีรูปแบบเฉพาะ เพื่อทำให้ระบบการรักษาความปลอดภัยใช้การไม่ได้ แต่โดยรวมแล้วมีผลลัพธ์ก็คือ เป็นการทำลายคุณสมบัติ 4 ประการของการรักษาความปลอดภัย คือ

- **ความลับ (Confidentiality)** การโจมตีที่เป็นการทำลายความลับของข้อมูล คือ การที่ผู้บุกรุกสามารถเข้าถึงข้อมูลโดยที่ไม่ได้รับอนุญาตจากเจ้าของไม่ว่าจะเป็นความตั้งใจหรือไม่ก็ตาม
- **ความคงสภาพ (Integrity)** การโจมตีที่ถือว่าเป็นการทำลายความคงสภาพของข้อมูล หมายถึงการที่ผู้บุกรุกสามารถเปลี่ยนแปลงแก้ไขระบบ หรือข้อมูลที่อยู่ในระบบหรือที่อยู่ระหว่างการถ่ายโอนผ่านเครือข่าย
- **ความพร้อมใช้งาน (Availability)** การโจมตีที่เป็นการทำลายความพร้อมใช้งาน หมายถึง การโจมตีที่ทำให้ผู้ใช้หรือผู้ที่ได้รับอนุญาตไม่สามารถเข้าถึงและใช้งานระบบหรือข้อมูลได้
- **การควบคุมการเข้าถึง (Access Control)** การโจมตีที่ทำให้ผู้ที่ไม่ได้รับอนุญาตหรือผู้บุกรุกสามารถเข้าใช้ระบบได้ ซึ่งเป็นผลให้ความลับ ความคงสภาพ และความพร้อมใช้งานถูกละเมิดด้วยเช่นกัน

2.2.2 ชนิดของเหตุการณ์ละเมิดความปลอดภัย

หน่วยงาน CERT (Computer Emergency Response Team) ซึ่งเป็นหน่วยงานที่มีหน้าที่รวบรวมและกระจายข่าวสารข้อมูลเกี่ยวกับปัญหาด้านความปลอดภัยระบบคอมพิวเตอร์ได้จำแนกชนิดของเหตุการณ์ละเมิดความปลอดภัยออกเป็น 9 ชนิด คือ

1) หยั่งเชิงด้วยการ Probe คือ การหาลู่ทางเพื่อเข้าสู่ระบบแบบไม่ปกติ และเปิดเผยข้อมูลเกี่ยวกับระบบ ยกตัวอย่างเช่นการล็อกอิน (Login) เข้าสู่ระบบด้วยชื่อบัญชีที่ถูกยกเลิกการใช้งานไปแล้ว การ Probe ก็คือ การใช้เครื่องมืออิเลคทรอนิคส์สำหรับการทดสอบการหาระบบที่เปิดช่องโหว่ไว้ เปรียบเสมือนการทดลองหมุนลูกบิดประตูบ้านใครสักคนเพื่อตรวจดูว่าลืมล็อกหรือไม่ จะได้สามารถเข้าไปข้างในได้อย่างสะดวก การ Probe บางครั้งก็ส่งผลให้เกิดเหตุการณ์ที่ร้ายแรงกว่า แต่ก็มักจะเป็นผลจากความอยากรู้อย่างเห็นนั่นเอง

2) กระจายการหยั่งเชิงด้วย Scan เป็นการ Probe ไปยังเป้าหมายจำนวนมากๆ ด้วยเครื่องมืออัตโนมัติ การ Scan บางครั้งก็เกิดจากการตั้งค่าผิดพลาด (misconfiguration) หรือข้อผิดพลาด (error) อื่นๆ ของระบบ แต่มักเป็นต้นกำเนิดของการโจมตีโดยตรงบนระบบที่ผู้บุกรุกพบว่ามีจุดอ่อน

3) ขโมยใช้ชื่อบัญชีของคนอื่น (Account Compromise) เป็นการใช้ชื่อบัญชีของผู้ใช้โดยไม่ได้รับอนุญาตจากเจ้าของบัญชีนั้น แล้วเข้ามาทำลายข้อมูลสำคัญของเจ้าของบัญชี ขโมยข้อมูลสำคัญออกไป หรือไม่ก็แอบใช้บริการต่างๆ ด้วยสิทธิของเจ้าของบัญชีที่ได้มา

4) ขโมยชื่อบัญชีของผู้ดูแลระบบ (Root Compromise) มีลักษณะคล้ายกับ Account Compromise ยกเว้นชื่อบัญชีที่ถูกขโมยไปนั้นมีสิทธิพิเศษในระบบมากกว่า นั่นคือการขโมยชื่อบัญชี root ที่กล่าวถึงบ่อยๆ นั่นก็คือ ชื่อบัญชีบนระบบยูนิกซ์ที่มีสิทธิ์ทำอะไรก็ได้กับระบบโดยไม่มีข้อจำกัด หรือเรียกว่าสิทธิ์ของซุปเปอร์ยูสเซอร์ (superuser) นั่นเอง เมื่อผู้บุกรุกได้ root มาครองแล้ว ก็สามารถทำอะไรต่ออะไรก็ได้บนเครื่องเพราะถือว่าได้สิทธิ์ของการเป็นผู้ดูแลระบบมาแล้ว

5) ดักขโมยข้อมูลกลางทางด้วย Packet Sniffer เป็นโปรแกรมที่ใช้ดักจับข้อมูลระหว่างที่กำลังวิ่งอยู่บนเครือข่าย ข้อมูลเหล่านั้นอาจจะเป็นชื่อบัญชี รหัสผ่าน และข้อมูลของเจ้าของซึ่งวิ่งอยู่บนเครือข่ายในลักษณะที่เป็นตัวอักษรซึ่งไม่ได้เข้ารหัส เมื่อรหัสผ่านเป็นร้อยๆ พันๆ ถูกจับไว้ด้วยโปรแกรมนี้ ผู้บุกรุกก็สามารถขยายการโจมตีต่อระบบได้อย่างง่ายดาย

6) โจมตีให้ระบบล่ม (Denial of Service) จุดประสงค์การโจมตีด้วย Denial of Service หรือ DoS ไม่ใช่การพยายามเข้าถึงระบบหรือข้อมูลโดยไม่ได้รับอนุญาต แต่เป็นการทำให้ผู้ใช้ระบบไม่สามารถเข้าใช้งานระบบได้ตามปกติ ลักษณะของ DoS มีหลากหลายรูปแบบ อาจจะเป็นการทำให้ระบบเครือข่ายท่วมท้นไปด้วยข้อมูลจำนวนมหาศาล หรือการจงใจใช้ทรัพยากรของระบบที่มีจำนวนจำกัดให้หมดไปจนระบบไม่สามารถให้บริการต่อไปได้อีก ตัวอย่างเช่น การปิดกั้นระบบควบคุมโพรเซสเซอร์ หรือการปิดกั้นการเชื่อมต่อเครือข่าย

7) อาศัยความเชื่อถือเพื่อโจมตี เครื่องคอมพิวเตอร์บนเครือข่ายมักจะมีคุณสมบัติเกี่ยวกับการไว้ใจ (trust) เครื่องอื่น ๆ ที่อยู่บนเครือข่ายเดียวกัน อย่างเช่น ก่อนที่จะเอ็กซีคิวท์ (execute) บางคำสั่งเครื่องก็จะตรวจสอบที่กลุ่มของไฟล์ซึ่งระบุว่าเครื่องอื่นๆ บนเครือข่ายไหนบ้างที่ได้รับอนุญาตให้ใช้คำสั่งดังกล่าว ดังนั้นถ้าผู้โจมตีสามารถปลอมเครื่องคอมพิวเตอร์ให้เป็นเครื่องที่ไว้ใจได้ในเครือข่ายนั้น ก็จะสามารถเข้าถึงเครื่องอื่นๆ ได้โดยง่าย

8) โปรแกรมเจตนาร้าย (Malicious Code) เป็นโปรแกรมที่เมื่อถูกเอ็กซีคิวท์แล้วจะส่งผลที่ไม่เป็นที่ต้องการให้แก่ระบบ โดยผู้ใช้มักจะไม่ได้ระวังโปรแกรมเหล่านี้จนกระทั่งพบว่ามีความเสียหายเกิดขึ้นแล้ว รวมถึงโปรแกรมม้าโทรจัน , ไวรัส, และโค้ดหนอน โดยที่โทรจันและไวรัสปกติแล้วจะถูกซ่อนไว้ในโปรแกรมหรือไฟล์ต่างๆ ซึ่งผู้โจมตีทำการดัดแปลงให้ทำงานแบบให้ผลเกินคาด ส่วนโค้ดหนอนเป็นโปรแกรมที่จำลองตัวเองแล้วกระจายไปทั่วโดยไม่ต้องอาศัยมนุษย์ แต่

ไวรัสซึ่งจำลองตัวเองแล้วต้องมีมนุษย์เป็นผู้กระทำให้เผยแพร่กระจายไปสู่ระบบหรือโปรแกรมอื่น โปรแกรมเหล่านี้มักนำมาซึ่งการสูญหายของข้อมูล เสียเวลา ระบบไม่สามารถทำงานได้ปกติ และเหตุการณ์ละเมิดความปลอดภัยอื่นๆ ตามมา

9) โจมตีระบบที่มีผลต่อคนหมู่มาก เหตุการณ์เช่นนี้อาจไม่ค่อยเกิดขึ้น แต่จัดว่าเป็นการโจมตีที่รุนแรง โดยเกี่ยวข้องกับส่วนประกอบหลักของโครงสร้างอินเตอร์เน็ต ไม่ใช่เฉพาะบางระบบที่อยู่บนอินเตอร์เน็ต ตัวอย่างเช่น โจมตีเครื่องเซิร์ฟเวอร์ซึ่งให้บริการสอบถามชื่อลงทะเบียนไว้ (Network Name Servers) โจมตีผู้ให้บริการเครือข่าย (Network Access Providers) และโจมตีไซต์ขนาดใหญ่ที่มีผู้ใช้จำนวนมากๆ เป็นต้น การโจมตีแบบนี้มักมีผลหลายๆ ไซต์ ที่อยู่บนอินเตอร์เน็ตและทำให้การทำงานของไซต์เหล่านั้นช้าลง

2.3 แนวความคิดเบื้องต้นของระบบตรวจจับผู้บุกรุก [5]

เพื่อป้องกันเครือข่ายคอมพิวเตอร์ภายในจากอันตรายที่มาจากเครือข่ายคอมพิวเตอร์ภายนอก เช่น ผู้บุกรุก หรือ Hacker รวมทั้งไวรัสประเภทต่างๆ ระบบ Firewall จะอนุญาตให้เฉพาะข้อมูลที่มีคุณลักษณะตรงกับเงื่อนไขที่กำหนดไว้ผ่านเข้าออกระบบเครือข่ายภายในเท่านั้น แต่อย่างไรก็ดี ระบบ Firewall นั้นก็ไม่สามารถป้องกันอันตรายที่มาจากอินเตอร์เน็ตได้ทุกรูปแบบ ดังนั้น จึงไม่สามารถรับรองความปลอดภัยและความลับของข้อมูลได้ ถึงแม้จะมีการใช้ระบบ Firewall แล้วก็ตาม เพื่อแก้ไขปัญหาดังกล่าวนี้จึงจำเป็นต้องมีอุปกรณ์ หรือเครื่องมือที่ใช้ในการตรวจจับพฤติกรรมต่างๆ ที่มีอยู่ในเครือข่าย ซึ่งระบบที่ใช้จัดการกับปัญหาเช่นนี้ ได้แก่ ระบบตรวจจับผู้บุกรุก (IDS : Intrusion Detection System)

- **Intrusion Detection System (IDS)** คือ ระบบที่คอยตรวจจับการบุกรุกของผู้ที่ไม่ประสงค์ดี รวมไปถึงข้อมูลจำพวกไวรัส โดยสามารถทำการวิเคราะห์ข้อมูลทั้งหมดที่ผ่านเข้าออกภายในเครือข่ายว่า มีลักษณะการทำงานที่เป็นความเสี่ยงที่ก่อให้เกิดความเสียหายต่อระบบเครือข่ายหรือไม่ โดยระบบ IDS นี้ จะทำการแจ้งเตือนให้ผู้ดูแลระบบทราบ
- **Intrusion Prevention System (IPS)** คือ ระบบที่มีลักษณะเช่นเดียวกับระบบ IDS แต่มีความสามารถพิเศษมากกว่าระบบ IDS กล่าวคือ เมื่อตรวจพบข้อมูลที่มีลักษณะการทำงานที่เป็นความเสี่ยงต่อระบบเครือข่ายก็จะทำการป้องกันข้อมูลดังกล่าวนั้น ไม่ให้เข้ามาภายในเครือข่ายได้

2.3.1 ประเภทของ IDS

IDS แบ่งเป็น 2 ประเภท คือ Host-Based IDS และ Network-Based IDS

Host-Based IDS

เป็นโปรแกรมที่ทำงานบนเครื่องคอมพิวเตอร์ ทำงานโดยการวิเคราะห์ Log เพื่อค้นหาข้อมูลเกี่ยวกับการบุกรุก สำหรับระบบปฏิบัติการวินโดวส์จะตรวจสอบ Event Log ต่าง ๆ เช่น System, Application, Security โดยปกติจะอ่านเหตุการณ์ใหม่ที่เกิดขึ้นเปรียบเทียบกับกฎที่ตั้งไว้ก่อนล่วงหน้า ถ้าตรงกันจะทำการแจ้งเตือนทันที หรือจะตรวจจับการบุกรุกได้ต้องมีการบันทึกเหตุการณ์ต่างๆ ที่สำคัญที่เกิดขึ้นกับระบบ ไม่เช่นนั้นจะไม่มีข้อมูลในการวิเคราะห์เปรียบเทียบว่ามีการบุกรุกหรือไม่

- **ข้อดีของ Host-Based IDS**
  - สามารถตรวจพบทุกการบุกรุกกับเครื่องคอมพิวเตอร์นั้นๆ ได้เสมอ ถ้าระบบสามารถบันทึกเหตุการณ์ดังกล่าวใน Log ได้ หรือการบุกรุกมีการเรียกใช้ System Call

- สามารถบอกได้ว่าการบุกรุกนั้นสำเร็จหรือไม่ โดยการวิเคราะห์ข้อความใน Log หรือจากหลักฐานอื่นๆ
- สามารถระบุได้ว่ามีการเข้าใช้งานระบบอย่างผิดปกติโดยผู้ใช้ของระบบเอง

– **ข้อเสียของ Host-Based IDS**

- โพรเซสของ IDS อาจถูกโจมตีเองจนไม่สามารถแจ้งเตือนได้
- จะแจ้งเตือนก็ต่อเมื่อเหตุการณ์ที่เกิดขึ้นนั้นตรงกับที่กำหนดไว้ก่อนหน้า ถ้าเป็นเทคนิคใหม่ๆ IDS จะไม่รู้จักและไม่แจ้งเตือน
- การทำงานของ IDS อาจมีผลกระทบต่อประสิทธิภาพของโฮสต์ เนื่องจากต้องตรวจสอบ Log File และ System Call

Network-Based IDS

เป็นโปนแกรมพิเศษที่ทำงานบนคอมพิวเตอร์หนึ่งเครื่องต่างหาก โดยจะมีเน็ตเวิร์คการ์ดที่ทำงานในโหมด Promiscuous ซึ่งในโหมดนี้ เน็ตเวิร์คการ์ดจะรับ Packet ทั้งหมดที่วิ่งอยู่บนเครือข่ายส่งต่อไปให้แอพพลิเคชั่นทำการโพรเซสต่อไป ในขณะที่เน็ตเวิร์คการ์ดที่รันในโหมดธรรมดาจะรับเอาเฉพาะ Packet ที่มีที่อยู่ปลายทางตรงกับเครื่องเท่านั้น เมื่อทุกๆ Packet ส่งผ่านไปให้แอพพลิเคชั่นทำการวิเคราะห์ข้อมูลที่อยู่ภายในว่าตรงกับรูปแบบการบุกรุกที่เก็บไว้ในฐานข้อมูลก่อนหน้าหรือไม่ ซึ่งหากตรงก็จะทำการแจ้งเตือนทันที โดยในแต่ละ IDS จะมีฐานข้อมูลที่ใช้สำหรับเปรียบเทียบเพื่อบอกว่าเป็นการพยายามที่จะบุกรุกเครือข่ายหรือไม่ ซึ่งเรียกว่า Signature โดยจะใช้เพื่อเปรียบเทียบกับข้อมูลที่ได้จากการวิเคราะห์แพ็กเก็ตที่วิ่งในเครือข่าย ซึ่งหากผู้บุกรุกมีเทคนิคใหม่ๆ และ IDS ไม่มี Signature นี้ ก็จะไม่สามารถตรวจจับการบุกรุกประเภทนี้ได้ โดยส่วนใหญ่ IDS ประเภทนี้จะมีเน็ตเวิร์คการ์ดอยู่ 2 ตัว โดยตัวแรกจะทำการเชื่อมต่อกับเครือข่ายที่ต้องการเฝ้าระวังหรือตรวจจับการบุกรุก ซึ่งตัวนี้จะไม่มีการกำหนดหมายเลขไอพี ส่วนเน็ตเวิร์คการ์ดอีกหนึ่งตัวจะทำการเชื่อมต่อกับอีกเครือข่ายหนึ่งเพื่อใช้สำหรับส่งการแจ้งเตือนไปยังเซิร์ฟเวอร์ โดยเครือข่ายตัวหลังนี้จะต้องไม่ถูกเชื่อมต่อกับเครือข่ายหลักที่ IDS ที่การเฝ้าระวังหรือตรวจจับอยู่ เพื่อเป็นการป้องกันไม่ให้ถูกโจมตีเสียเอง

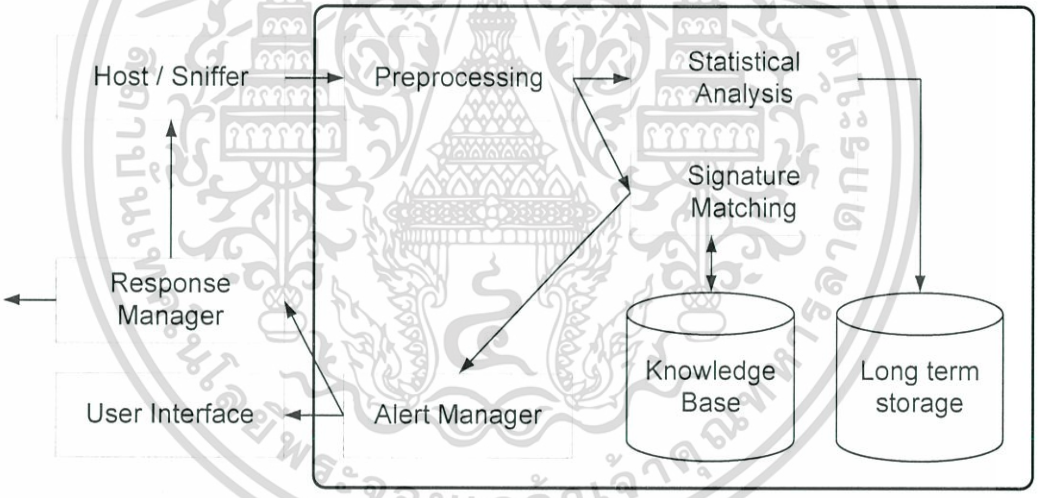
– **ข้อดีของ Network-Based IDS**

- จะถูกซ่อนอยู่ในเครือข่ายทำให้ผู้บุกรุกไม่รู้ว่าถูกเฝ้ามองอยู่
- Network-Based IDS หนึ่งเครื่องสามารถตรวจจับการบุกรุกได้หลายระบบหรือหลาย Host
- สามารถตรวจจับทุกๆ แพ็กเก็ตที่วิ่งไปยังระบบที่ถูกเฝ้าระวังอยู่

- **ข้อเสียของ Network-Based IDS**
  - จะแจ้งเตือนภัยก็ต่อเมื่อตรวจพบแพ็กเก็ตที่ตรงกับ Signature ที่กำหนดไว้ก่อนหน้าเท่านั้น
  - อาจจะไม่สามารถตรวจจับแพ็กเก็ตได้ทั้งหมด ในกรณีที่มีการใช้เครือข่ายอย่างหนาแน่น จนทำให้ IDS วิเคราะห์ Packet ที่วิ่งอยู่บนเครือข่ายไม่ทันหรือมีเส้นทางข้อมูลอื่นที่ไม่ต้องผ่าน IDS
  - ไม่สามารถสรุปได้ว่าการบุกรุกนั้นสำเร็จหรือไม่ และไม่สามารถตรวจสอบวิเคราะห์ Packet ที่ถูกเข้ารหัสไว้ได้
  - เครือข่ายที่ใช้สวิตช์ต้องทำการคอนฟิกเพื่อให้พอร์ตที่เชื่อมต่อกับ IDS นั้น สามารถมองเห็นทุก Packet ที่วิ่งผ่านสวิตช์หรือทำการ Port Mirroring

2.3.2 ส่วนประกอบของ IDS

โดยทั่วไป IDS จะประกอบด้วยส่วนต่างๆ ดังรูปที่ 2.1



รูปที่ 2.1 แสดงส่วนประกอบของระบบตรวจจับผู้บุกรุก

- Host System/Network Sniffer : ทำหน้าที่เป็นตัวตรวจจับเหตุการณ์ต่างๆ ที่เกิดขึ้นใน Host หรือเครือข่าย ข้อมูลจากส่วนนี้เป็นเหมือน Input ที่เข้าไปประมวลผลใน IDS
- Preprocessing : จัดรูปแบบของ Input ที่รับเข้ามาเพื่อให้นำไปประมวลผลได้สะดวกต่อไป
- Statistical Analysis : ส่วนวิเคราะห์ผลทางสถิติ

- Signature Matching : ส่วนวิเคราะห์จากพฤติกรรมที่มีแบบแผน แนวความคิดนี้นำมาจากหลักการของการบุกรุกซึ่งมักจะมีรูปแบบที่ค่อนข้างแน่นอน ส่วนนี้จะทำงานได้ก็ต่อเมื่อมีข้อมูลมากพอที่จะวิเคราะห์ได้ว่าพฤติกรรมที่ปรากฏในระบบเป็นรูปแบบของการบุกรุกหรือไม่
- Knowledge Base : เป็นตัวเก็บข้อมูลเกี่ยวกับพฤติกรรมของการบุกรุก ข้อมูลนี้ถูกใช้โดยส่วนของ Signature Matching ข้อมูลส่วนนี้มีความสำคัญมากกับระบบ เป็นตัวตัดสินว่าระบบฉลาดพอที่จะตรวจจับการบุกรุกได้หรือไม่
- Alert Manager : ทำหน้าที่เป็นตัวตัดสินใจว่าจะเหตุการณ์ที่เกิดขึ้นในระบบควรจะต้องเตือนหรือไม่ ระบบจะมีความอ่อนไหวมากน้อยแค่ไหนอยู่ที่ตัวนี้ด้วย
- User Interface : เป็นส่วนที่โต้ตอบกับผู้ใช้ อาจจะเป็นการแสดงผลที่หน้าจอ ส่งเสียงเตือนถึงการบุกรุก สั่งพิมพ์เป็น Hardcopy หรือแม้แต่เชื่อมกับโทรศัพท์ นอกจากนี้ User Interface ยังเป็นส่วนโต้ตอบระหว่างผู้ใช้กับระบบเพื่อเปลี่ยนแปลงหรือ Update ข้อมูลใน Knowledge Base
- Response Manager : รับข้อมูลจาก Alert Manager เพื่อนำมาตัดสินใจว่าจะโต้ตอบกับการบุกรุกอย่างไร

2.3.3 รูปแบบการทำงานของ IDS

รูปแบบการทำงานของ IDS แบ่งออกเป็น 5 ประเภทตามลักษณะของวิธีการตรวจจับ

1. **Anomaly Detection** : เป็นการตรวจหาสิ่งผิดปกติ กลไกก็คือต้องวิเคราะห์ระบบเครือข่าย (ขึ้นกับแหล่งข้อมูล) ให้ได้คำตอบก่อนว่าอะไรคือการทำงาน "ปกติ" การตรวจจับจะวัดตามค่าทางสถิติหรือ heuristic เพื่อดูว่าเหตุการณ์ที่เกิดอยู่ในขอบเขตของคำว่า "ปกติ" หรือไม่ ถ้าไม่แสดงว่าเหตุการณ์ที่เกิดเป็นสิ่งที่ไม่ปกติและหมายถึงการบุกรุก ปกติจะใช้การวิเคราะห์โดย Neural Nets, Statistical Analysis หรือ Atate-Changed Analysis ข้อดีของ Anomaly Detection คือ สามารถปรับให้ตรวจจับการบุกรุกได้ทุกประเภท รวมทั้งการบุกรุกที่ไม่เคยเกิดขึ้นมาก่อนด้วย ข้อเสียระบบนี้คือยังมีประสิทธิภาพที่ไม่ดีพอในปัจจุบัน ในการตรวจจับยังคงพบความผิดพลาด ตัวอย่างของระบบนี้คือ IDES/NIDES (Statistical + Rule-Based Detection), GrIDS (Graph-Based IDS), และ Emerald (Multi-Layer IDS)
2. **Misuse Detection** : หลักการคือ IDS จะหาว่าอะไรคือองค์ประกอบของการบุกรุกแล้วพยายามตรวจจับจุดนั้น วิธีการอาจจะใช้ "Network Grep" หา Strings ใน Network Connection ที่แสดงถึงการบุกรุกหรือ Pattern Matching ตรวจการเปลี่ยนแปลงของ State เช่น Owner ของ /etc/passwd ถูกเปลี่ยน ตามด้วย /etc/passwd ถูกเปิด แก้ไข และบันทึกลงไปใหม่ ข้อดีของ IDS ที่ทำงานแบบ Misuse Detection คือ การทำงานไม่ซับซ้อน ตรวจจับ

การบุกรุกได้รวดเร็ว โอกาสผิดพลาดมีน้อย แต่ก็มีข้อเสียคือต้องรู้จักวิธีการบุกรุกจึงจะตรวจจับได้ และมีโอกาสที่ระบบจะถูกลวงได้ง่าย ตัวอย่างของระบบนี้คือ ISS RealSecure, Cisco NetRanger, NAI CyberCop, NFR Network Flight Recorder

3. **Burglar Alarm** : คือ Misuse Detection ที่เจาะจงเป้าหมายที่แน่นอน การทำงานจะขึ้นอยู่กับการตั้ง Policy ของ Site นั้นๆ การตรวจจับการบุกรุกจะตรวจจับการฝ่าฝืน Policy ที่กำหนดไว้เป็นหลัก Burglar Alarm ตัวอย่าง IDS ที่ทำงานแบบ Burglar Alarm เช่น โปรแกรม NetLog, โปรแกรม Network Flight Recorder ข้อดีของ IDS ที่ทำงานแบบนี้คือ ความน่าเชื่อถือ การทำงานไม่ซับซ้อนและอาจจะตรวจจับการบุกรุกที่ไม่รู้จักมาก่อนได้ ข้อเสียคือเป็น Policy-Based ซึ่งต้องอาศัยความรู้และประสบการณ์ในการตั้ง Policy
4. **Honey Pots** : ระบบลวงตั้งเพื่อล่อให้ถูกบุกรุก ระบบต้องอ่อนพอที่จะบุกรุกได้และแข็งพอที่จะถ่วงเวลาผู้บุกรุกได้นานพอ ส่วนการตรวจจับสามารถใช้ Tools ต่างๆ เช่น Tcpwrapper, Burglar Alarm หรือแม้แต่ System Logs วิธีการนี้มีข้อดีในเรื่องของการทำงานที่ไม่ซับซ้อนและไม่ลดประสิทธิภาพของระบบ ส่วนข้อเสียของวิธีการนี้คือสามารถใช้ได้ผลกับผู้บุกรุกที่มีความสามารถระดับต่ำเท่านั้น
5. **Hybrid IDS** : ระบบตรวจจับผู้บุกรุกที่รวมเอาการทำงานของระบบตรวจจับผู้บุกหลายๆ ประเภทมาผสมกัน ระบบที่มีจำหน่ายในปัจจุบันส่วนใหญ่จะเป็น Hybrid IDS ซึ่งมีการทำงานแบบ Misuse Detection + Expert System + Statistical Anomaly Analysis ทั้งนี้ Hybrid IDS ดูเหมือนจะเป็นหลักการของระบบตรวจจับผู้บุกรุกที่ดีที่สุด เพราะใช้การทำงานหลายๆ อย่างเพื่อตรวจจับและกำจัดข้อเสีย แต่ยังมีข้อเสียอื่นๆ คือ Hybrid IDS ปัจจุบันเกิด False Positive มากเกินไป

2.3.4 การละเมิดความปลอดภัยที่มักตรวจพบโดยระบบตรวจจับผู้บุกรุก

การละเมิดความปลอดภัยส่วนใหญ่ที่ตรวจพบคือ การสแกนระบบ (System Scanning) การปฏิเสธการให้บริการ (Deny of Service) และการเจาะเข้าระบบ (System Penetration) ซึ่งอาจเริ่มจากภายนอกผ่านทางเครือข่ายหรือในองค์กรก็ได้

1. **System Scanning** หมายถึง การตรวจหาคุณสมบัติบางอย่างของระบบซึ่งอาจทำได้โดยส่ง Packet ประเภทต่างๆ ไปยังระบบเพื่อหาช่องโหว่หรือจุดอ่อนของระบบนั้น เช่น การตรวจหาโทโปโลยีของเครือข่ายที่โจมตี, ประเภททราฟฟิกที่อนุญาตให้ผ่านไฟร์วอลล์, Host ที่เปิดใช้งานอยู่, ประเภทและเวอร์ชั่นของระบบปฏิบัติการที่ Host ใช้อยู่, โปรแกรมที่ใช้งานอยู่ในเครื่องเซิร์ฟเวอร์ และเวอร์ชั่นของโปรแกรมนั้นๆ เนื่องจากการสแกนระบบหรือเครือข่ายไม่ใช่การโจมตีจริงๆ บางครั้งอาจไม่ผิดกฎหมาย แต่นับเป็นขั้นตอนแรกของความพยายามที่จะทำการโจมตีเครือข่าย จึงมีความจำเป็นที่จะต้องแจ้งเตือนให้ผู้ดูแลระบบ

ทราบเพื่อทำการเฝ้าระวังการโจมตีที่อาจเกิดตามมา การสแกนระบบหรือเครือข่ายนั้นเป็นสิ่งที่เกิดขึ้นเป็นประจำบนเครือข่ายอินเทอร์เน็ต ดังนั้นระบบตรวจจับผู้บุกรุกที่ดีส่วนใหญ่จะสามารถแยกแยะได้ว่าการสแกนใดเป็นการกระทำประสงค์ดีหรือประสงค์ร้าย

2. **Denial of Service Attacks** การโจมตีแบบปฏิเสธการให้บริการ (Denial of Service : DoS) เป็นการพยายามที่จะทำให้ระบบที่เป็นเป้าหมายทำงานช้าลง หรือไม่สามารถให้บริการต่อไปได้ บนอินเตอร์เน็ตนั้น DoS เป็นสิ่งที่เกิดเป็นประจำ แต่อาจไม่ส่งผลเสียมากนัก DoS สามารถใช้โจมตีองค์กรใหญ่ๆ ได้ การโจมตีประเภทนี้สามารถกระทำได้ 2 แบบ คือ
   - **การโจมตีช่องโหว่ (Flaw Exploitation DoS)** เป็นการโจมตีระบบเพื่อให้เกิดข้อผิดพลาด หรือ ทำให้ทรัพยากรของระบบเช่น CPU, Memory, Hard Disk, Buffer หรือ Band Width ของเครือข่าย ถูกใช้งานจนหมด ทำให้ระบบที่ถูกโจมตีไม่สามารถจัดการกับแพ็กเก็ตที่ผิดปกตินี้ได้และระบบล่มในที่สุด การป้องกันการโจมตีแบบนี้ทำได้โดยการปิดช่องโหว่หรือการอัพเกรดโปรแกรมต่างๆ ที่ใช้ในระบบ
   - **การผลัดดิง (Flooding)** เป็นการโจมตีโดยการส่งข้อมูลเกินกว่าที่ระบบสามารถรับได้ หรืออาจใช้ Band Width ของระบบไปจนหมด ทำให้ผู้อื่นไม่สามารถเข้ามาใช้งานระบบได้ การโจมตีแบบนี้ไม่ได้อาศัยช่องโหว่ของระบบ ดังนั้นการอัพเกรดโปรแกรมที่ใช้ในระบบไม่สามารถป้องกันการโจมตีแบบนี้ได้ นอกจากนี้การโจมตีแบบนี้ยังอาจมีการโจมตีแบบแยกกระจาย คือ การใช้คอมพิวเตอร์มากกว่าหนึ่งเครื่องทำการโจมตีเป้าหมาย โดยทำการควบคุมจากศูนย์กลาง โดยมีเครื่องของผู้บุกรุกเป็นผู้ควบคุมเพื่อทำการโจมตีระบบใหญ่ๆ
3. **Penetration Attacks** การโจมตีแบบเจาะเข้าระบบ เป็นการเข้ามาในระบบโดยไม่ได้รับอนุญาตและทำการเปลี่ยนแปลงสิทธิ์ ทรัพยากรและข้อมูลที่อยู่ในระบบ การโจมตีแบบนี้จะทำการหลบเลี่ยงหรือฝ่าฝืนระบบควบคุมการเข้าถึงและเป็นการทำลายความคงสภาพของระบบ เทคนิคการโจมตีแบบนี้อาจจำแนกออกเป็นดังนี้
   - **User to Root** : ยูสเซอร์ของระบบสามารถเลื่อนสิทธิ์ให้ตัวเองเป็นผู้ดูแลระบบ หรือ Root สำหรับ ยูนิกส์ และ Administrator ของวินโดวส์
   - **Remote to User** : ผู้บุกรุกจากเครือข่ายสามารถเลื่อนสิทธิ์ให้ตัวเองเป็นผู้ใช้ของเครื่องเป้าหมาย
   - **Remote to Root** : ผู้บุกรุกจากเครือข่ายสามารถเลื่อนสิทธิ์ให้ตัวเองเป็นผู้ดูแลระบบและคุมระบบได้ทั้งหมด
   - **Remote Disk Read** : ผู้บุกรุกจากเครือข่ายสามารถทำให้ตัวเองมีสิทธิ์ในการอ่านไฟล์ข้อมูลในระบบโดยที่ไม่ต้องได้รับอนุญาตจากเจ้าของไฟล์

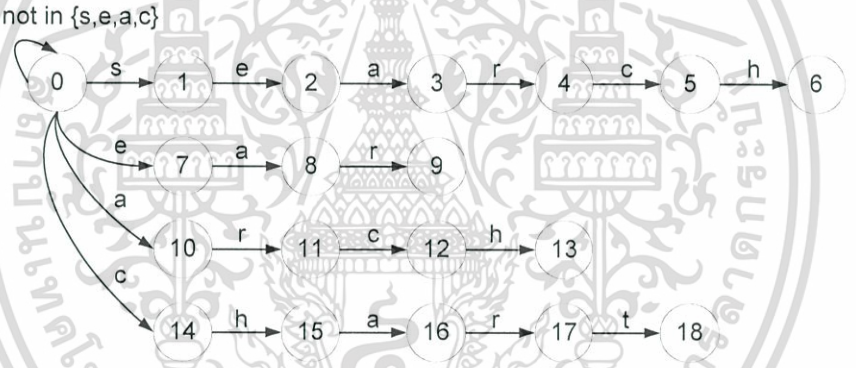
- **Remote Disk Write** : ผู้บุกรุกจากเครือข่ายสามารถทำให้ตัวเองมีสิทธิ์ในการเขียนหรือแก้ไขไฟล์ข้อมูลที่อยู่ในระบบโดยที่ไม่ต้องได้รับอนุญาตจากเจ้าของไฟล์

4. **Remote และ Local Attacks** การโจมตีแบบ DoS และแบบเจาะเข้าระบบ มีแหล่งที่มาของการโจมตี 2 แหล่ง คือ
   - **Authorized User Attacks** : เกิดจากบุคคลภายใน คือ ผู้ใช้ระบบที่พยายามเปลี่ยนแปลงสิทธิ์ตนเองให้สามารถเข้าใช้ระบบได้มากขึ้น
   - **Public User Attacks** : เกิดจากบุคคลภายนอกหรือผู้ใช้ทั่วไป ที่ไม่มีสิทธิ์ใดๆ ที่จะเข้าใช้ระบบ ซึ่งจะเริ่มจากเครื่องรีโมทโดยใช้ช่องทางที่ระบบเปิดไว้ให้ หรือเป็นช่องโหว่ของระบบเอง

2.4 อัลกอริธึม Pattern Matching ที่ใช้ในระบบตรวจจับผู้บุกรุก

2.4.1 อัลกอริธึม Aho-Corasick (AC) [6]

โครงสร้างการทำงานของอัลกอริธึม Aho-Corasick เป็น State Machine โดยในขั้นตอนการเตรียมโครงสร้างข้อมูลสำหรับการทำ Pattern Matching อัลกอริธึม Aho-Corasick จะอ่านข้อมูล Pattern ทุก Pattern แล้วนำมาสร้างเป็น State Machine ดังรูปที่ 2.2 เป็นตัวอย่างการสร้าง State Machine จาก Pattern "search, ear, arch" และ "chart" สำหรับ Node เริ่ม (Root Node) จะเป็น Empty Node มีสถานะเป็น Non-Matching State และเปลี่ยนสถานะโดยรับ Input เป็นตัวอักษรของแต่ละ Pattern ที่แต่ละ Node จะมี Failure Function ที่ชี้ไปยัง Node อื่นๆ ซึ่งเป็น Node ที่เมื่อรับ Input ตัวต่อมาแล้วจะเปลี่ยนไปตรวจสอบ Pattern อื่นๆ ที่เป็นไปได้แทน ตัวอย่างตาราง Failure Function ดังรูปที่ 2.3 และสุดท้ายอัลกอริธึมจะสร้างตาราง Output Function เป็นการบอกถึง Pattern ที่ Match หาก State ของ Machine เปลี่ยนไปยัง Node นั้นๆ ตัวอย่างตาราง Output Function ดังรูปที่ 2.4



รูปที่ 2.2 แสดงตัวอย่าง State Machine ของอัลกอริธึม Aho-Corasick

nodes	0	1	2	3	4	5	6	7	8	9
fail	0	0	7	8	9	12	13	0	10	11
nodes	10	11	12	13	14	15	16	17	18	
fail	0	0	14	15	0	0	10	11	0	

รูปที่ 2.3 แสดงตัวอย่างตาราง Failure Function ของอัลกอริธึม Aho-Corasick

nodes	4	6	9	13	18
out	{ear}	{search, arch}	{ear}	{arch}	{chart}

รูปที่ 2.4 แสดงตัวอย่างตาราง Output Function ของอัลกอริธึม Aho-Corasick

Step	Input	State	Output
1	s	1	
2	t	0	
3	r	0	
4	c	14	
5	m	0	
6	a	10	
7	t	0	
8	e	7	
9	c	0	
10	a	10	
11	d	0	
12	n	0	
13	s	1	
14	e	2	
15	a	3	
16	r	4	{ear}
17	c	5	
18	h	6	{search,arch}
19	o	0	
20	f	0	

รูปที่ 2.5 แสดงตัวอย่างกระบวนการ Search ของอัลกอริธึม Aho-Corasick

ตัวอย่างการทำงานของอัลกอริธึม Aho-Corasick ดังรูปที่ 2.5 เป็นตัวอย่างการค้นหา Pattern ในข้อความ "strcmatecadnsearchof" จะมีขั้นตอนการตรวจสอบโดยเริ่มจาก Root Node รับ Input ตัวแรกเป็น s State Machine จะเปลี่ยนสถานะไปยัง State 1 จากนั้นรับ Input ตัว t พบว่าไม่สามารถไปยัง State ถัดไปได้จึงต้องดูค่า State ถัดไปจากตาราง Failure Function ซึ่งพบว่า ต้องไปต่อยัง State 0 จากนั้นรับ Input ตัว r State Machine ไม่สามารถไปยัง State อื่นได้ ยังคงอยู่ที่ State 0 สำหรับ Input ตัวต่อๆ ไป การทำงานยังคงเหมือนเดิม รายละเอียดการทำงานของ State Machine ดังรูปที่ 2.5 โดย Step ที่ 16 จะได้ Output เป็น Pattern "ear" และ Step ที่ 17 ได้ Output เป็น Pattern "search" และ "arch" แสดงว่า ข้อความตัวอย่างนี้ Match กับ Pattern ทั้งหมด 3 Pattern

ความเร็วในการทำ Pattern Matching ของอัลกอริธึม Aho-Corasick ขึ้นอยู่กับจำนวน Pattern ที่ต้องทำการค้นหา ในการเปรียบเทียบต้องทำการเปรียบเทียบตัวอักษรต่อตัวอักษรโดยไม่มีการกระโดดข้าม ค่า Time Complexity เป็น $O(n\log\sigma)$ โดยที่ σ คือ จำนวนตัวอักษรที่เป็นไปได้ทั้งหมดของ Pattern

สำหรับการใช้งานจริงกับโปรแกรม snort นั้นพบว่า อัลกอริธึม Aho-Corasick เป็นอัลกอริธึมที่มีประสิทธิภาพดีที่สุดในแง่ของระยะเวลาในการประมวลผล (CPU Time) แต่มีข้อเสียในแง่ของปริมาณหน่วยความจำที่ใช้ เมื่อเปรียบเทียบกับอัลกอริธึมอื่นๆ

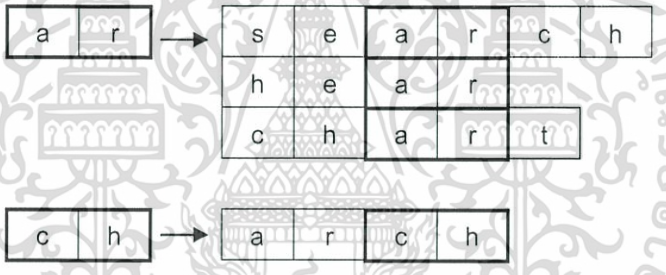
2.4.2 อัลกอริธึม Wu-Manber [7]

อัลกอริธึม Wu-Manber ทำงานอยู่บนพื้นฐานของอัลกอริธึม Boyer-Moore ใช้เทคนิค Bad-Character Shift ทำการพิจารณาข้อความ Input เป็นชุด (Block) แทนการพิจารณาทีละตัวอักษร ในขั้นตอนการเตรียมโครงสร้างข้อมูลอัลกอริธึม Wu-Manber จะสร้างตารางสำหรับใช้งาน 2 ตารางคือ ตาราง Shift-Table ดังรูปที่ 2.6 และตาราง Hash Table ดังรูปที่ 2.7

ตาราง Shift-Table จะเป็นตารางที่บอกค่าจำนวนตัวอักษรที่เลื่อนถัดไปเพื่อเปลี่ยนชุดข้อความที่จะนำมาพิจารณา ส่วนตาราง Hash Table จะเก็บข้อมูล Pattern ต่างๆ ซึ่งจะใช้เมื่อพบว่าค่า Shift ที่ได้จากตาราง Shift Table เป็น 0

BC	ar	ch	ea	ha	he	rc	se	others
shift	0	0	1	1	2	1	2	3

รูปที่ 2.6 แสดงตัวอย่างตาราง Shift-Table ของอัลกอริธึม Wu-Manber



รูปที่ 2.7 แสดงตัวอย่าง Hash Table ของอัลกอริธึม Wu-Manber

Step	s t r **c** **m** a t **e** c a **d** n s **e** a **r** **c** **h** **o** f	shift	Output
1	↑	1	
2	↑	3	
3	↑	3	
4	↑	3	
5	↑	2	
6	↑	0	{search}
7	↑	1	
8	↑	0	{arch}
9	↑	3	

รูปที่ 2.8 แสดงตัวอย่างการทำงานของอัลกอริธึม Wu-Manber

ตัวอย่างการทำงานของอัลกอริธึม Wu-Manber ดังรูปที่ 2.8 เป็นตัวอย่างการค้นหา Pattern ในข้อความ "strcmatecadnsearchof" กำหนดการพิจารณาอักษรชุดละ 4 ตัวอักษร ตัวอักษร 4 ตัวแรกที่นำมาพิจารณาคือ strc นำค่า rc ไปตรวจสอบใน Shift Table และได้ค่าเป็น 1 ดังนั้นจึงเลื่อนชุดอักษรที่จะทำการพิจารณาไป 1 ตัวอักษร ได้ชุดอักษรเป็น trcm จากนั้นนำค่า cm ไปตรวจสอบใน Shift Table และได้ค่าเป็น 3 (others) ดังนั้นจึงเลื่อนข้อมูลชุดอักษรที่จะทำการพิจารณาไป 3 ตัวอักษร ได้ชุดอักษรเป็น mate การทำงานจะเป็นเช่นนี้เรื่อยไป จนกระทั่งถึงขั้นตอนที่ 6 (Step 6) ซึ่งพบว่าค่าจากตาราง Shift Table เป็น 0 ดังนั้นจึงนำค่าตัวอักษร ar ไปทำการ Search หา Pattern จากตาราง Hash Table จนในที่สุดได้ Output Match กับ Pattern "search" กระบวนการทำงานเช่นนี้จะดำเนินเรื่อยไปจนครบทุกตัวอักษรของข้อความ

ประสิทธิภาพการทำงานของอัลกอริธึม Wu-Manber ไม่ขึ้นอยู่กับจำนวน Pattern มีค่า Best-Case Complexity เป็น O(B n/m) โดยที่ B คือ ขนาดของชุดข้อมูลที่ทำการพิจารณา (Block Size) n คือ ความยาวของข้อความ Input และ m คือ ความยาวของ Pattern จากค่า Complexity จะเห็นได้ว่า อัลกอริธึม Wu-Manber จะมีประสิทธิภาพลดลงหากความยาวของ Pattern มีค่าน้อย ซึ่งเป็นข้อด้อยของอัลกอริธึมนี้

สำหรับการใช้งานจริงกับโปรแกรม snort นั้นพบว่า อัลกอริธึม Wu-Manber เป็นอัลกอริธึมที่มีประสิทธิภาพดีในแง่ของระยะเวลาในการประมวลผล (CPU Time) และปริมาณหน่วยความจำที่ใช้เมื่อเปรียบเทียบกับอัลกอริธึมอื่นๆ

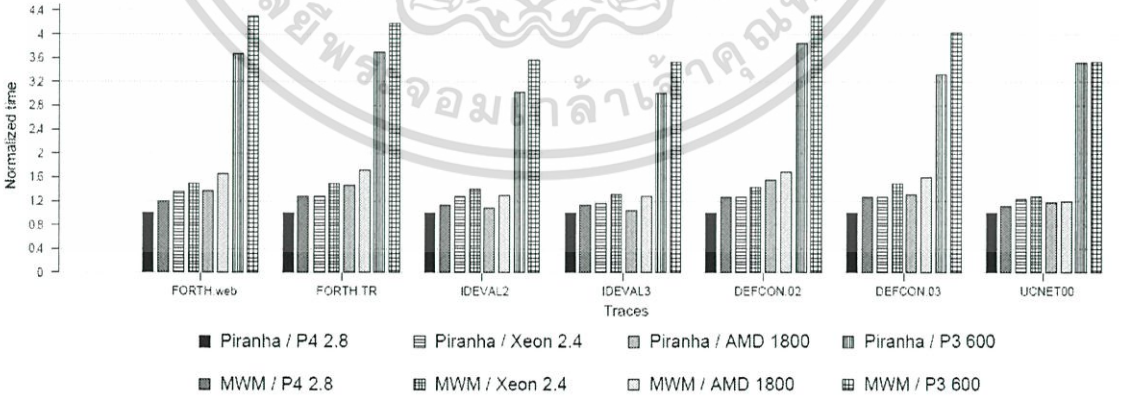
2.5 งานวิจัยที่เกี่ยวข้อง

2.5.1 งานวิจัยเรื่อง Piranha : Fast and Memory-efficient Pattern Matching for Intrusion Detection [3]

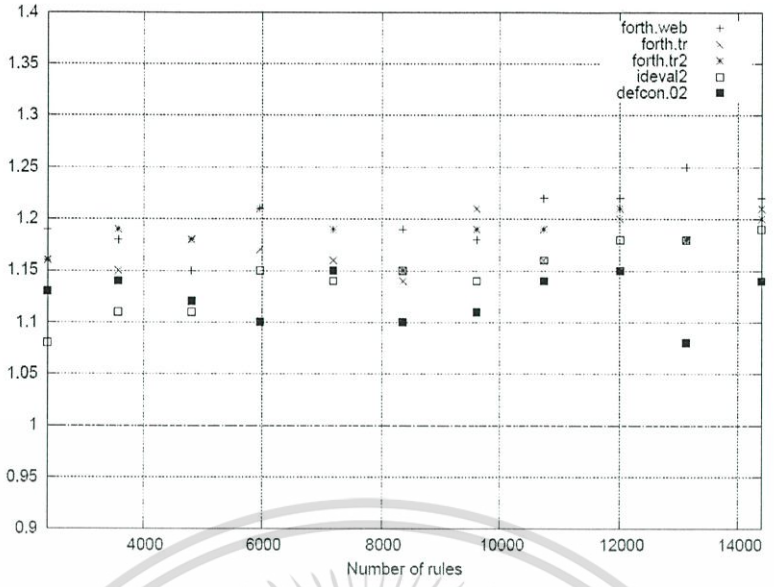
งานวิจัยนี้ได้นำเสนออัลกอริธึมใหม่ที่ใช้สำหรับการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุก โดยมีแนวความคิดของการตรวจสอบโดยใช้ข้อมูลเพียงส่วนหนึ่งของ Pattern (Sub-Pattern) ที่มีความยาว 4 ตัวอักษร (32 Bit) ทำการตรวจสอบกับ Payload ของ Packet และหากพบว่า Sub-Pattern ใดไม่ปรากฏใน Payload ของ Packet นั้นจะสามารถสรุปได้ทันทีว่า Packet นั้นไม่ได้ Match กับ Pattern นั้นๆ

อัลกอริธึม Piranha สามารถรองรับการตรวจสอบ Pattern ที่มีความยาวมากกว่า 4 ตัวอักษร ส่วน Pattern ที่มีความยาวน้อยกว่า 4 ตัวอักษร จะใช้วิธีการตรวจสอบด้วยวิธีการธรรมดา คือ ตรวจสอบตัวอักษรต่อตัวอักษร

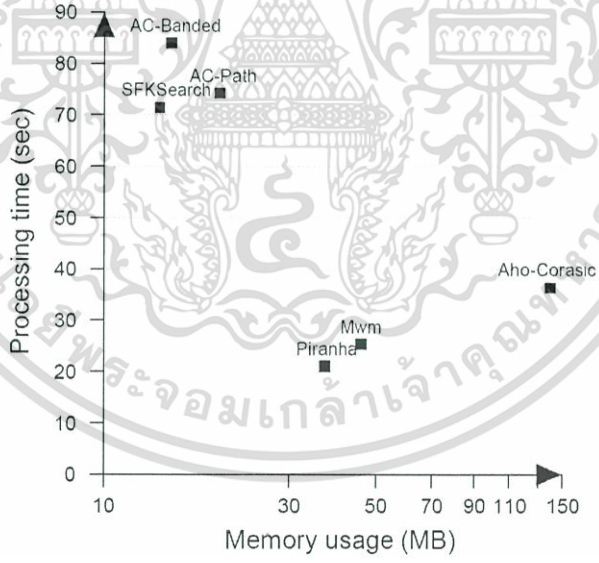
ด้วยแนวความคิดดังกล่าวทั้งหมดนี้ทำให้สามารถลดขั้นตอนการเปรียบเทียบได้มาก เนื่องจากไม่ต้องทำการเปรียบเทียบตลอดความยาวทั้ง Pattern และจากการทดลองพบว่า อัลกอริธึม Piranha สามารถทำงานได้ดีโดยไม่ขึ้นอยู่กับสถาปัตยกรรมคอมพิวเตอร์ ดังแสดงในรูปที่ 2.9 และสามารถทำงานได้ดีแม้จำนวน Pattern (จำนวน Rule) เพิ่มมากขึ้น ดังแสดงในรูปที่ 2.10 นอกจากนี้อัลกอริธึม Piranha สามารถประมวลผลได้เร็วกว่าอัลกอริธึมอื่นๆ ถึง 28 เปอร์เซ็นต์ และใช้ปริมาณหน่วยความจำน้อยกว่าอัลกอริธึมอื่นๆ 73 เปอร์เซ็นต์ ดังนั้นจึงสามารถสรุปได้ว่า อัลกอริธึม Piranha มีประสิทธิภาพที่ดีทั้งในแง่เวลาที่ใช้ในการประมวลผล (Processing Time) และปริมาณหน่วยความจำที่ใช้ (Memory Usage) เมื่อเปรียบเทียบกับอัลกอริธึมอื่นๆ ดังแสดงในรูปที่ 2.11



รูปที่ 2.9 แสดงประสิทธิภาพของอัลกอริธึม Piranha เมื่อทำงานในสถาปัตยกรรมคอมพิวเตอร์ต่างๆ [3]



รูปที่ 2.10 แสดงการเปรียบเทียบประสิทธิภาพเป็นอัตราส่วนของเวลาที่ใช้ในการประมวลผลของอัลกอริธึม Piranha และอัลกอริธึม Wu-Manber ในกรณีที่จำนวน Rule เพิ่มมากขึ้น คำนวณจากเวลาของอัลกอริธึม Wu-Manber หารด้วยเวลาของอัลกอริธึม Piranha [3]



รูปที่ 2.11 แสดงการเปรียบเทียบประสิทธิภาพของอัลกอริธึม Pattern Matching ต่างๆ โดย AC คือ Aho-Corasic และ Mwm คือ Modified Wu-Manber [3]

2.5.2 งานวิจัยเรื่อง Cuckoo Hashing [4]

งานวิจัยนี้ได้นำเสนอเทคนิคในการทำ Hashing แบบใหม่ มีโครงสร้างการทำงานที่ไม่ซับซ้อน โครงสร้างการเก็บข้อมูลจะใช้ Hash Table จำนวน 2 Table (T_1 และ T_2) และ Hash Function จำนวน 2 Function (H_1 และ H_2) โดยที่ Function H_1 ใช้สำหรับ Hash ข้อมูลลง Table T_1 และ Function H_2 ใช้สำหรับ Hash ข้อมูลลง Table T_2 สำหรับ Hash Function ที่ใช้งานจะต้องสามารถเปลี่ยน Function ได้เมื่อเกิดการ Rehash ในกรณีที่เกิด Collision และไม่สามารถเก็บข้อมูลลงทั้ง Table T_1 และ Table T_2 ได้

ประสิทธิภาพของอัลกอริธึม Cuckoo Hashing ที่เป็นจุดเด่นของงานวิจัยนี้คือ มี Worst Case Lookup Time เป็นค่าคงที่ (O(2)) นั่นคือแต่ละครั้งที่ทำการ Lookup ข้อมูล จะมีการ Access หน่วยความจำอย่างมากที่สุดไม่เกิน 2 ครั้ง สำหรับเนื้อที่ (Space) ที่ใช้ในการเก็บข้อมูลจะเท่ากับ 2n เมื่อ n คือจำนวนข้อมูลทั้งหมดที่ต้องการเก็บลง Hash Table

บทที่ 3

ความรู้พื้นฐานในเรื่องที่เกี่ยวข้องกับงานวิจัย

ในบทนี้จะได้กล่าวถึงทฤษฎี และความรู้พื้นฐานที่ใช้ประกอบการพัฒนาประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุกในงานวิจัยนี้ ได้แก่ ระบบตรวจจับผู้บุกรุก Snort, ความหมายของ Pattern Matching, อัลกอริธึม Piranha และอัลกอริธึม Cuckoo Hashing

3.1 ระบบตรวจจับผู้บุกรุก Snort

โปรแกรม Snort เป็นโปรแกรมแบบ Open Source ที่ทำหน้าที่ตรวจจับข้อมูลในระบบเครือข่ายเพื่อใช้ในการวิเคราะห์พฤติกรรมการใช้งานเครือข่าย โดยสามารถทำงานได้ 4 Mode คือ

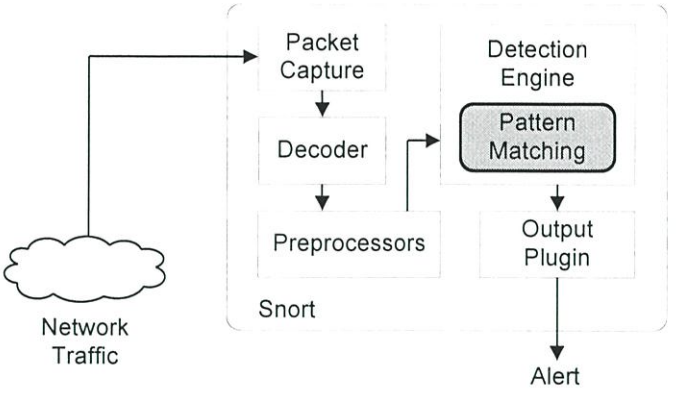
1) **Sniffer Mode** ใช้ในการดักจับข้อมูลบนเครือข่ายเพื่อใช้ในการวิเคราะห์พฤติกรรมการใช้งานของผู้ใช้ในระบบ

2) **Packet Locker Mode** หลักการทำงานคล้ายคลึงกับ Sniffer Mode แต่จะทำการเก็บข้อมูลไว้ใน Database เพื่อใช้ในการวิเคราะห์ในภายหลัง

3) **NIDS Mode (Network Intrusion Detection System Mode)** มีหลักการทำงานคือ ทำการวิเคราะห์ข้อมูลที่ไหลผ่านระบบ Network แบบ Real Time โดยเทียบกับ Signature หรือ Rules เพื่อใช้วิเคราะห์ข้อมูลเหล่านั้นว่าเป็นอันตรายต่อระบบหรือไม่

4) **Inline Mode** เป็น Mode ที่ทำหน้าที่เป็น IPS (Intrusion Protection System) ซึ่งสามารถวิเคราะห์ข้อมูลที่ไหลผ่านแบบ Real Time โดยเทียบกับเงื่อนไขแบบใหม่ (New Rules) เพื่อใช้ในการตัดสินใจและกรองข้อมูลที่ต้องสงสัยว่าจะเป็นอันตรายต่อระบบ

โปรแกรม Snort ได้เริ่มพัฒนามาตั้งแต่ช่วงปี 1980 โดยนาย Richard Stallman ที่มหาวิทยาลัย MIT มีจุดเด่นที่สามารถปรับแต่งการทำงานได้อย่างละเอียด กลไกการทำงานของโปรแกรม Snort ดังรูปที่ 3.1 เริ่มตั้งแต่การดักจับ Packet จากเครือข่ายด้วย libpcap [8] จากนั้นโมดูล Decoder จะทำการอ่านข้อมูล Packet และจัดเก็บลงโครงสร้างข้อมูลที่สามารถนำไปประมวลผลได้ ข้อมูลนี้จะถูกส่งต่อไปให้โมดูล Preprocessors ทำการตรวจสอบพฤติกรรมของ Packet ในเบื้องต้น และพิจารณาว่าจะต้องส่งไปให้ Detection Engine ประเภทใดทำการตรวจสอบต่อไป หาก Packet ดังกล่าวจำเป็นต้องทำการตรวจสอบ Payload ก็จะถูกส่งไปให้โมดูล Pattern Matching ซึ่งเป็นส่วนหนึ่งของ Detection Engine เพื่อตรวจหารูปแบบของการละเมิดความปลอดภัยตาม Signature หรือ Rule ที่มีอยู่ จากนั้นผลที่ได้จะถูกส่งต่อไปยัง Output Plugin เพื่อทำการเก็บบันทึกข้อมูลและทำการแจ้งเตือนไปยังผู้ดูแลระบบต่อไป หากตรวจพบการละเมิดความปลอดภัย



รูปที่ 3.1 แสดงส่วนประกอบและการส่งข้อมูลของโปรแกรม Snort

3.2 ความหมายของ Pattern Matching

Pattern Matching หรือ Content Matching หรือ String Matching คือ กระบวนการค้นหา string ย่อย $P = p_1 , p_2 , p_3 ,\ldots, p_m$ ในข้อความหรือ Text $T= t_1 ,t_2 ,t_3 ,\ldots, t_n$ โดยที่ $n>>m$

3.3 อัลกอริธึม Piranha [3]

อัลกอริธึม Piranha ทำงานโดยการตรวจหาตัวแทนของ Pattern (Sub-Pattern) ใน Payload ของแต่ละ Packet โดยตัวแทนของแต่ละ Pattern จะมีความยาว 4 ตัวอักษร หากตรวจพบตัวแทนของ Pattern ใน Payload ของ Packet ใดจะตั้งสมมติฐานทันทีว่า Packet นั้น Match กับ Pattern ดังกล่าว

ตัวแทนของ Pattern ขนาด 4 ตัวอักษร จะคัดเลือกโดยทำการเลือก 4 ตัวอักษรย่อยของ Pattern ที่พบใน Pattern อื่นน้อยที่สุด ทั้งนี้การทำงานของอัลกอริธึม Piranha จะสามารถใช้งานได้กับ Pattern ที่มีความยาวตั้งแต่ 4 ตัวอักษรขึ้นไป ส่วน Pattern ที่มีความยาวน้อยกว่า 3 ตัวอักษร จะใช้วิธีการอื่นในการตรวจหา

การทำงานของอัลกอริธึม Piranha แบ่งเป็น 2 ขั้นตอน คือ

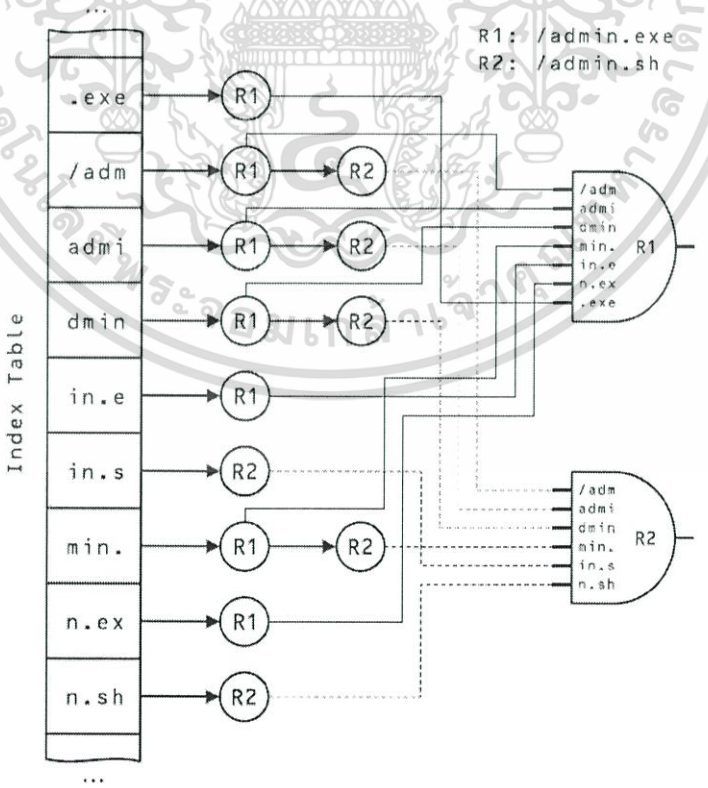
3.3.1 Preprocessing

ขั้นตอนนี้เป็นขั้นตอนที่อัลกอริธึม Piranha ทำการเตรียมโครงสร้างข้อมูลสำหรับใช้ในขั้นตอน Searching โดยเริ่มจากการรับข้อมูล Pattern จากแต่ละ Pattern อัลกอริธึม Piranha จะอ่านข้อมูลจาก Pattern ทีละ 4 ตัวอักษร (32 บิต) และเรียกข้อมูล 4 ตัวอักษรนี้ว่า Sub-Pattern ตัวอย่างเช่น Pattern R1 มีข้อมูลเป็น /admin.exe จะสร้างเป็น Sub-pattern ได้ทั้งหมด 7 Sub-Pattern

ได้แก่ /adm, admi, dmin, min., in.e, n.ex, และ .exe จากนั้นข้อมูล Sub-pattern เหล่านี้จะถูกเปลี่ยนชนิดข้อมูลจาก String เป็น Integer เพื่อให้สามารถประมวลผลได้เร็วขึ้น

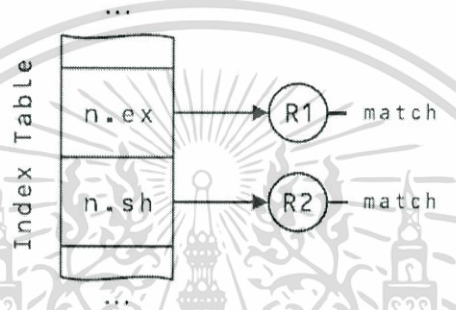
แนวความคิดในการเตรียมโครงสร้างข้อมูลของอัลกอริธึม Piranha นั้นเหมือนกับการสร้างวงจรดิจิตอล โดยแต่ละ Sub-Pattern จะถูกนำไปเป็นขา Input ของ And Gate ดังในรูปที่ 3.1 And Gate แต่ละตัวจะแทน Pattern 1 Pattern การตรวจหา Sub-Pattern ใน Payload ของ Packet หากตรวจพบ Sub-Pattern ใด ขา Input นั้นก็จะถูกตั้งค่าเป็น 1 ในกรณีที่พบทุก Sub-pattern ก็จะทำให้ขา Input ของ And Gate ทุกขาถูกตั้งค่าเป็น 1 Output ที่ได้จาก And Gate ก็จะมีค่าเป็น 1 ด้วย กรณีเช่นนี้สามารถสรุปได้ว่า Packet นั้นมีความเป็นไปได้ที่จะ Match กับ Pattern นั้นๆ

เพื่อให้การตรวจสอบเป็นไปอย่างรวดเร็ว Pattern ทุก Pattern จะถูกเก็บในตาราง Index Table สำหรับ Sub-Pattern ใดที่เป็นส่วนประกอบของ Pattern มากกว่า 1 Pattern จะถูกเก็บลงใน Index Table แค่ Sub-pattern เดียวเท่านั้น ตัวอย่างเช่น Pattern R1 มีข้อมูลเป็น /admin.exe และ Pattern R2 มีข้อมูลเป็น /admin.sh จะพบว่า Sub-pattern /adm, admi, dmin และ min. ปรากฏอยู่ในทั้ง Pattern R1 และ Pattern R2 กรณีนี้ Sub-pattern ดังกล่าวจะถูกเก็บลงใน Index Table เพียงรอบเดียวเท่านั้น ส่วน Sub-pattern in.e, n.ex ที่พบเฉพาะใน Pattern R1 และ Sub-pattern in.s, n.sh ที่พบเฉพาะใน Pattern R2 จะถูกเก็บลง Index Table เช่นเดียวกัน สุดท้ายข้อมูล Sub-Pattern ที่เก็บลง Index Table คือ .exe, /adm, admi, dmin, min., in.e, n.ex, in.s และ n.sh ดังรูปที่ 3.2



รูปที่ 3.2 แสดงตัวอย่างโครงสร้างข้อมูลที่ได้จาก Pattern /admin.exe และ Pattern /admin.sh

อย่างไรก็ตามหากทำการตรวจสอบทุก Sub-pattern จะทำให้การทำ Pattern Matching มีประสิทธิภาพต่ำเนื่องจากต้องมีการเปรียบเทียบหลายครั้งกว่าจะรู้ว่า Pattern นั้น Match หรือไม่ ในการพัฒนาการทำ Pattern Matching ให้มีประสิทธิภาพดีขึ้น จึงมุ่งไปที่การลดจำนวนครั้งที่ต้องเปรียบเทียบ ด้วยการใช้ตัวแทน Sub-Pattern แทนการใช้ Sub-Pattern ทั้งหมด ในการเลือก Sub-Pattern ที่จะเป็นตัวแทนของ Pattern นั้นจะเลือก Sub-Pattern ที่ซ้ำกับ Sub-Pattern ของ Pattern อื่นๆ น้อยที่สุด ผลจากการพัฒนานี้จะทำให้ได้ตาราง Index Table ดังรูปที่ 3.2 จะเห็นได้ว่า Sub-Pattern n.ex ถูกเลือกให้เป็นตัวแทนของ Pattern /admin.exe และ Sub-pattern n.sh ถูกเลือกให้เป็นตัวแทนของ Pattern /admin.sh ดังนั้น แต่ละ Pattern จะเหลือเพียง 1 Input เท่านั้น



รูปที่ 3.3 แสดงโครงสร้างข้อมูลหลังจากการพัฒนาแล้ว

3.3.2 Matching

ขั้นตอนการ Matching เป็นการค้นหา Pattern ใน Payload ของ Packet โดยใช้โครงสร้างข้อมูลที่ได้จากขั้นตอน Preprocessing เมื่อมี Payload ของ Packet เข้ามา อัลกอริธึม Piranha จะดึงข้อมูลทีละ 4 ตัวอักษรจาก Payload ของ Packet นั้นมาตรวจสอบกับ Index Table หากพบว่า ข้อมูล 4 ตัวอักษรนั้นตรงกับ Sub-Pattern ใด ก็จะส่ง Payload ของ Packet นั้นไปตรวจสอบข้อมูล 2 อักษรสุดท้าย กับ 2 อักษรสุดท้ายของ Pattern นั้นๆ หากพบว่าตรงกัน ก็จะสรุปผลว่า Payload ของ Packet นั้น Match กับ Pattern ทันที แต่ถ้าไม่ตรงกันก็สรุปได้เช่นกันว่า Payload ของ Packet นั้นไม่ Match เช่น หากมี Payload เข้ามาเป็น /login.sh ข้อมูลที่จะถูกนำไปตรวจสอบคือ /log, logi, ogin, gin., in.s และ n.sh จากข้อมูลในรูปที่ 3.3 จะพบว่า n.sh ตรงกับ Sub-pattern ที่อยู่ใน Index Table ดังนั้น Payload /login.sh จะถูกส่งไปตรวจสอบ 2 อักษรสุดท้าย หากพบว่าตรงกับ Pattern ก็จะถูกส่งไปตรวจสอบต่อไป

3.4 อัลกอริธึม Cuckoo Hashing

Cuckoo Hashing เป็นอัลกอริธึม Hashing ที่ใช้ Hash Table จำนวน 2 Table (T_1 และ T_2) แต่ละ Table มีขนาด r ช่อง ใช้ Hash Function จำนวน 2 Function โดย Hash Function 1 (H_1) ใช้คู่กับ Table T_1 ส่วน Hash Function 2 (H_2) จะใช้คู่กับ Table T_2

ให้ x เป็นข้อมูลที่เก็บลง Hash Table การ Lookup จะใช้ Function ดังนี้คือ

```
function lookup(x)
    return T1[H1(x)] = x OR T2[H2(x)] = x;
end
```

สำหรับขั้นตอนการ Insert ข้อมูล จะมีขั้นตอนการทำงานดังนี้คือ

```
procedure insert(x)
    if lookup(x) then return;
    loop MaxLoop times
      x ↔ T1[H1(x)];
      if x = NULL then return;
      x ↔ T2[H2(x)];
      if x = NULL then return;
    end loop
    rehash();
    insert(x);
end
```

ในการ Insert ข้อมูล จะต้องทำการตรวจสอบก่อนว่า มีข้อมูลนั้นใน Table หรือไม่ หากมีข้อมูลแล้วให้หยุดการ Insert แต่ถ้ายังไม่มีก็ดำเนินการต่อไปคือ นำค่าในตัวแปร x ไป Hash ด้วย Function H_1 ได้ค่าเป็น $H_1(x)$ จากนั้น สลับค่าที่อยู่ Table T_1 ตำแหน่ง $H_1(x)$ กับค่าในตัวแปร x จากนั้นทำการตรวจสอบว่า ค่าในตัวแปร x เป็นค่าว่าง (NULL) หรือไม่ หากเป็นค่าว่างแสดงว่าตำแหน่งข้อมูลนั้นว่าง สามารถเก็บข้อมูลได้เลย ขั้นตอนการ Insert ข้อมูลจะเสร็จสิ้นทันที แต่หากค่าในตัวแปร x ไม่ได้เป็นค่าว่าง แสดงว่าเกิดการ Collision ข้อมูลเดิมที่เคยอยู่ ณ ตำแหน่ง $H_1(x)$ ใน Table T_1 ก็จะถูกนำไปหาตำแหน่งในการเก็บข้อมูลใหม่ แต่เปลี่ยนมาใช้ Table T_2 และหาตำแหน่งด้วย Function H_2 ซึ่งจะได้ตำแหน่งใหม่เป็น $H_2(x)$ หลังจากนั้น สลับค่าที่อยู่ Table T_2 ตำแหน่ง

$H_2(x)$ กับค่าในตัวแปร x แล้วทำการตรวจสอบว่า ค่าในตัวแปร x เป็นค่าว่างหรือไม่ หากเป็นค่าว่าง แสดงว่าตำแหน่งข้อมูลนั้นว่าง สามารถเก็บข้อมูลได้เลย ขั้นตอนการ Insert ข้อมูลจะเสร็จสิ้นทันที แต่หากค่าในตัวแปร x ไม่ได้เป็นค่าว่าง จะเริ่มต้นกระบวน Insert ข้อมูลใหม่ โดยนำค่าที่อยู่ในตัวแปร x นี้กลับไปหาตำแหน่งที่จะเก็บลง Table T_1 ด้วย Function H_1 และทำเช่นเดิมต่อไปเรื่อยๆ จนถึงค่าที่กำหนด (MaxLoop) หากยังไม่สามารถเสร็จสิ้นการ Insert ข้อมูล ก็จะทำการ Hash ข้อมูลใหม่ ตั้งแต่ต้น (Rehash) และทำการเปลี่ยน Function Hash ทั้ง H_1 และ H_2 เพื่อให้ตำแหน่งของข้อมูลเปลี่ยนไป และสามารถเก็บข้อมูลทั้งหมดลง Table T_1 และ Table T_2 ได้ในที่สุด

(ก) (ข)

รูปที่ 3.4 แสดงตัวอย่างการ Insert ข้อมูล ลูกศรในภาพแทนการเปลี่ยนตำแหน่งของข้อมูล
(ก) กรณีข้อมูล x สามารถเก็บลง Table T_1 ได้
(ข) กรณีข้อมูล x สามารถเก็บลงทั้ง 2 Table ได้

จากรูปที่ 3.4 เป็นตัวอย่างการ Insert ข้อมูล x รูปที่ 3.4 (ก) เป็นกรณีที่สามารถ Insert ข้อมูลได้ กระบวนการจะเริ่มจาก เก็บค่า x ลง T_1 เกิด Collision ค่า y ถูกย้ายไปเก็บใน T_2 และเกิด Collision อีกครั้ง ค่า z ถูกย้ายไปเก็บใน T_1 เนื่องจากไม่เกิด Collision อีก การ Insert ข้อมูล x จึงเสร็จสมบูรณ์ ส่วนรูปที่ 3.4 (ข) เป็นกรณีที่ไม่สามารถ Insert ข้อมูลได้ ทำให้ต้องทำการ Rehash ใหม่ทั้งหมด กระบวนเริ่มจาก เก็บค่า x ลง T_1 เกิด Collision ค่า y ถูกย้ายไปเก็บใน T_2 เกิด Collision ค่า z ถูกย้ายไปเก็บใน T_1 เกิด Collision ค่า u ถูกย้ายไปเก็บใน T_2 เกิด Collision ค่า v ย้ายไปเก็บใน T_1 เกิด Collision สุดท้ายค่า x วนกลับมาเหมือนเดิม หากพบว่าไม่สามารถเก็บข้อมูลโดยเริ่มจาก Table T_1 ได้ ก็จะเปลี่ยนไปเก็บข้อมูลลง Table T_2 และเปลี่ยนมาใช้ Function H_2 จากนั้นกระบวนการเป็นดังนี้คือ เก็บค่า x ลง T_2 เกิด Collision ค่า t ถูกย้ายไปเก็บใน T_1 เกิด Collision ค่า s ถูกย้ายมาเก็บใน T_2 ซึ่งเป็นตำแหน่งเดิม สุดท้ายค่า x วนกลับมาเหมือนเดิมเช่นเดียวกัน ไม่สามารถเก็บข้อมูลลง Table ได้ หากเป็นเช่นนี้จะเกิดการ Rehash โดยเปลี่ยน Function Hash ทั้ง 2 Function เพื่อให้ตำแหน่งในการเก็บข้อมูลเปลี่ยนไป และเริ่มกระบวนการ Inster ข้อมูลลง Table ใหม่ทั้งหมด

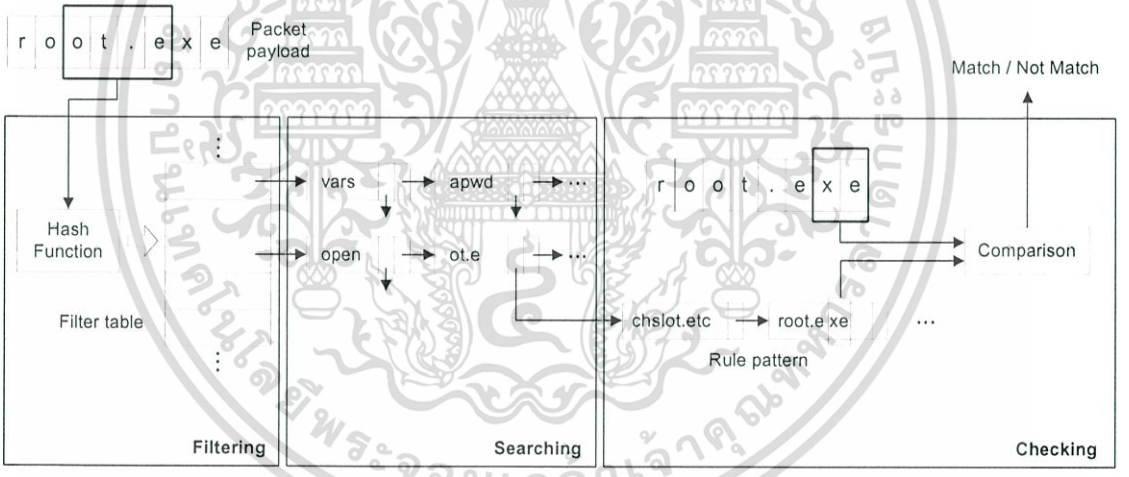
บทที่ 4

การเพิ่มประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุกด้วยวิธีการ Cuckoo Hashing

งานวิจัยนี้จะนำเสนอ การเพิ่มประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุก โดยพัฒนาบนอัลกอริธึม Piranha ซึ่งเป็นอัลกอริธึมที่มีประสิทธิภาพดีทั้งในแง่ของปริมาณหน่วยความจำที่ใช้และการใช้งาน CPU ในการพัฒนานั้นจะนำข้อดีของ Cuckoo Hashing มาใช้ช่วยในขั้นตอนการ Searching

ในบทนี้จะกล่าวถึงขั้นตอนการทำงานทั้งหมดในส่วนของการทำ Pattern Matching ของอัลกอริธึม Piranha+Cuckoo ที่นำเสนอเปรียบเทียบกับอัลกอริธึม Piranha แบบเดิม

4.1 โครงสร้างการทำ Pattern Matching ของอัลกอริธึม Piranha



รูปที่ 4.1 แสดงขั้นตอนการทำ Pattern Matching ของอัลกอริธึม Piranha

จากรูปที่ 4.1 กลไกการทำ Pattern Matching ของอัลกอริธึม Piranha จะเริ่มขึ้นเมื่อมี Payload ของ Packet เข้ามา Payload ของ Packet ดังกล่าวจะถูกนำมาตรวจสอบทีละ 4 ตัวอักษร (4 bytes sequence) และผ่านขั้นตอนการทำ Pattern Matching 3 ขั้นตอนคือ

1) Filtering

ในขั้นตอนนี้จะเป็นการตรวจสอบความเป็นไปได้ของการที่จะมีรูปแบบตรงกับ Pattern นั่นคือ ข้อมูล 4 ตัวอักษรจะถูกนำไป Hash ด้วยฟังก์ชัน Mod (Modulo Function) จากนั้นจะนำค่า Hash ดังกล่าวไปตรวจสอบใน Filter Table ซึ่ง Filter Table นี้ประกอบไปด้วย ข้อมูล 4 ตัวอักษรที่เป็นตัวแทนของแต่ละ Pattern (Sub-Pattern) และซ้ำกันน้อยที่สุด หากพบว่าเป็นช่องว่าง แสดงว่าข้อมูล 4 ตัวอักษรนั้น ไม่ตรงกับ Pattern ใดๆ ก็จะทำการตรวจสอบข้อมูล 4 ตัวอักษรที่อยู่ถัดไปทันที แต่หากพบว่าไม่เป็นช่องว่าง นั่นคือ มีการเชื่อมโยงไปยังโหนดของ Sub-Pattern แสดงว่ามีโอกาสที่จะตรงกับ Pattern นั้นๆ ดังนั้น ข้อมูล 4 ตัวอักษรนี้จะถูกส่งไปทำการตรวจหา Pattern ในขั้นตอน Searching ต่อไป

2) Searching

หลังจากผ่านขั้นตอน Filtering มาแล้ว ข้อมูล 4 ตัวอักษรที่มีความเป็นไปได้ที่จะ Match กับ Pattern จะถูกนำมาตรวจสอบในขั้นตอนนี้ โดยลักษณะโครงสร้างข้อมูลของ Sub-Pattern จะเป็น Link List ที่เชื่อมโยงต่อไปเรื่อยๆ ดังนั้น ข้อมูล 4 ตัวอักษรจะถูกนำมาเปรียบเทียบกับ Sub-Pattern ทีละโหนดจนกระทั่งสุด Link List

หากพบว่าข้อมูล 4 ตัวอักษรที่นำมาตรวจสอบไม่ตรงกับ Sub-Pattern ใดเลย ก็จะเริ่มขั้นตอนใหม่ตั้งแต่ต้นและเลื่อนไปตรวจสอบข้อมูลของ Packet 4 ตัวอักษรถัดไป แต่หากพบว่าข้อมูล 4 ตัวอักษรตรงกับ Sub-Pattern ใด แสดงว่า มีความเป็นไปได้ที่ข้อมูลของ Packet จะ Match กับ Pattern ที่มี Sub-Pattern นั้นเป็นตัวแทน ข้อมูลของ Packet ดังกล่าวจะถูกส่งไปตรวจสอบในขั้นตอน Checking ต่อไป

3) Checking

ขั้นตอนนี้จะเป็นการตรวจสอบ 2 อักษรของข้อมูลที่อยู่ใน Packet กับ 2 ตัวอักษรสุดท้ายของ Pattern ที่มีตัวแทนเป็น Sub-Pattern เดียวกัน ในการตรวจสอบจะใช้ตำแหน่งของ Sub-Pattern ที่พบในข้อมูล Packet เพื่อเป็นตำแหน่งอ้างอิงไปยัง 2 ตัวอักษรสุดท้ายที่ตรงกัน จากนั้นจะทำการตรวจสอบกับ Pattern ทีละ Pattern จนครบ หากพบว่า 2 ตัวอักษรสุดท้ายตรงกันก็จะสรุปได้ทันทีว่า Match แต่หากพบว่า 2 ตัวอักษรสุดท้าย ไม่ตรงกันก็สรุปได้ทันทีเช่นกันว่า ไม่ Match

จากโครงสร้างการทำ Pattern Matching ของอัลกอริทึม Piranha ดังรูปที่ 4.1 สามารถวิเคราะห์ปัญหาได้ 2 ประการคือ

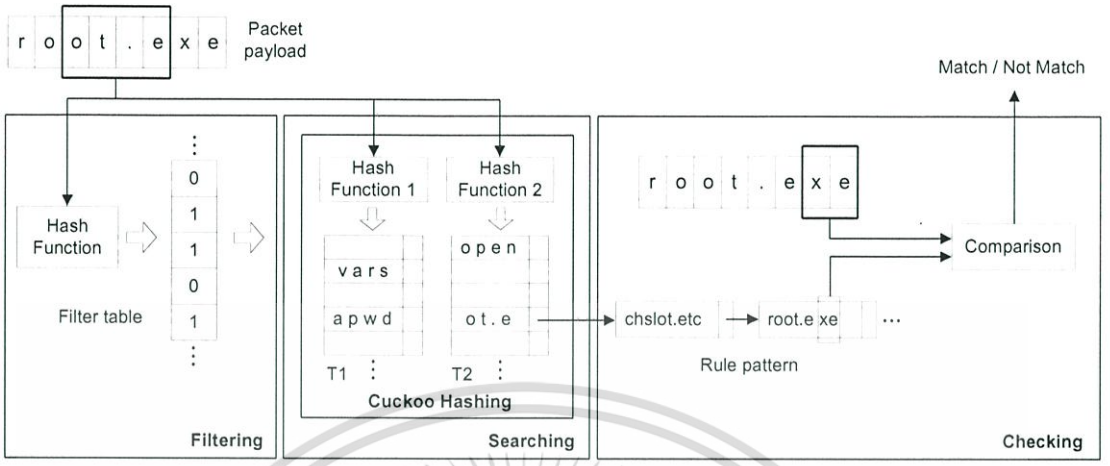
1. ปัญหาจำนวนโหนด Sub-Pattern ในแต่ละ Chain ของ Filter Table มีความยาวที่ไม่สามารถคาดเดาได้

เนื่องจากอัลกอริธึม Piranha ใช้โครงสร้างข้อมูล Hash ในการเก็บข้อมูลของ Sub Pattern และแก้ปัญหา Collision ด้วยวิธีการ Separate Chaining ประกอบกับการใช้ Modulo Function เป็น Hash Function ทำให้การกระจายตัวของข้อมูลนั้นไม่ดีพอ ส่งผลให้ไม่สามารถคาดเดาจำนวนโหนด Sub-Pattern ของแต่ละ Chain ได้ จากภาคผนวก ข จะเห็นได้ว่า Pattern Group 17 มีจำนวน Pattern ทั้งหมด 818 Pattern เมื่อนำมาทำ Filter Table จะเกิด Chain ที่มีความยาวถึง 25 โหนด และโดยเฉลี่ย Pattern Group ต่างๆ จะมีความยาวของ Chain อยู่ที่ 12 โหนด

2. ปัญหา Time Complexity ของการ Search

สืบเนื่องจากปัญหาในข้อ 1 โครงสร้างข้อมูลสำหรับการ Search มีลักษณะเป็น Link List ที่ไม่อาจคาดเดาความยาวได้ ประกอบกับลักษณะของ Packet ที่โดยส่วนใหญ่แล้ว มักจะไม่ Match กับ Pattern ใดๆ เนื่องจาก Traffic ส่วนใหญ่ไม่ได้เป็นTraffic ที่เป็นการละเมิดความปลอดภัย ในการ Search แต่ละครั้งจึงมักจะต้องทำการ Search จนถึงโหนดสุดท้ายของ Link List นั่นคือหากพิจารณากรณี Unsuccessful Search จะพบว่า มีค่า Time Complexity เป็น O(n) เมื่อ n คือ ความยาวของ Chain

4.2 การทำ Pattern Matching ของอัลกอริธึม Piranha+Cuckoo



รูปที่ 4.2 แสดงขั้นตอนการทำ Pattern Matching ของอัลกอริธึม Piranha + Cuckoo

ในโครงสร้างโดยรวม การทำ Pattern Matching ของอัลกอริธึม Piranha+Cuckoo ยังคงอยู่บนพื้นฐานของอัลกอริธึม Piranha ส่วนที่ได้ทำการพัฒนาโดยใช้วิธีการใหม่คือส่วนของ Filtering และ Searching มีรายละเอียดดังนี้คือ

1) Filtering

ในส่วนของ Filtering ได้ปรับเปลี่ยน Filter Table จาก Array of Pointer มาเป็น Array of Integer ดังรูปที่ 4.2 ในขั้นตอน Filtering ซึ่งทำให้การเปรียบเทียบสามารถทำได้เร็วขึ้น

2) Searching

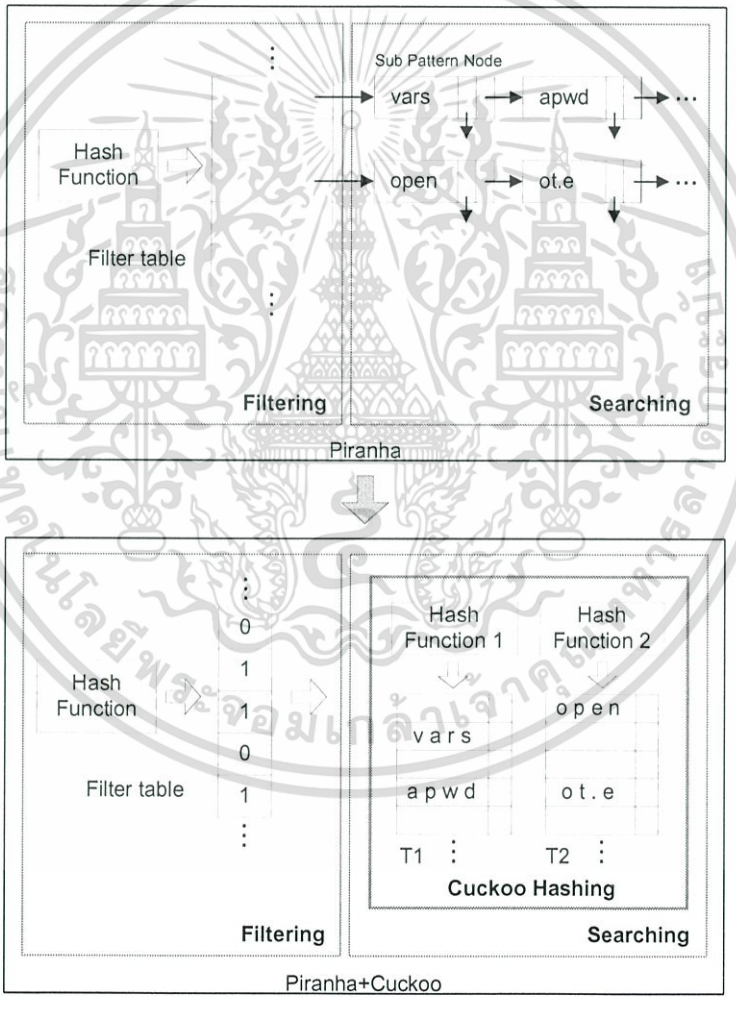
จากข้อเสียของอัลกอริธึม Piranha ดังที่กล่าวในหัวข้อ 4.1 สามารถแก้ปัญหาได้โดยนำแนวความคิดอัลกอริธึม Cuckoo Hashing มาประยุกต์ใช้ สอดคล้องกับงานวิจัยของ Tran NGoc Thinh [9] ที่ได้นำอัลกอริธึม Cuckoo Hashing ไปประยุกต์ใช้กับการทำ Pattern Matching บน Hardware (FPGA-BASED)

การพัฒนาอัลกอริธึมการทำ Pattern Matching จะพัฒนาโดยเปลี่ยนโครงสร้างข้อมูลของ Sub-Pattern ซึ่งเดิมเป็นโครงสร้างข้อมูลแบบ Link List มาเป็นการใช้โครงสร้างข้อมูลแบบ Cuckoo Hashing โดยใช้ Hash Table จำนวน 2 Table และ Hash Function จำนวน 2 Function ดังรูปที่ 4.2 ในขั้นตอน Searching ทั้งนี้ Hash Function จะต้องมีคุณสมบัติของการกระจายตัวของข้อมูลที่ดีและสามารถปรับเปลี่ยน Function ได้ในกรณีที่เกิด Collision เกินขอบเขตที่กำหนด

การพัฒนาดังกล่าวจะสามารถแก้ปัญหาของอัลกอริธึม Piranha ได้ทั้ง 2 ประการ คือ

1. **ปัญหาจำนวนโหนด Sub-Pattern ในแต่ละ Chain ของ Filter Table มีความยาวที่ไม่สามารถคาดเดาได้**

จากการที่ใช้โครงสร้างข้อมูลแบบ Cuckoo Hashing แทนการเก็บข้อมูลแบบ Link List ทำให้สามารถแก้ปัญหาความยาวของ Chain ได้เนื่องจากเป็นการเก็บ Sub-Pattern ลงตารางแทนการใช้โครงสร้างข้อมูลแบบ Link List ทั้งนี้การเก็บข้อมูลลงตารางจะใช้ Hash Table จำนวน 2 Table และ Hash Function จำนวน 2 Function ดังรูปที่ 4.3 ด้วยวิธีการนี้ทำให้สามารถมั่นใจได้ว่า ทุก Sub-Pattern จะสามารถเก็บลง Hash Table ใด Table หนึ่งได้ไม่ว่า Sub-Pattern จะมีจำนวนเท่าใดก็ตาม



รูปที่ 4.3 แสดงการเปลี่ยนโครงสร้างข้อมูลจากโครงสร้างข้อมูลแบบ Link List มาเป็นโครงสร้างข้อมูลแบบ Cuckoo Hashing

2. ปัญหา Time Complexity ของการ Search

ในการ Search เพื่อหา Pattern ที่ตรงกัน ค่า Time Complexity ของการ Search จะมีค่าเพียง O(2) (ค่าคงที่) ซึ่งเป็นข้อดีของอัลกอริธึม Cuckoo Hashing โดยในการค้นหาข้อมูลจะ Search อย่างมากที่สุดเพียง 2 ครั้งเท่านั้น

บทที่ 5

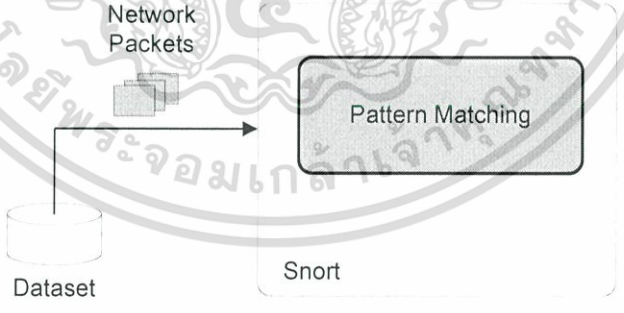
การทดลองและผลการทดลอง

5.1 บทนำ

ในบทนี้จะกล่าวถึงการวิเคราะห์ Hashing Function ที่ใช้งานสำหรับอัลกอริธึม Cuckoo Hashing การทดลองวัดจำนวนครั้งของการ Rehash การทดลองวัดเวลาที่ใช้ในการจัดเตรียมโครงสร้างข้อมูลของโปรแกรม Snort เมื่อใช้อัลกอริธึมต่างๆ การทดลองตรวจสอบความถูกต้องในการทำงานของอัลกอริธึม Piranha+Cuckoo การทดลองเปรียบเทียบจำนวนครั้งการ Search ของแต่ละอัลกอริธึม การทดลองวัดประสิทธิภาพของอัลกอริธึม Piranha+Cuckoo เปรียบเทียบกับอัลกอริธึม Piranha ในแง่ของเวลาที่ใช้ในการประมวลผล (CPU Time) ตลอดจนการทดลองวัดเวลาที่ใช้ในการประมวลผล (CPU Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม

5.2 โครงสร้างการทดลอง

ในการทดลองจะเก็บ Dataset ที่จะนำมาใช้ในการทดสอบทั้งหมดในฮาร์ดดิสก์ ปิด Preprocessor ทั้งหมดของโปรแกรม Snort จากนั้นทำการทดลองโดยใช้คำสั่ง snort –r filename โดยที่ filename คือไฟล์ของ Dataset ที่อยู่ในรูปของ TCPDump ดังนั้นโปรแกรม Snort จะอ่าน Dataset จากฮาร์ดดิสก์โดยตรง



รูปที่ 5.1 แสดงโครงสร้างการทดลอง

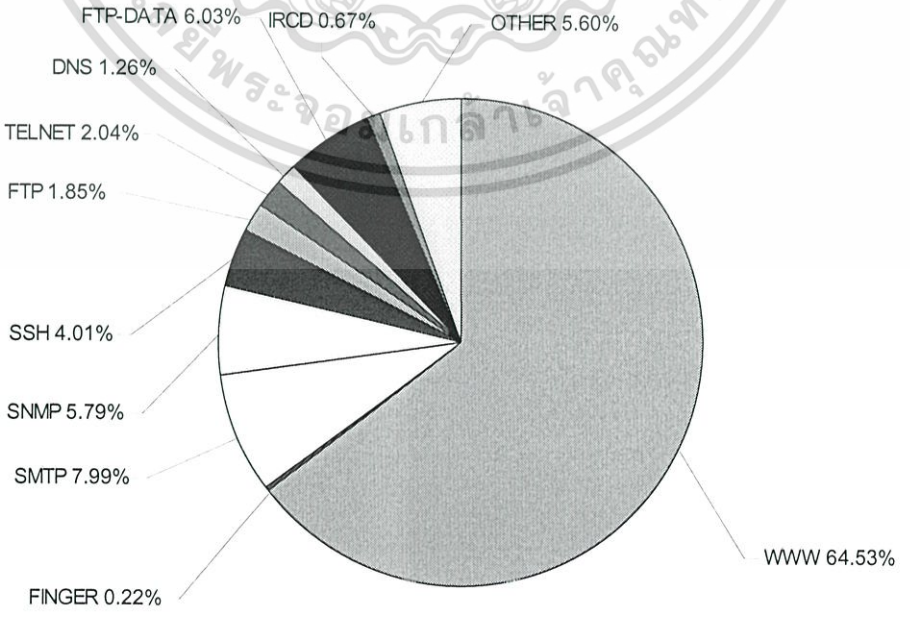
อุปกรณ์และโปรแกรมที่ใช้ในการทดลอง

1. CPU Pentium III 500 MHz
2. RAM 512 MB
3. OS Debian 3.1r5
4. IDS : Snort 2.4.2
5. Rule set : 7344 rules

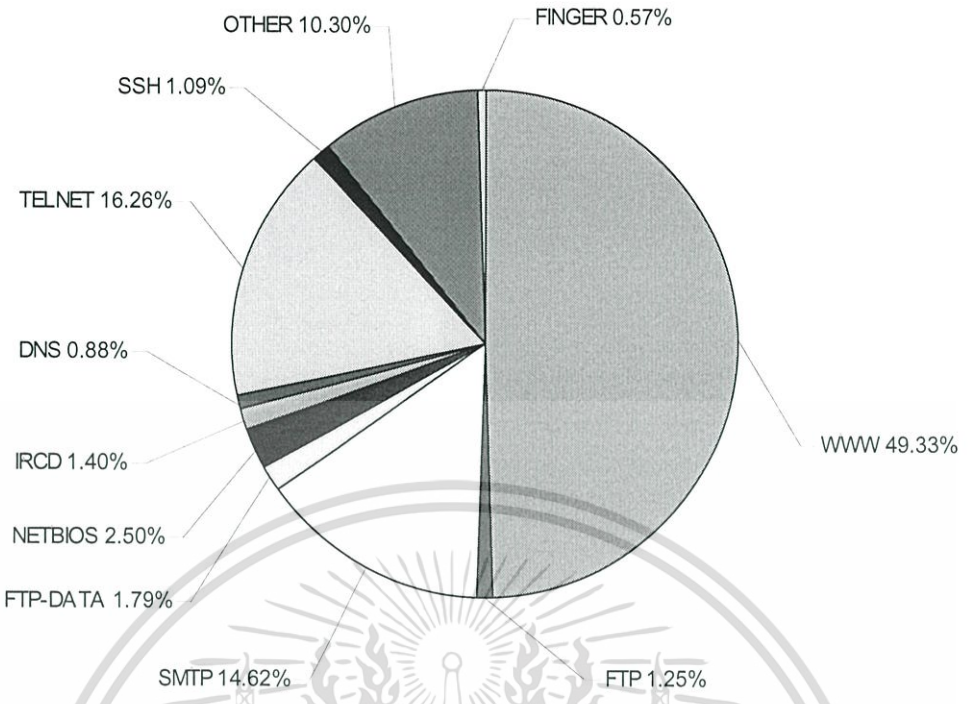
5.3 Dataset ที่ใช้ในการทดลอง

Dataset ที่ใช้ในการทดลองนี้เป็นไฟล์ที่เก็บตามรูปแบบของ TCPDump แบ่งเป็น 2 กลุ่มคือ

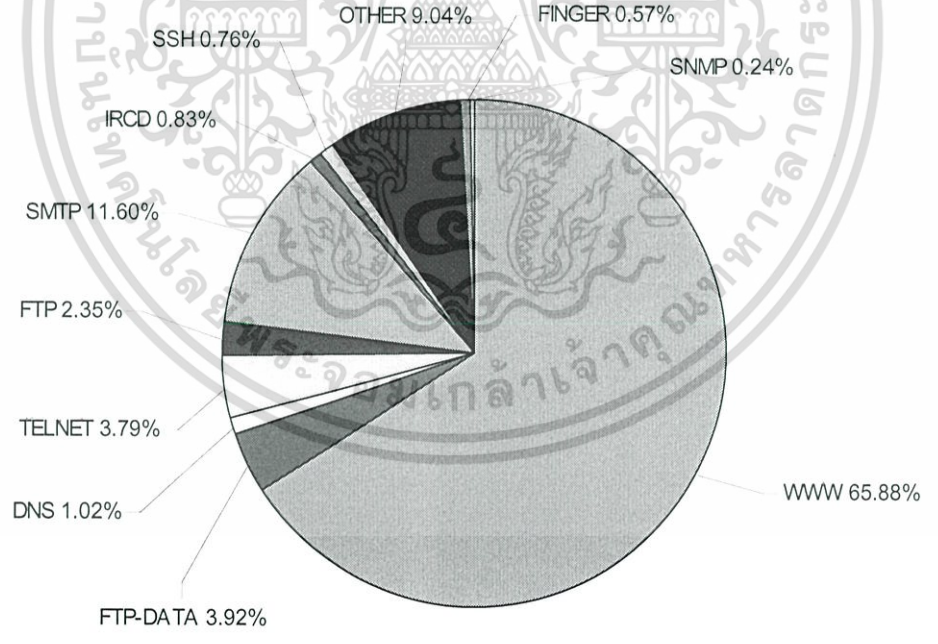
1) **Darpa** [10] เป็น Packet ที่เก็บบันทึกระหว่างการทดสอบ Off-line Intrusion Detection evaluation and a realtime evaluation ที่ Lincoln LAB สถาบัน MIT ปี พ.ศ.2533 แบ่งเป็น
   1.1) **Darpa01** เป็น Packet ที่เก็บบันทึกในวันพุธที่ 3 มีนาคม 2533 สัปดาห์แรกของการทดสอบ ประกอบด้วย Traffic ธรรมดาทั่วๆ ไป และ Traffic ที่เป็นการละเมิดความปลอดภัย จำนวนทั้งสิ้น 1,660,261 Packet
   1.2) **Darpa02** เป็น Packet ที่เก็บบันทึกในวันพุธที่ 10 มีนาคม 2533 สัปดาห์ที่สองของการทดสอบ ประกอบด้วย Traffic ธรรมดาทั่วๆ ไป ไม่มี Traffic ที่เป็นการละเมิดความปลอดภัย จำนวนทั้งสิ้น 901,884 Packet
   1.3) **Darpa03** เป็น Packet ที่เก็บบันทึกในวันพุธที่ 31 มีนาคม 2533 สัปดาห์ที่สี่ของการทดสอบ ประกอบด้วย Traffic ธรรมดาทั่วๆ ไป และ Traffic ที่เป็นการละเมิดความปลอดภัย จำนวนทั้งสิ้น 1,357,320 Packet
2) **Defcon** [11] เป็น Packet ที่เก็บบันทึกระหว่างการแข่งขัน Capture The Flag Contest ครั้งที่ 9 ปี พ.ศ.2544 แบ่งเป็น
   2.1) **Defcon9.1** ประกอบด้วย Traffic ธรรมดาทั่วๆ ไป และ Traffic ที่เป็นการละเมิดความปลอดภัย จำนวนทั้งสิ้น 2,566,219 Packet
   2.2) **Defcon9.2** ประกอบด้วย Traffic ธรรมดาทั่วๆ ไป และ Traffic ที่เป็นการละเมิดความปลอดภัย จำนวนทั้งสิ้น 1,050,363 Packet



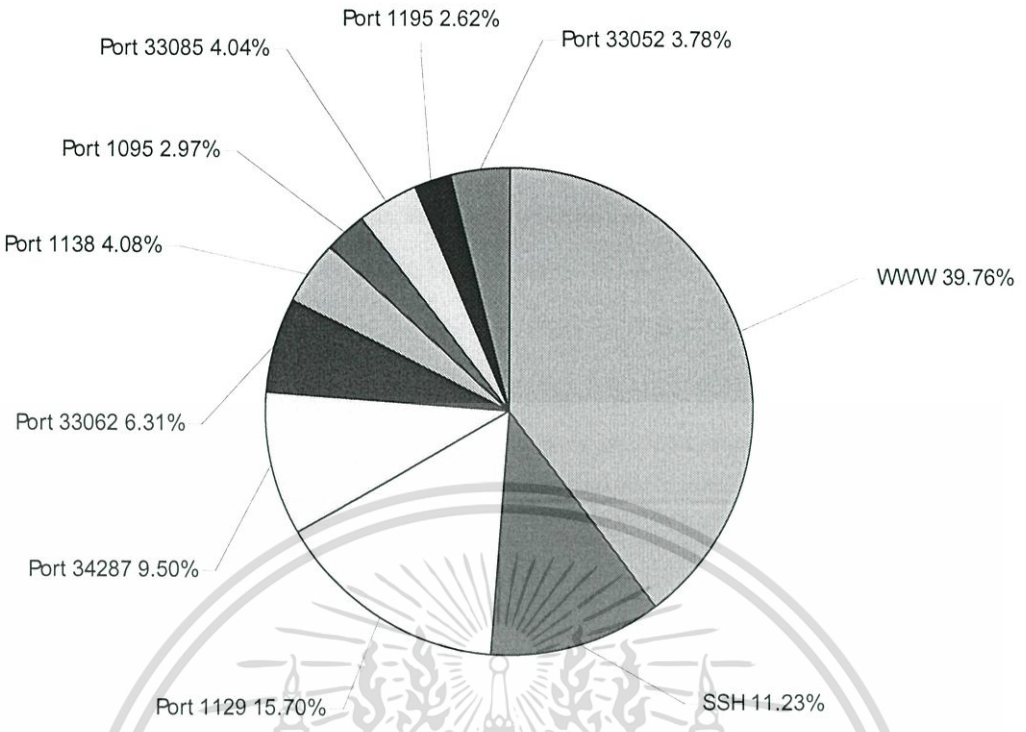
รูปที่ 5.2 แสดง Protocol Breakdown ของ Packet Darpa01



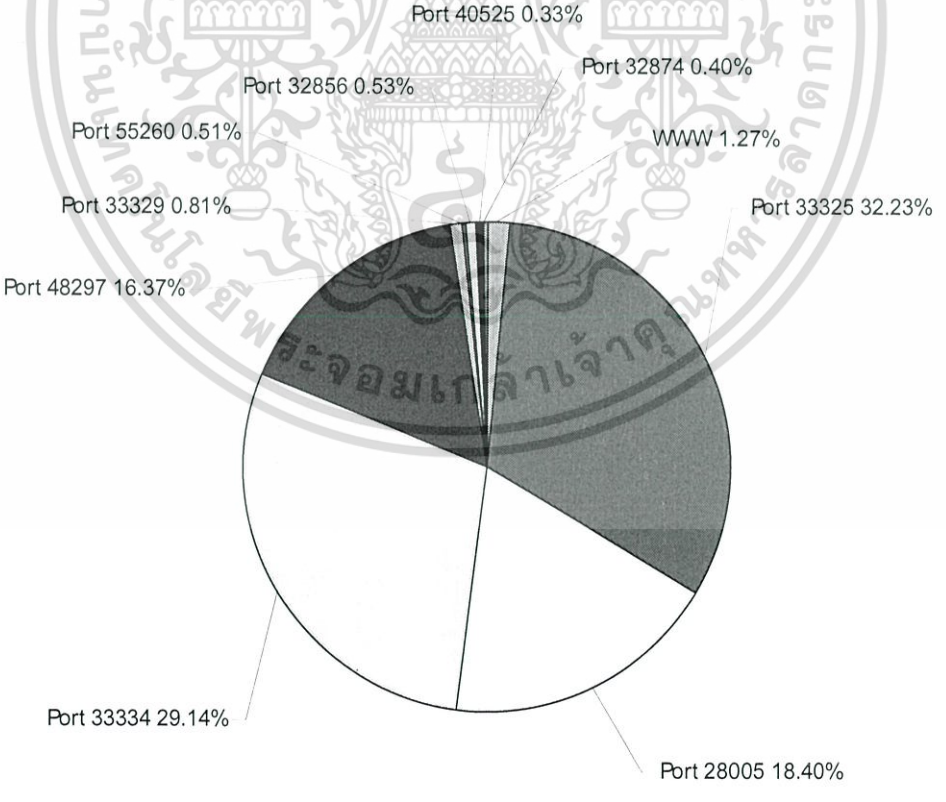
รูปที่ 5.3 แสดง Protocol Breakdown ของ Packet Darpa02



รูปที่ 5.4 แสดง Protocol Breakdown ของ Packet Darpa03



รูปที่ 5.5 แสดง Protocol Breakdown ของ Packet Defcon9.1



รูปที่ 5.6 แสดง Protocol Breakdown ของ Packet Defcon9.2

จากรูปที่ 5.7 Input ของ Hashing Function คือข้อมูลขนาด 32 บิต ในที่นี้คือ Sub-Pattern AFE0400C ฐาน 16 หรือ 2950709260 ฐาน 10 ขั้นตอนการทำงานจะเริ่มจาก การนำค่าบิตของข้อมูล (8 บิต) 3 ชุดจาก Input (Sub-Pattern) ไปเป็นตำแหน่งสำหรับ Lookup ค่าที่อยู่ใน Array A_1 โดยค่าที่อยู่ใน Array A_1 และ A_2 เป็นค่าที่เกิดจากการสุ่ม จากนั้นจะนำค่าที่ได้จาก Array A_1 ทั้ง 3 ค่ามาหาผลบวกแล้วนำผลบวกที่ได้ไป Xor (Exclusive Or) กับ Sub-Pattern สุดท้ายนำค่าที่ได้ไป Mod ด้วยขนาดของตาราง T_1 ผลลัพธ์จาก Hashing Function ในตัวอย่างคือ 1 ดังนั้น Sub-Pattern $AFE0400C_{16}$ จะถูกเก็บอยู่ในช่องที่ 1 ของตาราง T_1

จากขั้นตอนการทำงานของ Hashing Function จะเห็นได้ว่า มีหลายวิธีการในการเลือกข้อมูล 8 บิตจาก Sub-Pattern เพื่อนำไป Lookup ค่าที่อยู่ใน Array A_1 โดยถ้าให้ n_1 คือตำแหน่งของบิตข้อมูลตัวแรกของข้อมูลบล็อกที่ 1, n_2 คือตำแหน่งของบิตข้อมูลตัวแรกของข้อมูลบล็อกที่ 2 และ n_3 คือตำแหน่งของบิตข้อมูลตัวแรกของข้อมูลบล็อกที่ 3 ตัวอย่างดังรูปที่ 5.7 จะมีค่า n_1-n_2-n_3 เป็น 20-12-4 เมื่อพิจารณาค่า n_1-n_2-n_3 ที่เป็นไปได้ทั้งหมดจะพบว่ามีทั้งสิ้น 7 แบบ คือ 17-9-1, 18-10-2, 19-11-3, 20-12-4, 21-13-5, 22-14-6 และ 23-15-7

ในการเลือกค่า n_1-n_2-n_3 มาใช้งานใน Hashing Function มีความจำเป็นที่จะต้องเลือกค่าที่ทำให้เกิดการ Collision ของ Hashing Function น้อยที่สุดดังนั้นจึงได้ทำการวิเคราะห์หาค่าการกระจายของบิต 1 ในแต่ละตำแหน่งของแต่ละ Sub- Pattern โดยในการวิเคราะห์จะนำข้อมูล Sub-Pattern จาก Pattern Group ที่มีจำนวน Sub-Pattern มากที่สุด 10 อันดับแรก ดังตารางที่ 5.1 มาทำการวิเคราะห์ข้อมูล

ตารางที่ 5.1 แสดง Pattern Group ที่มีจำนวน Pattern มากที่สุด 10 อันดับแรก

ลำดับที่	Pattern Group	จำนวน Pattern
1	19	921
2	17	818
3	18	472
4	115	465
5	31	333
6	11	326
7	5	324
8	112	315
9	44	299
10	9	298

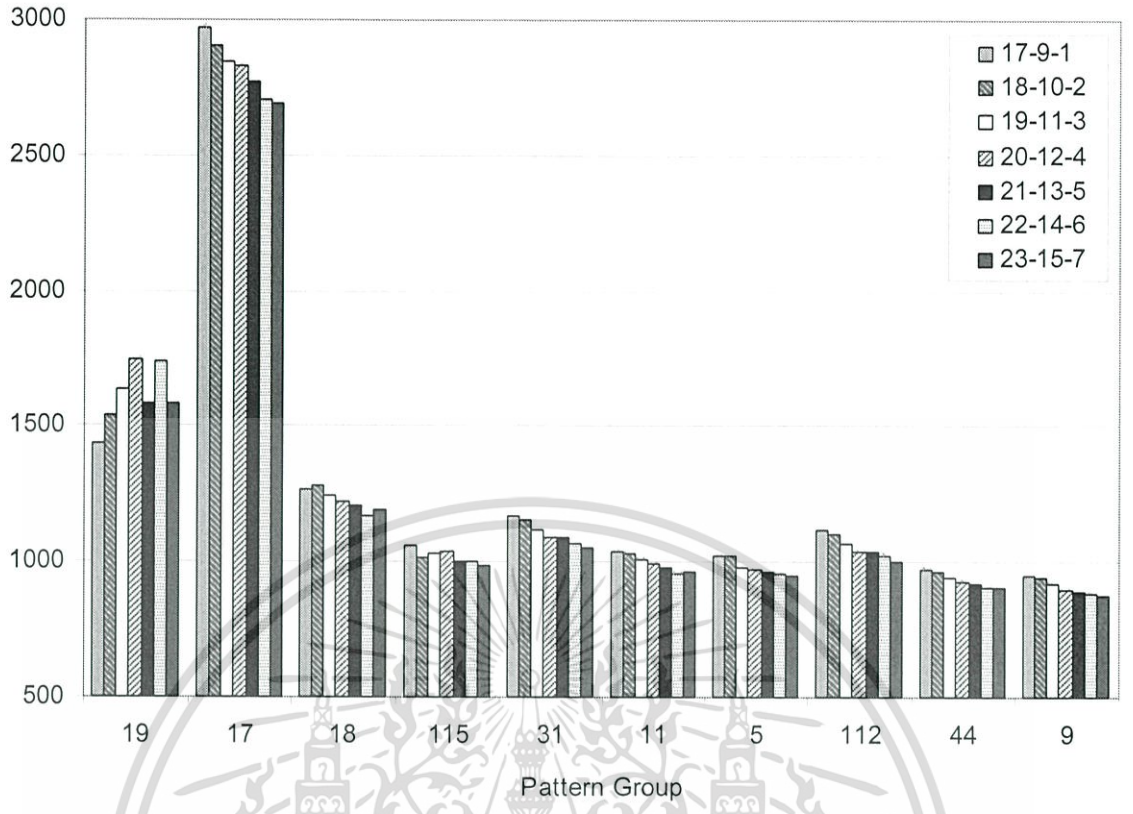
ในการวิเคราะห์หาการกระจายของบิต 1 ใน Sub-Pattern มีขั้นตอนการคำนวนดังนี้คือ

1) คำนวณหาจำนวนบิต 1 ที่ปรากฏในทุก Sub-Pattern โดยแยกหาผลรวมตามตำแหน่งของบิตข้อมูล เช่น หาผลรวมของบิต 1 ของบิตตำแหน่งที่ 31 จากทุก Sub-Pattern
2) คำนวณหาค่ากลางของ Pattern Group นั้น โดยค่านี้คำนวณโดยนำค่าจำนวน Pattern ทั้งหมดของ Pattern Group นั้นไปหารด้วย 2 เนื่องจากหากบิตข้อมูลมีการจายตัวที่ดี ควรจะมีจำนวนของบิต 1 และบิต 0 อย่างละครึ่ง
3) นำผลรวมของบิต 1 ที่ได้แต่ละตำแหน่งข้อมูลไปลบออกจากค่ากลาง จะได้ค่าเบี่ยงเบนที่บอกถึงระยะห่างจากค่ากลาง นั่นคือหากผลลัพธ์มีค่าเข้าใกล้ 0 แสดงว่า มีการกระจายของบิต 1 เกือบครึ่งหนึ่งของจำนวนบิตทั้งหมดของข้อมูลตำแหน่งนั้น
4) หาผลรวมของค่าเบี่ยงเบน โดยคิดผลรวมเฉพาะบิตที่ใช้งาน เช่น สำหรับชุด 17-9-1 ให้หาผลรวมของค่าเบี่ยงเบนของข้อมูลบิตที่ 24 ถึงบิตที่ 1

ตารางที่ 5.2 แสดงผลรวมค่าเบี่ยงเบนแยกตาม Pattern Group

n_1-n_2-n_3	ผลรวมค่าเบี่ยงเบน แยกตาม Pattern Group									
	19	**17**	**18**	**115**	**31**	**11**	**5**	**112**	**44**	**9**
17-9-1	1436	2972	1266	1062	1172	1036	1020	1118	975	950
18-10-2	1539	2905	1281	1019	1153	1030	1020	1102	967	944
19-11-3	1634	2845	1245	1033	1119	1007	980	1065	944	920
20-12-4	1749	2828	1226	1035	1092	991	972	1042	925	897
21-13-5	1585	2771	1210	1005	1087	980	966	1037	922	893
22-14-6	1736	2704	1172	999	1070	959	956	1027	909	883
23-15-7	1581	2691	1193	987	1051	963	949	999	909	878

จากผลการคำนวณหาผลรวมของค่าเบี่ยงเบนแสดงได้ดังตารางที่ 5.2 และเมื่อนำไปทำเป็นแผนภูมิแท่งจะได้ดังรูปที่ 5.8



รูปที่ 5.8 แผนภูมิแท่งแสดงผลรวมค่าเบี่ยงเบนของบิต 1 ของแต่ละ Pattern Group เมื่อใช้ค่า n_1-n_2-n_3 ต่างๆ กัน

ผลรวมค่าเบี่ยงเบนที่คำนวณได้จะบ่งบอกถึงการกระจายตัวของบิต 1 ในตำแหน่งเดียวกันของแต่ละ Sub-Pattern นั่นคือ ชุดข้อมูลที่มีผลรวมค่าเบี่ยงเบนน้อยกว่าแสดงว่า มีการกระจายตัวของบิต 1 ในแต่ละตำแหน่งข้อมูลของทุกๆ Sub-Pattern ดีกว่า

จากรูปที่ 5.8 จะเห็นได้ว่าทุก Pattern Group ยกเว้น Pattern Group 19 ชุดข้อมูล (n_1-n_2-n_3) ที่มีผลรวมค่าเบี่ยงเบนน้อยที่สุด 3 อันดับแรกคือ 21-13-5, 22-14-6 และ 23-15-7 แสดงให้เห็นว่าหากใช้ชุดข้อมูลเหล่านี้ใน Hashing Function จะทำให้ โอกาสเกิด Collision จากการ Hash มีค่าน้อย

5.5 การทดลองวัดจำนวนครั้งของการ Rehash

จากการวิเคราะห์ Hashing Function ในหัวข้อที่ 5.4 พบว่า ชุดข้อมูล (n_1-n_2-n_3) ที่มีผลรวมค่าเบี่ยงเบนน้อยที่สุด 3 อันดับแรกคือ 23-15-7, 22-14-6 และ 21-13-5 ดังนั้นจึงทำการทดสอบวัดประสิทธิภาพของ Hashing Function จากการใช้ชุดข้อมูลแต่ละแบบ โดยวัดจำนวนครั้งที่เกิดการ Rehash ได้ผลการทดลองดังตารางที่ 5.3

ตารางที่ 5.3 แสดงค่าเฉลี่ยจำนวนครั้งที่เกิดการ Rehash เมื่อใช้ค่า n_1-n_2-n_3 ต่างๆ กัน

n_1-n_2-n_3	23-15-7	22-14-6	21-13-5
ค่าเฉลี่ย จำนวนครั้งของการ Rehash	2816	2875	2858

จากผลการทดสอบพบว่า ชุดข้อมูลที่ทำให้เกิดการ Rehash น้อยที่สุดคือ 23-15-7 รองลงมาคือ 21-13-5 และชุดข้อมูลที่ทำให้เกิดการ Rehash มากที่สุดคือ 22-14-6 ดังนั้น ค่าชุดข้อมูลที่เหมาะสมสำหรับใช้ใน Hashing Function คือ 21-13-5 เนื่องจากมีค่าจำนวนครั้งของการ Rehash ที่ไม่แตกต่างจาก ชุดข้อมูล 23-15-7 มากนัก และการ Rehash จะเกิดขึ้นเฉพาะในขั้นตอนการเตรียมโครงสร้างข้อมูลเพียงครั้งแรกครั้งเดียวเท่านั้น ไม่มีผลต่อเวลาการประมวลผลในช่วงที่โปรแกรม Snort ทำ Pattern Matching ระหว่างการตรวจสอบ Packet

นอกจากนี้ในการประมวลผล ชุดข้อมูล 21-13-5 จะมีขั้นตอนการประมวลผลที่น้อยกว่าชุดข้อมูล 23-15-7 จากโค้ด Hashing Function ในรูปที่ 5.9 จะเห็นว่า จำนวนบิตที่ถูก Shift ของชุดข้อมูล 21-13-5 จะน้อยกว่า 23-15-7 ดังนั้นการใช้ชุดข้อมูล 21-13-5 จะทำให้ Hashing Function ทำงานได้เร็วกว่าการใช้ชุดข้อมูล 23-15-7

```
(key ^ (a[(key>>21)&255]+a[(key>>13)&255]+a[(key>>5)&255]) & hash_table_size
```

รูปที่ 5.9 แสดงโค้ดของ Hashing Function ที่ใช้งาน

5.6 การทดลองวัดเวลาที่ใช้ในการจัดเตรียมโครงสร้างข้อมูลของโปรแกรม Snort เมื่อใช้อัลกอริธึมต่างๆ

ตารางที่ 5.4 แสดงเวลาที่ใช้ในการจัดเตรียมโครงสร้างข้อมูลของโปรแกรม Snort เมื่อใช้อัลกอริธึมต่างๆ

อัลกอริธึม	เวลาที่ใช้ในการจัดเตรียมโครงสร้างข้อมูล (วินาที)
Piranha+Cuckoo	6.864
Piranha	6.264
MWM	7.214
AC	416.15

จากผลการทดลอง ตารางที่ 5.4 พบว่าอัลกอริธึม AC เป็นอัลกอริธึมที่ใช้เวลาในการจัดเตรียมโครงสร้างข้อมูลของโปรแกรม Snort มากที่สุด เนื่องจากอัลกอริธึม AC มีการทำงานแบบ State Machine ต้องเก็บตัวอักษรทุกตัวอักษรของ Pattern ลงโครงสร้างข้อมูล และต้องมีการคำนวณหาค่า Failure Function

อัลกอริธึม Piranha เป็นอัลกอริธึมที่ใช้เวลาในการจัดเตรียมโครงสร้างข้อมูลของโปรแกรม Snort น้อยที่สุด ส่วนอัลกอริธึม Piranha+Cuckoo ที่พัฒนาใช้เวลามากกว่าเพียงเล็กน้อย เนื่องจากในขั้นตอนการจัดเตรียมโครงสร้างข้อมูลต้องเสียเวลาไปกับการ Rehash เมื่อเกิดการชนกันของข้อมูลที่จะ Hash ลง Table T_1 และ Table T_2

5.7 การทดลองตรวจสอบความถูกต้องในการทำงานของอัลกอริธึม Piranha+Cuckoo

รายละเอียดการทดลอง

- คำสั่งที่ใช้ในการทดสอบ snort –r filename โดยที่ filename คือชื่อ file ของ Dataset ที่ใช้ในการทดลอง
- จำนวน rule set ทั้งหมด 7344 rule
- ทดลอง Dataset ละ 5 ครั้ง และเก็บบันทึกจำนวน Alert ที่โปรแกรม Snort สามารถตรวจจับได้

การทดลองนี้เป็นการทดลองเพื่อตรวจสอบความถูกต้องในการทำงานของอัลกอริธึม Piranha+Cuckoo ที่ได้พัฒนาขึ้น โดยตรวจสอบจาก Output ของโปรแกรม Snort ซึ่งก็คือ จำนวน Alert หรือการแจ้งเตือนถึงการละเมิดความปลอดภัย ผลการทดลองเป็นดังตารางที่ 5.5

ตารางที่ 5.5 แสดงจำนวน Alert ของแต่ละอัลกอริธึม

Dataset	Piranha	Piranha + Cuckoo
Darpa01	3701	3701
Darpa02	3572	3572
Darpa03	2424	2424
Defcon9.1	19910	19910
Defcon9.2	193	193

จากผลการทดลอง ตารางที่ 5.5 จะเห็นว่า Output ซึ่งก็คือจำนวน Alert ที่ได้จากโปรแกรม Snort ของอัลกอริธึม Piranha มีค่าเท่ากับจำนวน Alert ของอัลกอริธึม Piranha+Cuckoo แสดงว่าอัลกอริธึม Piranha+Cuckoo ที่พัฒนาขึ้นมีการทำงานที่ถูกต้องสมบูรณ์

5.8 การทดลองเปรียบเทียบจำนวนครั้งการ Search ของแต่ละอัลกอริธึม

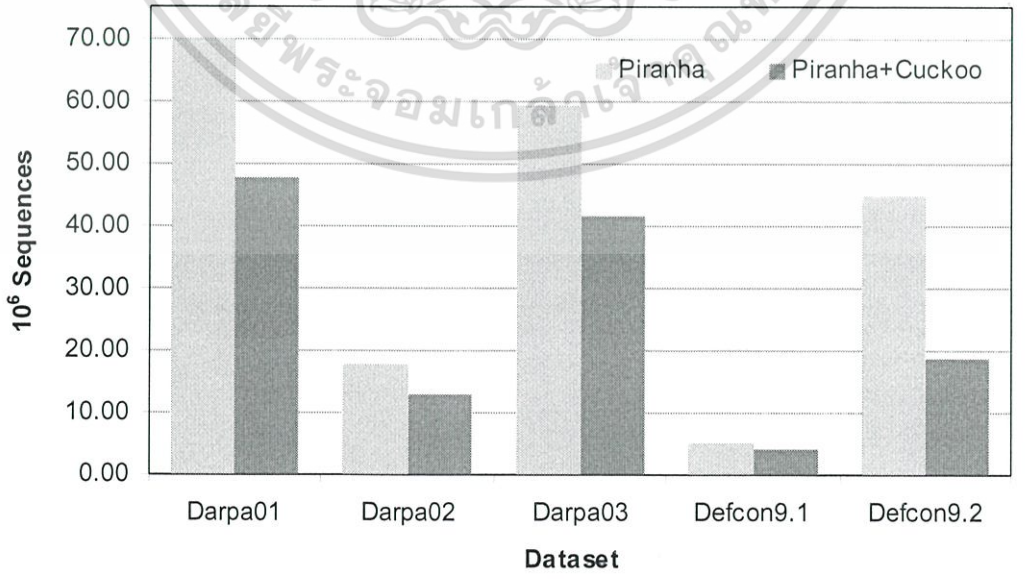
รายละเอียดการทดลอง

- ตรวจนับจำนวน Sequence ทั้งหมดที่ผ่านขั้นตอน Filtering
- ตรวจนับจำนวนครั้งการ Search ของอัลกอริธึม Piranha ในขั้นตอน Searching
- ตรวจนับจำนวนครั้งการ Search ของอัลกอริธึม Piranha+Cuckoo ในขั้นตอน Searching
- ตรวจนับจำนวน Alert ที่เกิดจากการ Match กับ Pattern

การทดลองนี้เป็นการทดลองเพื่อตรวจสอบการลดจำนวนการ Search ของแต่ละอัลกอริธึม ผลการทดลองได้ดังตารางที่ 5.6 และเมื่อนำข้อมูลมาแสดงในรูปของแผนภูมิแท่งจะได้ดังรูปที่ 5.10

ตารางที่ 5.6 แสดงจำนวนครั้งของการ Search ของแต่ละอัลกอริธึม ผลต่าง และจำนวน Alert เนื่องจากการ Match

Dataset	Input Sequences	Matched by Piranha	Matched by Piranha + Cuckoo	ผลต่าง	% ที่สามารถลดได้	จำนวน Alert เนื่องจากการ Match
Darpa01	118,867,310	69,966,901	47,738,766	22,228,135	31.77	1,201
Darpa02	35,344,980	17,671,526	13,030,892	4,640,634	26.26	347
Darpa03	98,831,474	59,096,662	41,367,680	17,728,982	30.00	900
Defcon9.1	591,277,512	5,049,044	4,179,714	869,330	17.22	7,740
Defcon9.2	202,432,131	44,602,569	18,597,123	26,005,446	58.30	60



รูปที่ 5.10 แผนภูมิแท่งแสดงจำนวนครั้งของการ Search ของแต่ละอัลกอริธึม

จากผลการทดลองพบว่า อัลกอริธึม Piranha+Cuckoo มีจำนวนการ Search เพื่อตรวจหา Pattern ที่เป็นการละเมิดความปลอดภัยน้อยกว่าอัลกอริธึม Piranha สำหรับทุกๆ Dataset ที่นำมาทดสอบ แสดงว่าอัลกอริธึม Piranha+Cuckoo ที่พัฒนาขึ้นสามารถลดจำนวนครั้งของการ Search ได้ตามสมมติฐานที่ตั้งไว้

5.9 การทดลองวัดประสิทธิภาพของอัลกอริธึม Piranha+Cuckoo เปรียบเทียบกับอัลกอริธึม Piranha ในแง่ของเวลาที่ใช้ในการประมวลผล (CPU Time)

รายละเอียดการทดลอง

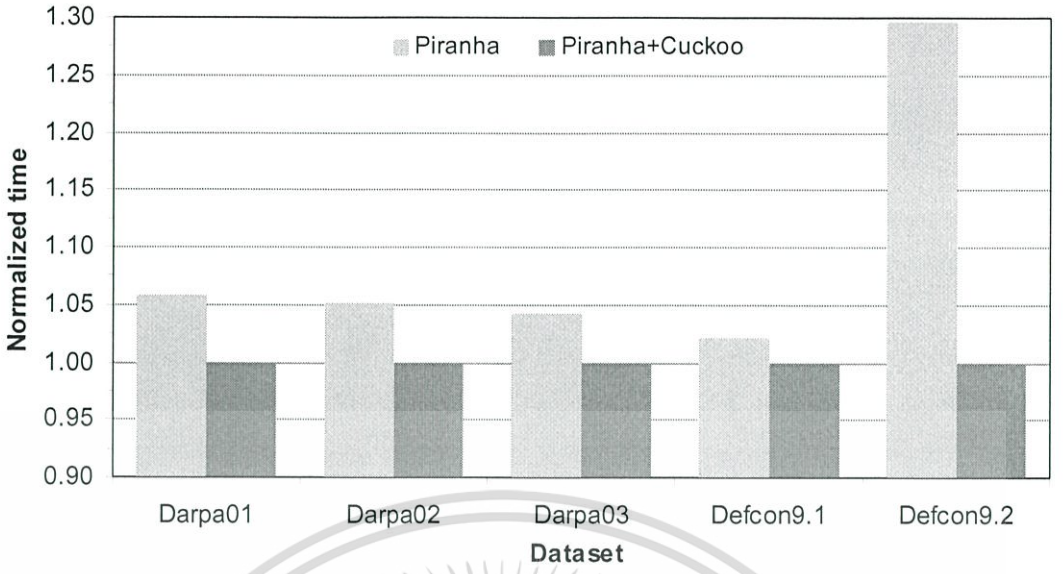
- วัดค่า CPU Time ที่ใช้ในการทำ Pattern Matching ของแต่ละ Dataset

การทดลองนี้เป็นการเปรียบเทียบการใช้งานทรัพยากรของระบบในด้าน CPU Time ของแต่ละอัลกอริธึม โดยตรวจวัดการใช้งาน CPU เฉพาะส่วนที่เป็นการทำ Pattern Matching เท่านั้น จากนั้นนำค่าที่ได้มาคำนวณหาผลต่างระหว่างอัลกอริธึม Piranha และอัลกอริธึม Piranha+Cuckoo แล้วคำนวณหาค่าเปอร์เซ็นต์ Speedup ที่ได้ ผลการทดลองแสดงดังตารางที่ 5.7

ตารางที่ 5.7 แสดง Matching Time ของแต่ละอัลกอริธึมและเปอร์เซ็นต์ Speedup

Dataset	Average Matching Time (seconds)		Speedup (%)
	Piranha	Piranha+Cuckoo	
Darpa01	15.709	14.841	5.85
Darpa02	4.335	4.125	5.09
Darpa03	13.796	13.223	4.33
Defcon9.1	12.313	12.048	2.20
Defcon9.2	8.201	6.329	29.58

จากผลการทดลองในตารางที่ 5.7 เมื่อนำค่า Matching Time ของอัลกอริธึม Piranha ไปทำการ Normalize เปรียบเทียบกับ Matching Time ของอัลกอริธึม Piranha+Cuckoo จะได้แผนภูมิแท่งดังรูปที่ 5.11



รูปที่ 5.11 แผนภูมิแท่งแสดง Matching Time ของแต่ละอัลกอริธึม

จากผลการทดลองทั้งหมดอัลกอริธึม Piranha+Cuckoo สามารถเพิ่ม Speedup ได้ตั้งแต่ 2.20 ถึง 29.58 เปอร์เซ็นต์ โดยส่วนใหญ่สามารถเพิ่ม Speedup ได้ 4.3 – 5.8 เปอร์เซ็นต์ สอดคล้องกับเปอร์เซ็นต์ของจำนวนครั้งของการ Search ที่สามารถลดลงได้ 26.26 – 31.77 เปอร์เซ็นต์

Dataset ที่อัลกอริธึม Piranha+Cuckoo สามารถเพิ่ม Speedup ได้น้อยที่สุด 2.20 เปอร์เซ็นต์ คือ Packet Defcon9.1 เนื่องจาก Packet นี้เป็น Packet ที่มีข้อมูลที่ Match กับ Pattern มากที่สุด 7,740 Packet ทำให้การ Search หา Pattern ในขั้นตอน Searching ของอัลกอริธึม Piranha นั้น มีโอกาสที่จะต้องทำการ Search จนสุด Link List น้อยลง เปอร์เซ็นต์ของจำนวนครั้งของการ Search ที่อัลกอริธึม Piranha+Cuckoo สามารถลดได้จึงมีค่าน้อยเพียง 17.22 เปอร์เซ็นต์ ค่า Speedup ที่ได้จึงมีค่าน้อยตามไปด้วย

ในทางตรงกันข้าม Dataset ที่อัลกอริธึม Piranha+Cuckoo สามารถเพิ่ม Speedup ได้มากที่สุดถึง 29.58 เปอร์เซ็นต์คือ Packet Defcon9.2 เนื่องจาก Packet นี้เป็น Packet ที่มีข้อมูลที่ Match กับ Pattern น้อยที่สุดเพียง 60 Packet ทำให้การ Search หา Pattern ในขั้นตอน Searching ของอัลกอริธึม Piranha นั้น มีโอกาสที่จะต้องทำการ Search จนสุด Link List มากขึ้น เปอร์เซ็นต์ของจำนวนครั้งของการ Search ที่อัลกอริธึม Piranha+Cuckoo สามารถลดได้จึงมีค่ามากถึง 58.30 เปอร์เซ็นต์ ส่งผลให้ได้ค่า Speedup มีค่ามากถึง 29.58 เปอร์เซ็นต์

จากผลการทดลองโดยรวมแล้ว อัลกอริธึม Piranha+Cuckoo สามารถเพิ่ม Speedup ได้กับทุก Dataset ที่ทำการทดลอง ดังนั้นแสดงว่า การใช้เทคนิคของ Cuckoo Hashing เข้ามาช่วยในการ Search หา Pattern สามารถช่วยลดจำนวนครั้งของการ Search ส่งผลให้การใช้ CPU ลดลงเป็นไปตามสมมติฐานที่ตั้งไว้

5.10 การทดลองวัดเวลาที่ใช้ในการประมวลผล (CPU Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม

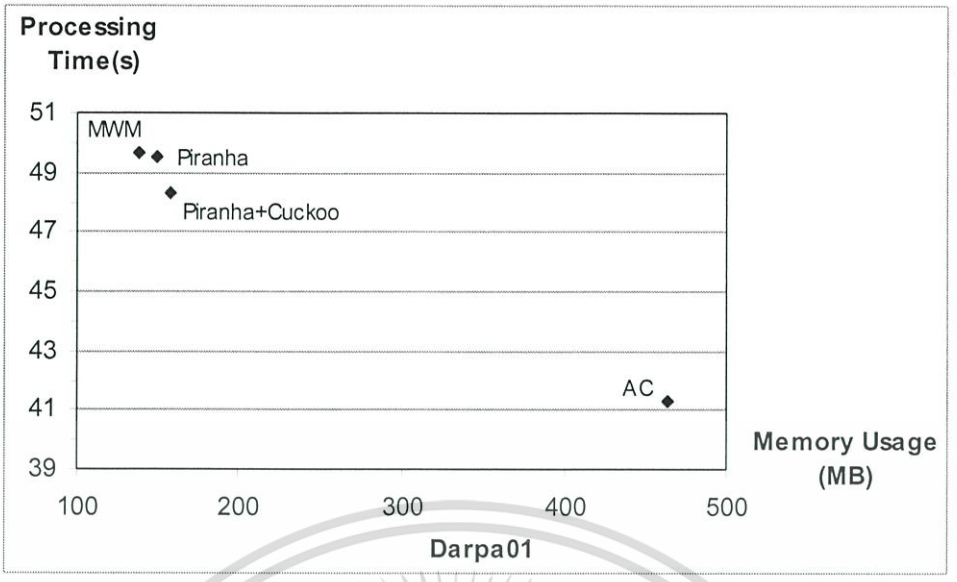
รายละเอียดการทดลอง

- วัดค่าเวลาที่ใช้ในการประมวลผล (CPU Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ในการตรวจสอบแต่ละ Dataset เมื่อใช้อัลกอริธึมต่างๆ
- วัดปริมาณหน่วยความจำที่ใช้งาน ด้วยโปรแกรม ps ของระบบปฏิบัติการลินุกซ์

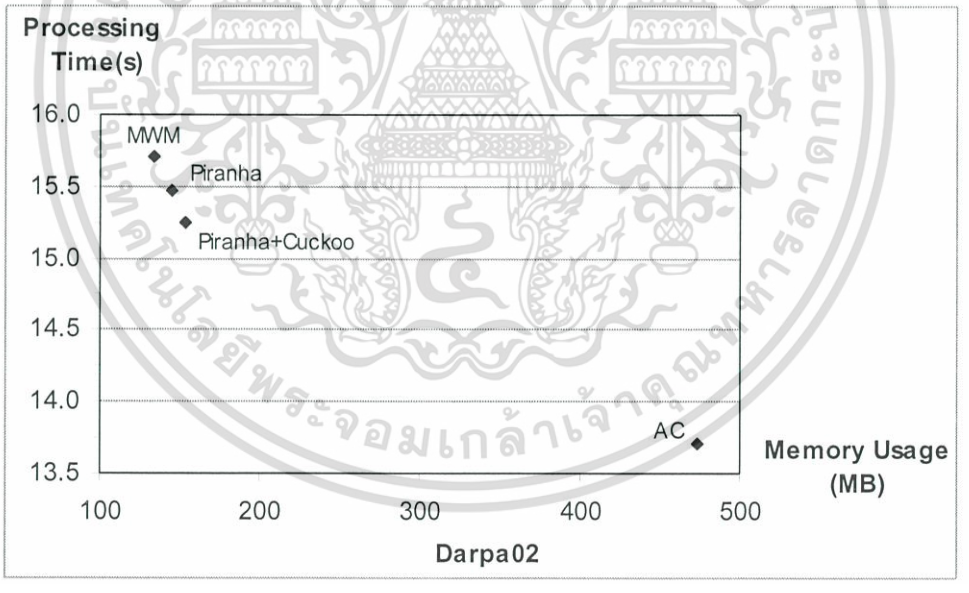
ตารางที่ 5.8 แสดงเวลาที่ใช้ในการประมวลผล (CPU Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม

Packet	เวลาที่ใช้ในการประมวลผล (วินาที)				ปริมาณหน่วยความจำที่ใช้งาน (MB)			
	Piranha	Piranha+ Cuckoo	MWM	AC	Piranha	Piranha+ Cuckoo	MWM	AC
Darpa01	49.530	48.300	49.660	41.290	148.760	157.540	138.488	464.868
Darpa02	15.470	15.240	15.700	13.710	144.520	153.296	134.352	474.104
Darpa03	41.520	40.690	42.700	34.890	148.528	157.484	138.388	461.032
Defcon9.1	46.650	44.100	32.340	41.100	151.072	159.920	140.836	434.376
Defcon9.2	34.070	32.120	33.230	25.410	140.980	149.828	130.768	436.744

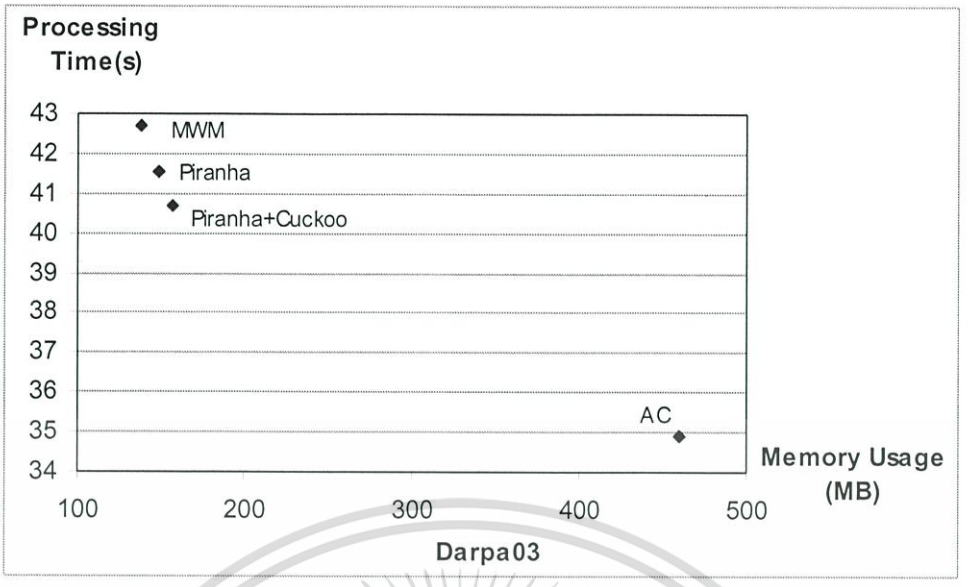
เมื่อนำข้อมูลไปแสดงในรูปแผนภูมิแท่งเปรียบเทียบแต่ละอัลกอริธึมเมื่อทดสอบด้วย Dataset เดียวกัน จะได้ดังรูปที่ 5.12 – 5.16



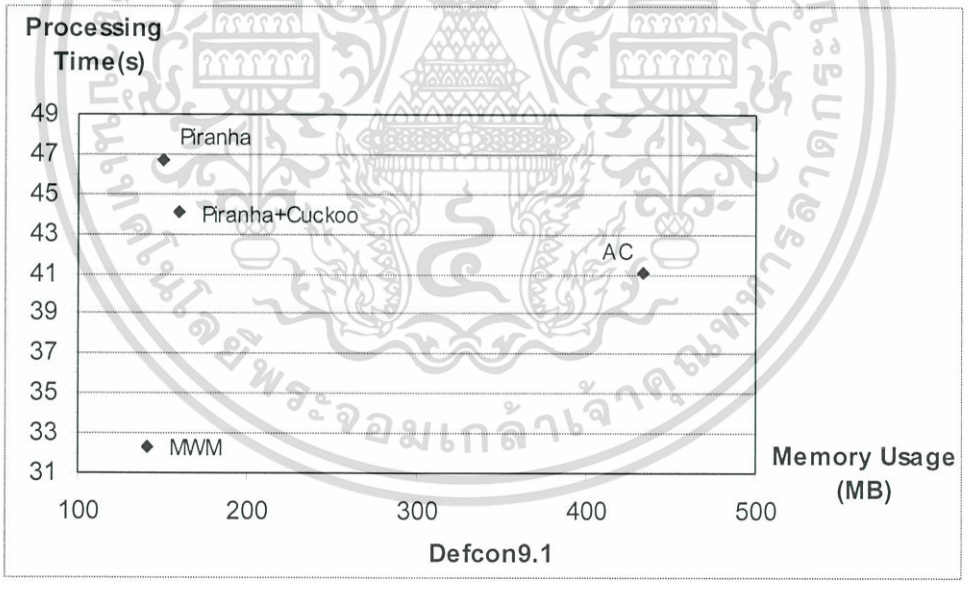
รูปที่ 5.12 แผนภูมิแท่งแสดงเวลาที่ใช้ในการประมวลผล (Processing Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม เมื่อทดสอบด้วย Packet Darpa01



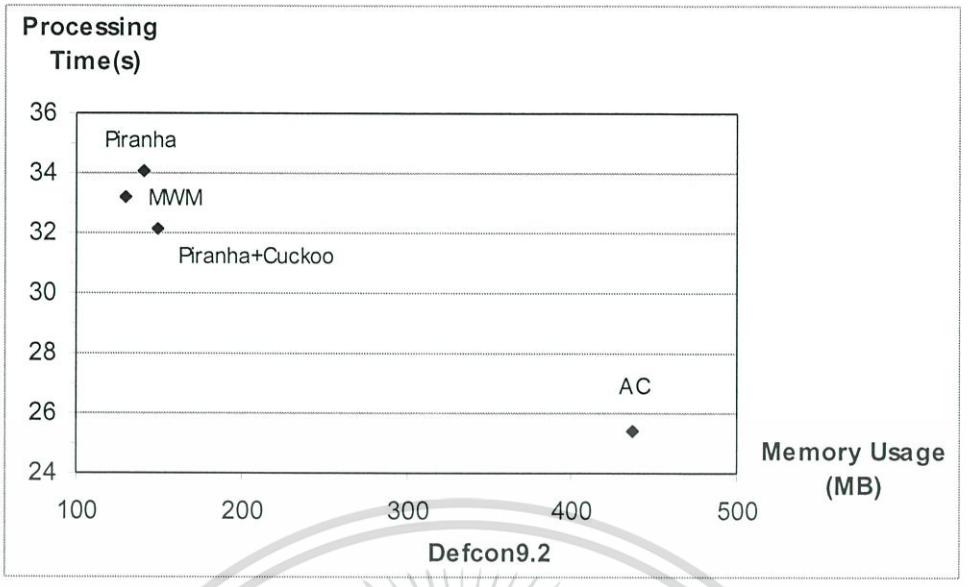
รูปที่ 5.13 แผนภูมิแท่งแสดงเวลาที่ใช้ในการประมวลผล (Processing Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม เมื่อทดสอบด้วย Packet Darpa02



รูปที่ 5.14 แผนภูมิแท่งแสดงเวลาที่ใช้ในการประมวลผล (Processing Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม เมื่อทดสอบด้วย Packet Darpa03



รูปที่ 5.15 แผนภูมิแท่งแสดงเวลาที่ใช้ในการประมวลผล (Processing Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม เมื่อทดสอบด้วย Packet Defcon9.1



รูปที่ 5.16 แผนภูมิแท่งแสดงเวลาที่ใช้ในการประมวลผล (Processing Time) และปริมาณหน่วยความจำที่ใช้งาน (Memory Usage) ของแต่ละอัลกอริธึม เมื่อทดสอบด้วย Packet Defcon9.2

จากผลการทดลองจะพบว่า อัลกอริธึม AC เป็นอัลกอริธึมที่มีความเร็วในการประมวลผลน้อยที่สุด แต่ใช้ปริมาณหน่วยความจำมากที่สุด อัลกอริธึม Piranha+Cuckoo ที่พัฒนาสามารถประมวลผลได้เร็วกว่าอัลกอริธึม Piranha ในทุกๆ Packet ที่ทำการทดสอบ ดังนั้นแสดงว่า การพัฒนาประสิทธิภาพของอัลกอริธึม Pattern Matching ให้มีประสิทธิภาพดีขึ้น สามารถช่วยลดเวลาที่ใช้ในการประมวลผลการตรวจสอบ Packet ทั้งหมดได้

ในเรื่องของหน่วยความจำที่ใช้งานพบว่า อัลกอริธึม Piranha+Cuckoo ใช้หน่วยความจำในการประมวลผลมากกว่าอัลกอริธึม Piranha 8.7 – 8.9 MB แต่เนื่องจากในปัจจุบัน หน่วยความจำมีราคาถูกซึ่งหากคิดราคาเป็นบาทต่อปริมาณหน่วยความจำเป็น MB พบว่า ราคาเฉลี่ยอยู่ที่ไม่ถึง 1 บาท ต่อ 1 MB ดังนั้นการที่อัลกอริธึม Piranha+Cuckoo ใช้ปริมาณหน่วยความจำในการประมวลผลมากกว่าอัลกอริธึม Piranha เพียงเล็กน้อยแต่ได้ประสิทธิภาพการประมวลผลที่ดีขึ้นจึงเป็นสิ่งที่ยอมรับได้ในการนำไปใช้งานจริง

บทที่ 6
สรุปผลการวิจัย และแนวทางการพัฒนา

6.1 สรุปผลการวิจัย

งานวิจัยนี้นำเสนอการเพิ่มประสิทธิภาพของการทำ Pattern Matching สำหรับระบบตรวจจับผู้บุกรุก โดยได้นำเทคนิคการ Hashing แบบ Cuckoo Hashing มาประยุกต์ใช้เพื่อเพิ่มประสิทธิภาพการทำงาน

งานวิจัยที่นำเสนอนี้มุ่งเน้นไปที่การลดจำนวนครั้งของการ Search หา Pattern ใน Payload ของ Packet ในกระบวนการทำงานของอัลกอริธึม Pattern Matching โดยเลือกที่จะทำการพัฒนาอัลกอริธึม Piranha ซึ่งเป็นอัลกอริธึมที่ใช้ในระบบตรวจจับผู้บุกรุกและเป็นอัลกอริธึมที่มีประสิทธิภาพดีทั้งในแง่ของเวลาในการประมวลผล (CPU Time) และปริมาณหน่วยความจำที่ใช้ (Memory Usage) จากการวิเคราะห์กระบวนการทำงานของอัลกอริธึม Piranha พบว่า มีข้อเสียหรือจุดด้อยในขั้นตอนการ Searching ที่ใช้โครงสร้างข้อมูล Hashing แบบ Separate Chaining ทำให้ความยาวของ Chain ส่งผลต่อจำนวนครั้งของการ Search หา Pattern ใน Payload ของ Packet ต่างๆ

เมื่อนำอัลกอริธึม Cuckoo Hashing มาประยุกต์เพื่อการพัฒนาประสิทธิภาพ โดยปรับเปลี่ยนในส่วนของโครงสร้างข้อมูลของขั้นตอน Searching ทำให้สามารถลดจำนวนครั้งของการ Search หา Pattern มากที่สุดไม่เกิน 2 ครั้ง เปรียบเทียบกับโครงสร้างข้อมูลและอัลกอริธึมเดิมที่การ Search หา Pattern อาจมากถึง 25 ครั้ง (จากตารางที่ ก.1 ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด 25 โหนด ของ Pattern Group 17)

จากการทดลองเพื่อตรวจสอบความถูกต้องในการทำงานของอัลกอริธึมที่นำเสนอ พบว่าอัลกอริธึมที่พัฒนาให้ค่า Output ซึ่งเป็นจำนวน Alert ที่พบตรงกับอัลกอริธึมเดิม ดังนั้นแสดงว่าอัลกอริธึมที่พัฒนาสามารถทำงานได้อย่างถูกต้อง การทดลองต่อมาเป็นการทดลองเปรียบเทียบจำนวน Packet ที่แต่ละอัลกอริธึมทำการตรวจสอบ จากผลการทดลองพบว่า อัลกอริธึมที่พัฒนาสามารถลดจำนวนครั้งของการ Search ลงได้ 17.22 ถึง 58.30 เปอร์เซ็นต์ในทุก Dataset ที่ทำการทดลอง ดังนั้นแสดงว่า เทคนิค Cuckoo Hashing สามารถช่วยลดจำนวนครั้งของการ Search ได้ ซึ่งเป็นไปตามสมมุติฐานที่ตั้งไว้

สำหรับการทดลองสุดท้ายเป็นการทดลองเพื่อวัดประสิทธิภาพของอัลกอริธึมที่นำเสนอเปรียบเทียบกับอัลกอริธึมเดิม จากผลการทดลองพบว่า อัลกอริธึมที่พัฒนาสามารถเพิ่มค่า Speedup ได้ตั้งแต่ 4.3 ถึง 5.8 เปอร์เซ็นต์ในทุก Dataset ที่ทำการทดลอง ซึ่งเป็นผลจากการลดจำนวนครั้งของการ Search ดังนั้นแสดงว่า อัลกอริธึมที่พัฒนามีแนวโน้มประสิทธิภาพการทำงานที่ดีกว่าอัลกอริธึมเดิมในแง่ของเวลาที่ใช้ในการประมวลผล

6.2 แนวทางการพัฒนาต่อ

จากผลการทดลองทั้งหมดจะเห็นได้ว่า อัลกอริธึมที่พัฒนานั้นสามารถลดจำนวนครั้งของการ Search ได้มากที่สุดถึง 58.30 เปอร์เซ็นต์ แต่ Speedup สามารถทำได้มากที่สุดเพียง 5.8 เปอร์เซ็นต์ เนื่องจาก การเสียเวลาในการประมวลผลในส่วนของ Hashing Function (Function H_1 และ Function H_2) ในขั้นตอน Searching (รูปที่ 4.3) ดังนั้นหากสามารถสร้าง Hashing Function ที่มีการกระจายตัวของข้อมูลที่ดีและสามารถประมวลผลได้เร็วขึ้น จะส่งผลต่อเวลาที่ใช้ในการประมวลผลอย่างมาก และสามารถเพิ่มประสิทธิภาพโดยรวมของการทำ Pattern Matching ได้

เอกสารอ้างอิง

[1] M. Roesch, "Snort: Lightweight intrusion detection for networks." In Proceedings of the 1999 USENIX LISA Systems Administration Conference, November 1999

[2] M. Fisk and G. Varghese, "An analysis of fast string matching applied to content-based forwarding and intrusion detection" Technical Report CS2001-0670 (updated version), 2002

[3] S. Antonatos, M. Polychronakis, P. Akritidis, Kostas D. Anagnostakis, and Evangelos P. Markatos, "Piranha: Fast and Memory-efficient Pattern Matching for Intrusion Detection" In the Proceedings 20th IFIP International Information Security Conference (SEC 2005), May 2005

[4] R. Pagh and F. Rodler, "Cuckoo Hashing" Journal of Algorithms, 51(2):122-144, 2004

[5] IDS/IPS ระบบรักษาความปลอดภัย, http://wiki.nectec.or.th/bu/ITM532Students_2008/SecuritySystem

[6] Aho AV, Corasick MJ, "Efficient string matching: and aid to bibliographic search" Communications of the ACM 18, June 1975, pp. 33-340

[7] Sun Wu, Udi Manber, "A Fast Algorithm For Multi-Pattern Searching" Technical Report TR 94-17, University of Arizona at Tuscon, May 1994

[8] S. McCanne, C. Leres, and V. Jacobson, "libpcap" Lawrence Berkeley Laboratory, Berkeley, CA, Available via anonymous ftp to ftp.ee.lbl.gov

[9] Tran Ngoc Thinh, "FPGA-Based Architecture For Pattern Matching Using Cuckoo Hashing In Network Intrusion Detection System, 2009

[10] 1999 DARPA Intrusion Detection Evaluation Data Set. http://www.ll.mit.edu/IST/ideval/data/1999/1999_data_index.html

[11] DefCon 9.0 Dataset, http://ictf.cs.ucsb.edu/data/defcon_ctf_09/

[12] Snort official website, http://www.snort.org

ภาคผนวก

ภาคผนวก ก.

การทดลองวัดความยาวโหนด Sub-Pattern ที่ยาวที่สุด

ตารางที่ ก.1 แสดงจำนวน Pattern ทั้งหมดของแต่ละ Pattern Group และความยาวของโหนด Sub-Pattern ที่ยาวที่สุดของ Pattern Group นั้นๆ

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
1	273	12
2	273	12
3	273	12
4	277	12
5	324	12
6	276	12
7	278	13
8	286	12
9	298	12
10	276	12
11	326	12
12	273	12
13	274	12
14	273	12
15	286	12
16	279	12
17	818	25
18	472	12
19	921	21
20	275	12
21	273	12
22	276	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
24	275	13
25	273	12
26	274	12
27	274	12
28	283	12
29	277	12
30	277	12
31	333	15
32	292	12
33	275	12
34	273	12
35	275	12
36	275	12
37	273	12
38	274	12
39	274	12
40	274	12
41	273	12
42	274	12
43	275	12
44	299	12
45	274	12
46	273	12
47	273	12
48	273	12
49	274	12
50	273	12
51	275	12
52	278	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
54	277	12
55	274	12
56	275	12
57	274	12
58	273	12
59	273	12
60	273	12
61	275	12
62	274	12
63	281	12
64	276	12
65	274	12
66	273	12
67	274	12
68	274	12
69	274	12
70	277	12
71	274	12
72	276	12
73	274	12
74	274	12
75	275	12
76	275	12
77	276	12
78	276	12
79	276	12
80	276	12
81	274	12
82	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
84	273	12
85	273	12
86	273	12
87	274	12
88	275	12
89	274	12
90	274	12
91	274	12
92	274	12
93	275	12
94	275	12
95	273	12
96	274	12
97	274	12
98	274	12
99	274	12
100	274	12
101	274	12
102	274	12
103	1	1
104	278	12
105	274	12
106	276	12
107	275	12
108	274	12
109	273	12
110	273	12
111	275	12
112	315	15

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
114	273	12
115	465	12
116	275	12
117	273	12
118	274	13
119	274	12
120	274	12
121	274	12
122	274	12
123	276	12
124	277	12
125	274	12
126	274	12
127	273	12
128	277	12
129	274	12
130	274	12
131	273	12
132	273	12
133	273	12
134	274	12
135	273	12
136	279	12
137	276	12
138	274	12
139	273	12
140	274	12
141	275	12
142	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
144	274	12
145	274	12
146	274	12
147	274	12
148	274	12
149	273	12
150	274	12
151	273	12
152	1	1
153	274	12
154	275	12
155	274	12
156	274	12
157	273	12
158	273	12
159	278	12
160	273	12
161	276	12
162	273	12
163	6	2
164	278	12
165	278	12
166	274	12
167	274	12
168	274	12
169	275	12
170	274	12
171	274	12
172	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
174	275	12
175	274	12
176	274	12
177	274	12
178	275	12
179	276	12
180	274	12
181	274	12
182	276	12
183	274	12
184	273	12
185	274	12
186	273	12
187	274	12
188	274	12
189	275	12
190	274	12
191	274	12
192	274	12
193	274	12
194	273	12
195	274	12
196	274	12
197	274	12
198	275	12
199	275	12
200	274	12
201	274	12
202	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
204	275	12
205	276	12
206	273	12
207	274	12
208	274	12
209	274	12
210	273	12
211	273	12
212	275	12
213	274	12
214	274	12
215	275	12
216	274	12
217	273	12
218	275	12
219	273	12
220	274	12
221	273	12
222	273	12
223	274	12
224	273	12
225	273	12
226	273	12
227	275	12
228	276	12
229	277	12
230	274	12
231	274	12
232	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
234	276	12
235	274	12
236	274	12
237	276	12
238	278	12
239	274	12
240	274	12
241	274	12
242	273	12
243	274	12
244	274	12
245	273	12
246	275	12
247	273	12
248	275	12
249	274	12
250	273	12
251	274	12
252	275	12
253	274	12
254	1	1
255	273	12
256	3	1
257	274	12
258	279	12
259	274	12
260	274	12
261	273	12
262	1	1

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
264	277	12
265	1	1
266	274	12
267	275	12
268	275	12
269	276	12
270	277	12
271	277	12
272	275	12
273	274	12
274	274	12
275	274	12
276	274	12
277	273	12
278	276	12
279	274	12
280	275	12
281	273	12
282	274	12
283	275	12
284	274	12
285	275	12
286	276	12
287	273	12
288	274	12
289	273	12
290	274	12
291	274	12
292	275	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
294	274	12
295	275	12
296	275	12
297	273	12
298	273	12
299	277	12
300	274	12
301	274	12
302	274	12
303	273	12
304	273	12
305	275	12
306	277	12
307	273	12
308	274	12
309	274	12
310	274	12
311	274	12
312	273	12
313	274	12
314	274	12
315	274	12
316	274	12
317	274	12
318	274	12
319	274	12
320	274	12
321	274	12
322	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
324	273	12
325	275	12
326	274	12
327	276	12
328	274	12
329	273	12
330	273	12
331	274	12
332	274	12
333	274	12
334	274	12
335	275	12
336	274	12
337	276	12
338	274	12
339	274	12
340	274	12
341	274	12
342	273	12
343	273	12
344	274	12
345	274	12
346	275	12
347	274	13
348	273	12
349	274	12
350	274	12
351	273	12
352	274	12

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
354	274	12
355	274	12
356	274	12
357	274	12
358	274	12
359	274	12
360	274	12
361	274	12
362	274	12
363	273	12
364	30	11
365	30	11
366	31	11
367	31	11
368	30	11
369	30	11
370	31	11
371	32	11
372	34	11
373	30	11
374	35	11
375	30	11
376	32	12
377	30	11
378	35	11
379	30	11
380	30	11
381	36	11
382	31	11

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
384	31	11
385	31	12
386	31	11
387	34	11
388	31	12
389	33	11
390	31	11
391	32	11
392	31	11
393	31	11
394	31	11
395	31	11
396	30	11
397	31	11
398	30	11
399	31	11
400	31	11
401	31	11
402	31	11
403	31	11
404	31	11
405	31	11
406	30	11
407	31	11
408	30	11
409	33	11
410	33	11
411	31	11
412	31	11

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
414	30	11
415	30	11
416	30	11
417	30	11
418	31	11
419	30	11
420	30	11
421	31	11
422	31	11
423	31	11
424	31	11
425	31	11
426	33	11
427	31	11
428	30	11
429	31	11
430	31	11
431	31	11
432	31	11
433	31	11
434	31	11
435	31	11
436	35	11
437	30	11
438	31	11
439	32	11
440	31	12
441	31	11
442	31	11

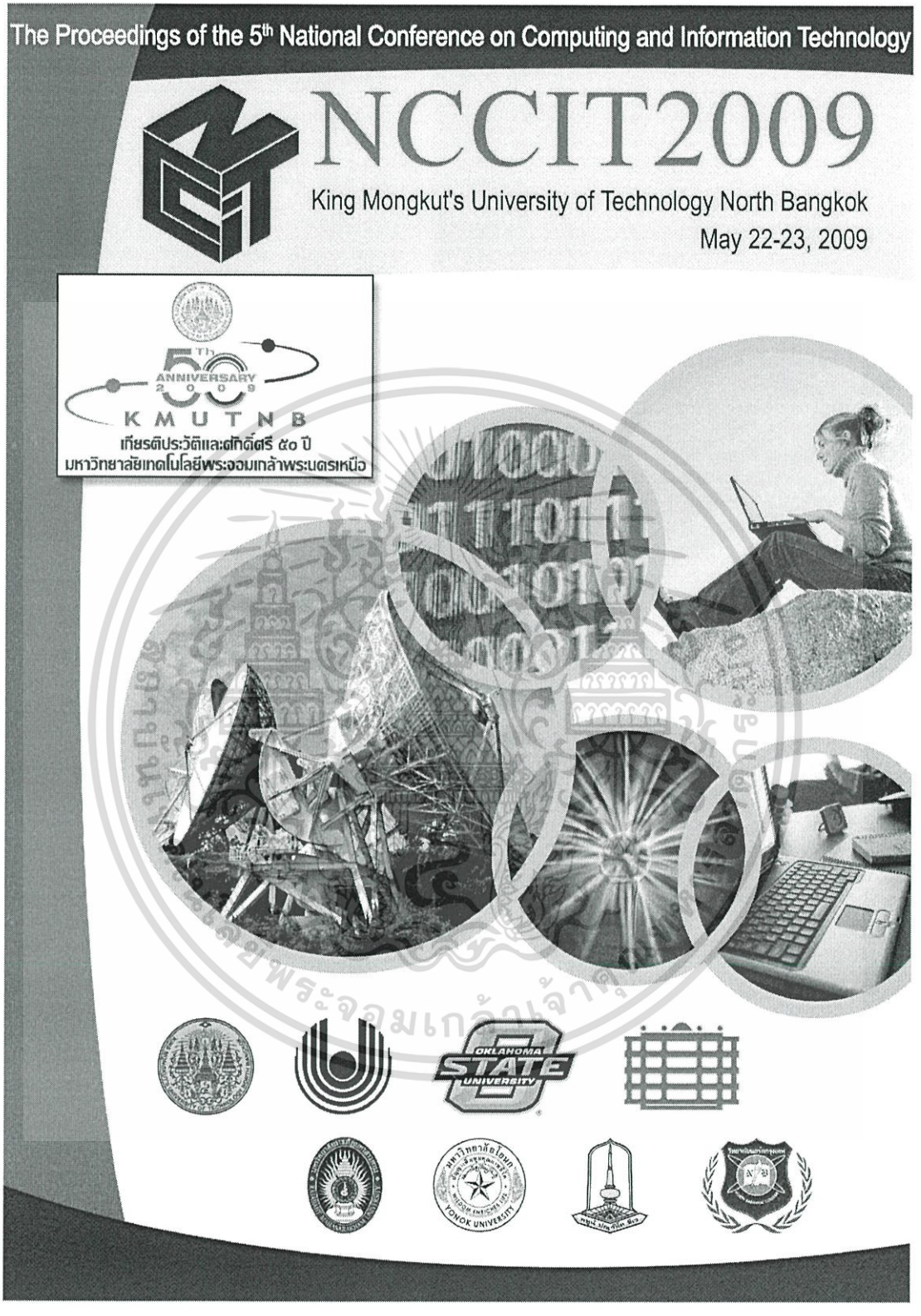
Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
444	30	11
445	10	1
446	2	1
447	2	1
448	2	1
449	2	1
450	2	1
451	2	1
452	2	1
453	27	3
454	2	1
455	2	1
456	2	1
457	2	1
458	2	1
459	2	1
460	2	1
461	2	1
462	2	1
463	2	1
464	2	1
465	2	1
466	2	1
467	2	1
468	2	1
469	2	1
470	2	1
471	2	1
472	2	1

Pattern Group	จำนวน Pattern	ความยาวของโหนด Sub-Pattern ที่ยาวที่สุด (โหนด)
474	2	1
475	1	1
476	1	1
477	1	1
478	1	1
479	1	1
480	1	1
481	1	1

ภาคผนวก ข.

ผลงานวิจัยในระหว่างการศึกษาที่ได้รับการตีพิมพ์เผยแพร่

1. Pornchai Korpraserttaworn, Surin Kittitornkun , "**Piranha + Cuckoo Snort NIDS**", The 5th National Conference on Computing and Information Technology (NCCIT 2009), pp. 431-435, 2009.



The 5th National Conference on Computing and Information Technology NCCIT 2009

Piranha + Cuckoo Snort NIDS

Pornchai Korpraserttaworn

Department of Computer Engineering
Faculty of Engineering
KMITL

Surin Kittitornkun

Department of Computer Engineering
Faculty of Engineering
KMITL

Abstract

Network Intrusion Detection Systems (NIDSs) provide an important security function to defend network attacks. As network speeds and workloads increase, it is important for NIDSs to be highly efficient. Most NIDSs must check for thousands of known attack patterns in every packet, making pattern matching the most expensive part of signature-based NIDSs in terms of processing and memory resources. Piranha+Cuckoo Snort combines filtering of Piranha and hashing of Cuckoo together. Especially the matching speed is faster up to 29% than that of Piranha. In this paper, we first introduce the background of Snort NIDS, Piranha, Cuckoo hashing and then present the experimental results. We then discuss the results and finally draw conclusions.

Keyword: Network Intrusion Detection System, Piranha, Cuckoo Hashing, Pattern Matching, Snort

1. Introduction

Network Intrusion Detection Systems (NIDSs) provide a powerful mechanism to defend against well-known attacks on a computer network or detect network abuse. NIDSs are mainly divided into two major categories: signature-based and anomaly detection. Anomaly-detection NIDSs try to spot abnormal behaviour on network based on statistics like rate of connections, traffic overload or unusual protocol headers. On the contrary, the detection mechanism of a signature-based NIDS is based on a set of signatures, each describing a known attack. As an example, a signature taken from latest Snort, is

alert tcp any any -> HTTP_SERVER 80 (content:"/root.exe"; nocase;)

This signature instructs that if "/root.exe" is found inside the payload of a TCP packet that is originating from any host and any source port and is destined to an HTTP server on port 80, then an attack on the web server is taking place. While this signature requires full packet inspection, there exist simpler signatures that require only header lookups. Pattern (string) matching inflicts a significant cost to the performance of signature-based NIDSs. Previous research results suggest that 30% of total processing time is spent on pattern matching [1], while in some case like Web-intensive traffic, this percentage rises up to 80% [2]. Apart from processing time, memory demands of an NIDS may reach at high levels due to rule-set growth. Although algorithms with lower memory demands have been developed, their performance in comparison with algorithms that consume more memory is still poor. Given the fact that network speed/throughput increases every year, pattern matching evolves to a highly demanding process that needs special considerations. Minimizing the demands of pattern matching leaves headroom for further heuristics to be applied for intrusion detection, like anomaly detection or sophisticated preprocessors.

This paper is organized as follows. Section 2 introduces Snort, an open source NIDS, related backgrounds. The proposed Piranha+Cuckoo Snort is elaborated in Section 3. Results and conclusions are discussed and drawn in Section 4 and 5, respectively.

2. Snort and related backgrounds

2.1 Snort

Snort is a libpcap-based packet sniffer and logger that can be used as a lightweight network intrusion detection system (NIDS). See the data flow of Snort in Figure 1. It features rule-based logging to perform content pattern matching and detect a variety of attacks. In addition, it can probe, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, and many more. Snort has real-time alerting capability, with alerts being sent to syslog, Server Message Block (SMB) "WinPopup" messages, or a separate "alert" file. Snort is configured using command line switches and optional Berkeley Packet Filter commands. The detection engine is programmed using a simple language that describes per packet tests and actions. Ease of use simplifies and expedites the development of new detection rules. For example, when the IIS Show code web exploits were revealed on the Bugtraq mailing list, Snort rules to detect the probes were available within a few hours.

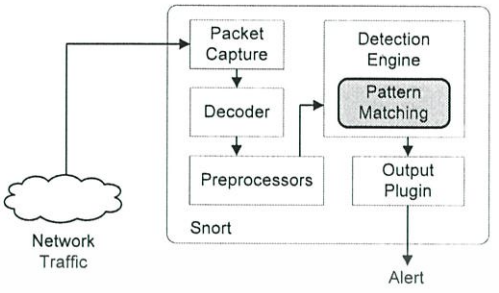


Figure 1: Snort's Data Flow and Pattern Matching Engine.

2.2 Piranha [3]

Piranha Snort consists of four main stages including Preprocessing and Searching.

- Preprocessing

Piranha treats every byte-aligned pattern as a set of 32-bit (4-byte) sub-patterns or sequences. For example, the pattern "*/admin.exe*" can be considered as the set of its 32-bit byte-aligned sequences as follows, "*/adm*", "*admi*", "*dmin*", "*min.*", "*in.e*","*n.ex*" and "*.exe*". The 32-bit partitioning is chosen because of faster native 32-bit integer operations of the common 32-bit Intel Architecture. The next step is to select rare sub-patterns to form a filter table (a linear hash table using modulo function) as shown in Figure 3.

- Searching

Piranha combines 4 bytes of the incoming payload as one sequence and hashes each sequence (unsigned 32-bit integer) using modulo function with the table size of 16,384. The hash value points to a linked list of 4-byte sequences. Each 4-byte sequence corresponds to the exact rule pattern for final comparison. Due to the poor hashing (modulo) function, numerous collisions occur resulting in a long chain of up to 25 nodes and consequently long matching time.

2.3 Cuckoo Hashing

Cuckoo Hashing [4] is a dynamic mapping of a static dictionary. The dictionary uses two hash tables, T_1 and T_2, each consisting of r words, and two hash functions $h_1, h_2 : U \rightarrow \{0, \ldots, r-1\}$. Every pattern $x \in S$ is stored in either cell $h_1(x)$ of T_1 or cell $h_2(x)$ of T_2, but never in both. The lookup function is

```
function lookup(x)
    return T1[h1(x)] = x OR T2[h2(x)] = x;
end
```

Each pattern lookup [function lookup(x)] needs two table accesses. In fact, it is optimal among all dictionaries using linear space, except for special cases, see [5]. It is also shown in [5] that if $r \geq (1+\varepsilon)n$ for some constant $\varepsilon > 0$ (i.e., the tables are a bit less than half full), and h_1, h_2 are picked uniformly at random from an (O(1), O(log n))-universal family, the probability that there is no way of arranging the patterns of S according to h_1 and h_2 is O(1/n).

We now consider a simple implementation of the above, still assuming $r \geq (1 + \varepsilon)n$ for some constant $\varepsilon > 0$. Deletion is very simple to perform in constant time, not counting the possible cost of shrinking the tables if they are becoming too sparse. As for pattern insertion [procedure insert(x)], it turns out that the "cuckoo approach", kicking other patterns away until every pattern has its own "home", works very well. Specifically, if a pattern x is to be inserted we first see if cell $h_1(x)$ of T_1 is occupied. If not, x is inserted. Otherwise we set $T_1[h_1(x)] \leftarrow x$ anyway, thus making the previous occupant "homeless". This pattern y is then inserted in T_2 in the same way, and so forth iteratively, see Figure 2(a).

It may happen that this process loops infinitely, as shown in Figure 2(b). Therefore the number of iterations is bounded by a value MaxLoop. If this MaxLoop is reached, the patterns shall be rehashed in tables T_1 and T_2 using two new hash functions. There is no need to allocate new tables for the rehashing: We may simply run through the tables to delete and perform the usual insertion procedure on all patterns found not to be at their intended position in the table. (Note that kicking away a pattern that is not in its intended position simply corresponds to starting a new insertion of this pattern.)

```
procedure insert(x)

  if lookup(x) then return;
  loop MaxLoop times
    x = T1[h1(x)];
    if x = NULL then return;
    x = T2[h2(x)];
    if x = NULL then return;
  end loop
  rehash();
  insert(x);
  end
```

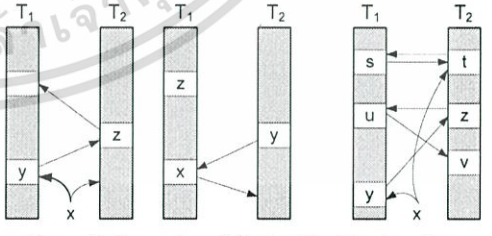


Figure 2: Examples of Cuckoo Hashing insertion. Arrows show possibilities for moving patterns.
(a) Pattern x is successfully inserted by moving patterns y and z from one table to the other.
(b) Pattern x cannot be accommodated and a rehash is necessary.

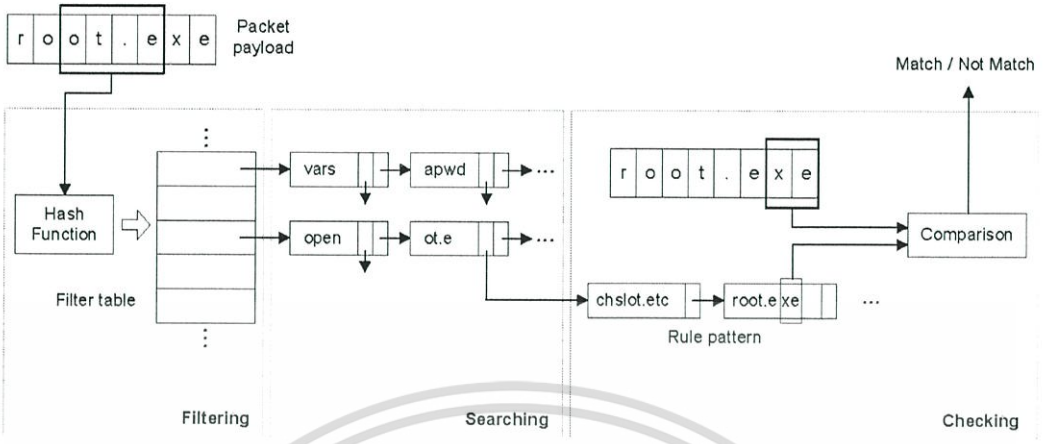


Figure 3: Piranha pattern matching algorithm.

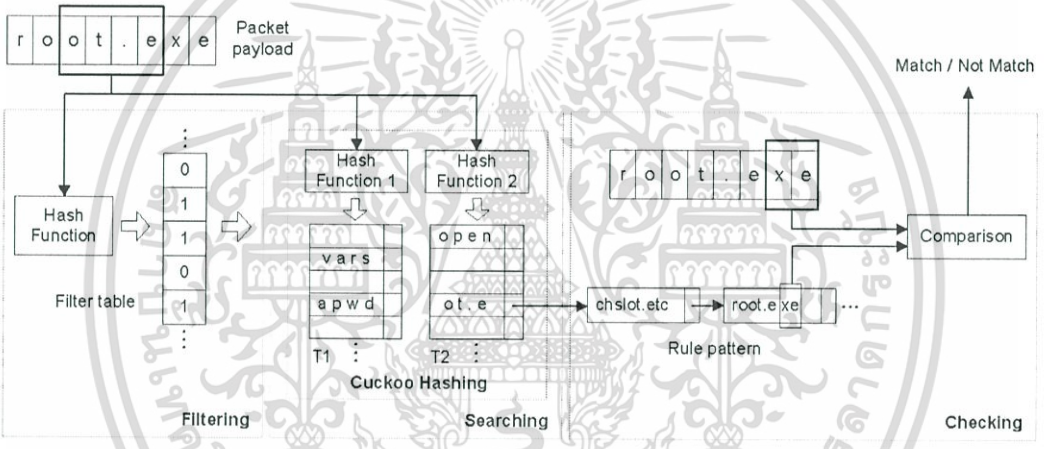


Figure 4: Piranha + Cuckoo pattern matching algorithm.

3. Piranha + Cuckoo Snort

As mentioned earlier, sequence filtering and searching of Piranha in Figure 3 can be replaced with Cuckoo hashing to not only solve the collision in Piranha linear/modulo hash table but also reduce the run time complexity. In the searching phase, Piranha+Cuckoo in Figure 4 only hashes two times while Piranha originally hashes once but has to traverse along the linked list up to 25 nodes to find whether it is matched or not matched. Hence, Piranha+Cuckoo Snort can support the increasing number of known attack patterns with almost constant look up time.

4. Experiment and results

We evaluate the performance of Piranha against Piranha-Cuckoo algorithms in Snort 2.4.2 using five packet traces. All Snort preprocessors were disabled.

4.1 Environment

All the experiments are conducted on a 500-MHz Pentium III machine with 32 KB of L1cache, 512 KB of L2 cache, and 512 MB of main memory. The host operating system is Linux (kernel version 2.4.27, Debian 3.1r5 Sarge). We use five full-packet traces Darpa01, Darpa02, Darpa03, which were collected during the DARPA evaluation tests at MIT Lincoln Laboratory in 1999 and Defcon9.1, Defcon9.2 which were collected during the Capture The Flag Contest 2001.

Each packet trace is read by Snort using –r option in a tcpdump format file as shown in Figure 5. Therefore, Snort can run at almost 100% processor utilization. Each trace is run 10 times to average out the random nature.

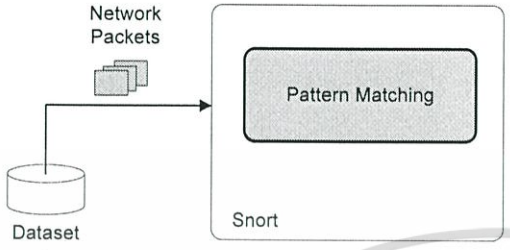


Figure 5: Pattern matching of network packets from a tcpdump data trace file.

4.2 Performance Comparison

We first make sure that Piranha+Cuckoo Snort works correctly as it can detect the same number of alerts as Piranha Snort in Table 1. Then, we can determine the performance of each algorithm by measuring the following parameters: number of matching sequences and amount of matching time. The number of matching sequences describes the run time complexity as shown in Table 2. Piranha+Cuckoo can greatly reduce the number of matched (searched) sequences because Cuckoo hashing requires two lookups to check if it is a match or not. Furthermore, Piranha must traverse along the chain to find out a match or no match.

The amount of matching time can be measured using *times*, the standard library C function. In each trace, the average matching time of Piranha (T1) is normalized to that of Piranha+Cuckoo (T2) in Table 3 and plotted in Figure 6. As a result of lower number of Matched Sequences in Table 2, the average matching time of Piranha+Cuckoo is faster than that of Piranha from 2.2% - 29.58%. It can be observed that the %Speed Up in Table 2 corresponds to the Difference of Table 2. For example, %SpeedUp of 2.2% corresponds to the Difference 869,330 Sequences of Defcon9.1. Accordingly, %SpeedUp of 29.58% corresponds to the Difference 26,005,446 Sequences of Defcon9.2.

Table 1: Number of Alerts Detected by Piranha+Cuckoo and Piranha.

Traces	Piranha	Piranha + Cuckoo
Darpa01	3701	3701
Darpa02	3572	3572
Darpa03	2424	2424
Defcon9.1	19910	19910
Defcon9.2	193	193

Table 2: Number of Input 4-byte Sequences, Matched by Piranha, Matched by Piranha+Cuckoo, and Difference.

Traces	Input Sequences	Matched by Piranha	Matched by Piranha + Cuckoo	Difference
Darpa01	118,867,310	69,966,901	47,738,766	22,228,135
Darpa02	35,344,980	17,671,526	13,030,892	4,640,634
Darpa03	98,831,474	59,096,662	41,367,680	17,728,982
Defcon9.1	591,277,512	5,049,044	4,179,714	869,330
Defcon9.2	202,432,131	44,602,569	18,597,123	26,005,446

Table 3: Average Matching Time (Seconds) of Piranha and Piranha+Cuckoo.

Packet	Average Matching Time (seconds)		SpeedUp (%)
	Piranha (T1)	Piranha+Cuckoo (T2)	
Darpa01	15.709	14.841	5.85
Darpa02	4.335	4.125	5.09
Darpa03	13.796	13.223	4.33
Defcon9.1	12.313	12.048	2.20
Defcon9.2	8.201	6.329	29.58

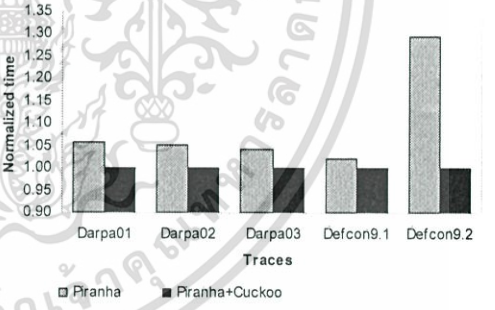


Figure 6: Normalized average matching time of Piranha and Piranha+Cuckoo algorithm in 5 packet traces.

In summary, the %SpeedUp depends on the nature of each packet trace. The 2.20%SpeedUp of Defcon9.1 trace is due to lower Difference of sequences matched by Piranha+Cuckoo but rather high number of Alerts detected. On the contrary, the 29.58%SpeedUp of Defcon9.2 trace is due to higher Difference but rather low number of Alerts detected. That means Piranha+Cuckoo is more efficient than Piranha by matching less but still able to detect the same number of Alerts as Piranha.

5. Conclusions and Future Works

In this paper, we presented a new pattern matching based on Cuckoo hashing of Snort NIDS. The algorithm is implemented on top of Piranha [3], which is almost optimal in terms of speed and memory resource compared with many famous existing ones [3]. Experiment results show that ours can significantly improve the matching speed of Piranha itself at up to 30% for current and future attack patterns.

For future works, we shall examine memory requirements of Piranha+Cuckoo Snort and improve our hashing function to support Gigabit networks.

6. Acknowledgment

I would like to thank Mr. Thinh Tran Ngoc for his support during his Doctoral study at KMITL.

7. References

[1] M. Roesch, "*Snort: Lightweight intrusion detection for networks*" In Proceedings of the 1999 USENIX LISA Systems Administration Conference, November 1999.http://www.snort.org/.

[2] M. Fisk and G. Varghese, "*An analysis of fast string matching applied to content-based forwarding and intrusion detection, Technical Report CS2001-0670 (updated version)*", University of California - San Diego, 2002.

[3] S. Antonatos, M. Polychronakis, P. Akritidis, Kostas D. Anagnostakis, and Evangelos P. Markatos: *"Piranha: Fast and Memory-efficient Pattern Matching for Intrusion Detection"*. In Proceedings of the 20th IFIP International Information Security Conference (IFIP/SEC 2005), May 2005

[4] R. Pagh and F.F. Rodler, "*Cuckoo hashing*". In ESA: Annual European Symposium on Algorithms, volume 2161 of LNCS. Springer, 2001.

[5] Rasmus Pagh. "*On the Cell Probe Complexity of Membership and Perfect Hashing*". In Proceedings of the 33rd Annual ACM Symposium on Theory of Computing (STOC '01), pages 425–432. ACM Press, 2001.

ประวัติผู้เขียน

ชื่อ-นามสกุล	นาย พรชัย กอประเสริฐถาวร
วัน เดือน ปีเกิด	15 พฤษภาคม 2527
การศึกษา	ปีการศึกษา 2539-2541 ระดับมัธยมศึกษาตอนต้น
	โรงเรียนศรัทธาสมุทร จังหวัดสมุทรสงคราม
	ปีการศึกษา 2542-2544 ระดับมัธยมศึกษาตอนปลาย
	โรงเรียนศรัทธาสมุทร จังหวัดสมุทรสงคราม
	ปีการศึกษา 2545-2548 ระดับปริญญาตรี
	วิศวกรรมศาสตรบัณฑิตสาขาวิชาวิศวกรรมคอมพิวเตอร์
	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง