

ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์
INTRUSION DETECTION SYSTEM



นายศุภกร รวยวาสนา
นายอานนท์ ลิ้มสิทธิรัตน์

5

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2545

เลขหมู่.....
เลขทะเบียน.....49909

b.....

เอกสารนี้เป็นเอกสารสงวนลิขสิทธิ์สำหรับการศึกษานี้ ไม่อนุญาตให้ทำซ้ำโดยไม่ได้รับอนุญาต
วัน เดือน ปี 22 ธ.ค. 2547
ไม่ว่ากรณีใดๆ ทั้งสิ้น ยกเว้นหากมีให้คัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีกา

ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์
INTRUSION DETECTION SYSTEM



โดย
นายสุกกร รวยวาสนา
นายอานนท์ ลิ้มสถิรนนท์

อาจารย์ที่ปรึกษา
อ.ธนา หงษ์สุวรรณ
อ.อัศรเดช วัชรภูพงษ์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2545

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ปีการศึกษา 2545

ภาควิชา วิศวกรรมคอมพิวเตอร์

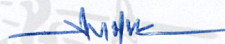
คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์

INTRUSION DETECTION SYSTEM

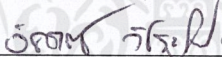
ผู้จัดทำ

- นายสุภกร รวยวาสนา รหัสประจำตัว 42010349
- นายอานนท์ ลิ้มสลิรานั้น รหัสประจำตัว 42010442



อาจารย์ที่ปรึกษา

(อ.ธนา หงษ์สุวรรณ)



อาจารย์ที่ปรึกษา

(อ.อัศรเดช วัชรภพพงษ์)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์

นายสุภกร	รวยวาสนา
นายอานนท์	ลิมสถิรานันท์
อ.ธนา	หงษ์สุวรรณ อาจารย์ที่ปรึกษา
อ.อักรเดช	วัชรภูพงษ์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2545	

บทคัดย่อ

ระบบปฏิบัติการลินุกซ์เป็นระบบปฏิบัติการที่เป็นที่รู้จักทั่วโลกซึ่งมีอัตราการเจริญเติบโตสูงและมีแนวโน้มที่จะถูกนำมาใช้อย่างแพร่หลายในอนาคต ลินุกซ์นั้นได้ถูกผ่านกระบวนการพัฒนามาเป็นเวลายาวนานโดยโปรแกรมเมอร์ทั่วโลกผ่านทางเครือข่ายอินเทอร์เน็ต แต่ลินุกซ์ยังมีปัญหาด้านความน่าเชื่อถือของระบบเนื่องมาจากเป็นซอฟต์แวร์ฟรี และเปิดเผยซอร์สโค้ดที่สามารถดาวน์โหลดได้ทางอินเทอร์เน็ต ดังนั้นลินุกซ์จึงยังมีความปลอดภัยไม่เพียงพอสำหรับการใช้งานในองค์กรที่ต้องการความปลอดภัยสูง

ปริญญาพนธ์ฉบับนี้จัดทำขึ้นเพื่อสร้างซอฟต์แวร์ตรวจจับผู้บุกรุกวิธีตรวจจับการบุกรุกที่ระบบรู้จักจากการเรียกใช้ฟังก์ชันที่ไม่ปลอดภัยในสแตนด์ดาร์ดซีไลบรารีฟังก์ชัน ซึ่งจะไม่มีการตรวจสอบขนาดของตัวแปรก่อนการเรียกใช้ โดยซอฟต์แวร์ทำการตรวจสอบขนาดของสแต็กถ้าเกิดโอเวอร์โฟลว์ขึ้นจะทำการบันทึกข้อความลงล็อกไฟล์ และแจ้งเตือนผู้ดูแลระบบ

Intrusion Detection System

Suprokorn	Luaywadsana	
Arnon	Limsatiranan	
Mr. Thana	Hongsuwan	Advisor
Mr. Akkradach	Watcharapupong	Advisor

ABSTRACT

Linux is well-known operating system which has high growth rate and trends to be used morewidely in the future. It is developed by world wide programmer for a long time though internet. But the problem is its reliability because it is free opened source code software which can download from internet. Consequently Linux does not have enough security for organization that need high security.

The objective of this thesis is to build software which is able to detect intruder by using misuse detection method by detecting unsave fuctions in standard C library which does not detect a boundary of variable before is used. This software will detect a boundary of stack. If buffer overflow happened, it will write information to log files and report to system administrator.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้ไม่สามารถเสร็จสมบูรณ์ได้หากไม่ได้รับคำแนะนำ คำเตือน ควบคุม เอาใจใส่ ช่วยเหลือและดูแลการทำปริญญานิพนธ์นี้อย่างใกล้ชิดอย่างสูงจากอาจารย์ธนา หงษ์สุวรรณและอาจารย์อัศรเดช วัชรภูปงษ์ นอกจากนี้ท่านอาจารย์ทั้งสองยังเป็นกำลังใจและแรงผลักดันให้ปริญญานิพนธ์ฉบับนี้เสร็จทันตามกำหนด สามารถเผยแพร่สู่สายตาสาธารณชนได้อย่างภาคภูมิใจ คณะผู้จัดทำขอขอบพระคุณยิ่งสำหรับทุกสิ่งทุกอย่างเป็นอย่างสูงไว้ ณ ที่นี้

นอกจากนี้ต้องขอขอบพระคุณอาจารย์ทุกท่านในสถาบันนี้ที่ได้สอนสั่งคณะผู้จัดทำจนมีความรู้ความสามารถจนถึงทุกวันนี้ รวมทั้งคณาจารย์ทุกท่านในภาควิชาวิศวกรรมคอมพิวเตอร์ที่ทำให้คณะผู้จัดทำได้เป็นวิศวกรคอมพิวเตอร์อย่างเต็มภาคภูมิ

ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์โดยเฉพาะห้องวิจัยและพัฒนาการรักษาความปลอดภัยข้อมูล (Information Security Advisory Group : ISAG) ที่ได้เอื้อเฟื้อสถานที่ให้คณะผู้จัดทำได้ทำการวิจัยและช่วยอำนวยความสะดวกต่าง ๆ

ขอขอบคุณพี่ต่อ, พี่ก่อ, พี่เซวี่, บอมบี้, เสือ, นัตตี้, แม็ก, เก่ง, นนท์, วงศ์, เบงค์ และบุคคลอื่นๆ ที่ไม่ได้กล่าวถึง ณ ที่นี้ ซึ่งอาจเกิดจากความหลงลืมของทางคณะผู้จัดทำนั้น ขอขอบคุณที่ให้ความช่วยเหลือคณะผู้จัดทำในการทำงานตลอดเวลา ให้คำปรึกษาในยามที่เกิดปัญหาขึ้น ให้ความรู้และสนับสนุนอุปกรณ์ในการทำงานต่าง ๆ ขอขอบคุณชุมชนประชาสัมพันธ์คณะวิศวกรรมศาสตร์ที่สร้างให้เติบโตมาขนาดนี้เรารักมากนะ

ประการสุดท้ายที่สำคัญที่สุด คือ ขอขอบพระคุณบิดา มารดาที่ได้ให้กำเนิด อบรม สั่งสอน และสนับสนุนการศึกษาและเป็นกำลังใจให้คณะผู้จัดทำอย่างดีมาโดยตลอด นับเป็นพระคุณที่เปรียบมิได้ ทางคณะผู้จัดทำขอขอบพระคุณและจรรโลงพระคุณของท่านไว้ตลอดไป

ศุภกร รวยวาสนา
อานนท์ ลัมสธิรานันท์

สารบัญ

	หน้าที่
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญรูป	VI
สารบัญตาราง	VIII
บทที่ 1 บทนำ	1
1.1 ความสำคัญและที่มา	1
1.2 วัตถุประสงค์	1
1.3 ขอบเขตของงานวิจัย	1
1.4 วิธีการดำเนินงาน	2
บทที่ 2 ทฤษฎีและหลักการเบื้องต้น	3
2.1 ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์(Intrusion Detection System)	3
2.1.1 รูปแบบของระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์	3
2.1.2 การทำงานของระบบตรวจจับผู้บุกรุก	5
2.2 ความรู้พื้นฐานเกี่ยวกับการบุกรุก	5
2.3 ระบบปฏิบัติการลินุกซ์(Linux)	8
2.3.1 โครงสร้างของระบบปฏิบัติการลินุกซ์	8
2.3.2 โครงสร้างเกี่ยวกับระบบไฟล์และไดเรกทอรี	9
2.3.3 คำสั่งพื้นฐานที่ใช้ในลินุกซ์/ยูนิกซ์	10
2.4 ความรู้พื้นฐานเกี่ยวกับการดูแลระบบ	11
2.5 เชล(Shell)	12
2.6 ไฟล์เพอร์มิสชันและไดเรกทอรีเพอร์มิสชัน	13
(File Permission and Directory Permission)	
2.7 การบันทึกเหตุการณ์บนระบบ(Logging)	18
2.7.1 การบันทึกเหตุการณ์ของลือกระบบ	18
2.7.2 แมสเสจคิว	20
2.7.3 การทำงานกับไฟล์	23
บทที่ 3 สแตนด์ออลไลบรารี	26
3.1 ไลบรารีที่ใช้งานทั่วไป	26
3.2 ไลบรารีที่ใช้งานร่วมกัน	27

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ(ต่อ)

	หน้าที่
3.3 ความหมายของสแตนด์คาร์ดซีไลบรารี	28
3.4 ฟังก์ชันที่ไม่ปลอดภัยในสแตนด์คาร์ดซีไลบรารี	29
บทที่ 4 โพรเซสเมโมรีและบัฟเฟอร์โอเวอร์โฟลว์	30
4.1 โครงสร้างของโพรเซสเมโมรี	30
4.2 บัฟเฟอร์ และการเกิดบัฟเฟอร์โอเวอร์โฟลว์	31
4.3 การโจมตีด้วยวิธีการบัฟเฟอร์โอเวอร์โฟลว์	33
4.4 การแก้ปัญหาการเกิดบัฟเฟอร์โอเวอร์โฟลว์	37
บทที่ 5 โครงสร้างการทำงานของระบบ	38
5.1 ภาพรวมและโครงสร้างของระบบตรวจจับผู้บุกรุก	38
5.2 ส่วนการโอเวอร์ไรด์ไลบรารีฟังก์ชัน	39
5.3 ส่วนการวิเคราะห์ข้อมูลหรือส่วนการตรวจสอบขนาดของสแตก	42
5.4 ส่วนการตอบสนองของระบบ	44
บทที่ 6 ทดสอบการทำงานและสรุป	47
6.1 ทดสอบการทำงานเบื้องต้นของโปรแกรม	47
6.2 ทดสอบการทำงานกับการโจมตีแบบบัฟเฟอร์โอเวอร์โฟลว์	53
6.3 ทดสอบประสิทธิภาพของระบบ	59
6.4 วิเคราะห์การทำงานของระบบ	60
6.5 สรุป	60
6.6 แนวทางในการพัฒนาต่อ	61
บรรณานุกรม	62

สารบัญรูป

	หน้าที่
รูปที่ 2.1 การทำงานของการตรวจจับผู้บุกรุกโดยวิธีเปรียบเทียบพฤติกรรมผู้ใช้ กับวิธีการบุกรุกที่ระบบรู้จัก	3
รูปที่ 2.2 การทำงานของการตรวจจับผู้บุกรุกโดยวิธีเปรียบเทียบพฤติกรรมผู้ใช้ที่ผิดปกติ	4
รูปที่ 2.3 แสดงการเพิ่มข้อมูลเข้าไปในแมสเสจคิว ของฟังก์ชัน msgsnd	22
รูปที่ 3.1 การลิงก์แบบสแตติก	26
รูปที่ 3.2 การลิงก์แบบไดนามิก	27
รูปที่ 4.1 โครงสร้างของโพรเซสเมโมรี	30
รูปที่ 4.2 บัฟเฟอร์ “wxy” ในหน่วยความจำ	31
รูปที่ 4.3 บัฟเฟอร์ “wxy” และ “abcde” ในหน่วยความจำ	31
รูปที่ 4.4 เนื้อที่ที่ไม่ได้ใช้	31
รูปที่ 4.5 โครงสร้างสแต็กเริ่มต้น	32
รูปที่ 4.6 ภายหลังการเกิดโอเวอร์โฟลว์	32
รูปที่ 4.7 ลักษณะของสแต็กก่อนการโจมตี	34
รูปที่ 4.8 ลักษณะของข้อมูลสแต็กที่เกิดการโอเวอร์โฟลว์แล้ว	35
รูปที่ 4.9 ลักษณะการรันของโปรแกรมเมื่อได้ทำการโอเวอร์โฟลว์แล้ว	36
รูปที่ 4.10 ลักษณะการป้องกันการเกิดโอเวอร์โฟลว์	37
รูปที่ 5.1 โครงสร้างและภาพรวมของโปรแกรม	38
รูปที่ 5.2 แสดงการโหลดไลบรารี	40
รูปที่ 5.3 แสดงการยกเลิกการ โหลดไลบรารี	40
รูปที่ 5.4 การทำงานโปรแกรมต่างๆเมื่อทำการโหลดไลบรารีของเราเข้าไปแล้ว	41
รูปที่ 5.5 แสดงวิธีการหาขนาดของสแต็ก	42
รูปที่ 5.6 ขั้นตอนการทำงานของส่วนวิเคราะห์ข้อมูลหรือส่วนการตรวจสอบขนาดของสแต็ก	43
รูปที่ 5.7 ขั้นตอนการทำงานของส่วนของการตอบสนองของระบบ	44
รูปที่ 5.8 แสดงแมสเสจคิวที่มารอรับเพื่อที่จะได้เขียนลงไปนีสต็อกไฟล์	45
รูปที่ 5.9 แสดงผลในล็อกไฟล์ที่เกิดขึ้น	46
รูปที่ 6.1 การเซตค่า s_level ในไฟล์ log.h ให้เป็น 2	47
รูปที่ 6.2 แสดงการใช้คำสั่ง ls เมื่อทำการโหลดไลบรารีเรียบร้อยแล้ว	48
รูปที่ 6.3 แสดงการเรียกใช้โปรแกรม test ซึ่งใช้ในการทดสอบการทำงานของไลบรารี	49
รูปที่ 6.4 แสดงการเรียกทดสอบฟังก์ชัน strcpy()	49
รูปที่ 6.5 แสดงการทดสอบฟังก์ชัน strcat()	50

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญรูป(ต่อ)

	หน้าที่
รูปที่ 6.6 แสดงการเข้าไปในไดเรกทอรีที่มีชื่อยาวมากเพื่อทดสอบฟังก์ชัน getwd	51
รูปที่ 6.7 แสดงการทดสอบฟังก์ชัน getwd() เกิดการโอเวอร์โฟลว์ที่ตัวแปร buf	51
รูปที่ 6.8 แสดงการทดสอบฟังก์ชัน gets()	52
รูปที่ 6.9 แสดงการทดสอบฟังก์ชัน realpath()	53
รูปที่ 6.10 การสำรวจไฟล์ /usr/bin/efstool และ การรันโปรแกรมโจมตี	54
รูปที่ 6.11 การทำงานของโปรแกรมที่ใช้โจมตี.	55
รูปที่ 6.12 การได้มาซึ่งเชลของผู้ดูแลระบบ	55
รูปที่ 6.13 การเปิดไฟล์ /etc/ld.so.preload	56
รูปที่ 6.14 เขียนที่อยู่ของไลบรารีของโปรแกรมลงในไฟล์ ld.so.preload	57
รูปที่ 6.15 แสดงการไม่ได้ผลใหม่ในการโจมตีระบบ	57
รูปที่ 6.16 แสดงการรับแมสเสจคิวจากไลบรารีของโปรแกรม	58
รูปที่ 6.17 แสดงข้อมูลที่อยู่ในล็อกไฟล์ของระบบ	58

สารบัญตาราง

	หน้าที่
ตารางที่ 2.1 แสดงคำสั่งพื้นฐานที่ใช้ในลินุกซ์/ยูนิกซ์	11
ตารางที่ 2.2 เซลชนิดต่างๆ	12
ตารางที่ 2.3 แสดงตัวแปรกำหนดการทำงานของเซลล์พื้นฐาน	13
ตารางที่ 2.4 แสดงพารามิเตอร์ของเซลล์	13
ตารางที่ 2.5 รายละเอียดในไฟล์ /etc/passwd	14
ตารางที่ 2.6 แสดงรายละเอียดของไฟล์ /etc/groups	14
ตารางที่ 2.7 ความหมายของฟิลด์เมื่อใช้คำสั่ง ls	16
ตารางที่ 2.8 กลุ่มของ permission	16
ตารางที่ 2.9 คำอธิบายเพิ่มเติม permission ในกลุ่มต่างๆ	16
ตารางที่ 2.10 แสดงชนิดของไฟล์	17
ตารางที่ 2.11 แสดง File permission	17
ตารางที่ 2.12 แสดง Directory permission	18
ตารางที่ 3.1 ตัวอย่างของฟังก์ชันที่ไม่ปลอดภัยในสแตนด์ออลไลบรารี	29
ตารางที่ 6.1 แสดงเวลาการทำงานของคำสั่ง ls -la 5000 ครั้ง	60
ตารางที่ 6.2 แสดงเวลาการทำงานของคำสั่ง perl -e 'print "A"x100' 500 ครั้ง	60

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันคอมพิวเตอร์ได้ถูกนำเข้ามาใช้อย่างแพร่หลาย ทั้งในด้านการเรียน การทำงาน การทำธุรกิจ ก็จะมีการใช้คอมพิวเตอร์เข้าเกี่ยวข้อง ข้อมูลต่างๆ ที่เก็บรักษาไว้อยู่ในเครื่องคอมพิวเตอร์มีทั้งที่สำคัญและไม่สำคัญ การป้องกันรักษาข้อมูลหรือป้องกันความเสียหายของระบบจากผู้ที่ทำกรบุกรุกเป็นสิ่งที่จะต้องเสียไม่ได้ เนื่องจากผู้ดูแลระบบนั้นไม่สามารถที่จะทำการเฝ้าทำการติดตามผู้บุกรุกระบบคอมพิวเตอร์ได้ตลอดเวลา จึงได้มีการใช้ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์เข้ามาช่วยดูแลระบบในการติดตามการกระทำที่บ่งบอกถึงการบุกรุกระบบ โดยระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์จะช่วยผู้ดูแลระบบในการดูแลระบบให้มีประสิทธิภาพโดยที่ผู้ดูแลระบบไม่จำเป็นต้องมาเฝ้าดูแลระบบด้วยตนเองตลอดเวลา

1.2 วัตถุประสงค์

1. สร้างระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์บนระบบปฏิบัติการลินุกซ์ โดยใช้วิธีการตรวจสอบวิธีการบุกรุกที่ระบบรู้จักจากการตรวจสอบการเรียกใช้งานฟังก์ชันในสแตนด์ดาร์ดซีไลบรารีที่ทำการโอเวอร์โฟลว์
2. ศึกษาการทำงานของระบบปฏิบัติการลินุกซ์ เพื่อเป็นพื้นฐานในการพัฒนาโครงการและการใช้งานในด้านอื่นๆ
3. ศึกษาการเขียนแอปพลิเคชันบนระบบปฏิบัติการลินุกซ์ ซึ่งนอกเหนือจากการพัฒนาโปรแกรมในโครงการจะทำให้สามารถนำไปใช้พัฒนาแอปพลิเคชันในด้านอื่นๆ ได้

1.3 ขอบเขตของงานวิจัย

โครงการนี้จะทำการสร้างระบบตรวจจับการบุกรุกโดยการตรวจจับการเรียกใช้ฟังก์ชันในสแตนด์ดาร์ดซีไลบรารี โดยจะตรวจว่ามีการเรียกใช้ฟังก์ชันที่อาจก่อให้เกิดการโอเวอร์โฟลว์ โดยฟังก์ชันที่อาจก่อให้เกิดการโอเวอร์โฟลว์นี้จะมียูติลิตี้กันหลายตัว ซึ่งวิธีการตรวจสอบนั้นก็แตกต่างกันไป โดยโครงการจะทำการ ตรวจจับฟังก์ชันจำนวนสิบตัว คือ strcpy(), strncpy(), strcat(), strncat(), getwd(), gets(), stpcpy(), memcpy(), realpath(), sprintf() เพื่อแสดงแนวความคิดในการสร้างระบบตรวจจับจากการทำให้เกิดโอเวอร์โฟลว์ในการเรียกใช้ฟังก์ชันในสแตนด์ดาร์ดซีไลบรารี ซึ่งผู้สนใจสามารถนำแนวคิดนี้ไปพัฒนาต่อเพื่อเพิ่มประสิทธิภาพในการตรวจจับฟังก์ชันได้มากขึ้น

อีกทั้งโครงการนี้จะไม่สนใจในเรื่องของเทคนิคในการสร้างโค้ดบุกรุกของผู้บุกรุกและความสามารถในการจัดการการเกิดโอเวอร์โฟลว์ของระบบปฏิบัติการเพราะถ้าผู้บุกรุกมีความชำนาญในการสร้างโค้ดโจมตีก็จะสามารถผ่านระบบปฏิบัติการโจมตีได้ โดยเราจะสนใจการเรียกใช้ฟังก์ชันในสแตนด์ดาร์ดซีไลบรารี

1.4 วิธีการดำเนินงาน

โครงการนี้จะเริ่มด้วยการศึกษาทฤษฎีพื้นฐานต่าง ๆ ที่เกี่ยวข้องซึ่งก็จะมีเรื่องเกี่ยวกับระบบตรวจจับผู้บุกรุก วิธีการบุกรุก โครงสร้างของระบบปฏิบัติการลินุกซ์ เทคนิคการเขียนโปรแกรม จากนั้นก็จะคิดหาวิธีที่จะเลือกใช้ในการตรวจสอบที่จะสร้างระบบตรวจจับผู้บุกรุกขึ้นมา เมื่อสามารถเลือกวิธีได้แล้วก็จะนำความพื้นฐานที่ได้ศึกษามาประยุกต์ใช้กับวิธีการที่ได้เลือกเพื่อออกแบบสถาปัตยกรรมของระบบโดยช่วงนี้ก็จะมีการศึกษาเพิ่มในเรื่องที่เกี่ยวข้องกับวิธีการที่ได้เลือกมา

เมื่อออกแบบเสร็จก็มาถึงในช่วงของการพัฒนาโปรแกรมโดยในส่วนขององค์ประกอบโดยรวมของระบบที่พัฒนาขึ้นมาทั้งหมดจะแสดงในบทที่ 5 และบทที่ 6 ซึ่งจะแสดงการทดสอบการทำงานของโปรแกรม สรุปการทำงาน และแนวทางในการพัฒนางานวิจัยนี้เพิ่มเติม



บทที่ 2

ทฤษฎีและหลักการเบื้องต้น

2.1 ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์

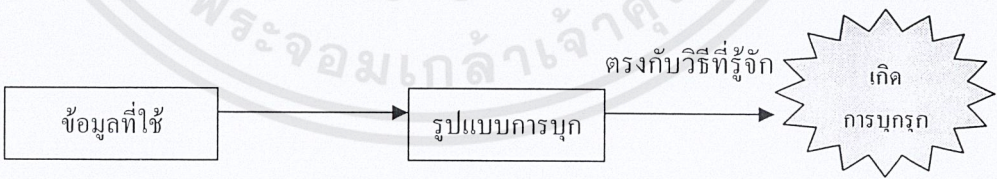
ระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์ คือ ระบบที่ทำหน้าที่ติดตามดูการทำงานที่เกิดขึ้นบนระบบคอมพิวเตอร์หรือระบบเครือข่าย เพื่อค้นหาร่องรอยที่บ่งบอกว่ามีผู้กำลังพยายามบุกรุก หรือค้นหาการกระทำที่เกินขอบเขตสิทธิของผู้ใช้ระบบ

ผู้ดูแลระบบบางคนอาจคิดว่าระบบที่ตนดูแลอยู่ข้อมูลไม่ได้มีความสำคัญอะไร ถึงจะโดนบุกรุกก็ไม่เป็นไร แต่นั่นอาจจะทำให้เกิดปัญหาใหญ่ได้เมื่อผู้บุกรุกใช้ระบบเป็นเส้นทางของการโจมตีระบบอื่นต่อ เมื่อมีการตรวจสอบกลับมา เจ้าของระบบจะต้องเป็นผู้รับผิดชอบอย่างหลีกเลี่ยงไม่ได้ ดังนั้นการรักษาให้ระบบของตนมีความปลอดภัยจึงเป็นสิ่งจำเป็นอย่างยิ่ง

2.1.1 รูปแบบของระบบตรวจจับผู้บุกรุกระบบคอมพิวเตอร์

เป็นวิธีหลักๆที่ใช้ในการป้องกันการบุกรุก คือ วิธีเปรียบเทียบพฤติกรรมผู้ใช้กับวิธีการบุกรุกที่ระบบรู้จัก

วิธีเปรียบเทียบพฤติกรรมผู้ใช้กับวิธีการบุกรุกที่ระบบรู้จัก เป็นวิธีการที่ระบบจะทำการเปรียบเทียบพฤติกรรมการใช้งานของผู้ใช้ว่าตรงกับรูปแบบการบุกรุกที่ระบบรู้จักหรือไม่ โดยบางระบบมีการใช้กฎ (Rule-base expert system) โดยตั้งกฎจากพฤติกรรมที่น่าสงสัย เช่น ถ้าทำการล็อกอินล้มเหลวเกิน 3 ครั้ง ต่อเนื่องกันในเวลาที่กำหนด ถือว่าพยายามบุกรุก



รูปที่ 2.1 การทำงานของการตรวจจับผู้บุกรุกโดยวิธีเปรียบเทียบพฤติกรรมผู้ใช้กับวิธีการบุกรุกที่ระบบรู้จัก

จากรูปข้อมูลที่ใช้ตรวจสอบจะถูกนำมาเปรียบเทียบกับวิธีการบุกรุกที่ระบบรู้จัก ถ้าตรงกับวิธีการบุกรุกที่ระบบรู้จักก็จะแจ้งตรวจพบการบุกรุก

จุดอ่อนของวิธีเปรียบเทียบพฤติกรรมผู้ใช้กับวิธีการบุกรุกที่ระบบรู้จัก คือ

1. ประสิทธิภาพของระบบตรวจจับชนิดนี้จะขึ้นอยู่กับรูปแบบการบุกรุกที่ระบบรู้จัก
2. ถ้าเป็นรูปแบบการบุกรุกที่ระบบไม่รู้จักก็จะไม่สามารถตรวจสอบได้

วิธีตรวจสอบการใช้งานที่ผิดปกติ

วิธีตรวจสอบการใช้งานที่ผิดปกติ เป็นวิธีตรวจจับผู้บุกรุกโดยจะตรวจสอบจากพฤติกรรมการทำงานของผู้ใช้ที่ผิดปกติไปจากที่เคยเป็นอยู่โดยการเปรียบเทียบจากพฤติกรรมการทำงานก่อนหน้านี้ที่เคยเก็บไว้ในประวัติผู้ใช้และตั้งสมมติฐานว่าพฤติกรรมการทำงานที่ผิดปกติไปจากเดิมคือการบุกรุก



รูปที่ 2.2 การทำงานของการตรวจจับผู้บุกรุกโดยวิธีเปรียบเทียบพฤติกรรมผู้ใช้ที่ผิดปกติ

จากรูปจะนำข้อมูลที่ใช้ตรวจคือพฤติกรรมของผู้ใช้ทำการเปรียบเทียบกับสถิติการใช้งานของผู้ใช้แต่ละคน หากพบว่ามีความผิดปกติก็จะระบุว่าอยู่ในสถานะมีการบุกรุกเกิดขึ้น

จุดอ่อนของวิธีตรวจสอบการใช้งานที่ผิดปกติ คือ

1. พฤติกรรมบางอย่างที่ผิดปกติอาจไม่ใช่การบุกรุก ทำให้ระบบตรวจจับผิดพลาด
2. การตรวจจับต้องใช้เวลาในการเปรียบเทียบความผิดปกติ โดยจะไม่สามารถตรวจจับทันทีที่มีการบุกรุกได้
3. ถ้าใช้วิธีการค่อยๆเปลี่ยนแปลงพฤติกรรมการใช้งานที่ละเอียดละน้อยระบบจะไม่สามารถตรวจจับความผิดปกติได้

2.1.2 การทำงานของระบบตรวจจับผู้บุกรุก

ระบบตรวจจับผู้บุกรุกแต่ละแบบก็จะมีหน้าที่แตกต่างกันไป บางตัวจะตรวจความผิดปกติในเครือข่าย บางตัวจะตรวจความผิดปกติในระบบฐานข้อมูล แต่โดยการทำงานทั้งหมดแล้ว ระบบตรวจจับผู้บุกรุกจะมีการทำงานอยู่ 3 ขั้นตอนด้วยกันคือ

1. การเก็บข้อมูลระบบ
2. การวิเคราะห์ข้อมูลที่เก็บได้
3. การตอบสนองและรายงานผล

2.2 ความรู้พื้นฐานเกี่ยวกับการบุกรุก

สำหรับบุคคลต่างๆ ที่พยายามจะเจาะเข้าสู่ระบบของเรา เราจะเรียกว่า ผู้บุกรุก "intruder" ซึ่งผู้บุกรุกจะถูกแบ่งเป็น 2 ประเภทคือ

1. Outsides--จากภายนอก

หมายถึง ผู้บุกรุกจากภายนอกเครือข่ายของท่าน และบุคคลที่อาจจะโจมตีมาจากภายนอกเช่น การเปลี่ยนแปลงหน้าภาพของเว็บเซิร์ฟเวอร์ (Web Server) ของท่าน หรือการ ส่งเมลล์ผ่านทางอี-เมลล์ เซิร์ฟเวอร์ (e-mail server) ซึ่งการบุกรุกจากภายนอกอาจมาจากอินเทอร์เน็ต, การเคิล-อัพ (dial-up), การบุกเข้าไปเครือข่ายของลูกค้าที่เชื่อมต่อมายังเครือข่ายของท่าน

2. Insider--จากภายใน

คือ ผู้บุกรุกที่มีสิทธิ์ในการใช้เครือข่ายภายใน รวมทั้งผู้ใช้ที่ใช้สิทธิ์ในทางที่ผิด หรือการลักลอบใช้สิทธิ์ของผู้ใช้คนอื่นๆ ที่มีสิทธิ์เหนือกว่า

เส้นทางที่ผู้บุกรุกจะเข้าสู่ระบบมี 3 ทางหลักๆ ที่ผู้บุกรุกจะเจาะเข้าสู่ระบบ คือ

1. Physical Intrusion ถ้าหากว่าผู้บุกรุกมีการเชื่อมต่อทางกายภาพกับเครื่องหรือ ระบบเครือข่าย การเจาะเข้าสู่ระบบจะเกิดขึ้นได้
2. System Intrusion การบุกรุกในลักษณะนี้สมมติว่าผู้บุกรุกมีรหัสผ่านเรียบร้อยแล้วแต่มีสิทธิ์ต่ำ ถ้าไม่ได้มีการอัปเดตแพทช์ให้กับระบบ ผู้บุกรุกจะใช้ช่องโหว่ของระบบในการครอบครองสิทธิ์ของผู้ดูแลระบบ
3. Remote Intrusion ผู้บุกรุกพยายามที่จะเจาะเข้าสู่ระบบข้ามเครือข่าย ซึ่งผู้บุกรุกจะไม่มีสิทธิ์ใดๆ เลยบนเครือข่ายนั้น

สาเหตุที่อาจเป็นต้นเหตุทำให้ผู้บุกรุกสามารถเข้ามาในระบบได้ คือ

ความบกพร่องของโปรแกรม (Software bugs)

จะปรากฏอยู่ในเซิร์ฟเวอร์ เดมอน (server daemons), โปรแกรมต่างๆ ของไคลเอ็นต์, ระบบปฏิบัติการ ซึ่งสามารถแบ่งความบกพร่องของโปรแกรม ออกเป็นประเภทต่างๆ ได้ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1. Buffer Overflow

ช่องโหว่ของความปลอดภัยคอมพิวเตอร์จะเกิดจากปัญหานี้ ตัวอย่างของความบกพร่องของโปรแกรม เช่น โปรแกรมเมอร์ได้ทำการตั้งค่าจำนวนคาเร็กเตอร์ ที่จะรับล็อกอินชื่อผู้ใช้เป็น 256 เนื่องจากเขาคิดว่าคงจะไม่มีใครที่จะใช้ชื่อผู้ใช้ที่ยาวกว่า 256 คาเร็กเตอร์ ตัวอย่างเช่น 300 คาเร็กเตอร์ จะเกิดอะไรขึ้นกับอีก 50 คาเร็กเตอร์ ที่เหลือ ซึ่งอาจประกอบด้วยโค้ดที่จะถูกเอ็กคิวต์โดยเชิร์ฟเวอร์ และทำให้ผู้บุกรุกจะเข้าสู่ระบบได้ ผู้บุกรุกจะทำการค้นความผิดพลาดของโปรแกรมเหล่านี้โดยวิธีต่างๆ เช่น

- อาจจะค้นหาได้จากอินเทอร์เน็ต

- ผู้บุกรุกอาจจะทำการศึกษาโปรแกรมนั้นๆ โดยตัวเอง

- ผู้บุกรุกอาจจะตรวจสอบทุกๆ ส่วนของโปรแกรม ที่มีอินพุต และพยายามที่จะโอเวอร์โฟลว์โดยใส่ค่าสุ่ม เข้าไป

ซึ่งถ้าโปรแกรมนั้นหยุดการทำงานอาจจะทำให้ผู้บุกรุกสามารถจะเข้าสู่ระบบนั้นได้ ซึ่งปัญหาดังกล่าวจะเกิดขึ้นกับโปรแกรม ซึ่งเขียนด้วยซี/ซี++ แต่จะไม่เจอกับโปรแกรมซึ่งเขียนด้วยจาวา

2.Unexpectes Combinations

โดยทั่วไปโปรแกรมจะประกอบโค้ด หลายๆ เลเยอร์ ซึ่งเลเยอร์ที่อยู่ต่ำที่สุดคือ ระบบปฏิบัติการ ซึ่งเมื่อผู้บุกรุกส่งอินพุตเข้าสู่โปรแกรมอินพุตนั้นอาจจะไม่มีความหมายอะไรเลยสำหรับเลเยอร์หนึ่ง แต่อาจจะทำให้เกิดผลกับอีกเลเยอร์หนึ่ง จะเห็นได้ว่าโดยส่วนใหญ่ ภาษาเพิร์ล(PERL) คือภาษาที่จะทำการโพรเซสอินพุตของผู้ใช้บนเว็บซึ่งปกติแล้วเพิร์ลจะส่งอินพุตไปให้อีกโปรแกรมหนึ่ง

3.อินพุตที่ไม่ถูกการประมวลผล

โดยส่วนใหญ่โปรแกรมเมอร์จะเขียนโปรแกรม ที่จะจัดการกับอินพุตที่ถูกต้องแต่จะไม่พิจารณากรณีที่อินพุตไม่ได้ตรงตามรูปแบบที่ออกแบบไว้ (specification)

4.Race Conditions

เนื่องจากปัจจุบันระบบส่วนใหญ่จะเป็น " Multitasking หรือ Multithreaded" หมายถึงว่า โปรแกรมหลายๆ โปรแกรมสามารถถูกเอ็กคิวต์ พร้อมๆ กันได้ ซึ่งจะเป็นอันตรายอย่างเช่น ถ้าโปรแกรมนั้นๆ ใช้ข้อมูลเดียวกัน ตัวอย่างเช่น โปรแกรม A และ B จำเป็นที่จะต้องแก้ไขไฟล์เดียวกัน ในการที่จะแก้ไขไฟล์นี้ โปรแกรมจะต้องอ่านไฟล์ไปเก็บไว้ในหน่วยความจำแล้วทำการแก้ไข เนื้อความในหน่วยความจำแล้วคัดลอกหน่วยความจำกลับเข้าสู่ไฟล์ race conditions จะเกิดขึ้นเมื่อ โปรแกรม A อ่านไฟล์เข้าสู่หน่วยความจำ แล้วทำการแก้ไข แต่ก่อนที่ A จะเขียนกลับลงสู่ไฟล์ โปรแกรม B ได้ทำการอ่าน / แก้ไข / เขียน ลงสู่ไฟล์เรียบร้อยแล้ว หลังจากนั้น A ทำการเขียนกลับไปในไฟล์ จะเห็นได้ว่าการแก้ไขโปรแกรมทั้งหมดของ โปรแกรม B จะหายไป แต่ยังงี้ก็แล้วแต่ race conditions ก่อนข้างจะเกิดขึ้นได้ยาก ผู้บุกรุกต้องทำการทดลองเป็นพันๆ ครั้ง จึงจะสามารถจะเข้าสู่ระบบได้

System Configuration

ข้อบกพร่องที่เกิดจาก System Configuration สามารถจำแนกได้ดังต่อไปนี้

1. Default Configurations

ระบบส่วนใหญ่จะถูกจัดส่งจากผู้ขายด้วย default configuration ซึ่งง่ายในการใช้ แต่นั่นก็หมายถึงง่ายในการเจาะด้วย

2. ความเกียจคร้านของผู้ดูแลระบบ

มีระบบอยู่ไม่น้อยทีเดียวที่ถูกคอนฟิกให้ไม่ต้องใส่รหัสผ่านของผู้ดูแลระบบ เนื่องจากผู้ดูแลระบบเกียจคร้านที่จะคอนฟิกแต่อยากให้อุปกรณ์ทำงานได้เร็วที่สุด เนื่องจากว่าการแก้ไขรหัสผ่านทีหลังทำได้ไม่ง่ายเลย ซึ่งจุดนี้อาจทำให้ผู้บุกรุกเจาะเข้าสู่ระบบได้

3. Hole Creation

บางครั้งผู้ดูแลระบบจะเปิดช่องโหว่บนเครื่องไว้ ซึ่งคู่มือการดูแลระบบส่วนใหญ่จะแนะนำให้ทำการปิดทุกอย่างที่ไม่จำเป็นเพื่อป้องกันการเกิดช่องโหว่ขึ้นมาโดยไม่ตั้งใจ

Password Cracking

1. การเลือกใช้รหัสผ่านที่ค่อนข้างง่ายต่อการบุกรุก

คนส่วนใหญ่จะเลือกใช้รหัสผ่านที่เป็นชื่อของตนเอง, ชื่อลูก, ชื่อสามี/ภรรยา, สัตว์เลี้ยง, รุ่นของรถ หรือบางคนอาจใช้เป็นรหัสผ่านหรืออาจไม่ใส่เลย จะเห็นว่าผู้บุกรุกทำการเดาสุ่มรหัสผ่าน เหล่านี้ได้ง่าย

2. Dictionary Attacks

ขั้นตอนต่อไปนี้ ผู้บุกรุกอาจจะใช้โปรแกรมที่จะทำการแคร็กรหัสผ่านโดยที่โปรแกรมดังกล่าวจะเลือกคำที่เป็นไปได้ในดิกชันนารี dictionary attacks แล้วเปรียบเทียบกับคำในดิกชันนารีที่ถูกเข้ารหัส

3. Brute force attacks

จะคล้ายกับ dictionary attacks ผู้บุกรุกจะพยายามนำคำเรีกรเตอร์ต่างๆ มาผสมกันเป็นรหัสผ่าน อย่างเช่น รหัสผ่านที่มีอักษร 4 ตัว และเป็นตัวเลข เพียงอย่างเดียว อาจจะถูกแคร็กภายในเวลาแค่ไม่กี่นาที หรือถ้าเป็นรหัสผ่านความยาว 7 ตัวอักษร ทั้งที่เป็นตัวใหญ่ และ ตัวเล็ก อาจต้องใช้เวลากหลายเดือนในการแคร็ก

Sniffing Unsecured traffic

สามารถทำได้หลายวิธีด้วยกัน

1. medium

ที่ใช้ร่วมกัน ตัวอย่างเช่น ถ้าเป็นอีเทอร์เน็ตรุ่นเก่าๆ เพียงแค่โปรแกรมที่ใช้ดักจับข้อมูลที่เข้ามาผ่านเครือข่ายคอมพิวเตอร์ (sniffer) ไปติดตั้งบนสายก็สามารถมองเห็นทราฟฟิกทั้งหมดได้

2. Server Sniffing

การติดตั้งโปรแกรมที่ใช้ดักจับข้อมูลที่ข้ามผ่านเครือข่ายคอมพิวเตอร์บนเซิร์ฟเวอร์นั้นอาจได้ข้อมูลที่จะนำไปใช้ในการเจาะเข้าสู่เครื่องไคลเอนต์ได้

ข้อบกพร่องของการออกแบบ

ถึงแม้ว่าการใช้ซอฟต์แวร์จะถูกต้องตามดีไซน์แต่อาจจะมีข้อบกพร่องในตัวดีไซน์ ที่จะนำไปสู่การเจาะระบบได้ เช่น

1. TCP/IP protocol

เนื่องจาก TCP/IP ได้ถูกออกมาก่อนที่วิวัฒนาการทางด้านการเจาะระบบจะเหมือนในปัจจุบัน เพราะฉะนั้นเป็นไปได้ที่อาจจะมีข้อผิดพลาดทางด้านการออกแบบที่จะนำไปสู่ปัญหาทางด้านความปลอดภัยได้ ตัวอย่างเช่น Smurt attack, IP spoofing หรือ SYN floods ได้มีการพัฒนา IPSec เพื่อที่จะแก้ปัญหาต่างๆเหล่านี้ แต่ยังไม่มีการใช้อย่างแพร่หลาย

2. Unix

มีช่องโหว่ในระบบปฏิบัติการยูนิกซ์ซึ่งอาจทำให้ระบบถูกเจาะได้ซึ่งปัญหาที่สำคัญที่สุดคือ access control system

2.3 ระบบปฏิบัติการลินุกซ์

ลินุกซ์เป็นระบบปฏิบัติการแบบ 32 บิตที่พัฒนามาจากระบบปฏิบัติการยูนิกซ์ที่แจกจ่ายฟรี พร้อมทั้งสนับสนุนการใช้งานแบบหลายงาน และแบบหลายผู้ใช้ มีระบบ X Windows ซึ่งเป็นระบบการติดต่อผู้ใช้แบบกราฟิกส์ที่ไม่ขึ้นกับโอเอสหรือฮาร์ดแวร์ใดๆ

ลินุกซ์มีความเข้ากันได้ (compatible) กับมาตรฐาน POSIX ซึ่งเป็นมาตรฐานอินเทอร์เฟซที่ระบบยูนิกซ์ส่วนใหญ่จะต้องมี และมีรูปแบบบางส่วนที่คล้ายกับระบบปฏิบัติการยูนิกซ์จากค่าย Berkeley และ System V โดยความหมายทางเทคนิคแล้วลินุกซ์เป็นเพียง "แก่น" หรือเคอร์เนล(kernel)ของระบบปฏิบัติการ ซึ่งจะทำหน้าที่ในด้านของการจัดสรรทรัพยากรในระบบ การประมวลผล การจัดการไฟล์ และอุปกรณ์ I/O ต่างๆ

ลินุกซ์มีทีมพัฒนาโปรแกรมที่ต่อเนื่อง ไม่จำกัดจำนวนของอาสาสมัครผู้ร่วมงาน ซึ่งส่วนใหญ่จะติดต่อกันผ่านทางอินเทอร์เน็ต และมีแผนการพัฒนาในระยะยาว ทำให้ลินุกซ์เป็นระบบปฏิบัติการที่มีการพัฒนาไปอย่างรวดเร็ว

2.3.1 โครงสร้างของระบบปฏิบัติการลินุกซ์

ระบบปฏิบัติการลินุกซ์เป็นระบบที่มีสภาพแวดล้อมการทำงานประกอบด้วย 3 ส่วนคือ ตัวระบบปฏิบัติการ ไลบรารีของระบบ และยูทิลิตี้กับการจัดการระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ตัวระบบปฏิบัติการ

ตัวระบบปฏิบัติการของลินุกซ์นั้น เราเรียกกันอีกอย่างว่า เคอร์เนล (kernel) ซึ่งทำหน้าที่หลักในการจัดการทรัพยากรต่างๆของระบบเช่น

- จัดการหน่วยความจำ
- จัดการคิวสำหรับโปรแกรมต่างๆ
- จัดการอุปกรณ์ต่างๆ ผ่านไดรเวอร์
- ระบบจัดการแฟ้มข้อมูล

ไลบรารีของระบบ

ไลบรารีเป็นที่รวบรวมฟังก์ชันมาตรฐานที่ใช้ในการติดต่อกับเคอร์เนล ทำให้โปรแกรมทั่วไปสามารถติดต่อกับระบบได้โดยผ่านทางฟังก์ชันมาตรฐานเหล่านี้ ซึ่งก็เหมือนกับในยูนิกซ์นั่นเอง เพียงแต่มีโครงสร้างภายในบางอย่างที่อาจแตกต่างออกไปตามความเหมาะสมของระบบ นอกจากนี้ลินุกซ์ยังสร้างไลบรารีของระบบตามข้อกำหนด POSIX (Portable Operating System Interface for Unix) ซึ่งเป็นข้อกำหนดในการเขียนฟังก์ชันสำหรับการติดต่อกับโอเอสอื่น ดังนั้นไม่ว่าโปรแกรมใดก็ตามที่สามารถใช้งานบนยูนิกซ์หรือ POSIX ได้ ก็จะสามารถใช้บนลินุกซ์ได้

ยูทิลิตี้กับการจัดการระบบ

ส่วนนี้มีโปรแกรมที่ทำหน้าที่จัดการระบบในส่วนต่างๆ เช่น ระบบไฟล์ ผู้ใช้งานระบบ โมดูลระบบรักษาความปลอดภัย ระบบเน็ตเวิร์ก เป็นต้น รวมถึง เดมอน(demons) ซึ่งเป็นโปรแกรมที่รองรับบริการต่างๆ รวมทั้งเครื่องมือสำหรับการพัฒนาโปรแกรม โดยเฉพาะ GNU C ซึ่งเป็นคอมไพเลอร์มาตรฐานบนลินุกซ์ โปรแกรมส่วนใหญ่ที่ทำงานบนลินุกซ์นั้นมักจะพัฒนาโดยใช้ GNU C แล้วแจกซอร์สโค้ดของโปรแกรมออกไป ทำให้การพัฒนาโปรแกรมบนลินุกซ์สามารถแลกเปลี่ยนกันได้อย่างสะดวก และยังสามารถนำโปรแกรมที่สร้างแล้วมาพัฒนาต่อให้สามารถใช้งานได้มีประสิทธิภาพมากขึ้น

2.3.2 โครงสร้างเกี่ยวกับระบบไฟล์และไดเรกทอรี

โครงสร้างเกี่ยวกับระบบไฟล์และไดเรกทอรีในลินุกซ์และระบบปฏิบัติการอื่นที่เข้ากันกับยูนิกซ์

/	ไดเรกทอรีราก
/bin	โปรแกรมคำสั่งทั่วไปของระบบ
/boot	ไฟล์ที่ใช้ทำการบูตระบบ
/dev	ดีไวซ์ไฟล์สำหรับติดต่อกับระบบ
/etc	คอนฟิกไฟล์ของระบบ
/home	โฮมไดเรกทอรีของผู้ใช้งานทั่วไปในระบบ
/lib	แชร์ไลบรารีและเคอร์เนลโมดูล

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

/mnt	ไดเรกทอรีสำหรับการเมาท์พาร์ติชันขึ้นมาชั่วคราว
/opt	ซอฟต์แวร์แพ็คเกจที่ติดตั้งเพิ่มเติมเข้ามาในระบบ
/root	โฮมไดเรกทอรีสำหรับผู้ดูแลระบบ
/sbin	โปรแกรมคำสั่งสำหรับการจัดการระบบ
/tmp	ไฟล์ชั่วคราว
/usr	เก็บโปรแกรมและข้อมูลที่สามารถใช้งานร่วมกันกับเครื่องอื่นๆได้
/var	ข้อมูลทั่วไปของระบบที่ถูกใช้โดยโปรแกรมต่างๆ

2.3.3 คำสั่งพื้นฐานที่ใช้ในลินุกซ์/ยูนิกซ์

คำสั่ง	คำอธิบาย
at	เป็นการสั่งให้เชลสคริปต์ ทำงานตามเวลาที่กำหนด
cal	แสดงปฏิทินตามช่วงเวลาที่กำหนด
calender	แจ้งเตือนผู้ใช้งานถึงกำหนดการต่างๆ
cat	ใช้ในการเชื่อมต่อไฟล์จากไฟล์กลุ่มหนึ่งไปยังอีกไฟล์หนึ่ง
cd	ใช้ในการเปลี่ยนตำแหน่งไดเรกทอรีใช้งาน
chgrp	ใช้กำหนดกลุ่มผู้ใช้งานให้กับไฟล์หรือไดเรกทอรีในระบบไฟล์ของยูนิกซ์
chmod	ใช้ในการเปลี่ยนแปลงคุณสมบัติสิทธิ์ในการเข้าใช้งานไฟล์
chown	ใช้ในการเปลี่ยนแปลงความเป็นเจ้าของไฟล์
comm	ใช้เปรียบเทียบไฟล์ข้อความที่ได้รับการจัดเรียงลำดับมาแล้ว
cp	ใช้ในการสำเนาไฟล์หนึ่งไฟล์หรือมากกว่าหนึ่งไฟล์
crontap	ใช้ในการกำหนดเวลาในการสั่งปฏิบัติคำสั่งตามกำหนด
date	แสดงหรือกำหนดค่าวันและเวลาในปัจจุบัน
df	แสดงเนื้อที่ของหน่วยบันทึกข้อมูลที่เหลืออยู่
finger	ใช้แสดงข้อมูลของผู้ใช้งานซึ่งเข้าใช้งานระบบปฏิบัติการยูนิกซ์
find	เป็นคำสั่งที่ใช้ในการตรวจสอบ ค้นหาตำแหน่งของไฟล์หรือไดเรกทอรี
ftp	คำสั่งพื้นฐานในการรับส่งไฟล์ ข้อมูลข้ามเครือข่ายคอมพิวเตอร์
grep	ใช้เลือกเอาต์พุต ที่มีความสอดคล้องกับที่ระบุไว้ขึ้นมา
head	แสดงบรรทัดภายในไฟล์ข้อความนับจากส่วนต้นลงมา ตามจำนวนบรรทัด ที่ผู้ใช้งานเป็นผู้กำหนด
kill	ทำการลบโปรเซส
ipr	ใช้ในการส่งไฟล์ซึ่งเกี่ยวข้องกับการจัดการด้านการพิมพ์
ls	แสดงรายชื่อไฟล์และไดเรกทอรี
mesg	ไว้สำหรับให้ผู้ใช้งานเลือกว่า จะยอมรับข้อความ ผ่านทาง session การทำงานของ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

	คนในปัจจุบันหรือไม่
mkdir	สร้างไดเรกทอรีขึ้นใหม่
more	มีความสามารถในการควบคุม การแสดงข้อความได้ครั้งละ กลุ่มบรรทัดตามแต่ ผู้ใช้ งานทำการกำหนด
mv	ใช้สำหรับ เคลื่อนย้ายไฟล์จาก ไดเรกทอรีหนึ่ง ไปยังอีกไดเรกทอรีหนึ่ง
ps	ตรวจสอบรายชื่อ โพรเซส
rm	เป็นคำสั่งที่ใช้ในการลบไฟล์หรือไดเรกทอรีที่ถูกสร้างขึ้นในระบบไฟล์
rmdir	ลบไดเรกทอรี
spell	ทำการตรวจสอบตรวจสอบสะกดของคำศัพท์ต่างๆ
tail	แสดงบรรทัดภายในไฟล์ข้อความนับจากส่วนท้ายขึ้นมา ตามจำนวนบรรทัด ที่ผู้ใช้ งานเป็นผู้กำหนด
tar	สำหรับใช้ในการบันทึกข้อมูล จากเครื่องคอมพิวเตอร์ ไปยังเทปบันทึกข้อมูล
telnet	ทำล็อกอินจากระยะไกล (remote login) ไปยังเครื่องคอมพิวเตอร์เครื่องอื่นๆ ซึ่งเชื่อมต่อ อยู่ด้วยกัน
tr	การจัดการกับไฟล์ข้อความ (text file) ในลักษณะของการประมวลผลข้อความ (word processing)
tty	เป็นคำสั่งที่ใช้แสดงชื่อของอุปกรณ์เทอร์มินัล
w	ใช้ในการแสดงรายละเอียดต่างๆ ของผู้ใช้งาน
wc	ใช้ตรวจสอบจำนวนบรรทัด จำนวนคำ และจำนวนตัวอักษร ในไฟล์ข้อความ
who	ตรวจสอบว่ามีใครที่กำลังล็อกอินอยู่บ้าง

ตารางที่ 2.1 แสดงคำสั่งพื้นฐานที่ใช้ในลินุกซ์/ยูนิกซ์

2.4 ความรู้พื้นฐานเกี่ยวกับการดูแลระบบ

ผู้ใช้ทั่วไปจะถูกจำกัดการใช้งานให้อยู่ในขอบเขตตามที่ผู้ดูแลระบบกำหนด แต่สำหรับงานที่เกี่ยวข้องกับการดูแลรักษาระบบแล้ว จำเป็นต้องใช้สิทธิ์ในระดับของ super user หรือ root ซึ่งจะมีสิทธิ์อำนาจสูงสุดในระบบ โดยสามารถที่จะทำการแก้ไขและเปลี่ยนแปลงได้ทุกอย่าง แต่ก็ต้องระมัดระวังเมื่อจะทำการแก้ไขดังกล่าวนี้ด้วย เนื่องจากหากมีความผิดพลาดอาจหมายถึงการทำลายระบบลงได้

เราสามารถจะใช้เครื่องมือเล็กๆสองตัวคือ grep กับ find เพื่อช่วยในการบริหารงานระบบได้ สำหรับ grep จะใช้เลือก output ที่มีความสอดคล้องกับที่เราระบุไว้ขึ้นมาและ find จะใช้สำหรับตรวจหาไฟล์ที่ต้องการ

ตัวอย่างการตรวจหาโพรเซส (โปรแกรมที่ถูกโหลดไว้ในหน่วยความจำ) ของผู้ใช้ชื่อ "twg"

```
# ps -aux | grep twg

twg      524  0.0  2.3 2416   728 ?  S   14:03   0:04 fvw95-2
twg      531  0.0  1.6 2208   512 ?  S   14:03   0:00 /usr/X11R6/lib/X11/fv
twg      532  0.0  2.4 2292   740 ?  S   14:03   0:01 /usr/X11R6/lib/X11/fv
twg      536  0.0  1.4 2192   432 ?  S   14:03   0:00 /usr/X11R6/lib/X11/fv
twg      537  0.0  2.9 3336   896 ?  S   14:03   0:00 xclock -bg #c0c0c0 -p
twg      538  0.0  2.8 3372   880 ?  S   14:03   0:00 xload -nolabel -bg gr
twg      541  0.0  1.8 1268   572 p1  S   14:03   0:00 -bash
twg      554  0.0  1.9 1264   612 p2  S   14:03   0:00 -bash
twg      649  0.0  0.6 1552   204 ?  S   14:16   0:00 /opt/applix/axdata/ax
twg     1325  1.0 36.7 16968 11304 ?  S   15:09   0:45 netscape -geometry 79
twg     1326  0.0  3.0 10276   936 ?  S   15:09   0:00 (dns helper)
twg     1648  0.0  2.0 1212   636 p2  S   16:15   0:00 vi sysadmin1.txt
twg     1661  0.0  1.6 1188   500 p2  S   16:19   0:00 /bin/bash -c (ps -aux
twg     1662  0.0  1.7 1192   540 p2  S   16:19   0:00 /bin/bash -c (ps -aux
twg     1663  0.0  1.3  928   416 p2  R   16:19   0:00 ps -aux
twg     1664  0.0  1.0  912   308 p2  R   16:19   0:00 grep twg
```

2.5 เชล

เชลเป็นโปรแกรมที่ทำหน้าที่ติดต่อระหว่างผู้ใช้งานและยูนิกซ์ ทำให้ผู้ใช้งานสามารถป้อนคำสั่งให้ยูนิกซ์รันตามที่ต้องการ โดยจะซ่อนการทำงานของเคอร์เนลไว้เบื้องหลัง ทำให้ผู้ใช้งานทำงานได้ง่ายขึ้น เช่น การเปลี่ยนทิศทางข้อมูล โดยใช้ > หรือ ถ้าต้องการจะเชื่อมต่อการทำงานของโพรเซสทำได้โดยใช้ Pipe, |, หรืออีกนัยหนึ่งถือเป็นภาษาระดับสูง ที่ใช้ในการเขียนโปรแกรมบนยูนิกซ์

ยูนิกซ์มีเชลอยู่หลายชนิด ส่วนใหญ่แล้วจะปรับปรุงมาจากบอร์นเชล

ชื่อของเชล	คำอธิบาย
sh(Bourne)	บอร์นเชล เชลดั้งเดิมของยูนิกซ์
csh, tcsh ,zsh	ซีเชล
ksh,pdksh	คอร์นเชล
bash	ได้รับการปรับปรุงมาจากบอร์นเชล
rc	เป็นเชลที่มีลักษณะเป็นภาษาซีมากกว่า csh

ตารางที่ 2.2 เชลชนิดต่างๆ

ในส่วนของตัวแปรเชลที่เราจะกล่าวถึงเพียงตัวแปรสแกนหรือตัวแปรของระบบที่เป็นพื้นฐานที่ควรรู้

ตัวแปรกำหนดการทำงานของเซล

เมื่อมีการรันเซลล์ จะมีการรันตัวแปรบางอย่างที่ได้กำหนดค่าเอาไว้ ประกติจะเป็นตัวอักษรพิมพ์ใหญ่ เพื่อให้แตกต่างกับ ตัวแปรที่ผู้ใช้งานกำหนดขึ้นมาใช้งานในเซลล์สคริปต์ ซึ่งปกติจะกำหนดให้เป็นตัวอักษรพิมพ์เล็ก ตารางนี้จะแสดงตัวแปรที่ใช้บ่อยที่ควรรู้

ตัวแปรเซลล์	คำอธิบาย
SPS1	พรมดค่าสั่ง ประกติจะใช้ \$
SPS2	เป็นพรมดที่แสดงเมื่อเซลล์รอรับข้อมูล (Second Promt) ประกติใช้ >
SPATH	กำหนดตำแหน่งที่จะให้ เซลล์ไปค้นหาไฟล์หรือโปรแกรมที่จะรัน
SIFS	เป็นเครื่องหมายแยกคำ เมื่อป้อนให้เซลล์
SHOME	กำหนดโฮมไดเรกทอรีสำหรับผู้ใช้งานที่กำลังใช้งาน

ตารางที่ 2.3 แสดงตัวแปรกำหนดการทำงานของเซลล์พื้นฐาน

พารามิเตอร์

เมื่อมีการเรียกสคริปต์ จะมีการกำหนดตัวแปรชนิดหนึ่งขึ้นมาโดยอัตโนมัติ ดังตาราง

พารามิเตอร์	คำอธิบาย
S1,S2,...	ค่าพารามิเตอร์ที่กำหนดโดยคำสั่ง set เรียงตามลำดับ
S0	ชื่อของไฟล์เซลล์สคริปต์
S#	จำนวนพารามิเตอร์ที่ใช้งาน
SS	หมายเลขโปรเซสของเซลล์สคริปต์
S*	แสดงรายชื่อของพารามิเตอร์ เหมือนกับที่กำหนดบนเซลล์
S@	รูปแบบหนึ่งของ S* โดย IFS ไม่มีผลต่อการทำงาน

ตารางที่ 2.4 แสดงพารามิเตอร์ของเซลล์

2.6 ไฟล์เพอร์มิสชันและไดเรกทอรีเพอร์มิสชัน (File Permission and Directory Permission)

ไฟล์เพอร์มิสชันในลินุกซ์ ซึ่งนับเป็นเรื่องหนึ่งที่มีความซับซ้อนมากเพราะนำแนวความคิดมาจาก ระบบปฏิบัติการยูนิกซ์ จากสถิติพบว่าโดยส่วนใหญ่การบุกรุกที่เกิดจากผู้ไ้ภายในระบบพยายามจะเพิ่ม สิทธิให้เทียบเท่ากับผู้ดูแลระบบนั้น มักมาจากความไม่รอบคอบในเรื่องนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในระบบลินุกซ์ ใช้วิธีการยืนยันตัวตนและสิทธิ์ที่ได้รับของผู้ที่จะเข้ามาในระบบโดยผ่านกระบวนการล็อกอิน โดยใช้รหัสผู้ใช้ (username) เพื่อระบุผู้ใช้ที่จะเข้ามาในระบบและรหัสผ่าน (password) ของผู้ใช้นั้นเพื่อยืนยันการเข้าสู่ระบบ ซึ่งรหัสผู้ใช้ ของผู้ใช้ในสมัยก่อนอนุญาตให้มีรหัสผู้ใช้ ได้ยาวมากที่สุดเพียงแปดตัวอักษร แต่ในปัจจุบันรวมถึงลินุกซ์ อนุญาตให้รหัสผู้ใช้ ได้ยาวมากที่สุดถึง 256 ตัวอักษร สมมุติในระบบมีผู้ใช้นั้นชื่อ sample พิจารณาไฟล์ /etc/passwd ที่ระบุชื่อผู้ใช้ชื่อ sample

```
sample:x:100:200:Sample account:/home/sample:/bin/bash
```

อธิบายฟิลด์ต่างๆ ได้ดังนี้ (แต่ละฟิลด์คั่นด้วยเครื่องหมาย ':')

Field	Description
sample	ชื่อ username คือ sample
x	เป็นรหัสผ่านซึ่งจะเก็บค่าไว้ในไฟล์ /etc/shadow
100	เป็น User ID หรือ UID ซึ่งลินุกซ์จะใช้ในการอ้างถึงเจ้าของไฟล์นั้นๆ
200	อ้างถึง GID ซึ่งลินุกซ์ใช้ในการบอกกรุปของผู้ใช้นี้
Sample account	รายละเอียดอื่นๆ เพิ่มเติมของผู้ใช้ชื่อ sample
/home/sample	เป็นโฮมไดเรกทอรีของผู้ใช้ sample
/bin/bash	เชลล์หลักของผู้ใช้

ตารางที่ 2.5 รายละเอียดในไฟล์ /etc/passwd

และผู้ใช้ที่ชื่อ sample อยู่ในกรุป staff พิจารณาไฟล์ /etc/group

```
staff:x:200:
```

อธิบายถึงฟิลด์ต่างๆ ได้ดังนี้

Field	Description
staff	กรุปชื่อ staff
x	ใช้ในการระบุรหัสผ่านให้กับกรุป
200	เป็น Group ID หรือ GID

ตารางที่ 2.6 แสดงรายละเอียดของไฟล์ /etc/groups

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผู้ใช้ชื่อ sample มี UID เป็น 100 และกรุปเป็น staff ซึ่งมี GID เป็น 200 ซึ่ง UID และ GID เป็นเลขจำนวนจริงขนาด 16 บิตมีค่าตั้งแต่ 0 จนถึง 65535 บางระบบอาจมี UID หรือ GID เป็นเลข 32 บิตได้ มนุษย์จะใช้ตัวอักษรในการอ้างถึง ส่วนลินุกซ์ จะอ้างถึงเป็นตัวเลขเสมอ เป็นแนวความคิดที่ไม่ซับซ้อนและใช้ได้มีประสิทธิภาพมาก โดย UID และ GID ของผู้ใช้จะกำหนดสิทธิ์ในการเข้าถึงไฟล์ในระบบ โดยจับคู่กันระหว่างชื่อผู้ใช้กับ UID และกรุปกับ GID

ถ้าผู้ใช้สองคนมี UID เหมือนกัน ลินุกซ์จะถือว่าผู้ใช้สองคนนี้เป็นคนเดียวกันแม้ว่ารหัสผู้ใช้และรหัสผ่านจะต่างกัน เนื่องมาจากเป็นคนเดียวกันในระบบทำให้มีสิทธิ์เหมือนกันทุกอย่าง เช่นสามารถลบไฟล์ของอีกคนหนึ่งก็ได้ สามารถหยุดโพรเซสของอีกคนหนึ่งได้ (ความหมายของคำว่าอีกคนหนึ่งในที่นี้คือใช้รหัสผู้ใช้คนละชื่อ) ความไม่รอบคอบในลักษณะนี้อาจจะทำให้ระบบทำงานได้ไม่ถูกต้องและมีผลในแง่ความปลอดภัยของระบบ

ผู้ใช้ในระบบที่ UID เป็น 0 คือ Superuser หรือผู้ดูแลระบบมีสิทธิ์ทุกอย่างในระบบ ซึ่งเป็นจุดประสงค์หลักของผู้บุกรุกในการพยายามที่จะทำให้ตัวเองมีสิทธิ์เทียบเท่าผู้ดูแลระบบไม่วิธีใดก็วิธีหนึ่ง ผู้ดูแลระบบควรระวังป้องกันไว้ด้วยวิธีต่างๆ เช่น

- เก็บไฟล์สำคัญไว้ในอุปกรณ์บันทึกข้อมูลที่ถอดเข้าออกได้เช่นฟลอปปีดิสก์ ซีดีรอม เทปแบ็กอัพ เป็นต้น หรือเก็บข้อมูลบนสื่อที่อ่านได้อย่างเดียว
- พยายามเข้ารหัสไฟล์ทุกไฟล์ที่สำคัญและมีสิทธิ์เข้าถึงได้โดยผู้ดูแลระบบเท่านั้น
- อย่าลืมแบ็กอัพระบบอย่างสม่ำเสมอ ทำให้ข้อมูลแบ็กอัพตามทันข้อมูลปัจจุบันอย่างสม่ำเสมอ

ลินุกซ์มองทุกสิ่งทุกอย่างในระบบเป็นไฟล์ ดังนั้น File permission จึงเป็นเรื่องสำคัญที่สุดต่อการใช้งาน และมักจะพบว่าเป็นความผิดพลาดเบื้องต้น ที่ผู้ดูแลระบบมองข้ามหรือไม่สนใจอยู่บ่อยครั้งทำให้ระบบเกิดจุดอ่อนได้ง่าย

คำสั่ง ls ใช้ในการแสดงไฟล์หรือไดเรกทอรีที่มีในระบบ สมมติตัวอย่างการใช้คำสั่ง ls -l เพื่อแสดงไฟล์ test ในระบบ

```
-rwxr-xr-- 1 sample staff 852 Jul 9 10:10 test
```

ไฟล์ชื่อ test มีเจ้าของเป็น sample และอยู่ในกรุปที่ชื่อ staff อธิบายถึงความหมายในฟิลด์ต่างๆ ได้ดังนี้

Field	Description
-rwxr-xr--	file permission
1	จำนวน hard link ของไฟล์นี้
sample	user ที่เป็นเจ้าของไฟล์
staff	กรุป
852	ขนาดของไฟล์ หน่วยเป็นไบต์
Jul 9	วันที่มีการแก้ไขล่าสุด
10:10	เวลาที่มีการแก้ไขล่าสุด
test	ชื่อไฟล์

ตารางที่ 2.7 ความหมายของฟิลด์เมื่อใช้คำสั่ง ls

พิจารณา File permission -rwxr-xr-- ได้ว่า

-	r	w	x	r	-	x	r	-	-
type	owner (user)			group			others		

ตารางที่ 2.8 กลุ่มของ permission

ในแต่ละส่วนอธิบายเพิ่มเติม

Description	คำอธิบาย
type	บอกถึงชนิดของไฟล์
owner (user)	สิทธิ์ในการเข้าถึงไฟล์ของเจ้าของไฟล์
group	สิทธิ์ในการเข้าถึงไฟล์ของผู้ใช้ในระบบที่อยู่ในกรุปที่ระบุไว้
others	สิทธิ์ในการเข้าถึงไฟล์ของทุกคนในระบบที่ไม่ใช่เจ้าของไฟล์และไม่ได้อยู่ในกรุป เดียวกันกับ group ที่ระบุไว้

ตาราง 2.9 คำอธิบายเพิ่มเติม permission ในกลุ่มต่างๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

โดยที่ types จะอธิบายถึงชนิดของไฟล์

types	ความหมาย
-	ไฟล์ปกติ
d	ไดเรกทอรี
c	ดีไวซ์แบบ character
b	ดีไวซ์แบบ block
l	symbolic link
s	socket
p	FIFO

ตารางที่ 2.10 แสดงชนิดของไฟล์

ความหมายของ permission หรือสิทธิในการเข้าถึงไฟล์ปกติที่ไม่ใช่ไดเรกทอรี คือ

File		
Character	Permission	ความหมาย
r	READ	สามารถเปิดไฟล์และอ่านเนื้อความในไฟล์ได้
w	WRITE	สามารถเพิ่มข้อความ ลบข้อความ หรือเปลี่ยนแปลงเนื้อความในไฟล์ได้
x	EXECUTE	จะใช้ในกรณีที่ไฟล์นี้สามารถทำงานได้หรือสั่งให้ทำงานได้ ซึ่งจะมีสิทธิในการทำงานนี้ปรากฏอยู่

ตารางที่ 2.11 แสดง File permission

แต่เมื่อนำมาใช้กับไคลเรททอรีจะมีสิทธิ์ในอีกความหมายหนึ่งดังนี้

Directory		
Character	Permission	ความหมาย
R	READ	สามารถใช้คำสั่ง ls ในการดูไฟล์ภายในไคลเรททอรีได้
W	WRITE	สามารถเพิ่มไฟล์ ลบไฟล์ เปลี่ยนชื่อไฟล์ที่อยู่ในไคลเรททอรีได้
X	EXECUTE	สามารถใช้คำสั่ง cd ทำการเปลี่ยนไคลเรททอรีปัจจุบันเข้าไปได้ หรือใช้ในการเปิดไฟล์ภายในไคลเรททอรีรวมไปถึงไคลเรททอรีย่อยภายใน

ตารางที่ 2.12 แสดง Directory permission

การทำความเข้าใจเรื่อง permission หรือสิทธิ์ในการเข้าถึงทั้งไฟล์หรือไคลเรททอรี เป็นสิ่งจำเป็น เพราะจะช่วยแก้ปัญหาที่เกิดขึ้นได้อย่างรวดเร็ว มีประเด็นสำคัญที่ควรให้ความสนใจบางประเด็นเช่น

- จำเป็นต้องมีสิทธิ์ EXECUTE สำหรับไคลเรททอรีในการเปลี่ยนเป็นไคลเรททอรีปัจจุบันโดยการใช้นำคำสั่ง cd หรือ chdir รวมไปถึงการเปลี่ยนไคลเรททอรีที่เป็นไคลเรททอรีภายใต้ไคลเรททอรีที่มีสิทธิ์ดังกล่าวอยู่นี้
- ถ้าไม่มีสิทธิ์ EXECUTE ของไคลเรททอรีใด จะไม่สามารถเข้าถึงไฟล์ใดๆ ภายใต้ไคลเรททอรีนั้นได้เลย ไม่มีข้อยกเว้น รวมไปถึงเจ้าของไคลเรททอรีด้วย ก็ไม่สามารถทำได้
- ถ้ามีสิทธิ์เฉพาะ EXECUTE แต่ไม่มีสิทธิ์ READ จะสามารถเข้าไปในไคลเรททอรีนั้นได้ แต่จะไม่สามารถใช้คำสั่ง ls ในการดูไฟล์ภายในไคลเรททอรีนั้นได้ (คือไม่มีสิทธิ์ในการอ่านเนื้อความภายในไคลเรททอรี) แต่ถ้าทราบชื่อไฟล์และได้รับสิทธิ์เพียงพอในการเข้าใช้งานไฟล์เหล่านั้น ซึ่งจะเป็นประโยชน์ต่อเจ้าของไฟล์ในการซ่อนเนื้อความในไคลเรททอรีนั้น ซึ่งจะพบเห็นได้บ่อยครั้ง
- ถ้าจะสร้างลิงก์ไปยังไฟล์ใดในไคลเรททอรีจำเป็นต้องมีสิทธิ์ WRITE และ EXECUTE ในไคลเรททอรีจึงจะสร้างได้ หรือกรณีลบก็เช่นเดียวกัน

2.7 การบันทึกเหตุการณ์บนระบบ

2.7.1 การบันทึกเหตุการณ์ของลือกระบบ

โปรแกรมที่รันโดยระบบมักจะแสดงข้อความออกมาทางหน้าจอแสดงผล หรือเก็บไว้ในล็อกไฟล์ ข้อความอาจจะแสดงข้อผิดพลาด หรือเตือนหรือแสดงข้อมูลทั่วไปของระบบ ข้อมูลเหล่านี้จะเก็บไว้ในไคลเรททอรีที่กำหนดให้เก็บข้อความเหล่านี้โดยเฉพาะ เช่น /usr/adm หรือ /var/log สำหรับลินุกซ์จะกำหนดไว้ที่ /var/log/messages เพื่อเก็บข้อความของระบบ, /var/log/syslog เก็บข้อความอื่นๆ หรือ /var/log/debug เก็บข้อความของการดีบั๊กโปรแกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในยูนิกซ์มีข้อกำหนดให้มีฟังก์ชันเพื่อให้โปรแกรมสามารถเข้าถึงข้อมูลเหล่านี้ได้ คือ syslog ซึ่งมีรูปแบบดังต่อไปนี้

```
#include <syslog.h>

void syslog(int priority,const char *messages,arguments)
```

- ฟังก์ชันจะส่งข้อความที่เกิดขึ้น ไปยังส่วนต่างๆที่ทำหน้าที่เกี่ยวกับข้อความดังกล่าว
- priority จะควบคุมการทำงานต่างๆ เกี่ยวกับข้อความ ซึ่งเกิดจากการทำ bitwise OR ระหว่าง severity level และ facility level
- facility จะกำหนดไว้ที่ไฟล์ syslog.h เช่น LOG_USER หมายถึงข้อความจากแอปพลิเคชันของผู้ใช้งาน(เดฟฟอลต์)

ส่วนค่า severity levels เรียงตามลำดับความสำคัญของข้อความดังนี้

ลำดับความสำคัญ	คำอธิบาย
LOG_EMERG	เป็นเรื่องสำคัญมาก
LOG_ALERT	คำเตือนเรื่องสำคัญ เช่นฐานข้อมูลถูกทำลาย
LOG_CRIT	ข้อผิดพลาดบางอย่าง เช่นฮาร์ดแวร์ไม่ทำงาน
LOG_ERR	ข้อผิดพลาดที่เกิดขึ้น
LOG_WARNING	คำเตือน
LOG_NOTICE	เงื่อนไขบางอย่าง ที่ควรจะทราบ
LOG_INFO	ข้อมูล
LOG_DEBUG	ข้อความเกี่ยวกับการดีบั๊กโปรแกรม

โดยค่า severity levels ที่แตกต่างกัน ก็จะไปบันทึกในล็อกไฟล์ที่แตกต่างกันไป

ข้อความที่สร้างขึ้นด้วยฟังก์ชัน syslog จะประกอบด้วยส่วน header และ body ของข้อความ ส่วน header จะประกอบด้วยส่วน facility, ข้อมูลวันที่และเวลา ส่วน body จะกำหนดโดยใช้พารามิเตอร์ของข้อความ โดยเฉพาะอย่างยิ่ง %m จะแสดงข้อความที่สอดคล้องกับค่าที่เกิดขึ้นในตัวแปร errno ดูตัวอย่างดังโปรแกรมนี้


```
/*logxxxx.c*/
#include <syslog.h>
#include <stdio.h>
int main()
{
    FILE *file_ptr;
    File_ptr = fopen("non_file","r");
    If(!file_ptr)
    Syslog(LOG_ERR|LOG_USER,"XXXX - %m\n");
    Return(0);
}
```

เมื่อรันแล้วจะไม่เห็นเอาท์พุทเกิดขึ้นที่จอ แต่เมื่อตรวจสอบดูที่ไฟล์ /var/log/syslog จะประกอบด้วยบรรทัดต่อไปนี้ที่ท้ายไฟล์

```
Apr 20 11:56:00 smile logxxxx : XXXX - No such file or directory
```

การบันทึกเหตุการณ์บนระบบโดยการใช้ล็อกของระบบนั้นอาจจะไม่ปลอดภัยนักในกรณีที่ผู้บุกรุกรู้จักล็อกของระบบเป็นอย่างดี อาจจะกระทำการอย่างใดอย่างหนึ่งให้ล็อกของระบบหยุดทำงาน ทำให้การบันทึกเหตุการณ์ของระบบซึ่งถือเป็นหัวใจสำคัญอย่างหนึ่งของระบบรักษาความปลอดภัยของระบบคอมพิวเตอร์ต้องหยุดทำงานลง ทำให้ผู้บุกรุกสามารถกระทำการอื่นได้กับระบบก็ได้โดยที่ไม่มีการบันทึกพฤติกรรมและสิ่งหลักฐานเอาไว้

ดังนั้นในโครงการนี้จึงได้ใช้วิธีการเขียนล็อกไฟล์ขึ้นมาเพื่อบันทึกข้อความรายละเอียดพฤติกรรม การบุกรุกของผู้ใช้ขึ้นมาเอง โดยใช้วิธีการโปรแกรมที่เกี่ยวข้องในหัวข้อต่อไป

2.7.2 **แอสเสจคิว (message queue)**

เป็นวิธีการที่เขียนโปรแกรมส่งข้อมูลจากโพรเซสไปยังอีกโพรเซสหนึ่งผ่านเคอร์เนล โดยจะทำการส่งข้อมูลเป็นบล็อกๆ แต่ละบล็อกข้อมูลจะมีหมายเลขประจำบล็อกนั้น โดยโพรเซสสามารถจะเขียนข้อมูลไปยังคิว และจบการรันโพรเซส โดยที่โพรเซสอื่นสามารถที่จะมาอ่านข้อความได้ในภายหลัง

ฟังก์ชันที่ใช้งานในแอสเสจคิว

```
#include <sys/msg.h>;

int msgget(key_t key, int msgflg);

int msqctl(int msqid, int cmd, struct msqid_ds *buf );
```



```
int msgsnd(int msqid, const void *msgp, size_t msgsz,int msgflg);
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,int msgflg);
```

msgget

ทำหน้าที่ในการสร้างเมสเสจคิว

```
int msgget(key_t key, int msgflg);
```

key เป็นตัวแปร int เพื่อจำแนกออบเจกต์ IPC
msgflg เป็นบิตเพอร์มิสชัน 9 บิต

ฟังก์ชันจะส่งค่ากลับเป็น ค่าตัวเลขบ่งบอกเมสเสจ (message identifier) จำนวนเต็มบวกหรือ -1 ถ้าทำงานไม่สำเร็จ

msgctl

ทำหน้าที่คล้ายกับฟังก์ชันที่ควบคุมแชร์หน่วยความจำ มีรูปแบบการใช้งานดังนี้

```
int msgctl(int msqid, int cmd, struct msqid_ds *buf);
```

msqid เป็น message identifier ค่าส่งกลับจากฟังก์ชัน msgget
cmd เป็นคำสั่งที่จะให้ฟังก์ชันทำงานดังนี้

คำสั่ง	คำอธิบาย
IPC_STAT	แสดงสถานะของ message queue
IPC_SET	กำหนดสถานะของ message queue
IPC_RMID	ลบ message queue ออกจากระบบ

โครงสร้างข้อมูล msqid_ds จะมีฟิลด์ข้อมูลอย่างน้อยดังนี้

```
Struct{
    Uid_t msg_perm.uid;
    Uid_t msg_perm.gid;
    Mode_t msg_perm.mode;
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ฟังก์ชันจะส่งค่ากลับเป็น 0 ถ้าทำงานสำเร็จ หรือส่งค่า -1 เมื่อเกิดข้อผิดพลาด

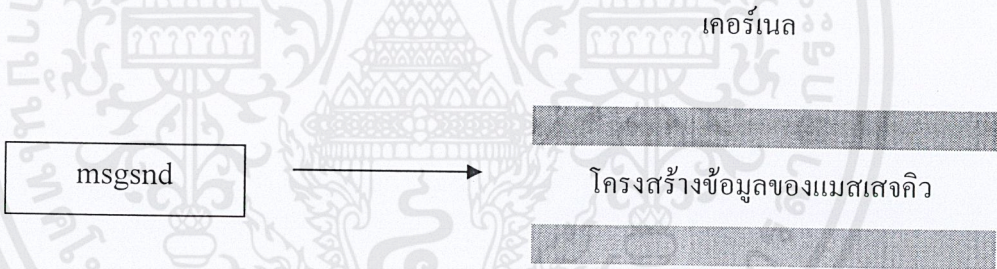
msgsnd

ทำหน้าที่เพิ่มข้อมูลเข้าไปในแมสเสจคิว มีรูปแบบการใช้งานดังนี้

```
int msgsnd(int msqid, const void *msgp, size_t msgsz,int msgflg);
```

- msqid เป็น message identifier ค่าส่งกลับจากฟังก์ชัน msgget
- msgp เป็นพอยน์เตอร์ชี้ไปยังข้อมูลที่จะส่งไป
- msgsz เป็นขนาดข้อมูลที่ชี้โดยพอยน์เตอร์ msgp
- msgflg เป็นแฟลกกำหนดการทำงาน

ถ้าทำงานสำเร็จข้อมูลจะถูกคัดลอกไปยังคิว และส่งค่ากลับเป็น 0 ถ้าทำงานไม่สำเร็จจะส่งค่ากลับเป็น -1



รูปที่ 2.3 แสดงการเพิ่มข้อมูลเข้าไปในแมสเสจคิว ของฟังก์ชัน *msgsnd*

โครงสร้างข้อมูลในบล็อก จะประกอบด้วย 2 ส่วนคือ

- ข้อมูล โดยจะต้องเล็กกว่าความจุบล็อก
- ตัวแปรที่จะอ้างอิงถึงข้อมูล เพื่อให้อ้างอิงข้อมูลในฟังก์ชันรับข้อมูล

ดังนั้น โครงสร้างข้อมูลของข้อมูลควรมีลักษณะดังนี้

```
struct msg_q{
    long int msg_ident;
    char text[BUFSIZ];
}
```


msgrcv

ทำหน้าที่แอ็กเซสข้อมูลในเมสเสจคิว มีรูปแบบการใช้งานดังนี้

```
int msgrcv(int msqid, void *msgp, size_t msgsz, long msgtyp,int msgflg);
```

msqid	เป็น message identifier ค่าส่งกลับจากฟังก์ชัน msgget
msgp	เป็นพอยน์เตอร์ชี้ไปยังโครงสร้างข้อมูลที่จะเก็บข้อมูลจากคิว
msgsz	เป็นขนาดข้อมูลที่ชี้โดยพอยน์เตอร์ msgp
msgtyp	กำหนดข้อความที่ต้องการ ถ้าเป็น 0 จะแอ็กเซสข้อความตามลำดับที่ส่งเข้ามาในคิว ถ้าต้องการจะแอ็กเซสข้อความที่ต้องการ โดยกำหนดเป็นหมายเลขของข้อความนั้น
msgflg	เป็นแฟล็กกำหนดการทำงาน

ถ้าฟังก์ชันทำงานสำเร็จ จะส่งค่ากลับเป็นจำนวนไบต์ที่นำไปเก็บไว้ในบัพเฟอร์ msgp และข้อมูลในคิวจะถูกลบออกไป โดยถ้าเกิดข้อผิดพลาด จะส่งค่ากลับเป็น -1

2.7.3 การทำงานกับไฟล์

ฟังก์ชันพื้นฐานที่ทำงานกับไฟล์ มีดังนี้

open

ทำหน้าที่เปิดไฟล์ หรือเริ่มทำหน้าที่ติดต่อกับอุปกรณ์

```
#include <fcntl.h>
#include <sys/types.h>
#include <sys/stat.h>

int open(const char *path,int oflags);
int open(const char *path,int oflags,mode_t mode);
```

path	จะกำหนดไฟล์ที่เปิดหรือสร้างขึ้นมาใหม่
oflags	จะกำหนดเพอร์มิชชันการแอ็กเซสเมื่อเปิดไฟล์
mode	กำหนดเมื่อมีการสร้างไฟล์ขึ้นมาใหม่

write

ทำหน้าที่เขียนข้อมูลลงไฟล์หรืออุปกรณ์

```
#include <unistd.h>

size_t write(int fildes,const void *buf,size_t nbytes);
```

จะนำข้อมูลจำนวน *nbytes* จากบัฟเฟอร์ *buf* ไปเขียนลงในไฟล์ที่มี file descriptor *fildes* โดยจะส่งค่ากลับเป็นจำนวนไบต์ที่เขียนจริงๆ ถ้าส่งค่ากลับ 0 แสดงว่าถึงจุดสิ้นสุดไฟล์ ถ้าส่งค่ากลับเป็น -1 แสดงว่าเกิดข้อผิดพลาดขึ้นในการทำงานของฟังก์ชัน *write*

read

ทำหน้าที่อ่านข้อมูลจากไฟล์หรืออุปกรณ์

```
#include <unistd.h>

size_t read(int fildes,const void *buf,size_t nbytes);
```

จะอ่านข้อมูล *nbytes* จาก file descriptor *fildes* ไปเก็บไว้ในในบัฟเฟอร์ *buf* จะส่งค่ากลับเป็นจำนวนไบต์ที่อ่านได้จริง ถ้าส่งค่ากลับ 0 หมายถึงไม่มีการอ่านข้อมูล ถ้าส่งค่ากลับเป็น -1 แสดงว่าเกิดข้อผิดพลาดขึ้นในการทำงานของฟังก์ชัน *read*

close

ทำหน้าที่ปิดไฟล์หรืออุปกรณ์

```
#include <unistd.h>

size_t close(int fildes);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะยกเลิกการใช้งาน file descriptor กับไฟล์นั้น จะทำให้ fildes สามารถจะนำไปใช้งานใหม่ได้ จะส่งค่ากลับ 0 ถ้าทำงานสำเร็จ หรือ ส่งค่ากลับเป็น -1 ถ้ามีข้อผิดพลาดเกิดขึ้น



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

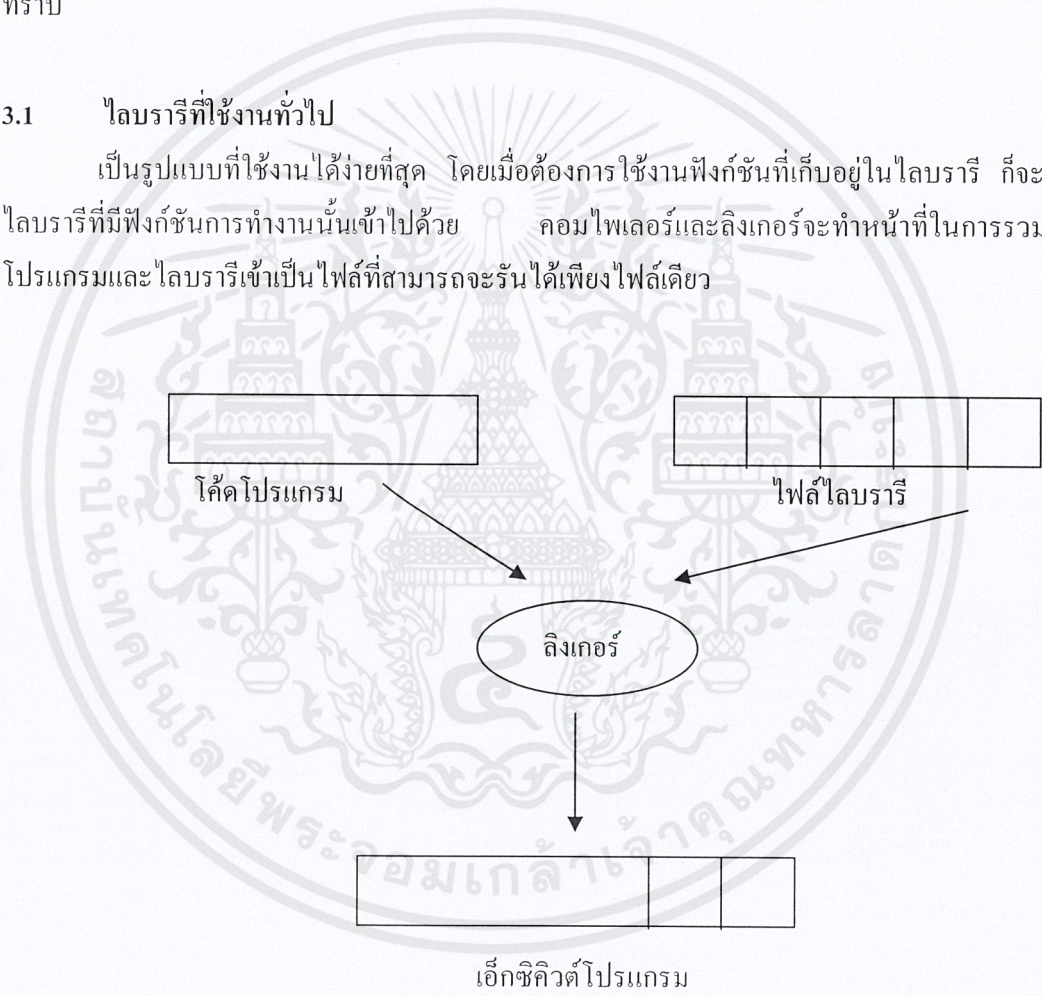
บทที่ 3

สแตนด์ดาร์ดซีไลบรารี

ไลบรารีเป็นฟังก์ชันที่คอมไพล์ไว้แล้ว สามารถที่จะเรียกใช้งานฟังก์ชันที่อยู่ในไลบรารีได้ เช่น ไลบรารีที่เกี่ยวกับการรับหรือแสดงข้อมูล (stdio library) หรือไลบรารีเกี่ยวกับฐานข้อมูล (dbm library) โดยทั่วไปไลบรารีของระบบจะถูกเก็บไว้ในไดเรกทอรี /lib และ /usr/lib คอมไพเลอร์จะค้นหาไลบรารีเฉพาะที่ไดเรกทอรีมาตรฐานนี้เท่านั้น ถ้ามีไลบรารีที่ตำแหน่งต่างออกไป จะต้องบอกให้คอมไพเลอร์ทราบ

3.1 ไลบรารีที่ใช้งานทั่วไป

เป็นรูปแบบที่ใช้งานได้ง่ายที่สุด โดยเมื่อต้องการใช้งานฟังก์ชันที่เก็บอยู่ในไลบรารี ก็จะรวมไฟล์ไลบรารีที่มีฟังก์ชันการทำงานนั้นเข้าไปด้วย คอมไพเลอร์และลิงเกอร์จะทำหน้าที่ในการรวมโค้ดของโปรแกรมและไลบรารีเข้าเป็นไฟล์ที่สามารถจะรันได้เพียงไฟล์เดียว



รูปที่ 3.1 การลิงก์แบบสแตติก

ข้อเสียของสแตติกไลบรารี ก็คือเมื่อมีการรันหลายๆ โปรแกรมในเวลาเดียวกันและทั้งหมดใช้ฟังก์ชันจากไลบรารีเดียวกัน หมายถึงจะมีฟังก์ชันการทำงานเดียวกันในหน่วยความจำซ้ำกัน นอกจากนั้นยังซ้ำกันในตัวโปรแกรมอีกด้วย ซึ่งเป็นการสูญเสียพื้นที่หน่วยความจำและพื้นที่ฮาร์ดดิสก์ไปโดยเปล่าประโยชน์

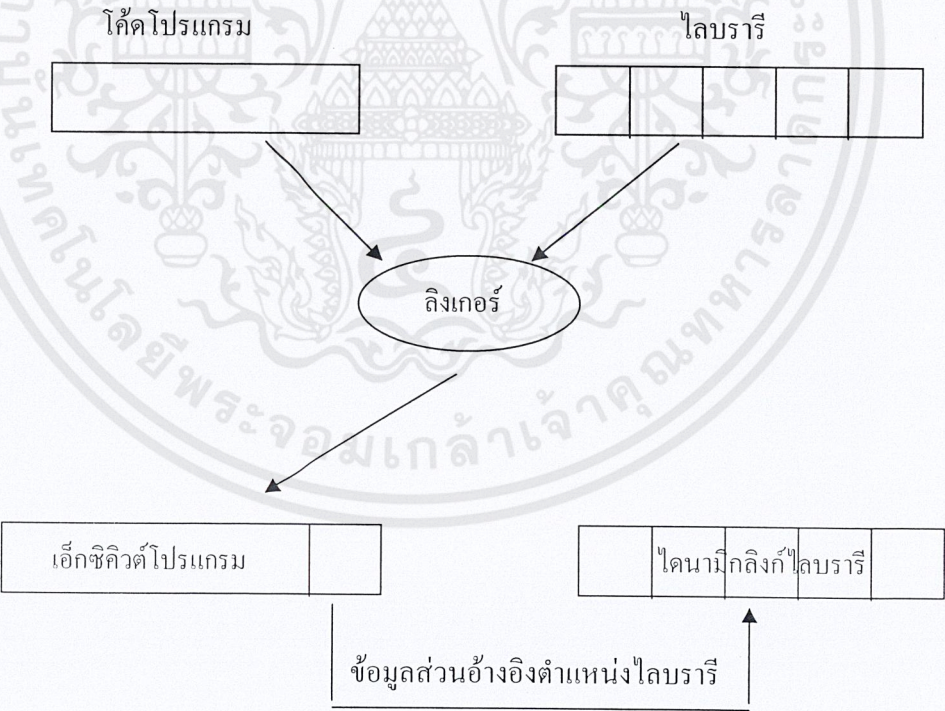
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2 ไลบรารีที่ใช้งานร่วมกัน

ไลบรารีที่สามารถใช้งานร่วมกันหรือแชร์ไลบรารีเป็นไลบรารีที่ให้โปรแกรมต่างๆเรียกใช้งานร่วมกัน โดยตัวไลบรารีไม่ได้รวมเป็นส่วนหนึ่งของตัวโปรแกรม เมื่อต้องการที่จะใช้ไลบรารีจะทำการเรียกแบบไดนามิก (Dynamix Linking) ไปที่ไฟล์ไลบรารี สำหรับลินุกซ์เองจะเก็บไลบรารีดังกล่าวไว้ที่เดียวกับสแตติกไลบรารี เพียงแต่จะมีการกำหนดชื่อที่ต่างกัน บนลินุกซ์แชร์ไลบรารีของภาษาซีคือ /lib/libc.so.N เมื่อ N หมายถึงหมายเลขเวอร์ชัน

เมื่อโปรแกรมมีการใช้งานแชร์ไลบรารี จะเรียกไปยังไครเรททอรี /usr/lib/libc.sa ซึ่งเป็นไลบรารีที่มีการใช้งานร่วมกัน หรืออาจมีการโหลดไลบรารีเพิ่มเติมเมื่อต้องการใช้งาน โดยการทำงานในลักษณะนี้จะทำให้ระบบแก้ไขปัญหาไลบรารีสำหรับหลายแอปพลิเคชันเข้าซ้กันกันได้ สำหรับลินุกซ์โปรแกรมที่ทำหน้าที่จัดการเกี่ยวกับแชร์ไลบรารี คือ ld.so หรือ ld-linux.so ส่วนตำแหน่งที่จะกำหนดให้ระบบไปค้นหาแชร์ไลบรารี จะกำหนดไว้ที่ไฟล์ /etc/ld.so.conf

การตรวจสอบว่า โปรแกรมใช้งานแชร์ไลบรารีอยู่หรือไม่ อะไรบ้าง ใช้คำสั่ง : ldd



รูปที่ 3.2 การลิงก์แบบไดนามิก

3.3 ความหมายของสแตนด์ดาร์ดซีไลบรารี

สแตนด์ดาร์ดซีไลบรารี คือไลบรารีมาตรฐานของภาษาซี ประกอบด้วยฟังก์ชันการทำงานพื้นฐานของภาษาซี ภายในสแตนด์ดาร์ดซีไลบรารีจะประกอบด้วยไฟล์เฮดเดอร์หลายไฟล์ตัวอย่างเช่น

- <assert.h> ฟังก์ชันเกี่ยวกับการวินิจฉัย
- <ctype.h> ฟังก์ชันเกี่ยวกับการทดสอบประเภทตัวอักษร
- <errno.h> ฟังก์ชันเกี่ยวกับการแสดงข้อผิดพลาด
- <float.h> ฟังก์ชันเกี่ยวกับขอบเขตของระบบ
- <limit.h> ฟังก์ชันเกี่ยวกับขอบเขตของระบบ
- <locale.h> ฟังก์ชันเกี่ยวกับข้อมูลจำเพาะ
- <math.h> ฟังก์ชันเกี่ยวกับคณิตศาสตร์
- <setjmp.h> ฟังก์ชันเกี่ยวกับการกระโดดนอกท้องถิ่น
- <signal.h> ฟังก์ชันเกี่ยวกับสัญญาณ
- <stdarg.h> ฟังก์ชันเกี่ยวกับอาร์กิวเมนต์แปรความยาวได้
- <stddef.h> ฟังก์ชันเกี่ยวกับการนิยามการใช้ทั่วไป
- <stdio.h> ฟังก์ชันเกี่ยวกับการนำเข้าและออก
- <stdlib.h> ฟังก์ชันอรรถประโยชน์อื่นๆ
- <string.h> ฟังก์ชันเกี่ยวกับสายอักขระ
- <time.h> ฟังก์ชันเกี่ยวกับเรื่องวันและเวลา

ภายในเฮดเดอร์ไฟล์จะประกอบด้วยฟังก์ชันการทำงานพื้นฐานที่สามารถเรียกใช้ได้เมื่อมีการประกาศเฮดเดอร์ไฟล์ไว้ที่ส่วนหัวของโค้ดโปรแกรม ดังตัวอย่าง

```
#include <stdio.h>
#include <stdlib.h>
int main{
    /.../
    {
```


3.4 ฟังก์ชันที่ไม่ปลอดภัยในสแตนด์ดาร์ดซีไลบรารี

ภายในสแตนด์ดาร์ดซีไลบรารีจะมีฟังก์ชันที่เกี่ยวกับการใช้งานในหน่วยความจำ เวลาที่ถูกใช้งานจะไม่ได้ทำการตรวจสอบข้อมูลอินพุตก่อน สามารถทำให้เกิดการโอเวอร์โฟลว์ได้ ซึ่งจะเป็นช่องทางของผู้บุกรุกในการโจมตีระบบได้ ฟังก์ชันดังกล่าวเป็นฟังก์ชันที่ไม่ปลอดภัยในเวลาเรียกใช้ ผู้ดูแลระบบต้องมีการป้องกัน โดยในโครงงานนี้จะทำหน้าที่ในการจัดการป้องกันในจุดๆนี้

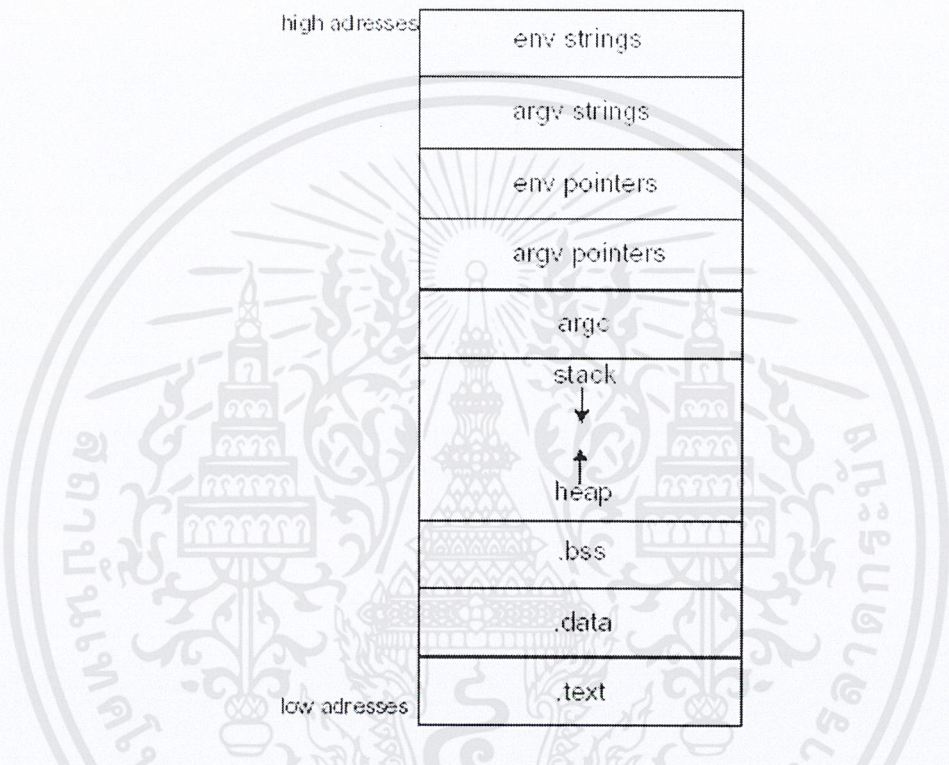
ฟังก์ชัน	ปัญหา
strcpy(char *dest, const char *src)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ dest
strcat(char *dest, const char *src)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ dest
getwd(char *buf)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ buf
gets(char *s)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ s
fscanf(FILE *stream, const char *format, ...)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ที่อาร์กิวเมนต์
scanf(const char *format, ...)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ที่อาร์กิวเมนต์
realpath(char *path, char resolved path[])	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ path
sprintf(char *str, const char *format, ...)	อาจเกิดโอเวอร์โฟลว์ที่บัฟเฟอร์ str

ตารางที่ 3.1 ตัวอย่างของฟังก์ชันที่ไม่ปลอดภัยในสแตนด์ดาร์ดซีไลบรารี

บทที่4

โปรเซสเมโมรีและบัฟเฟอร์โอเวอร์โฟลว์

4.1 โครงสร้างของโปรเซสเมโมรี



รูปที่ 4.1 โครงสร้างของโปรเซสเมโมรี

ในส่วนของแอดเดรสสูงจะประกอบด้วยสภาพแวดล้อมของโปรเซส (process environment) และอาร์กิวเมนต์(argument)

ในส่วนต่อไปของหน่วยความจำจะแบ่งออกเป็น 2 ส่วน คือส่วนของสแตก(stack) และฮีป(heap) ซึ่งจะถูกใช้งานตอนเวลารัน

สแตก จะใช้เก็บค่าของอาร์กิวเมนต์ตัวแปรโลคอล(Local Variable) และข้อมูลบางประเภทที่ต้องการเก็บไว้ในสแตกก่อนมีการเรียกฟังก์ชันคอล สแตกทำงานในพื้นที่ฐานของเข้าก่อน ออกหลัง (LIFO:Last In,First Out)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

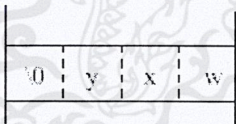
ฮิป/ ใช้งานในขณะรันไทม์เพื่อกำหนดพื้นที่หน่วยความจำให้เพียงพอต่อความต้องการใช้งาน และใช้เก็บข้อมูลพอยน์เตอร์โดยพอยน์เตอร์จะชี้มาที่แอดเดรสในส่วนของฮิป

- .bss และ .data จะเก็บตัวแปรโกลบอล(Global Variable) โดย
- .bss จะเก็บข้อมูลประเภท uninitialized data
- .data จะเก็บข้อมูลประเภท initialized data
- และส่วนสุดท้าย
- .text ใช้เก็บภาษาเครื่อง (machine instruction) ที่ใช้รัน โดยปกติถูกกำหนดให้มีเพอร์มิสชันเป็น read only ดังนั้นโปรแกรมจึงไม่สามารถแก้ไขข้อมูลส่วนนี้ได้

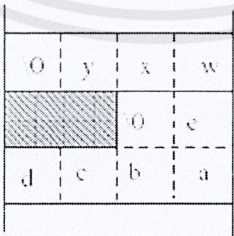
4.2 บัฟเฟอร์ และการเกิดบัฟเฟอร์โอเวอร์โฟลว์

ในภาษาซี สตริงหรือบัฟเฟอร์จะถูกอ้างโดยพอยน์เตอร์ชี้ไปยังแอดเดรสที่ตำแหน่งไบต์แรก และจะพิจารณาว่าถึงจุดสุดท้ายของสตริงหรือบัฟเฟอร์นั้นๆเมื่อเจอตำแหน่งบัต์ (NULL byte) นั้นหมายความว่าไม่มีทางที่จะกำหนดขนาดเนื้อที่ในหน่วยความจำให้พอดีแน่นอนกับเนื้อที่ที่บัฟเฟอร์ใช้งาน ซึ่งทั้งหมดนั้นจะขึ้นอยู่กับจำนวนคาแร็กเตอร์ของข้อมูลที่จะใส่ในบัฟเฟอร์

อย่างแรกเราจะมาดูว่าบัฟเฟอร์ถูกจัดสรรในหน่วยความจำอย่างไร จากรูปภาพประกอบตัวอย่างแรกจะแสดงการเก็บของบัฟเฟอร์ “wxy” รูปที่สองจะแสดงการเก็บของบัฟเฟอร์ “wxy” และ “abcde” ที่ต่อเนื่องกัน



รูปที่ 4.2 บัฟเฟอร์ “wxy” ในหน่วยความจำ



รูปที่ 4.3 บัฟเฟอร์ “wxy” และ “abcde” ในหน่วยความจำ



รูปที่ 4.4 เนื้อที่ที่ไม่ได้ใช้

จากตัวอย่างจะเห็นว่าจะมีอยู่ 2 ไบต์ที่ไม่ได้ใช้เพราะเวลาใส่ข้อมูลนั้นจะใส่เป็นจำนวนเวิร์ด (words) ซึ่งจะเท่ากับ 4 ไบต์ ดังนั้นบัฟเฟอร์ขนาด 6 ไบต์ก็จะใช้เนื้อที่ในหน่วยความจำ 2 เวิร์ด ซึ่งเท่ากับ 8 ไบต์ และทำให้มีเนื้อที่ที่ไม่ได้ใช้ 2 ไบต์

ต่อไปจะแสดงตัวอย่างโค้ดอย่างง่ายๆที่จะโจมตีจุดอ่อนของบัฟเฟอร์

```
#include <stdio.h>

int main(int argc, char **argv){
char jayce[4]="Oum";
char herc[8]="Gillian";
strcpy(herc, "BrookFlora");
printf("%s\n", jayce);
return 0;
}
```

บัฟเฟอร์สองตัวถูกใส่เข้าไปในสแตก ดังรูป ทันทีที่เคาเร็กเตอร์10 ตัวถูกถือปี่ไปที่บัฟเฟอร์ herc ซึ่งได้จองเนื้อที่ไว้ 8 ไบต์ จะทำให้ค่าในบัฟเฟอร์ jayce เกิดการเปลี่ยนแปลง

O	m	u	O
O	n	a	i
I	I	i	G

รูปที่ 4.5 โครงสร้างสแตกเริ่มต้น

O	O	a	r
o	I	F	k
o	o	r	B

รูปที่ 4.6 ภายหลังการเกิดโอเวอร์โฟลว์

ผลลัพธ์ของการเกิดโอเวอร์โฟลว์

```
เมื่อทำการรันโปรแกรมและสังเกต
alfred@atlantis:~$ gcc jayce.c
alfred@atlantis:~$ ./a.out
ra
alfred@atlantis:~$
```

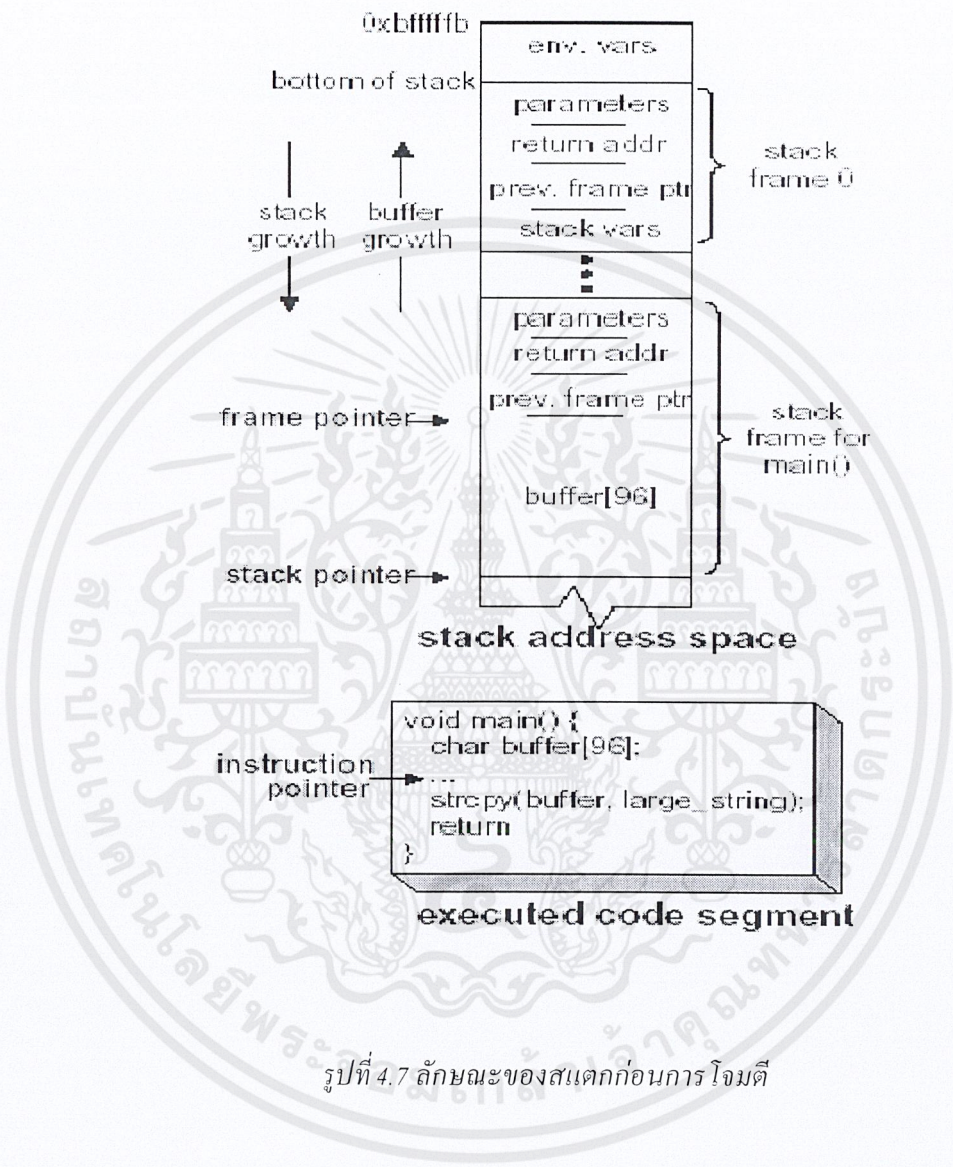

4.3 การโจมตีด้วยวิธีบัฟเฟอร์โอเวอร์โฟลว์

หลักการโจมตีด้วยวิธีการบัฟเฟอร์โอเวอร์โฟลว์นั้นคือการพยายามที่จะใส่ค่าข้อมูลที่ใหญ่มากเกินกว่าที่ตัวแปรของโปรแกรมจะรับได้ ทำให้ข้อมูลที่เกินเข้าไปสั่นหรือโอเวอร์โฟลว์อยู่บนสแตก และโดยการทำเช่นนี้ผู้บุกรุกจะสามารถเอ็กซ์คิวต์โค้ดคำสั่งใดๆก็ได้ตามต้องการ

ตัวอย่างโค้ดการโจมตี ดังนี้

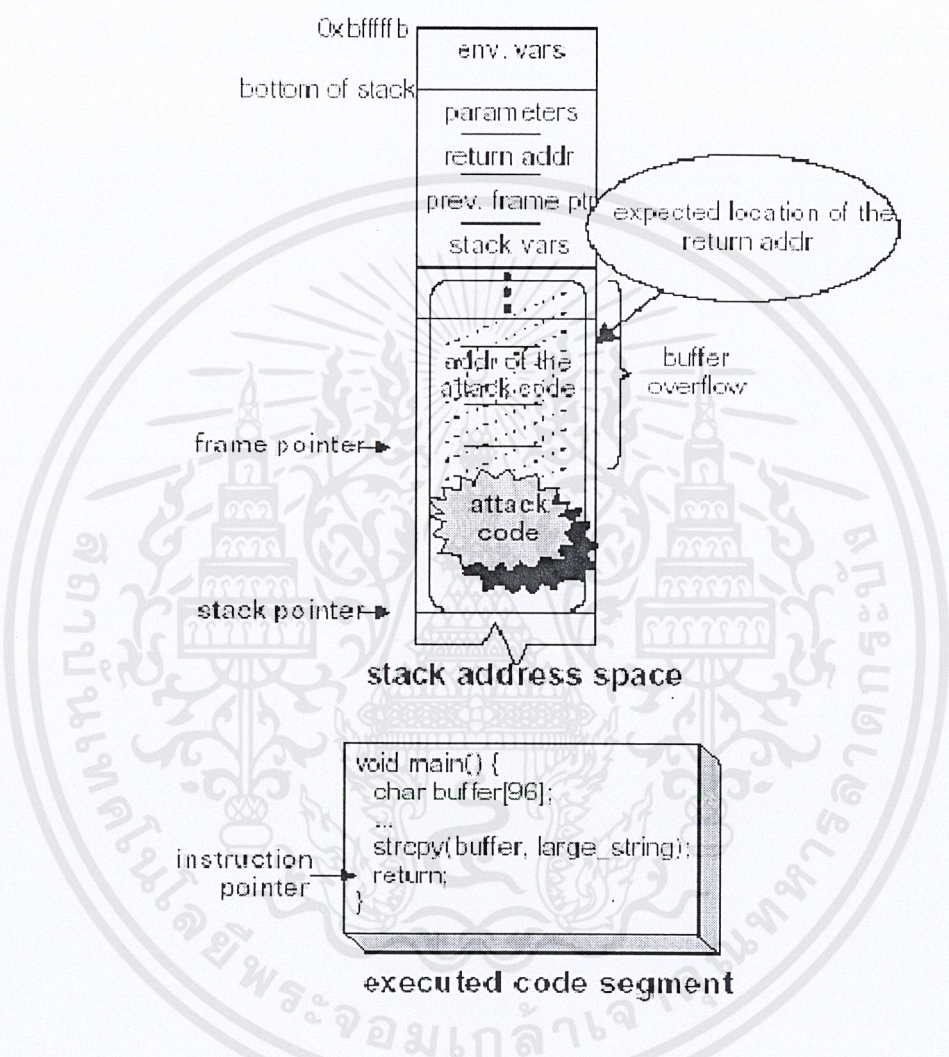
```
#include <stdio.h>
char shellcode[ ] =
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";
char large string[128];
int i;
long *long ptr;
int main() {
char buffer[96];
long ptr = (long *)large string;
for (i=0; i<32; i++)
*(long ptr+i) = (int)buffer;
for (i=0; i<(int)strlen(shellcode); i++)
large string[i] = shellcode[i];
strcpy(buffer, large string);
return 0;
}
```


ลักษณะของข้อมูลบนสแตกก่อนที่จะทำการ โจมตีดังรูป 4.7 โปรแกรมทำงานตามปกติ



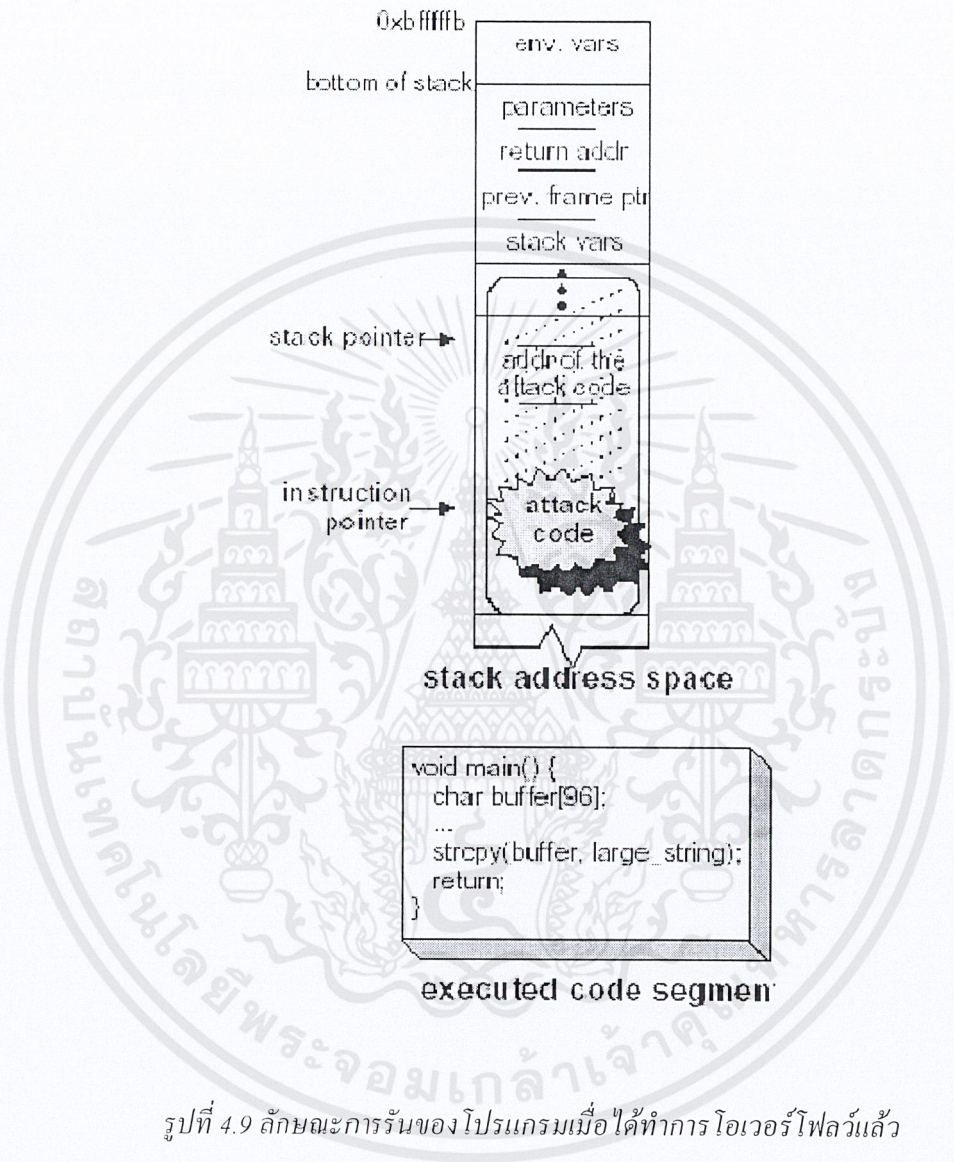
รูปที่ 4.7 ลักษณะของสแตกก่อนการ โจมตี

หลังจากโค้ดที่ผู้บุกรุกเตรียมเอาไว้ได้ถูกบรรจุลงในสแตกแล้วข้อมูลจะเกิดการโอเวอร์โฟลว์จนไปทับค่ารีเทิร์นแอดเดรส



รูปที่ 4.8 ลักษณะข้อมูลของสแตกหลังจากที่เกิดการเวอร์โฟลว์แล้ว

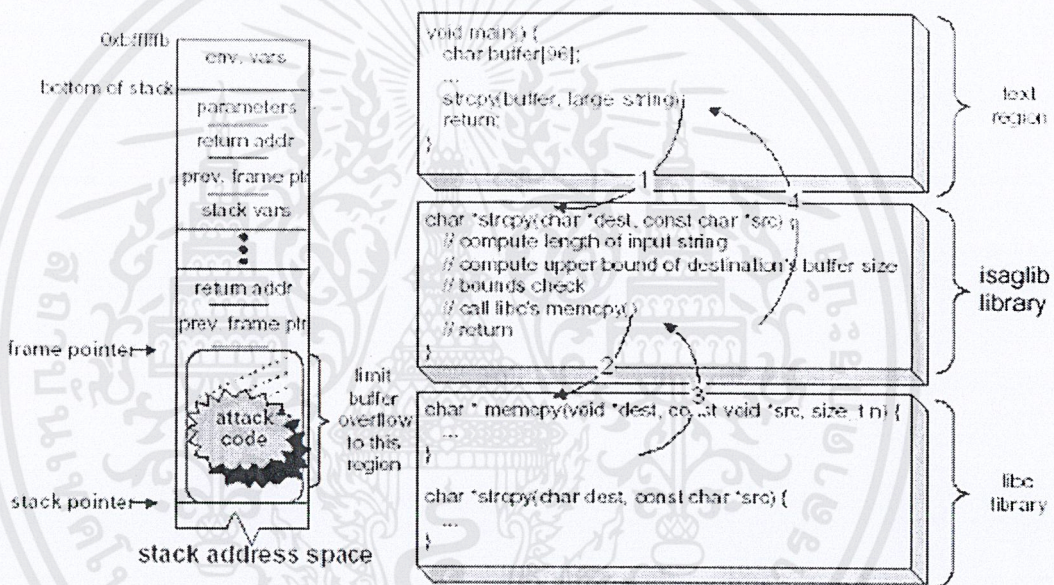
สุดท้ายเมื่อฟังก์ชันได้รับเทิร์นค่ากลับมาจारीเทิร์นเป็นค่าที่ผู้บุกรุกได้เตรียมไว้ให้ไปรันโค้ดในส่วน
ของที่ได้วางเอาไว้ในสแตกทำให้เกิดการรันโค้ดได้ตามที่ผู้บุกรุกต้องการ



รูปที่ 4.9 ลักษณะการรันของโปรแกรมเมื่อได้ทำการโอเวอร์โฟลว์แล้ว

4.4 การแก้ปัญหาการเกิดบัฟเฟอร์โอเวอร์โฟลว์

ในโครงงานนี้ได้นำเสนอวิธีการตรวจจับและป้องกันการเกิดบัฟเฟอร์โอเวอร์โฟลว์จากการเรียกใช้งานฟังก์ชันที่ไม่ปลอดภัยในสแตนด์การ์ดซีไลบรารีฟังก์ชัน โดยจะทำการโอเวอร์ไรด์ฟังก์ชันในสแตนด์การ์ดซีไลบรารี โดยฟังก์ชันที่ได้ทำการโอเวอร์ไรด์เข้าไบนั้นจะทำการตรวจสอบค่าอาร์กิวเมนต์ที่ผ่านเข้ามาในฟังก์ชันแล้วทำการตรวจสอบตำแหน่งของบัฟเฟอร์ในสแตนด์การ์ดก่อนที่จะมีการทำงานของฟังก์ชัน ซึ่งถ้าตรวจพบการเกิดบัฟเฟอร์โอเวอร์โฟลว์ก็จะทำการระงับการเรียกใช้ฟังก์ชันนั้นๆ ซึ่งอาจจะเกิดจากความตั้งใจหรือไม่ตั้งใจก็ได้แต่ระบบก็จะทำการบันทึกประวัติของผู้ต้องสงสัยในการโจมตีระบบลงในล็อกไฟล์ที่ได้ทำการเขียนขึ้นเองโดยเฉพาะ



รูปที่ 4.10 ลักษณะการป้องกันการเกิดโอเวอร์โฟลว์

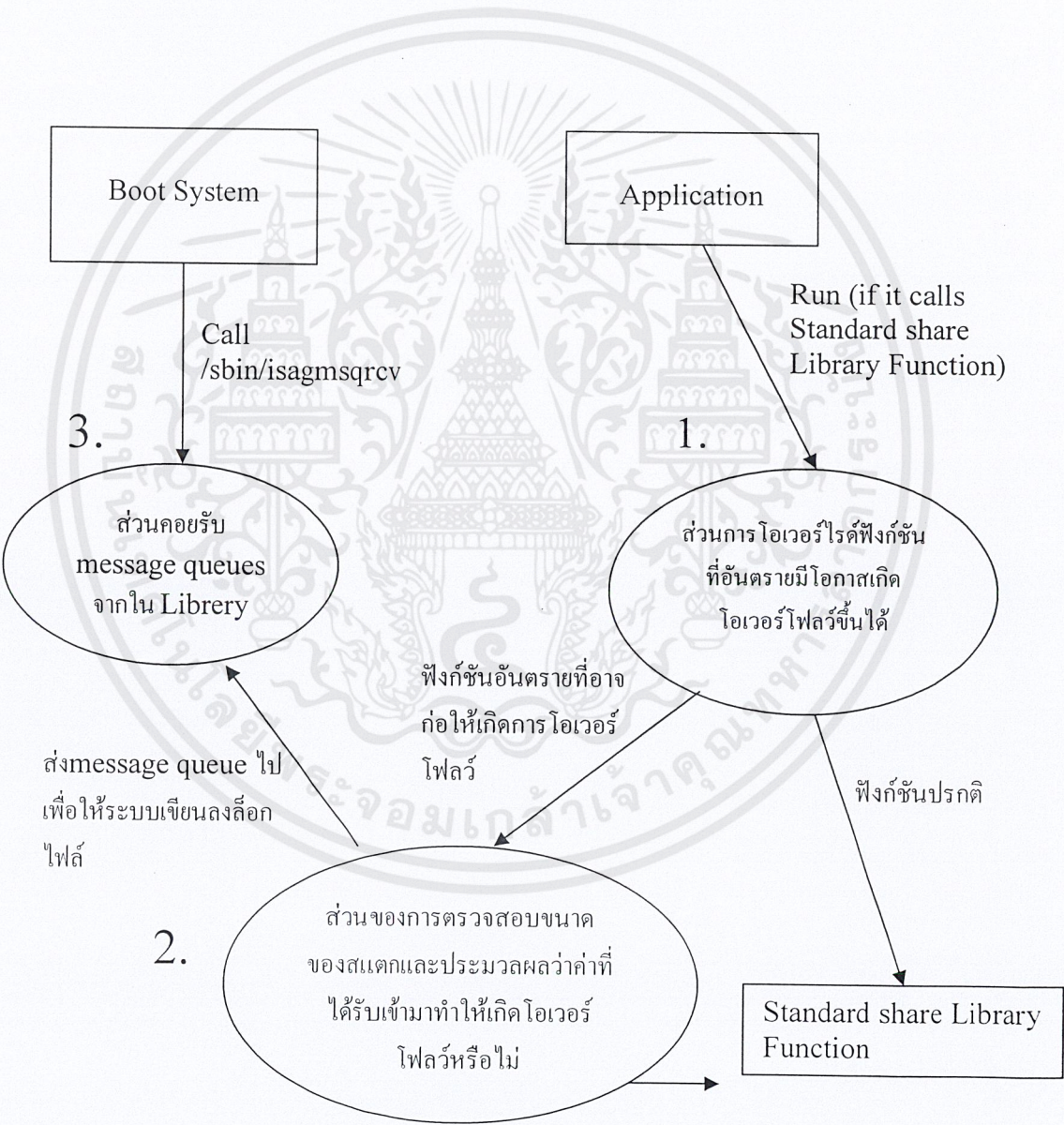
บทที่ 5

โครงสร้างการทำงานของระบบ

5.1 ภาพรวมและโครงสร้างของระบบตรวจจับผู้บุกรุก

ระบบตรวจจับที่สร้างขึ้นนี้ประกอบไปด้วยส่วนใหญ่อะไร 3 ส่วนคือ

- ส่วนของการโอเวอร์ไรด์ฟังก์ชัน
- ส่วนของการตรวจสอบสแตก
- ส่วนของการตอบสนอง(ส่วนของการรายงานผลลงล็อกไฟล์)



รูปที่ 5.1 โครงสร้างและภาพรวมของโปรแกรม

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

5.2 ส่วนการโอเวอร์ไรต์ไลบรารีฟังก์ชัน

ในส่วนนี้จะทำให้โปรแกรมทุกๆโปรแกรมที่ใช้แชร์ไลบรารีต้องมาโหลดเอาไลบรารีนั้นคือฟังก์ชันที่เราต้องการโอเวอร์ไรต์ ซึ่งเป็นฟังก์ชันอันตรายเสี่ยงต่อการทำบัฟเฟอร์โอเวอร์โฟลว์ได้แก่ strcpy(), strncpy(), strcat(), stncat(), getwd(), gets(), realpath(), stpcpy(), memcpy() และsprintf() ซึ่งฟังก์ชันที่กล่าวมานี้เสี่ยงต่อการเกิดโอเวอร์โฟลว์ได้ ดังนี้คือ

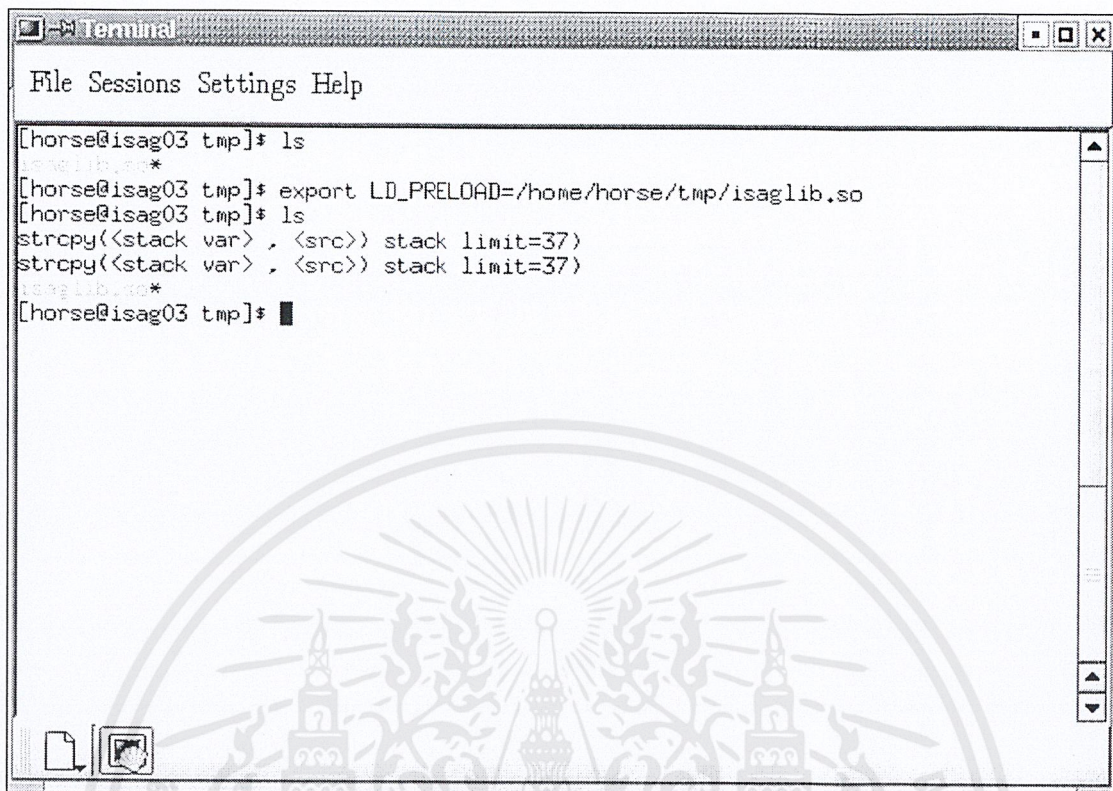
รูปแบบฟังก์ชัน	ปัญหาที่อาจจะเกิดขึ้น
strcpy(char *dest,const char*src)	- อาจจะ โอเวอร์โฟลว์ที่ dest บัฟเฟอร์
strcat(char *dest,const char*src)	- อาจจะ โอเวอร์โฟลว์ที่ dest บัฟเฟอร์
stpcpy(char *dest,const char*src)	- อาจจะ โอเวอร์โฟลว์ที่ dest บัฟเฟอร์
memcpy(char *dest,const char*src)	- อาจจะ โอเวอร์โฟลว์ที่ dest บัฟเฟอร์
getwd(char *buf)	- อาจจะ โอเวอร์โฟลว์ที่ buf บัฟเฟอร์
gets(char *s)	- อาจจะ โอเวอร์โฟลว์ที่ s บัฟเฟอร์
realpath(char *path,char resolved_path[])	- อาจจะ โอเวอร์โฟลว์ที่ path บัฟเฟอร์
sprintf(char *str,const car *format,...)	- อาจจะ โอเวอร์โฟลว์ที่ str บัฟเฟอร์

ทำให้ทุกๆแอปพลิเคชันที่ต้องเรียกใช้ฟังก์ชันดังที่ได้กล่าวมาข้างต้นนั้นเรียกผ่านทางไลบรารีของเราแทน เนื่องจากในสแตนด์การ์ดไลบรารีนั้นไม่มีการตรวจสอบขนาดของบัฟเฟอร์ก่อนที่จะทำการเรียกใช้ฟังก์ชันต่างๆ ดังนั้นในฟังก์ชันที่เราได้ทำการโอเวอร์ไรต์นั้น เราจะทำการตรวจสอบขนาดของสแต็กก่อนที่จะทำการเรียกฟังก์ชันนั้นๆ

การทำให้แอปพลิเคชันต่างๆในระบบของเซลนั้นๆ ไปโหลดในไลบรารีของเราก่อนที่จะเข้าไปโหลดในสแตนด์การ์ดแชร์ไลบรารีจริงๆนั้นทำได้โดยใช้คำสั่ง

```
Sexport LD_PRELOAD=../library
```

โดย library คือชื่อไฟล์ที่เราต้องการให้ไปโหลดไว้ก่อน ในกรณีนี้การโหลดที่เกิดขึ้นนี้จะมีผลอยู่ในเซลนั้นๆเพียงเซลเดียวเท่านั้น เซลอื่นๆ จะยังไปโหลดเอาสแตนด์การ์ดไลบรารีเดิมอยู่ ในการเขียนไลบรารีขึ้นมาเพื่อทดสอบการทำงานว่าไลบรารีของเราทำงานถูกต้องหรือไม่นั้น ควรใช้การโหลดแชร์ไลบรารีอย่างนี้ เพื่อป้องกันไม่ใหัระบบทั้งระบบเกิดการดำเนินงานที่ผิดพลาดขึ้น ดังรูปตัวอย่าง



```

Terminal
File Sessions Settings Help

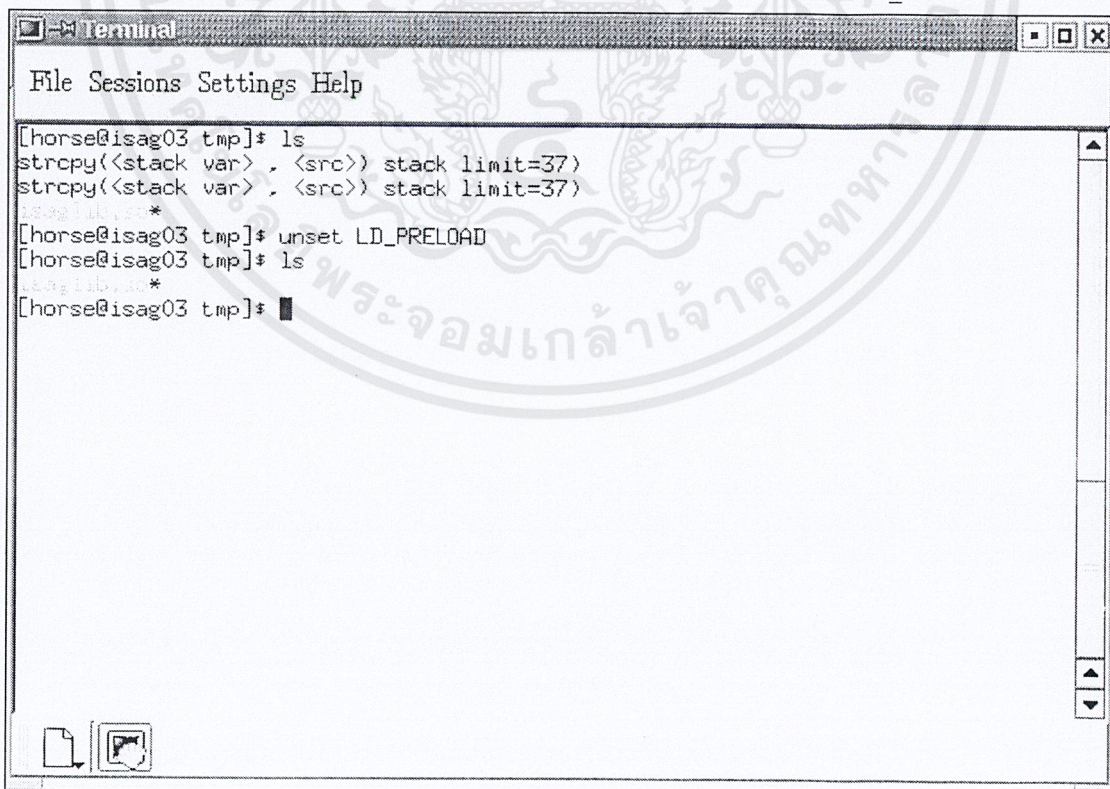
[horse@isag03 tmp]$ ls
isaglib.so*
[horse@isag03 tmp]$ export LD_PRELOAD=/home/horse/tmp/isaglib.so
[horse@isag03 tmp]$ ls
strcpy(<stack var> , <src>) stack limit=37)
strcpy(<stack var> , <src>) stack limit=37)
isaglib.so*
[horse@isag03 tmp]$ █

```

รูปที่ 5.2 แสดงการโหลดไลบรารี

เมื่อไลบรารีของเราเกิดการทำงานผิดพลาดขึ้น ให้ใช้คำสั่ง

Sunset LD_PRELOAD



```

Terminal
File Sessions Settings Help

[horse@isag03 tmp]$ ls
strcpy(<stack var> , <src>) stack limit=37)
strcpy(<stack var> , <src>) stack limit=37)
isaglib.so*
[horse@isag03 tmp]$ unset LD_PRELOAD
[horse@isag03 tmp]$ ls
strcpy(<stack var> , <src>) stack limit=37)
strcpy(<stack var> , <src>) stack limit=37)
isaglib.so*
[horse@isag03 tmp]$ █

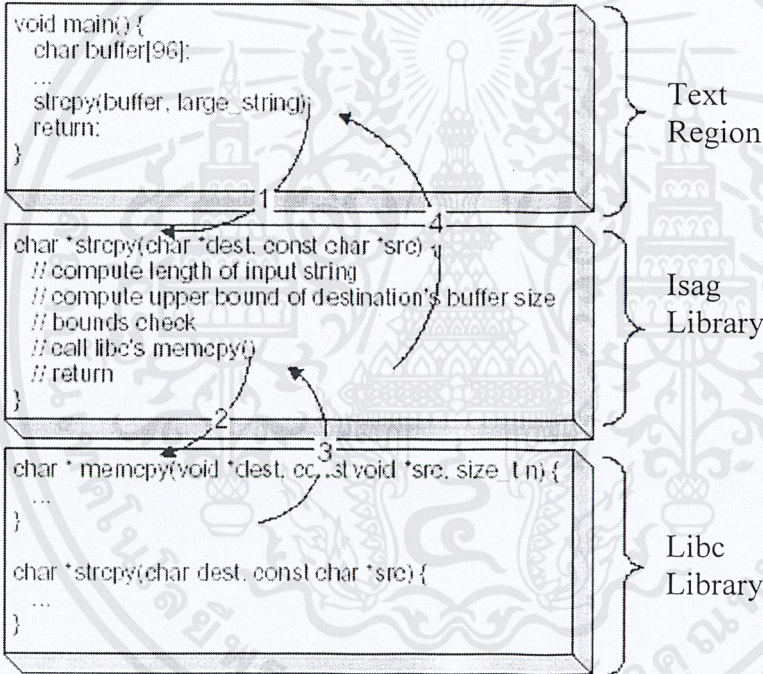
```

รูปที่ 5.3 แสดงการยกเลิกการโหลดไลบรารี

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

คำตั้งนี้จะทำการยกเลิกการโหลดแชร์ไลบรารีในเซกชั่นๆ ทำให้ระบบกลับมาโหลดสแตนด์การ์ดแชร์ไลบรารีเดิม เพื่อให้ระบบกลับมาทำงานโดยปกติ

ในการทำให้แอปพลิเคชันต่างๆทั้งหมด มาโหลดที่ไลบรารีเราก่อนเสมอนั้นทำได้โดยไปเขียนที่ไฟล์ /etc/ld.so.preload ว่าจะให้ไปโหลดที่ /lib/isaglib.so ก่อน จากนั้นทุกๆแอปพลิเคชันที่เรียกใช้ในส่วน ofแชร์ไลบรารีก็จะเข้ามาโหลดไลบรารีของเราก่อนเสมอ ในการศึกษาโปรแกรมหากต้องการคอมไพล์ไฟล์ isaglib.so ขึ้นมาใหม่นั้น จะต้องระวังไม่ให้ไฟล์ /lib/isaglib.so เปลี่ยนแปลงเด็ดขาดฉะนั้นแล้วจะทำให้ระบบนั้นๆไม่สามารถทำงานต่อไปได้เลย ถ้าจะทำการคอมไพล์ไฟล์ isaglib.so ใหม่แล้วคัดลอกไปไว้ใน /lib แล้วจะต้องไปแก้ที่ไฟล์ /etc/ld.so.preload ก่อนเพื่อให้ระบบทำงานต่อไปได้ตามปกติ

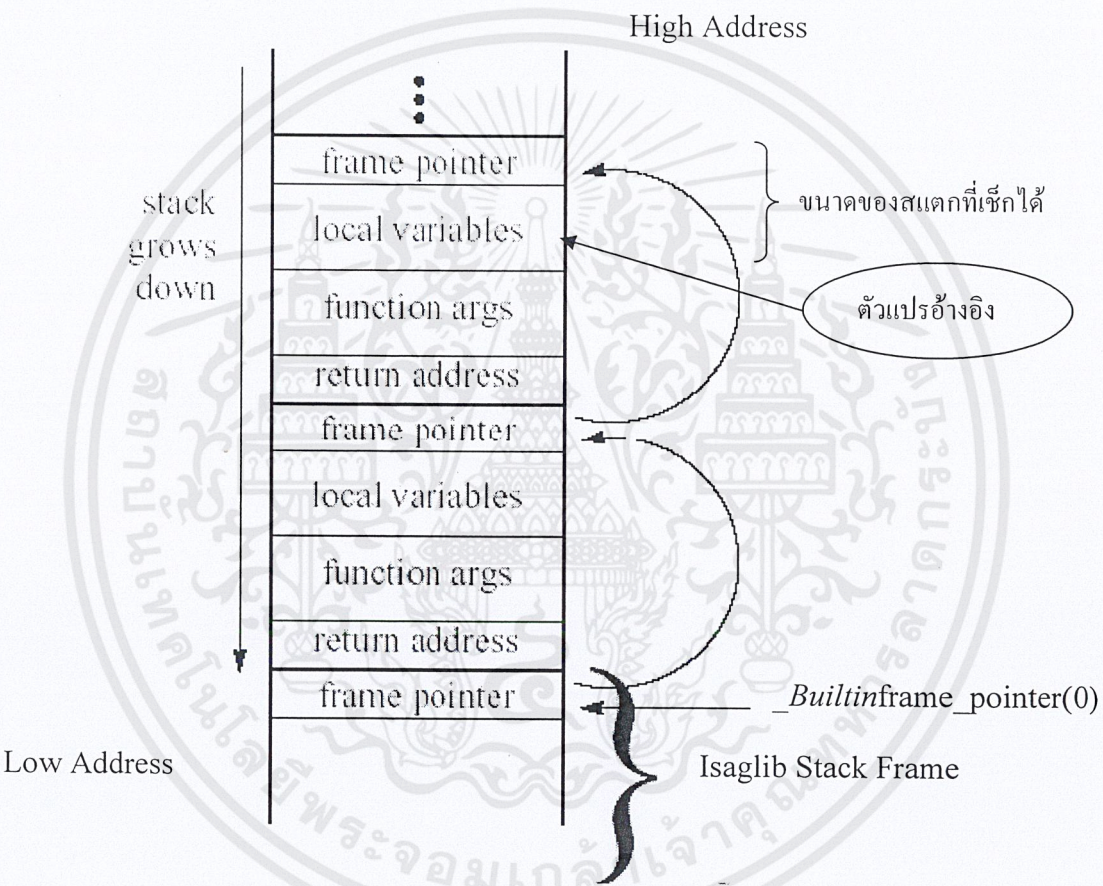


รูปที่ 5.4 การทำงานโปรแกรมต่างๆเมื่อทำการโหลดไลบรารีของเราเข้าไปแล้ว

และถ้าระบบเกิดการโหลดไลบรารีที่ได้เขียนไว้ในไฟล์ /etc/ld.so.preload ไม่ได้ระบบจะเกิดการแฮงค์ คือระบบจะไม่สามารถทำงานต่อไปได้เลยแม้แต่ตอนบูตเครื่อง วิธีแก้คือให้นำเอาแผ่นบูตลินุกซ์ระบบใดก็ได้เข้ามาบูตเครื่องจากนั้นให้เมาท์ฮาร์ดดิสก์และไปลบไฟล์ /etc/ld.so.preload ออกไปจากระบบจากนั้นระบบจึงจะกลับมาทำงานได้ตามปกติ

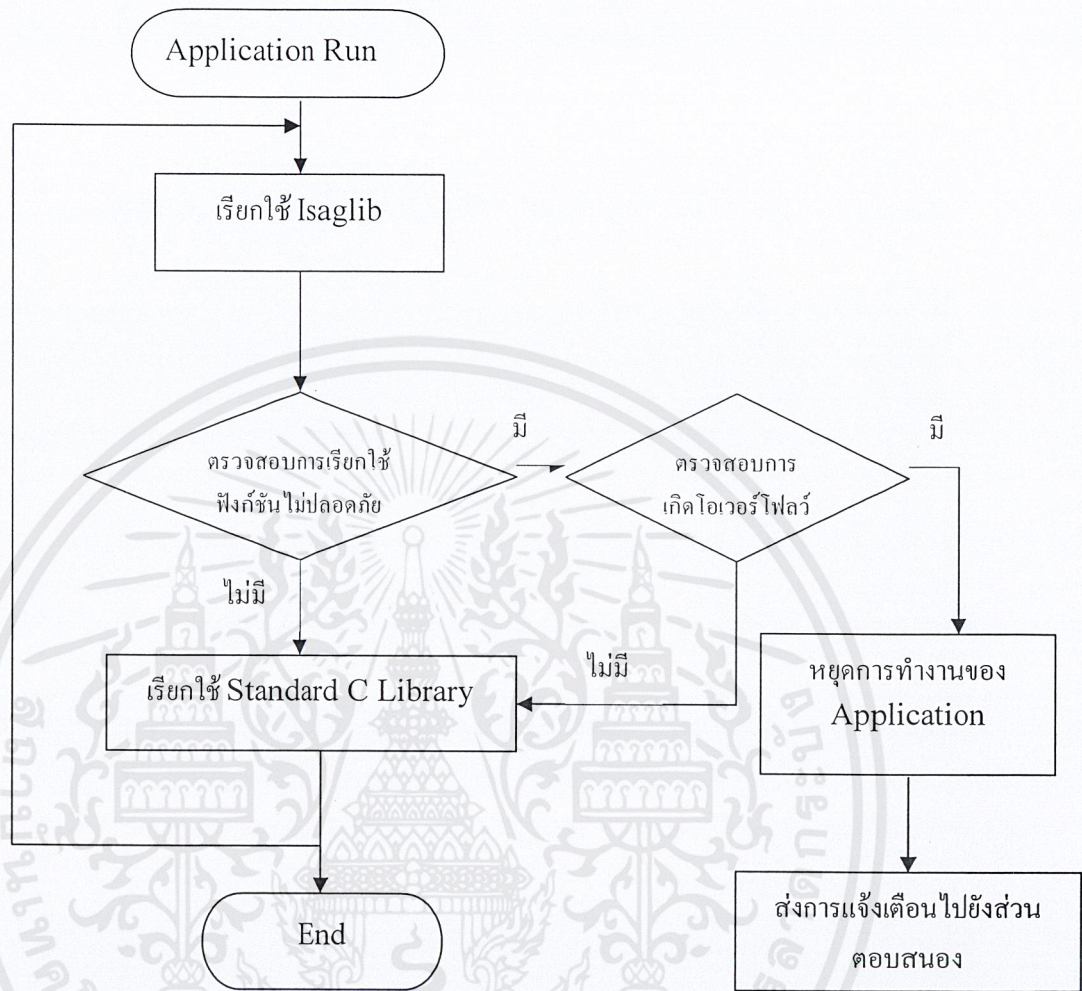
5.3 ส่วนการวิเคราะห์ข้อมูลหรือส่วนการตรวจสอบขนาดของสแตก

ขนาดของสแตกที่เราตรวจสอบได้นี้เป็นขนาดของแอดเดรสอ้างอิงไปยังเฟรมพอยน์เตอร์นั้นๆ เมื่อตัวแปรอ้างอิงนั้นอยู่ในสแตกเท่านั้น เนื่องจากตัวแปรที่ชี้กได้นั้นเป็นขนาด ที่ชี้กได้ระหว่างเฟรมพอยน์เตอร์นั้นกับแอดเดรสอ้างอิงนั้น ดังนั้นในแอปพลิเคชันอาจจะเกิดการโอเวอร์แลประหว่างตัวแปรภายในกันเองได้ซึ่งอาจจะทำให้แอปพลิเคชันนั้นๆเกิดการทำงานผิดพลาดแต่การ โอเวอร์โฟลล์เพื่อไปเปลี่ยนค่ารีเทิร์นแอดเดรสนั้นจะไม่เกิดขึ้น



รูปที่ 5.5 แสดงวิธีการหาขนาดของสแตก

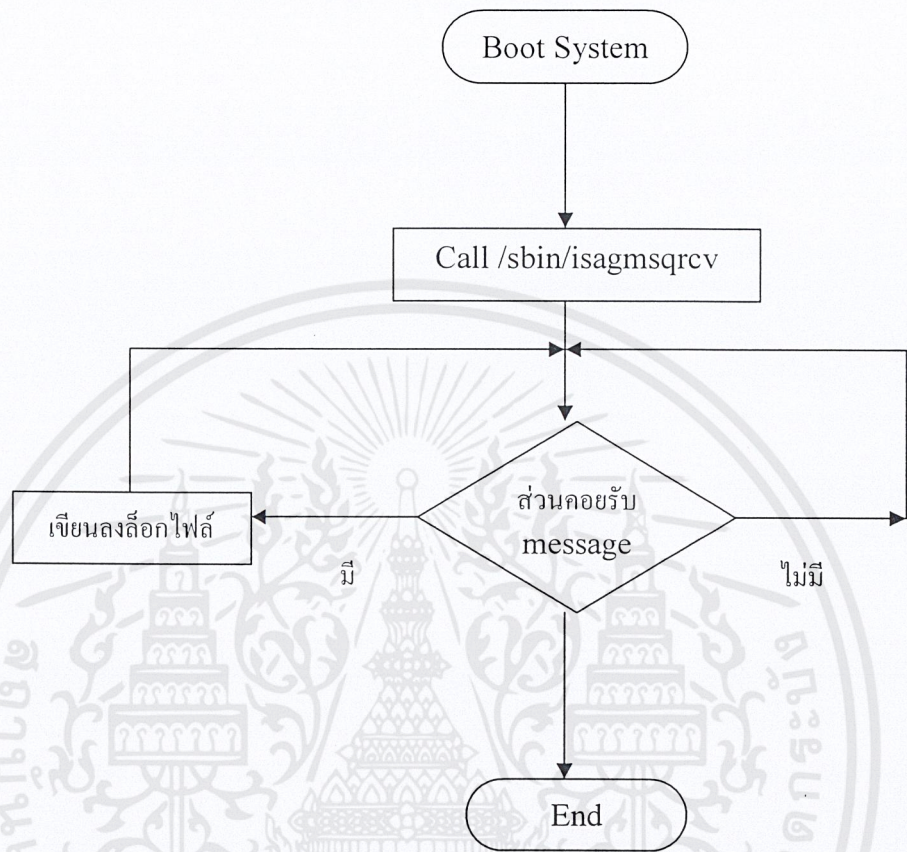
การทำงานในส่วนนี้นั้นจะใช้พอยน์เตอร์ชี้ไปยังเฟรมแอดเดรสนั้นๆ โดยใช้ฟังก์ชัน `_builtin_frame_pointer(0)` ฟังก์ชันนี้จะรีเทิร์นค่าแอดเดรสของเฟรมพอยน์เตอร์นั้นๆกลับมา แล้วนำมาเปรียบเทียบกับแอดเดรสของตัวแปรอ้างอิง ถ้าเฟรมพอยน์เตอร์นั้นมากกว่าแอดเดรสอ้างอิงแสดงว่าเฟรมพอยน์เตอร์นั้นชื่ออยู่ที่เฟรมที่ตัวแปรนั้นอยู่แล้ว เราก็จะทราบค่าขนาดสแตกได้



รูปที่ 5.6 ขั้นตอนการทำงานของส่วนวิเคราะห์ข้อมูลหรือส่วนการตรวจสอบขนาดของสแตก

กรณีที่เช็คนานี้ใช้ได้เฉพาะกับตัวแปรที่อยู่สแตกเท่านั้น ถ้าเกิดว่าตัวแปรไม่ได้อยู่ในสแตกแล้วฟังก์ชันจะรีเทิร์นค่าศูนย์ให้ไลบรารีและไลบรารีรีเทิร์นค่าฟังก์ชันเดิมให้ทำงานตามปกติต่อไป และเมื่อเช็คได้แล้วว่าเกิดการโอเวอร์โฟลว์เกิดขึ้น จะเรียกฟังก์ชันเพื่อทำการส่งเมสเสจไปยังเมสเสจคิวที่ระบบได้เปิดรอรับไว้ เพื่อให้โปรเซสนั้นเขียนลงล็อกไฟล์ต่อไป

5.4 ส่วนการตอบสนองของระบบ



รูปที่ 5.7 ขั้นตอนการทำงานของส่วนของการตอบสนองของระบบ

ในส่วนการตอบสนองนี้เมื่อผู้ใดได้ทำการพยายามโอเวอร์โฟลว์จากฟังก์ชันที่เราโอเวอร์ไรด์ไว้แล้วนั้น เมื่อไลบรารีเช็คว่าเกิดการโอเวอร์โฟลว์ขึ้นจะทำการเรียกฟังก์ชันให้ส่งเมสเสจไปยังเมสเสจคิวที่ระบบได้เปิดไว้ในตอนบูตเครื่อง ทำได้โดยเขียนใส่ไว้ในไฟล์ /etc/rc.sysinit โดยจะมีเดมอนรันเอาไว้เพื่อคอยรับเมสเสจจากเมสเสจคิวนี้ เดมอนที่รันนี้จะรันในสิทธิ์ผู้ดูแลระบบเพื่อใช้เพอร์มิชชันในการเขียนล็อกไฟล์ ดังนั้นจะป้องกันไม่ให้ผู้ธรรมดาแก้ไขล็อกไฟล์ได้ง่าย เมื่อได้รับข้อมูลจากเมสเสจคิวแล้วจะทำการเปิดล็อกไฟล์แล้วเขียนรายละเอียดต่างๆที่ส่งมากับเมสเสจควีนั่นๆลงไปในล็อกไฟล์ ดังนี้คือ

- Warning : เป็นการบอกให้ผู้ดูแลระบบรู้ว่าเกิดการโอเวอร์โฟลว์ขึ้นกับฟังก์ชันอะไร
- UID : ยูสเซอร์ไอดีของผู้ที่ทำการโอเวอร์โฟลว์
- GID : กรุปไอดีของผู้ที่ทำการโอเวอร์โฟลว์
- Application : ชื่อแอปพลิเคชันที่เกิดการโอเวอร์โฟลว์ขึ้น

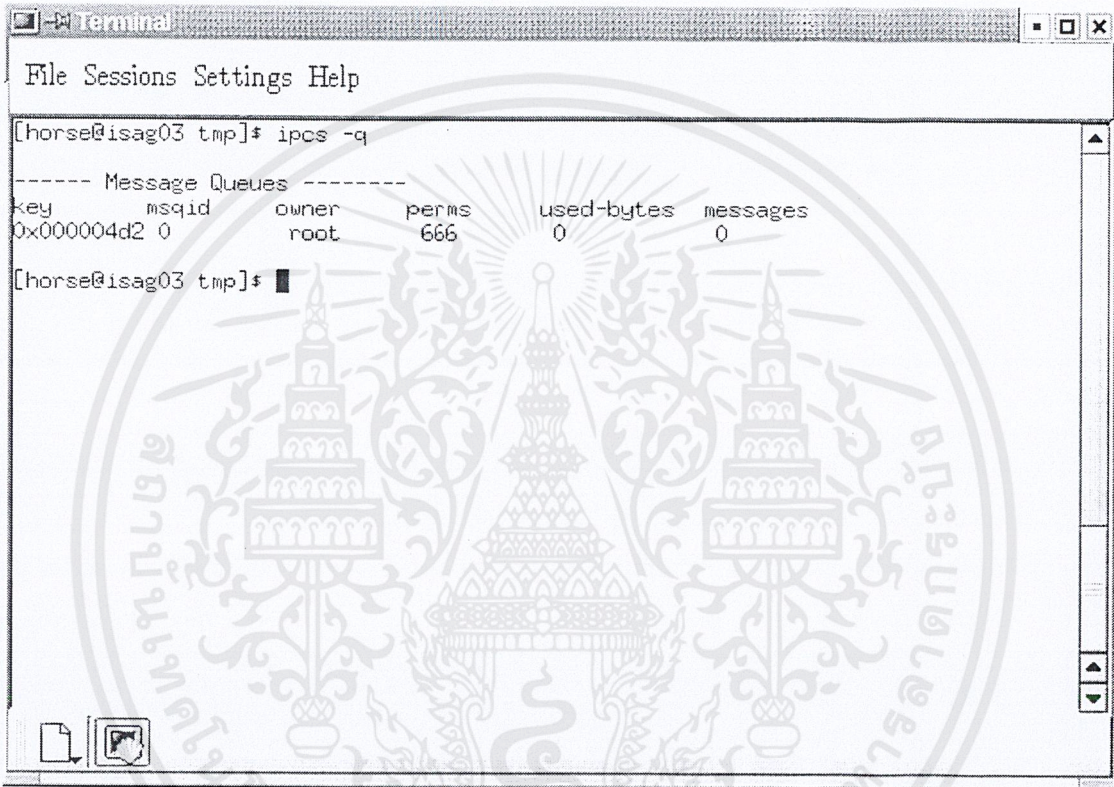
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Time : เวลาที่ได้เกิดกระทำการโอเวอร์โฟลว์

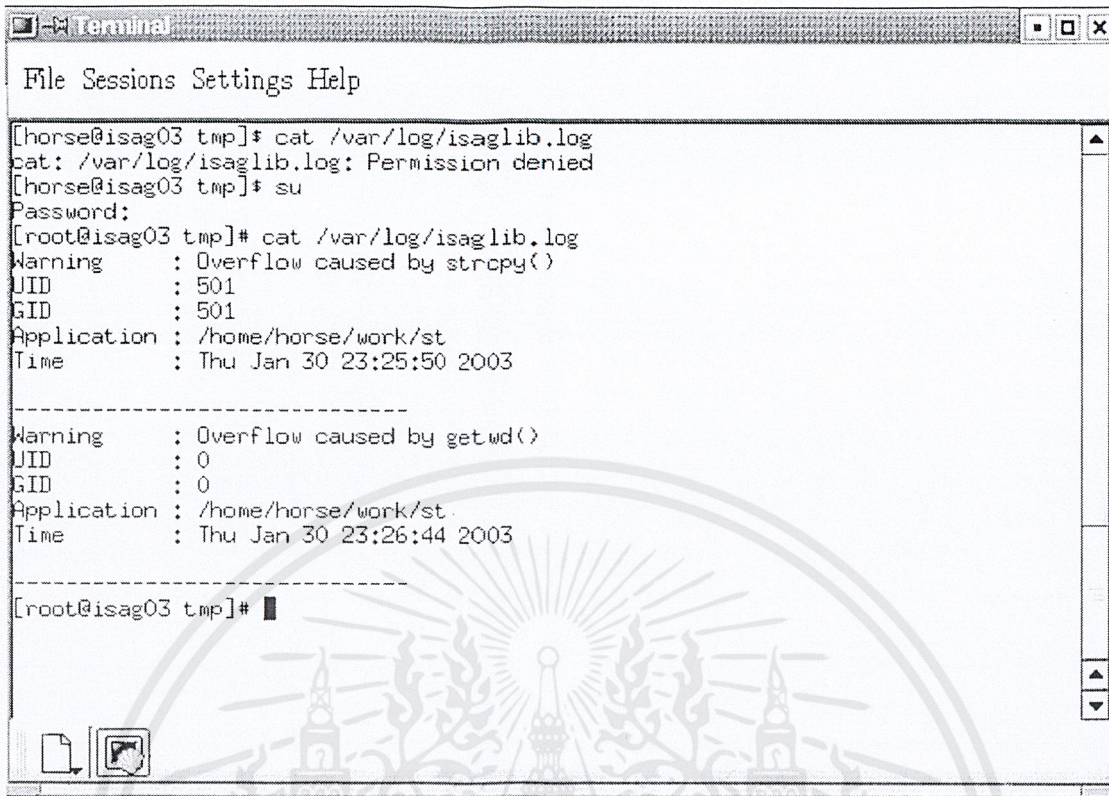
การเช็คเมสเสจคิวมารอรับการโอเวอร์โฟลว์นั้นทำได้โดยใช้คำสั่ง

Sipcs -q

ส่วนล็อกไฟล์นั้นเราจะเขียนลงในไฟล์ /var/log/isaglib.log



รูปที่ 5.8 แสดงเมสเสจคิวที่มารอรับเพื่อที่จะได้เขียนลงไปล็อกไฟล์



```

Terminal
File Sessions Settings Help

[horse@isag03 tmp]$ cat /var/log/isaglib.log
cat: /var/log/isaglib.log: Permission denied
[horse@isag03 tmp]$ su
Password:
[root@isag03 tmp]# cat /var/log/isaglib.log
Warning      : Overflow caused by strcpy()
UID          : 501
GID          : 501
Application  : /home/horse/work/st
Time        : Thu Jan 30 23:25:50 2003

-----
Warning      : Overflow caused by getwd()
UID          : 0
GID          : 0
Application  : /home/horse/work/st
Time        : Thu Jan 30 23:26:44 2003

-----
[root@isag03 tmp]#

```

รูปที่ 5.9 แสดงผลในล็อกไฟล์ที่เกิดขึ้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 6

ทดสอบการทำงานและสรุปผล

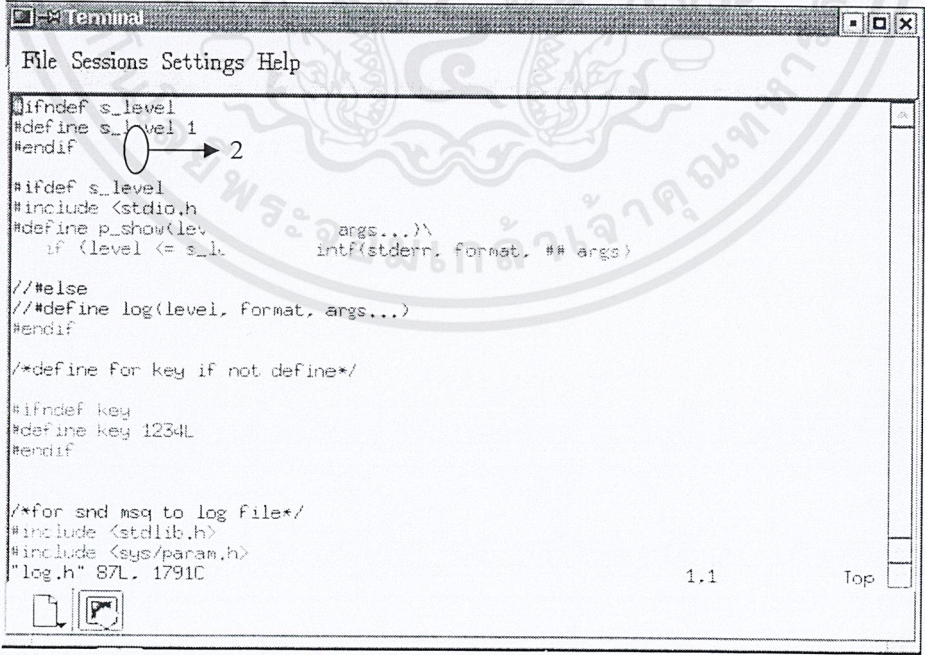
6.1 ทดสอบการทำงานเบื้องต้นของโปรแกรม

ในการทดสอบนี้ได้เขียนโปรแกรมสำหรับทดสอบขึ้นมาโดยเฉพาะ โปรแกรมหนึ่งชื่อ test.c เมื่อรันแล้วตัวโปรแกรมจะมีตัวเลือกให้เราต้องการจะทดสอบการโอเวอร์โฟลว์กับฟังก์ชันอะไร เมื่อได้ทำการเลือกแล้วโปรแกรมจะให้เราใส่ค่าเองเพื่อทดสอบการโอเวอร์โฟลว์

ในตัวโปรแกรมที่เขียนขึ้นมานั้น สามารถที่จะ set ค่า s_level ใน log.h ได้ซึ่งค่านี้จะเป็นตัวไว้ทำไว้เพื่อให้ดีบั๊กโปรแกรมโดยง่าย โดยค่า s_level นั้นสามารถตั้งค่าได้ 4 ระดับ ดังนี้คือ

- s_level = 0 : ป้องกันการโอเวอร์โฟลว์ที่เกิดขึ้นโดยไม่มีการรายงานผลลงล็อกไฟล์
- s_level = 1 : ป้องกันการ โอเวอร์โฟลว์ที่เกิดขึ้นและรายงานผลลงล็อกไฟล์เมื่อเกิดการโอเวอร์โฟลว์ขึ้น
- s_level = 2 : รายงานผลลัพธ์ออกหน้าจอเมื่อเกิดการเรียกใช้ฟังก์ชันที่เป็นอันตรายในการเรียกใช้ฟังก์ชันที่บัฟเฟอร์อยู่ในสแตก
- s_level = 3 : รายงานผลลัพธ์ออกหน้าจอเมื่อเกิดการเรียกใช้ฟังก์ชันที่เป็นอันตรายในการเรียกใช้ฟังก์ชันที่บัฟเฟอร์อยู่ในสแตกและอยู่ในฮีปด้วย

ทำการเซตค่า s_level = 2 เพื่อการทดสอบโปรแกรมเบื้องต้นดังรูปที่ 6.1



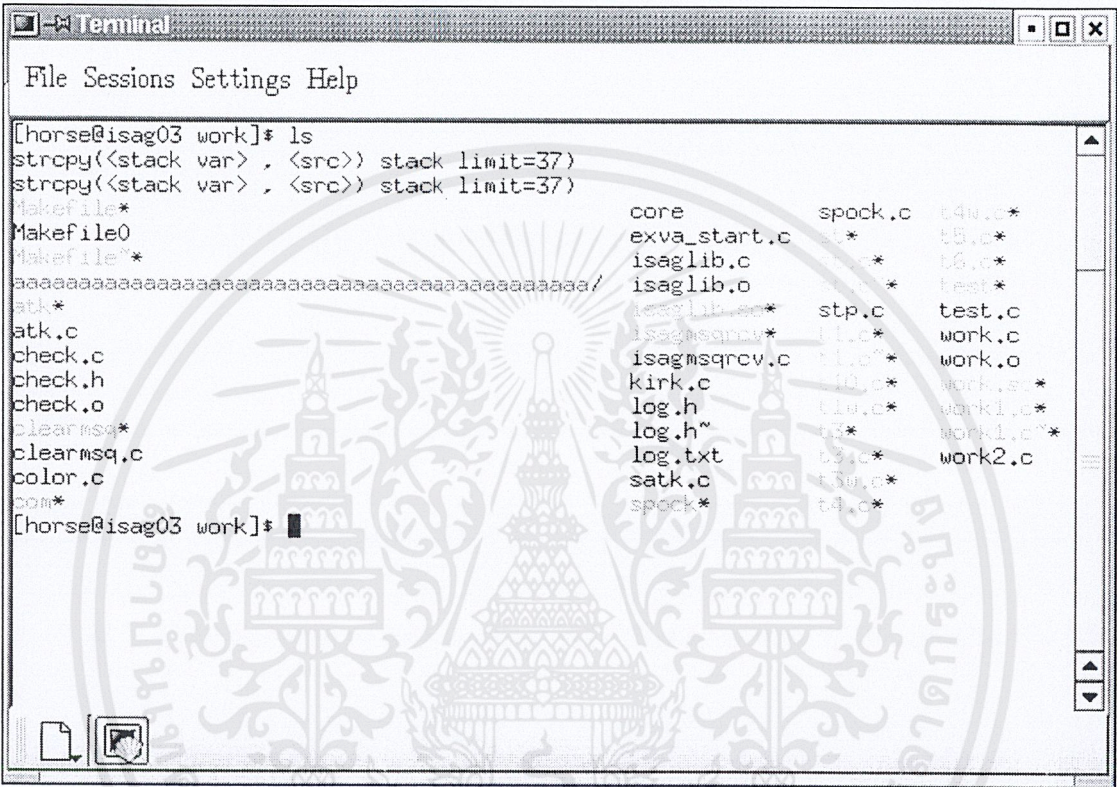
รูปที่ 6.1 การเซตค่า s_level ในไฟล์ log.h ให้เป็น 2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เมื่อได้ทำการเซตค่า s_level เรียบร้อยแล้วทำการคอมไพล์แล้วใช้คำสั่งดังนี้ เพื่อการทดสอบเบื้องต้นโปรแกรม

```
Sexport LD_PRELOAD=/home/horse/work/isaglib.so
```

ลองรันคำสั่ง ls จะได้ผลลัพธ์ดังนี้



รูปที่ 6.2 แสดงการใช้คำสั่ง ls เมื่อทำการ โหลดไลบรารีเรียบร้อยแล้ว

จะเห็นได้ว่าการแสดงคำสั่ง strcpy(...) ขึ้นบนหน้าจอ ทั้งนี้เพราะว่าคำสั่ง ls นั้นเป็นคำสั่งหนึ่ง ที่เรียกใช้แชร์ไลบรารีด้วย การที่เราโหลดไลบรารีเข้าไปใหม่จึงมีผลกับคำสั่ง ls ด้วยจากรูป 6.2 สรุปได้ว่า คำสั่ง ls นั้นมีการเรียกใช้ strcpy() เพื่อคัดลอกไปยังสแตก 2 ครั้ง

จากรูป 6.4 แสดงการทดสอบไลบรารีฟังก์ชัน strcpy() โดยโปรแกรมจะให้รับค่าสตริงเข้ามา แล้วจะทำการเรียกใช้ฟังก์ชัน strcpy() โดยจะทำการคัดลอกข้อมูลขนาดประมาณ 150 ตัวอักษรลงไปใน ตัวแปร buf ซึ่งขนาดของสแตกสูงสุดที่ตรวจสอบได้นี้มีค่าเท่ากับ 120 ไบต์ ส่วนที่แสดงการเรียกฟังก์ชัน strcpy() อีก 2 ครั้งนั้นเป็นการเรียกเพื่อบันทึกข้อมูลลงล็อกไฟล์ของในไลบรารี จากนั้นเมื่อไลบรารีตรวจสอบได้ว่าเกิดโอเวอร์โฟลว์ขึ้นจะทำการส่งสัญญาณ kill ไปทำให้จบโปรแกรมลง


```

File Sessions Settings Help

[horse@isag03 work]$ ./test
++++ Test the new library ++++
+ 0. Test for heap
+ 1. Test the strcpy()
+ 2. Test the strncpy()
+ 3. Test the strcat()
+ 4. Test the strncat()
+ 5. Test the getwd()
+ 6. Test the gets()
+ 7. Test the stpcpy()
+ 8. Test the memcpy()
+ 9. Test the realpath()
+ a. Test the sprintf()
+ Q. quit
+++++
Choose One : █

```

รูปที่ 6.3 แสดงการเรียกใช้โปรแกรม *test* ซึ่งใช้ในการทดสอบการทำงานของไลบรารี

```

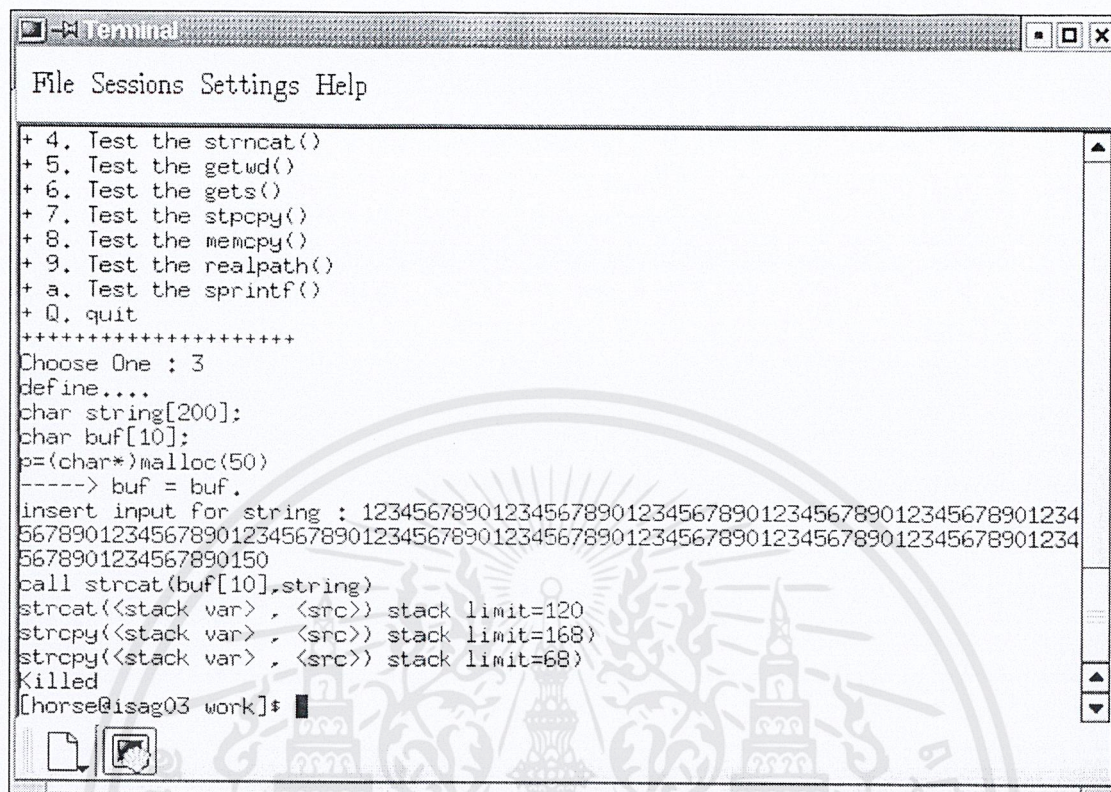
File Sessions Settings Help

+ 3. Test the strcat()
+ 4. Test the strncat()
+ 5. Test the getwd()
+ 6. Test the gets()
+ 7. Test the stpcpy()
+ 8. Test the memcpy()
+ 9. Test the realpath()
+ a. Test the sprintf()
+ Q. quit
+++++
Choose One : 1
define....
char string[200];
char buf[10];
p=(char*)malloc(50)
insert input for string : 123456789012345678901234567890123456789012345678901234
5678901234567890123456789012345678901234567890123456789012345678901234
5678901234567890150
call strcpy(buf[10],string)
strcpy(<stack var> . <src>) stack limit=120)
strcpy(<stack var> . <src>) stack limit=168)
strcpy(<stack var> . <src>) stack limit=68)
Killed
[horse@isag03 work]$ █

```

รูปที่ 6.4 แสดงการเรียกทดสอบฟังก์ชัน *strcpy()*

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



```

Terminal
File Sessions Settings Help

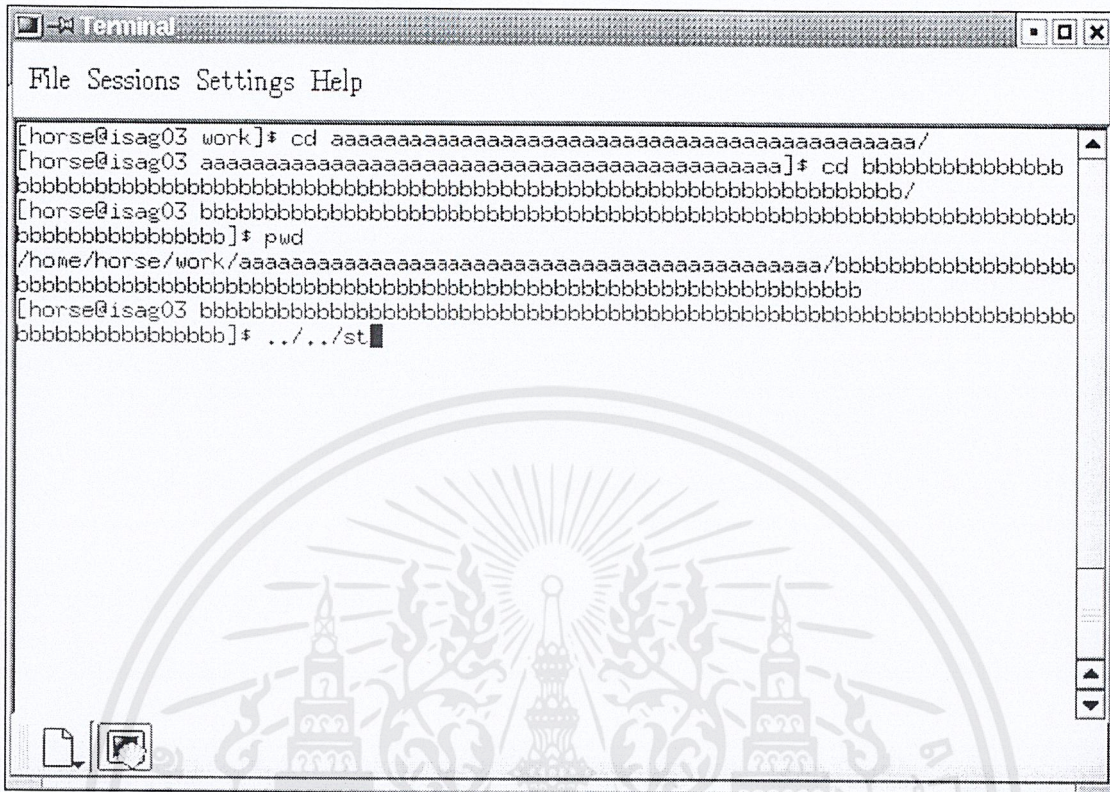
+ 4. Test the strcat()
+ 5. Test the getwd()
+ 6. Test the gets()
+ 7. Test the strcpy()
+ 8. Test the memcpy()
+ 9. Test the realpath()
+ a. Test the sprintf()
+ Q. quit
+++++
Choose One : 3
define....
char string[200]:
char buf[10]:
p=(char*)malloc(50)
-----> buf = buf.
insert input for string : 123456789012345678901234567890123456789012345678901234
5678901234567890123456789012345678901234567890123456789012345678901234
5678901234567890150
call strcat(buf[10],string)
strcat(<stack var> , <src>) stack limit=120
strcpy(<stack var> , <src>) stack limit=168)
strcpy(<stack var> , <src>) stack limit=68)
Killed
[horse@isag03 work]$

```

รูปที่ 6.5 แสดงการทดสอบฟังก์ชัน `strcat()`

จากรูปที่ 6.5 ทดสอบฟังก์ชัน `strcat()` โดยป้อนอินพุตขนาดประมาณ 150 ลงไป จะเห็นได้ว่าตัวไลบรารีนั้นสามารถที่จะจัดการโอเวอร์โฟลว์ได้ดังรูป ในกรณีของ `strcpy()` และ `strncat()` ก็จะเป็นลักษณะเช่นนี้ เหมือนกัน

ในการทดสอบฟังก์ชัน `getwd()` นั้นเราต้องให้เวิร์กไดเรกทอรีปัจจุบันมีค่ายาวมาก ๆ โดยการสร้างไดเรกทอรีจำลองขึ้นมาแล้วเข้าไปในไดเรกทอรีนั้น จากนั้นเรียกใช้โปรแกรมทดสอบ `../test` ดังรูป 6.6 โปรแกรมจะทำการเรียกใช้ฟังก์ชัน `getwd()` โดยจะทำการเก็บไว้ในตัวแปร `buf` ในที่นี้ชื่อของเวิร์กไดเรกทอรีนั้นยาวเกินกว่าที่ตัวแปร `buf` จะรับได้เกิดการโอเวอร์โฟลว์ขึ้น ตัวไลบรารีจึงทำการส่งสัญญาณ `kill` ไปเพื่อจบโปรแกรกดังรูปที่ 6.7



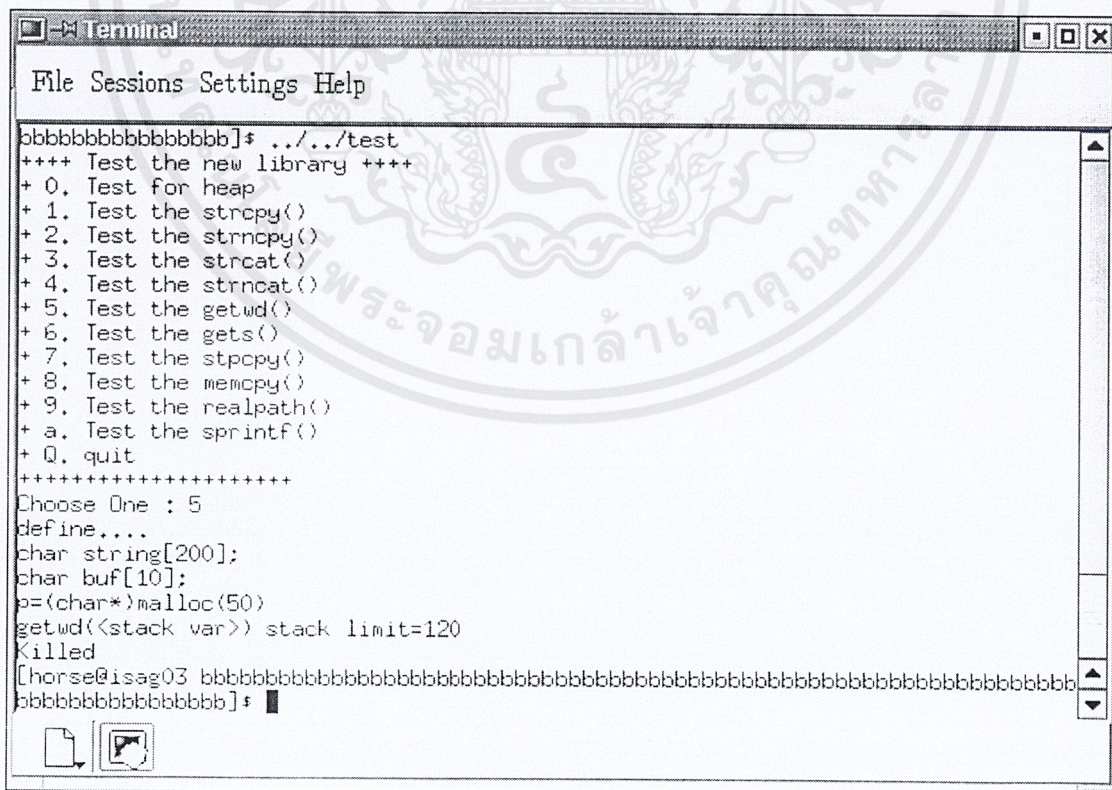
```

Terminal
File Sessions Settings Help

[horse@isag03 work]$ cd aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa/
[horse@isag03 aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa]$ cd bbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb/
[horse@isag03 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbb]$ pwd
/home/horse/work/aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa/bbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
[horse@isag03 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbb]$ ../../st

```

รูปที่ 6.6 แสดงการเข้าไปในไดเรกทอรีที่มีชื่อยาวมากเพื่อการทดสอบฟังก์ชัน `getwd`



```

Terminal
File Sessions Settings Help

bbbbbbbbbbbbbbbbbb]$ ../../test
++++ Test the new library ++++
+ 0. Test for heap
+ 1. Test the strcpy()
+ 2. Test the strncpy()
+ 3. Test the strcat()
+ 4. Test the strncat()
+ 5. Test the getwd()
+ 6. Test the gets()
+ 7. Test the strcpy()
+ 8. Test the memcpy()
+ 9. Test the realpath()
+ a. Test the sprintf()
+ Q. quit
++++
Choose One : 5
define....
char string[200];
char buf[10];
p=(char*)malloc(50)
getwd(<stack var>) stack limit=120
Killed
[horse@isag03 bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb
bbbbbbbbbbbbbbbbbb]$

```

รูปที่ 6.7 แสดงการทดสอบฟังก์ชัน `getwd()` เกิดการโอเวอร์โฟลว์ที่ตัวแปร `buf`
 เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า
 ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้


```

Terminal
File Sessions Settings Help

+ 1. Test the strcpy()
+ 2. Test the strncpy()
+ 3. Test the strcat()
+ 4. Test the strncat()
+ 5. Test the getwd()
+ 6. Test the gets()
+ 7. Test the stpcpy()
+ 8. Test the memcpy()
+ 9. Test the realpath()
+ a. Test the sprintf()
+ Q. quit
+++++
Choose One : 6
define....
char string[200];
char buf[10];
p=(char*)malloc(50)
gets(<stack var>) stack limit=120
test for gets()insert string for input gets(buf) : 12345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789
0123456789012345678901234567890123456789012345678901234567890123456789
-----> buf = 123456789012345678901234567890123456789012345678901234567
89012345678901234567890123456789012345678901234567890123456789
[horse@isag03 work]*

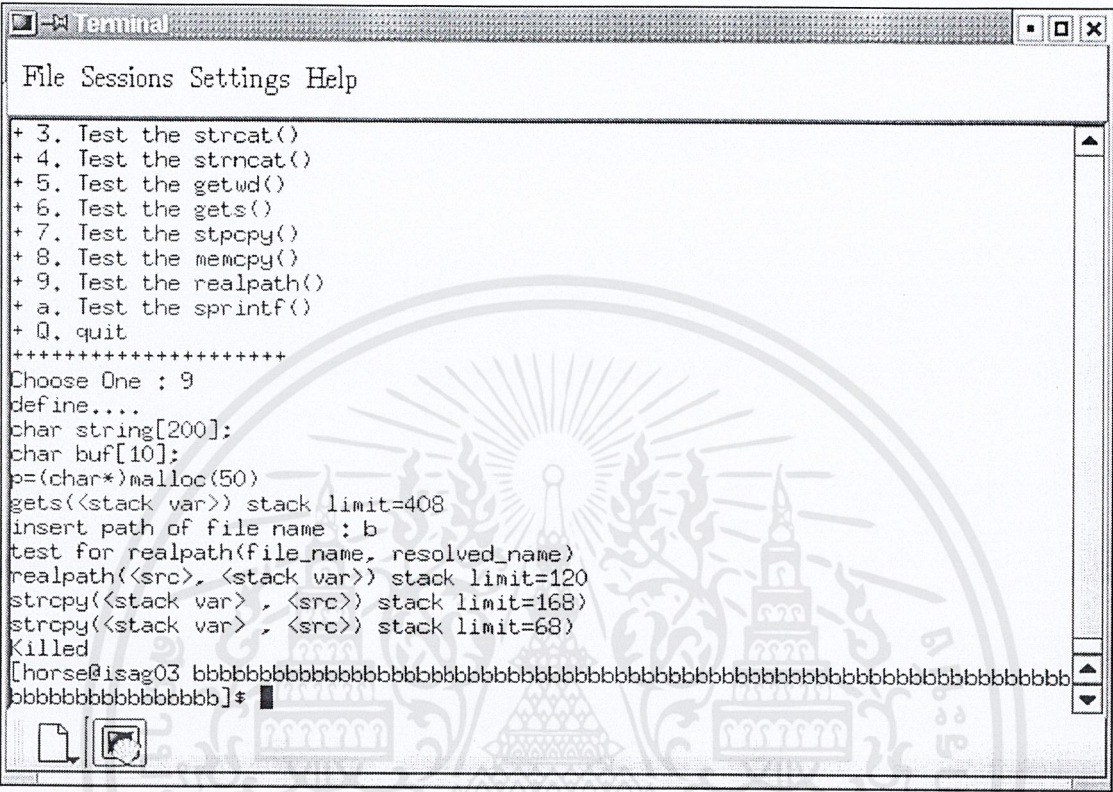
```

รูปที่ 6.8 แสดงการทดสอบฟังก์ชัน `gets()`

ในการทดสอบฟังก์ชัน `gets()` นั้น เราจะให้สามารถรับค่าได้สูงสุดที่เป็นไปได้ในสแต็กเท่านั้น จากรูป 6.8 จะเห็นได้ว่าเราทำการป้อนค่าให้กับตัวแปรที่มารับค่าเป็นจำนวนประมาณ 150 ไบต์ แต่ค่าที่สามารถรับได้สูงสุดเพียง 120 ไบต์ เท่านั้นค่าที่ถูกเก็บไว้ในตัวแปร `buf` จึงมีความยาวเพียง 120 ไบต์ ดังรูป

สำหรับในฟังก์ชัน `stpcpy()`, `memcpy()` จะมีลักษณะทำงานเหมือนกันกับฟังก์ชัน `strcpy()` จึงไม่แสดงมา ณ ที่นี้

การทดสอบฟังก์ชัน `realpath()` นั้นจะทดสอบโดยตรวจสอบชื่อไฟล์ที่ได้รับเข้ามาและนำ `path` ที่ได้ใส่ลงในตัวแปร `buf` ดังในรูปที่ 6.9 มี ไฟล์ `b` อยู่ใน `path` ที่ยาวมากๆ โลบารี่สามารถตรวจสอบได้ว่าเกิดการโอเวอร์โฟลว์ที่ตัวแปร `buf`



รูปที่ 6.9 แสดงการทดสอบฟังก์ชัน realpath()

6.2 ทดสอบการทำงานกับการโจมตีแบบบัฟเฟอร์โอเวอร์โฟลล์

การทดสอบโจมตีกับระบบปฏิบัติการลินุกซ์ redhat7.2 จากรูป 6.10 เป็นการเริ่มทดสอบว่าบนระบบปฏิบัติการนี้มีช่องโหว่ของโปรแกรมอยู่หรือไม่โดยการทดสอบนี้จะใช้ช่องโหว่ที่มีจากโปรแกรม efstool

efstool เป็นแอปพลิเคชันบน GNOME ซึ่งแอปพลิเคชันนี้โดยปกติตอนลงเครื่องใหม่ค่าเริ่มต้นของสิทธิ์ของไฟล์ efstool นี้จะมี s บิตด้วย ซึ่งทำให้การโจมตีประสบความสำเร็จ โดยโค้ดโปรแกรมซึ่งนำมาทดลองนี้จะทำการป้อนค่าอินพุตที่มากเกินไปที่บัฟเฟอร์ภายในโปรแกรมของ efstool จะรับได้ทำให้เกิดการโอเวอร์โฟลล์ขึ้น และได้สิทธิ์ของผู้ดูแลระบบไป โดยการรันนั้นจะใช้การรันโปรแกรมแบบ brute force offset โดยรันไฟล์โดยใช้คำสั่ง

```
S:/efs -l 2 800 1100
```



```

Konsole - qwe@isag04:~/exred - Konsole
File Sessions Settings Help

[qwe@isag04 exred]$ ls -la /usr/bin/efstool
-rwsr-sr-x  1 root    root      11132 Aug 21  2001 /usr/bin/efstool
[qwe@isag04 exred]$ gcc efs.c -o efs
[qwe@isag04 exred]$ ./efs
my first exploit - efstool - j0ker@daforest.org
usage: ./efs <offset> <os>
os types are: 1. linux x86 *untested*
              2. experimental x86
              3. bind shell to port 30464 x86 *untested*
use -1 offset for bruteforce
usage: ./efs -1 <os> <start offset> <stop offset>
!!! error: no offset specified !!!
[qwe@isag04 exred]$

```

รูปที่ 6.10 การสำรวจไฟล์ /usr/bin/efstool และการรันโปรแกรมโจมตี

การ brute force ช่วง offset นี้จะทำในช่วง 800 – 1000 (จากการสำรวจค่าพารามิเตอร์ต่างๆ ในอินเทอร์เน็ต และในโค้ด) รูป 6.11 แสดงการทำงานของโปรแกรมโจมตีเมื่อถูกเรียก

ในการรันโปรแกรมโจมตินั้นในแบบการ brute force offset จะได้เซตใหม่ออกมาหลายเซต ซึ่งถ้าค่าของออฟเซตนั้นยังไม่ตรงกับช่วงเวลาโปรแกรมที่มีสิทธิเป็นผู้ดูแลระบบ เซตที่ได้มาก็ยังไม่มีสิทธิเป็นผู้ดูแลระบบ เราต้องออกจากเซตนั้นแล้วให้โปรแกรมทำการ brute force ต่อไป (ถ้าโปรแกรมหยุดทำงาน ให้กด Ctrl+c เพื่อให้โปรแกรมทำการ brute force ต่อไป)

เมื่อทำการ brute force ไปสักระยะหนึ่งเราก็จะได้เซตของผู้ดูแลระบบมาดังรูปที่ 6.12 จะเห็นว่าค่าออฟเซตของโปรแกรมจะมีค่าในช่วงหนึ่งที่เมื่อใช้ค่าออฟเซตนั้นๆแล้วได้สิทธิเป็นผู้ดูแลระบบ ดังรูปค่าออฟเซตที่ได้มีค่าเท่ากับ 44F ในฐานสิบหกซึ่งเท่ากับค่า 1103 ในฐานสิบ


```

Konsole - qwe@isag04:~/exred - Konsole
File Sessions Settings Help

my first exploit - efstool - j0ker@daforest.org
stack pointer .. 0xbffff998
offset ..... 0x401
return addr .... 0xbffff597
sh-2.05# exit

```

รูปที่ 6.11 การทำงานของโปรแกรมที่ใช้โจมตี

```

Konsole - qwe@isag04:~/exred - Konsole
File Sessions Settings Help

my first exploit - efstool - j0ker@daforest.org
stack pointer .. 0xbffff998
offset ..... 0x44f
return addr .... 0xbffff549
sh-2.05# id
uid=0(root) gid=501(qwe) groups=501(qwe)
sh-2.05#

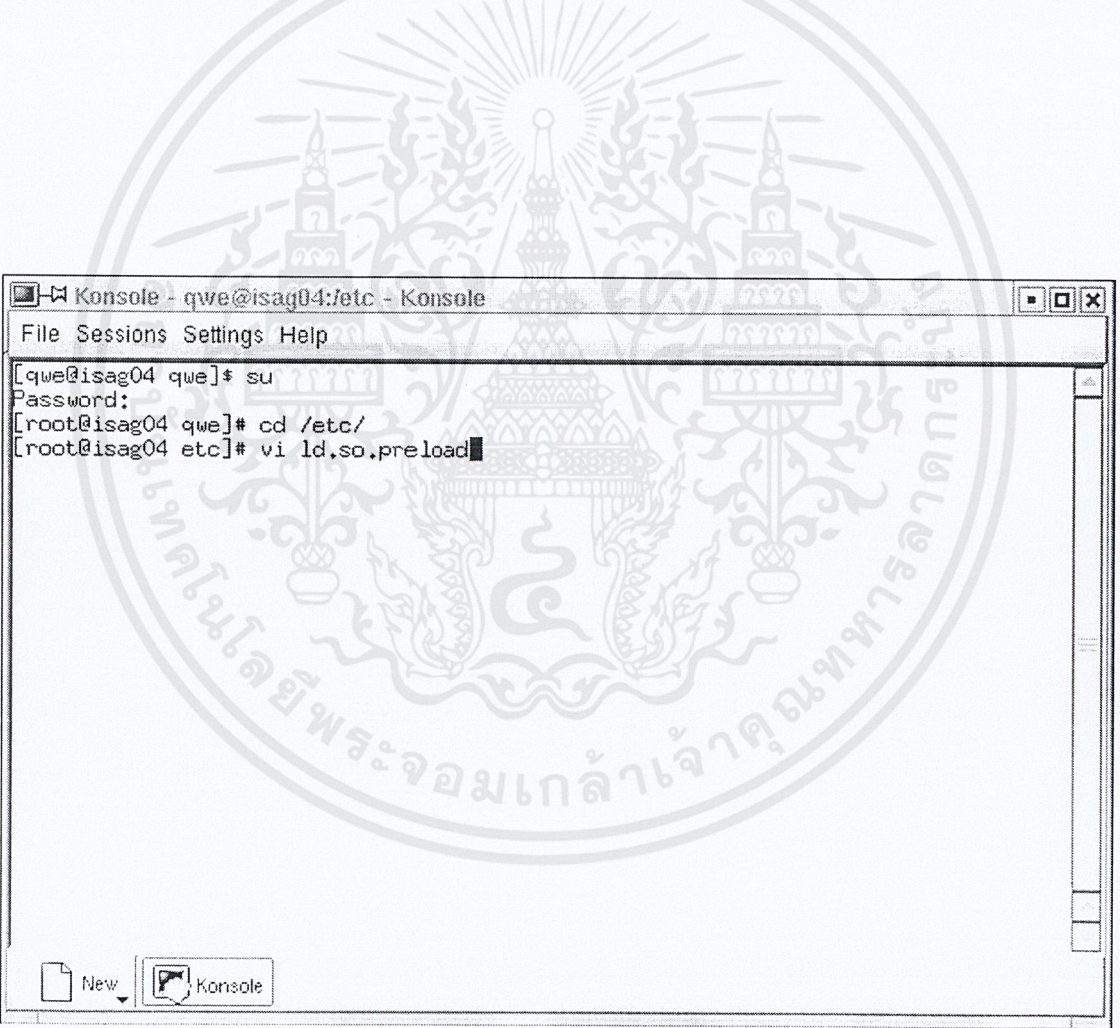
```

รูปที่ 6.12 การได้มาซึ่งเชลของผูู้ดูแลระบบ

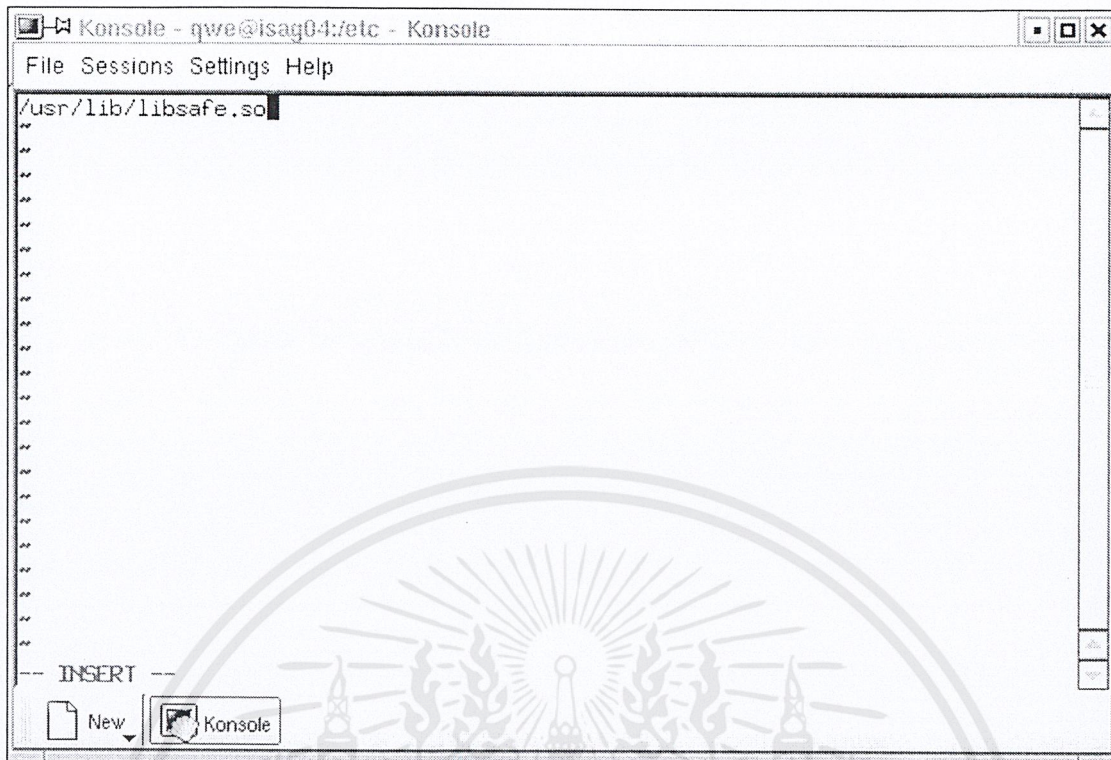
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การโหลดไลบรารีทำได้โดยไปเขียนในไฟล์ /etc/ld.so.preload (ซึ่งได้อธิบายไว้ในบทที่ 5) ดังรูปที่ 6.13 แล้วทำการโหลดไลบรารีของโปรแกรมเข้าไปในระบบ ดังในรูป 6.14 จากนั้นทำการโจมตีอีกครั้งหนึ่ง โดยเปิดอีกเชลหนึ่งไว้และเปิดให้รองรับเมสเสจคิวและให้แสดงผลออกทางหน้าจอด้วย(เขียนเพิ่มในโค้ด)

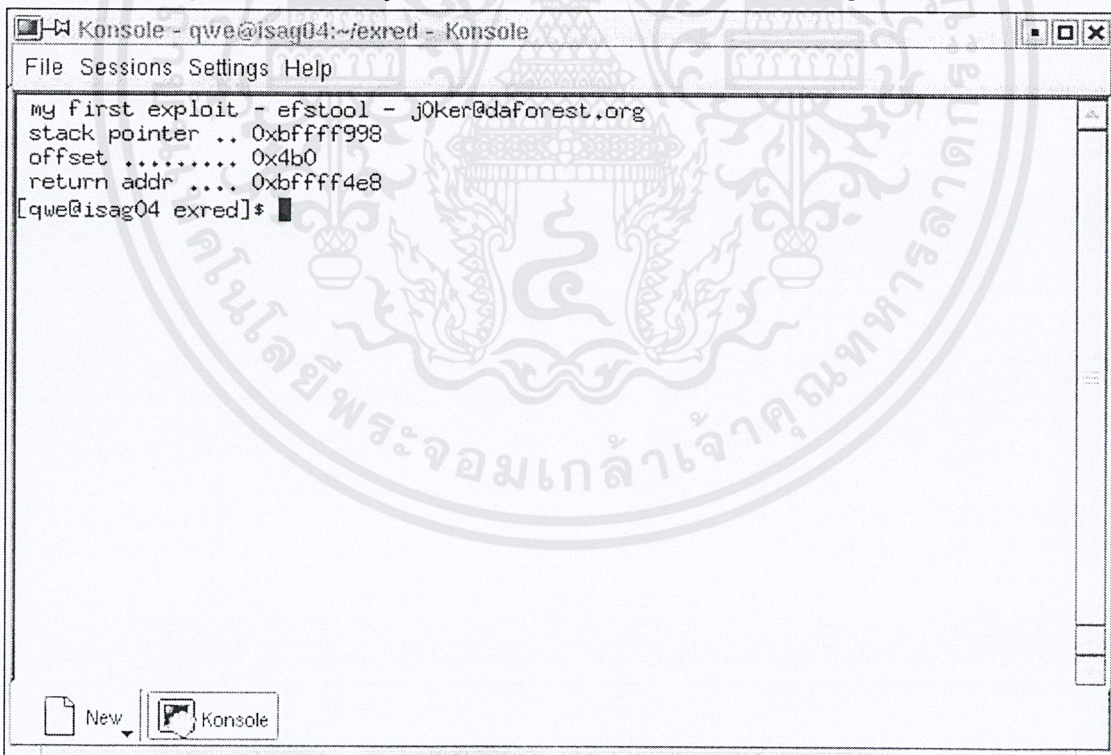
หลังจากทำการโจมตีอีกครั้งหนึ่งแล้วจาก รูปที่ 6.15 จะเห็นได้ว่าจะไม่มีการได้เชลใหม่ขึ้นมาเลย แม้แต่เชลเดียว แสดงให้เห็นว่าค่ารีเทิร์นแอดเดรสในสแตกทำงานได้อย่างปกติ ไม่เกิดการโอเวอร์โฟลว์ขึ้น (ค่าออฟเซตที่เห็นในรูปมีค่าเท่ากับ 1200 ในฐานสิบ ซึ่งเท่ากับค่าสุดท้ายที่ทำการ burte force ระบบ) ในขณะที่อีกเชลหนึ่งที่เปิดรับเมสเสจคิวไว้รอจะเห็นว่าได้รับค่าที่ส่งมาจากไลบรารีของโปรแกรมแสดงในรูป 6.16 และแจ้งการเกิดโอเวอร์โฟลว์ลงในล็อกไฟล์ของโปรแกรมซึ่งอยู่ใน /var/log/isaglib.log จากรูป 6.17 แสดงการเปิดล็อกไฟล์ซึ่งรายงานผลการโอเวอร์โฟลว์



รูปที่ 6.13 การเปิดไฟล์ /etc/ld.so.preload



รูปที่ 6.14 เขียนที่อยู่ของไลบรารีของโปรแกรมลงในไฟล์ ld.so.preload



รูปที่ 6.15 แสดงการไม่ได้เซตใหม่ในการโจมตีระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้


```

Konsole - qwe@isag04:/sbin - Konsole
File Sessions Settings Help

uid = 501
gid = 501
format = Overflow caused by strcpy()
date&time = Mon Mar 24 00:45:05 2003

application name = "/usr/bin/efstool"
uid = 501
gid = 501
format = Overflow caused by strcpy()
date&time = Mon Mar 24 00:45:05 2003

application name = "/usr/bin/efstool"
uid = 501
gid = 501
format = Overflow caused by strcpy()
date&time = Mon Mar 24 00:45:05 2003

application name = "/usr/bin/efstool"
uid = 501
gid = 501
format = Overflow caused by strcpy()
date&time = Mon Mar 24 00:45:05 2003

```

รูปที่ 6.16 แสดงการรับแมสเสจคิวจากไลบรารีของโปรแกรม

```

Konsole - qwe@isag04:/sbin - Konsole
File Sessions Settings Help

-----
Warning      : Overflow caused by strcpy()
UID          : 501
GID          : 501
Application  : /usr/bin/efstool
Time         : Mon Mar 24 00:45:05 2003

-----
Warning      : Overflow caused by strcpy()
UID          : 501
GID          : 501
Application  : /usr/bin/efstool
Time         : Mon Mar 24 00:45:05 2003

-----
Warning      : Overflow caused by strcpy()
UID          : 501
GID          : 501
Application  : /usr/bin/efstool
Time         : Mon Mar 24 00:45:05 2003

-----
[root@isag04 sbin]#

```

รูปที่ 6.17 แสดงข้อมูลที่อยู่ในล็อกไฟล์ของระบบ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากการทดสอบโปรแกรมกับการโจมตีกับระบบจริงๆ ซึ่งการโจมตีนี้ได้ทำการทดสอบกับลินุกซ์ในหลายๆเวอร์ชันซึ่งในบางเวอร์ชันนั้นไฟล์ `efstool` ไม่ได้มีค่า `s` บิต แต่ทว่าหากเซตให้ `efstool` มีค่า `s` bit แล้ว การโจมตีด้วยโปรแกรมที่ทำการทดสอบปรากฏว่าใช้ได้กับระบบปฏิบัติการ `redhat 7.2` `suse 8.0`

การทำงานของโปรแกรมสามารถดักการเกิดบัฟเฟอร์โอเวอร์โฟลว์ได้ ซึ่งตรงตามจุดประสงค์ในการพัฒนาโปรแกรมนี้นี้ขึ้นมา

6.3 ทดสอบประสิทธิภาพของระบบ

โปรแกรมจะทำงานโดยเรียกใช้ฟังก์ชันที่เราได้ทำการโอเวอร์ไรด์ลงไปเมื่อแอปพลิเคชันนั้นๆ ได้เรียกใช้ฟังก์ชัน แล้วทำการประมวลผลของสแตกต่างๆ ครั้งที่เรียกใช้ ทำให้ประสิทธิภาพในการประมวลผลของระบบลดลงไป เราได้ทำการทดสอบประสิทธิภาพที่ลดลงโดยการเรียกใช้คำสั่งบางคำสั่งแล้วทำการจับเวลา จากนั้นนำเวลาที่ได้นามานวนหาประสิทธิภาพของระบบ

การทดสอบนั้นจะทำการทดสอบกับคำสั่ง 2 คำสั่งคือ `ls -la` และ `perl -e 'print "A"x100'` โดยเขียนโปรแกรมขึ้นมาทดสอบโดยโปรแกรมที่นำมาทดสอบมีดังนี้

```
//performance.c
#include <stdio.h>
#include <stdlib.h>
int main(int argc, char *argv[])
{
    int i;
    if(argc<2) printf("use : %s <command> <loop>\n", argv[0]);
    printf("test for command : ", argv[0]);
    for(i=1; i<atoi(argv[2]); i++)
    {
        system(argv[1]);
    }
}
```

การทดสอบระบบกับคำสั่ง `ls -la` และ `perl -e 'print "A"x100'` ทำโดยเรียกใช้คำสั่งต่อไปนี้

```
-S time ./performance "/bin/ls -la" 5000 //แสดงการเรียกใช้คำสั่ง ls -la 5,000 ครั้ง
-S time ./performance "perl -e 'print "A"x100'" 500 //แสดงการเรียกใช้คำสั่ง print 500 ครั้ง
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ls -la 5000 ครั้ง <โหลดโปรแกรม>	ls -la 5000 ครั้ง <ปกติ>	ประสิทธิภาพ (%)
0m48.207s	0m44.861s	93.059
0m48.391s	0m45.819s	94.685
0m48.231s	0m44.423s	92.105
0m48.166s	0m46.340s	96.209
0m48.191s	0m46.345s	96.169

ตารางที่ 6.1 แสดงเวลาการทำงานของคำสั่ง ls -la 5000 ครั้ง

perl -e 'print "A"x100' 500 ครั้ง <โหลดโปรแกรม>	perl -e 'print "A"x100' 500 ครั้ง <ปกติ>	ประสิทธิภาพ (%)
0m2.797s	0m2.653s	94.852
0m2.802s	0m2.659s	94.897
0m2.797s	0m2.662s	95.173
0m2.799s	0m2.630s	93.962
0m2.661s	0m2.662s	95.412

ตารางที่ 6.2 แสดงเวลาการทำงานของคำสั่ง perl -e 'print "A"x100' 500 ครั้ง

จากตารางแสดงเวลาการทำงานของคำสั่งที่ใช้ทดสอบประสิทธิภาพของระบบทำให้ทราบว่า ประสิทธิภาพการทำงานของระบบลดลงไปบ้างดังค่าที่แสดงในตาราง สาเหตุที่ทำให้ประสิทธิภาพของระบบลดลงมาเนื่องมาจากก่อนที่จะมีการเรียกใช้งานฟังก์ชันในไลบรารีปกติจะต้องมีการเรียกใช้ฟังก์ชันที่ได้โอเวอร์โหลดไว้ เพื่อที่จะตรวจสอบขนาดของสแตกก่อน

6.4 วิเคราะห์การทำงานของระบบ

จากที่ได้ทำการทดสอบตัวโปรแกรมที่เขียนขึ้นมา จะเห็นได้ว่าในตัวโปรแกรมที่เขียนขึ้นมานั้นทำงานได้อย่างถูกต้อง ในการโจมตีระบบนั้นการโอเวอร์โฟลว์ในแบบต่างๆ อาจเกิดขึ้นโดยไม่ผ่านทางฟังก์ชันในสแตนด์ออลไลบรารีก็ได้ นอกจากนั้นการโจมตีด้วยบัฟเฟอร์โอเวอร์โฟลว์จะต้องอาศัยบั๊กของโปรแกรมต่างๆ ซึ่งสามารถที่จะมีช่วงที่เปลี่ยนไปเป็นผู้ดูแลระบบได้ อย่างไรก็ตามโปรแกรมที่ได้สร้างขึ้นมานี้ไม่ได้ครอบคลุมถึงการโจมตีในลักษณะ โอเวอร์โฟลว์ที่ไม่ได้เกิดจากฟังก์ชันที่ได้โอเวอร์โหลดไปในโปรแกรมที่สร้างขึ้น

6.5 สรุป

โปรแกรมระบบตรวจจับผู้บุกรุกที่ได้พัฒนาจะสามารถตรวจจับการโจมตีจากการเรียกใช้ฟังก์ชันที่ไม่ปลอดภัยในสแตนด์ออลไลบรารีที่ทำให้เกิดการโอเวอร์โฟลว์ ซึ่งถ้าผู้บุกรุกใช้วิธีการโจมตีดังกล่าวก็จะสามารถตรวจจับได้ แต่ทว่าโปรแกรมก็ยังสามารถที่จะตรวจจับการเกิดโอเวอร์โฟลว์แบบที่ไม่ได้ตั้งใจบุกรุกได้ด้วย โดยที่ผู้ดูแลระบบจะต้องพิจารณาอีกทีหนึ่งว่าผู้ใช้คนไหนมีเจตนาในการบุกรุก

ในส่วนขอประสิทธิภาพของระบบที่ติดตั้งโปรแกรมลงไปจะลดลงไปบ้าง เนื่องจากจะต้องมีการตรวจสอบขอบเขตของสแตกก่อนมีการเรียกใช้ฟังก์ชันในสแตนด์ออลไลบรารีตามปกติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

6.6 แนวทางในการพัฒนาต่อ

- พัฒนาให้ระบบสามารถตรวจจับฟังก์ชันที่ไม่ปกติได้มากขึ้น
- พัฒนาให้ระบบสามารถแยกแยะระหว่างการเกิดโอเวอร์โพล์ที่ เป็นการเจตนาถูกรบกวน ระบบ กับการเกิดโอเวอร์โพล์โดยไม่ได้ตั้งใจถูกรบกวน
- พัฒนาให้ระบบสามารถตรวจจับการโจมตีในรูปแบบอื่น
- พัฒนาให้ระบบสามารถตรวจจับได้แบบเรียลไทม์ (Real Time) โดยสามารถแจ้งเตือนต่อผู้ดูแลระบบได้ทันทีเมื่อมีการถูกรบกวน



บรรณานุกรม

หนังสืออ้างอิง

- [1] (ผู้เรียบเรียง) ดร.วิทยา วัชรวิทยากุล : “ภาษาและการโปรแกรม C ”, บริษัท ซีเอ็ดดูเคชั่น จำกัด (มหาชน), 2541
- [2] (ผู้เรียบเรียง) เอกสิทธิ์ วิริยาริ : “Hacking Exposed”, บริษัท ซีเอ็ดดูเคชั่น จำกัด (มหาชน), 2544
- [3] (บรรณาธิการ)วสิน เพิ่มทรัพย์, สิดารัสมิ์ ชำรงสมบัติสกุล : “คู่มือติดตั้งและใช้งาน Linux”, โครงการลินุกซ์ภาษาไทย บริษัท ไกวัล ซอฟต์แวร์ จำกัด, 2542
- [4] W, Richard Stevens : “Avanced Programming in the UNIX Environment”, Pearson Education, 2002
- [5] สันติ ศรีลาศักดิ์, วรวุฒิ เทียงธรรม : “เจาะประเด็นงานเขียนโปรแกรมบนลินุกซ์”, ออฟเซทเพรส, 2542
- [6] WILLIAM STALLINGS : “Opearating Systems”, Prentice Hall, 2001
- [7] Maurice J. Bach : “The Design of the Unix Operating System”, Bell Telephone Laboratories, 1986

เว็บไซต์อ้างอิง

- [1] <http://www.cs.cf.ac.uk/Dave/C/CE.html>
- [2] <http://www.infosys.utas.edu.au/info/documentation/C/CStdLib.html>
- [3] <http://www.linuxgazette.com/issue24/rogers.html>
- [4] http://www.cs.utah.edu/dept/old/texinfo/glibc-manual-0.02/library_toc.html
- [5] <http://www.bell-labs.com/org/11356/libsafe.html>
- [6] <http://ouah.sysdoor.net/bofstdd.pdf>
- [7] <http://www.cnhonker.net>
- [8] <http://www.hack.co.za>
- [9] <http://www.packetstromsecurity.org>
- [11] <http://www.securiteam.com/>
- [10] <http://community.core-sdi.com/~juliano/P56-14-adjacent-mem.txt>
- [11] <http://thaicert.nectec.or.th/>
- [12] <http://www.linuxeasy.net>

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้าไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหาและต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้