การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล

นางสาวสิริสรา เจียมวงศ์แพทย์

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรมหาบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2552

THE IMPLEMENTATION OF SECURE CANARY WORD FOR BUFFER-OVERFLOW PROTECTION

Ms. Sirisara Chiamwongpaet

A Thesis Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Engineering Program in Computer Engineering

Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University

Academic Year 2009

หัวข้อวิทยานิพนธ์	การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล
โดย	นางสาวสิริสรา เจียมวงศ์แพทย์
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก	อาจารย์ ดร. เกริก ภิรมย์โสภา

คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย อนุมัติให้นับวิทยานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญามหาบัณฑิต

..... คณบดีคณะวิศวกรรมศาสตร์
(รองศาสตราจารย์ ดร.บุญสม เลิศหิรัญวงศ์)

คณะกรรมการสอบวิทยานิพนธ์

..... ประธานกรรมการ
(อาจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์)

..... อาจารย์ที่ปรึกษาวิทยานิพนธ์หลัก
(อาจารย์ ดร. เกริก ภิรมย์โสภา)

..... กรรมการ
(ผู้ช่วยศาสตราจารย์ ดร.เฉลิมเอก อินทนากรวิวัฒน์)

..... กรรมการภายนอกมหาวิทยาลัย
(ดร.พงศ์ธวัช ชีพพิมลชัย)

สิริสรา เจียมวงศ์แพทย์ : การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล. (THE IMPLEMENTATION OF SECURE CANARY WORD FOR BUFFER-OVERFLOW PROTECTION) อ.ที่ปรึกษาวิทยานิพนธ์หลัก : อ.ดร.เกริก ภิรมย์โสภา, 116 หน้า.

การป้องกันบัฟเฟอร์โอเวอร์โฟลว์เป็นที่สนใจอย่างกว้างขวางในด้านความมั่นคงของคอมพิวเตอร์ จุดประสงค์ของการวิจัยนี้คือ นำเสนอวิธีการนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผลวิธีใหม่ และประเมินผลในสองด้าน ได้แก่ ด้านการป้องกัน และด้านประสิทธิภาพ ซีเคียวคานารีเวิร์ดเป็นการแก้ปัญหาด้วยสถาปัตยกรรมที่มีพื้นฐานมาจากสองวิธีการที่มีอยู่แล้ว ได้แก่ ซีเคียวบิต และคานารีเวิร์ด เพื่อป้องกันการโจมตีด้วยบัฟเฟอร์โอเวอร์โฟลว์ในข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ เช่น ตัวแปร และอาร์กิวเมนต์ เป็นต้น การทำให้เกิดผลนี้จัดทำบนซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์ (BOCHS Emulator) ที่ดัดแปรแล้ว ซึ่งรันระบบปฏิบัติการลีนุกซ์ (เรดเฮด 6.2 เคอร์เนล 2.2.14) เมื่อประเมินผลแล้ว ผลลัพธ์แสดงให้เห็นว่า ซีเคียวคานารีเวิร์ดสามารถตรวจหาการโจมตีด้วยบัฟเฟอร์โอเวอร์โฟลว์ในข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบได้และยังครอบคลุมมากกว่าซีเคียวบิต นอกจากนี้ ประสิทธิภาพดีกว่างานเดิมอย่างเห็นได้ชัด การนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผลด้วยวิธีการใหม่นี้ชี้ให้เห็นว่า ระบบมีความปลอดภัยมากขึ้น

ภาควิชา : .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่อนิสิต : สิริสรา เจียมวงศ์แพทย์
สาขาวิชา : .....วิศวกรรมคอมพิวเตอร์..... ลายมือชื่ออ.ที่ปรึกษาวิทยานิพนธ์หลัก : .....
ปีการศึกษา : ..2552.....

5170499521 : MAJOR COMPUTER ENGINEERING

KEYWORDS: SYSTEM ARCHITECTURES / SECURITY AND PROTECTION

SIRISARA CHIAMWONGPAET : THE IMPLEMENTATION OF SECURE CANARY WORD FOR BUFFER-OVERFLOW PROTECTION. THESIS ADVISOR : KRERK PIROMSOPA, Ph.D., 116 pp.

The possibility of buffer-overflow protection has generated wide interest in Computer Security. The purposes of this study are to propose a new method for implementing Secure Canary Word and to evaluate it in two aspects: protection and efficiency. Secure Canary Word is an architectursal approach based on two existing schemes, Secure Bit and Canary Word, for protecting against buffer-overflow attacks on non-control data (variables and arguments). The implementation was conducted in the modified BOCHS emulators running Linux (Red Hat 6.2 Kernel 2.2.14). When it is evaluated, the results showed that Secure Canary Word is not only able to detect buffer-overflow attacks on non-control data but it can also provide more coverage than that of Secure Bit. Furthermore, its efficiency is clearly better than that of the previous work. Our implementation suggests that system can be more secure with the new approach.

Department : ..Computer Engineering Student's signature : Sirisara Chiamwongpaet

Field of study : Computer Engineering Advisor's signature : .....

Academic year : ...2009.....

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงไปได้ด้วยความอนุเคราะห์อย่างยิ่งของอาจารย์ ดร. เกริก ภิรมย์โสภา อาจารย์ที่ปรึกษา ซึ่งท่านได้ให้ความรู้ แนะนำแนวทางการวิจัย ตรวจสอบให้คำแนะนำ และสนับสนุนเป็นอย่างดี จนทำให้การวิจัยในครั้งนี้สำเร็จออกมาด้วยดี

ขอขอบพระคุณ อาจารย์ ดร. ณัฐวุฒิ หนูไพโรจน์ อาจารย์ ดร. พงศ์ธวัช ชีพพิมลชัย และผู้ช่วยศาสตราจารย์ ดร. เฉลิมเอก อินทนากรวิวัฒน์ กรรมการสอบวิทยานิพนธ์ ที่กรุณาเสียสละเวลา ให้คำแนะนำ ตรวจสอบ และแก้ไขวิทยานิพนธ์ฉบับนี้

ขอขอบพระคุณ บัณฑิตวิทยาลัย ที่ได้มอบทุนอุดหนุนการศึกษาในระดับบัณฑิตศึกษา จุฬาลงกรณ์มหาวิทยาลัย เพื่อเฉลิมฉลองวโรกาสที่พระบาทสมเด็จพระเจ้าอยู่หัวทรงเจริญพระชนมายุครบ 72 พรรษาให้

ท้ายที่สุด ผู้เสนอวิทยานิพนธ์ขอขอบคุณเพื่อน ๆ ทุก ๆ คน รวมทั้งครอบครัว ที่คอยติดตาม ให้กำลังใจและสนับสนุน รวมถึงท่านอื่น ๆ ที่มิได้กล่าวชื่อไว้ ณ ที่นี้ที่มีส่วนช่วยให้วิทยานิพนธ์สำเร็จได้ด้วยดี

สารบัญ

หน้า

บทคัดย่อภาษาไทย ..... ง
บทคัดย่อภาษาอังกฤษ ..... จ
กิตติกรรมประกาศ ..... ฉ
สารบัญ ..... ช
สารบัญตาราง ..... ฌ
สารบัญภาพ ..... ญ
บทที่ 1 บทนำ 1
1.1 ความเป็นมาและความสำคัญของปัญหา 1
1.2 วัตถุประสงค์ของการวิจัย 2
1.3 ขอบเขตของการวิจัย 2
1.4 คำจำกัดความที่ใช้ในการวิจัย 3
1.5 ประโยชน์ที่คาดว่าจะได้รับ 3
1.6 วิธีดำเนินการวิจัย 4
1.7 ลำดับขั้นตอนในการเสนอผลการวิจัย 4
1.8 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์ 4
บทที่ 2 ทฤษฎีและงานวิจัยที่เกี่ยวข้อง 6
2.1 แนวคิดและทฤษฎี 6
2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง 18
บทที่ 3 หลักการของซีเคียวคานารีเวิร์ด (Secure Canary Word) 25
3.1 แนวคิดที่เกี่ยวข้องกับซีเคียวคานารีเวิร์ด 25
3.2 หลักการของซีเคียวคานารีเวิร์ด (Secure Canary Word) 27
บทที่ 4 แนวทางการนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผล 29
4.1 เครื่องมือที่ใช้ในการวิจัย 29
4.2 การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล (The Implementation of Secure Canary Word) 30
บทที่ 5 วิธีประเมินการวิจัย สรุปและอภิปรายผล 38
5.1 วิธีประเมินการวิจัย 38
5.2 สรุปผลการวิจัย 40

5.3 อภิปรายผลการวิจัย 43
5.4 วิเคราะห์ผลการวิจัยเพิ่มเติม 45
บทที่ 6 บทสรุป 48
6.1 สิ่งที่ได้จากการวิจัย (Contribution) 48
6.2 ประโยชน์ของซีเคียวคานารีเวิร์ด 48
6.3 แนวทางการวิจัยต่อ 48
6.4 บทสรุป 51
รายการอ้างอิง 52
ภาคผนวก 59
ภาคผนวก ก. การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลในโครงงานทางวิศวกรรม (Senior Project) ในระดับปริญญาตรี 60
ภาคผนวก ข. โค้ดของโปรแกรมที่ใช้ทดลองความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆ 64
ภาคผนวก ค. โค้ดของโปรแกรมที่ใช้ทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ 68
ภาคผนวก ง. ข้อมูลที่ใช้ทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ 112
ภาคผนวก จ. ขั้นตอนการสร้าง (Make) 114
ประวัติผู้เขียนวิทยานิพนธ์ 116

สารบัญตาราง

หน้า

ตารางที่ 1 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่จองไว้ 7
ตารางที่ 2 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy() 7
ตารางที่ 3 ผังสแต็ก (Stack Layout) ของหน่วยความจำก่อนการเรียกใช้ฟังก์ชัน strcpy() 10
ตารางที่ 4 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy() 10
ตารางที่ 5 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy() 12
ตารางที่ 6 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy() 13
ตารางที่ 7 ผังสแต็ก (Stack Layout) 16
ตารางที่ 8 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันด้วยคานารีเวิร์ด กรณีที่ไม่สามารถตรวจพบโอเวอร์โฟลว์ได้ 26
ตารางที่ 9 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวบิต 26
ตารางที่ 10 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวบิต กรณีที่ไม่สามารถตรวจพบโอเวอร์โฟลว์ได้ 27
ตารางที่ 11 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวคานารีเวิร์ด 28
ตารางที่ 12 ตารางสรุปรหัสคำสั่งเดิมและรหัสคำสั่งใหม่ 32
ตารางที่ 13 ผลการทดสอบความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆ 40
ตารางที่ 14 ประสิทธิภาพของโปรแกรมในด้านความเร็วของการทำงาน 41
ตารางที่ 15 ประสิทธิภาพของโปรแกรมในด้านขนาดของหน่วยความจำที่ใช้ในขณะที่โปรแกรมทำงาน 41
ตารางที่ 16 ประสิทธิภาพของโปรแกรมในด้านขนาดของโปรแกรม 41
ตารางที่ 17 ขนาดของโปรแกรมเรียงลำดับแบบฟองโดยใช้ออปชันต่างกัน 42
ตารางที่ 18 ขนาดของโปรแกรมควิกซอร์ตโดยใช้ออปชันต่างกัน 42
ตารางที่ 19 ขนาดของโปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอลโดยใช้ออปชันต่างกัน 43
ตารางที่ 20 ตารางเปรียบเทียบก่อนและหลังดัดแปรคอมไพเลอร์ (Modify Compiler) 49

สารบัญภาพ

หน้า

รูปที่ 1 วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบต่างๆและวิธีที่เลือกใช้การวิจัยนี้ 2
รูปที่ 2 วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบต่างๆ 18
รูปที่ 3 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันด้วยคานารีเวิร์ด 25
รูปที่ 4 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวบิต 26
รูปที่ 5 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวคานารีเวิร์ด 28
รูปที่ 6 รูปแบบคำสั่ง (Instruction Format) ของสถาปัตยกรรมของอินเทล (Intel) 386 31
รูปที่ 7 แผนภาพวงจร (Circuit Diagram) ของโครงสร้างหน่วยประมวลผลกลางใหม่ 46
รูปที่ 8 แผนภาพบล็อก (Block Diagram) ของส่วนต่อประสาน (Interface) ระหว่างอุปกรณ์ฮาร์ดแวร์ทั้งหมด และตัวควบคุมซีเคียวบิต (Secure Bit Controller) 47
รูปที่ 9 แผนภาพบล็อก (Block Diagram) ของการแปลงแอดเดรส (Address Translation) 47

บทที่ 1
บทนำ

ในบทนำนี้จะแบ่งเป็นแปดหัวข้อย่อย กล่าวถึงความเป็นมาและความสำคัญของปัญหา วัตถุประสงค์ ขอบเขตของการวิจัย คำจำกัดความ ประโยชน์ที่คาดว่าจะได้รับ วิธีดำเนินการวิจัย ลำดับขั้นตอนในการเสนอผลการวิจัย และผลงานที่ตีพิมพ์จากวิทยานิพนธ์ ตามลำดับ ดังนี้

1.1 ความเป็นมาและความสำคัญของปัญหา

ทุกวันนี้ การโจมตีเครื่องคอมพิวเตอร์ต่างๆทั่วโลกได้ทวีความรุนแรงมากยิ่งขึ้น โดยเฉพาะหนอนและไวรัสคอมพิวเตอร์ที่มีจำนวนเพิ่มขึ้นมาเรื่อยๆ จนก่อให้เกิดความเสียหายอย่างร้ายแรงแก่ระบบเป็นมูลค่ามหาศาล ในบรรดาหนอนและไวรัสคอมพิวเตอร์ที่พบทั่วไป ส่วนใหญ่จะใช้เทคนิคการโจมตีด้วยบัฟเฟอร์โอเวอร์โฟลว์ (Buffer-overflow Attacks) ซึ่งเป็นช่องโหว่ (Vulnerability) ของระบบคอมพิวเตอร์ที่ถูกใช้เป็นช่องทางในการเข้าสู่ระบบเพื่อสร้างความเสียหายหรือขโมยข้อมูลต่อไป ตัวอย่างเช่น เมื่อ พ.ศ.2531 หนอนคอมพิวเตอร์ที่โด่งดังชื่อว่า MORRIS WORM [1] ก็เจาะเข้าสู่ระบบด้วยเทคนิคเช่นกัน แม้กระทั่งปัจจุบัน เมื่อวันที่ 7 กันยายน พ.ศ.2552 ก็ยังมีรายงานว่า พบช่องโหว่ที่อาจถูกโจมตีด้วยเทคนิคนี้ได้ (VU#336053 Cyrus IMAPd buffer overflow vulnerability) [2] นอกจากนี้ เมื่อวันที่ 11 สิงหาคม พ.ศ.2551 ก็ยังมีข่าวว่า ระบบปฏิบัติการวินโดวส์วิสตา (Windows Vista Operating System) พบช่องโหว่ที่อาจถูกโจมตีด้วยเทคนิคนี้ได้เช่นกัน [3]

การโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์เป็นวิธีการโจมตีระบบโดยนำข้อมูลขนาดใหญ่กว่าขนาดของบัฟเฟอร์ (Buffer) ซึ่งเป็นหน่วยความจำที่จองไว้เพื่อรองรับข้อมูลนั้นเข้ามาเก็บลงในบัฟเฟอร์ ทำให้ข้อมูลบางส่วนที่ล้นออกมาเกินจะไปทับข้อมูลส่วนอื่นๆ ที่อาจเป็นส่วนที่เป็นตัวควบคุมระบบ (Control Data) หรือส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) ได้แก่ ตัวแปรภายในโปรแกรม เช่นเดียวกันกับบัฟเฟอร์ นอกจากวิธีดังกล่าวแล้ว ยังรวมถึงวิธีอื่นๆที่เขียนทับส่วนที่เกินขอบเขตของข้อมูลที่กำหนดไว้

ปัจจุบัน แม้จะมีผู้เสนอวิธีการป้องกันการโจมตีด้วยเทคนิคนี้จำนวนมาก แต่ส่วนใหญ่จะเน้นป้องกันไม่ให้ข้อมูลล้นไปทับหน่วยความจำที่เป็นตัวควบคุมระบบ มีส่วนน้อยที่คำนึงถึงการป้องกันข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ เมื่อข้อมูลส่วนที่เป็นตัวควบคุมระบบส่วนใหญ่ได้รับการป้องกัน เป้าหมายการโจมตีก็เปลี่ยนมาเป็นข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ ซึ่งปัจจุบันเป็นสาเหตุหลักของการโจมตีไม่น้อยไปกว่าข้อมูลส่วนที่เป็นตัวควบคุมระบบ การวิจัยนี้จึงถูกจัดทำขึ้น

เพื่อศึกษาวิธีการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์โดยวิธีซีเคียวคานารีเวิร์ด (Secure Canary Word) ซึ่งคาดว่าจะสามารถป้องกันหน่วยความจำข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบได้ และสามารถนำไปปรับปรุงวิธีการป้องกันนี้ให้ดียิ่งขึ้นต่อไป

หากวิธีการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์โดยวิธีซีเคียวคานารีเวิร์ดนี้สามารถป้องกันได้จริงดังที่คาดหวังไว้ จะช่วยให้ปิดช่องโหว่ของระบบได้อีกจำนวนมาก ทำให้ระบบมีความปลอดภัยจากการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์เพิ่มขึ้นอย่างมาก

1.2 วัตถุประสงค์ของการวิจัย

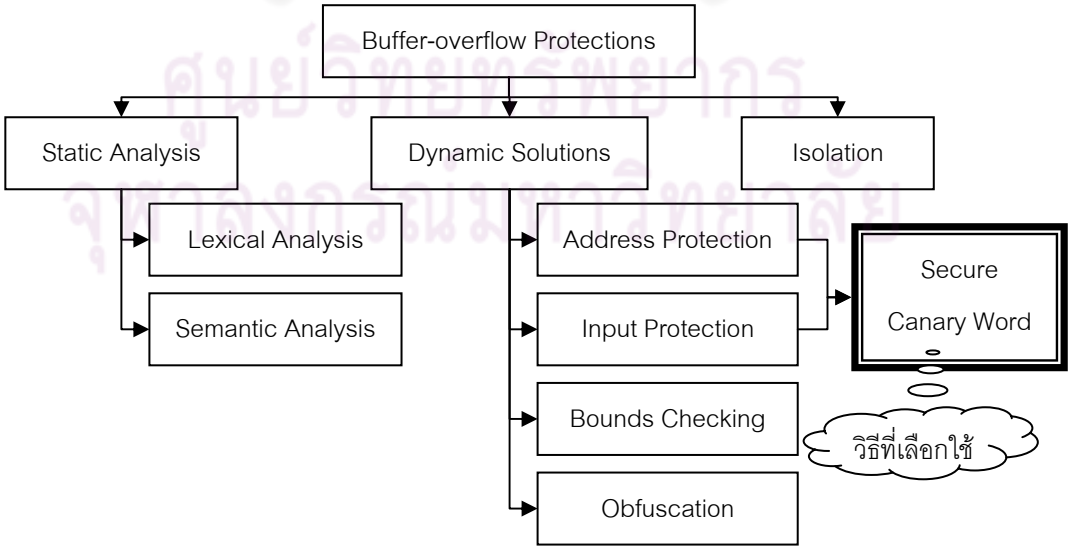
การวิจัยมีวัตถุประสงค์ ดังนี้

1. เพื่อศึกษาวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด
2. เพื่อนำวิธีการป้องกันดังกล่าวมาประยุกต์ใช้จริง โดยปรับปรุงจากโครงงานทางวิศวกรรม (Senior Project) ที่ทำไว้ในระดับปริญญาตรีให้มีประสิทธิภาพดียิ่งขึ้น
3. เพื่อทดลองหารูปแบบการโจมตีเครื่องคอมพิวเตอร์ด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆที่วิธีการป้องกันดังกล่าวสามารถป้องกันได้ และไม่สามารถป้องกันได้
4. เพื่อวัดและวิเคราะห์ประสิทธิภาพของวิธีการป้องกันดังกล่าวที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆหลังจากปรับปรุงวิธีทำให้เกิดผล (Implement) แล้ว

1.3 ขอบเขตของการวิจัย

ขอบเขตของการวิจัยถูกกำหนดไว้ ดังนี้

1. เลือกเฉพาะวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดเท่านั้นในการทดลอง ดังรูปที่ 1



รูปที่ 1 วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบต่างๆและวิธีที่เลือกใช้การวิจัยนี้

2. ศึกษาและทดลองการโจมตีเครื่องคอมพิวเตอร์ด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์แบบต่างๆ ที่ครอบคลุมเฉพาะการโจมตีบนเครื่องคอมพิวเตอร์ที่มีหน่วยประมวลผลกลาง (Central Processing Unit: CPU) เป็น Intel x86 บนระบบปฏิบัติการลีนุกซ์ (Linux Operating System) เรดเฮด 6.2 เคอร์เนล 2.2.14 (Red Hat 6.2 Kernel 2.2.14) เท่านั้น

1.4 คำจำกัดความที่ใช้ในการวิจัย

คำจำกัดความที่ใช้ในการวิจัย มีดังต่อไปนี้

1. บัฟเฟอร์ (Buffer) หมายถึง หน่วยความจำที่ถูกจองไว้ เพื่อรองรับข้อมูล
2. ข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data) หมายถึง ข้อมูลต่างๆที่ระบบสร้างขึ้นมาเอง เช่น รีเทิร์นแอดเดรส (Return Address) ฟังก์ชันพอยท์เตอร์ (Function Pointer) เป็นต้น
3. ข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) หมายถึง ข้อมูลต่างๆที่ระบบไม่ได้สร้างขึ้นมาเอง แต่อาจเกิดจากผู้เขียนโปรแกรมกำหนดขึ้นมา หรือผู้ใช้ป้อนเข้าไปเอง เช่น ตัวแปรภายในโปรแกรม (Local Variable) อาร์กิวเมนต์ (Argument) เป็นต้น
4. บัฟเฟอร์โอเวอร์โฟลว์ (Buffer Overflows) [4] หมายถึง เหตุการณ์ที่ข้อมูลที่มีขนาดใหญ่กว่าขนาดของบัฟเฟอร์ ถูกนำเข้ามาเก็บลงในบัฟเฟอร์นั้น ทำให้ข้อมูลบางส่วนที่ล้นออกมาเกินจะไปทับข้อมูลส่วนอื่นๆที่อยู่ข้างเคียง ซึ่งอาจเป็นส่วนที่เป็นตัวควบคุมระบบ หรือส่วนที่ไม่เป็นตัวควบคุมระบบ
5. การโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ (Buffer-overflow Attacks) หมายถึง การโจมตีระบบโดยอาศัยบัฟเฟอร์โอเวอร์โฟลว์ รวมถึงวิธีอื่นๆที่เขียนทับส่วนที่เกินขอบเขตของข้อมูลที่กำหนดไว้ ทำให้ระบบที่ถูกโจมตีทำงานผิดปกติ ในการวิจัยนี้การโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ ถูกแบ่งเป็นสามประเภท ได้แก่ 1) สแต็กโอเวอร์โฟลว์ (Stack Overflows) 2) ฮีปโอเวอร์โฟลว์ (Heap Overflows) 3) อาร์เรย์อินเด็กซิงเออเรอร์ (Array Indexing Error)

1.5 ประโยชน์ที่คาดว่าจะได้รับ

ประโยชน์ที่คาดว่าจะได้รับจากการวิจัย ได้แก่

1. เข้าใจวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด

2. ได้ความรู้เกี่ยวกับรูปแบบการโจมตีที่วิธีการป้องกันดังกล่าวสามารถป้องกันได้ และไม่สามารถป้องกันได้พร้อมเหตุผล
3. ได้ความรู้เกี่ยวกับประสิทธิภาพของวิธีการป้องกันดังกล่าวที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ หลังจากปรับปรุงวิธีทำให้เกิดผล (Implement) แล้ว
4. สามารถนำความรู้จากผลการวิจัยนี้ไปประยุกต์ใช้จริงต่อไปในอนาคต

1.6 วิธีดำเนินการวิจัย

วิธีดำเนินการวิจัย ถูกแบ่งเป็นห้าขั้นตอน ดังนี้

1. ศึกษางาน ถูกแบ่งเป็นสองขั้นตอนย่อย ได้แก่ ศึกษางานวิจัยที่เกี่ยวข้อง และ ศึกษาการใช้เครื่องมือที่ใช้ในงานวิจัย
2. นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล
3. ทดลองโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆ
4. ทดลองวัดประสิทธิภาพของวิธีการป้องกันดังกล่าวที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ
5. สรุปผลการวิจัยและจัดทำวิทยานิพนธ์

1.7 ลำดับขั้นตอนในการเสนอผลการวิจัย

วิทยานิพนธ์นี้แบ่งเนื้อหาออกเป็น 5 บท ดังต่อไปนี้ บทที่ 1 เป็นบทนำซึ่งกล่าวถึง ความเป็นมาและความสำคัญของปัญหา รวมถึงวัตถุประสงค์ของการวิจัย บทที่ 2 กล่าวถึงทฤษฎีพื้นฐาน และงานวิจัยที่เกี่ยวข้องกับการวิจัยนี้ บทที่ 3 กล่าวถึงหลักการของซีเคียวคานารีเวิร์ด บทที่ 4 กล่าวถึงแนวทางการนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผล และบทที่ 5 กล่าวถึงวิธีประเมินการวิจัย สรุปและอภิปรายผลการวิจัย

1.8 ผลงานที่ตีพิมพ์จากวิทยานิพนธ์

ส่วนหนึ่งของวิทยานิพนธ์นี้ได้รับการตอบรับให้ตีพิมพ์เป็นบทความทางวิชาการในหัวข้อเรื่อง “Secure Bit Enhanced Canary: Hardware Enhanced Buffer-Overflow Protection” [5] โดยนางสาวสิริสรา เจียมวงศ์แพทย์ และอาจารย์ ดร. เกริก ภิรมย์โสภา, ในงานประชุมวิชาการ “IFIP International Workshop on Network and System Security (NSS 2008)” ณ เมืองเซี่ยงไฮ้ ประเทศสาธารณรัฐประชาชนจีน วันที่ 18-19 ตุลาคม พ.ศ. 2551 และในหัวข้อเรื่อง “The Implementation of Secure Canary Word for Buffer-Overflow Protection.” [6] โดยนางสาวสิริสรา เจียมวงศ์แพทย์ และอาจารย์ ดร. เกริก ภิรมย์โสภา, ในงานประชุมวิชาการ “2009 Electro/Information Technology Conference, sponsored by the IEEE Region 4 (R4) (EIT

2009)" ณ เมืองวินด์เซอร์ มณฑลออนแทรีโอ ประเทศสมาพันธรัฐแคนาดา วันที่ 7-9 มิถุนายน พ.ศ. 2552

บทที่ 2
ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงแนวคิดและทฤษฎี รวมทั้งเอกสารและงานวิจัยที่เกี่ยวข้อง ดังนี้

2.1 แนวคิดและทฤษฎี

แนวคิดและทฤษฎีที่จะอธิบายในการวิจัยนี้ แบ่งเป็นสองส่วน ได้แก่

2.1.1 หลักการของการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์

จากการสำรวจพบว่า วิธีการโจมตีเครื่องคอมพิวเตอร์ในปัจจุบัน ส่วนใหญ่จะใช้เทคนิคบัฟเฟอร์โอเวอร์โฟลว์ (Buffer-overflow Attacks) ซึ่งมีขั้นตอนหลักสี่ขั้นตอน ดังนี้

1. จองหน่วยความจำโดยการประกาศตัวแปรชนิดอาร์เรย์ (Array) ซึ่งเรียกหน่วยความจำที่จองนี้ว่า “บัฟเฟอร์ (Buffer)” สำหรับรอรับข้อมูล
2. นำข้อมูลที่มีขนาดใหญ่กว่าขนาดของบัฟเฟอร์ที่จองไว้มาเก็บลงในบัฟเฟอร์ โดยเรียกใช้ฟังก์ชันสำหรับนำข้อมูลจากตัวแปรอาร์เรย์หนึ่งไปใส่ตัวแปรอาร์เรย์อีกตัว เช่น strcpy() ในภาษาซี
3. จากขั้นตอนที่สอง จะทำให้ข้อมูลบางส่วนที่ล้นออกมานอกบัฟเฟอร์ หรือเรียกอีกอย่างว่า “บัฟเฟอร์โอเวอร์โฟลว์ (Buffer Overflows)” ไปทับหน่วยความจำส่วนอื่นที่อยู่ติดกับบัฟเฟอร์ ซึ่งหน่วยความจำส่วนนี้อาจเป็นข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data) หรือส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) เช่น ตัวแปรภายในโปรแกรม เป็นต้น
4. ถ้าการเปลี่ยนแปลงค่าของหน่วยความจำส่วนนั้นมีผลทำให้เป็นอันตรายต่อเครื่องคอมพิวเตอร์ จะทำให้คอมพิวเตอร์เครื่องนั้นเกิดความเสียหายขึ้น

สมมติ ตัวอย่างโค้ดของโปรแกรมภาษาซีโปรแกรมหนึ่ง ดังนี้

```
#include <stdio.h>
#include <string.h>
int main(char *p) {
      char a[8];
      int b = 0;
      strcpy(a,p);
      printf(“%d”,b);
      return 0;
}
```

ตัวอย่างดังกล่าวสามารถถูกโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์อย่างง่าย ดังขั้นตอนต่อไปนี้

1. ประกาศตัวแปรชนิดสตริง (เป็นตัวแปรชนิดอาร์เรย์แบบหนึ่ง) ชื่อ a ขนาด 8 ไบต์ และตัวแปรชนิดจำนวนเต็มชื่อ b กำหนดค่าให้เท่ากับ 0 การประกาศตัวแปรทำให้เกิดการจองหน่วยความจำ ดังตารางที่ 1

ตารางที่ 1 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่จองไว้

a								b	
-	-	-	-	-	-	-	-	0	0

2. เรียกใช้ฟังก์ชัน strcpy() โดยจะนำค่าจากตัวแปรชนิดสตริง p ที่รับเข้ามาเป็น input สมมติตัวแปร p นี้มีค่าเท่ากับ “overflows” มาใส่ลงในตัวแปร a ค่าภายในหน่วยความจำเกิดการเปลี่ยนแปลง ดังตารางที่ 2

ตารางที่ 2 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy()

a								b	
o	v	e	r	f	l	o	w	s	\0

3. แสดงค่าของตัวแปร b ซึ่งตอนนี้พบว่า ค่าของ b จากเดิมที่เป็น 0 เปลี่ยนไปเป็นค่าอื่นที่ไม่ใช่ 0 อีกต่อไป

ตัวอย่างดังกล่าวพบว่า หน่วยความจำส่วนที่เป็นตัวแปร b เป็นส่วนที่ผู้เรียกใช้โปรแกรมตัวอย่างนี้ไม่สามารถแก้ไขค่าของตัวแปร b ได้โดยตรง แต่ผู้เรียกใช้โปรแกรมนี้สามารถแก้ไขค่าของตัวแปร b ได้โดยการใช้เทคนิคบัฟเฟอร์โอเวอร์โฟลว์นั่นเอง และถ้าหากเปลี่ยนจากตัวแปร b นี้เป็น รีเทิร์นแอดเดรส (Return Address) แสดงว่า ผู้เรียกใช้โปรแกรมนี้สามารถเปลี่ยนแปลงตำแหน่งของหน่วยความจำที่โปรแกรมนั้นกำลังทำงานอยู่ให้ไปทำงานในส่วนอื่นแทน ซึ่งส่วนนั้นอาจเป็นโค้ดที่อันตรายต่อเครื่องคอมพิวเตอร์ได้

กล่าวโดยสรุปคือ เทคนิคบัฟเฟอร์โอเวอร์โฟลว์นี้จะช่วยให้ผู้โจมตีที่ไม่สิทธิ์เปลี่ยนแปลงค่าของหน่วยความจำส่วนนั้นให้เปลี่ยนแปลงค่าได้ตามที่ต้องการ ดังนั้นเทคนิคนี้จึงเป็นเพียงการปูทางในขั้นต้นเพื่อนำไปใช้ในการโจมตีเครื่องคอมพิวเตอร์ต่อไป

2.1.2 รูปแบบของการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์

การโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์สามารถแบ่งเป็นหลายรูปแบบ ซึ่งมีรูปแบบเฉพาะตัวแตกต่างกันไป ในการวิจัยนี้จะกล่าวถึงสามรูปแบบหลักๆที่เกี่ยวข้อง ได้แก่

2.1.2.1 **สแต็กโอเวอร์โฟลว์** (Stack Overflows)

สแต็กโอเวอร์โฟลว์เป็นรูปแบบการโจมตีที่เปลี่ยนแปลงค่าภายในหน่วยความจำส่วนที่เป็นสแต็ก (Stack) ของโพรเซส (Process) ส่วนใหญ่จะเปลี่ยนแปลงค่ารีเทิร์นแอดเดรส (Return Address) ซึ่งถูกเก็บอยู่ในสแต็กนั่นเอง โดยอาศัยการคัดลอกข้อมูลที่มีขนาดใหญ่กว่าบัฟเฟอร์ที่ถูกประกาศไว้ในฟังก์ชันนั้น ซึ่งถูกเก็บอยู่ในสแต็กเช่นกัน ทำให้ข้อมูลส่วนที่ล้นออกมาจากบัฟเฟอร์ไปทับรีเทิร์นแอดเดรสซึ่งเป็นข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data) ค่ารีเทิร์นแอดเดรสจึงถูกเปลี่ยนแปลงไปตามที่ผู้โจมตีต้องการ เมื่อฟังก์ชันนั้นจบการทำงานแล้วโปรแกรมนั้นกลับไปทำงานในโค้ดที่ผู้โจมตีต้องการ แทนที่จะเป็นโปรแกรมเดิมที่กำลังทำงานอยู่

ในที่นี้จะยกตัวอย่างการใช้เทคนิคบัฟเฟอร์โอเวอร์โฟลว์ที่เป็นสแต็กโอเวอร์โฟลว์ 2 ตัวอย่าง ดังนี้

2.1.2.1.1 **ตัวอย่างที่ 1 ผู้โจมตีต้องการเปลี่ยนแปลงข้อมูลส่วนที่เป็นตัวควบคุมระบบ** (Control Data)

มีตัวอย่างโค้ดของโปรแกรมภาษาซีดังนี้

```
#include <stdio.h>
#include <string.h>
void foo(const char* input) {
  char buf[10];
  //View stack
  printf("My stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");

  //Pass input to buf
  strcpy(buf,input);
  printf("%s\n",buf);

  printf("Now stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
}
void bar() {
  printf("Pongo hack YOU!\n");
}
int main(int argc, char *argv[]) {
  printf("Address of foo = %p\n",foo);
  printf("Address of bar = %p\n",bar);
  if(argc != 2){
    printf("Enter string as an argument!\n");
    return -1;
  }
  foo(argv[1]);
  return 0;
}
```

ตัวอย่างดังกล่าวจะเห็นได้ว่า ผู้โจมตีต้องการเปลี่ยนแปลงค่ารีเทิร์นแอดเดรสซึ่งเป็นข้อมูลส่วนที่เป็นตัวควบคุมระบบ เพื่อให้กลับไปทำงานที่ฟังก์ชันที่ผู้โจมตีต้องการแทน (ในที่นี้คือฟังก์ชัน bar) มีขั้นตอนการทำงานดังนี้

1. หาตำแหน่งของค่ารีเทิร์นแอดเดรส เพื่อคำนวณระยะห่างระหว่างตำแหน่งของค่ารีเทิร์นแอดเดรสกับตำแหน่งของบัฟเฟอร์ที่ถูกประกาศไว้ภายในฟังก์ชันนั้น ด้วยโค้ดบรรทัดนี้

```
printf("My stack looks like:\n%p\n%p\n%p\n%p\n%p\n%p\n%p\n\n");
```

โค้ดบรรทัดดังกล่าวจะแสดงค่าภายในสแต็กของโพรเซส ณ เวลาที่ทำงานโค้ดบรรทัดนี้ โดย %p ตัวแรกจะแสดงค่าเป็นตัวเลขฐาน 16 ตั้งแต่ข้อมูลตำแหน่งบนสุดของสแต็ก (ตำแหน่งแอดเดรสค่าน้อยกว่า) และ %p ตัวต่อมาจะแสดงตั้งแต่ข้อมูลตำแหน่งที่ 2 นับจากบนสุดของสแต็ก (ตำแหน่งแอดเดรสค่ามากกว่า) ไล่ตามลำดับ หากสั่งให้โปรแกรมทำงานเพื่อทดลองหาตำแหน่งของค่ารีเทิร์นแอดเดรส ดังนี้

```
# ./stack_control 01234567890
```

จะได้ผลลัพธ์ดังนี้

```
Address of foo = 0x8048400
Address of bar = 0x8048444
My stack looks like:
0xbffffb98
0x401081ec
0xbffffb98
0xbffffb98
0x80484ae
0xbffffcd7
0xbffffbb8

01234567890
Now stack looks like:
0x33323130
0x37363534
0x303938
0xbffffb98
0x80484ae
0xbffffcd7
0xbffffbb8
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า ตำแหน่งเริ่มต้นของฟังก์ชัน foo อยู่ที่ 8048400_{16} ส่วนตำแหน่งเริ่มต้นของฟังก์ชัน bar อยู่ที่ 8048444_{16} และผังสแต็กก่อนเรียกใช้ฟังก์ชัน strcpy() เป็นดังตารางที่ 3

ตารางที่ 3 ผังสแต็ก (Stack Layout) ของหน่วยความจำก่อนการเรียกใช้ฟังก์ชัน strcpy()

Address	Content	Symbol
[E]SP+0x00	0xbffffb98	buf
[E]SP+0x04	0x401081ec	
[E]SP+0x08	0xbffffb98	
[E]SP+0x0c	0xbffffb98	-
[E]SP+0x10	0x80484ae	return address
[E]SP+0x14	0xbffffcd7	-
[E]SP+0x18	0xbffffbb8	-

หมายเหตุ [E]SP คือเรจีสเตอร์ (Register) อันหนึ่งในหน่วยประมวลผลกลาง (CPU) ทำหน้าที่เก็บค่าสแต็กพอยท์เตอร์ (Stack Pointer) ซึ่งจะเก็บตำแหน่งบนสุดของสแต็ก (Stack) ณ ขณะนั้นๆ

จากตารางจะเห็นได้ว่าเนื่องจากตัวแปร buf ซึ่งประกาศไว้เป็นบัฟเฟอร์ภายในฟังก์ชัน foo ดังนั้นจึงสรุปได้ว่าหน่วยความจำที่ตำแหน่ง [E]SP+0x00 ถึง [E]SP+0x08 เป็นตัวแปร buf นั่นเอง โดยหน่วยความจำนั้นจะถูกจองจะเป็นจำนวนเวิร์ด (Word) หรือ 4 เท่าของจำนวนไบต์เสมอ (Byte) เนื่องจาก 1 เวิร์ดเท่ากับ 4 ไบต์นอกจากนี้หน่วยความจำที่ตำแหน่ง [E]SP+0x10มีค่าใกล้เคียงกับตำแหน่งเริ่มต้นของฟังก์ชันทั้งสอง และเนื่องจากโค้ดของโปรแกรมจะถูกโหลดขึ้นมาบนหน่วยความจำบริเวณเดียวกันเสมอ ดังนั้นจึงสรุปได้ว่าตำแหน่งนั้นคือค่ารีเทิร์นแอดเดรส (Return Address)

ผังสแต็กหลังเรียกใช้ฟังก์ชัน strcpy() เพื่อคัดลอกค่าจากอาร์กิวเมนต์ (Argument) ซึ่งถือว่าเป็นข้อมูลนำเข้า (Input Data) เข้ามาใส่บัฟเฟอร์ที่ถูกประกาศไว้ในฟังก์ชัน foo เป็นดังตารางที่ 4

ตารางที่ 4 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy()

Address	Content	Symbol
[E]SP+0x00	0x33323130	buf
[E]SP+0x04	0x37363534	
[E]SP+0x08	0x00303938	
[E]SP+0x0c	0xbffffb98	-
[E]SP+0x10	0x80484ae	return address
[E]SP+0x14	0xbffffcd7	-
[E]SP+0x18	0xbffffbb8	-

จากตารางจะเห็นได้ว่า ตำแหน่งของค่ารีเทิร์นแอดเดรสห่างจากตำแหน่งของบัฟเฟอร์ที่ถูกประกาศไว้อยู่ 16 ไบต์ (Bytes) ซึ่งค่าของผังสแต็กจะเก็บเป็นค่ารหัสแอสกี (ASCII) เช่น ตัวอักษร 0 ที่ใส่เข้ามามีค่ารหัสแอสกีเท่ากับ 30_{16} (แปลว่า 30 ฐาน 16) หรือเท่ากับ 48_{10} (แปลว่า 48 ฐาน 10) นั่นเอง โดยเก็บย้อนลำดับกับที่ใส่เป็นอาร์กิวเมนต์เข้ามาตอนสั่งให้โปรแกรมทำงาน

2. ถ้าต้องการเปลี่ยนแปลงค่ารีเทิร์นแอดเดรส ให้เป็นตำแหน่งเริ่มต้นของฟังก์ชัน bar แทน จะต้องใส่ค่าไป 16 ตัวอักษรก่อน (1 ตัวอักษรเท่ากับ 1 ไบต์) แล้วใส่ตัวอักษรที่มีค่ารหัสแอสกีเท่ากับ 44_{16} 84_{16} 04_{16} 08_{16} (หรือ 68_{10} $132_{10}4_{10}$ 8_{10}) ซึ่งเป็นค่าที่กลับลำดับกับตำแหน่งเริ่มต้นของฟังก์ชัน bar โดยกดปุ่ม Alt ฝั่งซ้ายบนคีย์บอร์ดค้างไว้ แล้วกดตัวเลขรหัสแอสกีเป็นฐาน 10 บนแผงแป้นตัวเลข (Numeric Keypad) หรือกดปุ่ม Alt ฝั่งขวาค้างไว้แล้วกดรหัสแอสกีเป็นฐาน 16 หลังจากนั้นจึงปล่อยปุ่ม Alt เช่น กดปุ่ม Alt ค้างไว้ และกดปุ่ม 6 กับปุ่ม 8 แล้วจึงปล่อยปุ่ม Alt จะได้ตัวอักษร D ขึ้นมา ดังนั้นจึงสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./stack_control 0123456789012345Dä♦◘
```

จะได้ผลลัพธ์ดังนี้

```
Address of foo = 0x8048400
Address of bar = 0x8048444
My stack looks like:
0xbffffba8
0x401081ec
0xbffffba8
0xbffffba8
0x80484ae
0xbffffcdf
0xbffffbc8

0123456789012345D
Now stack looks like:
0x33323130
0x37363534
0x31303938
0x35343332
0x8040044
0xbffffcdf
0xbffffbc8

Pongo hack YOU!
Segmentation fault (core dumped)
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า ฟังก์ชัน bar ซึ่งไม่ได้ถูกเรียกใช้ภายในโค้ดของโปรแกรมนี้ กลับถูกเรียกใช้ให้ทำงาน เนื่องจากผังสแต็ก (Stack Layout) หลังเรียกใช้ฟังก์ชัน strcpy() เป็นดังตารางที่ 5

ตารางที่ 5 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy()

Address	Content	Symbol
[E]SP+0x00	0x33323130	buf
[E]SP+0x04	0x37363534	
[E]SP+0x08	0x31303938	
[E]SP+0x0c	0x35343332	-
[E]SP+0x10	0x8048444	bar
[E]SP+0x14	0xbffffce5	-
[E]SP+0x18	0xbffffbc8	-

จากตารางจะเห็นได้ว่า ค่ารีเทิร์นแอดเดรสสำหรับตำแหน่งของโปรแกรมที่ต้องกลับไปทำงานหลังจากทำงานภายในฟังก์ชัน foo เสร็จแล้วถูกเปลี่ยนแปลงเป็นตำแหน่งเริ่มต้นของฟังก์ชัน bar ทำให้จากเดิมที่จะต้องกลับไปทำงานต่อภายในฟังก์ชัน main ตามปกติ แต่กลับไปทำงานในฟังก์ชัน bar แทน

2.1.2.1.2 **ตัวอย่างที่ 2 ผู้โจมตีต้องการเปลี่ยนแปลงข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ** (Non-control Data)

ถ้าหากข้อมูลที่ถูกเปลี่ยนแปลงไม่ใช่รีเทิร์นแอดเดรส แต่อาจเป็นข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ หากกล่าวให้ถูกต้องชัดเจนคือเป็นตัวแปรที่ถูกประกาศภายในฟังก์ชัน (Local Variable) ซึ่งถูกเก็บอยู่ในสแต็กเช่นเดียวกับบัฟเฟอร์ที่ถูกประกาศไว้ติดกัน ตัวอย่างเช่นมีตัวอย่างโค้ดของโปรแกรมภาษาซีดังนี้

```
int main(int argc,char* argv[])
{
  int data2 = 0;
  char buf[10];

  if(argc != 2)
  {
    printf("Enter an argument! e.g. 01234567890123456\n");
    return -1;
  }

  printf("buf at %p\ndata2 at %p\n",buf,data2);

  printf("Before copy\nbuf = %s\n",buf);
```

```
  printf("data2 = %d at %p\n",data2,&data2);

  strcpy(buf,argv[1]);

  printf("After copy\nbuf = %s\n",buf);
  printf("data2 = %d at %p\n",data2,&data2);

  return 0;
}
```

ตัวอย่างดังกล่าวแสดงให้เห็นว่า หากคัดลอกข้อมูลจากอาร์กิวเมนต์ (Argument) ที่มีขนาดใหญ่กว่าบัฟเฟอร์ที่ถูกประกาศไว้ต่อจากตัวแปรชนิดจำนวนเต็ม มายังบัฟเฟอร์นั้น อาจทำให้ตัวแปรที่ติดกันอาจถูกเปลี่ยนแปลงค่าได้เนื่องจากตัวแปรที่ถูกประกาศภายในโปรแกรม โดยเฉพาะตัวแปรที่ประกาศภายในฟังก์ชัน ในที่นี้คือฟังก์ชันหลักของโปรแกรม (Main Function) จะถูกเก็บอยู่ในสแต็ก หากมีหลายตัวแปร หน่วยความจำส่วนที่เป็นสแต็กจะถูกจองตามลำดับ โดยตัวแปรที่ถูกประกาศตัวสุดท้ายจะอยู่ตำแหน่งบนสุดของสแต็ก (ตามคุณสมบัติของสแต็ก) ดังนั้น ทั้งตัวแปร data2 และ buf จะถูกเก็บอยู่ในสแต็กซึ่งอยู่ติดกัน

หากสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./stack_noncontrol 01234567890123456
```

จะได้ผลลัพธ์ดังนี้

```
buf at 0xbffffb98
data2 at 0xbffffba4
Before copy
buf = ¨ûÿ¿+„€–
-----------------
data2 = 0 at 0xbffffba4
After copy
buf = 01234567890123456
data2 = 892613426 at 0xbffffba4
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า ตัวแปร data2 ซึ่งไม่ได้ถูกเปลี่ยนแปลงค่าใหม่ภายในโค้ดของโปรแกรมนี้ กลับมีค่าเปลี่ยนไปจากเดิม เนื่องจากผังสแต็ก (Stack Layout) หลังเรียกใช้ฟังก์ชัน strcpy() เป็นดังตารางที่ 6

ตารางที่ 6 ผังสแต็ก (Stack Layout) ของหน่วยความจำหลังการเรียกใช้ฟังก์ชัน strcpy()

Address	Content	Symbol
buf+0x00	0x33323130	buf
buf+0x04	0x37363534	
buf+0x08	0x31303938	
buf+0x0c	0x35343332	data2

จากตารางจะเห็นได้ว่า ตัวแปร data2 ซึ่งเป็นตัวแปรชนิดจำนวนเต็มที่มีขนาด 4 ไบต์มีค่าเปลี่ยนไปจากเดิมที่มีค่าเป็น 0 กลายเป็น 35343332_{16} หรือ 392613426_{10} เนื่องจากถูกข้อมูลนำเข้าส่วนที่ล้นบัฟเฟอร์มาทับตัวแปร data2 นั่นเอง

2.1.2.2 **ฮีปโอเวอร์โฟลว์** (Heap Overflows)

ฮีปโอเวอร์โฟลว์เป็นรูปแบบการโจมตีที่คล้ายคลึงกับสแต็กโอเวอร์โฟลว์ (Stack Overflows) แต่จะเปลี่ยนแปลงค่าภายในหน่วยความจำส่วนที่เป็นฮีป (Heap) ของโพรเซส (Process) แทนสแต็ก (Stack) ซึ่งภายในฮีปจะเก็บข้อมูลที่ถูกจองเพิ่มขึ้นมาในขณะที่โปรแกรมกำลังทำงาน (Dynamic Allocation) เช่น ข้อมูลที่ถูกจองเพิ่มขึ้นมาด้วยฟังก์ชัน malloc เป็นต้น ดังนั้นข้อมูลส่วนที่ล้นออกมาจากบัฟเฟอร์อาจไปเปลี่ยนแปลงค่าของข้อมูลเหล่านั้นตามที่ผู้โจมตีต้องการ แล้วอาจทำให้โปรแกรมนั้นกลับไปทำงานในโค้ดที่ผู้โจมตีต้องการเช่นเดียวกับสแต็กโอเวอร์โฟลว์

2.1.2.3 **อาร์เรย์อินเด็กซิงเออเรอร์** (Array Indexing Errors)

อาร์เรย์อินเด็กซิงเออเรอร์เป็นรูปแบบการโจมตีที่แตกต่างกับสแต็กโอเวอร์โฟลว์ (Stack Overflows) และ ฮีปโอเวอร์โฟลว์ (Heap Overflows) ตรงที่จะไม่ได้อาศัยการคัดลอกข้อมูลที่มีขนาดใหญ่กว่าบัฟเฟอร์ แต่จะอาศัยตัวแปรชนิดอาร์เรย์ (Array) ที่ถูกประกาศไว้ในฟังก์ชัน แล้วอ้างอิงอินเด็กซ์ (Index) ที่อยู่นอกเหนือขอบเขตของอาร์เรย์นั้นเช่น ประกาศตัวแปรชนิดอาร์เรย์ขนาด 10 ช่อง ซึ่งจะสามารถอ้างอิงอินเด็กซ์ตั้งแต่อินเด็กซ์ที่ 0 ถึง 9 แต่ถ้าเราอ้างอิงอินเด็กซ์ที่ -1 หรือ 10 ก็ถือว่าอยู่นอกเหนือขอบเขตของอาร์เรย์นั้น ซึ่งทำให้ค่าของตัวแปรเหล่านั้นถูกเปลี่ยนแปลงไปตามที่ผู้โจมตีต้องการ แล้วอาจทำให้โปรแกรมนั้นกลับไปทำงานในโค้ดที่ผู้โจมตีต้องการเช่นเดียวกับสแต็กโอเวอร์โฟลว์และฮีปโอเวอร์โฟลว์

ตัวอย่างการใช้เทคนิคบัฟเฟอร์โอเวอร์โฟลว์ที่เป็นอาร์เรย์ อินเด็กซิงเออเรอร์ มีตัวอย่างโค้ดของโปรแกรมภาษาซีดังนี้

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

int* IntVector;

void bar()
{
  printf("Pongo hack YOU!\n");
}

void InsertInt(unsigned long index, unsigned long value)
{
  printf("Address of index = %p, index = %x\n",&index,index);
  printf("My stack looks like:\n%p\n%p\n%p\n%p\n%p\n");
```

```
  printf("Writing memory at %p\n",&(IntVector[index]));
  IntVector[index] = value;
  printf("Writed successfully\n");
}
bool InitVector(int size)
{
  IntVector = (int*)malloc(sizeof(int)*size);
  printf("Address of IntVector is %p\n",IntVector);
  if(IntVector == NULL)
    return false;
  else
    return true;
}

int main(int argc, char *argv[])
{
  unsigned long index, value;

  if(argc != 3)
  {
    printf("Usage is %s [index] [value]\n");
    return -1;
  }

  printf("Address of bar = %p\n",bar);

  //Let's initialize our vector - 64 KB ought to be enough for
anyone <g>.
  if(!InitVector(0xffff))
  {
    printf("Cannot initialize vector!");
    return -1;
  }

  index = atol(argv[1]);
  value = atol(argv[2]);
  InsertInt(index, value);
  return 0;
}
```

ตัวอย่างดังกล่าวจะเห็นได้ว่า ผู้โจมตีต้องการเปลี่ยนแปลงค่ารีเทิร์นแอดเดรสเพื่อให้กลับไปทำงานที่ฟังก์ชันที่ผู้โจมตีต้องการแทน (ในที่นี้คือฟังก์ชัน bar) มีขั้นตอนการทำงานดังนี้

1. หาตำแหน่งของค่ารีเทิร์นแอดเดรส (Return Address) เพื่อคำนวณระยะห่างระหว่างตำแหน่งของค่ารีเทิร์นแอดเดรสกับตำแหน่งช่องที่ 0 ของตัวแปรอาร์เรย์ที่ถูกประกาศไว้ ในที่นี้คือตัวแปร IntVector ที่จองหน่วยความจำไว้ $ffff_{16}$ (หรือ 65535_{10}) เท่าของขนาดตัวแปรชนิดจำนวนเต็ม (4 ไบต์) หากสั่งให้โปรแกรมทำงาน เพื่อทดลองหาตำแหน่งของค่ารีเทิร์นแอดเดรส ดังนี้

```
# ./array_control 1234567890 9876543210
```

จะได้ผลลัพธ์ดังนี้

```
Address of bar = 0x8048430
Address of IntVector is 0x4010d008
Address of index = 0xbffffb68, index = 499602d2
My stack looks like:
0x401081ec
0xbffffb78
0x80485a5
0x499602d2
0x7fffffff
Writing memory at 0x6668db50
Segmentation fault (core dumped)
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า ตำแหน่งเริ่มต้นของฟังก์ชัน bar อยู่ที่ 8048430_{16}หรือ 134513712_{10} ส่วนตำแหน่งช่องที่ 0 ของตัวแปร IntVector อยู่ที่ $4010d008_{16}$ส่วนตำแหน่งของตัวแปร index ซึ่งเป็นอาร์กิวเมนต์ (Argument) ของฟังก์ชันที่มีโค้ดคำสั่ง printf ที่แสดงค่าของผังสแต็ก อยู่ที่ $bffffb68_{16}$

เนื่องจากสิ่งที่เกิดขึ้นเมื่อเรียกใช้ฟังก์ชันที่มีอาร์กิวเมนต์หลายตัว คือ จะนำค่าของอาร์กิวเมนต์ตัวสุดท้ายเก็บลงในสแต็ก (Push into stack) ตามลำดับจนมาถึงอาร์กิวเมนต์ตัวแรกของฟังก์ชัน แล้วค่ารีเทิร์นแอดเดรส (Return Address) ก็ถูกเก็บลงในสแต็กด้วยเช่นกันก่อนที่โปรแกรมจะทำงานในฟังก์ชันที่ถูกเรียกใช้ ดังนั้น ตำแหน่งของรีเทิร์นแอดเดรสจึงต้องอยู่ถัดจากอาร์กิวเมนต์ของฟังก์ชันนั้น 1 ตำแหน่ง เท่ากับ 4 ไบต์ ได้ว่า ตำแหน่งของรีเทิร์นแอดเดรสอยู่ที่ $bffffb68_{16} - 4_{16}$ เท่ากับ $bffffb64_{16}$ เพื่อให้เข้าใจได้อย่างชัดเจน สามารถดูค่าของผังสแต็กเป็นดังตารางที่ 7

ตารางที่ 7 ผังสแต็ก (Stack Layout)

Address	Content	Symbol
0xbffffb6c	0x401081ec	-
0xbffffb70	0xbffffb78	-
0xbffffb74	0x80485a5	return address
0xbffffb78	0x499602d2	index
0xbffffb7c	0x7fffffff	value

2. คำนวณค่าอินเด็กซ์ใหม่ที่ต้องการอ้างอิงถึงตำแหน่งของค่ารีเทิร์นแอดเดรส ได้จากสมการดังนี้

ตำแหน่งของสมาชิกที่อ้างอิงอินเด็กซ์นั้น = ตำแหน่งของสมาชิกตัวแรก + อินเด็กซ์ x 4

อินเด็กซ์ = (ตำแหน่งของสมาชิกที่อ้างอิงอินเด็กซ์นั้น – ตำแหน่งของสมาชิกตัวแรก))/ 4

ดังนั้น อินเด็กซ์ใหม่ = (ตำแหน่งของค่ารีเทิร์นแอดเดรส – ตำแหน่งของสมาชิกตัวแรก)/ 4

อินเด็กซ์ใหม่ = $(1bffffb64_{16} - 4010d008_{16}) / 4_{16} = 5ffbcad7_6 = 1610336983_{10}$

หมายเหตุ ค่า $1bffffb64_{16}$กับ $bffffb64_{16}$ในสถาปัตยกรรมคอมพิวเตอร์ 32 บิต ถือว่าเป็นค่าเดียวกัน เนื่องจากสามารถเก็บค่าได้สูงสุดเพียง 32 บิตเท่านั้น ทำให้เลข 1 ที่อยู่หน้าสุดถูกตัดทิ้งออกไป

อีกทั้ง เนื่องจากประกาศตัวแปรอาร์เรย์ด้วยฟังก์ชัน malloc ทำให้หน่วยความจำที่ถูกจองขึ้นมาจึงอยู่ในส่วนของฮีป (Heap) ซึ่งจะเรียงอินเด็กซ์ตามลำดับจากตำแหน่งแอดเดรสค่าน้อยไปตำแหน่งแอดเดรสค่ามาก สมการจึงเป็นการบวกจากตำแหน่งของสมาชิกตัวแรก

3. ถ้าต้องการเปลี่ยนแปลงค่ารีเทิร์นแอดเดรส ให้เป็นตำแหน่งเริ่มต้นของฟังก์ชัน bar แทน จะต้องใส่ค่าอาร์กิวเมนต์ตัวแรกเป็นอินเด็กซ์ใหม่ที่คำนวณได้ คือ 1610336983 และอาร์กิวเมนต์ตัวที่ 2 เป็นตำแหน่งเริ่มต้นของฟังก์ชัน bar คือ 134513712 ดังนั้นจึงสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./array_control 1610336983 134513712
```

จะได้ผลลัพธ์ดังนี้

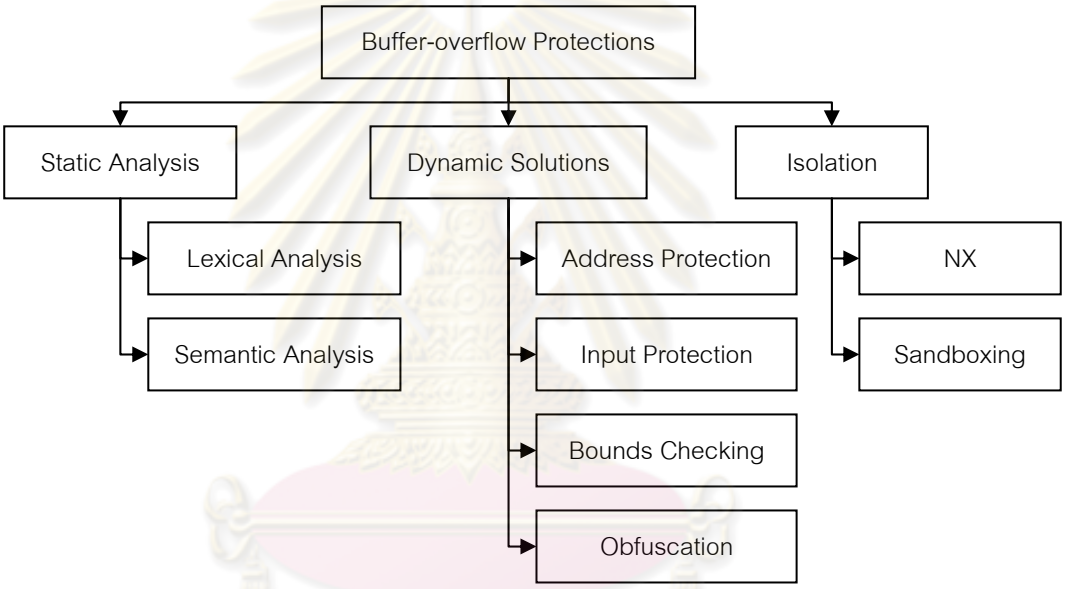
```
Address of bar = 0x8048430
Address of IntVector is 0x4010d008
Address of index = 0xbffffb68, index = 5ffbcad7
My stack looks like:
0x401081ec
0xbffffb78
0x80485a5
0x5ffbcadb
0x8048430
Writing memory at 0xbffffb64
Writed successfully
Pongo hack YOU!
Segmentation fault (core dumped)
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า ฟังก์ชัน bar ซึ่งไม่ได้ถูกเรียกใช้ภายในโค้ดของโปรแกรมนี้ กลับถูกเรียกใช้ให้ทำงาน

หากสังเกตเพิ่มเติมรูปแบบการโจมตีนี้ สามารถแก้ไขค่าในหน่วยความจำในตำแหน่งใดๆที่ไม่จำเป็นต้องอยู่ในตำแหน่งแอดเดรสค่ามากกว่าเท่านั้น ซึ่งไม่เหมือนกับสแต็กโอเวอร์โฟลว์ที่ต้องอาศัยการล้นทับข้อมูลจากบัฟเฟอร์ในทิศทางจากตำแหน่งแอดเดรสค่าน้อยไปยังตำแหน่งแอดเดรสค่ามากเพียงทิศทางเดียวเท่านั้น

2.2 เอกสารและงานวิจัยที่เกี่ยวข้อง

ถึงแม้ว่าวิธีการป้องกันในเบื้องต้นที่ดีที่สุดคือ การเขียนโค้ดของโปรแกรมให้ถูกต้อง (Correct Code) เพื่อไม่ให้เกิดการบัฟเฟอร์โอเวอร์โฟลว์ได้ เช่น การตรวจสอบขนาดของข้อมูลที่จะนำมาใส่ลงในบัฟเฟอร์ว่า มีขนาดใหญ่เกินขนาดของบัฟเฟอร์หรือไม่ เป็นต้น วิธีนี้ถือว่าเป็นวิธีการป้องกันที่ดีที่สุด แต่เนื่องจากจะทำให้โปรแกรมทำงานได้ช้าลง ถือว่าทำให้ประสิทธิภาพของโปรแกรมต่ำลง ผู้เขียนโปรแกรมจึงมักไม่ได้ตรวจสอบขนาดไว้ หรืออาจพยายามระมัดระวังในการเขียนโปรแกรมแล้ว แต่ไม่สามารถป้องกันได้ทุกกรณี จึงทำให้เกิดช่องโหว่ที่สามารถถูกนำมาใช้ในการโจมตีได้ ดังนั้น ในปัจจุบันจึงมีนักวิจัยเสนอวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์ออกมาหลากหลายวิธี ซึ่งสามารถแบ่งวิธีการต่างๆเป็นดังรูปที่ 2



รูปที่ 2 วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบต่างๆ

จากภาพจะเห็นได้ว่า การป้องกันสามารถแบ่งเป็นสามรูปแบบใหญ่ๆ [7] ได้แก่

2.2.1 การวิเคราะห์โค้ดของโปรแกรม (Static Analysis)

การวิเคราะห์ตรวจสอบโค้ดของโปรแกรมคือ การแจ้งเตือนให้นักเขียนโปรแกรมแก้ไขหรือแทนที่ฟังก์ชันหรือส่วนของโปรแกรมที่อาจเกิดบัฟเฟอร์โอเวอร์โฟลว์ได้ ก่อนที่จะนำโปรแกรมนั้นไปใช้งานจริง (Deploy the Program) เช่น การใช้ฟังก์ชัน strcpy() อาจทำให้เกิดช่องโหว่ของโปรแกรมได้ง่าย จึงแก้ไขโค้ดของโปรแกรมให้เปลี่ยนไปใช้ฟังก์ชัน strncpy() ซึ่งมีการตรวจสอบขอบเขตของบัฟเฟอร์แทน เป็นต้น

นอกจากนี้ วิธีการรูปแบบนี้ยังมีข้อจำกัดในการตรวจหาการเกิดบัฟเฟอร์โอเวอร์โฟลว์ในโค้ดส่วนที่เป็นฟังก์ชันผู้ใช้กำหนด (User-defined Function) และแมโคร (Macro) รวมทั้งมีจุดอ่อนคือ ตรวจหาได้เฉพาะรูปแบบการโจมตีที่รู้จักแล้วเท่านั้น เพราะไม่มีข้อมูลขณะโปรแกรมทำงาน (Runtime Information) จึงมีโอกาสที่จะไม่สามารถตรวจหาได้ทุกรูปแบบ และอาจเกิดสัญญาณเตือนที่ผิดพลาดได้ (False Alarm) อีกทั้งในขั้นตอนสุดท้าย นักเขียนโปรแกรมเป็นผู้ตัดสินใจว่าจะแก้ไขตามคำเตือนที่ได้รับจากเครื่องมือนั้นหรือไม่

วิธีการรูปแบบนี้ สามารถแบ่งเป็นรูปแบบย่อยได้อีกสองรูปแบบย่อย ได้แก่

2.2.1.1 **การวิเคราะห์คำ** (Lexical Analysis)

วิธีนี้จะใช้ขั้นตอนวิธี (Algorithm) เพื่อตรวจสอบแต่ละคำในโค้ดของโปรแกรมว่าพบคำ หรือกลุ่มของคำที่มีโอกาสเกิดบัฟเฟอร์โอเวอร์โฟลว์ได้หรือไม่

ตัวอย่างของเครื่องมือที่ใช้วิธีนี้ ได้แก่ ITS4 [8] FlawFinder [9] RATS [10] STOBO [11] และ LibSafe [12]

2.2.1.2 **การวิเคราะห์ความหมาย** (Semantic Analysis)

วิธีนี้ต่างกับวิธีการวิเคราะห์คำ (Lexical Analysis) โดยใช้ตัวแจงส่วน (Parser) ในการวิเคราะห์ความหมายของโค้ดของโปรแกรม แทนการใช้ขั้นตอนวิธี (Algorithm) เพื่อตรวจสอบแต่ละคำ

ตัวอย่างของเครื่องมือที่ใช้วิธีนี้ ได้แก่ Splint [13] และ BOON [14]

2.2.2 **การตรวจสอบขณะโปรแกรมทำงาน** (Dynamic Solution)

การตรวจสอบขณะโปรแกรมทำงาน จะตรวจสอบความสมเหตุสมผลของบูรณภาพ (Validate the Integrity) ของข้อมูล ในขณะที่โปรแกรมทำงาน (Run-time Environment) โดยตรวจสอบเมทาดาตา (Metadata) หรือคำอธิบายข้อมูลของข้อมูลแทน ข้อมูลที่ถูกตรวจสอบอาจเป็นได้ทั้งหน่วยความจำข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data) และส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) เช่น ตัวแปรภายในโปรแกรม เป็นต้น

เมทาดาตา (Metadata) มีหลากหลายชนิด แต่สามารถแบ่งเป็นสองประเภทหลัก ได้แก่ ประเภทที่สนับสนุนโดยฮาร์ดแวร์ (Hardware Supported) และประเภทที่จัดการโดยซอฟต์แวร์ (Software Managed)

วิธีการรูปแบบนี้สามารถแบ่งเป็นรูปแบบย่อยได้อีกสี่รูปแบบย่อย โดยแบ่งตามข้อสมมติ (Assumption) ชนิดของเมทาดาตา การจัดการเมทาดาตา และรูทีนจัดกระทำ (Handling Routine) ได้แก่

2.2.2.1 **การป้องกันแอดเดรส** (Address Protection)

วิธีนี้สามารถแบ่งออกได้อีกหลายวิธี โดยมีข้อสมมติเดียวกันคือ เน้นป้องกันเฉพาะหน่วยความจำส่วนที่เก็บข้อมูลที่เป็นแอดเดรส (ตัวแปรที่เก็บตำแหน่งของหน่วยความจำ) เนื่องจากเป็นข้อมูลวิกฤต (Critical Data) และควรถูกติดป้ายระบุไว้ เมทาดาตา (Metadata) จะถูกสร้างจากคำสั่งที่สร้างแอดเดรส และถูกตรวจสอบเมื่อมีคำสั่งมาใช้แอดเดรสนั้น โดยแต่ละวิธีเลือกใช้ชนิดของเมทาดาตาแตกต่างกัน ตัวอย่างของวิธีนี้ ได้แก่

2.2.2.1.1 **คานารีเวิร์ด** (Canary Word)

เนื่องจากวิธีนี้เป็นหนึ่งในวิธีที่เกี่ยวข้องกับการวิจัย รายละเอียดดังกล่าวจึงอยู่ในหัวข้อ 3.1.1 หน้า 25

ตัวอย่างของวิธีนี้ ได้แก่ StackGuard [13],[15],[16],[17] เน้นปกป้องรีเทิร์นแอดเดรส (Return Address) และ ProPolice [18] เน้นปกป้องฟังก์ชันพอยท์เตอร์ (Function Pointer) โดยใช้วิธีการสลับลำดับข้อความประกาศ (Declaration Statement)

2.2.2.1.2 **การเข้ารหัสแอดเดรส** (Address Encode)

วิธีนี้ใช้การเข้ารหัสเพื่อรักษาบูรณภาพ (Integrity) ของแอดเดรส (Address) หลักการคือ เข้ารหัสแอดเดรสก่อนที่จะนำไปเก็บลงในหน่วยความจำ และถอดรหัสเมื่อต้องการอ่านกลับเข้าไปในหน่วยประมวลผลกลาง ในที่นี้เมทาดาตา (Metadata) คือ กุญแจที่ใช้เข้ารหัส

ดังนั้น ประเด็นของวิธีนี้คือการจัดการกุญแจในระยะยาว และจุดอ่อนคือมีปัญหากับอาร์เรย์ (Array) รวมทั้งสายอักขระ (String) ในภาษาซี และการกำหนดค่าเมื่อคอมไพล์ (Compile)

ตัวอย่างของวิธีนี้ ได้แก่ PointGuard [19] และ Hardware Supported PointGuard [20],[21] ซึ่งวิธีเหล่านี้ สมมติเมื่อพอยท์เตอร์ (Pointer) ถูกสร้างขึ้นมาแล้ว จะต้องไม่ถูกเปลี่ยนแปลงค่าอีก และใช้กุญแจสุ่มสำหรับแต่ละโพรเซส (Per-process Random Key) ในการเข้ารหัสแอดเดรส

2.2.2.1.3 **การสำเนาแอดเดรส** (Copy of address)

วิธีนี้ใช้การรักษาสำเนาอีกอันของแอดเดรส (Address) เพื่อรักษาบูรณภาพ (Integrity) ของแอดเดรสนั้น หลักการคือ เมื่อสร้างแอดเดรส ให้สร้างสำเนาอีกอันของแอดเดรสนั้นและเก็บไว้อย่างปลอดภัย เมื่อนำแอดเดรสมาใช้ ให้ทวนสอบ (Verify) กับสำเนานั้นก่อนนำมาใช้

วิธีนี้สามารถแบ่งออกได้อีกหลายวิธี โดยแบ่งออกตามการจัดการสำเนาของแอดเดรส

ตัวอย่างของวิธีนี้ ได้แก่ StackGhost [22] ใช้เรจีสเตอร์วินโดว์ (Register Window) ของหน่วยประมวลผลสปาร์ค (SPARC Processor) ส่วน RAS [23],[24],[25] ใช้รีเทิร์นแอดเดรสสแต็ก (Return Address Stack) ซึ่งเป็นฮาร์ดแวร์สำหรับคาดเดารีเทิร์นแอดเดรสในหน่วยประมวลผลบางชนิด ส่วน Split Stack [24] SmashGuard [26] RAD Compiler [27] RAD Binary Rewrite [28] DISE [29] StackShield [30] และ LibVerify [12] จะจองหน่วยความจำแยกเป็นสแต็กสำหรับเก็บรีเทิร์นแอดเดรส และ SCACHE [31] ใช้แคช (Cache) เพื่อจัดการสำเนาของรีเทิร์นแอดเดรส

2.2.2.1.4 **แท็ก** (Tags)

วิธีนี้ใช้แท็กเพื่อระบุว่าข้อมูลที่บรรจุอยู่ในหน่วยความจำนั้นเป็นข้อมูลชนิดไหน เช่น เป็นข้อมูลทั่วไป หรือเป็นแอดเดรส เป็นต้น

จุดอ่อนของวิธีนี้คือ ไม่สามารถใช้กับโค้ดของโปรแกรมเก่าๆได้ ทำให้โปรแกรมเหล่านั้นอาจต้องถูกดัดแปรใหม่เพื่อให้ใช้กับวิธีนี้ได้

ตัวอย่างของวิธีนี้ ได้แก่ สถาปัตยกรรมแบบแท็ก (Tagged Architecture) [32]

2.2.2.2 **การป้องกันข้อมูลนำเข้า** (Input Protection)

วิธีนี้สามารถแบ่งออกได้อีกหลายวิธี โดยมีข้อสมมติเดียวกันคือ ข้อมูลนำเข้า (Input Data) ไม่ควรนำมาใช้เป็นตัวควบคุมระบบ (Control Data) ดังนั้นจึงต้องแบ่งให้เห็นความแตกต่างระหว่างข้อมูล 2 ชนิดนี้ เนื่องจากถ้ามองในระดับฮาร์ดแวร์เอง จะไม่สามารถแบ่งแยกได้ว่าหน่วยความจำตำแหน่งนั้นเป็นตัวควบคุมระบบหรือไม่เป็นตัวควบคุมระบบ แต่ถ้ามองในระบบผู้เขียนโปรแกรมหรือคอมไพเลอร์ จึงจะสามารถแบ่งแยกความแตกต่างนี้ได้ แต่แต่ละวิธีมีการจัดการเมทาดาตา (Metadata) โดยมีวิธีการทำให้เกิดผล (Implementation) แตกต่างกัน ตัวอย่างของวิธีนี้ ได้แก่

2.2.2.2.1 **ซีเคียวบิต** (Secure Bit) [33]

เนื่องจากวิธีนี้เป็นหนึ่งในวิธีที่เกี่ยวข้องกับการวิจัย รายละเอียดดังกล่าวจึงอยู่ในหัวข้อ 3.1.2 หน้า 26

2.2.2.2.2 **ไมนอส** (Minos) [34] [35]

วิธีนี้มีหลักการคล้ายซีเคียวบิต (Secure Bit) แต่มองข้อมูลที่เกิดจากการนำเข้าข้ามมาจากเซกเมนต์ (Segment) แทน โดยมองการแบ่งส่วน (Segmentation) เป็นขอบเขต (Boundary)

จุดอ่อนของวิธีนี้คือ การแบ่งส่วน (Segmentation) ไม่มีอยู่บนทุกระบบ อีกทั้งยังไม่มีคุณสมบัติโปร่งใส (Transparent) เหมือนซีเคียวบิต ดังนั้น จึงก่อให้เกิดอุปสรรคในการทำให้เกิดผล (Implement) อย่างมากในทางปฏิบัติ

2.2.2.2.3 **พอยท์เตอร์ที่ถูกทำให้จุดด่างพร้อย** (Tainted Pointer) [36]

วิธีนี้เน้นป้องกันไม่ให้ข้อมูลนำเข้าถูกนำมาใช้เป็นพอยท์เตอร์ โดยมองข้อมูลที่เกิดจากการนำเข้าจากระบบย่อยอินพุต/เอาต์พุต (I/O Subsystem) ของระบบปฏิบัติการ (Operating System) แทน

จุดอ่อนของวิธีนี้คือ บางครั้งข้อมูลนำเข้าถูกนำมาใช้เป็นส่วนหนึ่งของการคำนวณค่าพอยท์เตอร์ (Pointer) เช่น การทำอินเด็กซิง (Indexing) เป็นต้น ส่งผลให้การปกป้องพอยท์เตอร์ของวิธีนี้มีผลกระทบให้โปรแกรมที่มีการประมวลผลในลักษณะดังกล่าวไม่สามารถทำงานได้ อีกทั้งยังมีปัญหากับโปรแกรมแบบหลายสายโยงใย (Multi-threaded Program) ซึ่งมีการส่งแอดเดรสระหว่างกันในโพรเซส (Process) จึงมีคำสั่งเพื่อให้ล้างค่าความด่างพร้อยของพอยท์เตอร์ได้ จนเป็นเหตุทำให้เกิดช่องโหว่ที่อาจถูกโจมตีได้

2.2.2.3 **การตรวจสอบขอบเขต** (Bounds Checking)

วิธีนี้สามารถแบ่งออกได้อีกหลายวิธี โดยมีข้อสมมติเดียวกันคือ การเข้าถึงข้อมูลสามารถทำให้เฉพาะขอบเขตของตัวแปรนั้นเท่านั้น โดยเมทาดาตา (Metadata) จะเกี่ยวเนื่องกับทุกบล็อก (Block) ของข้อมูลที่ถูกจองพื้นที่ และใช้เพื่อจำกัดขอบเขตการเข้าถึง วิธีนี้สามารถแบ่งเป็นสองประเภทหลัก ได้แก่

2.2.2.3.1 **ประเภทฮาร์ดแวร์** (Hardware)

วิธีนี้ใช้การแบ่งส่วน (Segmentation) โดยมีเลขที่อยู่ฐาน (Base Address) การตรวจสอบขอบเขต (Boundary Check) และวงแหวน (Ring) ซึ่งได้ทำให้เกิดผล (Implement) ในหน่วยประมวลผลไอ 432 (I432) [37]

จุดอ่อนของวิธีนี้ คือ ระบบปฏิบัติการส่วนใหญ่จะหลีกเลี่ยงการแบ่งส่วน (Segmentation) โดยตั้งค่าให้หน่วยความจำทั้งหมดเป็น 1 เซกเมนต์ (Segment) ใหญ่ เพื่อทำให้

ยังคงทำงานได้บนหน่วยประมวลผลนี้ และเพิ่มประสิทธิภาพด้วย เนื่องจากหน่วยประมวลผลนี้ใช้เวลาคำนวณมากกว่า 10-20 เท่าของเครื่อง VAX 11/780 [38]

2.2.2.3.2 ประเภทซอฟต์แวร์ (Software)

วิธีนี้จะใช้กับการตรวจสอบขอบเขตที่สามารถเข้ากันกับแบบย้อนหลังได้ (Backward Compatible) ทำให้เข้ากันได้กับคลังภาษาซีมาตรฐาน (Standard C Library) ด้วย

จุดอ่อนของวิธีนี้คือ เกิดค่าใช้จ่ายอื่น (Overhead) สูง ทำให้โปรแกรมทำงานช้าลงอย่างมาก

ตัวอย่างของวิธีนี้ ได้แก่ การตรวจขอบเขตของอาร์เรย์ (Array Bound Checking) [39] ซึ่งกำหนดว่า ค่าพอยท์เตอร์จะมีผลใช้ได้ (Valid) สำหรับหน่วยความจำบริเวณหนึ่ง (Memory Region) เท่านั้น แต่เมื่อส่งผลกระทบต่อโปรแกรมทำให้ทำงานช้าลงมากกว่า 30 เท่า ส่วน Rational PurifyPlus [40] BoundsChecker [41] SafeC [42] และ Fail-Safe [43],[44] จะแบ่งส่วน (Segmentation) โดยใช้ตารางสัญลักษณ์ (Symbol Table) เป็นตัวบอกเซกเมนต์ (Segment Descriptor)

2.2.2.4 การทำให้ยุ่งเหยิง (Obfuscation)

วิธีนี้มีหลักการว่า ความสับสนจะเพิ่มความยากในการโจมตีได้ เช่น การสลับตำแหน่งของแอดเดรส (Address Obfuscation) [45] เป็นต้น

จุดอ่อนของวิธีนี้คือ ไม่ได้ทำให้ช่องโหว่ที่จะถูกโจมตีหายไป เพียงแต่ทำให้การโจมตีทำได้ยากขึ้นเท่านั้น

ตัวอย่างของวิธีนี้ ได้แก่ พีเอเอกซ์ (PAX) [46] เป็นต้น

2.2.3 การจำกัดความเสียหายจากการถูกโจมตี (Isolation)

วิธีการรูปแบบนี้จะจำกัดความเสียหายจากการถูกโจมตีให้น้อยลงแทนการป้องกันการโจมตี เช่น การจำกัดให้โปรแกรมไม่สามารถทำงานบนส่วนอื่นที่อยู่นอกเหนือจากขอบเขตที่กำหนดไว้เท่านั้น เป็นต้น วิธีการรูปแบบนี้สามารถแบ่งเป็นสองแนวคิดหลัก ได้แก่

2.2.3.1 หน่วยความจำที่ไม่สามารถทำงานได้ (Non-executable Memory)

วิธีนี้ถูกเรียกอีกชื่อว่า เอ็นเอกซ์ (NX) โดยสมมติฐานของแนวคิดนี้มาจากข้อสังเกตว่า บัฟเฟอร์โอเวอร์โฟลว์มักจะเกิดบนสแต็ก (Stack) หรือหน่วยความจำส่วนที่เก็บข้อมูล ดังนั้น แนวทางป้องกัน การกำหนดให้หน่วยความจำส่วนที่เป็นสแต็กไม่สามารถทำงาน (Execute) เป็นโค้ดของโปรแกรมได้ นอกจากนี้ แนวคิดนี้ได้ถูกนำไปเป็นส่วนหนึ่งของหน่วยประมวลผลหลายอัน [47]

จุดอ่อนของวิธีนี้คือ ไม่ใช่การโจมตีทุกครั้งที่ใช้การแทรกโค้ด ดังนั้น วิธีนี้ป้องกันได้เฉพาะการโจมตีลักษณะที่มีการแทรกโค้ดเท่านั้น

2.2.3.2 **กล่องทราย** (Sandboxing)

วิธีนี้ใช้กลไกการบังคับด้วยนโยบาย (Policy-enforcement Mechanism) กล่าวคือ การนำโปรแกรมที่ไม่มีความน่าเชื่อถือมาให้ทำงานในสภาวะแวดล้อมที่จำกัด เช่น โปรแกรมที่ดาวน์โหลดมาจากอินเตอร์เน็ต เป็นต้น โปรแกรมดังกล่าวจะถูกห้ามแก้ไขไฟล์ระบบหรือห้ามติดต่อเรียกใช้เอพีไอ (API) บางอย่างของระบบปฏิบัติการ ดังนั้น เมื่อถูกโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์เพื่อแก้ไขโปรแกรมดังกล่าวได้ ความเสียหายก็จะถูกจำกัดอยู่ในวงแคบ และไม่เกิดผลกระทบร้ายแรงต่อระบบโดยรวม ตัวอย่างเช่น จาวาแอปเพล็ต (Java Applet) จะทำงานอยู่ภายในกล่องทรายนี้ ทำให้ไม่สามารถอ่านเขียนไฟล์ภายในเครื่องผู้ใช้ได้ เป็นต้น

นอกจากนี้ วิธีนี้ยังสามารถทำได้บนหลายระดับ ได้แก่ ระดับเคอร์เนล (Kernel Level) [48] ระดับผู้ใช้ (User Level) [49],[50],[51] และระดับที่สนับสนุนโดยฮาร์ดแวร์ (Hardware-supported) เช่น Intel LaGrande [52] TCPA [53],[54] TrustZone [55] Microsoft NGSCB [56] ChipLock [57] Bear [53] เป็นต้น

ดังนั้น วิธีนี้จะใช้ได้ผล ก็ต่อเมื่อ การรวมตัวระหว่างนโยบายความมั่นคง (Security Policy) และการทำให้เกิดผล (Implementation) กำหนดได้เหมาะสม

บทที่ 3
หลักการของซีเคียวคานารีเวิร์ด (Secure Canary Word)

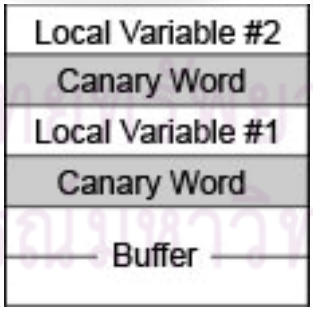
วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์ส่วนใหญ่ เป็นการป้องกันหน่วยความจำข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data) แต่การวิจัยนี้ได้เลือกวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด (Secure Canary Word) เนื่องจากคาดว่าจะสามารถป้องกันหน่วยความจำข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) ได้ด้วย ในบทนี้จะกล่าวถึงแนวคิดที่เกี่ยวข้องกับซีเคียวคานารีเวิร์ด และหลักการของซีเคียวคานารีเวิร์ด ดังนี้

3.1 แนวคิดที่เกี่ยวข้องกับซีเคียวคานารีเวิร์ด

แนวคิดที่เกี่ยวข้องกับซีเคียวคานารีเวิร์ด ได้แก่ คานารีเวิร์ด (Canary Word) และซีเคียวบิต (Secure Bit) ดังต่อไปนี้

3.1.1 หลักการของคานารีเวิร์ด (Canary Word)

คานารีเวิร์ดเป็นวิธีหนึ่งในรูปแบบของการป้องกันแอดเดรส (Address Protection) ในหัวข้อ 2.2.2.1 หน้า 20 หลักการของคานารีเวิร์ดคือ การนำเวิร์ด (Word) 1 อัน (ซึ่งถูกเรียกว่า คานารีเวิร์ด) ไปคั่นก่อนถึงหน่วยความจำที่เก็บแอดเดรส (Address) หรือพอยท์เตอร์ (Pointer) โดยอยู่บนสมมติฐานว่า บัฟเฟอร์โอเวอร์โฟลว์นั้นเป็นไปในทิศทางเดียวเท่านั้น ดังนั้น เมื่อเกิดบัฟเฟอร์โอเวอร์โฟลว์เพื่อเปลี่ยนแปลงค่าของแอดเดรสดังกล่าว จะทำให้ค่าของเวิร์ดนั้นถูกแก้ไขด้วย จึงต้องตรวจสอบค่าของเวิร์ดก่อนการนำค่าของแอดเดรสนั้นไปใช้ ดังรูปที่ 3



รูปที่ 3 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันด้วยคานารีเวิร์ด

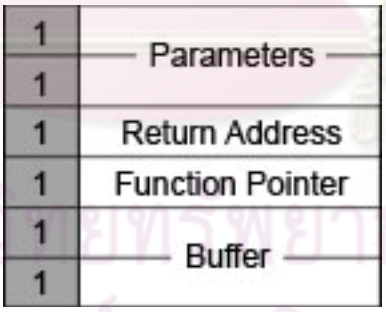
จุดอ่อนของคานารีเวิร์ดคือ ถ้าสามารถทำให้เมื่อเกิดบัฟเฟอร์โอเวอร์โฟลว์แล้วค่าของเวิร์ดนั้นเหมือนเดิม จะไม่สามารถตรวจสอบได้ว่าค่าของแอดเดรสนั้นถูกแก้ไขไปแล้ว เนื่องจากไม่มีกลไกในการปกป้องคานารีเวิร์ด ดังตารางที่ 8

ตารางที่ 8 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันด้วยคานารีเวิร์ด กรณีที่ไม่สามารถตรวจพบโอเวอร์โฟลว์ได้

ก่อนเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	Buffer			Canary Word	Pointer
	ค่า	-	-	-	0	5
หลังเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	Buffer			Canary Word	Pointer
	ค่า	A	A	A	0	A

3.1.2 **หลักการของซีเคียวบิต** (Secure Bit) [33]

ซีเคียวบิตเป็นวิธีหนึ่งในรูปแบบของการป้องกันข้อมูลนำเข้า (Input Protection) ในหัวข้อ 2.2.2.2 หน้า 21 หลักการของซีเคียวบิตคือ ในแต่ละตำแหน่งของหน่วยความจำ จะมีบิตบิตหนึ่งทำหน้าที่บอกว่าข้อมูลในหน่วยความจำตำแหน่งนี้เป็นข้อมูลที่เกิดจากการนำเข้าจากภายนอกโปรเซส (Process) ผ่านทางเคอร์เนล (Kernel) โดยบิตนั้นถูกเรียกว่า ซีเคียวบิต ถ้าบิตนี้มีค่าเป็น 1 แสดงว่าหน่วยความจำตำแหน่งนั้นเก็บค่าที่เป็นข้อมูลนำเข้า (Input Data) ซึ่งไม่ควรนำมาใช้เป็นตัวควบคุมระบบ (Control Data) เช่น รีเทิร์นแอดเดรส ดังรูปที่ 4 และตารางที่ 9



รูปที่ 4 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวบิต

ตารางที่ 9 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวบิต

ก่อนเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	Buffer					Return Address
	ค่า	-	-	-	-	-	5
	Secure Bit	0	0	0	0	0	0
หลังเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	Buffer					Return Address
	ค่า	A	A	A	A	A	A
	Secure Bit	1	1	1	1	1	1

นอกจากนี้ ซีเคียวบิตมีคุณสมบัติโปร่งใส (Transparent) นั่นคือ ผู้พัฒนาซอฟต์แวร์ไม่จำเป็นต้องแก้ไขหรือคอมไพล์ (Compile) ซอฟต์แวร์ใหม่ เพื่อให้ได้รับการป้องกันจากซีเคียวบิต เนื่องจากความสามารถในการตรวจสอบและป้องกันบัฟเฟอร์โอเวอร์โฟลว์ได้ฝังอยู่ในฮาร์ดแวร์แล้ว แต่ซีเคียวบิตยังมีจุดอ่อนคือ ไม่สามารถตรวจสอบกรณีที่เกิดบัฟเฟอร์โอเวอร์โฟลว์ข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบได้ (Non-control Data) เช่น มีตัวแปร a และ b เป็นตัวแปรภายในโปรแกรมทั้งคู่ ในครั้งแรกรับค่า b มาเก็บไว้ซึ่งเท่ากับ 5 และต่อมารับค่า a มาเก็บไว้ซึ่งเท่ากับ “AAAAA” เนื่องจากข้อมูลที่รับเข้ามาในตัวแปร a มีขนาดใหญ่เกินไป จึงเกิดบัฟเฟอร์โอเวอร์โฟลว์ไปเปลี่ยนแปลงค่าตัวแปร b ทำให้ b มีค่าเป็น 0 แทน แต่ไม่สามารถตรวจสอบได้ว่าค่า b ได้ถูกเปลี่ยนแปลง เพราะซีเคียวบิตยังคงมีค่าเท่ากับ 1 เหมือนเดิม ดังตารางที่ 10

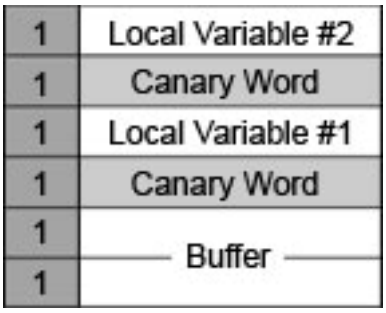
ตารางที่ 10 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวบิต กรณีที่ไม่สามารถตรวจพบโอเวอร์โฟลว์ได้

ก่อนเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	a					b
	ค่า	-	-	-	-	-	5
	Secure Bit	0	0	0	0	0	1
หลังเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	a					b
	ค่า	A	A	A	A	A	\0
	Secure Bit	1	1	1	1	1	1

3.2 **หลักการของซีเคียวคานารีเวิร์ด** (Secure Canary Word)

หลักการของซีเคียวคานารีเวิร์ดได้นำหลักการของคานารีเวิร์ด (Canary Word) และซีเคียวบิต (Secure Bit) มาใช้ร่วมกัน ทำให้มีประสิทธิภาพในการป้องกันข้อมูลทั้งส่วนที่เป็นตัวควบคุมระบบ (Control Data) และส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data)

เนื่องจากซีเคียวบิตมีบิตที่บ่งบอกว่า ข้อมูลในหน่วยความจำตำแหน่งนี้เป็นข้อมูลที่เกิดจากการนำเข้าหรือไม่ เมื่อนำคานารีเวิร์ดเข้ามาคั่นเพื่อปกป้องหน่วยความจำที่เก็บแอดเดรส (Address) หรือพอยท์เตอร์ (Pointer) คานารีเวิร์ดนี้จะถือว่าเป็นตัวควบคุมระบบอย่างหนึ่ง และอาศัยซีเคียวบิตของคานารีเวิร์ดนั้น เพื่อปิดจุดอ่อนของวิธีป้องกันด้วยคานารีเวิร์ดแบบดั้งเดิม โดยซีเคียวบิตจะตรวจสอบว่า ค่าของคานารีเวิร์ดดังกล่าวถูกเปลี่ยนแปลงแก้ไขโดยข้อมูลนำเข้าหรือไม่ ดังรูปที่ 5 และตารางที่ 11



รูปที่ 5 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวคานารีเวิร์ด

ตารางที่ 11 ผังสแต็ก (Stack Layout) ของหน่วยความจำที่ใช้วิธีป้องกันโดยวิธีซีเคียวคานารีเวิร์ด

ก่อนเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	Buffer					Canary Word	Pointer
	ค่า	-	-	-	-	-	0	5
	Secure Bit	0	0	0	0	0	0	0
หลังเกิดบัฟเฟอร์โอเวอร์โฟลว์	**ชนิดข้อมูล**	Buffer					Canary Word	Pointer
	ค่า	A	A	A	A	A	0	A
	Secure Bit	1	1	1	1	1	1	1

บทที่ 4
แนวทางการนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผล

เนื่องจากหลักการการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด (Secure Canary Word) เป็นวิธีการที่ได้นำหลักการของคานารีเวิร์ด (Canary Word) และซีเคียวบิต (Secure Bit) มาใช้ร่วมกันดังที่ได้กล่าวไว้ในบทที่ 3 ในบทนี้จะกล่าวถึงเครื่องมือที่ใช้ในการวิจัย และแนวทางการนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผล ดังนี้

4.1 เครื่องมือที่ใช้ในการวิจัย

เครื่องมือที่ใช้ในการวิจัยนี้ประกอบด้วย

4.1.1 ซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์ (BOCHS Emulator) [58]

ซอฟต์แวร์โอเพนซอร์ส (Open source Software) ที่ถูกเขียนขึ้นด้วยภาษาซีพลัสพลัส (C++) โดย Kevin Lawton เพื่อเลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ (Hardware) ตระกูล IA-32 (x86) ซึ่งเป็นหน่วยประมวลผลกลางของเครื่องคอมพิวเตอร์ส่วนบุคคล (Personal Computer: PC) ในที่นี้ได้เลือกหน่วยประมวลกลางเป็นอินเทล (Intel) 386 โดยสามารถแก้ไขปรับแต่งโครงสร้างสถาปัตยกรรม (Architecture) เหล่านั้นได้ อีกทั้งสามารถเลือกใช้ระบบปฏิบัติการ (Operating System: OS) ได้หลากหลายแบบ เพื่อจำลองเป็นเครื่องคอมพิวเตอร์ส่วนบุคคลได้

4.1.2 ลีนุกซ์อิมเมจ (Linux Image)

ไฟล์อิมเมจ (Image File) ที่เปิดใช้กับซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อโบชส์ เพื่อจำลองแทนจานบันทึกแบบแข็ง (Hard Disk) ของจริง โดยที่ติดตั้งระบบปฏิบัติการลีนุกซ์ (Linux Operating System)

4.1.3 ซอฟต์แวร์จำลองการทำงานบนระบบปฏิบัติการยูนิกซ์ชื่อ ซิกวิน (Cygwin)

ซอฟต์แวร์จำลองการทำงานบนระบบปฏิบัติการยูนิกซ์ (Unix Operating System) เพื่อคอมไพล์โค้ดของซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อโบชส์ที่ถูกแก้ไขให้นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลเข้าไปในอุปกรณ์ฮาร์ดแวร์ที่ได้ถูกจำลองการทำงานขึ้นด้วยซอฟต์แวร์ดังกล่าว

4.1.4 ซอฟต์แวร์ของระบบ (System Software)

ซอฟต์แวร์ของระบบ เช่น จีซีซี (GNU project C and C++: GCC) [59] เวอร์ชัน egcs-2.91.66 ซึ่งเป็นคอมไพเลอร์ (Compiler) ภาษาซีและซีพลัสพลัส บนระบบปฏิบัติการลีนุกซ์ เป็นต้น

4.1.5 เครื่องโน้ตบุ๊กคอมพิวเตอร์ (Notebook Computer)

เครื่องโน้ตบุ๊กคอมพิวเตอร์ รุ่น HP Pavilion dv2626tx เพื่อใช้สำหรับการทดลองวัดประสิทธิภาพของวิธีการป้องกันดังกล่าวที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ โดยมีข้อกำหนดคุณลักษณะระบบ (System Specification) คือ หน่วยประมวลผลกลาง (CPU) เป็น Intel Core 2 Duo 2.2 จิกะเฮิรตซ์ (GHz) แรม (RAM) 2 จิกะไบต์ (GB)

4.2 การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล (The Implementation of Secure Canary Word)

การวิจัยนี้ได้นำงานที่ได้นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวบิต [33] มาทำให้เกิดผลแล้วมาปรับปรุงต่อ โดยแบ่งงานออกเป็น 2 ส่วนหลัก ดังนี้

4.2.1 ส่วนฮาร์ดแวร์ (Hardware)

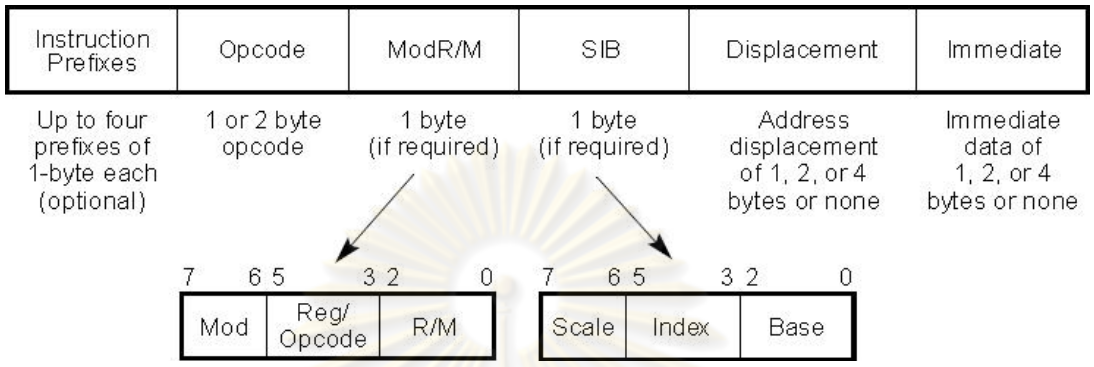
ส่วนนี้จะดัดแปรหน่วยประมวลผลกลาง (Modify CPU) โดยเพิ่มคำสั่งที่เกี่ยวข้องกับการอ่านค่าจากหน่วยความจำ ในหน่วยประมวลผลกลาง (CPU) ให้สามารถทำงาน 2 อย่างพร้อมกัน ได้แก่ การอ่านค่าข้อมูล และตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของข้อมูลที่อ่านด้วย จากการสังเกตโค้ดแอสเซมบลี (Assembly code) ของโปรแกรมต่างๆ พบว่า มีคำสั่ง 4 คำสั่งที่เกี่ยวข้อง ได้แก่

1. คำสั่งบรรจุ (Load Instruction) ค่าจากหน่วยความจำมายังเรจีสเตอร์ (Register)
2. คำสั่งเปรียบเทียบ (Compare Instruction) ค่าในหน่วยความจำกับค่าในเรจีสเตอร์
3. คำสั่งเปรียบเทียบ (Compare Instruction) ค่าในเรจีสเตอร์กับค่าในหน่วยความจำ
4. คำสั่งเปรียบเทียบ (Compare Instruction) ค่าที่ใช้ทันที (Immediate Value) กับค่าในหน่วยความจำ

คำสั่งใหม่ 4 คำสั่งนี้ทำงานเหมือนคำสั่งเดิมแต่มีการตรวจสอบเพิ่มโค้ดการตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของข้อมูลที่อ่านจากหน่วยความจำ วิธีนี้จะช่วยลดเวลาในการทำงานลงจากวิธีในโครงงานทางวิศวกรรม (Senior Project) ในระดับปริญญาตรี เนื่องจากค่าใช้จ่ายอื่น (Overhead) จะถูกฝังลงในระดับฮาร์ดแวร์แทนแบบเดิมที่ค่าใช้จ่ายอื่นเกิดขึ้นในระดับซอฟต์แวร์ โดยขั้นตอนการตรวจสอบมีสามขั้นตอน ดังต่อไปนี้

4.2.1.1 **ตรวจสอบรูปแบบการกำหนดเลขที่อยู่** (Addressing Form)

รูปแบบคำสั่ง (Instruction Format) ของสถาปัตยกรรมของอินเทล (Intel) 386 นั้นมีไบต์ชื่อ MODR/M และ SIB สำหรับระบุรูปแบบการกำหนดเลขที่อยู่และข้อมูลที่ใช้ในการคำนวณตำแหน่งแอดเดรสของซีเคียวคานารีเวิร์ด ดังรูปที่ 6



รูปที่ 6 รูปแบบคำสั่ง (Instruction Format) ของสถาปัตยกรรมของอินเทล (Intel) 386

ในขั้นตอนนี้ได้เตรียมข้อมูลไว้เพื่อใช้ในขั้นตอนต่อไป

4.2.1.2 **คำนวณค่าแอดเดรสของซีเคียวคานารีเวิร์ดที่อยู่ติดกับข้อมูลที่ต้องการอ่านจากหน่วยความจำ**

แม้ว่าซีเคียวคานารีเวิร์ดจะอยู่ติดกับข้อมูลที่ต้องการอ่านจากหน่วยความจำ แต่มันอาจอยู่ตำแหน่งที่สูงกว่าหรือต่ำกว่าข้อมูลนั้นได้ ดังนั้น เมื่อทราบรูปแบบการกำหนดเลขที่อยู่แล้ว ก็สามารถคำนวณตำแหน่งคานารีเวิร์ดโดยใช้ค่าในเขตข้อมูล (Field) MOD และ R/M รวมทั้งเขตข้อมูลการเคลื่อน (Displacement Field) ได้ ถ้าค่าในเขตข้อมูล MOD เป็น 00 และ R/M เป็น 100 ในเลขฐานสอง แสดงว่าข้อมูลที่ต้องการอ่านเป็นตัวแปรชนิดอาร์เรย์ (Array) สมมติเลขที่อยู่ฐาน (Base Address) มีค่าเท่ากับ A ตำแหน่งแอดเดรสของซีเคียวคานารีเวิร์ดจึงมีค่าเท่ากับ A-1 แต่ถ้าเขตข้อมูล MOD และ R/M มีค่าเป็นอย่างอื่น แสดงว่าข้อมูลที่ต้องการอ่านเป็นตัวแปรทั่วไป (Variable) สมมติตำแหน่งแอดเดรสของตัวแปรนั้นมีค่าเท่ากับ X ถ้าค่าในเขตข้อมูลการเคลื่อนเป็นบวก แสดงว่าอยู่ภายในฮีป (Heap) ของโพรเซส (Process) ตำแหน่งแอดเดรสของซีเคียวคานารีเวิร์ดจะมีค่าเท่ากับ X+4 แต่ถ้าไม่ใช่ แสดงว่าอยู่ภายในสแต็ก (Stack) ของโพรเซส ตำแหน่งแอดเดรสของซีเคียวคานารีเวิร์ดจะมีค่าเท่ากับ X-1

4.2.1.3 **ตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ด**

ค่าซีเคียวบิตของคานารีเวิร์ดจะถูกตรวจสอบ โดยถ้ามีค่าเป็น 0 หน่วยประมวลผลกลางจะทำงานต่อไปตามปกติ แต่ถ้ามีค่าเป็น 1 จะเกิดสิ่งผิดปรกติ (Exception) นั่นคือ Segmentation Fault และทำให้การทำงานของโปรแกรมนั้นสิ้นสุดลงทันที

แนวทางในการทำให้เกิดผล (Implement) จริงในทางฮาร์ดแวร์สามารถทำได้ [60] แต่ในการวิจัยนี้จะดัดแปรบนซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์ (BOCHS Emulator) [58] โดยเลือกใช้รหัสคำสั่ง (Operation Code) เป็น 0x0F1B, 0x0F1C, 0x0F1D และ 0x0F1E ที่ยังว่างอยู่ กำหนดรหัสคำสั่งสำหรับแต่ละคำสั่งดังตารางที่ 12 และแก้ไขโค้ดในไฟล์ (File) ที่เกี่ยวข้องกับการทำงานของหน่วยประมวลผลกลาง ซึ่งอยู่ในไดเร็กทอรี (Directory) ชื่อ cpu ดังนี้

ตารางที่ 12 ตารางสรุปรหัสคำสั่งเดิมและรหัสคำสั่งใหม่

รหัสคำสั่ง		คำสั่ง	คำอธิบาย
เดิม	ใหม่		
0x8B	0x0F1B	MOV	คำสั่งบรรจุค่าจากหน่วยความจำมายังเรจีสเตอร์
0x3B	0x0F1C	CMP	คำสั่งเปรียบเทียบค่าในหน่วยความจำกับค่าในเรจีสเตอร์
0x39	0x0F1D	CMP	คำสั่งเปรียบเทียบค่าในเรจีสเตอร์กับค่าในหน่วยความจำ
0x83	0x0F1E	CMPL	คำสั่งเปรียบเทียบค่าที่ใช้ทันทีกับค่าในหน่วยความจำ

ในไฟล์ชื่อ data_xfer32.cc ที่ท้ายไฟล์ สร้างฟังก์ชันใหม่สำหรับคำสั่งบรรจุค่าจากหน่วยความจำมายังเรจีสเตอร์ โดยคัดลอกการทำงานของฟังก์ชันเดิมของคำสั่งนี้ แล้วเพิ่มโค้ดการตรวจสอบตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของข้อมูลที่อ่านจากหน่วยความจำ (โค้ดส่วนที่เป็นตัวหนา) ดังนี้

```
  void
BX_CPU_C::MOV_GdEEd_CHKSBIT(bxInstruction_c *i)
{
  // 2nd modRM operand Ex, is known to be a memory operand, Ed.
  Bit32u op2_32;
  // sbit by KPR
  int sbit;
  bx_address offset=RMAddr(i);

  read_virtual_dword(i->seg(), offset, &op2_32, &sbit);
      sbit=(sbit!=0)?1:0;
      sbit=(BX_CPU_SBIT_MODE)? 1:sbit;

  BX_WRITE_32BIT_REGZ(i->nnn(), op2_32);
        BX_CPU_RSBIT[i->nnn()]=sbit;

  Bit32u can;
  int canSBit;
  if((i->mod()==0)&&(i->rm()==4)&&(i->sibScale()==0)){
    //For check base of array
      switch(i->sibBase()){
        case 0 : offset = EAX; break;
        case 1 : offset = ECX; break;
        case 2 : offset = EDX; break;
```

```
        case 3 : offset = EBX; break;
        case 4 : offset = ESP; break;
        case 5 : offset = i->displ32u();break;
        case 6 : offset = ESI; break;
        case 7 : offset = EDI; break;
    }
      read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }else{
      int displ=i->displ32u();
      if(displ>0)
        read_virtual_dword(i->seg(), offset+4, &can, &canSBit);
      else
        read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }
  if(canSBit != 0)
    exception(BX_GP_EXCEPTION, 0, 0);
}
```

ในไฟล์ชื่อ arith32.cc ที่ท้ายไฟล์ สร้างฟังก์ชันใหม่สำหรับคำสั่งเปรียบเทียบค่าในหน่วยความจำกับค่าในเรจีสเตอร์ โดยคัดลอกการทำงานของฟังก์ชันเดิมของคำสั่งนี้ แล้วเพิ่มโค้ดการตรวจสอบตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของข้อมูลที่อ่านจากหน่วยความจำ (โค้ดส่วนที่เป็นตัวหนา) ดังนี้

```
  void
BX_CPU_C::CMP_GdEd_CHKSBIT(bxInstruction_c *i)
{
  Bit32u op1_32, op2_32;
  bx_address offset=RMAddr(i);

  op1_32 = BX_READ_32BIT_REG(i->nnn());

  if (i->modC0()) {
    op2_32 = BX_READ_32BIT_REG(i->rm());
    }
  else {
    read_virtual_dword(i->seg(), offset, &op2_32);
    }

#if (defined(__i386__) && defined(__GNUC__) && BX_SupportHostAsms)
  Bit32u flags32;

  asmCmp32(op1_32, op2_32, flags32);
  setEFlagsOSZAPC(flags32);
#else
  Bit32u diff_32;
  diff_32 = op1_32 - op2_32;

  SET_FLAGS_OSZAPC_32(op1_32, op2_32, diff_32, BX_INSTR_CMP32);
#endif

  Bit32u can;
  int canSBit;
  if((i->mod()==0)&&(i->rm()==4)&&(i->sibScale()==0)){
    //For check base of array
      switch(i->sibBase()){
        case 0 : offset = EAX; break;
```

```
        case 1 : offset = ECX; break;
        case 2 : offset = EDX; break;
        case 3 : offset = EBX; break;
        case 4 : offset = ESP; break;
        case 5 : offset = i->displ32u(); break;
        case 6 : offset = ESI; break;
        case 7 : offset = EDI; break;
    }
      read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }else{
      int displ=i->displ32u();
      if(displ>0)
        read_virtual_dword(i->seg(), offset+4, &can, &canSBit);
      else
        read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }
  if(canSBit != 0)
    exception(BX_GP_EXCEPTION, 0, 0);
}
```

ในไฟล์ชื่อ arith32.cc ที่ท้ายไฟล์ สร้างฟังก์ชันใหม่สำหรับคำสั่งเปรียบเทียบค่าในเรจีสเตอร์กับค่าในหน่วยความจำ โดยคัดลอกการทำงานของฟังก์ชันเดิมของคำสั่งนี้ แล้วเพิ่มโค้ดการตรวจสอบตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของข้อมูลที่อ่านจากหน่วยความจำ (โค้ดส่วนที่เป็นตัวหนา) ดังนี้

```
  void
BX_CPU_C::CMP_EdGd_CHKSBIT(bxInstruction_c *i)
{
  Bit32u op2_32, op1_32;
  bx_address offset=RMAddr(i);

  op2_32 = BX_READ_32BIT_REG(i->nnn());

  if (i->modC0()) {
    op1_32 = BX_READ_32BIT_REG(i->rm());
    }
  else {
    read_virtual_dword(i->seg(), offset, &op1_32);
    }

#if (defined(__i386__) && defined(__GNUC__) && BX_SupportHostAsms)
  Bit32u flags32;

  asmCmp32(op1_32, op2_32, flags32);
  setEFlagsOSZAPC(flags32);
#else
  Bit32u diff_32;
  diff_32 = op1_32 - op2_32;

  SET_FLAGS_OSZAPC_32(op1_32, op2_32, diff_32, BX_INSTR_CMP32);
#endif

  Bit32u can;
  int canSBit;
  if((i->mod()==0)&&(i->rm()==4)&&(i->sibScale()==0)){
    //For check base of array
```

```
      switch(i->sibBase()){
        case 0 : offset = EAX; break;
        case 1 : offset = ECX; break;
        case 2 : offset = EDX; break;
        case 3 : offset = EBX; break;
        case 4 : offset = ESP; break;
        case 5 : offset = i->displ32u(); break;
        case 6 : offset = ESI; break;
        case 7 : offset = EDI; break;
    }
      read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }else{
      int displ=i->displ32u();
      if(displ>0)
        read_virtual_dword(i->seg(), offset+4, &can, &canSBit);
      else
        read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }
  if(canSBit != 0)
    exception(BX_GP_EXCEPTION, 0, 0);
}
```

ในไฟล์ชื่อ arith32.cc ที่ท้ายไฟล์ สร้างฟังก์ชันใหม่สำหรับคำสั่งเปรียบเทียบค่าที่ใช้ทันทีกับค่าในหน่วยความจำ โดยคัดลอกการทำงานของฟังก์ชันเดิมของคำสั่งนี้ แล้วเพิ่มโค้ดการตรวจสอบตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของข้อมูลที่อ่านจากหน่วยความจำ (โค้ดส่วนที่เป็นตัวหนา) ดังนี้

```
  void
BX_CPU_C::CMP_EdId_CHKSBIT(bxInstruction_c *i)
{
  Bit32u op2_32, op1_32;
  bx_address offset=RMAddr(i);

  op2_32 = i->Id();

  if (i->modC0()) {
    op1_32 = BX_READ_32BIT_REG(i->rm());
    }
  else {
    read_virtual_dword(i->seg(), offset, &op1_32);
    }

#if (defined(__i386__) && defined(__GNUC__) && BX_SupportHostAsms)
  Bit32u flags32;

  asmCmp32(op1_32, op2_32, flags32);
  setEFlagsOSZAPC(flags32);
#else
  Bit32u diff_32;
  diff_32 = op1_32 - op2_32;

  SET_FLAGS_OSZAPC_32(op1_32, op2_32, diff_32, BX_INSTR_CMP32);
#endif

  Bit32u can;
  int canSBit;
```

```
  if((i->mod()==0)&&(i->rm()==4)&&(i->sibScale()==0)){
    //For check base of array
      switch(i->sibBase()){
        case 0 : offset = EAX; break;
        case 1 : offset = ECX; break;
        case 2 : offset = EDX; break;
        case 3 : offset = EBX; break;
        case 4 : offset = ESP; break;
        case 5 : offset = i->displ32u(); break;
        case 6 : offset = ESI; break;
        case 7 : offset = EDI; break;
    }
      read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }else{
      int displ=i->displ32u();
      if(displ>0)
        read_virtual_dword(i->seg(), offset+4, &can, &canSBit);
      else
        read_virtual_dword(i->seg(), offset-1, &can, &canSBit);
  }
  if(canSBit != 0)
    exception(BX_GP_EXCEPTION, 0, 0);
}
```

ในไฟล์ชื่อ cpu.h ประมาณบรรทัดที่ 2127 เพิ่มโค้ดเพื่อประกาศชื่อฟังก์ชันใหม่ที่สร้างไว้ข้างต้น ดังนี้

```
BX_SMF void MOV_GdEEd_CHKSBIT(bxInstruction_c *i);
BX_SMF void CMP_GdEd_CHKSBIT(bxInstruction_c *i);
BX_SMF void CMP_EdGd_CHKSBIT(bxInstruction_c *i);
BX_SMF void CMP_EdId_CHKSBIT(bxInstruction_c *i);
```

ในไฟล์ชื่อ fetchdecode.cc ประมาณบรรทัดที่ 1303 แก้ไขโค้ดเพื่อให้รหัสคำสั่งที่ยังว่างอยู่ทั้ง 4 รหัส เรียกใช้ฟังก์ชันที่ประกาศไว้ ดังนี้

```
/* 0F 1B */  { BxAnother, &BX_CPU_C::MOV_GdEEd_CHKSBIT },
/* 0F 1C */  { BxAnother, &BX_CPU_C::CMP_GdEd_CHKSBIT },
/* 0F 1D */  { BxAnother, &BX_CPU_C::CMP_EdGd_CHKSBIT },
/* 0F 1E */  { BxAnother | BxImmediate_Ib_SE, &BX_CPU_C::CMP_EdId_CHKSBIT },
```

เมื่อแก้ไขโค้ดเสร็จเรียบร้อย จึงสั่งสร้าง (Make) ซอฟต์แวร์นี้ขึ้นมาใหม่ (รายละเอียดของขั้นตอนนี้อยู่ในภาคผนวก จ. หน้า 114)

4.2.2 **ส่วนซอฟต์แวร์** (Software)

ส่วนนี้จะดัดแปรโปรแกรมที่ต้องการให้นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลเพื่อให้เรียกใช้คำสั่งของหน่วยประมวลผลกลาง (CPU instruction) ใหม่ที่สร้างขึ้นแทนคำสั่งเดิมดังตารางที่ 12 แต่เนื่องจากไม่สามารถเขียนโค้ดแอสเซมบลี (Assembly Code) ได้โดยตรง จึงต้องแปลงมาจากโค้ดภาษาซี

ดังนั้น วิธีการแปลงให้โค้ดภาษาซีกลายเป็นภาษาแอสเซมบลีที่มีรหัสคำสั่งและโค้ดภาษาซีแทรกอยู่ด้วย ต้องสั่งคอมไพล์ตามลำดับ ดังนี้

```
# gcc –g –o stack stack.c
# objdump -d -S stack > stack.s
```

จากคำสั่งข้างต้นคำสั่งแรกเป็นการส่งคอมไพล์โดยใส่ออปชัน (Option) -g เพื่อแทรกข้อมูลที่ใช้ในการแก้จุดบกพร่อง (Debug) ลงในโค้ดแอสเซมบลี และออปชัน -o ตามด้วยชื่อของไฟล์ที่สามารถทำงาน (Execute) ได้ ต่อมาคำสั่งที่ 2 เป็นคำสั่งให้แสดงข้อมูลจากไฟล์อ็อบเจกต์ (Object File) นั้นโดยใส่ออปชัน -d เพื่อให้ถอดแยกภาษาแอสเซมบลี (Disassemble) และ -S เพื่อแสดงโค้ดต้นฉบับ (Source Code) ด้วย แล้วต่อด้วยไปป์ (Pipe) เพื่อให้ผลลัพธ์ถูกบันทึกอยู่ในไฟล์แทนการแสดงออกทางหน้าจอ เมื่อทำวิธีนี้แล้ว ทำให้สังเกตเห็นว่า โปรแกรมนั้นเมื่อเขียนด้วยภาษาแอสเซมบลีแล้วจะมีโค้ดอย่างไรบ้าง เพื่อนำไปใช้ในการดัดแปรโปรแกรมนั้นให้นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผล โดยใช้อินไลน์แอสเซมบลี (In-line Assembly) ในการเรียกใช้คำสั่งของหน่วยประมวลผลกลางใหม่ที่สร้างขึ้นแทน (โค้ดของโปรแกรมที่ใช้ในการวิจัยนี้สามารถดูได้ที่ภาคผนวก ข. หน้า 64 และภาคผนวก ค. หน้า 68)

บทที่ 5
วิธีประเมินการวิจัย สรุปและอภิปรายผล

ในบทนี้จะกล่าวถึงวิธีประเมินการวิจัย สรุป อภิปราย และวิเคราะห์ผลการวิจัย ดังนี้

5.1 วิธีประเมินการวิจัย

การวิจัยนี้ได้กำหนดวิธีประเมินการวิจัยในสองแง่มุม ได้แก่

5.1.1 การทดสอบความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่าง ๆ

การทดลองในการวิจัยนี้จะทดสอบความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆที่สนใจสี่รูปแบบ ได้แก่

1. สแต็กโอเวอร์โฟลว์ (Stack Overflows) ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data)
2. สแต็กโอเวอร์โฟลว์ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data)
3. อาร์เรย์อินเด็กซิงเออเรอร์ (Array Indexing Errors) ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่เป็นตัวควบคุมระบบ
4. อาร์เรย์อินเด็กซิงเออเรอร์ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ

โดยตัวอย่างโปรแกรมในหัวข้อ 2.1.2.1.1 2 หน้า 8 จะใช้ทดสอบรูปแบบการโจมที่รูปแบบที่ 1 และตัวอย่างโปรแกรมในหัวข้อ 2.1.2.3 หน้า 14 จะใช้ทดสอบรูปแบบการโจมที่รูปแบบที่ 3 เนื่องจากไม่มีการดัดแปรโปรแกรมเพื่อรองรับการโจมตีเพิ่มเติม

ส่วนการโจมที่รูปแบบที่ 2 และ 4 นั้น จะต้องเขียนโปรแกรมทดสอบขึ้นมาใหม่เพื่อเรียกใช้คำสั่งของหน่วยประมวลผลกลาง (CPU instruction) ใหม่ที่สร้างขึ้นมา โดยแต่ละรูปแบบจะถูกทดสอบด้วยโปรแกรมสองโปรแกรม โปรแกรมละหนึ่งคำสั่ง คือ คำสั่งบรรจุ (Load Instruction) และคำสั่งเปรียบเทียบ (Compare Instruction) ในหัวข้อ 4.2.1 หน้า 30 เนื่องจากคำสั่งใหม่ทุกคำสั่งนี้มีขั้นตอนการตรวจสอบค่าซีเคียวบิต (Secure Bit) ของคานารีเวิร์ด (Canary Word) ของข้อมูลที่อ่านเหมือนกัน ในการทดลองจึงเลือกเพียงสองคำสั่งก็เพียงพอแล้ว แต่ละโปรแกรมมีสองแบบ ได้แก่ แบบดั้งเดิม และแบบที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด (Secure Canary Word) มาทำให้เกิดผลแล้ว (โค้ดของโปรแกรมทั้งหมดอยู่ในภาคผนวก ข. หน้า 64)

5.1.2 การทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ

การทดลองในการวิจัยนี้จัดทำขึ้นบนเครื่องโน้ตบุ๊คคอมพิวเตอร์ (ดังหัวข้อ 4.1.5 หน้า 30) โดยเขียนโปรแกรมภาษาซีเพื่อใช้ทดลองประสิทธิภาพสามโปรแกรม ได้แก่ โปรแกรมเรียงลำดับแบบฟอง (Bubble Sort) โปรแกรมควิกซอร์ต (Quick Sort) และโปรแกรมค้นหาแบบทวิภาค (Binary Search) ด้วยต้นไม้แบบเอวีแอล (AVL Tree) เนื่องจากโปรแกรมทั้งสามนี้สามารถใช้เป็นตัวแทนของโปรแกรมทั่วไปได้ โดยโปรแกรมเรียงลำดับแบบฟองเป็นตัวแทนของโปรแกรมที่มีตัวแปรต่างๆรวมทั้งตัวแปรชนิดอาร์เรย์ (Array) อยู่ภายในสแต็ก (Stack) ของโพรเซส (Process) โปรแกรมควิกซอร์ตเป็นตัวแทนของโปรแกรมที่มีตัวแปรต่างๆอยู่ภายในสแต็กของโพรเซสเช่นกัน และมีการใช้ฟังก์ชันเรียกซ้ำ (Recursive Function) ส่วนโปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอลเป็นตัวแทนของโปรแกรมที่มีตัวแปรชนิดโครงสร้าง (Structure) อยู่ภายในฮีป (Heap) ของโพรเซส เพราะเรียกใช้ฟังก์ชัน malloc ในการจองหน่วยความจำ

แต่ละโปรแกรมจะมีสามแบบ ได้แก่ 1) แบบดั้งเดิมที่เขียนด้วยภาษาซีและไม่ได้ใช้อินไลน์แอสเซมบลี (In-line Assembly) 2) แบบที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบเก่า ซึ่งเป็นแบบที่ใช้ในโครงงานทางวิศวกรรม (Senior Project) ในระดับปริญญาตรี (วิธีนี้อธิบายอยู่ในภาคผนวก ก. หน้า 60) และ 3) แบบที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่ ซึ่งเป็นแบบที่ใช้ในการวิจัยนี้ โดยประกาศคานารีเวิร์ดคั่นระหว่างตัวแปรภายในโปรแกรม และแทนที่โค้ดในส่วนหลักของโปรแกรม โดยใช้อินไลน์แอสเซมบลี เพื่อให้สามารถเรียกใช้รหัสคำสั่งใหม่ที่สร้างขึ้นมาทดแทนในหัวข้อ 4.2.1 หน้า 30 ได้ (โค้ดของโปรแกรมทั้งหมดอยู่ในภาคผนวก ค. หน้า 68)

หลังจากนั้น ทดลองวัดประสิทธิภาพของโปรแกรม เพื่อนำมาเปรียบเทียบโปรแกรมเดียวกันระหว่างสองแบบ โดยคอมไพล์ (Compile) ด้วยโปรแกรมจีซีซี (GNU project C and C++: GCC) [59] ซึ่งไม่ใช้ออปชัน (Option) ใดๆเลย รวมทั้งการออปทิไมเซชัน (Optimization) ในการวิจัยนี้ได้ทดลองวัดประสิทธิภาพทั้งหมดสามด้าน ได้แก่

1. ด้านความเร็วของการทำงาน โดยใช้คำสั่ง time ของระบบปฏิบัติการในการจับเวลาซึ่งเป็นเวลานาฬิกา (Wall-clock Time) ในที่นี้ได้ทดลองทั้งหมด 10 ครั้ง แล้วหาค่าเฉลี่ย และหักลบด้วยค่าใช้จ่ายอื่น (Overhead) ที่เกี่ยวข้องกับการอ่านข้อมูลจากไฟล์เข้ามาในหน่วยความจำทั้งหมด 1,000 ข้อมูล (ข้อมูลที่นำมาทดลองอยู่ในภาคผนวก ง. หน้า 112) ด้วย สำหรับโปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอลนั้นได้ทดลองทั้งกรณีที่ค้นหาข้อมูลพบและกรณีที่ค้นหาข้อมูลไม่พบ

2. ด้านขนาดของหน่วยความจำที่ใช้ในขณะที่โปรแกรมทำงาน โดยดูจากขนาดของหน่วยความจำเสมือน (VmSize: Virtual Memory Size) ในไฟล์ชื่อ status ซึ่งอยู่ในไดเร็กทอรี (Directory) ชื่อ /proc/pid โดย pid (Process ID) คือรหัสประจำตัวของโพรเซสของโปรแกรมนั้น ดังนั้นจึงต้องแทรกโค้ด scanf ก่อนบรรทัด return 0; เพื่อให้โปรแกรมยังไม่จบการทำงานทันทีเมื่อสั่งให้เป็นการประมวลผลส่วนหลัง (Background Processing) โดยใส่ & ไว้ท้ายสุดหลังคำสั่งที่สั่งให้โปรแกรมทำงาน แต่ในที่นี้จะไม่หักลบด้วยค่าใช้จ่ายอื่นที่เกี่ยวข้องกับการอ่านข้อมูลจากไฟล์ เหมือนกับการวัดความเร็วของการทำงาน เนื่องจากไม่สามารถกระทำได้โดยตรง
3. ด้านขนาดของโปรแกรม การทดลองนี้จะลบโค้ดที่เกี่ยวข้องกับการอ่านข้อมูลจากไฟล์ก่อน แล้วจึงคอมไพล์ใหม่ จากนั้นจะใช้คำสั่ง ls -al ของระบบปฏิบัติการในการวัดขนาด

5.2 สรุปผลการวิจัย

เนื่องจากการวิจัยนี้ได้ทดลองในสองแง่มุม และสรุปผลได้ดังนี้

5.2.1 ผลการทดสอบความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่าง ๆ

จากการทดสอบในหัวข้อ 5.1.1 หน้า 38 สามารถสรุปผลได้ดังตารางที่ 13

ตารางที่ 13 ผลการทดสอบความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆ

รูปแบบการโจมตี	ต้องการเปลี่ยนแปลงส่วนที่	
	เป็นตัวควบคุมระบบ	ไม่เป็นตัวควบคุมระบบ
สแต็กโอเวอร์โฟลว์	ป้องกันได้	ป้องกันได้
อาร์เรย์อินเด็กซิงเออเรอร์	ป้องกันได้	ป้องกันไม่ได้

5.2.2 ผลการวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่าง ๆ

จากการทดลองในหัวข้อ 5.1.2 หน้า 39 สามารถนำผลลัพธ์มาเปรียบเทียบระหว่างโปรแกรมทั้งสามแบบ และคำนวณผลต่างของผลลัพธ์เป็นหน่วยร้อยละ โดยผลต่างในด้านความเร็วของการทำงานคำนวณจากกฎของอัมดาห์ล (Amdahl's Law) [61] นั่นคือ ผลต่าง = $\left(1 - \frac{1}{\text{Speedup}}\right) \times 100$ โดยสปีดอัพ (Speedup) = $\frac{t1}{t0}$ ซึ่ง t0 คือ เวลาที่ใช้ในการทำงานของโปรแกรมแบบดั้งเดิม และ t1 คือเวลาที่ใช้ในการทำงานของโปรแกรมที่นำวิธีการป้องกัน

บัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้ว ส่วนผลต่างในอีกสองด้านนั้น คำนวณจาก ผลต่าง = $\frac{s1-s0}{s0} \times 100$ โดย s0 คือ ขนาดของโปรแกรมแบบดั้งเดิม และ s1 คือ ขนาดของโปรแกรมที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้ว ดังนั้น สามารถสรุปผลได้ดังตารางที่ 14 ตารางที่ 15 และตารางที่ 16

ตารางที่ 14 ประสิทธิภาพของโปรแกรมในด้านความเร็วของการทำงาน

ตัวอย่างโปรแกรม	**เวลาที่ใช้ในการทำงาน (วินาที)**		
	ดั้งเดิม	**เก่า**	**ใหม่**
โปรแกรมเรียงลำดับแบบฟอง	2.106	27.885	2.316
ผลต่าง (ร้อยละ)	-	-92.448	-9.067
โปรแกรมควิกซอร์ต	0.029	0.574	0.064
ผลต่าง (ร้อยละ)	-	-94.948	-54.688
โปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอล	0.359	1.5755	0.4725
ผลต่าง (ร้อยละ)	-	-77.214	-24.021

ตารางที่ 15 ประสิทธิภาพของโปรแกรมในด้านขนาดของหน่วยความจำที่ใช้ในขณะที่โปรแกรมทำงาน

ตัวอย่างโปรแกรม	**ขนาดของหน่วยความจำ (กิโลไบต์)**		
	ดั้งเดิม	**เก่า**	**ใหม่**
โปรแกรมเรียงลำดับแบบฟอง	1,084	1,080	1,084
ผลต่าง (ร้อยละ)	-	-0.369	0
โปรแกรมควิกซอร์ต	1,084	1,084	1,084
ผลต่าง (ร้อยละ)	-	0	0
โปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอล	1,100	1,112	1,112
ผลต่าง (ร้อยละ)	-	+1.091	+1.091

ตารางที่ 16 ประสิทธิภาพของโปรแกรมในด้านขนาดของโปรแกรม

ตัวอย่างโปรแกรม	**ขนาดของโปรแกรม (วินาที)**		
	ดั้งเดิม	**เก่า**	**ใหม่**
โปรแกรมเรียงลำดับแบบฟอง	11,910	12,705	12,109
ผลต่าง (ร้อยละ)	-	+6.675	+1.671
โปรแกรมควิกซอร์ต	12,137	13,044	12,536
ผลต่าง (ร้อยละ)	-	+7.473	+3.287
โปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอล	13,033	14,340	13,980
ผลต่าง (ร้อยละ)	-	+10.028	+7.266

นอกจากนี้ ยังได้ทดลองคอมไพล์โดยใช้ออปชันต่างๆสามแบบ ได้แก่

1. ใส่ออปชันการออปทิไมเซชันเพื่อลดขนาดของโค้ด –Os
2. ใส่ออปชันการออปทิไมเซชันเพื่อลดขนาดของโค้ดและการทำงาน ในระดับที่ 3 ซึ่งรวมทั้งออปชันการทำอินไลน์ฟังกชัน (-finline-functions) ด้วย -O3
3. ใส่ออปชันการออปทิไมเซชันเพื่อลดขนาดของโค้ดและการทำงาน ในระดับที่ 3 ซึ่งรวมทั้งออปชันการทำอินไลน์ฟังกชัน (-finline-functions) ด้วย -O3 และใส่ออปชันเพื่อแทรกข้อมูลที่ใช้ในการแก้จุดบกพร่อง (Debug) ลงในโค้ดแอสเซมบลี (Assembly Code) -g

ดังนั้น สามารถสรุปผลได้ดังตารางที่ 17 ตารางที่ 18 และตารางที่ 19

ตารางที่ 17 ขนาดของโปรแกรมเรียงลำดับแบบฟองโดยใช้ออปชันต่างกัน

ออปชันที่ใช้	ขนาดของโปรแกรม (ไบต์)		
	ดั้งเดิม	เก่า	ใหม่
ไม่ใช้	11,910	12,705	12,109
ผลต่าง (ร้อยละ)	-	+6.675	+1.671
แบบที่ 1	11,638	12,545	12,093
ผลต่าง (ร้อยละ)	-	+7.793	+3.910
แบบที่ 2	11,638	12,545	12,093
ผลต่าง (ร้อยละ)	-	+7.793	+3.910
แบบที่ 3	16,086	17,321	16,625
ผลต่าง (ร้อยละ)	-	+7.677	+3.351

ตารางที่ 18 ขนาดของโปรแกรมควิกซอร์ตโดยใช้ออปชันต่างกัน

ออปชันที่ใช้	ขนาดของโปรแกรม (ไบต์)		
	ดั้งเดิม	เก่า	ใหม่
ไม่ใช้	12,137	13,044	12,536
ผลต่าง (ร้อยละ)	-	+7.473	+3.287
แบบที่ 1	11,961	12,756	12,520
ผลต่าง (ร้อยละ)	-	+6.647	+4.674
แบบที่ 2	11,945	12,852	12,520
ผลต่าง (ร้อยละ)	-	+7.593	+4.814
แบบที่ 3	17,237	18,460	17,412
ผลต่าง (ร้อยละ)	-	+7.095	+1.015

ตารางที่ 19 ขนาดของโปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอลโดยใช้ออปชันต่างกัน

ออปชันที่ใช้	ขนาดของโปรแกรม (ไบต์)		
	ดั้งเดิม	เก่า	ใหม่
ไม่ใช้	13,033	14,340	13,980
ผลต่าง (ร้อยละ)	-	+10.028	+7.266
แบบที่ 1	12,601	13,764	13,932
ผลต่าง (ร้อยละ)	-	+9.229	+10.563
แบบที่ 2	13,017	14,980	13,932
ผลต่าง (ร้อยละ)	-	+15.080	+7.029
แบบที่ 3	25,717	28,548	23,036
ผลต่าง (ร้อยละ)	-	+11.008	-10.425

5.3 อภิปรายผลการวิจัย

การวิจัยนี้จะอภิปรายผลการวิจัย โดยแบ่งตามวิธีประเมินการวิจัย ดังนี้

5.3.1 ความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์

วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดสามารถป้องกันการโจมตีที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่เป็นตัวควบคุมระบบ (Control Data) ทั้งที่เป็นสแต็กโอเวอร์โฟลว์ (Stack Overflows) และอาร์เรย์อินเด็กซิงเออเรอร์ (Array Indexing Errors) ได้ เนื่องจากวิธีนี้ยังคงความสามารถในการป้องกันของซีเคียวบิต (Secure Bit) ที่เน้นป้องกันการโจมตีที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่เป็นตัวควบคุมระบบได้เหมือนเดิม

นอกจากนี้ วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบคานารีเวิร์ด (Canary Word) เพียงอย่างเดียวจะสามารถป้องกันการโจมตีที่มีรูปแบบเป็นสแต็กโอเวอร์โฟลว์ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่เป็นตัวควบคุมระบบหรือส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) ได้เพียงบางส่วนเท่านั้น แต่วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดนั้นสามารถป้องกันการโจมตีที่มีรูปแบบนี้ทั้งสองส่วนนี้ได้อย่างสมบูรณ์ เพราะอาศัยกลไกของซีเคียวบิตของคานารีเวิร์ด ซึ่งทำหน้าที่เปรียบเสมือนตัวควบคุมระบบที่ปกป้องตัวแปรภายในโปรแกรมอีกชั้นหนึ่ง

แต่อย่างไรก็ตาม วิธีนี้ไม่สามารถป้องกันการโจมตีที่มีรูปแบบเป็นอาร์เรย์อินเด็กซิงเออเรอร์ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบได้ เนื่องจากอาร์เรย์อินเด็กซิงเออเรอร์เป็นรูปแบบที่ไม่ได้อาศัยการล้นของข้อมูลในบัฟเฟอร์ไปทับข้อมูลส่วนอื่นๆ ทำให้สามารถข้ามไปเขียนทับข้อมูลในตำแหน่งใดๆในหน่วยความจำได้โดยไม่ต้องเขียนข้อมูลผ่านคานารีเวิร์ดที่นำมาคั่นไว้ ยิ่งไปกว่านั้น ซีเคียวบิตก็ไม่สามารถตรวจสอบการคำนวณค่าพอยท์เตอร์ (Pointer Arithmetic) ภายในระบบได้ เพราะไม่ได้ถือว่าเป็นข้อมูลนำเข้า

5.3.2 ประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่าง ๆ

จากการทดลองพบว่า โปรแกรมที่นำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่ใช้เวลาในการทำงานลดลงกว่าโปรแกรมที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบเก่าอย่างมาก โดยในที่นี้จะพิจารณาเวลานาฬิกาเท่านั้น มิได้พิจารณาในแง่รอบคำสั่งเครื่อง (Instruction Cycle) เนื่องจากต้องคำนึงถึงการออกแบบหน่วยประมวลผลกลางใหม่ทั้งหมดที่รองรับซีเคียวคานารีเวิร์ดนี้ ซึ่งอยู่นอกขอบเขตของการวิจัยนี้ แต่อย่างไรก็ตาม การหน่วงที่เกิดขึ้นอาจถูกซ่อนด้วยการทำงานแบบสายท่อ (Pipeline) ของหน่วยประมวลผลกลางได้

ขนาดของหน่วยความจำที่ใช้ในขณะที่โปรแกรมทำงานของโปรแกรมแบบดั้งเดิมและโปรแกรมที่นำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วเกือบจะเท่ากัน เนื่องจากโปรแกรมจะจองหน่วยความจำเป็นเพจ (Page) ซึ่งมีหน่วยเป็นกิโลไบต์ (Kilobyte) จึงทำให้ไม่เห็นความแตกต่างอย่างชัดเจน ยกเว้นโปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอลที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วทั้งแบบเก่าและแบบใหม่ จะมีขนาดใหญ่กว่าแบบดั้งเดิมเล็กน้อย เนื่องจากต้องจองหน่วยความจำเพิ่มสำหรับคานารีเวิร์ดเป็นจำนวนมากขึ้นกับจำนวนข้อมูล โดยสังเกตได้จากขนาดของหน่วยความจำเสมือนส่วนที่อยู่ในหน่วยความจำหลัก (VmRSS: Virtual Memory Resident Set Size) ในไฟล์ชื่อ status เช่นเดียวกับขนาดของหน่วยความจำเสมือน และพบว่า ค่าทั้งสองนั้นเพิ่มขึ้นเท่ากัน เนื่องจากขนาดของหน่วยความจำเสมือนเท่ากับผลรวมของขนาดของหน่วยความจำเสมือนส่วนที่อยู่ในหน่วยความจำหลักและขนาดของหน่วยความจำเสมือนส่วนที่อยู่ในพื้นที่สับค่า (Swap Space) นั่นเอง และสังเกตได้จากขนาดของหน่วยความจำเสมือนส่วนที่เป็นข้อมูลในฮีปของโพรเซส (VmData: Virtual Memory Data) ในไฟล์ชื่อ status เช่นกัน ก็มีค่าเพิ่มขึ้นเท่ากับขนาดของหน่วยความจำเสมือนด้วย แต่ค่าอื่นๆได้แก่ ขนาดของหน่วยความจำเสมือนส่วนที่เป็นข้อมูลในสแต็ก (VmStk: Virtual Memory Stack) ขนาดของหน่วยความจำเสมือนส่วนที่เป็นโค้ดการทำงานของโปรแกรมและคลังโยงแบบสถิต (Statically Linked Libraries) (VmExe: Executable Virtual Memory) และขนาดของหน่วยความจำเสมือนส่วนที่เป็นคลังโยงแบบพลวัต (Dynamically Linked Libraries) (VmLib: Virtual Memory Libraries) มีขนาดเท่าเดิม

ขนาดของโปรแกรมที่นำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่ก็เล็กลงกว่าแบบเก่าเช่นกัน โปรแกรมที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่จะมีขนาดของโปรแกรมเท่ากันระหว่างการใส่ออปชัน -Os และการใส่ออปชัน -O3 แม้ว่าออปชัน -O3 อาจทำให้ขนาดของโปรแกรมลดลงได้

ด้วย แต่ในกรณีนี้ ออปชัน -O3 จะไม่สามารถใช้ได้ เพราะมีรหัสคำสั่งที่ไม่รู้จักจำนวนมาก ซึ่งเป็นคำสั่งที่เราสร้างขึ้นใหม่นั่นเอง และไม่ได้ใช้คำกำหนดคุณลักษณะ (Specifier) ที่เป็นอินไลน์ (Inline) ฟังก์ชันเหมือนโปรแกรมที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบเก่า นอกจากนี้ การคอมไพล์โดยใช้ออปชัน -g จะสร้างข้อมูลที่ใช้ในการแก้จุดบกพร่อง เช่น โค้ดของโปรแกรมภาษาซี ทำให้มีขนาดของโปรแกรมส่วนที่ไม่จำเป็นอยู่ในผลลัพธ์ของโครงงานทางวิศวกรรมในระดับปริญญาตรีด้วย อีกทั้งขนาดของโปรแกรมค้นหาแบบทวิภาคด้วยต้นไม้แบบเอวีแอลที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่มีขนาดเล็กลงกว่าแบบดั้งเดิมด้วย เนื่องจากแบบใหม่ไม่มีโค้ดของโปรแกรมภาษาซีแทรกอยู่ในขนาดของโปรแกรมเหมือนกับแบบดั้งเดิมนั่นเอง

5.4 วิเคราะห์ผลการวิจัยเพิ่มเติม

นอกจากการอภิปรายผลการวิจัยในแง่ความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ และประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆแล้ว ยังมีการวิเคราะห์ในแง่อื่นที่มีความสำคัญไม่น้อยเช่นกัน ได้แก่

5.4.1 การวิเคราะห์ค่าใช้จ่าย (Cost Analysis)

ผลการทดลองในหัวข้อ 5.2 หน้า 40 สามารถนำมาคำนวณค่าเฉลี่ยเรขาคณิต (Geometric Mean) เนื่องจากเป็นตัวเลขจำนวนเท่า เมื่อวิเคราะห์ผลแล้ว จึงสรุปได้ว่า ถ้าหากต้องการให้ระบบมีสามารถป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ด้วยวิธีซีเคียวคานารีเวิร์ดได้นั้น ระบบต้องยอมแลกกับค่าใช้จ่าย (Cost) ดังต่อไปนี้

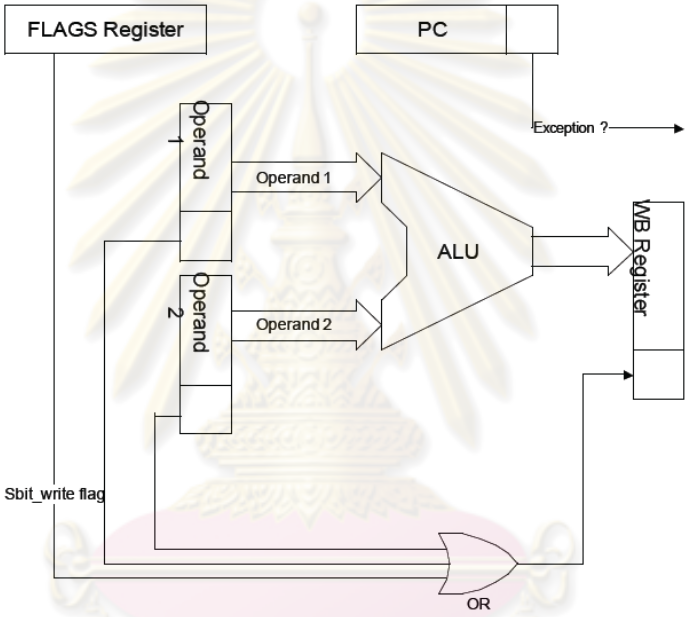
1. ความเร็วของโปรแกรมที่ช้าลงกว่าเดิมประมาณร้อยละ 32.099 (คำนวณจาก $1 - \sqrt[3]{0.90933 \times 0.45312 \times 0.75979} \approx 0.32099$)
2. ขนาดของหน่วยความจำที่ใช้ในขณะที่โปรแกรมทำงานที่ใหญ่กว่าเดิมประมาณร้อยละ 2.946 (คำนวณจาก $\sqrt[3]{1 \times 1 \times 1.091} \approx 1.029457$)
3. ขนาดโปรแกรมที่ใหญ่กว่าเดิม
   a. กรณีที่ไม่ใช้ออปชันใดเลยตอนคอมไพล์ ขนาดโปรแกรมที่ใหญ่กว่าเดิมประมาณร้อยละ 6.178 (คำนวณจาก $\sqrt[3]{1.08044 \times 1.03287 \times 1.07266} \approx 1.06178$)
   b. กรณีที่ใช้ออปชัน -Os ตอนคอมไพล์ ขนาดโปรแกรมที่ใหญ่กว่าเดิมประมาณร้อยละ 8.250 (คำนวณจาก $\sqrt[3]{1.09606 \times 1.04674 \times 1.10563} \approx 1.08250$)

c. กรณีที่ใช้ออปชัน -O3 ตอนคอมไพล์ ขนาดโปรแกรมที่ใหญ่กว่าเดิมประมาณร้อยละ 7.132 (คำนวณจาก $\sqrt[3]{1.09606 \times 1.04814 \times 1.07029} \approx 1.07132$)

5.4.2 **การดีพลอยเมนต์** (Deployment)

วิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดในการวิจัยนี้สามารถนำไปดีพลอย (Deploy) ได้จริง โดยต้องเปลี่ยนอุปกรณ์ฮาร์ดแวร์เหมือนวิธีซีเคียวบิต ได้แก่

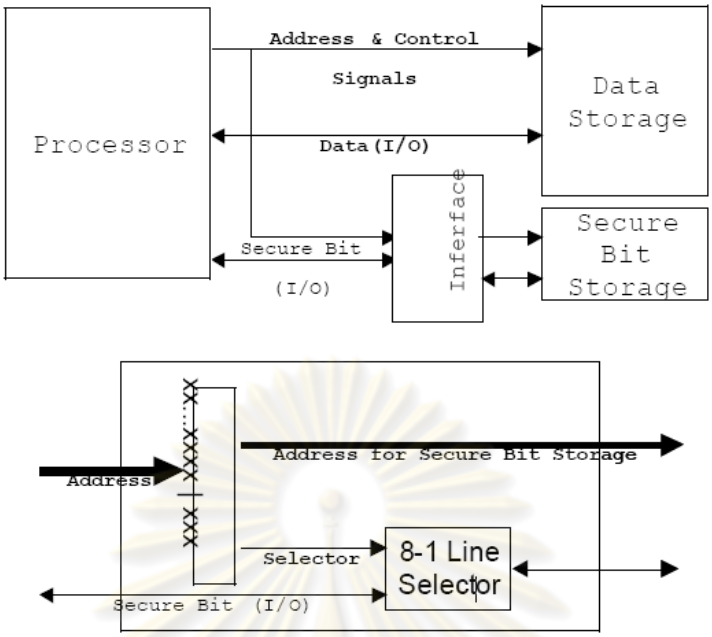
1. หน่วยประมวลผลกลาง (CPU) ที่ออกแบบใหม่เพื่อรองรับซีเคียวบิต ดังรูปที่ 7 โดยนอกจากการทำงานตามปกติแล้ว จะใช้อีกรอบคำสั่งเครื่อง เพื่อตรวจสอบซีเคียวบิตของคานารีเวิร์ดนั้น



รูปที่ 7 แผนภาพวงจร (Circuit Diagram) ของโครงสร้างหน่วยประมวลผลกลางใหม่

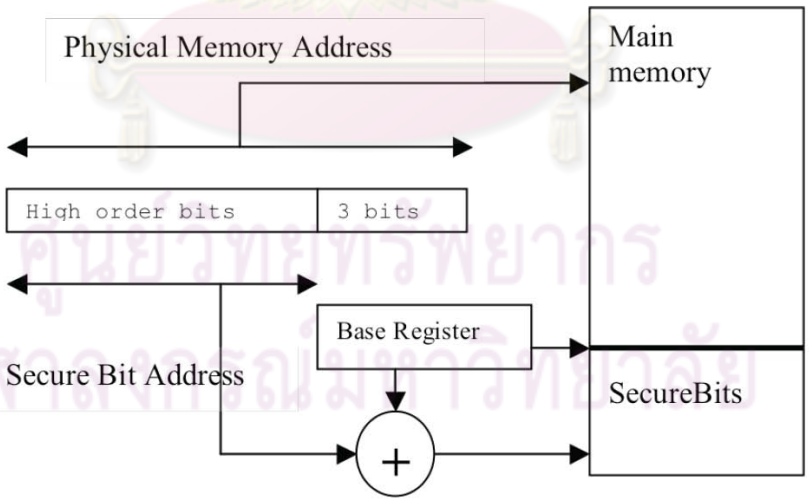
2. หน่วยความจำและแผงหลัก (Mother Board) ที่ออกแบบใหม่เพื่อรองรับซีเคียวบิต

ดังนั้น อุปกรณ์ฮาร์ดแวร์ทั้งหมดนี้มีส่วนต่อประสาน (Interface) และตัวควบคุมซีเคียวบิต (Secure Bit Controller) ดังรูปที่ 8



รูปที่ 8 แผนภาพบล็อก (Block Diagram) ของส่วนต่อประสาน (Interface) ระหว่างอุปกรณ์ฮาร์ดแวร์ทั้งหมด และตัวควบคุมซีเคียวบิต (Secure Bit Controller)

นอกจากนี้ ยังมีวิธีดีพลอยอีกวิธีหนึ่ง นั่นคือ เปลี่ยนเฉพาะหน่วยประมวลผลกลางแบบเดียวกับในรูปที่ 7 แต่ไม่ต้องเปลี่ยนหน่วยความจำและแผงหลัก โดยอาศัยวิธีการแปลงแอดเดรส (Address Translation) (รายละเอียดเพิ่มเติมสามารถอ่านได้ที่ [60]) ดังรูปที่ 9



รูปที่ 9 แผนภาพบล็อก (Block Diagram) ของการแปลงแอดเดรส (Address Translation)

บทที่ 6
บทสรุป

ในบทนี้จะกล่าวถึงสิ่งที่ได้จากการวิจัย ประโยชน์ของซีเคียวคานารีเวิร์ด (Secure Canary Word) แนวทางการวิจัยต่อ และบทสรุป ดังนี้

6.1 สิ่งที่ได้จากการวิจัย (Contribution)

สิ่งที่ได้จากการวิจัยนี้ ได้แก่

1. อธิบายขั้นตอนวิธีการโจมตีเครื่องคอมพิวเตอร์ด้วยบัฟเฟอร์โอเวอร์โฟลว์ (Buffer-overflow Attacks) รูปแบบต่างๆ
2. การสำรวจและแบ่งรูปแบบของวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์ที่มีอยู่ในปัจจุบัน โดยนำเสนอหลักการ และจุดอ่อนของวิธีแต่ละรูปแบบ รวมทั้งตัวอย่างของเครื่องมือที่ใช้วิธีรูปแบบต่างๆด้วย
3. นำเสนอหลักการของวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด ซึ่งได้นำหลักการของคานารีเวิร์ด (Canary Word) และซีเคียวบิต (Secure Bit) มาใช้ร่วมกัน ทำให้มีประสิทธิภาพในการป้องกันข้อมูลทั้งส่วนที่เป็นตัวควบคุมระบบ (Control Data) และส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) ได้ รวมทั้งแนวทางการนำซีเคียวคานารีเวิร์ดมาทำให้เกิดผล (Implementation) ด้วย
4. แสดงให้เห็นว่า ความมั่นคง (Security) ของข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data) เกิดได้ โดยไม่ทำให้โปรแกรมใหญ่ขึ้นหรือช้าลง

6.2 ประโยชน์ของซีเคียวคานารีเวิร์ด

ซีเคียวคานารีเวิร์ดสามารถป้องกันการโจมตีด้วยบัฟเฟอร์โอเวอร์โฟลว์ในข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบได้ ถือว่าเป็นหนึ่งในวิธีจำนวนไม่มากที่สามารถป้องกันข้อมูลส่วนนี้ได้ โดยที่ยังสามารถป้องกันการโจมตีข้อมูลส่วนที่เป็นตัวควบคุมระบบได้เหมือนเดิม เมื่อเปรียบเทียบผลกระทบต่อประสิทธิภาพของโปรแกรมกับความปลอดภัยของระบบที่สูงขึ้นแล้ว ซีเคียวคานารีเวิร์ดเป็นวิธีการที่คุ้มค่าต่อการนำไปใช้จริงอย่างยิ่ง

6.3 แนวทางการวิจัยต่อ

การวิจัยนี้ได้นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลในส่วนฮาร์ดแวร์ (Hardware) (ดังหัวข้อ 4.2.1 หน้า 30) สำเร็จแล้ว แต่ยังมีแนวทางการปรับปรุง

ให้ดีขึ้นอีก เนื่องจากในส่วนมีข้อจำกัดในส่วนซอฟต์แวร์ (Software) (ดังหัวข้อ 4.2.2 หน้า 36) ที่ไม่สามารถประกาศคานารีเวิร์ดแทรกระหว่างแต่ละอาร์กิวเมนต์ (Argument) ได้ ดังนั้น ถ้าสามารถดัดแปรคอมไพเลอร์ (Modify Compiler) ให้สามารถแทรกคานารีเวิร์ดระหว่างแต่ละตัวแปรภายในโปรแกรม (Local Variable) และอาร์กิวเมนต์ได้ จะช่วยให้วิธีนี้มีประสิทธิภาพในการป้องกันได้ครอบคลุมยิ่งขึ้นด้วย เนื่องจากวิธีในการวิจัยนี้ใช้วิธีการประกาศคานารีเวิร์ดเป็นตัวแปรคั่นได้เฉพาะระหว่างตัวแปรภายในโปรแกรมเท่านั้น จึงไม่สามารถปกป้องครอบคลุมถึงอาร์กิวเมนต์ได้ นอกจากนี้ ยังช่วยให้มีคุณสมบัติโปร่งใส (Transparent) ต่อผู้เขียนโปรแกรมด้วย เนื่องจากเมื่อสั่งคอมไพล์แล้ว คอมไพเลอร์จะเปลี่ยนคำสั่งบรรจุ (Load Instruction) และคำสั่งเปรียบเทียบ (Compare Instruction) ที่เกี่ยวข้องให้เรียกคำสั่งที่สร้างใหม่แทนโดยอัตโนมัติได้

สมมติ ตัวอย่างโค้ดของโปรแกรม ดังต่อไปนี้

```
int main(int argc, char* argv[]){
  char a=*argv[1];
  if(a=='a') return 0; else return 1;
}
```

เมื่อสั่งคอมไพล์แล้ว จะได้โค้ดแอสเซมบลี (Assembly Code) ในส่วนฟังก์ชันหลัก (Main Function) และผังสแต็ก (Stack Layout) ทั้งก่อนและหลังดัดแปรคอมไพเลอร์ ดังตารางที่ 20

ตารางที่ 20 ตารางเปรียบเทียบก่อนและหลังดัดแปรคอมไพเลอร์ (Modify Compiler)

การเปรียบเทียบ	ก่อนดัดแปร	หลังดัดแปร

โค้ดแอสเซมบลี	`main:`
	`  pushl %ebp`
	`  movl %esp,%ebp`
	`  movl 12(%ebp),%eax`
	`  movl 4(%eax),%eax`
	`  cmpb $97,(%eax)`
	`  je .L2`
	`  movl $1,%eax`
	`  jmp .L4`
	`  .p2align 4,,7`
	`.L2:`
	`  xorl %eax,%eax`
	`.L4:`
	`  leave`
	`  ret`
		`main:`
		`  pushl %ebp`
		`  movl %esp,%ebp`
		`  movl_vd 12(%ebp),%eax`
		`  movl_vd 4(%eax),%eax`
		`  cmpb $97,(%eax)`
		`  je .L2`
		`  movl $1,%eax`
		`  jmp .L4`
		`  .p2align 4,,7`
		`.L2:`
		`  xorl %eax,%eax`
		`.L4:`
		`  leave`
		`  ret`

ผังสแต็ก	1 | a	1 | a 0 | canWord_a

จากตารางที่ 20 สังเกตว่า จะเปลี่ยนคำสั่งบรรจุข้อมูลจากหน่วยความจำ ซึ่งในที่นี้คือคำสั่ง movl ที่มีอาร์กิวเมนต์ตัวแรกเป็นตำแหน่งของหน่วยความจำ มาเป็นคำสั่ง movl_vd ซึ่งเป็น

คำสั่งใหม่สำหรับตรวจสอบค่าซีเคียวบิตของคานารีเวิร์ดของตัวแปรที่เก็บค่าอยู่ในตำแหน่งนี้ไป พร้อมกับบรรจุค่าของตัวแปรเข้ามาในเรจีสเตอร์ ในหน่วยประมวลผลกลาง ซึ่งในที่นี้คือ EAX (หรือ %eax)

อย่างไรก็ตาม ในการวิจัยนี้ได้ไล่ดูโค้ดของคอมไพเลอร์ชื่อ จีซีซี (GCC) เวอร์ชัน egcs-2.91.66 (ดาว์นโหลดจาก [59]) หลังจากการไล่ดูโค้ดแล้วระดับหนึ่ง สามารถสรุปขั้นตอนการทำงาน ดังนี้

1. เริ่มต้นที่ไฟล์ชื่อ toplev.c มีฟังก์ชันหลัก (Main Function) อยู่ ซึ่งการทำงานภายในฟังก์ชันนี้จะไปเรียกใช้ฟังก์ชัน compile_file(name) อยู่ในไฟล์นี้เช่นกัน
2. ภายในฟังก์ชัน compile_file(name) จะกำหนดชื่อไฟล์ที่เก็บโค้ดแอสเซมบลี โดยถ้าหากตอนสั่งคอมไพล์ด้วยโปรแกรมจีซีซีนี้ไม่ได้ใส่ออปชัน -S ด้วย ไฟล์นี้จะถูกเก็บอยู่ที่ไดเร็กทอรีชื่อ tmp แต่ถ้าใส่ออปชันนี้ ไฟล์นี้จะถูกเก็บในที่ที่กำหนดไว้ตามออปชัน
3. ในฟังก์ชันหลักจะเรียกใช้ฟังก์ชัน init_tree_codes() และ init_parse(name) เพื่อจองพื้นที่ตารางการแจงส่วน (Parsing Table) [62] สำหรับเตรียมรับข้อมูล
4. ในฟังก์ชันหลักจะเรียกใช้ฟังก์ชัน yyparse() ซึ่งเป็นตัวแจงส่วน (Parser) ชนิด LR [63] โดยอ่านข้อมูลนำเข้าจากซ้ายไปขวา เนื่องจากสร้างโดย gnu bison ซึ่งเป็นคอมไพเลอร์-คอมไพเลอร์ (Compiler-compiler) หรือตัวสร้างคอมไพเลอร์ [64] และมีการใช้ ตารางแอ็คชัน (Action Table) กับ ตารางโกทู (Goto Table)
5. ฟังก์ชัน yyparse() อยู่ในไฟล์ชื่อ c-parse.c พบว่ามีหลายสถานะ (State) ซึ่งมีสถานะที่เกี่ยวข้องนั่นคือ สถานะชื่อ reduction สามารถแบ่งเป็นอีกหลายกรณี (Case) ในที่นี้พบว่า การประกาศตัวแปรนั้น จะเกี่ยวข้องกับกรณีที่ 164 เรียกใช้ฟังก์ชัน start_decl() อยู่ในไฟล์ชื่อ c-decl.c และกรณีที่ 165 เรียกใช้ฟังก์ชัน finish_decl() อยู่ในไฟล์ชื่อ c-decl.c เช่นกัน
6. ในฟังก์ชัน finish_decl() จะเรียกใช้ฟังก์ชัน layout_decl() อยู่ในไฟล์ชื่อ stor_layout.c มีตัวแปรที่น่าจะเกี่ยวข้อง 2 ตัว นั่นคือ DECL_SIZE(decl) และ DECL_ALIGN(decl)
7. เมื่อฟังก์ชัน yyparse() ทำงานเสร็จสิ้น จะได้ต้นไม้แจงส่วน (Parse Tree) [65] และตารางสัญลักษณ์ (Symbol Table) [66] ที่เป็นตัวแปร globals ซึ่งได้มาจากการเรียกใช้ฟังกชัน getdecls()

จากขั้นตอนที่หก เมื่อแก้ไขโค้ดโดยเพิ่มค่า DECL_SIZE(decl) หรือ DECL_ALIGN(decl) อีก 4 เพื่อให้ตำแหน่งแอดเดรส (Address) ของตัวแปรที่ประกาศไว้เลื่อนไปอีก 4 ไบต์ (เท่ากับ 1

เวิร์ด) เปรียบเสมือนมีคานารีเวิร์ดคั่นอยู่ แต่เมื่อทดลองสร้าง (Make) คอมไพเลอร์ใหม่ขึ้นมาใช้งาน (รายละเอียดของขั้นตอนนี้อยู่ในภาคผนวก จ. หน้า 114) ก็ยังพบว่าเมื่อคอมไพล์โค้ดของโปรแกรมที่เขียนขึ้นแล้ว ตำแหน่งแอดเดรสของตัวแปรยังคงอยู่ที่เดิม ดังนั้น จึงทำให้งานส่วนนี้ยังไม่สามารถทำให้เสร็จสมบูรณ์ได้ในตอนนี้

6.4 บทสรุป

หากมีสถาปัตยกรรมของซีเคียวบิตแล้ว เราสามารถใช้ประโยชน์จากซีเคียวบิตได้หลายทาง โดยการวิจัยนี้เป็นการนำซีเคียวบิตมาประยุกต์ใช้ร่วมกับคานารีเวิร์ด เพื่อป้องกันการโจมตีด้วยบัฟเฟอร์โอเวอร์โฟลว์ในข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ เช่น ตัวแปร และอาร์กิวเมนต์ เป็นต้น ให้ปลอดภัยยิ่งขึ้นได้

รายการอ้างอิง

[1] Schmidt, C. and Darby, T. The What, Why, and How of the 1988 Internet Worm [Online]. Available from: http://www.snowplow.org/tom/worm/worm.html [1988, Nov 3]

[2] U.S. Department of Homeland Security. US-CERT Vulnerability Notes [Online]. Available from: http://www.kb.cert.org/vuls/bypublic [2009, Nov 24]

[3] ARS Technica. The sky isn't falling: a look at a new Vista security bypass [Online]. Available from: http://arstechnica.com/news.ars/post/20080811-the-sky-isnt-falling-a-look-at-a-new-vista-security-bypass.html [2008, Aug 11]

[4] Webopedia. What is buffer overflow? [Online]. Available from: http://www.webopedia.com/TERM/B/buffer_overflow.html [2003, July 29]

[5] Piromsopa, K. and Chiamwongpaet, S. Secure Bit Enhanced Canary: Hardware Enhanced Buffer-Overflow Protection. NSS 2008: Proceedings of IFIP International Workshop on Network and System Security, pp. 125-131, Shanghai, China : IEEE Computer Society, 2008.

[6] Piromsopa, K. and Chiamwongpaet, S. The Implementation of Secure Canary Word for Buffer-Overflow Protection. EIT 2009: Proceedings of Electro/Information Technology Conference, pp. 56-61, Windsor, Ontario, Canada : IEEE Computer Society, 2009.

[7] Piromsopa, K. and Enbody, R., J. Survey of Protection from Buffer-Overflow Attacks. Technical Report, ACM Computing Surveys, 2006.

[8] Viega, J., Bloch, T., J., Kohno, Y. and McGraw, G. ITS4: A static vulnerability scanner for C and C++ code. ACSAC '00: Proceedings of the 16th Annual Computer Security Applications Conference, pp. 257, Washington : IEEE Computer Society, 2000.

[9] Wheeler, D., A. Flawfinder Home Page [Online]. Available from: http://www.dwheeler.com/flawfinder/

[10] RATS. RATS [Online]. Available from: http://www.securesw.com/rats/

[11] Haugh, E. and Bishop, M. Testing C Programs for Buffer Overflow Vulnerabilities. SNDSS 2003: Proceedings of the Network and Distributed System Security, pp. 123-130, San Diego: The Internet Society, 2003.

[12] Baratloo, A., Singh, N. and Tsai, T. Transparent run-time defense against stack smashing attacks. ATEC '00: Proceedings of the annual conference on USENIX Annual Technical Conference, pp. 21, Berkeley: USENIX Association, 2000.

[13] Evans, D., and Larochelle, D. Improving Security Using Extensible Lightweight Static Analysis. IEEE Software vol. 19 (January-February 2002) : pp. 42-51.

[14] Wagner, D., Foster, J., S., Brewer, E., A. and Aiken, A. A First Step Towards Automated Detection of Buffer Overrun Vulnerabilities. NDSS '00: Proceedings of the Network and Distributed System Security Symposium, pp. 3-17, San Diego: Internet Society, 2000.

[15] Cowan, C., Beattie, S., Day, R., F., Pu, C., Wagle, P. and Walthinsen, E. Protecting Systems from Stack Smashing Attacks with StackGuard. Proceedings of the Fifth Linux Expo, Raleigh, 1999.

[16] Cowan, C., Wagle, P., Pu, C., Beattie, S. and Walpole, J. Buffer Overflows: Attacks and Defenses for the Vulnerability of the Decade. DISCEX '00: Proceedings of DARPA Information Survivability Conference and Exposition, pp. 119-129, Hilton Head: IEEE Computer Society Customer Service Center, 2000.

[17] Hinton, H., Cowan, C., Delcambre, L. and Bowers, S. SAM: Security Adaptation Manager. ACSAC '99: Proceedings of the 15th Annual Computer Security Applications Conference, pp. 361, Washington: IEEE Computer Society, 1999.

[18] Etoh, H. GCC extension for protecting applications from stack-smashing attacks [Online]. Available from: http://www.trl.ibm.com/projects/security/ssp/ [2005, August 22]

[19] Cowan, C., Beattie, S., Johansen, J. and Wagle, P. Pointguard: protecting pointers from buffer overflow vulnerabilities. SSYM'03: Proceedings of the 12th

conference on USENIX Security Symposium, pp. 7, Berkeley: USENIX Association, 2003.

[20] Shao, Z., Zhuge, Q., He, Y. and Sha, E., H.-M. Defending Embedded Systems Against Buffer Overflow via Hardware/Software. ACSAC '03: Proceedings of the 19th Annual Computer Security Applications Conference, pp. 352, Washington: IEEE Computer Society, 2003.

[21] Tuck, N., Calder, B. and Varghese, G. Hardware and Binary Modification Support for Code Pointer Protection From Buffer Overflow. MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, pp. 209-220, Washington: IEEE Computer Society, 2004.

[22] Frantzen, M., and Shuey, M. StackGhost: Hardware facilitated stack protection. SSYM '01: Proceedings of the 10th conference on USENIX Security Symposium, pp. 5, Berkeley: USENIX Association, 2001.

[23] Mcgregor, J., P., Karig, D., K., Shi, Z. and Lee, R., B. A Processor Architecture Defense against Buffer Overflow Attacks. ITRE 2003: IEEE International Conference on Information Technology: Research and Education, pp. 243-250, Newark: IEEE Computer Society, 2003.

[24] Xu, J., Kalbarczyk, Z., Patel, S. and Iyer, R., K. Architecture support for defending against buffer overflow attacks. EASY 2002: Proceedings of the Second Workshop on Evaluating and Architecting, San Jose, 2002.

[25] Ye, D. and Kaeli, D. A reliable return address stack: microarchitectural features to defeat stack smashing. SIGARCH Comput. Archit. News (ACM) vol. 33 (March 2005) : pp. 73-80.

[26] Ozdoganoglu, H., Vijaykumar, T., N., Brodley, C., E., Kuperman, B., A. and Jalote, A. SmashGuard: A Hardware Solution to Prevent Security Attacks on the Function Return Address. IEEE Trans. Comput. (IEEE Computer Society) vol. 55 (October 2006) : pp. 1271-1285.

[27] Chiueh, T. and Hsu, F. RAD: A Compile-Time Solution to Buffer Overflow Attacks. ICDCS '01: Proceedings of the The 21st International Conference on

Distributed Computing Systems, pp. 409, Washington: IEEE Computer Society, 2001.

[28] Prasad, M. and Chiueh, T. A Binary Rewriting Defense Against Stack based Buffer Overflow Attacks. Proceedings of the General Track: 2003 USENIX Annual Technical Conference, pp. 211-224, San Antonio: USENIX, 2003.

[29] Corliss, M., L., Lewis, E., C. and Roth, A. Using DISE to protect return addresses from attack. SIGARCH Comput. Archit. News (ACM) vol. 33 (March 2005) : pp. 65-72.

[30] Vendicator. Stack Shield technical info file v0.7 [Online]. Available from: http://www.angelfire.com/sk/stackshield/ [2000, January 8]

[31] Inoue, K. Energy-security tradeoff in a secure cache architecture against buffer overflow attacks. SIGARCH Comput. Archit. News (ACM) vol. 33 (March 2005) : pp. 81-89.

[32] Gehringer, E., F. and Keedy, J., L. Tagged architecture: how compelling are its advantages? SIGARCH Comput. Archit. News (ACM) vol. 13 (June 1985) : pp. 162-170.

[33] Piromsopa, K. and Enbody, R., J. Secure Bit: Transparent, Hardware Buffer-Overflow Protection. Dependable and Secure Computing, IEEE Transactions vol. 3 (October-December 2006) : pp. 365-376.

[34] Crandall, J., R. and Chong, F., T. A security assessment of the minos architecture. SIGARCH Comput. Archit. News (ACM) vol. 33 (March 2005) : pp. 48-57.

[35] Crandall, J., R. and Chong, F., T. Minos: Control Data Attack Prevention Orthogonal to Memory Model. MICRO 37: Proceedings of the 37th annual IEEE/ACM International Symposium on Microarchitecture, pp. 221-232, Washington: IEEE Computer Society, 2004.

[36] Xu, J. and Nakka, N. Defeating Memory Corruption Attacks via Pointer Taintedness Detection. DSN '05: Proceedings of the 2005 International Conference on Dependable Systems and Networks, pp. 378-387, Washington: IEEE Computer Society, 2005.

[37] Organick, E., I. A Programmer's View of the Intel 432 System. New York: McGraw-Hill, 1983.

[38] Colwell, R., P., Hitchcock, C., Y., III, Jensen, E., D., Brinkley, S., H.M. and Kollar, C., P. Instruction Sets and Beyond: Computers, Complexity, and Controversy. Computer (IEEE Computer Society Press) vol. 18 (September 1985) : pp. 8-19.

[39] Jones, R., W.M. and Kelly, P., H.J. Backwards-Compatible Bounds Checking for Arrays and Pointers in C Programs. AADEBUG '97, Proceedings of the Third International Workshop on Automatic Debugging, pp. 255-283, 1997.

[40] IBM Corporation. IBM Software - Rational PurifyPlus [Online]. Available from: http://www-306.ibm.com/software/awdtools/purifyplus/

[41] Compuware Corporation. Compuware DevPartner for Visual C++ [Online]. Available from: http://www.compuware.com/products/devpartner/visualc.htm

[42] Austin, T., M., Breach, S., E. and Sohi, G., S. Efficient detection of all pointer and array access errors. SIGPLAN Not. (ACM) vol. 29 (June 1994) : pp. 290-301.

[43] Yutaka, O., Sekiguchi, T., Sumii, E. and Yonezawa, A. Fail-Safe ANSI-C Compiler: An Approach to Making C Programs Secure: Progress Report. Proceedings of International Symposium on Software Security 2002, pp. 133-153, Berlin: Springer, 2002.

[44] Lhee, K.and Chapin, S., J. Buffer overflow and format string overflow vulnerabilities. Softw. Pract. Exper. (John Wiley and Sons, Inc.) vol. 33 (April 2003) : pp. 423-460.

[45] Bhatkar, S., DuVarney, D., C. and Sekar, R. Address obfuscation: An efficient approach to combat a broad range of memory error exploits. Proceedings of the 12th Usenix Security Symposium, pp. 105–120, Washington: USENIX Association, 2003.

[46] The PaX Team. Homepage of PaX [Online]. Available from: http://pax.grsecurity.net/ [2008, December 27]

[47] Krazit, T. PC World - AMD Chips Guard Against Trojan Horses [Online]. Available from:http://www.pcworld.com/article/114328/amd_chips_guard_against_trojan_horses.html [2004, January 15]

[48] Peterson, D., S., Bishop, M. and Pandey, R. A Flexible Containment Mechanism for Executing Untrusted Code. Proceedings of the 11th USENIX Security Symposium, pp. 207-225, Berkeley: USENIX Association, 2002.

[49] Chang, F., Itzkovitz, A. and Karamcheti, V. User-level resource-constrained sandboxing. WSS'00: Proceedings of the 4th conference on USENIX Windows Systems Symposium, pp. 3, Berkeley: USENIX Association, 2000.

[50] Wahbe, R., Lucco, S., Anderson, T., E. and Graham, S., L. Efficient software-based fault isolation. SOSP '93: Proceedings of the fourteenth ACM symposium on Operating systems principles, pp. 203-216, New York: ACM, 1993.

[51] Small, C. MiSFIT: A tool for constructing safe extensible C++ systems. Proceedings of the Third USENIX Conference on Object-Oriented Technologies, pp. 174-184, Portland: USENIX, 1997.

[52] Intel Corporation. LaGrande Technology Architectural Overview [Online]. Available from:ftp://download.intel.com/technology/security/downloads/LT_Arch_Overview.pdf [2003, September]

[53] MacDonald, R., Smith, S., W., Marchesini, J. and Wild, O. Bear: An Open-Source Virtual Secure Coprocessor based on TCPA. TR2003-471, Computer Science, Dartmouth College, Hanover, 2003.

[54] Trusted Computing Platform Alliance. TCPA IT white paper. 2004.

[55] ARM. TrustZone Technology Overview [Online]. Available from: http://www.arm.com/products/security/trustzone/index.html

[56] Microsoft Corporation. Next-Generation Secure Computing Base [Online]. Available from: http://www.microsoft.com/resources/ngscb/default.mspx

[57] Kgil, T., Falk, L. and Mudge, T. ChipLock: support for secure microarchitectures. SIGARCH Comput. Archit. News (ACM) vol. 33 (March 2005) : pp. 134-143.

[58] Ruppert, V. bochs: The Open Source IA-32 Emulation Project (Home Page) [Online]. Available from: http://bochs.sourceforge.net/ [2009, November 12]

[59] The GCC team. GCC, the GNU Compiler Collection - GNU Project - Free Software Foundation (FSF) [Online]. Available from: http://gcc.gnu.org/, [2009, December 2]

[60] Piromsopa, K., and Enbody, R., J. Architecting security: a secure implementation of hardware buffer-overflow protection. ACST'07: Proceedings of the third conference on IASTED International Conference, pp. 17-22, Phuket: ACTA Press, 2007.

[61] Wikipedia. Amdahl's law - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/Amdahl's_law [2009, December 3]

[62] Wikipedia. Parsing table - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/Parsing_table [2009, April 29]

[63] Wikipedia. LR parser - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/LR_parser [2009, November 23]

[64] Wikipedia. Compiler-compiler - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/Compiler-compiler [2009, October 23]

[65] Wikipedia. Concrete syntax tree - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/Concrete_syntax_tree [2009, December 10]

[66] Wikipedia. Symbol table - Wikipedia, the free encyclopedia [Online]. Available from: http://en.wikipedia.org/wiki/Symbol_table [2009, October 14]

ภาคผนวก

ภาคผนวก ก.
การนำการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลในโครงงานทางวิศวกรรม (Senior Project) ในระดับปริญญาตรี

เนื่องจากการวิจัยนี้เป็นงานที่ต่อยอดมาจากโครงงานทางวิศวกรรม (Senior Project) ในระดับปริญญาตรี จึงได้เคยนำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลไว้แล้ว ดังนี้

เพิ่มคำสั่งของหน่วยประมวลผลกลาง (CPU instruction)

ส่วนนี้จะดัดแปรหน่วยประมวลผลกลาง (Modify CPU) โดยเพิ่มคำสั่งใหม่ที่จะตรวจสอบค่าซีเคียวบิต (Secure Bit) ของค่าในเรจีสเตอร์ (Register) EAX ในหน่วยประมวลผลกลาง (CPU) ในโครงงานนี้จะดัดแปรบนซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์ (BOCHS Emulator) [58] โดยเลือกใช้รหัสคำสั่ง (Operation Code) เป็น 0x0F1A ที่ยังว่างอยู่ และแก้ไขโค้ดในไฟล์ (File) ที่เกี่ยวข้องกับการทำงานของหน่วยประมวลผลกลาง ซึ่งอยู่ในไดเร็กทอรี (Directory) ชื่อ cpu ดังนี้

ในไฟล์ชื่อ data_xfer32.cc ที่ท้ายไฟล์ สร้างฟังก์ชันใหม่ ดังนี้

```
  void
BX_CPU_C::CHKSBIT(bxInstruction_c *i)
{
    int sbit,result;
    sbit = BX_CPU_RSBIT[BX_32BIT_REG_EAX];
    result = (sbit == 0) ? 0 : 0xFF;
    BX_WRITE_32BIT_REGZ(0, result);
}
```

ในไฟล์ชื่อ cpu.h ประมาณบรรทัดที่ 2125 เพิ่มโค้ดเพื่อประกาศชื่อฟังก์ชันใหม่ที่สร้างไว้ข้างต้น ดังนี้

```
BX_SMF void CHKSBIT(bxInstruction_c *);
```

ในไฟล์ชื่อ fetchdecode.cc ประมาณบรรทัดที่ 1302 แก้ไขโค้ดเพื่อให้รหัสคำสั่ง 0x0F1A ที่ยังว่างอยู่ เรียกใช้ฟังก์ชันที่ประกาศไว้ ดังนี้

```
/* 0F 1A */  { 0, &BX_CPU_C::CHKSBIT },
```

ฟังก์ชันใหม่นี้มีหน้าที่ตรวจสอบค่าซีเคียวบิตของค่าในเรจีสเตอร์ EAX ในหน่วยประมวลผลกลาง ณ ขณะนั้น แล้วคืนค่า 0 เมื่อพบว่าซีเคียวบิตมีค่าเป็น 0 หรือคืนค่า 255 เมื่อพบว่าซีเคียวบิตมีค่าเป็น 1 เนื่องจากการตรวจสอบค่าซีเคียวบิตในงานที่ได้นำวิธีการป้องกัน

บัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวบิตมาทำให้เกิดผลนั้นจะกระทำได้โดยเคอร์เนล (Kernel) เท่านั้น แต่ยังไม่ครอบคลุมให้ผู้ใช้ (User) สามารถตรวจสอบค่าซีเคียวบิตนี้ได้เอง โดย

สร้างฟังก์ชันในโปรแกรมภาษาซีเพื่อเรียกใช้เมื่อต้องการตรวจสอบซีเคียวบิตของคานารีเวิร์ดนั้น

ในที่นี้ สร้างฟังก์ชันในโปรแกรมภาษาซีขึ้นมา 3 ฟังก์ชัน ได้แก่

ฟังก์ชัน void chkSBit(char canWord)

ฟังก์ชันนี้จะถูกเรียกใช้เมื่อต้องการตรวจสอบซีเคียวบิตของคานารีเวิร์ดนั้น โดยใช้อินไลน์แอสเซมบลี (In-line Assembly) แทรกภายในฟังก์ชันโปรแกรมภาษาซีเพื่อเรียกใช้รหัสคำสั่งดังกล่าว ซึ่งค่าที่ได้รับคืนมาหลังจากเรียกใช้คำสั่งดังกล่าวจะเก็บอยู่ในตัวแปร sbit โดยหากพบว่าค่าของซีเคียวบิตไม่ใช่ 0 (นั่นคือค่าของตัวแปร sbit เป็น 255) จะแสดงข้อความเตือนออกมาทางหน้าจอ ฟังก์ชันนี้มีโค้ดเป็นดังนี้

```
inline void chkSBit(char canWord)
{
  int sbit = 0;

  asm volatile
  ("
    movl %1,%%eax;
    .short 0x1A0F;
    movl %%eax,%0;
  "
    :"=m"(sbit)
    :"m"(canWord)
    :"%eax"
  );

  if(sbit != 0)
    printf("\nWarning: The following variable is
modified abnormally!\n");
}
```

จากโค้ดดังกล่าวจะมีคำว่า inline เติมอยู่หน้าสุดของส่วนหัวของฟังก์ชัน เพื่อใช้ในการออปทิไมซ์ (Optimize) โค้ดของโปรแกรม ทำให้โปรแกรมไม่ต้องเสียเวลาในการเรียกใช้ฟังก์ชันนี้ เพราะโค้ดของฟังก์ชันจะถูกแทรกอยู่แทนการเรียกใช้ฟังก์ชัน ในขณะที่คอมไพล์ (Compile) เป็นภาษาแอสเซมบลีแล้ว

ฟังก์ชัน int read_int(int *data)

ฟังก์ชันนี้จะถูกเรียกใช้เมื่อต้องการตรวจสอบซีเคียวบิตของคานารีเวิร์ดที่คั่นอยู่ในตำแหน่งแอดเดรสค่าน้อยกว่าของตัวแปรชนิดจำนวนเต็ม (Integer) นั้น โดยรับอาร์กิวเมนต์ (Argument)

เป็นตัวแปรชนิดพอยท์เตอร์ (Pointer) ของตัวแปรชนิดจำนวนเต็ม แล้วนำมาใช้คำนวณตำแหน่งของคานารีเวิร์ด เพื่อส่งให้ฟังก์ชัน chkSBit ตรวจสอบค่าต่อไป และคืนค่าจำนวนเต็มที่ตัวแปรชนิดพอยท์เตอร์นั้นชี้อยู่ฟังก์ชันนี้มีโค้ดเป็นดังนี้

```
inline int read_int(int *data)
{
  char *p = data;
  char canWord;
  p--;
  canWord = *p;
  chkSBit(canWord);
  return *data;
}
```

การคืนค่าเป็นจำนวนเต็มนั้นออกมาด้วย เพื่อความสะดวกในแทรกการตรวจสอบลงในโค้ดของโปรแกรมที่ต้องการนำวิธีการป้องกันดังกล่าวมาทำให้เกิดผล เช่น ตัวแปรที่ใช้ในการตรวจสอบเงื่อนไขของลูปฟอร์ (For Loop) เป็นต้น

ฟังก์ชัน void *read(void *data)

ฟังก์ชันนี้จะถูกเรียกใช้เมื่อต้องการตรวจสอบซีเคียวบิตของคานารีเวิร์ดที่คั่นอยู่ในตำแหน่งแอดเดรสค่ามากกว่าของตัวแปรชนิดพอยท์เตอร์ (Pointer) นั้น โดยรับอาร์กิวเมนต์ (Argument) เป็นตัวแปรชนิดพอยท์เตอร์ของตัวแปรชนิดจำนวนเต็ม แล้วคำนวณตำแหน่งของคานารีเวิร์ด เพื่อส่งให้ฟังก์ชัน chkSBit ตรวจสอบค่าต่อไป และคืนค่าที่ได้รับมาจากอาร์กิวเมนต์เหมือนเดิม ฟังก์ชันนี้มีโค้ดเป็นดังนี้

```
inline void *read(void *data)
{
  char *p = data;
  char canWord;
  p+=4;
  canWord = *p;
  chkSBit(canWord);
  return data;
}
```

การคืนค่าเดิมที่ได้รับจากอาร์กิวเมนต์ออกมาด้วย เพื่อความสะดวกในแทรกการตรวจสอบลงในโค้ดของโปรแกรมที่ต้องการนำวิธีการป้องกันดังกล่าวมาทำให้เกิดผล เช่น ตัวแปรชนิดพอยท์เตอร์ของตัวแปรที่เก็บกลุ่มของข้อมูลที่เรียกว่า โครงสร้างตัวแปร (Structure) เป็นต้น

เมื่อต้องการสั่งคอมไพล์โค้ดของฟังก์ชันให้เป็นคลัง (Library) ให้ใส่ออปชัน -c ด้วย เช่น

```
# gcc -c -o chkSBit.o chkSBit.c
```

เมื่อต้องการสั่งคอมไพล์โค้ดของโปรแกรมโดยเรียกใช้คลังของฟังก์ชันที่สร้างไว้ ให้ใส่ชื่อของคลังต่อท้ายด้วย เช่น

```
# gcc -o bubble_cwsb bubble_cwsb.c chkSBit.o
```

ภาคผนวก ข.
โค้ดของโปรแกรมที่ใช้ทดลองความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่าง ๆ

ในภาคผนวก ข. นี้ จะแสดงโค้ดของโปรแกรมที่ใช้ทดลองความสามารถในการป้องกันการโจมตีด้วยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์ในรูปแบบต่างๆที่ต้องดัดแปรเพิ่มเติมก่อนการทดสอบ ดังนี้

สแต็กโอเวอร์โฟลว์ (Stack Overflows) ที่ต้องการเปลี่ยนแปลงข้อมูลส่วนที่ไม่เป็นตัวควบคุมระบบ (Non-control Data)

โปรแกรมที่ 1

สมมติ โปรแกรมภาษาซีแบบดั้งเดิม ดังนี้

```
#include <string.h>
int main(int argc,char* argv[])
{
  int c = 5;
  char d[2];
  int e = 3;

  printf("Before copy, e = %d\n",e);

  strcpy(d,argv[1]);

  e = c;

  printf("After copy, e = %d\n",e);
  return 0;
}
```

หากสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./stack_noncontrol_mov 12345
```

จะได้ผลลัพธ์ดังนี้

```
Before copy, e = 3
After copy, e = 3486771
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า แม้ว่าในโค้ดจะไม่ได้แก้ไขค่าของตัวแปร c เลย แต่เมื่อให้ค่าของตัวแปร e เท่ากับค่าของตัวแปร c แล้ว ค่าของตัวแปร e ควรจะมีค่าเท่ากับ 5 ซึ่งค่าเริ่มต้นของค่าของตัวแปร c แต่ในผลลัพธ์นั้น ค่าของตัวแปร e กลับมีค่าเท่ากับ 3486771 แทน แสดงว่าค่าของตัวแปร c ถูกแก้ไขด้วยเทคนิคสแต็กโอเวอร์โฟลว์นั่นเอง กล่าวคือ เนื่องจากได้ป้อนข้อมูล

นำเข้าเป็น 12345 ซึ่งมีขนาดใหญ่กว่าบัฟเฟอร์ เมื่อหลังจากเรียกใช้ฟังก์ชัน strcpy() แล้ว ทำให้ค่าของคานารีเวิร์ดของตัวแปร c และตัวแปร c ที่มีตำแหน่งแอดเดรสอยู่ติดกับบัฟเฟอร์ d ถูกแก้ไขค่าเป็น '1' และ '2' ตามลำดับ ดังนั้น ค่าของตัวแปร c จึงมีค่าเท่ากับ 3486771_{10} หรือเท่ากับ 353433_{16} ซึ่งเป็นค่ารหัสแอสกี (ASCII) ของตัวอักษร '5' '4' และ '3' ตามลำดับ (ตัวอย่างและคำอธิบายอย่างละเอียดเพิ่มเติมอยู่ในหัวข้อ 2.1.2.1.1 หน้า 8) แต่เมื่อนำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้ว โดยเรียกใช้คำสั่งที่สร้างขึ้นมาใหม่ด้วยอินไลน์แอสเซมบลี (In-line Assembly) แทนที่โค้ด c = e; (รายละเอียดของขั้นตอนนี้อยู่ในหัวข้อ 4.2.2 หน้า 36) ทำให้โค้ดของโปรแกรมเป็นดังนี้

```
#include <string.h>
int main(int argc,char* argv[])
{
  int c = 5;
  char canWord_c = 0;
  char d[2];
  int e = 3;

  printf("Before copy, e = %d\n",e);

  strcpy(d,argv[1]);

  asm volatile
  ("
    .long 0xFC451B0F;
    mov %%eax,%0;
  "
  :"=m"(e)
  :/*no input*/
  :"%eax"
  );

  printf("After copy, e = %d\n",e);
  return 0;
}
```

หากสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./stack_noncontrol_mov_asm 12345
```

จะได้ผลลัพธ์ดังนี้

```
Before copy, e = 3
Segmentation fault (core dumped)
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า มีข้อความระบุว่าเกิดข้อผิดพลาดในระบบแสดงออกมาทางหน้าจอ และหยุดการทำงานทันทีเมื่อตรวจพบว่าค่าของตัวแปร c ถูกเปลี่ยนแปลงค่าไป จึง

สรุปได้ว่า สามารถป้องกันการโจมตีรูปแบบนี้ได้ด้วยวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด

โปรแกรมที่ 2

สมมติ โปรแกรมภาษาซีแบบดั้งเดิม ดังนี้

```
#include <string.h>
int main(int argc,char* argv[])
{
  int c = 5;
  char d[2];
  int e = 3;

  printf("Before copy, c = %d, e = %d\n",c,e);

  strcpy(d,argv[1]);

  if(c == 5)
    e = 1;
  else
    e = 0;

  printf("After copy, c = %d, e = %d\n",c,e);
  return 0;
}
```

หากสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./stack_noncontrol_cmp 12345
```

จะได้ผลลัพธ์ดังนี้

```
Before copy, c = 5, e = 3
After copy, c = 3486771, e = 0
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า แม้ว่าในโค้ดจะไม่ได้แก้ไขค่าของตัวแปร c เลยเช่นกัน แต่เมื่อเปรียบเทียบค่าของตัวแปร c ว่าเท่ากับ 5 เหมือนเดิมหรือไม่ ค่าของตัวแปร c ควรจะมีค่าเท่ากับ 5 เหมือนเดิม ทำให้ค่าของตัวแปร e มีค่าเท่ากับ 1 แต่ในผลลัพธ์นั้น ค่าของตัวแปร c กลับมีค่าเท่ากับ 3486771 แทน ทำให้ค่าของตัวแปร e มีค่าเท่ากับ 0 แสดงว่าค่าของตัวแปร c ถูกแก้ไขด้วยเทคนิคสแต็กโอเวอร์โฟลว์เช่นกันนั่นเอง กล่าวคือ เนื่องจากได้ป้อนข้อมูลนำเข้าเป็น 12345 ซึ่งมีขนาดใหญ่กว่าบัฟเฟอร์ เมื่อหลังจากเรียกใช้ฟังก์ชัน strcpy() แล้ว ทำให้ค่าของคานารีเวิร์ดของตัวแปร c และตัวแปร c ที่มีตำแหน่งแอดเดรสอยู่ติดกับบัฟเฟอร์ d ถูกแก้ไขค่าเป็น ‘1’ และ ‘2’ ตามลำดับ ดังนั้น ค่าของตัวแปร c จึงมีค่าเท่ากับ 3486771_{10} หรือเท่ากับ 353433_{16} ซึ่งเป็นค่ารหัสแอสกี (ASCII) ของตัวอักษร ‘5’ ‘4’ และ ‘3’ ตามลำดับ แต่เมื่อนำวิธีการป้องกันบัฟเฟอร์โอเวอร์

โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้ว โดยเรียกใช้คำสั่งที่สร้างขึ้นมาใหม่ด้วยอินไลน์แอสเซมบลี (In-line Assembly) แทนที่โค้ด if(c == 5) e = 1; else e = 0; (รายละเอียดของขั้นตอนนี้อยู่ในหัวข้อ 4.2.2 หน้า 36) ทำให้โค้ดของโปรแกรมเป็นดังนี้

```
#include <string.h>
int main(int argc,char* argv[])
{
  int c = 5;
  char canWord_c = 0;
  char d[2];
  int e = 3;

  printf("Before copy, c = %d, e = %d\n",c,e);

  strcpy(d,argv[1]);

  asm volatile
  ("
    .long 0xFC7D1E0F;
    .byte 0x05;
    jne FALSE;
    TRUE:
    movl $0x1,%0;
    jmp END;
    FALSE:
    movl $0x0,%0;
    END:
  "
  :"=m"(e)
  :/*no input*/
  :"%eax"
  );

  printf("After copy, c = %d, e = %d\n",c,e);
  return 0;
}
```

หากสั่งให้โปรแกรมทำงาน ดังนี้

```
# ./stack_noncontrol_cmp_asm 12345
```

จะได้ผลลัพธ์ดังนี้

```
Before copy, c = 5, e = 3
Segmentation fault (core dumped)
```

จากผลลัพธ์ดังกล่าวจะเห็นไว้ว่า มีข้อความระบุว่าเกิดข้อผิดพลาดในระบบแสดงออกมาทางหน้าจอ และหยุดการทำงานทันทีเมื่อตรวจพบว่าค่าของตัวแปร c ถูกเปลี่ยนแปลงค่าไป จึงสรุปได้ว่า สามารถป้องกันการโจมตีรูปแบบนี้ได้ด้วยวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดเช่นกัน

ภาคผนวก ค.
โค้ดของโปรแกรมที่ใช้ทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ

ในภาคผนวก ค. นี้ จะแสดงโค้ดของโปรแกรมที่ใช้ทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆ ทั้ง 3 โปรแกรม ได้แก่ โปรแกรมเรียงลำดับแบบฟอง (Bubble Sort) โปรแกรมควิกซอร์ต (Quick Sort) และโปรแกรมค้นหาแบบทวิภาค (Binary Search) ด้วยต้นไม้แบบเอวีแอล (AVL Tree) แต่ละโปรแกรมมีสี่แบบ ได้แก่

1. แบบดั้งเดิม
2. แบบดั้งเดิมที่ใช้อินไลน์แอสเซมบลี (In-line Assembly)
3. แบบที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด (Secure Canary Word) มาทำให้เกิดผลแล้วแบบเก่า ซึ่งเป็นแบบที่ใช้ในโครงงานทางวิศวกรรม (Senior Project) ในระดับปริญญาตรี
4. แบบที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ด มาทำให้เกิดผลแล้วแบบใหม่ ซึ่งเป็นแบบที่ใช้ในการวิจัยนี้

โปรแกรมเรียงลำดับแบบฟอง (Bubble Sort) แบบดั้งเดิม

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  int size = 0;
  int i;
  int j;
  int tmp;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;
```

```
    for(i = 0; i < size; i++)
    {
      for(j = 0; j < size - 1 - i; j++)
      {
        if(a[j] > a[j+1])
        {
          tmp = a[j+1];
          a[j+1] = a[j];
          a[j] = tmp;
        }
      }
    }
  }
  return 0;
}
```

โปรแกรมเรียงลำดับแบบฟอง (Bubble Sort) แบบดั้งเดิมที่ใช้อินไลน์แอสเซมบลี

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  int size = 0;
  int i;
  int j;
  int tmp;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

  asm volatile
  ("
    movl   $0,%1;
   LOOP_I_COND:
    .long  0xF058858B;
    .short 0xFFFF;
    .long  0xF05C853B;
    .short 0xFFFF;
    jl     INIT_LOOP_J;
    jmp    END_LOOP_J;
   INIT_LOOP_J:
    movl   $0,%2;
   LOOP_J_COND:
    .long  0xF058858B;
    .short 0xFFFF;
```

```
    inc    %%eax;
    .long  0xF05C958B;
    .short 0xFFFF;
    sub    %%eax,%%edx;
    .long  0xF0549539;
    .short 0xFFFF;
    jl     IF_COND;
    jmp    INC_I;
    lea    0x0(%%esi,1),%%esi;
  IF_COND:
    .long  0xF054858B;
    .short 0xFFFF;
    mov    %%eax,%%edx;
    lea    0x0(,%%edx,4),%%eax;
    lea    %0,%%edx;
    .long  0xF0549D8B;
    .short 0xFFFF;
    inc    %%ebx;
    lea    0x0(,%%ebx,4),%%ecx;
    lea    %0,%%ebx;
    .short 0x048B;
    .byte  0x10;
    .short 0x043B;
    .byte  0x19;
    jle    INC_J;
    .long  0xF054858B;
    .short 0xFFFF;
    inc    %%eax;
    lea    0x0(,%%eax,4),%%edx;
    lea    %0,%%eax;
    .short 0x148B;
    .byte  0x02;
    mov    %%edx,%3;
    .long  0xF054958B;
    .short 0xFFFF;
    inc    %%edx;
    lea    0x0(,%%edx,4),%%eax;
    lea    %0,%%edx;
    .long  0xF0548D8B;
    .short 0xFFFF;
    mov    %%ecx,%%ebx;
    lea    0x0(,%%ebx,4),%%ecx;
    lea    %0,%%ebx;
    .short 0x0C8B;
    .byte  0x19;
    mov    %%ecx,(%%eax,%%edx,1);
    .long  0xF054858B;
    .short 0xFFFF;
    mov    %%eax,%%edx;
    lea    0x0(,%%edx,4),%%eax;
    lea    %0,%%edx;
    .long  0xF0508D8B;
    .short 0xFFFF;
    mov    %%ecx,(%%eax,%%edx,1);
  INC_J:
    incl   %2;
    jmp    LOOP_J_COND;
  INC_I:
    incl   %1;
    jmp    LOOP_I_COND;
```

```
    END_LOOP_J:
  "
  :"=m"(a[0])
  :"m"(i),"m"(j),"m"(tmp)
  :"%eax","%ebx","%ecx","%edx","%esi"
  );
  }
  return 0;
}
```

โปรแกรมเรียงลำดับแบบฟอง (Bubble Sort) ที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบเก่า

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  char canWord_a;
  int size = 0;
  char canWord_size;
  int i;
  char canWord_i;
  int j;
  char canWord_j;
  int tmp;
  char canWord_tmp;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

    for(i = 0; read_int(&i) < read_int(&size); i++)
    {
      for(j = 0; read_int(&j) < read_int(&size) - 1 - read_int(&i); j++)
      {
        chkSBit(canWord_a);
        chkSBit(canWord_a);
        if(a[read_int(&j)] > a[read_int(&j)+1])
        {
          chkSBit(canWord_a);
          tmp = a[read_int(&j)+1];
          chkSBit(canWord_a);
          a[read_int(&j)+1] = a[read_int(&j)];
          a[read_int(&j)] = read_int(&tmp);
        }
```

```
      }
    }
  }
  return 0;
}
```

โปรแกรมเรียงลำดับแบบฟอง (Bubble Sort) ที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  char canWord_a;
  int size = 0;
  char canWord_size;
  int i;
  char canWord_i;
  int j;
  char canWord_j;
  int tmp;
  char canWord_tmp;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

  asm volatile
  ("
    movl   $0,%1;
   LOOP_I_COND:
    .long  0x50851B0F;
    .short 0xFFF0;
    .byte  0xFF;
    .long  0x58851C0F;
    .short 0xFFF0;
    .byte  0xFF;
    jl     INIT_LOOP_J;
    jmp    END_LOOP_J;
   INIT_LOOP_J:
    movl   $0,%2;
   LOOP_J_COND:
    .long  0x50851B0F;
    .short 0xFFF0;
    .byte  0xFF;
```

```
    inc    %%eax;
    .long  0x58951B0F;
    .short 0xFFF0;
    .byte  0xFF;
    sub    %%eax,%%edx;
    .long  0x48951D0F;
    .short 0xFFF0;
    .byte  0xFF;
    jl     IF_COND;
    jmp    INC_I;
    lea    0x0(%%esi,1),%%esi;
   IF_COND:
    .long  0x48851B0F;
    .short 0xFFF0;
    .byte  0xFF;
    mov    %%eax,%%edx;
    lea    0x0(,%%edx,4),%%eax;
    lea    %0,%%edx;
    .long  0x489D1B0F;
    .short 0xFFF0;
    .byte  0xFF;
    inc    %%ebx;
    lea    0x0(,%%ebx,4),%%ecx;
    lea    %0,%%ebx;
    .long  0x02041B0F;
    .long  0x0B041C0F;
    jle    INC_J;
    .long  0x48851B0F;
    .short 0xFFF0;
    .byte  0xFF;
    inc    %%eax;
    lea    0x0(,%%eax,4),%%edx;
    lea    %0,%%eax;
    .long  0x10141B0F;
    mov    %%edx,%3;
    .long  0x48951B0F;
    .short 0xFFF0;
    .byte  0xFF;
    inc    %%edx;
    lea    0x0(,%%edx,4),%%eax;
    lea    %0,%%edx;
    .long  0x488D1B0F;
    .short 0xFFF0;
    .byte  0xFF;
    mov    %%ecx,%%ebx;
    lea    0x0(,%%ebx,4),%%ecx;
    lea    %0,%%ebx;
    .long  0x0B0C1B0F;
    mov    %%ecx,(%%eax,%%edx,1);
    .long  0x48851B0F;
    .short 0xFFF0;
    .byte  0xFF;
    mov    %%eax,%%edx;
    lea    0x0(,%%edx,4),%%eax;
    lea    %0,%%edx;
    .long  0x408D1B0F;
    .short 0xFFF0;
    .byte  0xFF;
    mov    %%ecx,(%%eax,%%edx,1);
   INC_J:
```

```
    incl   %2;
    jmp    LOOP_J_COND;
   INC_I:
    incl   %1;
    jmp    LOOP_I_COND;
    END_LOOP_J:
  "
  :"=m"(a[0])
  :"m"(i),"m"(j),"m"(tmp)
  :"%eax","%ebx","%ecx","%edx","%esi"
  );
  }
  return 0;
}
```

โปรแกรมควิกซอร์ต (Quick Sort) แบบดั้งเดิม

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

void quicksort(int a[], int left, int right)
{
  int i;

  if(left < right)
  {
    i = partition(a, left, right);
    quicksort(a, left, i-1);
    quicksort(a, i+1, right);
  }
}
int partition(int a[], int left, int right)
{
  int p;
  int i;
  int j;
  int tmp;

  p = a[left];
  i = left;
  j = right + 1;
  while(i < j)
  {
    j--;
    while(p < a[j])
      j--;
    i++;
    while(a[i] < p)
    {
      if(i == right)
        break;
      i++;
    }
    if(i < j)
    {
      tmp = a[i];
      a[i] = a[j];
      a[j] = tmp;
    }
  }
```

```
  tmp = a[left];
  a[left] = a[j];
  a[j] = tmp;
  return j;
}
int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  int size = 0;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

    quicksort(a, 0, size - 1);
  }
  return 0;
}
```

โปรแกรมควิกซอร์ต (Quick Sort) แบบดั้งเดิมที่ใช้อินไลน์แอสเซมบลี

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

void quicksort(int a[], int left, int right)
{
  int i;

  asm volatile
  ("
    .short 0x458B;
    .byte  0x0C;
    .short 0x453B;
    .byte  0x10;
    jge    END_QUICKSORT;
    .short 0x458B;
    .byte  0x10;
    push   %%eax;
    .short 0x458B;
    .byte  0x0C;
    push   %%eax;
    .short 0x458B;
    .byte  0x08;
    push   %%eax;
    .short 0x458B;
    .byte  0x08;
    call   partition;
```

```
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
    .short 0x458B;
    .byte  0xFC;
    dec    %%eax;
    push   %%eax;
    .short 0x458B;
    .byte  0x0C;
    push   %%eax;
    .short 0x458B;
    .byte  0x08;
    push   %%eax;
    call   quicksort;
    add    $0xc,%%esp;
    .short 0x458B;
    .byte  0x10;
    push   %%eax;
    .short 0x458B;
    .byte  0xFC;
    inc    %%eax;
    push   %%eax;
    .short 0x458B;
    .byte  0x08;
    push   %%eax;
    call   quicksort;
    add    $0xc,%%esp;
   END_QUICKSORT:
  "
  :/*no output*/
  :"m"(i)
  :"%eax","%esp"
  );
}
int partition(int a[], int left, int right)
{
  int p;
  int i;
  int j;
  int tmp;

  asm volatile
  ("
    .short 0x458B;
    .byte  0x0C;
    lea    0x0(,%%eax,4),%%edx;
    .short 0x458B;
    .byte  0x08;
    .short 0x148B;
    .byte  0x10;
    mov    %%edx,%0;
    .short 0x458B;
    .byte  0x0C;
    mov    %%eax,%1;
    .short 0x758B;
    .byte  0x10;
    inc    %%esi;
    mov    %%esi,%2;
    lea    0x0(%%esi,1),%%esi;
   WHILE_IJ_COND:
```

```
    .short 0x458B;
    .byte  0xF8;
    .short 0x453B;
    .byte  0xF4;
    jl     WHILE_IJ_TRUE;
    jmp    END_WHILE_IJ;
    lea    0x0(%%esi),%%esi;
  WHILE_IJ_TRUE:
    decl   %2;
  WHILE_PAJ_COND:
    .short 0x558B;
    .byte  0xF4;
    lea    0x0(,%%edx,4),%%eax;
    .short 0x558B;
    .byte  0x08;
    .short 0x4D8B;
    .byte  0xFC;
    .short 0x0C3B;
    .byte  0x02;
    jl     WHILE_PAJ_TRUE;
    jmp    END_WHILE_PAJ;
    lea    0x0(%%esi),%%esi;
  WHILE_PAJ_TRUE:
    decl   %2;
    jmp    WHILE_PAJ_COND;
  END_WHILE_PAJ:
    incl   %1;
  WHILE_AIP_COND:
    .short 0x458B;
    .byte  0xF8;
    lea    0x0(,%%eax,4),%%edx;
    .short 0x458B;
    .byte  0x08;
    .short 0x148B;
    .byte  0x10;
    .short 0x553B;
    .byte  0xFC;
    jl     IF_IRIGHT_COND;
    jmp    IF_IJ_COND;
  IF_IRIGHT_COND:
    .short 0x458B;
    .byte  0xF8;
    .short 0x453B;
    .byte  0x10;
    jne    END_IF_IRIGHT;
    jmp    IF_IJ_COND;
    lea    0x0(%%esi),%%esi;
  END_IF_IRIGHT:
    incl   %1;
    jmp    WHILE_AIP_COND;
  IF_IJ_COND:
    .short 0x458B;
    .byte  0xF8;
    .short 0x453B;
    .byte  0xF4;
    jge    END_IF_IJ;
    .short 0x458B;
    .byte  0xF8;
    lea    0x0(,%%eax,4),%%edx;
    .short 0x458B;
```

```
    .byte  0x08;
    .short 0x148B;
    .byte  0x10;
    mov    %%edx,%3;
    .short 0x558B;
    .byte  0xF8;
    lea    0x0(,%%edx,4),%%eax;
    .short 0x558B;
    .byte  0x08;
    .short 0x4D8B;
    .byte  0xF4;
    lea    0x0(,%%ecx,4),%%ebx;
    .short 0x4D8B;
    .byte  0x08;
    .short 0x1C8B;
    .byte  0x19;
    mov    %%ebx,(%%edx,%%eax,1);
    .short 0x558B;
    .byte  0xF4;
    lea    0x0(,%%edx,4),%%eax;
    .short 0x558B;
    .byte  0x08;
    .short 0x4D8B;
    .byte  0xF0;
    mov    %%ecx,(%%edx,%%eax,1);
  END_IF_IJ:
    jmp    WHILE_IJ_COND;
  END_WHILE_IJ:
    .short 0x458B;
    .byte  0x0C;
    lea    0x0(,%%eax,4),%%edx;
    .short 0x458B;
    .byte  0x08;
    .short 0x148B;
    .byte  0x10;
    mov    %%edx,%3;
    .short 0x558B;
    .byte  0x0C;
    lea    0x0(,%%edx,4),%%eax;
    .short 0x558B;
    .byte  0x08;
    .short 0x4D8B;
    .byte  0xF4;
    lea    0x0(,%%ecx,4),%%ebx;
    .short 0x4D8B;
    .byte  0x08;
    .short 0x1C8B;
    .byte  0x19;
    mov    %%ebx,(%%edx,%%eax,1);
    .short 0x558B;
    .byte  0xF4;
    lea    0x0(,%%edx,4),%%eax;
    .short 0x558B;
    .byte  0x08;
    .short 0x4D8B;
    .byte  0xF0;
    mov    %%ecx,(%%edx,%%eax,1);
    .short 0x558B;
    .byte  0xF4;
    mov    %%edx,%%eax;
```

```
    jmp    RETURN;
   RETURN:
  "
  :/*no output*/
  :"m"(p),"m"(i),"m"(j),"m"(tmp)
  :"%eax","%ebx","%ecx","%edx","%esi"
  );
}
int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  int size = 0;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

  asm volatile
  ("
    .long  0xF05C858B;
    .short 0xFFFF;
    dec    %%eax;
    push   %%eax;
    push   $0;
    lea    %0,%%eax;
    push   %%eax;
    call   quicksort;
    add    $0xc,%%esp;
  "
  :"=m"(a[0])
  :/*no input*/
  :"%eax","%esp"
  );
  }
  return 0;
}
```

โปรแกรมควิกซอร์ต (Quick Sort) ที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบเก่า

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

void quicksort(int a[], int left, int right)
{
  int i;
  char canWord_i;
```

```
  if(left < right)
  {
    i = partition(a, left, right);
    quicksort(a, left, read_int(&i)-1);
    quicksort(a, read_int(&i)+1, right);
  }
}
int partition(int a[], int left, int right)
{
  int p;
  char canWord_p;
  int i;
  char canWord_i;
  int j;
  char canWord_j;
  int tmp;
  char canWord_tmp;

  read_int(a);
  p = a[left];
  i = left;
  j = right + 1;
  while(read_int(&i) < read_int(&j))
  {
    j--;
    read_int(a);
    while(read_int(&p) < a[read_int(&j)])
    {
      j--;
      read_int(a);
    }
    i++;
    read_int(a);
    while(a[read_int(&i)] < read_int(&p))
    {
      if(read_int(&i) == right)
        break;
      i++;
      read_int(a);
    }
    if(read_int(&i) < read_int(&j))
    {
      read_int(a);
      tmp = a[read_int(&i)];
      read_int(a);
      a[read_int(&i)] = a[read_int(&j)];
      a[read_int(&j)] = read_int(&tmp);
    }
  }

  read_int(a);
  tmp = a[left];
  read_int(a);
  a[left] = a[read_int(&j)];
  a[read_int(&j)] = read_int(&tmp);
  return read_int(&j);
}
int main(int argc, char* argv[])
{
```

```
  int a[ARRAY_SIZE];
  char canWord_a;
  int size = 0;
  char canWord_size;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

    quicksort(a, 0, read_int(&size) - 1);
  }
  return 0;
}
```

โปรแกรมควิกซอร์ต (Quick Sort) ที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่

```
#define ARRAY_SIZE      1000  /*Must be divisible by 2*/
#include <stdio.h> //for using NULL

void quicksort(int a[], int left, int right)
{
  int i;
  char canWord_i;

  asm volatile
  ("
    mov    %1,%%eax;
    cmp    %2,%%eax;
    jge    END_QUICKSORT;
    mov    %2,%%eax;
    push   %%eax;
    mov    %1,%%eax;
    push   %%eax;
    mov    %0,%%eax;
    push   %%eax;
    mov    %0,%%eax;
    call   partition;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%3;
    .long  0xFC451B0F;
    dec    %%eax;
    push   %%eax;
    mov    %1,%%eax;
    push   %%eax;
    mov    %0,%%eax;
```

```
    push   %%eax;
    call   quicksort;
    add    $0xc,%%esp;
    mov    %2,%%eax;
    push   %%eax;
    .long  0xFC451B0F;
    inc    %%eax;
    push   %%eax;
    mov    %0,%%eax;
    push   %%eax;
    call   quicksort;
    add    $0xc,%%esp;
   END_QUICKSORT:
  "
  :"=m"(a)
  :"m"(left),"m"(right),"m"(i)
  :"%eax","%esp"
  );
}
int partition(int a[], int left, int right)
{
  int p;
  char canWord_p;
  int i;
  char canWord_i;
  int j;
  char canWord_j;
  int tmp;
  char canWord_tmp;

  asm volatile
  ("
    mov    %1,%%eax;
    lea    0x0(,%%eax,4),%%edx;
    mov    %0,%%eax;
    .long  0x10141B0F;
    mov    %%edx,%3;
    mov    %1,%%eax;
    mov    %%eax,%4;
    mov    %2,%%esi;
    inc    %%esi;
    mov    %%esi,%5;
    lea    0x0(%%esi,1),%%esi;
   WHILE_IJ_COND:
    .long  0xF4451B0F;
    .long  0xEC451C0F;
    jl     WHILE_IJ_TRUE;
    jmp    END_WHILE_IJ;
    lea    0x0(%%esi),%%esi;
   WHILE_IJ_TRUE:
    decl   %5;
   WHILE_PAJ_COND:
    .long  0xEC551B0F;
    lea    0x0(,%%edx,4),%%eax;
    mov    %0,%%edx;
    .long  0xFC4D1B0F;
    .long  0x020C1C0F;
    jl     WHILE_PAJ_TRUE;
    jmp    END_WHILE_PAJ;
    lea    0x0(%%esi),%%esi;
```

```
    WHILE_PAJ_TRUE:
     decl   %5;
     jmp    WHILE_PAJ_COND;
    END_WHILE_PAJ:
     incl   %4;
    WHILE_AIP_COND:
     .long  0xF4451B0F;
     lea    0x0(,%%eax,4),%%edx;
     mov    %0,%%eax;
     .long  0x10141B0F;
     .long  0xFC551C0F;
     jl     IF_IRIGHT_COND;
     jmp    IF_IJ_COND;
    IF_IRIGHT_COND:
     .long  0xF4451B0F;
     cmp    %2,%%eax;
     jne    END_IF_IRIGHT;
     jmp    IF_IJ_COND;
     lea    0x0(%%esi),%%esi;
    END_IF_IRIGHT:
     incl   %4;
     jmp    WHILE_AIP_COND;
    IF_IJ_COND:
     .long  0xF4451B0F;
     .long  0xEC451C0F;
     jge    END_IF_IJ;
     .long  0xF4451B0F;
     lea    0x0(,%%eax,4),%%edx;
     mov    %0,%%eax;
     .long  0x10141B0F;
     mov    %%edx,%6;
     .long  0xF4551B0F;
     lea    0x0(,%%edx,4),%%eax;
     mov    %0,%%edx;
     .long  0xEC4D1B0F;
     lea    0x0(,%%ecx,4),%%ebx;
     mov    %0,%%ecx;
     .long  0x191C1B0F;
     mov    %%ebx,(%%edx,%%eax,1);
     .long  0xEC551B0F;
     lea    0x0(,%%edx,4),%%eax;
     mov    %0,%%edx;
     .long  0xE44D1B0F;
     mov    %%ecx,(%%edx,%%eax,1);
    END_IF_IJ:
     jmp    WHILE_IJ_COND;
    END_WHILE_IJ:
     mov    %1,%%eax;
     lea    0x0(,%%eax,4),%%edx;
     mov    %0,%%eax;
     .long  0x10141B0F;
     mov    %%edx,%6;
     mov    %1,%%edx;
     lea    0x0(,%%edx,4),%%eax;
     mov    %0,%%edx;
     .long  0xEC4D1B0F;
     lea    0x0(,%%ecx,4),%%ebx;
     mov    %0,%%ecx;
     .long  0x191C1B0F;
     mov    %%ebx,(%%edx,%%eax,1);
```

```
    .long  0xEC551B0F;
    lea    0x0(,%%edx,4),%%eax;
    mov    %0,%%edx;
    .long  0xE44D1B0F;
    mov    %%ecx,(%%edx,%%eax,1);
    .long  0xEC551B0F;
    mov    %%edx,%%eax;
    jmp    RETURN;
   RETURN:
  "
  :"=m"(a)
  :"m"(left),"m"(right),"m"(p),"m"(i),"m"(j),"m"(tmp)
  :"%eax","%ebx","%ecx","%edx","%esi"
  );
}
int main(int argc, char* argv[])
{
  int a[ARRAY_SIZE];
  char canWord_a;
  int size = 0;
  char canWord_size;

  FILE *fp;
  if(argc != 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&a[size++]);
    while(!feof(fp))
      fscanf(fp,"%d",&a[size++]);
    fclose(fp);
    size--;

  asm volatile
  ("
    .long  0x58851B0F;
    .short 0xFFF0;
    .byte  0xFF;
    dec    %%eax;
    push   %%eax;
    push   $0;
    lea    %0,%%eax;
    push   %%eax;
    call   quicksort;
    add    $0xc,%%esp;
  "
  :"=m"(a[0])
  :/*no input*/
  :"%eax","%esp"
  );
  }
  return 0;
}
```

โปรแกรมค้นหาแบบทวิภาค (Binary Search) ด้วยต้นไม้แบบเอวีแอล (AVL Tree) แบบดั้งเดิม

```
#include <stdio.h> //for using NULL
#include <stdlib.h> //for using rand()

struct avlNode
{
  int value;
  struct avlNode *left;
  struct avlNode *right;
  int height;
}; //don't forget this semi-collon =_=!

typedef struct avlNode *Node;

struct avlTree
{
  int size;
  Node root;
};

typedef struct avlTree *Tree;

int getHeight(Node n)
{
  return (n == NULL ? -1 : n->height);
}

void setHeight(Node n)
{
  int lheight;
  int rheight;

  lheight = getHeight(n->left);
  rheight = getHeight(n->right);
  if(lheight > rheight)
    n->height = 1 + lheight;
  else
    n->height = 1 + rheight;
}

int balanceValue(Node n)
{
  return getHeight(n->right) - getHeight(n->left);
}

Node newNode(int v, Node l, Node r)
{
  Node n;
  n = (Node)malloc(sizeof(struct avlNode));

  n->value = v;
  n->left = l;
  n->right = r;
  setHeight(n);
  return n;
}
```

```
Tree newTree()
{
  Tree t;
  t = (Tree)malloc(sizeof(struct avlTree));

  t->size = 0;
  t->root = NULL;
  return t;
}

Node rotateLeftChild(Node n)
{
  Node newRoot;

  if(n->left != NULL)
  {
    newRoot = n->left;
    n->left = newRoot->right;
    newRoot->right = n;
    n = newRoot;
  }
  return n;
}

Node rotateRightChild(Node n)
{
  Node newRoot;

  if(n->right != NULL)
  {
    newRoot = n->right;
    n->right = newRoot->left;
    newRoot->left = n;
    n = newRoot;
  }
  return n;
}

Node rebalance(Node n)
{
  int balance;

  if(n != NULL)
  {
    balance = balanceValue(n);
    if(balance > 1)
    {
      if(balanceValue(n->right) < 0)
        n->right = rotateLeftChild(n->right);
      n = rotateRightChild(n);
    }
    else if(balance < -1)
    {
      if(balanceValue(n->left) > 0)
        n->left = rotateRightChild(n->left);
      n = rotateLeftChild(n);
    }
    if(n != NULL)
      setHeight(n);
  }
```

```
  return n;
}

Node find(Tree t, int v)
{
  Node n;
  int cmp;

  n = t->root;
  while(n != NULL)
  {
    cmp = n->value - v;
    if(cmp > 0)
      n = n->left;
    else if(cmp < 0)
      n = n->right;
    else
      return n;
  }
  return NULL;
}

Node add2(Tree t, Node n, int v)
{
  int cmp;

  if(n == NULL)
  {
    n = newNode(v, NULL, NULL);
    t->size++;
  }
  else
  {
    cmp = n->value - v;
    if(cmp > 0)
      n->left = add2(t, n->left, v);
    else if(cmp < 0)
      n->right = add2(t, n->right, v);
  }
  n = rebalance(n);
  return n;
}

Tree add(Tree t, int v)
{
  if(t == NULL)
    t = newTree();
  t->root = add2(t, t->root, v);
  return t;
}

int main(int argc, char* argv[])
{
  int findData;
  int data;
  Tree t = NULL;

  FILE *fp;
  if(argc < 2)
  {
```

```
    printf("Enter data file name\n");
    return -1;
  }
  else if(argc < 3)
  {
    printf("Enter find data\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&data);
    while(!feof(fp))
    {
      t = add(t,data);
      fscanf(fp,"%d",&data);
    }
    fclose(fp);

    findData = 421;//atoi(argv[2]);
    find(t,findData);
  }
  return 0;
}
```

โปรแกรมค้นหาแบบทวิภาค (Binary Search) ด้วยต้นไม้แบบเอวีแอล (AVL Tree) แบบดั้งเดิมที่ใช้อินไลน์แอสเซมบลี

```
#include <stdio.h> //for using NULL
#include <stdlib.h> //for using rand()

struct avlNode
{
  int value;
  struct avlNode *left;
  struct avlNode *right;
  int height;
}; //don't forget this semi-collon =_=!

typedef struct avlNode *Node;

struct avlTree
{
  int size;
  Node root;
};

typedef struct avlTree *Tree;

int getHeight(Node n)
{
  asm volatile
  ("
    cmpl   $0,%0;
    je     N_EQ_NULL;
    mov    %0,%%edx;
    .short 0x428B;
    .byte  0x0C;
```

```
    jmp    END_N_NULL;
   N_EQ_NULL:
    mov    $0xffffffff,%%eax;
   END_N_NULL:
    jmp    END_GETHEIGHT;
   END_GETHEIGHT:
  "
  :/*no output*/
  :"m"(n)
  :"%eax","%edx"
  );
}

void setHeight(Node n)
{
  int lheight;
  int rheight;

  asm volatile
  ("
    mov    %2,%%eax;
    .short 0x508B;
    .byte  0x04;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
    mov    %2,%%eax;
    .short 0x508B;
    .byte  0x08;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%1;
    .short 0x458B;
    .byte  0xFC;
    .short 0x453B;
    .byte  0xF8;
    jle    ELSE_IF_LHEIGHT_RHEIGHT;
    mov    %2,%%eax;
    .short 0x4D8B;
    .byte  0xFC;
    inc    %%ecx;
    mov    %%ecx,0xc(%%eax);
    jmp    END_SETHEIGHT;
   ELSE_IF_LHEIGHT_RHEIGHT:
    mov    %2,%%eax;
    .short 0x4D8B;
    .byte  0xF8;
    inc    %%ecx;
    mov    %%ecx,0xc(%%eax);
   END_SETHEIGHT:
  "
  :"=m"(lheight),"=m"(rheight)
  :"m"(n)
  :"%eax","%ecx","%edx","%esp"
  );
}
```

```
int balanceValue(Node n)
{
  asm volatile
  ("
    mov    %0,%%eax;
    .short 0x508B;
    .byte  0x08;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%ebx;
    mov    %0,%%eax;
    .short 0x508B;
    .byte  0x04;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%ebx,%%edx;
    sub    %%eax,%%edx;
    mov    %%edx,%%eax;
    jmp    END_BALANCEVALUE;
   END_BALANCEVALUE:
  "
  :/*no output*/
  :"m"(n)
  :"%eax","%ebx","%edx","%esp"
  );
}

Node newNode(int v, Node l, Node r)
{
  Node n;
  n = (Node)malloc(sizeof(struct avlNode));

  asm volatile
  ("
    .short 0x458B;
    .byte  0xFC;
    mov    %0,%%edx;
    mov    %%edx,(%%eax);
    .short 0x458B;
    .byte  0xFC;
    mov    %1,%%edx;
    mov    %%edx,0x4(%%eax);
    .short 0x458B;
    .byte  0xFC;
    mov    %2,%%edx;
    mov    %%edx,0x8(%%eax);
    .short 0x458B;
    .byte  0xFC;
    push   %%eax;
    call   setHeight;
    add    $4,%%esp;
    .short 0x558B;
    .byte  0xFC;
    mov    %%edx,%%eax;
    jmp    END_NEWNODE;
   END_NEWNODE:
```

```
  "
  :/*no output*/
  :"m"(v),"m"(l),"m"(r)
  :"%eax","%edx","%esp"
  );
}

Tree newTree()
{
  Tree t;
  t = (Tree)malloc(sizeof(struct avlTree));

  asm volatile
  ("
    .short 0x458B;
    .byte  0xFC;
    movl   $0,(%%eax);
    .short 0x458B;
    .byte  0xFC;
    movl   $0,0x4(%%eax);
    .short 0x558B;
    .byte  0xFC;
    mov    %%edx,%%eax;
    jmp    END_NEWTREE;
   END_NEWTREE:
  "
  :/*no output*/
  :/*no input*/
  :"%eax","%edx"
  );
}

Node rotateLeftChild(Node n)
{
  Node newRoot;

  asm volatile
  ("
    mov    %1,%%eax;
    cmpl   $0,0x4(%%eax);
    je     END_ROTATELEFTCHILD;
    mov    %1,%%eax;
    .short 0x508B;
    .byte  0x04;
    mov    %%edx,%0;
    mov    %1,%%eax;
    .short 0x558B;
    .byte  0xFC;
    .short 0x4A8B;
    .byte  0x08;
    mov    %%ecx,0x4(%%eax);
    .short 0x458B;
    .byte  0xFC;
    mov    %1,%%edx;
    mov    %%edx,0x8(%%eax);
    .short 0x458B;
    .byte  0xFC;
    mov    %%eax,%1;
    mov    %1,%%edx;
    mov    %%edx,%%eax;
```

```
    jmp    END_ROTATELEFTCHILD;
   END_ROTATELEFTCHILD:
  "
  :"=m"(newRoot)
  :"m"(n)
  :"%eax","%edx"
  );
}

Node rotateRightChild(Node n)
{
  Node newRoot;

  asm volatile
  ("
    mov    %1,%%eax;
    cmpl   $0,0x8(%%eax);
    je     END_ROTATERIGHTCHILD;
    mov    %1,%%eax;
    .short 0x508B;
    .byte  0x08;
    mov    %%edx,%0;
    mov    %1,%%eax;
    .short 0x558B;
    .byte  0xFC;
    .short 0x4A8B;
    .byte  0x04;
    mov    %%ecx,0x8(%%eax);
    .short 0x458B;
    .byte  0xFC;
    mov    %1,%%edx;
    mov    %%edx,0x4(%%eax);
    .short 0x458B;
    .byte  0xFC;
    mov    %%eax,%1;
    mov    %1,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ROTATERIGHTCHILD;
   END_ROTATERIGHTCHILD:
  "
  :"=m"(newRoot)
  :"m"(n)
  :"%eax","%edx"
  );
}

Node rebalance(Node n)
{
  int balance;

  asm volatile
  ("
    cmpl   $0,%1;
    je     END_REBALANCE;
    mov    %1,%%eax;
    push   %%eax;
    call   balanceValue;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
```

```
    cmpl   $1,%0;
    jle    ELSE_BAL_1;
    mov    %1,%%eax;
    .short 0x508B;
    .byte  0x08;
    push   %%edx;
    call   balanceValue;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    test   %%eax,%%eax;
    jge    END_IF_BAL_R_0;
    mov    %1,%%eax;
    .short 0x508B;
    .byte  0x08;
    push   %%edx;
    call   rotateLeftChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %1,%%edx;
    mov    %%eax,0x8(%%edx);
   END_IF_BAL_R_0:
    mov    %1,%%eax;
    push   %%eax;
    call   rotateRightChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%1;
    jmp    END_IF_BAL_1;
    lea    0x0(%%esi),%%esi;
   ELSE_BAL_1:
    cmpl   $0xffffffff,%0;
    jge    END_IF_BAL_1;
    mov    %1,%%eax;
    .short 0x508B;
    .byte  0x04;
    push   %%edx;
    call   balanceValue;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    test   %%eax,%%eax;
    jge    END_IF_BAL_L_0;
    mov    %1,%%eax;
    .short 0x508B;
    .byte  0x04;
    push   %%edx;
    call   rotateRightChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %1,%%edx;
    mov    %%eax,0x4(%%edx);
   END_IF_BAL_L_0:
    mov    %1,%%eax;
    push   %%eax;
    call   rotateLeftChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%1;
   END_IF_BAL_1:
    cmpl   $0,%1;
    je     END_REBALANCE;
```

```
    mov    %1,%%eax;
    push   %%eax;
    call   setHeight;
    add    $4,%%esp;
    mov    %1,%%edx;
    mov    %%edx,%%eax;
   END_REBALANCE:
  "
  :"=m"(balance)
  :"m"(n)
  :"%eax","%edx","%esp","%esi"
  );
}

Node find(Tree t, int v)
{
  Node n;
  int cmp;

  asm volatile
  ("
    mov    %2,%%eax;
    .short 0x508B;
    .byte  0x04;
    mov    %%edx,%0;
    nop;
    lea    0x0(%%esi,1),%%esi;
   FIND_WHILE_N_NULL_COND:
    cmpl   $0,%0;
    jne    FIND_WHILE_N_NULL_TRUE;
    jmp    FIND_END_WHILE_N_NULL;
   FIND_WHILE_N_NULL_TRUE:
    .short 0x458B;
    .byte  0xFC;
    .short 0x108B;
    mov    %3,%%eax;
    mov    %%edx,%%ecx;
    sub    %%eax,%%ecx;
    mov    %%ecx,%1;
    cmpl   $0,%1;
    jle    FIND_ELSE_IF_CMP_0;
    .short 0x458B;
    .byte  0xFC;
    .short 0x508B;
    .byte  0x04;
    mov    %%edx,%0;
    jmp    FIND_END_IF_CMP_0;
   FIND_ELSE_IF_CMP_0:
    cmpl   $0,%1;
    jge    FIND_ELSE_ELSE_IF_CMP_0;
    .short 0x458B;
    .byte  0xFC;
    .short 0x508B;
    .byte  0x08;
    mov    %%edx,%0;
    jmp    FIND_END_IF_CMP_0;
    lea    0x0(%%esi,1),%%esi;
   FIND_ELSE_ELSE_IF_CMP_0:
    .short 0x558B;
    .byte  0xFC;
```

```
    mov    %%edx,%%eax;
    jmp    END_FIND;
   FIND_END_IF_CMP_0:
    jmp    FIND_WHILE_N_NULL_COND;
    lea    0x0(%%esi),%%esi;
   FIND_END_WHILE_N_NULL:
    xor    %%eax,%%eax;
    jmp    END_FIND;
   END_FIND:
  "
  :"=m"(n),"=m"(cmp)
  :"m"(t),"m"(v)
  :"%eax","%ecx","%edx","%esi"
  );
}

Node add2(Tree t, Node n, int v)
{
  int cmp;

  asm volatile
  ("
    cmpl   $0,%2;
    jne    ADD2_ELSE_N_NULL;
    push   $0;
    push   $0;
    mov    %3,%%eax;
    push   %%eax;
    call   newNode;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%2;
    mov    %1,%%eax;
    incl   (%%eax);
    jmp    ADD2_END_IF_N_NULL;
   ADD2_ELSE_N_NULL:
    mov    %2,%%eax;
    .short 0x108B;
    mov    %3,%%eax;
    mov    %%edx,%%ecx;
    sub    %%eax,%%ecx;
    mov    %%ecx,%0;
    cmpl   $0,%0;
    jle    ADD2_ELSE_IF_CMP_0;
    mov    %3,%%eax;
    push   %%eax;
    mov    %2,%%eax;
    .short 0x508B;
    .byte  0x04;
    push   %%edx;
    mov    %1,%%eax;
    push   %%eax;
    call   add2;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %2,%%edx;
    mov    %%eax,0x4(%%edx);
    jmp    ADD2_END_IF_N_NULL;
   ADD2_ELSE_IF_CMP_0:
    cmpl   $0,%0;
```

```
    jge    ADD2_END_IF_N_NULL;
    mov    %3,%%eax;
    push   %%eax;
    mov    %2,%%eax;
    .short 0x508B;
    .byte  0x08;
    push   %%edx;
    mov    %1,%%eax;
    push   %%eax;
    call   add2;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %2,%%edx;
    mov    %%eax,0x8(%%edx);
   ADD2_END_IF_N_NULL:
    mov    %2,%%eax;
    push   %%eax;
    call   rebalance;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%2;
    mov    %2,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ADD2;
   END_ADD2:
  "
  :"=m"(cmp)
  :"m"(t),"m"(n),"m"(v)
  :"%eax","%ecx","%edx","%esp","%esi"
  );
}

Tree add(Tree t, int v)
{
  asm volatile
  ("
    cmpl   $0,%0;
    jne    END_IF_T_NULL;
    call   newTree;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
   END_IF_T_NULL:
    mov    %1,%%eax;
    push   %%eax;
    mov    %0,%%eax;
    .short 0x508B;
    .byte  0x04;
    push   %%edx;
    mov    %0,%%eax;
    push   %%eax;
    call   add2;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %0,%%edx;
    mov    %%eax,0x4(%%edx);
    mov    %0,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ADD;
   END_ADD:
  "
```

```
  :/*no output*/
  :"m"(t),"m"(v)
  :"%eax","%edx","%esp"
  );
}

void printNode(Node n)
{
  if(n!=NULL)
  {
    printNode(n->left);
    printf("%d ",n->value);
    printNode(n->right);
  }
}
int main(int argc, char* argv[])
{
  int findData;
  int data;
  Tree t = NULL;

  FILE *fp;
  if(argc < 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  else if(argc < 3)
  {
    printf("Enter find data\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&data);
    while(!feof(fp))
    {
      asm volatile
      ("
        .short 0x458B;
        .byte  0xF8;
        push   %%eax;
        .short 0x458B;
        .byte  0xF4;
        push   %%eax;
        call   add;
        add    $8,%%esp;
        mov    %%eax,%%eax;
        mov    %%eax,%0;
      "
      :"=m"(t)
      :/*no input*/
      :"%eax","%esp"
      );
      fscanf(fp,"%d",&data);
    }
    fclose(fp);
```

```
    findData = atoi(argv[2]);
    asm volatile
    ("
      .short 0x458B;
      .byte  0xFC;
      push   %%eax;
      .short 0x458B;
      .byte  0xF4;
      push   %%eax;
      call   find;
      add    $8,%%esp;
    "
    :/*no output*/
    :/*no input*/
    :"%eax","%esp"
    );
  }
  return 0;
}
```

โปรแกรมค้นหาแบบทวิภาค (Binary Search) ด้วยต้นไม้แบบเอวีแอล (AVL Tree) ที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบเก่า

```
#include <stdio.h> //for using NULL
#include <stdlib.h> //for using rand()

struct avlNode
{
  int value;
  char canWord_value;
  struct avlNode *left;
  char canWord_left;
  struct avlNode *right;
  char canWord_right;
  int height;
  char canWord_height;
}; //don't forget this semi-collon =_=!

typedef struct avlNode *Node;

struct avlTree
{
  int size;
  char canWord_size;
  Node root;
  char canWord_root;
};

typedef struct avlTree *Tree;

int getHeight(Node n)
{
  return (n == NULL ? -1 : read_int(&(n->height)));
}

void setHeight(Node n)
{
  int lheight;
  char canWord_lheight;
```

```
  int rheight;
  char canWord_rheight;

  lheight = getHeight(*((Node*)read(&(n->left))));
  rheight = getHeight(*((Node*)read(&(n->right))));
  if(read_int(&lheight) > read_int(&rheight))
    n->height = 1 + read_int(&lheight);
  else
    n->height = 1 + read_int(&rheight);
}

int balanceValue(Node n)
{
  return getHeight(*((Node*)read(&(n->right)))) -
getHeight(*((Node*)read(&(n->left))));
}

Node newNode(int v, Node l, Node r)
{
  Node n;
  char canWord_n;
  n = (Node)malloc(sizeof(struct avlNode));

  (*((Node*)read(&n)))->value = v;
  (*((Node*)read(&n)))->left = l;
  (*((Node*)read(&n)))->right = r;
  setHeight(*((Node*)read(&n)));
  return *((Node*)read(&n));
}

Tree newTree()
{
  Tree t;
  char canWord_t;
  t = (Tree)malloc(sizeof(struct avlTree));

  (*((Tree*)read(&t)))->size = 0;
  (*((Tree*)read(&t)))->root = NULL;
  return *((Tree*)read(&t));
}

Node rotateLeftChild(Node n)
{
  Node newRoot;
  char canWord_newRoot;

  if(*((Node*)read(&(n->left))) != NULL)
  {
    newRoot = *((Node*)read(&(n->left)));
    n->left = *((Node*)read(&((*((Node*)read(&newRoot)))->right)));
    (*((Node*)read(&newRoot)))->right = n;
    n = *((Node*)read(&newRoot));
  }
  return n;
}

Node rotateRightChild(Node n)
{
  Node newRoot;
  char canWord_newRoot;
```

```
  if(*((Node*)read(&(n->right))) != NULL)
  {
    newRoot = *((Node*)read(&(n->right)));
    n->right = *((Node*)read(&((*((Node*)read(&newRoot)))->left)));
    (*((Node*)read(&newRoot)))->left = n;
    n = *((Node*)read(&newRoot));
  }
  return n;
}

Node rebalance(Node n)
{
  int balance;
  char canWord_balance;

  if(n != NULL)
  {
    balance = balanceValue(n);
    if(read_int(&balance) > 1)
    {
      if(balanceValue(*((Node*)read(&(n->right)))) < 0)
        n->right = rotateLeftChild(*((Node*)read(&(n->right))));
      n = rotateRightChild(n);
    }
    else if(read_int(&balance) < -1)
    {
      if(balanceValue(*((Node*)read(&(n->left)))) > 0)
        n->left = rotateRightChild(*((Node*)read(&(n->left))));
      n = rotateLeftChild(n);
    }
    if(n != NULL)
      setHeight(n);
  }
  return n;
}

Node find(Tree t, int v)
{
  Node n;
  char canWord_n;
  int cmp;
  char canWord_cmp;

  n = *((Node*)read(&(t->root)));
  while(*((Node*)read(&n)) != NULL)
  {
    cmp = read_int(&((*((Node*)read(&n)))->value)) - v;
    if(read_int(&cmp) > 0)
      n = *((Node*)read(&((*((Node*)read(&n)))->left)));
    else if(read_int(&cmp) < 0)
      n = *((Node*)read(&((*((Node*)read(&n)))->right)));
    else
      return *((Node*)read(&n));
  }
  return NULL;
}

Node add2(Tree t, Node n, int v)
{
```

```
  int cmp;
  char canWord_cmp;

  if(n == NULL)
  {
    n = newNode(v, NULL, NULL);
    t->size++;
  }
  else
  {
    cmp = read_int(&(n->value)) - v;
    if(read_int(&cmp) > 0)
      n->left = add2(t, *((Node*)read(&(n->left))), v);
    else if(read_int(&cmp) < 0)
      n->right = add2(t, *((Node*)read(&(n->right))), v);
  }
  n = rebalance(n);
  return n;
}

Tree add(Tree t, int v)
{
  if(t == NULL)
    t = newTree();
  t->root = add2(t, *((Node*)read(&(t->root))), v);
  return t;
}

int main(int argc, char* argv[])
{
  int findData;
  char canWord_findData;
  int data;
  char canWord_data;
  Tree t = NULL;
  char canWord_t;

  FILE *fp;
  if(argc < 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  else if(argc < 3)
  {
    printf("Enter find data \n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&data);
    while(!feof(fp))
    {
      t = add(*((Tree*)read(&t)),read_int(&data));
      fscanf(fp,"%d",&data);
    }
    fclose(fp);
```

```
    findData = 421;//atoi(argv[2]);
    find(*((Tree*)read(&t)),read_int(&findData));
  }
  return 0;
}
```

โปรแกรมค้นหาแบบทวิภาค (Binary Search) ด้วยต้นไม้แบบเอวีแอล (AVL Tree) ที่นำวิธีการป้องกันบัฟเฟอร์โอเวอร์โฟลว์แบบซีเคียวคานารีเวิร์ดมาทำให้เกิดผลแล้วแบบใหม่

```
#include <stdio.h> //for using NULL
#include <stdlib.h> //for using rand()

struct avlNode
{
  int value;
  char canWord_value;
  struct avlNode *left;
  char canWord_left;
  struct avlNode *right;
  char canWord_right;
  int height;
  char canWord_height;
}; //don't forget this semi-collon =_=!

typedef struct avlNode *Node;

struct avlTree
{
  int size;
  char canWord_size;
  Node root;
  char canWord_root;
};

typedef struct avlTree *Tree;

int getHeight(Node n)
{
  asm volatile
  ("
    cmpl   $0,%0;
    je     N_EQ_NULL;
    mov    %0,%%edx;
    .long  0x18421B0F;
    jmp    END_N_NULL;
   N_EQ_NULL:
    mov    $0xffffffff,%%eax;
   END_N_NULL:
    jmp    END_GETHEIGHT;
   END_GETHEIGHT:
  "
  :/*no output*/
  :"m"(n)
  :"%eax","%edx"
  );
}

void setHeight(Node n)
{
```

```
  int lheight;
  char canWord_lheight;
  int rheight;
  char canWord_rheight;

  asm volatile
  ("
    mov    %2,%%eax;
    .long  0x08501B0F;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
    mov    %2,%%eax;
    .long  0x10501B0F;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%1;
    .long  0xFC451B0F;
    .long  0xF4451C0F;
    jle    ELSE_IF_LHEIGHT_RHEIGHT;
    mov    %2,%%eax;
    .long  0xFC4D1B0F;
    inc    %%ecx;
    mov    %%ecx,0x18(%%eax);
    jmp    END_SETHEIGHT;
   ELSE_IF_LHEIGHT_RHEIGHT:
    mov    %2,%%eax;
    .long  0xF44D1B0F;
    inc    %%ecx;
    mov    %%ecx,0x18(%%eax);
   END_SETHEIGHT:
  "
  :"=m"(lheight),"=m"(rheight)
  :"m"(n)
  :"%eax","%ecx","%edx","%esp"
  );
}

int balanceValue(Node n)
{
  asm volatile
  ("
    mov    %0,%%eax;
    .long  0x10501B0F;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%ebx;
    mov    %0,%%eax;
    .long  0x08501B0F;
    push   %%edx;
    call   getHeight;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%ebx,%%edx;
    sub    %%eax,%%edx;
```

```
    mov    %%edx,%%eax;
    jmp    END_BALANCEVALUE;
   END_BALANCEVALUE:
  "
  :/*no output*/
  :"m"(n)
  :"%eax","%ebx","%edx","%esp"
  );
}

Node newNode(int v, Node l, Node r)
{
  Node n;
  char canWord_n;
  n = (Node)malloc(sizeof(struct avlNode));

  asm volatile
  ("
    .long  0xFC451B0F;
    mov    %0,%%edx;
    mov    %%edx,(%%eax);
    .long  0xFC451B0F;
    mov    %1,%%edx;
    mov    %%edx,0x8(%%eax);
    .long  0xFC451B0F;
    mov    %2,%%edx;
    mov    %%edx,0x10(%%eax);
    .long  0xFC451B0F;
    push   %%eax;
    call   setHeight;
    add    $4,%%esp;
    .long  0xFC551B0F;
    mov    %%edx,%%eax;
    jmp    END_NEWNODE;
   END_NEWNODE:
  "
  :/*no output*/
  :"m"(v),"m"(l),"m"(r)
  :"%eax","%edx","%esp"
  );
}

Tree newTree()
{
  Tree t;
  char canWord_t;
  t = (Tree)malloc(sizeof(struct avlTree));

  asm volatile
  ("
    .long  0xFC451B0F;
    movl   $0,(%%eax);
    .long  0xFC451B0F;
    movl   $0,0x8(%%eax);
    .long  0xFC551B0F;
    mov    %%edx,%%eax;
    jmp    END_NEWTREE;
   END_NEWTREE:
  "
  :/*no output*/
```

```
  :/*no input*/
  :"%eax","%edx"
  );
}

Node rotateLeftChild(Node n)
{
  Node newRoot;
  char canWord_newRoot;

  asm volatile
  ("
    mov    %1,%%eax;
    .long  0x08781E0F;
    .byte  0x00;
    je     END_ROTATELEFTCHILD;
    mov    %1,%%eax;
    .long  0x08501B0F;
    mov    %%edx,%0;
    mov    %1,%%eax;
    .long  0xFC551B0F;
    .long  0x104A1B0F;
    mov    %%ecx,0x8(%%eax);
    .long  0xFC451B0F;
    mov    %1,%%edx;
    mov    %%edx,0x10(%%eax);
    .long  0xFC451B0F;
    mov    %%eax,%1;
    mov    %1,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ROTATELEFTCHILD;
   END_ROTATELEFTCHILD:
  "
  :"=m"(newRoot)
  :"m"(n)
  :"%eax","%edx"
  );
}

Node rotateRightChild(Node n)
{
  Node newRoot;
  char canWord_newRoot;

  asm volatile
  ("
    mov    %1,%%eax;
    .long  0x10781E0F;
    .byte  0x00;
    je     END_ROTATERIGHTCHILD;
    mov    %1,%%eax;
    .long  0x10501B0F;
    mov    %%edx,%0;
    mov    %1,%%eax;
    .long  0xFC551B0F;
    .long  0x084A1B0F;
    mov    %%ecx,0x10(%%eax);
    .long  0xFC451B0F;
    mov    %1,%%edx;
    mov    %%edx,0x8(%%eax);
```

```
    .long  0xFC451B0F;
    mov    %%eax,%1;
    mov    %1,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ROTATERIGHTCHILD;
   END_ROTATERIGHTCHILD:
  "
  :"=m"(newRoot)
  :"m"(n)
  :"%eax","%edx"
  );
}

Node rebalance(Node n)
{
  int balance;
  char canWord_balance;

  asm volatile
  ("
    cmpl   $0,%1;
    je     END_REBALANCE;
    mov    %1,%%eax;
    push   %%eax;
    call   balanceValue;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
    .long  0xFC7D1E0F;
    .byte  0x01;
    jle    ELSE_BAL_1;
    mov    %1,%%eax;
    .long  0x10501B0F;
    push   %%edx;
    call   balanceValue;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    test   %%eax,%%eax;
    jge    END_IF_BAL_R_0;
    mov    %1,%%eax;
    .long  0x10501B0F;
    push   %%edx;
    call   rotateLeftChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %1,%%edx;
    mov    %%eax,0x10(%%edx);
   END_IF_BAL_R_0:
    mov    %1,%%eax;
    push   %%eax;
    call   rotateRightChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%1;
    jmp    END_IF_BAL_1;
    lea    0x0(%%esi),%%esi;
   ELSE_BAL_1:
    .long  0xFC7D1E0F;
    .byte  0xFF;
    jge    END_IF_BAL_1;
```

```
    mov    %1,%%eax;
    .long  0x08501B0F;
    push   %%edx;
    call   balanceValue;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    test   %%eax,%%eax;
    jge    END_IF_BAL_L_0;
    mov    %1,%%eax;
    .long  0x08501B0F;
    push   %%edx;
    call   rotateRightChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %1,%%edx;
    mov    %%eax,0x8(%%edx);
   END_IF_BAL_L_0:
    mov    %1,%%eax;
    push   %%eax;
    call   rotateLeftChild;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%1;
   END_IF_BAL_1:
    cmpl   $0,%1;
    je     END_REBALANCE;
    mov    %1,%%eax;
    push   %%eax;
    call   setHeight;
    add    $4,%%esp;
    mov    %1,%%edx;
    mov    %%edx,%%eax;
   END_REBALANCE:
  "
  :"=m"(balance)
  :"m"(n)
  :"%eax","%edx","%esp","%esi"
  );
}

Node find(Tree t, int v)
{
  Node n;
  char canWord_n;
  int cmp;
  char canWord_cmp;

  asm volatile
  ("
    mov    %2,%%eax;
    .long  0x08501B0F;
    mov    %%edx,%0;
    nop;
    lea    0x0(%%esi,1),%%esi;
   FIND_WHILE_N_NULL_COND:
    .long  0xFC7D1E0F;
    .byte  0x00;
    jne    FIND_WHILE_N_NULL_TRUE;
    jmp    FIND_END_WHILE_N_NULL;
   FIND_WHILE_N_NULL_TRUE:
```

```
    .long  0xFC451B0F;
    .short 0x1B0F;
    .byte  0x10;
    mov    %3,%%eax;
    mov    %%edx,%%ecx;
    sub    %%eax,%%ecx;
    mov    %%ecx,%1;
    .long  0xF47D1E0F;
    .byte  0x00;
    jle    FIND_ELSE_IF_CMP_0;
    .long  0xFC451B0F;
    .long  0x08501B0F;
    mov    %%edx,%0;
    jmp    FIND_END_IF_CMP_0;
   FIND_ELSE_IF_CMP_0:
    .long  0xF47D1E0F;
    .byte  0x00;
    jge    FIND_ELSE_ELSE_IF_CMP_0;
    .long  0xFC451B0F;
    .long  0x10501B0F;
    mov    %%edx,%0;
    jmp    FIND_END_IF_CMP_0;
    lea    0x0(%%esi,1),%%esi;
   FIND_ELSE_ELSE_IF_CMP_0:
    .long  0xFC551B0F;
    mov    %%edx,%%eax;
    jmp    END_FIND;
   FIND_END_IF_CMP_0:
    jmp    FIND_WHILE_N_NULL_COND;
    lea    0x0(%%esi),%%esi;
   FIND_END_WHILE_N_NULL:
    xor    %%eax,%%eax;
    jmp    END_FIND;
   END_FIND:
  "
  :"=m"(n),"=m"(cmp)
  :"m"(t),"m"(v)
  :"%eax","%ecx","%edx","%esi"
  );
}

Node add2(Tree t, Node n, int v)
{
  int cmp;
  char canWord_cmp;

  asm volatile
  ("
    cmpl   $0,%2;
    jne    ADD2_ELSE_N_NULL;
    push   $0;
    push   $0;
    mov    %3,%%eax;
    push   %%eax;
    call   newNode;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%2;
    mov    %1,%%eax;
    incl   (%%eax);
```

```
    jmp    ADD2_END_IF_N_NULL;
   ADD2_ELSE_N_NULL:
    mov    %2,%%eax;
    .short 0x1B0F;
    .byte  0x10;
    mov    %3,%%eax;
    mov    %%edx,%%ecx;
    sub    %%eax,%%ecx;
    mov    %%ecx,%0;
    .long  0xFC7D1E0F;
    .byte  0x00;
    jle    ADD2_ELSE_IF_CMP_0;
    mov    %3,%%eax;
    push   %%eax;
    mov    %2,%%eax;
    .long  0x08501B0F;
    push   %%edx;
    mov    %1,%%eax;
    push   %%eax;
    call   add2;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %2,%%edx;
    mov    %%eax,0x8(%%edx);
    jmp    ADD2_END_IF_N_NULL;
   ADD2_ELSE_IF_CMP_0:
    .long  0xFC7D1E0F;
    .byte  0x00;
    jge    ADD2_END_IF_N_NULL;
    mov    %3,%%eax;
    push   %%eax;
    mov    %2,%%eax;
    .long  0x10501B0F;
    push   %%edx;
    mov    %1,%%eax;
    push   %%eax;
    call   add2;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %2,%%edx;
    mov    %%eax,0x10(%%edx);
   ADD2_END_IF_N_NULL:
    mov    %2,%%eax;
    push   %%eax;
    call   rebalance;
    add    $4,%%esp;
    mov    %%eax,%%eax;
    mov    %%eax,%2;
    mov    %2,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ADD2;
   END_ADD2:
  "
  :"=m"(cmp)
  :"m"(t),"m"(n),"m"(v)
  :"%eax","%ecx","%edx","%esp","%esi"
  );
}

Tree add(Tree t, int v)
```

```
{
  asm volatile
  ("
    cmpl   $0,%0;
    jne    END_IF_T_NULL;
    call   newTree;
    mov    %%eax,%%eax;
    mov    %%eax,%0;
   END_IF_T_NULL:
    mov    %1,%%eax;
    push   %%eax;
    mov    %0,%%eax;
    .long  0x08501B0F;
    push   %%edx;
    mov    %0,%%eax;
    push   %%eax;
    call   add2;
    add    $0xc,%%esp;
    mov    %%eax,%%eax;
    mov    %0,%%edx;
    mov    %%eax,0x8(%%edx);
    mov    %0,%%edx;
    mov    %%edx,%%eax;
    jmp    END_ADD;
   END_ADD:
  "
  :/*no output*/
  :"m"(t),"m"(v)
  :"%eax","%edx","%esp"
  );
}

int main(int argc, char* argv[])
{
  int findData;
  char canWord_findData;
  int data;
  char canWord_data;
  Tree t = NULL;
  char canWord_t;
  Node n = NULL;

  FILE *fp;
  if(argc < 2)
  {
    printf("Enter data file name\n");
    return -1;
  }
  else if(argc < 3)
  {
    printf("Enter find data\n");
    return -1;
  }
  if((fp = fopen(argv[1],"r")) == NULL)
    printf("File not found\n");
  else
  {
    fscanf(fp,"%d",&data);
    while(!feof(fp))
    {
```

```
      asm volatile
      ("
        .long  0xF4451B0F;
        push   %%eax;
        .long  0xEC451B0F;
        push   %%eax;
        call   add;
        add    $8,%%esp;
        mov    %%eax,%%eax;
        mov    %%eax,%0;
      "
      :"=m"(t)
      :/*no input*/
      :"%eax","%esp"
      );
      fscanf(fp,"%d",&data);
    }
    fclose(fp);

    findData = 998;//atoi(argv[2]);
    asm volatile
    ("
      .long  0xFC451B0F;
      push   %%eax;
      .long  0xEC451B0F;
      push   %%eax;
      call   find;
      add    $8,%%esp;
    "
    :/*no output*/
    :/*no input*/
    :"%eax","%esp"
    );
  }
  return 0;
}
```

ภาคผนวก ง.
ข้อมูลที่ใช้ทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่าง ๆ

ในภาคผนวก ง. นี้ จะแสดงข้อมูลที่ใช้ทดลองวัดประสิทธิภาพที่มีผลกระทบต่อการทำงานของโปรแกรมต่างๆที่ใช้ในการวิจัยนี้ ซึ่งเป็นตัวเลขที่มีค่าระหว่าง 0 ถึง 999 จำนวนทั้งหมด 1,000 ข้อมูล ดังต่อไปนี้

383	795	776	90	528	261	270	88	272	932
886	570	404	684	189	42	699	919	122	382
777	434	443	376	872	360	417	710	154	21
915	378	763	542	908	117	839	732	335	266
793	467	613	936	958	23	569	294	821	563
335	601	538	107	498	761	363	17	457	860
386	97	606	445	36	81	622	346	365	682
492	902	840	756	808	309	794	235	747	211
649	317	904	179	753	190	173	137	171	685
421	492	818	418	248	425	847	691	776	86
362	652	128	887	303	996	431	153	269	285
27	756	688	412	333	367	462	943	218	930
690	301	369	348	133	677	682	573	701	990
59	280	917	172	648	234	390	328	703	583
763	286	917	659	890	690	292	925	653	314
926	441	996	9	754	626	791	291	933	476
540	865	324	336	567	524	57	710	907	116
426	689	743	210	746	57	115	18	959	820
172	444	470	342	368	614	521	217	728	892
736	619	183	587	529	168	157	836	806	525
211	440	490	206	500	205	574	963	797	528
368	729	499	301	46	358	491	55	720	839
567	31	772	713	788	312	947	90	84	525
429	117	725	372	797	386	951	858	308	490
782	97	644	321	249	100	231	130	334	136
530	771	590	255	990	346	21	904	698	360
862	481	505	819	303	726	537	571	991	618
123	675	139	599	33	994	740	661	376	643
67	709	954	721	363	916	54	633	898	337
135	927	786	904	497	552	30	685	715	928
929	567	669	939	253	578	98	789	52	582
802	856	82	811	892	529	325	73	171	621
22	497	542	940	686	946	81	604	189	310
58	353	464	667	125	290	516	851	559	955
69	586	197	705	152	647	516	805	506	888
167	965	507	228	996	970	2	250	10	225
393	306	355	127	975	51	231	868	16	815
456	683	804	150	188	80	139	503	224	570
11	219	348	984	157	631	796	485	109	437
42	624	611	658	729	593	404	6	539	853
229	528	622	920	436	857	338	195	0	8
373	871	828	224	460	627	580	639	378	722
421	732	299	422	414	312	218	949	109	783
919	829	343	269	921	886	21	120	53	350
784	503	746	396	460	214	970	967	81	657
537	19	568	81	304	355	862	226	114	97

198	270	340	630	28	512	812	763	338	827
324	368	422	84	27	90	379	677	989	126
315	708	311	292	50	412	977	596	426	269
370	715	810	972	748	479	685	981	67	71
413	340	605	672	556	610	536	865	147	651
526	149	801	850	902	969	904	560	223	149
91	796	661	625	794	189	176	36	787	910
980	723	730	385	697	274	483	955	231	528
956	618	878	222	699	355	207	770	532	639
873	245	305	299	43	641	759	518	122	398
862	846	320	640	39	620	857	211	281	888
170	451	736	42	2	433	744	342	875	610
996	921	444	898	428	987	499	532	850	393
281	555	626	713	403	888	911	196	179	577
305	379	522	298	500	338	127	379	590	890
925	488	465	190	681	566	950	321	254	976
84	764	708	524	647	770	236	270	350	199
327	228	416	590	538	284	560	984	131	552
336	841	282	209	159	856	818	172	813	931
505	350	258	581	151	417	105	427	857	87
846	193	924	819	535	606	563	234	494	777
729	500	637	336	134	260	49	40	181	99
313	34	62	732	339	849	244	283	81	657
857	764	624	155	692	237	711	72	603	566
124	124	600	994	215	205	805	398	720	952
895	914	36	4	127	59	934	830	433	17
582	987	452	379	504	217	291	63	982	641
545	856	899	769	629	518	375	347	181	735
814	743	379	273	49	945	955	950	487	368
367	491	550	776	964	783	614	30	415	298
434	227	468	850	285	873	589	573	296	184
364	365	71	255	429	458	768	714	825	195
43	859	973	860	343	873	993	59	404	776
750	936	131	142	335	637	918	522	722	805
87	432	881	579	177	289	805	47	892	266
808	551	930	884	900	483	882	924	551	428
276	437	933	993	238	607	822	82	297	954
178	228	894	205	971	478	982	435	32	528
788	275	660	621	949	757	717	232	134	308
584	407	163	567	289	314	30	204	181	593
403	474	199	504	367	471	93	954	506	278
651	121	981	613	988	729	574	443	415	197
754	858	899	961	292	100	126	898	57	555
399	395	996	754	795	459	593	486	708	672
932	29	959	326	743	618	486	640	595	774
60	237	773	259	144	438	253	278	999	445
676	235	813	944	829	25	543	159	962	0
368	793	668	202	390	388	74	262	297	325
739	818	190	202	682	74	814	262	483	997
12	428	95	506	340	233	713	683	776	283
226	143	926	784	541	157	179	41	154	412
586	11	466	21	569	681	377	848	977	127
94	928	84	842	826	493	762	723	309	382
539	529	340	868	232	358	775	324	587	421

ภาคผนวก จ.
ขั้นตอนการสร้าง (Make)

ในภาคผนวก จ. นี้ จะอธิบายขั้นตอนการสร้างซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์ (BOCHS Emulator) [58] และคอมไพเลอร์ (Compiler) ชื่อ จีซีซี (GCC) [59] ดังต่อไปนี้

ขั้นตอนการสร้างซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์

เมื่อเข้าไปในไดเร็กทอรี (Directory) ที่มีโค้ดของซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อ โบชส์ (สามารถดาว์นโหลดโค้ดได้ที่ [58]) โดยอาศัยซอฟต์แวร์จำลองการทำงานบนระบบปฏิบัติการยูนิกซ์ (Unix Operating System)ชื่อ ซิกวิน (Cygwin) แล้ว มีขั้นตอนการสร้างใหม่ ดังนี้

```
# source configure
# makefile
# make
```

ขั้นตอนการสร้างคอมไพเลอร์ชื่อ จีซีซี

เนื่องจากซอฟต์แวร์เลียนแบบการทำงานของอุปกรณ์ฮาร์ดแวร์ชื่อโบชส์นี้ใช้ไฟล์อิมเมจ (Image File) แทนจานบันทึกแบบแข็ง (Hard Disk) ของจริง โดยที่ติดตั้งระบบปฏิบัติการลีนุกซ์ (Linux Operating System) ไว้ ดังนั้น ถ้าต้องการนำโค้ดของคอมไพเลอร์ชื่อ จีซีซี (สามารถดาว์นโหลดโค้ดได้ที่ [59] โดยเลือกให้เหมาะสมกับระบบปฏิบัติการที่จะนำไปใช้) มาใส่ลงในไฟล์อิมเมจนี้ โดยกระทำบนระบบปฏิบัติการวินโดวส์ (Windows Operating System) จะต้องทำตามขั้นตอนดังต่อไปนี้

1. ตั้งการอนุญาต (Permission) ให้ไฟล์อิมเมจนี้สามารถอ่านและเขียนได้โดยผู้ใช้ (User) ของระบบปฏิบัติการวินโดวส์ (สมมติ ไฟล์อิมเมจชื่อ c.img และผู้ใช้ชื่อ user)
2. สั่งแชร์โฟลเดอร์ (Share Folder) ที่มีไฟล์อิมเมจนี้และโค้ดของคอมไพเลอร์ชื่อจีซีซี โดยตั้งชื่อโฟลเดอร์ด้วย (สมมติ ตั้งชื่อ WindowsFolder)
3. ใช้โปรแกรมเครื่องเสมือน (Virtual Machine) ตั้งค่าไอพีแอดเดรส (IP Address) ของเครื่องคอมพิวเตอร์จริงบนระบบปฏิบัติการลีนุกซ์ในโปรแกรมเครื่องเสมือน โดยแก้ไขไฟล์ชื่อ /etc/resolv.conf ให้เพิ่มบรรทัดต่อท้ายไฟล์ว่า “nameserver 192.168.19.2” (สมมติ ไอพีแอดเดรสคือ 192.168.19.2)

4. จากนั้นทำการใส่ (Mount) โฟล์เดอร์ที่แชร์ไว้เข้าไปในโปรแกรมเสมือน โดยมีขั้นตอนการสั่งดังนี้

```
# cd /mnt
# mkdir data
# mount.cifs //192.168.19.2/WindowsFolder data -ouser=user
```

5. ทำการใส่ (Mount) ไฟล์อิมเมจนี้เข้าไปในโปรแกรมเสมือนเช่นกัน โดยมีขั้นตอนการสั่งดังนี้

```
# cd /mnt
# mkdir bochsHdd
# losetup -o 32256 /dev/loop0 c.img
# mount /dev/loop0 /mnt/bochsHdd
```

6. สั่งคัดลอก (Copy) โค้ดเข้าไปไฟล์อิมเมจ ดังนี้

```
# mnt/data/gcc-core-1.1.2.tar.gz /mnt/bochsHdd
```

แต่หากกระทำบนระบบปฏิบัติการลีนุกซ์ สามารถสั่งดาว์นโหลดโค้ดมาได้ตามปกติ เมื่อนำโค้ดของคอมไพเลอร์ชื่อจีซีซีมาใส่ในไฟล์อิมเมจเรียบร้อยแล้ว มีขั้นตอนการสร้างคอมไพเลอร์ชื่อจีซีซีใหม่ (ในที่นี้คือเวอร์ชัน egcs-2.91.66) ดังนี้

```
# tar -xzvf egcs-core-1.1.2.tar.gz
# cd egcs-1.1.2
# ./configure --host=i386-linux
# make
# make install
```

ข้อควรระวังในขั้นตอนนี้คือ ต้องกระทำบนระบบปฏิบัติการที่เป็นตัวดั้งเดิม ไม่ใช่ตัวที่ถูกดัดแปรแล้ว เพราะอาจพบความผิดพลาด (Error) ได้ เมื่อสั่งคำสั่ง ./configure --host=i386-linux แล้ว จะใช้เวลาสักพักหนึ่ง ส่วนขั้นตอนคำสั่ง make ใช้เวลาประมาณสองชั่วโมงครึ่ง บนเครื่องโน้ตบุ๊กคอมพิวเตอร์ที่ใช้ทดลอง (ในหัวข้อ 4.1.5) จากนั้นจึงสั่ง make install โดยขั้นตอนนี้สามารถใช้ระบบปฏิบัติการที่ถูกดัดแปรแล้วได้ รอจนขั้นตอนนี้ทำงานเสร็จก็สั่งให้รีสตาร์ทใหม่ก่อนที่จะนำคอมไพเลอร์ใหม่นี้ไปใช้ โดยเรียกใช้ที่ /usr/local/bin/gcc (คอมไพเลอร์ดั้งเดิมจะอยู่ที่ /usr/bin/gcc) เมื่อทดสอบดูเวอร์ชันของคอมไพเลอร์ดั้งเดิม และของใหม่ จะได้ผลดังนี้

```
# gcc -v
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/egcs-2.91.66/specs
gcc version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)
# /usr/bin/gcc -v
Reading specs from /usr/lib/gcc-lib/i386-redhat-linux/egcs-2.91.66/specs
gcc version egcs-2.91.66 19990314/Linux (egcs-1.1.2 release)
```

ประวัติผู้เขียนวิทยานิพนธ์

นางสาวสิริสรา เจียมวงศ์แพทย์ เกิดเมื่อวันที่ 27 พฤศจิกายน พ.ศ. 2529 ที่จังหวัดกรุงเทพมหานคร สำเร็จการศึกษาหลักสูตรวิศวกรรมศาสตรบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ จากภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2550 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ ที่ภาควิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย ในปีการศึกษา 2551