วิศวกรรมซอฟต์แวร์เพื่อการรักษาความปลอดภัย

SECURE SOFTWARE ENGINEERING

นายทวีชัย สนธิลัคนากุล

นายธนพัฒน์ เหลืองรุ่งเรือง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

วิศวกรรมซอฟต์แวร์เพื่อการรักษาความปลอดภัย

SECURE SOFTWARE ENGINEERING

โดย

นายทวีชัย สนธิลัคนากุล

นายธนพัฒน์ เหลืองรุ่งเรือง

อาจารย์ที่ปรึกษา

อาจารย์ อัครเดช วัชระภูพงษ์

ผู้ช่วยศาสตราจารย์ ธนา หงษ์สุวรรณ

อาจารย์ ธนัญชัย ตรีภาค

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง วิศวกรรมซอฟต์แวร์เพื่อการรักษาความปลอดภัย

SECURE SOFTWARE ENGINEERING

ผู้จัดทำ

1. นายทวีชัย สนธิลัคนากุล รหัสประจำตัว 46010253
2. นายธนพัฒน์ เหลืองรุ่งเรือง รหัสประจำตัว 46010279

_____อาจารย์ที่ปรึกษา

(อาจารย์ อัครเดช วัชระภูพงษ์)

_____อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ธนา หงษ์สุวรรณ)

_____อาจารย์ที่ปรึกษา

(อาจารย์ ธนัญชัย ตรีภาค)

วิศวกรรมซอฟต์แวร์เพื่อการรักษาความปลอดภัย

นาย ทวีชัย สนธิลัคนากุล	46010253
นาย ธนพัฒน์ เหลืองรุ่งเรือง	46010279
อาจารย์ อัครเดช วัชระภูพงษ์	อาจารย์ที่ปรึกษา
ผศ. ธนา หงษ์สุวรรณ	อาจารย์ที่ปรึกษา
อาจารย์ ธนัญชัย ตรีภาค	อาจารย์ที่ปรึกษา

ปีการศึกษา 2549

บทคัดย่อ

โครงงานนี้เป็นโครงงานใหม่ที่มุ่งศึกษาวงจรชีวิตซอฟต์แวร์ (Software Development Life Cycle)ให้ทนทานต่อการถูกละเมิดความปลอดภัยโดยมุ่งเน้นไปที่ขั้นตอนการเขียนโปรแกรม (Coding) และทดสอบ(Testing)โค้ดภาษาซีบนระบบปฏิบัติการลีนุกซ์ ป้องกันปัญหาหลักคือการถูกโจมตีด้วยวิธีบัฟเฟอร์โอเวอร์โฟลว์และสร้างแบบตรวจการเขียนโค้ดให้ปลอดภัยเพื่อเป็นข้อกำหนดพื้นฐานของความปลอดภัยไม่ว่าซอฟต์แวร์เป็นซอฟต์แวร์ประยุกต์สำหรับใช้งานทั่วไป ซอฟต์แวร์อรรถประโยชน์สำหรับระบบปฏิบัติการหรือซอฟต์แวร์สำหรับระบบรักษาความปลอดภัยเองก็ตาม โดยเฉพาะซอฟต์แวร์เพื่อความปลอดภัยที่พัฒนาขึ้นโดยห้องวิจัย ISAG หรือจากภาควิชา ตลอดจนพัฒนาโปรแกรมเพื่อใช้สำหรับตรวจสอบช่องโหว่ความปลอดภัยซอฟต์แวร์ต่างๆ เหล่านั้น รวมทั้งพัฒนาโปรแกรมป้องกันปัญหาที่อาจเกิดขึ้น โปรแกรมอัพเดตเพื่อแก้ไขปัญหาที่มีอยู่และโปรแกรมทดสอบความทนทานของโปรแกรมต่างๆ

SECURE SOFTWARE ENGINEERING

Mr. Taweechai Sontilakanakul	46010253
Mr. Thanapat Luangrungruang	46010279
Mr. Akkradach Watcharapupong	Advisor
Asst.Prof. Thana Hongsuwan	Advisor
Mr. Thananchai Treepak	Advisor

Academic Year 2006

ABSTRACT

This project aims to study Software Development Life Cycle (SDLC) and harden each phase within SDLC for more security by concentrating on implementation (coding) and testing phase using c , c++ languages on UNIX operating system. The major problem is buffer overflow attack, which can be prevented by implementing the Secure Coding Checklist, which is used as basic security regulation applied to many types of software such as application software, OS utility software, security software and especially security software that has been developed by either ISAG project room or Computer Engineering Department. In addition, three more programs have been developed in assistance for vulnerability detection, security flaws testing and program version updating respectively.

กิตติกรรมประกาศ

การทำปริญญานิพนธ์ฉบับนี้คณะผู้จัดทำขอขอบพระคุณบิดา มารดา อันเป็นที่เคารพรัก ที่ช่วยอบรมสั่งสอนเลี้ยงดูคณะผู้จัดทำมาเป็นอย่างดี รวมถึงกำลังใจในการทำงานที่คอยให้โดยมาตลอด และส่งเสริมด้านการศึกษาให้คณะผู้จัดทำได้เป็นบุคคลที่มีความรู้มาถึง ณ ปัจจุบันนี้

โครงงานนี้คงจะเสร็จสมบูรณ์มิได้ หากขาดอาจารย์ที่ปรึกษาทั้งสามท่าน คือ อาจารย์อัครเดช วัชระภูพงษ์ ผศ.ธนา หงษ์สุวรรณ และ อาจารย์ธนัญชัย ตรีภาค ที่คอยให้ความรู้ คำปรึกษาและคำแนะนำต่างๆ เกี่ยวกับโครงงานนี้มาโดยตลอด ตลอดจนเพื่อนๆ พี่ๆ น้องๆ ทุกคนทั้งในและนอกห้องวิจัยที่คอยให้คำปรึกษา และช่วยเหลือด้วยดีเสมอมา รวมถึงสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบังที่ได้จัดเตรียมสิ่งอำนวยความสะดวก เพื่อให้งานวิจัยดำเนินไปได้อย่างราบรื่น สะดวก รวดเร็ว

ขอบคุณพี่น้องและคนที่อยู่ข้างๆ ที่คอยให้กำลังใจเวลาทุกข์ ตลอดจนรับฟังปัญหา และช่วยเหลือเสมอ

ขอขอบคุณคณาจารย์และนักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ทุกคนที่คอยให้คำแนะนำในสิ่งดี ๆ ประกอบขึ้นมาเป็นโครงงานชิ้นนี้ได้สำเร็จ

นาย ทวีชัย สนธิลัคนากุล

นาย ธนพัฒน์ เหลืองรุ่งเรือง

สารบัญ

สารบัญรูปภาพ

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ปัจจุบันมีการพัฒนาโปรแกรมเกิดขึ้นมากมาย แต่โปรแกรมที่พัฒนาขึ้นมานั้น ส่วนใหญ่มักจะมีปัญหาต่างๆ หรืออาจมีช่องโหว่ความปลอดภัยเกิดขึ้น ซึ่งอาจส่งผลให้ผู้บุกรุกใช้ประโยชน์จากช่องโหว่เหล่านี้ เจาะเข้ามาในระบบได้ ก่อให้เกิดความเสียหายในด้านต่างๆ ทั้งด้านทรัพยากรที่ต้องใช้ในการปรับปรุงแก้ไขระบบที่เกิดช่องโหว่ความปลอดภัย ดังจะเห็นได้จากข่าวคราวการแจ้งเตือนให้ทราบเกี่ยวกับช่องโหว่ความปลอดภัยของโปรแกรมต่างๆ อยู่เสมอ โครงงานนี้จึงมุ่งทั้งกระบวนการและวิธีการพัฒนาโปรแกรมให้มีความปลอดภัยในตัวเอง โดยครอบคลุมตลอดวงจรชีวิตของซอฟต์แวร์โดยจะเน้นไปทางด้านการเขียนโค้ดให้ปลอดภัยต่อการถูกละเมิดความปลอดภัย และการทดสอบความปลอดภัยของโปรแกรม อาศัยความเข้าใจที่เกี่ยวกับช่องโหว่ความปลอดภัยของโปรแกรม และกระบวนการขั้นตอนการผลิตซอฟต์แวร์

1.2 วัตถุประสงค์ของปริญญานิพนธ์

1. เพื่อศึกษาแนวคิดและมาตรฐานทางวิศวกรรมซอฟต์แวร์เพื่อการรักษาความปลอดภัย
2. เพื่อศึกษาการเขียนโค้ดภาษาซี ให้ทนทานต่อการถูกละเมิดความปลอดภัย
3. เพื่อพัฒนาโปรแกรม, เครื่องมือ สำหรับช่วยหาช่องโหว่จากการเขียนโปรแกรมภาษาซี รวมทั้งป้องกันการถูกเอ็กซ์พลอยต์ด้วยบัฟเฟอร์โอเวอร์โฟลว์
4. เพื่อสร้างข้อควรปฏิบัติ ในการพัฒนาซอฟต์แวร์ให้ปลอดภัย ครอบคลุมวงจรชีวิตซอฟต์แวร์ โดยเน้นที่ขั้นตอนการเขียนซอร์สโค้ด

1.3 ขอบเขตของปริญญานิพนธ์

ในปริญญานิพนธ์ฉบับนี้ศึกษากระบวนการ และวิธีพัฒนาโปรแกรมให้มีความปลอดภัยโดยเน้นไปที่ส่วนการเขียนโปรแกรมของซอฟต์แวร์ที่พัฒนาขึ้นด้วยภาษาซี บนระบบปฏิบัติการลีนุกซ์เป็นหลัก และการทดสอบโปรแกรม ตลอดจนสามารถสร้างเอกสารและเครื่องมือตรวจสอบความปลอดภัยเพื่อเป็นแนวทางในการพัฒนาโปรแกรมที่ปราศจากช่องโหว่ความปลอดภัยได้ โดยจะมีการทดสอบกับโครงงานอื่นๆ ของห้องวิจัย ISAG

1.4 วิธีการดำเนินการ

1. ศึกษารายละเอียดต่างๆ ของโครงงาน
   - ศึกษาวิศวกรรมซอฟต์แวร์
   - ศึกษากระบวนการพัฒนาซอฟต์แวร์ให้มีความปลอดภัย
   - ศึกษาการเกิด บัฟเฟอร์โอเวอร์โฟลว์, สแตกโอเวอร์โฟลว์ , ฮีปโอเวอร์โฟลว์
2. Security Tools
   - โปรแกรมสแกนโค้ดภาษาซี เพื่อตรวจหาการเขียนโปรแกรมที่ไม่ปลอดภัย
   - Overflow Guard ที่สามารถป้องกันการเกิด บัฟเฟอร์โอเวอร์โฟลว์ของซอฟแวร์
   - Fault injection tool เพื่อใช้ทดสอบการทนทานต่อการบุกรุกของซอฟแวร์
   - เขียนโค้ดสำหรับการปรับปรุงเวอร์ชันซอฟต์แวร์ของโครงงานอื่นๆ ของห้อง ISAG
3. จัดทำเอกสารคำแนะนำเกี่ยวกับการเขียนโปรแกรมที่มีความปลอดภัยเพื่อป้องกันการถูกบุกรุก

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ความเข้าใจเกี่ยวกับการบุกรุกช่องโหว่ทางซอฟต์แวร์
2. ได้รับความรู้ความเข้าใจในการเขียนโปรแกรมด้วยภาษาซีอย่างปลอดภัย
3. สร้างเครื่องมือ สำหรับหาช่องโหว่ และทดสอบความปลอดภัยของซอฟต์แวร์
4. สามารถนำเอาเอกสารและเครื่องมือมาปรับปรุงซอฟต์แวร์จากโครงงานอื่นๆของห้อง ISAG เพื่อให้มีความทนทานต่อการละเมิดความปลอดภัยยิ่งขึ้น

1.6 ส่วนประกอบของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 6 บท 3 ภาคผนวกด้วยกันคือ

บทที่ 1 กล่าวถึงความสำคัญและที่มาของโครงงาน วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน วิธีการดำเนินการ ประโยชน์ที่คาดว่าจะได้รับและส่วนประกอบของปริญญานิพนธ์

บทที่ 2 กล่าวถึงทฤษฎีต่างๆที่เกี่ยวข้องวิศวกรรมซอฟต์แวร์เหตุผลที่ต้องพัฒนาโปรแกรมให้ปลอดภัย ประเภทของโปรแกรมที่ควรมีความปลอดภัย ปัญหาการเกิดบัฟเฟอร์โอเวอร์โฟลว์ และรูปแบบการตรวจสอบการปรับปรุงเวอร์ชัน

บทที่ 3 กล่าวถึงความปลอดภัยในขั้นตอนการวิเคราะห์และออกแบบ หลักการในการออกแบบซอฟต์แวร์ให้ปลอดภัยด้วยภาษาซี

บทที่ 4 กล่าวถึงความปลอดภัยในขั้นตอนการเขียนโค้ดและตรวจสอบ โปรแกรมต่างๆ ที่พัฒนาขึ้นมา โปรแกรมตรวจสอบการเขียนโค้ดด้วยภาษาซี โปรแกรมป้องกันการเกิดบัฟเฟอร์โอเวอร์โฟลว์, ฟอลต์อินเจ็คชัน , การตรวจสอบการปรับปรุงเวอร์ชัน

บทที่ 5 กล่าวถึงการทดลองและผลการทดลองโดยได้เขียนการทดลองและผลการทดลองสำหรับโปรแกรมต่างๆ ในบทที่ 4

บทที่ 6 เป็นบทวิจารณ์และสรุป ซึ่งกล่าวถึงบทสรุปของโครงงาน วิจารณ์สิ่งที่ได้รับจากโครงงาน และข้อเสนอแนะสำหรับเป็นแนวทางในการพัฒนาต่อ

ภาคผนวก ก. ผลจากการตรวจโค้ดภาษาซีของกลุ่มโครงงานต่างๆ ในห้อง ISAG

ภาคผนวก ข. คู่มือป้องกันการเขียนโค้ดให้ปลอดภัยด้วยภาษาซี

ภาคผนวก ค. ฟังก์ชันที่ไม่ปลอดภัย สำหรับภาษาซี

บทที่ 2
ทฤษฎีต่างๆ ที่เกี่ยวข้อง

2.1 วิศวกรรมซอฟต์แวร์ (Software engineering)

วิศวกรรมซอฟต์แวร์ (Software engineering) เป็นศาสตร์เกี่ยวกับการผลิตซอฟต์แวร์ตั้งแต่การเริ่มเก็บความต้องการ, การตั้งเป้าหมายของระบบ , การออกแบบ , ไปจนถึงกระบวนการพัฒนาและการประเมินผล วิศวกรรมซอฟต์แวร์ประยุกต์ความรู้และเทคโนโลยีวิทยาการคอมพิวเตอร์ การบริหารจัดการโครงการ และสาขาอื่นๆ ที่เกี่ยวข้องเข้าด้วยกัน เพื่อสร้างซอฟต์แวร์ที่สามารถปฏิบัติงานตามเป้าหมาย ภายใต้เงื่อนไขที่กำหนด

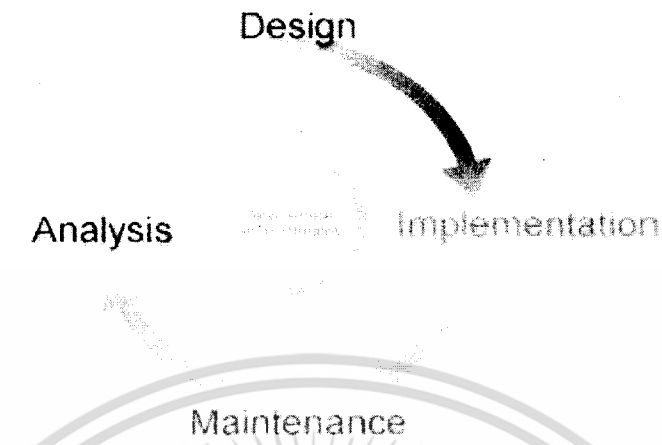
วิศวกรรมซอฟต์แวร์เป็นศาสตร์ที่ทวีความสำคัญเพิ่มขึ้นเรื่อยๆ เนื่องจากในปัจจุบันซอฟต์แวร์มีความซับซ้อนเพิ่มขึ้น จำเป็นต้องมีวิธีที่จะควบคุมและดำเนินการผลิต ที่มีประสิทธิภาพ สามารถวัดผลได้ และสามารถตรวจหาข้อผิดพลาดพร้อมสาเหตุได้อย่างสะดวกและรวดเร็ว เพื่อให้สามารถปรับปรุงแก้ไขซอฟต์แวร์ตั้งแต่อยู่ในระหว่างการผลิตได้

การผลิตผลิตภัณฑ์ที่จะต้องใช้วิธีการพัฒนาทางวิศวกรรมทั้งสิ้น ไม่ว่าจะเป็นรถยนต์ซึ่งเป็นวิศวกรรมเครื่องกล เครื่องโทรทัศน์ซึ่งเป็นวิศวกรรมไฟฟ้า ตลอดไปจนถึงงานการก่อสร้างสะพานหรืออาคารสูงซึ่งเป็นวิศวกรรมโยธา ล้วนต้องมีกระบวนการหรือขั้นตอนพัฒนาและบำรุงรักษาที่ประกอบไปด้วยวิธีการต่างๆมากมาย ในงานพัฒนาระบบซอฟต์แวร์ขนาดใหญ่ก็เช่นเดียวกัน จำเป็นต้องมีกระบวนการเชิงวิศวกรรมที่เรียกว่า วิศวกรรมซอฟต์แวร์ ในการพัฒนาและบำรุงรักษาซอฟต์แวร์อย่างเป็นขั้นตอน เพื่อให้งานสำเร็จลุล่วงตามเวลาและบรรลุเป้าหมายที่ต้องการ

การพัฒนาระบบซอฟต์แวร์ขนาดใหญ่ๆ เช่น ระบบสินค้าคงคลังในงานธุรกิจ ระบบการลงทะเบียนเรียนในมหาวิทยาลัย หรือระบบบัญชีลูกหนี้เจ้าหนี้ของบริษัทร้านค้า นับเป็นงานที่ค่อนข้างซับซ้อน มีขอบเขตเกินกว่าที่สมองของมนุษย์จะจดจำได้อย่างครบถ้วน หรือโปรแกรมเล็กๆ สำหรับผู้พัฒนาเพียงคนเดียว จำเป็นต้องอาศัยผู้ร่วมพัฒนาหลายคนทำงานร่วมกันในช่วงเวลาที่ยาวพอสมควร เพราะในระหว่างการพัฒนาอาจมีปัญหาอุปสรรคเกิดขึ้นได้เสมอ เช่น เป้าหมายหรือข้อกำหนดต่างๆ ของระบบอาจมีการปรับเปลี่ยนให้เหมาะสมขึ้น หรือปัญหาด้านบุคลากรที่ร่วมโครงการอาจมีการสับเปลี่ยนเนื่องจากการเปลี่ยนตำแหน่ง หรือ ย้ายงานใหม่ เป็นต้น

วิศวกรรมซอฟต์แวร์ไม่ใช่เป็นเรื่องที่เกี่ยวข้องเฉพาะกับปัญหาด้านเทคนิคของระบวนการพัฒนาเท่านั้น ยังจะรวมไปถึงปัญหาด้านบุคลากรและการควบคุมติดตามโครงการ ซึ่งในที่นี้จะ

กล่าวเฉพาะกระบวนการพัฒนาซอฟต์แวร์และระบบซอฟต์แวร์ที่มีการหมุนเวียนใช้งานเป็นวัฏจักรซอฟต์แวร์ ดังรูป



รูปที่ 2.1 วัฏจักรซอฟต์แวร์

ซอฟต์แวร์ที่พัฒนามาแล้ว จะเข้าสู่วัฏจักรของการนำไปใช้งานแล้วนำมาปรับปรุงแก้ไขและย้อนกลับนำมาใช้งานใหม่ ตลอดระยะเวลาการใช้งานซอฟต์แวร์นั้น จนกว่าจะมีซอฟต์แวร์ใหม่มาแทนที่ ผลิตภัณฑ์อุตสาหกรรมต่างๆ มีวัฏจักรเช่นเดียวกัน เพียงแต่ว่า วัฏจักรของผลิตภัณฑ์ไม่ใช่การปรับปรุงแก้ไข แต่จะเป็นการซ่อมบำรุงให้ใช้งานต่อไปได้ ซอฟต์แวร์ต่างกับผลิตภัณฑ์ตรงที่ไม่มีส่วนสึกหรอ

การปรับปรุงแก้ไขซอฟต์แวร์ อาจเกิดขึ้นจากข้อผิดพลาดของซอฟต์แวร์ที่ยังหลงค้างอยู่ เช่น ปัญหาด้านความปลอดภัย , ปัญหาจากการทำงานผิดพลาด หรืออาจเกิดจากข้อกำหนดของเงื่อนไขภายในซอฟต์แวร์ที่มีการเปลี่ยนแปลง เช่น การเปลี่ยนแปลงวิธีการจัดเก็บภาษีของระบบบัญชี ต้องแก้ไขเงื่อนไขในซอฟต์แวร์ใหม่ ตามปกติซอฟต์แวร์ที่พัฒนามาแล้วมักมีการปรับปรุงแก้ไขอยู่เสมอ เพราะเป้าหมายความต้องการของผู้ใช้งานมักจะเปลี่ยนแปลงภายหลังการทดลองใช้ไประยะหนึ่ง

ปัญหาสำคัญของการปรับปรุงแก้ไขซอฟต์แวร์ ส่วนใหญ่จะอยู่ที่ผู้ปรับปรุงแก้ไขเป็นคนละคนกับผู้พัฒนา ดังนั้นผู้ปรับปรุงจะต้องศึกษาโปรแกรมและเอกสารประกอบให้เข้าใจอย่างถ่องแท้เสียก่อนจึงจะสามารถปรับปรุงได้ ซึ่งต้องใช้เวลาพอสมควร และถ้าโปรแกรมและเอกสารประกอบไม่ได้มีรูปแบบโครงสร้างที่ดีพอแล้ว ผู้ปรับปรุงซอฟต์แวร์จะประสบปัญหากับการแก้ไขมากยิ่งขึ้น จนบางครั้งต้องพัฒนาซอฟต์แวร์ขึ้นใหม่ เหตุการณ์เช่นนี้พบเห็นกันอยู่เสมอๆ

ตามปกติระยะเวลาที่ใช้เพื่อการพัฒนาซอฟต์แวร์จะน้อยกว่าระยะเวลาของการปรับปรุงแก้ไขเพราะเมื่อซอฟต์แวร์นำไปใช้งานแล้วจะมีการย้อนกลับมาปรับปรุงแก้ไขได้ตลอดเวลา การปรับปรุงแก้ไขซอฟต์แวร์สามารถทำได้ง่ายและรวดเร็วหากมีการเตรียมการตั้งแต่ขั้นตอนการพัฒนา

ซอฟต์แวร์ โดยพยายามศึกษาวิเคราะห์และออกแบบให้ละเอียดครบถ้วน และให้เอื้ออำนวยต่อการนำไปแก้ไขปรับปรุงหรือพัฒนาต่อได้ง่ายในภายหลัง

2.1.1 ขั้นตอนการพัฒนาซอฟต์แวร์

การพัฒนาซอฟต์แวร์ที่ใช้กันทั่วไป จะแบ่งออกเป็นขั้นตอน ดังนี้ การวิเคราะห์ การออกแบบ การเขียนโปรแกรม หรือการสร้างชิ้นงานจริง และการตรวจสอบซอฟต์แวร์

2.1.1.1 การวิเคราะห์ (Analysis)

การวิเคราะห์เป็นขั้นตอนแรกของการพัฒนาซอฟต์แวร์ ซึ่งอาจแบ่งได้เป็นสองตอน ตอนแรกจะเป็นการสำรวจความต้องการ และเหตุผลของการตัดสินใจนำคอมพิวเตอร์เข้ามาช่วยในการทำงานให้เป็นไปโดยอัตโนมัติ ตอนที่สองจึงเป็นการวิเคราะห์ระบบงานที่ใช้อยู่ปัจจุบัน หากนำระบบที่ใช้คอมพิวเตอร์มาช่วยงานจะตอบสนองความต้องการได้อย่างไร ข้อมูลที่ใช้และระบบซอฟต์แวร์จะต้องกำหนดได้อย่างเด่นชัด ผลลัพธ์จากการวิเคราะห์ จะทำให้เราได้ชุดของข้อกำหนดของระบบเพื่อนำไปใช้ในการออกแบบซอฟต์แวร์ต่อไป

2.1.1.2 การออกแบบ (Design)

ในการออกแบบซอฟต์แวร์ จะเป็นงานพัฒนาทางด้านเทคนิคเพื่อแบ่งแยกงานให้เป็นหน่วยย่อยเรียก “โมดูล” (module) ที่สามารถแยกจัดการเฉพาะส่วนได้โดยง่าย การนำระบบใหญ่มาแบ่งย่อยเป็นส่วนเล็ก ๆ และสามารถนำมาเชื่อมรวมกันเป็นระบบใหญ่ถือเป็นส่วนสำคัญที่ช่วยให้งานใหญ่สำเร็จลงได้ เพราะการสร้างระบบซอฟต์แวร์ขนาดใหญ่แต่เพียงลำพัง เป็นเรื่องสุดวิสัยเหนือกำลังของคนคนเดียว การแยกส่วนแบ่งงานกันทำจะทำให้ผู้พัฒนาแต่ละคนสามารถทำงานเฉพาะในส่วนของตนได้ดี ในขณะเดียวกันจะง่ายแก่การบำรุงรักษาในอนาคตอีกด้วย

2.1.1.3 การเขียนโปรแกรมหรือการสร้างชิ้นงานจริง (Coding)

เป็นขั้นตอนการสร้างหรือเขียนโปรแกรม การสร้างแฟ้มข้อมูลและการพัฒนาฐานข้อมูล การออกแบบซอฟต์แวร์เป็นหน่วยย่อยหลายๆ โมดูลทำให้สามารถแบ่งงานเขียนโปรแกรมหรือสร้างชิ้นงานให้กับนักเขียนโปรแกรมหลายๆ คนสามารถทำงานไปพร้อมๆกันได้

2.1.1.4 การตรวจสอบและบำรุงรักษาซอฟต์แวร์ (Maintenance)

ขั้นตอนสุดท้ายเป็นการตรวจสอบซอฟต์แวร์ว่าทำงานได้ครบถ้วนตามต้องการหรือไม่ และตรวจสอบการทำงานของโปรแกรมว่ามีความปลอดภัยในการทำงานหรือไม่ โดยมีการตรวจแก้ไขซอฟต์แวร์เป็นชุดโมดูล และตรวจสอบการทำงานร่วมกันของโมดูลต่างๆ ซึ่งขั้นตอนนี้จะต้องพิถีพิถันกระทำการตรวจสอบอย่างละเอียด และต้องมีการดูแล ปรับปรุงแก้ไขเมื่อพบว่าเกิดช่องโหว่ต่างๆ ภายในโปรแกรม

การพัฒนาซอฟต์แวร์ขนาดใหญ่จำเป็นต้องดำเนินตามขั้นตอนที่กล่าวแล้วข้างต้น เพื่อให้ได้ระบบซอฟต์แวร์ที่สมบูรณ์มีข้อผิดพลาดน้อยที่สุด ในทุกขั้นตอนควรเขียนเอกสารประกอบอย่างครบถ้วน เพื่อให้ผู้ร่วมงานคนอื่น ๆ เข้าใจและทำงานร่วมกันได้

2.1.2 มาตรฐานความปลอดภัย

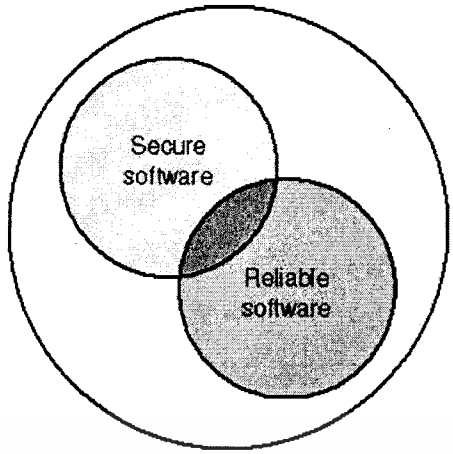
มาตรฐานด้านความมั่นคงปลอดภัยทางด้านคอมพิวเตอร์นั้นมีหลายหน่วยงานที่เป็นผู้ออกมาตรฐาน ซึ่งมีจุดมุ่งหมายเพื่อให้ข้อมูล หรือระบบต่างๆ มีความปลอดภัย ดังตัวอย่างดังนี้

- ISO 17799 "Information Technology: Code of Practice for Information Security Management"
- ISO/IEC 15408 "Evaluation Criteria for IT Security" (the "Common Criteria")
- SSE-CMM "System Security Engineering Capability Maturity Model"
- ISO/IEC WD 15443 "Information Technology: Security Techniques" (for an overview)

ISO 17799 เป็นมาตรฐานที่ช่วยสร้างกระบวนการในการรักษาความมั่นคงปลอดภัยให้กับองค์กร หรือกระบวนการต่างๆในองค์กร กระบวนการดังกล่าวเริ่มจากการประเมินความเสี่ยง การจัดทำนโยบายการรักษาความมั่นคงปลอดภัย และการออกแบบข้อกำหนดหรือมาตรการ หลังจากได้มีการประกาศมาตรฐานสากล ISO 17799 อย่างเป็นทางการแล้ว ผลจากการนำมาตรฐานดังกล่าวนี้ไปประยุกต์ใช้โดยองค์กรธุรกิจ หรืออุตสาหกรรม อาจมีการนำไปประยุกต์ใช้ในรูปแบบที่แตกต่างกันไป ผู้ใช้มาตรฐานเหล่านี้จึงรวมตัวกันเป็นลักษะของกลุ่มผู้ใช้ หรือ อินเตอร์เนชันแนลยูสเซอร์กรุ๊ป(ไอยูจี)

2.1.3 โปรแกรมประยุกต์ที่มีคุณภาพ

ในการพัฒนาโปรแกรมประยุกต์ที่มีคุณภาพนั้น การทำให้โปรแกรมประยุกต์นั้นมีความปลอดภัย ถือว่าเป็นส่วนหนึ่งในการพัฒนาโปรแกรม เพราะถ้าโปรแกรมประยุกต์ปราศจากความปลอดภัย ผู้ใช้ก็จะไม่เชื่อมั่นในการใช้งานโปรแกรม โดยโปรแกรมประยุกต์ที่มีคุณภาพนั้นจะประกอบไปด้วยความปลอดภัย และความน่าเชื่อถือ ดังรูป



รูปที่ 2.2 โปรแกรมประยุกต์ที่มีคุณภาพ

2.2 ทำไมต้องพัฒนาโปรแกรมให้ปลอดภัย

โปรแกรมจำนวนมากที่มาการนำออกมาใช้งาน ส่วนใหญ่นั้นจะเกิดปัญหาด้านความปลอดภัย เนื่องจากผู้พัฒนาส่วนมากนั้น ไม่ให้ความสำคัญกับความปลอดภัยของโปรแกรม ส่งผลให้เกิดปัญหาต่างๆ ตามมา และมีการแจ้งเตือนให้ผู้ใช้ติดตั้งชุดแก้ไขต่างๆให้กับระบบตามที่มีการแจ้งเตือนอยู่เสมอ ดังที่เป็นข่าว เช่น ไอซีคิว ซึ่งเป็นโปรแกรมที่ใช้ในการติดต่อสื่อสารกับผู้ใช้อื่นๆ บนอินเตอร์เน็ตและยังเป็นโปรแกรมที่ใช้กันอย่างแพร่หลาย ตามสถิติที่บันทึกโดยบริษัท ไอซีคิว (บริษัทในเครือของเอโอแอลไทม์วอร์เนอร์) ว่ามีผู้ใช้ถึง 122 ล้านคน ช่องโหว่บัฟเฟอร์โอเวอร์โฟลว์ เกิดขึ้นกับไคลเอ็นต์ที่ใช้วินโดวส์ และช่องโหว่นี้จะเกิดขึ้นระหว่างขั้นตอนของข้อความการร้องเสียง วีดีโอ และเกมส์ ข้อความนี้จะเชิญชวนให้ผู้ใช้ไอซีคิวเข้าร่วมการสนทนาโดยผ่านโปรแกรมที่สามที่มีช่องโหว่ บัฟเฟอร์โอเวอร์โฟลว์ เกิดขึ้นในโปรแกรมไอซีคิว ผู้บุกรุกสามารถโจมตีผ่านช่องโหว่นี้เพื่อรันโค้ดที่เป็นอันตรายด้วยสิทธิของผู้ใช้ที่เป็นเหยื่อ จากกรณีดังกล่าว ซึ่งเกิดขึ้นในปี ค.ศ. 2002 ทำให้ทาง ไอซีคิว ต้องออกประกาศแจ้งเตือนให้ผู้ใช้ทำการติดตั้งชุดแก้ไข (แพทช์) หรือทำการดาวน์โหลดเวอร์ชันใหม่ ซึ่งทำให้เกิดผลกระทบอย่างมาก ซึ่งวิธีการติดตั้งชุดแก้ไขนั้นเป็นการแก้ปัญหาตามหลักวิศวกรรมก็จริง แต่เป็นการแก้ปัญหาที่ปลายเหตุ ดังนั้นในการพัฒนาโปรแกรมควรคำนึงถึงความปลอดภัยในทุกๆ ขั้นตอนของการพัฒนาโปรแกรม เพื่อความสมบูรณ์ของโปรแกรมให้มากที่สุด

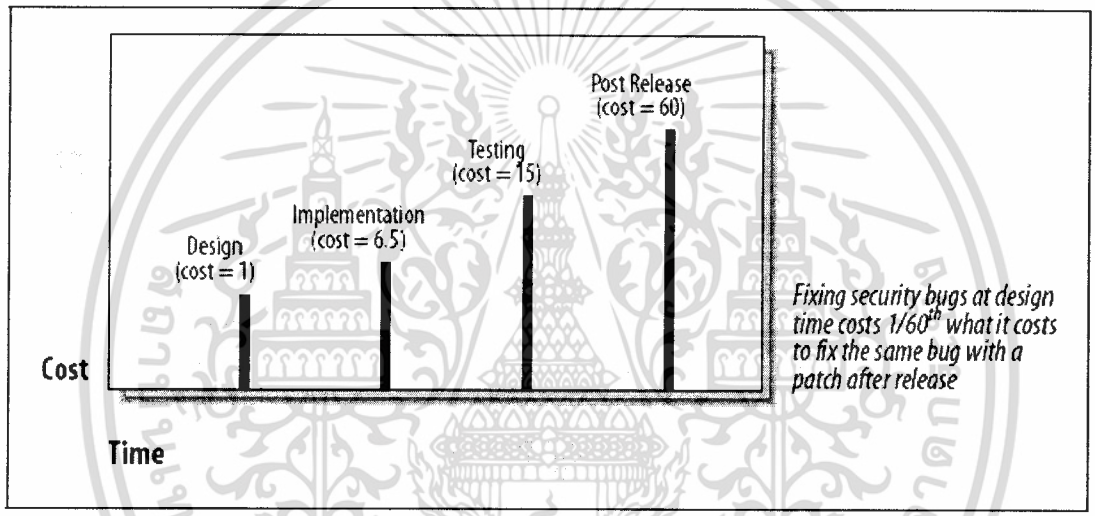
2.2.1 ผลกระทบจากปัญหาด้านความปลอดภัย

2.2.1.1 ผลกระทบด้านค่าใช้จ่าย

ในการพัฒนาโปรแกรมประยุกต์ตามหลักวิศวกรรมซอฟต์แวร์ อาจจะต้องมีการแก้ปัญหาด้านความปลอดภัย ซึ่งจะมีผลกระทบต่อการทำงานหรือค่าใช้จ่ายต่างๆ ในการแก้ปัญหาด้านความ

ปลอดภัย นักวิจัยจึงได้วิเคราะห์ผลกระทบจากค่าใช้จ่ายที่เกิดขึ้นในการแก้ปัญหาความปลอดภัยในแต่ละขั้นตอนการทำงาน

ผลกระทบจากค่าใช้จ่ายที่เกิดขึ้นจากการแก้ปัญหาด้านความปลอดภัยในเฟสต่างๆ จะเห็นได้จากรูปที่ 2.3 ว่า การแก้ปัญหาด้านความปลอดภัยในระหว่างที่อยู่ในกระบวนการออกแบบ (Design) นั้นมีการใช้ค่าใช้จ่ายที่น้อยที่สุด และเพิ่มขึ้นเป็นทวีคูณเมื่อทำการแก้ไขในกระบวนการต่อมา และจะต้องใช้ค่าใช้จ่ายมากที่สุดหากซอฟต์แวร์ประยุกต์นั้นมีการนำออกมาใช้งานแล้ว ผลกระทบนี้มีความสำคัญอย่างมากในการทำธุรกิจที่เกี่ยวข้องกับการพัฒนาโปรแกรมประยุกต์ ซึ่งเป็นสิ่งสำคัญที่หลายคนอาจมองข้ามไป ดังนั้นจึงควรมีการตรวจสอบปัญหาด้านความปลอดภัยในทุกกระบวนการทำงานพัฒนาโปรแกรม



รูปที่ 2.3 กราฟแสดงค่าใช้จ่ายในการแก้ไขปัญหาความปลอดภัย ในช่วงต่างๆ ของการพัฒนาโปรแกรม

2.2.1.2 ผลกระทบต่อระบบ

การโจมตีด้วยวิธีเอ็กซ์พลอยต์แบบบัฟเฟอร์โอเวอร์โฟลว์ จะมุ่งเน้นไปที่การทำให้โปรแกรมเกิดการล้นของบัฟเฟอร์(บัฟเฟอร์โอเวอร์โฟลว์) เพื่อให้ผู้ถูกโจมตีได้ทำการเรียกโปรแกรมที่ผู้โจมตีต้องการให้ทำงานขึ้นมา โดยส่วนใหญ่แล้วสิ่งที่ผู้โจมตีต้องการคือสิทธิของการเป็นรูตของเครื่องผู้ถูกโจมตี ในทางทฤษฎีแล้วดูเรียบง่ายไม่ซับซ้อนเลย นั่นคือ โปรแกรมที่ผู้โจมตีส่งเข้ามาจะถูกใส่ไว้ในบัฟเฟอร์ ซึ่งจะถูกทำให้ล้นออกมา โดยขั้นตอนการแก้ไขส่วนต่างๆของหน่วยความจำเท่านั้นเอง

2.2.2 ตัวอย่างการ เอ็กซ์พลอยต์ ด้วยวิธีฮีปโอเวอร์โฟลว์

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

int main(int argc, char *argv[])
{
FILE *filed;
char *userinput = malloc(20);
char *outputfile = malloc(20);

if(argc < 2)
{
printf("Usage: %s <string to be written to /tmp/notes>\n",argv[0]);
exit(0);
}

strcpy(outputfile, "/tmp/notes");
strcpy(userinput,argv[1]);

printf("userinput @ %p: %s\n", userinput,userinput);
printf("outputfile @ %p: %s\n",outputfile,outputfile);

filed = fopen(outputfile, "a");
if(filed == NULL)
{
fprintf(stderr, "error opening file %s\n", outputfile);
exit(1);
}
fprintf(filed,"%s\n",userinput);
fclose(filed);
return 0;
}
```

จากโปรแกรม โปรแกรมจะทำการประกาศตัวแปรแบบไดนามิกขึ้นมา 2 ตัว นั่นคือ userinput และ outputfile โดยจองหน่วยความจำไว้มีขนาด 20 ชาร์แรคเตอร์จากนั้นโปรแกรมจะทำการรับค่าจากผู้ใช้แล้วนำไปเก็บไว้ในตัวแปร userinput ส่วนตัวแปร outputfile จะเก็บค่า /tmp/notes หลังจากนั้น โปรแกรมก็จะทำการเขียนไฟล์ที่อยู่บนตัวแปร outputfile ด้วยข้อความที่เก็บไว้ในตัวแปร userinput

จากนั้นลองทำการประมวลผลโปรแกรม

```
waan@Jinx:~/project$ ./heap_overflow overflow
userinput @ 0x8049958: overflow
outputfile @ 0x8049970: /tmp/notes
waan@Jinx:~/project$ cat /tmp/notes
overflow
waan@Jinx:~/project$
```

จะเห็นว่าโปรแกรมจะทำการเขียนข้อความคำว่า overflow ลงไปยัง /tmp/notes แต่ถ้าเราลองใส่สตริงที่มีค่าเกินกว่าที่โปรแกรมจองไว้

```
waan@Jinx:~/project$ ./heap_overflow 12345678901234567890123456789O
userinput @ 0x8049958: 12345678901234567890123456789O
outputfile @ 0x8049970: 567890
waan@Jinx:~/project$
```

จะเห็นว่าตัวแปร outputfile จะถูกเขียนทับอันเนื่องมาจากการล้นของตัวแปร userinput ทำให้โปรแกรมทำการเขียนข้อความ 12345678901234567890123456789O ลงไปยังไฟล์ 567890 ในกรณีแบบนี้ถ้าโปรแกรมนี้รูตเป็นเจ้าของไฟล์และมีการตั้งค่า SUID จะทำให้โปรแกรมนี้สามารถเข้าไปแก้ไขไฟล์สำคัญๆได้ โดยเฉพาะ /etc/passwd ผู้โจมตีก็สามารถเข้าไปเพิ่มผู้ใช้ที่มีสิทธิเป็นรูตได้ นั่นคือเราพยายามที่จะให้ userinput เป็น rooted::0:0:m:/root:/bin/bash แล้ว outputfile มีค่าเป็น /etc/passwd แต่ถ้าเราใช้คำสั่ง ./heap overflow rooted::0:0:m:/root:/bin/bash ค่า outputfile ที่ได้จะมีค่าเป็น /bash เพราะค่า outputfile จะถูกทับที่ตัวอักษรที่ 25 ซึ่งไม่เป็นประโยชน์ สิ่งที่เราต้องการคือทำให้ตัวที่ 25 ของ userinputfile มีค่าเป็น /etc/passwd สิ่งที่เราต้องทำคือสร้างไดเรคทอรี่และทำการสร้าง SYM LINK ดังนี้

```
waan@Jinx:~$ mkdir /tmp/etc
waan@Jinx:~$ ln -s /bin/bash /tmp/etc/passwd
waan@Jinx:~$
```

ซึ่งจะทำให้ /tmp/etc/passwd ลิ้งค์ไปที่ /bin/bash จากนั้นใช้คำสั่ง ./heap_overflow rooted::0:0:m:/root:/tmp/etc/passwd ซึ่งจะได้ผลดังนี้

```
waan@Jinx:~/project$ ./heap_overflow rooted::0:0:m:/root:/tmp/etc/passwd
userinput @ 0x8049958: rooted::0:0:m:/root:/tmp/etc/passwd
outputfile @ 0x8049970: /etc/passwd
waan@Jinx:~/project$ su rooted
Jinx:/home/waan/project# id
uid=0(root) gid=0(root) groups=0(root)
Jinx:/home/waan/project#
```

จะเห็นว่าเราสามารถเพิ่มผู้ใช้ลงไปใน /etc/passwd และสามารถได้สิทธิเป็นรูต การเอ็กซ์พลอยต์ด้วยวิธีฮีปโอเวอร์โฟลว์

ดังที่กล่าวมา บัฟเฟอร์โอเวอร์โฟลว์ เป็นแค่วิธีการหนึ่งซึ่งเป็นแนวทางของการเกิดเอ็กซ์พลอยต์ของโปรแกรม ผลกระทบของมันนั้นร้ายแรงมาก โดยผู้ไม่ประสงค์ดีที่ได้สิทธิเป็นรูตแล้วนั้นสามารถทำอะไรกับระบบของเราก็ได้ อันจะก่อให้เกิดความเสียหายมากมาย

2.3 ประเภทของโปรแกรมที่ต้องการความปลอดภัย

โปรแกรมหลายๆ ประเภทนั้นต้องการความปลอดภัยในการใช้งาน ยกตัวอย่างเช่น

- โปรแกรมประยุกต์ที่มีการติดต่อกับเครือข่ายภายนอก
- โปรแกรมประยุกต์ที่ใช้โดยผู้ดูแลระบบ (รูต)
- เซิร์ฟเวอร์ท้องถิ่น (แดมอน)
- เซิร์ฟเวอร์ที่มีการเข้าถึงโดยผู้ใช้ภายนอกเครือข่าย (เน็ทเวิร์กแดมอน)
- เว็บแอพพลิเคชัน (ที่มีการใช้งานสคริปประเภทซีจีไอ)
- โปรแกรมประเภท SETUID/SETGID

ซึ่งในการพัฒนาโปรแกรมประยุกต์นั้น เป็นไปได้ยากมากที่จะทำให้ปราศจากช่องโหว่ความปลอดภัย แต่การศึกษาถึงช่องโหว่ความปลอดภัย ที่ก่อให้เกิดการโจมตีระบบจากผู้ที่ไม่ประสงค์ดีหรือผู้บุกรุก จะช่วยให้การพัฒนาโปรแกรมนั้นมีความปลอดภัยที่สูงขึ้น และกำจัดช่องโหว่ความปลอดภัยให้มากที่สุด

2.4 บัฟเฟอร์โอเวอร์โฟลว์

การโจมตีของผู้บุกรุกโดยใช้วิธีการที่เรียกว่า บัฟเฟอร์โอเวอร์โฟลว์ จะเน้นไปที่การทำให้โปรแกรมเกิดการล้นของบัฟเฟอร์(บัฟเฟอร์โอเวอร์โฟลว์) เพื่อให้โปรแกรมประยุกต์ของผู้ถูกโจมตีได้ทำการเรียกโปรแกรมที่ผู้โจมตีต้องการให้ทำงานขึ้นมา โดยส่วนใหญ่แล้วสิ่งที่ผู้โจมตีต้องการคือสิทธิของการเป็นรูตของเครื่องผู้ถูกโจมตี ในทางทฤษฎีการโจมตีดูง่ายมากนั่นคือ โปรแกรมที่ผู้โจมตีส่งเข้ามาจะถูกใส่ไว้ในบัฟเฟอร์ซึ่งจะถูกทำให้ล้นออกมา โดยขั้นตอนการแก้ไขส่วนต่างๆ ของหน่วยความจำเท่านั้น

ส่วนใหญ่ บัฟเฟอร์โอเวอร์โฟลว์ เกิดขึ้นเพราะภาษาซี และส่วนหนึ่งเกิดจากลักษณะการเขียนโปรแกรมที่ไม่ดี ภาษาซีจะไม่มีการเปลี่ยนแปลง แต่ลักษณะการเขียนโปรแกรมที่ไม่ดีสามารถพัฒนาได้

ภาษาซีเกิดขึ้นมาตอนต้นทศวรรษที่ 70 ในฐานะเป็นเครื่องมือสำหรับการย้ายระบบปฏิบัติการยูนิกซ์ และยูทิลิตี้ไปยังเครื่องที่มีสถาปัตยกรรมหลากหลาย เริ่มแรกยูนิกซ์ถูกเขียนขึ้นจากภาษาแอสแซมบลี แล้วจึงย้ายไปสู่ภาษาซี ทำให้ยูนิกซ์เป็นระบบปฏิบัติการแรกที่สามารถย้ายข้ามเครื่องได้อย่างแท้จริงภาษาซีได้รับการออกแบบให้กะทัดรัดและเร็ว ความเร็วเป็นรองเพียงแค่ภาษาแอสแซมบลีเท่านั้น

ภาษาซีเป็นภาษาโปรแกรมแบบโครงสร้างเช่นกัน ภาษาแบบ Object oriented เช่น จาวา ถูกจัดระบบให้ล้อมรอบข้อมูลภาษาโปรแกรมแบบโครงสร้างใช้ฟังก์ชันคอลเป็นส่วนหนึ่งของหน่วย

ของการจัดระบบ ผู้ออกแบบภาษาซีช่วยให้ leap (ภาษาสำหรับการแสดงกระบวนการที่มีความสัมพันธ์กัน) มีการพัฒนาก้าวไปข้างหน้า (ไม่ใช่ภาษาแรกที่ใช้ เนื่องจากภาษา ALGOL ก็เป็นภาษาแบบโครงสร้างเช่นกัน) ภาษาเหล่านี้ยังได้สร้างโครงสร้างสำหรับ บัฟเฟอร์โอเวอร์โฟลว์ ด้วย

แต่ละครั้งที่ฟังก์ชันหนึ่งๆ ถูกเรียกอาร์กิวเมนต์(ค่าหรือการอ้างอิงหนึ่งที่จะถูกส่งไปยังฟังก์ชัน) ที่จะถูกส่งไปยังฟังก์ชันนั้นจะถูกทำสำเนาไปยังพื้นที่ของหน่วยความจำที่เรียกว่า สแตก ในภาษาแอสแซมบลี คุณสามารถเก็บข้อมูลลงในสแตก โดยการพุช(การใส่บางสิ่งลงใน สแตก) และเรียกคืนโดยการพ็อพ (การนำบางสิ่งออกจากส่วนบนของสแตก) ทุกสถาปัตยกรรมของซีพียูที่ใช้กันในขณะนี้สนับสนุนแนวคิดเกี่ยวกับสแตก และมีรีจิสเตอร์(สแตกพอยต์เตอร์) พิเศษ และการทำงานสำหรับการพุชซึ่งและพ็อพพึ่งและมีตัวดำเนินการ (โอเปอร์เรเตอร์) ที่นำค่าที่อยู่ในหน่วยความจำ (เอดเดรส) ออกจากสแตกและทำสำเนาที่อยู่ในหน่วยความจำนี้ไปยังโปรแกรมเคาน์เตอร์ ซึ่งเป็นรีจิสเตอร์ซึ่งกำหนดที่อยู่ในหน่วยความจำของ คำสั่งต่อไปที่จะถูกรัน การเรียกฟังก์ชันทำให้มีการพุช ที่อยู่ในหน่วยความจำที่ส่งคืน(รีเทิร์นแอดเดรส)ไปยังสแตก

ปัญหาเกี่ยวกับการออกแบบนี้แสดงให้เห็นได้จากฟังก์ชันที่ถูกเรียก ตัวแปรใดๆ ก็ตามที่อยู่ภายในฟังก์ชัน นี้ถูกเก็บไว้ในพื้นที่จัดสรรของสแตกด้วย เช่น มีสายอักขระหนึ่ง เช่น ชื่อของไฟล์ที่ถูกเปิดขึ้นมา หรือรหัสผ่าน จำเป็นที่จะต้องได้รับการกำหนดค่าไว้ในฟังก์ชัน จำนวนของไบต์จะต้องได้รับการจัดสรรพื้นที่ในสแตกฟังก์ชัน จึงจะสามารถใช้พื้นที่ความจำส่วนนี้ได้ แต่พื้นที่ความจำส่วนนี้จะถูกเคลียร์โดยอัตโนมัติหลังจากฟังก์ชัน คืนค่าที่อยู่ในหน่วยความจำให้แล้ว เป็นการออกแบบที่มีระเบียบมาก แต่ภาษาซี ไม่ได้มีการตรวจสอบขนาดของข้อมูลเมื่อข้อมูลถูกเก็บไว้ใน ส่วนนี้ ทำให้มีช่องโหว่เล็ก ๆ สำหรับผู้โจมตี

ซับรูทีนภาษาซี ที่ทำสำเนาข้อมูลแต่ไม่ได้มีการตรวจสอบขนาดของข้อมูลเป็นสาเหตุของข้อผิดพลาด (เช่นเดียวกับโปรแกรมเมอร์ที่ใช้ฟังก์ชันคอลเหล่านี้) ฟังก์ชันคอล strcat(), strcpy(), sprintf(), vsprintf(), bcopy(), gets() และ scanf() สามารถนำมาใช้ประโยชน์จากช่องโหว่ได้ เนื่องจากฟังก์ชันเหล่านี้ไม่มีการตรวจสอบว่าบัฟเฟอร์ (พื้นที่ของหน่วยความจำ ใช้สำหรับเก็บข้อมูล) ที่ได้รับการจัดสรรพื้นที่ในสแตกมีขนาดใหญ่เพียงพอสำหรับข้อมูลที่จะถูกทำสำเนาลงในบัฟเฟอร์ หรือไม่ ขึ้นอยู่กับโปรแกรมเมอร์ว่าจะใช้ฟังก์ชันที่มีการตรวจสอบ(เช่น strncpy())หรือนับจำนวนไบต์ของข้อมูล ก่อนที่จะทำสำเนาลงในสแตกหรือไม่ ถ้าสามารถรู้ถึงซับรูทีนคอล ที่ถูกใช้ในทางที่ผิด แล้วคุณคิดว่าจะสามารถตรวจสอบการใช้ฟังก์ชันเหล่านี้ทั้งหมด แล้วปัญหาจะรับการแก้ไขตลอดไป อย่างนั้นใช่หรือไม่ จริงๆ แล้วมันไม่ง่ายเช่นนั้น ยังมีวิธีอื่น ๆ ที่จะทำคล้าย ๆ กัน และทำให้สามารถใช้ประโยชน์จากช่องโหว่ได้ และทำให้มีข้อผิดพลาด (เช่น การเพิ่มตัวอักษรหลาย ๆ ตัวเข้าไปในลูป

เมื่อขั้นตอนได้ถูกเซ็ตไว้แล้ว แต่การเข้าใจในส่วนนี้ยังเป็นส่วนที่เข้าใจได้ง่าย ผู้โจมตีต้องเข้าใจ ภาษาแอสแซมบลีอย่างเพียงพอด้วยไบต์โค้ด (ไบนารีไฟล์ที่ประกอบด้วยโปรแกรมที่

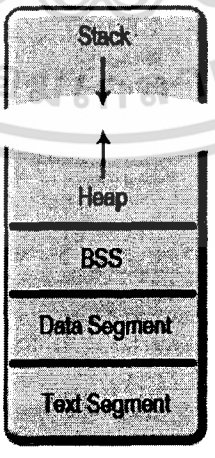
สามารถรันได้ สร้างมาจากลำดับคู่ของอ๊อปโค้ดและข้อมูล)ที่ถูกใช้โดยโพรเซสเซอร์ เช่น อินเทล เพนเทียม สำหรับการเขียนโค้ดเอ็กซ์พลอยต์ ในโค้ดของเอ็กซ์พลอยต์สำหรับบัฟเฟอร์โอเวอร์โฟลว์ โค้ดจะถูกเขียนลงในสแตก ให้เกินกว่าค่าที่อยู่ในหน่วยความจำที่ส่งค่าคืนและอาร์กิวเมนต์ของฟังก์ชันคอลต่อมา ค่าที่อยู่ในหน่วยความจำที่ส่งคืนจะถูกดัดแปลงเพื่อให้ชี้ไปยังจุดเริ่มต้นของโค้ด(โดยประมาณ) ต่อมาเมื่อฟังก์ชันคอลส่งคืนค่าแล้ว โค้ดของผู้โจมตีจะถูกรันแทนที่การรันของโปรแกรมธรรมดา

การทำให้ค่าที่อยู่ในหน่วยความจำที่ส่งคืนให้ชี้ไปยังที่ที่ถูกต้องเป็นส่วนหนึ่งของ "ศิลป์" ของการเอ็กซ์พลอยต์ของบัฟเฟอร์โอเวอร์โฟลว์ โดยทั่วไปอาร์กิวเมนต์ที่สามารถเปลี่ยนแปลงค่าถ่วงดุล (อ๊อปเซ็ต) ทำให้ผู้โจมตี สามารถพยายามเปลี่ยนที่อยู่ในหน่วยความจำต่าง ๆ โดยการปรับเปลี่ยนค่าเพื่อที่จะเปลี่ยนตำแหน่งของค่า ของที่อยู่ในหน่วยความจำที่ถูกส่งคืนที่ได้รับการดัดแปลงแล้วนั้น

ส่วนอื่นของศิลปะของโค้ดในระบบยูนิกซ์ค่อนข้างง่าย โปรแกรมสั้นๆ ที่สามารถถูกทำสำเนาจากตัวเอ็กซ์พลอยต์เพื่อรันโปรแกรมที่อยู่ในระบบ โดยส่วนมากจะเป็นตัวแปลคำสั่งของยูนิกซ์ คือ /bin/sh ถ้าผู้โจมตีสามารถเข้าสู่ระบบได้แล้วผลจากการรันเอ็กซ์พลอยต์มักจะทำให้ตัวแปรคำสั่งนั้นรันด้วยสิทธิของผู้ดูแลระบบ ถ้าผู้โจมตีอยู่ที่เครื่องอื่น การเชื่อมโยงแบบทีซีพีที่ใช้เพื่อเริ่มต้นการโจมตีก็จะเชื่อมโยง ไปสู่เชลล์ (ตัวแปลคำสั่งในยูนิกซ์) และมักจะมีสิทธิของผู้ดูแลระบบด้วย

2.4.1 การจัดการหน่วยความจำ

โพรเซส คือส่วนของโปรแกรมที่กำลังทำงานอยู่ ซึ่งในการที่จะทำให้โปรแกรมทำงานได้นั้น ระบบปฏิบัติการ จะต้องทำการโหลดโปรแกรมไปยังหน่วยความจำก่อน โดยที่การจัดสรรหน่วยความจำให้แต่ละโพรเซสนั้น เราสามารถแบ่งออกเป็น 5 ส่วนใหญ่ๆ ด้วยกันคือ



รูปที่ 2.4 ส่วนต่างๆ ของหน่วยความจำ

1. ส่วนของข้อความ

ส่วนของข้อความ หรือส่วนของโค้ด โดยในส่วนนี้ จะประกอบไปด้วยคำสั่งต่างๆ (อินสตรัคชัน) และโค้ดของโปรแกรม โดยยูนิกซ์และลินุกซ์จะใช้ส่วนนี้ร่วมกันในแต่ละโพรเซสที่มาจากโปรแกรมเดียวกัน จึงทำให้มีส่วนของคำสั่งเพียงหนึ่งเดียวสำหรับโปรแกรมเดียวกันอยู่บนหน่วยความจำในขณะใดขณะหนึ่งเท่านั้น

2. ส่วนของข้อมูล

ส่วนนี้จะเก็บตัวแปรสากล และตัวแปรคงที่ ที่มีการให้ค่าเริ่มต้นแล้ว เราเรียกส่วนนี้ว่า ส่วนของข้อมูลที่ให้ค่า (ข้อมูลเริ่มต้น) โดยแต่ละโพรเซสก็จะมีส่วนนี้เป็นของตนเอง

3. ส่วนบีเอสเอส

ตัวแปรสากล และตัวแปรคงที่ ที่มีค่าคงที่เป็นศูนย์โดยอัตโนมัติจะถูกเก็บไว้ในส่วนนี้ โดยสามารถเรียกส่วนนี้ได้อีกว่าส่วนของตัวแปรที่มีค่าเป็นศูนย์ โดยแต่ละโพรเซสก็จะมีส่วนนี้แยกจากกัน

4. ส่วนฮีป

ฮีปเป็นส่วนที่ตัวแปรแบบไดนามิก(ซึ่งเกิดจากการใช้คำสั่ง malloc()) โดยฮีปจะขยายไปทางด้านบน นั่นคือ เมื่อเราใส่ค่าลงไปในฮีปค่านั้นจะถูกบรรจุลงในตำแหน่งที่มีค่าสูงกว่าค่าที่ใส่ก่อนหน้า

5. ส่วนสแตก

ส่วนของสแตกจะเป็นส่วนของตัวแปรเฉพาะ ที่ถูกเก็บไว้ ตัวอย่างของตัวแปรเฉพาะที่คือตัวแปรที่อยู่ในฟังก์ชันย่อยโดยไม่ได้ประกาศให้เป็นตัวแปรคงที่ โดยสแตกจะขยายลงไปทางด้านล่าง นั่นคือเมื่อเราใส่ค่าลงไปในสแตกค่านั้นจะอยู่ใน ตำแหน่งที่มีค่าน้อยกว่าค่าที่ใส่ก่อนหน้านั้น ซึ่งจะสวนทางกับส่วนของฮีป

การเรียกฟังก์ชัน ในระบบยูนิกซ์ การรียกใช้ฟังก์ชันแบ่งออกเป็น 3 ขั้นตอนด้วยกันคือ

1. ส่วนนำ โดยเฟรมพอยต์เตอร์ปัจจุบันจะถูกเก็บเอาไว้ และจะทำการจองพื้นที่หน่วยความจำที่จำเป็นสำหรับฟังก์ชัน
2. การเรียกค่า โดย ค่าพารามิเตอร์ต่างๆจะถูกเก็บในสแตกและอินสตรัคชันพอยต์เตอร์ จะถูกจัดเก็บลงไปในสแตกสำหรับโปรแกรมจะได้ทราบว่าจะต้องทำคำสั่งไหนต่อไปหลังจากได้เรียกใช้ฟังก์ชันจบแล้ว
3. การคืนค่า หรือส่วนสุดท้าย สถานะของสแตก ก่อนเรียกฟังก์ชันจะถูกเรียกกลับคืน

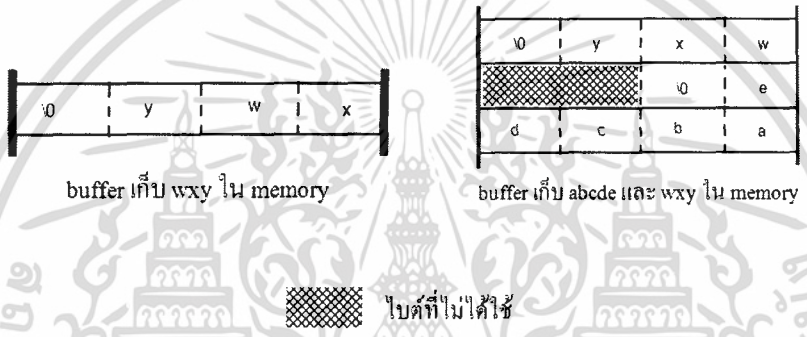
การบุกรุกสามารถเป็นไปได้เพราะว่า เมื่อฟังก์ชันถูกเรียก ในขั้นตอนที่จะกลับมาส่วนก่อนที่ฟังก์ชันนั้นจะถูกเรียกไปนั้น ตำแหน่งของคำสั่งที่จะทำต่อไปจะถูกทำสำเนาจากสแตกไปยังรีจิสเตอร์อีไอพี ซึ่งถ้าผู้โจมตีสามารถแก้ไขสแตก ในส่วนนั้นให้ชี้ไปยังคำสั่งอันไม่พึงประสงค์ที่ผู้โจมตีส่งเข้ามา เครื่องก็จะทำการเรียกคำสั่งนั้นซึ่งการโจมตีก็จะสมบูรณ์

2.4.2 บัฟเฟอร์ และส่วนที่เสี่ยงต่อการถูกโจมตี

ในภาษาซี ข้อความหรือบัฟเฟอร์จะถูกแทนด้วยตัวชี้ตำแหน่งซึ่งชี้ไปยังตำแหน่งไบต์ตัวแรกและเราจะสามารถรู้ได้ว่าถึงจุดสิ้นสุดแล้วเมื่อพบไบต์ว่าง ซึ่งหมายความว่าเราไม่มีทางที่จะทราบค่าที่แน่นอนที่เราต้องสำรองไว้สำหรับบัฟเฟอร์

ทีนี้เรามาลองดูการจัดการกับบัฟเฟอร์ในหน่วยความจำ อย่างแรกเนื่องจากเราไม่สามารถรู้ขนาดของบัฟเฟอร์ได้ แต่หน่วยความจำมีพื้นที่ให้บัฟเฟอร์อย่างจำกัด ในการที่จะป้องกันการล้น (โอเวอร์โฟลว์) นั้นค่อนข้างจะลำบากในการป้องกัน นี่จึงเป็นปัญหาที่สามารถพบได้บ่อยๆ อย่างเช่นการใช้คำสั่ง strcpy() โดยไม่ระวัง ทำให้ผู้ใช้สามารถนำสำเนาบัฟเฟอร์ไปยังอีกส่วนได้

นี่เป็นเหตุการณ์ที่จะทำให้เห็นภาพได้ชัดเจนขึ้น โดยตัวอย่างแรกนั้นบัฟเฟอร์ได้เก็บค่า wxy ส่วนที่สองเก็บค่า wxy และตามด้วย abcde เอาไว้



รูปที่ 2.5 แสดงบัฟเฟอร์ในหน่วยความจำ

จากรูปในกรณีด้านขวา เรามีไบต์ที่ไม่ใช้อยู่ 2 อันเพราะเวิร์ด (4 ไบต์) จะถูกใช้ในการเก็บข้อมูล แต่จากตัวอย่างเราต้องการเพียง 6 ไบต์ ซึ่งต้องใช้อีก 2 เวิร์ด จึงมีที่ว่างเหลือ 2 ไบต์

ตัวอย่างโปรแกรมที่มีความเสี่ยงในการโจมตีจากบัฟเฟอร์แสดงให้เห็นดังนี้

```
// jc.c
#include <stdio.h>
int main()
{   char jayce[4]= “Oum” ;
    char herc[8]= “Gillian” ;
    strcpy(herc, “BrookFlora”) ;
    printf(“%s\n”, jayce) ;
    return 0 ;
}
```

บัฟเฟอร์ ทั้ง 2 ส่วนจะถูกจัดเก็บในสแตก ดังรูปข้างล่าง เมื่อตัวอักษร 10 ตัว ถูกทำสำเนาไปยัง บัฟเฟอร์ ซึ่งสามารถรองรับได้เพียง 8 ตัวอักษรนั้น บัฟเฟอร์ แรกจะถูกเปลี่ยนค่าไป

การทำสำเนาครั้งนี้ก่อให้เกิดบัฟเฟอร์โอเวอร์โฟลว์ซึ่งในหน่วยความจำ ก่อนและหลังที่เกิดบัฟเฟอร์โอเวอร์โฟลว์นั้นเป็นดังนี้

\0	m	u	O
\0	n	a	i
l	l	i	G

ก่อนเกิด buffer overflow

\0	\0	a	r
o	l	F	k
o	o	r	B

หลังเกิด buffer overflow

รูปที่ 2.6 แสดงการเกิดบัฟเฟอร์โอเวอร์โฟลว์

นี่คือสิ่งที่จะเกิดขึ้นเมื่อทำการเรียกโปรแกรมนี้ขึ้นมา

```
SW@ce.kmitl.ac.th:~$ gcc jayce.c
SW@ce.kmitl.ac.th:~$ ./a.out
ra
SW@ce.kmitl.ac.th:~$
```

2.4.3 การล้นของบัฟเฟอร์ (บัฟเฟอร์โอเวอร์โฟลว์)

จากที่กล่าวไปแล้วบัฟเฟอร์จะถูกใช้ในการเก็บข้อมูล ซึ่งอยู่ในหน่วยความจำ สิ่งที่เราสนใจนั่นคือการเก็บสตริงในตัวบัฟเฟอร์เองนั้น ไม่มีวิธีการในการจัดการกับการรับค่าข้อมูลที่มากเกินจำนวนที่จองไว้ เช่น ถ้าเราทำการประกาศตัวแปรชนิดสตริงโดยให้มีขนาด 10 ไบต์

```
//overflow.c
main() {
char str1[10];
strcpy (str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAA");
}
```

รูปที่ 2.7 เมื่อใส่อินพุตเป็น A จำนวนมาก

เมื่อทำการคอมไพล์ และประมวลผล

```
Jinx:/home/waan/project# gcc -ggdb -o overflow overflow.c
Jinx:/home/waan/project# ./overflow
Segmentation fault
Jinx:/home/waan/project#
```

รูปที่ 2.8 แสดงการรันโปรแกรมโอเวอร์โฟลว์

สังเกตได้ว่าจะเกิดเซ็กเมนเทชันฟอลต์ขึ้น

```
Jinx:/home/waan/project# gdb ./overflow
GNU gdb 6.3-debian
Copyright 2004 Free Software Foundation, Inc.
GDB is free software, covered by the GNU General Public License, and you are
welcome to change it and/or distribute copies of it under certain conditions.
Type "show copying" to see the conditions.
There is absolutely no warranty for GDB.  Type "show warranty" for details.
This GDB was configured as "i386-linux"...Using host libthread_db library "/lib/
libthread_db.so.1".

(gdb) run
Starting program: /home/waan/project/overflow

Program received signal SIGSEGV, Segmentation fault.
0x41414141 in ?? ()
(gdb) info reg eip
eip            0x41414141       0x41414141
(gdb)
```

รูปที่ 2.9 แสดงการตรวจดีบักโปรแกรมโอเวอร์โฟลว์

เมื่อเราลองทำการดีบักจะเห็นได้ว่าโปรแกรมเกิดความผิดพลาดขึ้นเมื่อพยายามจะประมวลผลคำสั่งที่ตำแหน่ง 0x41414141 ซึ่งเป็นเลขฐาน 16 ของตัวอักษร A ซึ่งจะเห็นได้ว่ารีจิสเตอร์อีไอพีจะถูกเขียนทับด้วย A ซึ่งเป็นสาเหตุที่โปรแกรมพยายามประมวลผลที่ตำแหน่ง 0x41414141 ซึ่งอยู่ภายนอกโพรเซสทำให้เกิดเซ็กเมนเทชันฟอลต์ขึ้น โดยในตัวอย่างนี้จะเป็นการล้นที่เรียกว่าสแตก (สแตกโอเวอร์โฟลว์)

2.4.4 การล้นของสแตก (สแตกโอเวอร์โฟลว์)

เมื่อมีการเรียกฟังก์ชันขึ้นมาทำงาน ค่าที่อยู่ในรีจิสเตอร์อีไอพีจะถูกสำเนาเก็บไว้ในสแตก เมื่อฟังก์ชันนั้นทำงานเสร็จ โปรแกรมจะนำค่าในอีไอพีที่ได้ทำการสำเนาลงไปในสแตก กลับมาใส่อีไอพี ตามเดิมเพื่อที่จะทำงานต่อไปได้ จุดนี้เองที่เป็นจุดที่ทำให้เกิดการโจมตีโดยอาศัยการแก้ไขค่าอีไอพีในสแตก เมื่อฟังก์ชันนั้นทำงานเสร็จ

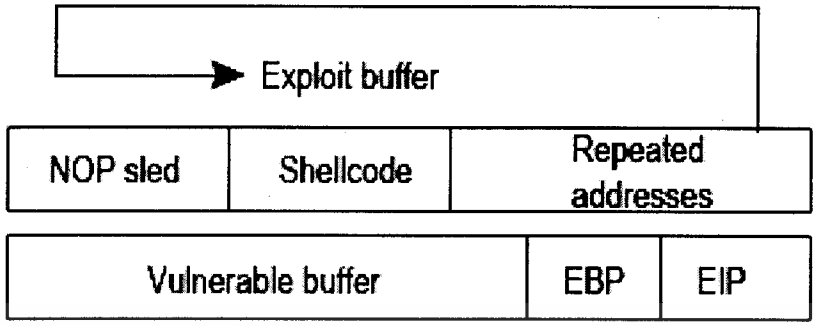
การแก้ไขค่าการคืนค่าแอดเดรสจะต้องอยู่ในตำแหน่งที่ตรงพอดีและวนซ้ำมันจนกระทั่งทำการเขียนทับค่าอีไอพีที่สำเนาไว้ในสแตก ถึงแม้ว่าเราสามารถที่จะชี้ตำแหน่งไปยังตำแหน่งของเชลล์โค้ดได้เลย แต่เป็นการง่ายกว่าที่จะนำล้อเลื่อนเอ็นโอพีมาใช้ โดยวางไว้หน้าตำแหน่งเชลล์โค้ดแล้วชี้ไปที่ล้อเลื่อนเอ็นโอพีแทน

- ล้อเลื่อนเอ็นโอพี มีความหมายว่าไม่มีการทำงานอะไรเลย แต่ให้เลื่อนไปยังคำสั่งต่อไป คำสั่งนี้จึงถูกนำมาประยุกต์ใช้ในการเติมเข้าไปด้านหน้าเอ็กซ์พลอยต์บัฟเฟอร์ ซึ่งจะเรียกว่าล้อเลื่อนเอ็นโอพีถ้า อีไอพี ชี้ไปยังตำแหน่งเอ็นโอพี โปรแกรมก็จะทำการเลื่อนต่อไปเรื่อยๆ โดยทั่วไปแล้วในระบบ x86 จะใช้อ็อปโค้ดเป็น 0x90
- เชลล์โค้ดเป็นคำสั่งที่ใช้เรียกภาษาเครื่อง ที่แฮกเกอร์ใส่เข้ามาเพื่อให้ทำคำสั่งต่างๆ โดยส่วนมากจะอยู่ในรูปเลขฐาน 16

```
//shellcode.c
char shellcode[] =
"\x31\xc0\x31\xdb\xb0\x17\xcd\x80"
"\xeb\x1f\x5e\x89\x76\x08\x31\xc0\x88\x46\x07\x89\x46\x0c\xb0\x0b"
"\x89\xf3\x8d\x4e\x08\x8d\x56\x0c\xcd\x80\x31\xdb\x89\xd8\x40\xcd"
"\x80\xe8\xdc\xff\xff\xff/bin/sh";

void main() {
int *ret;
ret = (int *)&ret + 2;
(*ret) = (int)shellcode;
}
~
```

รูปที่ 2.10 แสดงลักษณะของเชลล์โค้ด



รูปที่ 2.11 แสดงขั้นตอนการคืนค่าแอดเดรส

จากรูปจะเห็นได้ว่าค่าแอดเดรสจะเขียนทับค่าอีไอพีซึ่งจะชี้ไปที่ล้อเลื่อนเอ็นโอพี ซึ่งจะเลื่อนไปที่เชลล์โค้ดในที่สุด

2.4.5 การล้นของ ฮีป (ฮีปโอเวอร์โฟลว์)

ฮีปเป็นส่วนที่อยู่บนหน่วยความจำ ซึ่งเก็บตัวแปรแบบไดนามิกโดยตัวแปรพวกนี้เกิดขึ้นมาจากการใช้คำสั่งจำพวก malloc() หรือ new() โดยฮีปจะขยายจากส่วนต่ำของหน่วยความจำไปสู่ส่วนมากของหน่วยความจำซึ่งกันข้ามกับสแตก ในการโจมตีด้วยด้วยช่องโหว่แบบสแตกจะเน้นไปที่การเขียนทับค่าคืนค่าแอดเดรส แต่ถ้าเป็นฮีปนั้นจะเป็นการพยายามเขียนค่าตัวแปรที่สำคัญ โดยส่วนมากทำเพื่อให้ได้สิทธิการเป็นรูตของโปรแกรมมาครอบครอง

2.5 การเอ็กซ์พลอยต์

ในวิชาการทางความปลอดภัยทางคอมพิวเตอร์ เอ็กซ์พลอยต์คือส่วนหนึ่งของซอฟต์แวร์ , ก้อนข้อมูลชุดหนึ่ง , หรือลำดับของคำสั่งที่หาประโยชน์จากบั๊ก , ข้อบกพร่อง หรือช่องโหว่เพื่อที่จะทำพฤติกรรมใดๆ ตามความมุ่งหมาย ซึ่งส่วนใหญ่แล้วจะเข้าควบคุมระบบคอมพิวเตอร์หรือการยกระดับสิทธิ หรือการโจมตีแบบการโจมตีเพื่อปิดบริการ

มีหลายวิธีที่แยกแยะประเภทของการเอ็กซ์พลอยต์โดยปกติแล้วคือ วิธีการที่จะเข้าไปเอ็กซ์พลอยต์ซอฟต์แวร์ที่อ่อนแอ แบ่งได้เป็น

- รีโมตเอ็กซ์พลอยต์ ทำงานผ่านเน็ทเวิร์กและเอ็กซ์พลอยต์ช่องโหว่ทางความปลอดภัยโดยไม่ต้องเข้าไปยังระบบก่อน
- โลคอลเอ็กซ์พลอยต์ จำเป็นต้องเข้าไปอยู่ในระบบนั้นก่อน แล้วยกระดับตัวเองขึ้นเป็นผู้ดูแลระบบ

- เอ็กซ์พลอยต์แอปพลิเคชันเครื่องลูกข่าย ประกอบด้วยเซิร์ฟเวอร์ที่ส่ง เอ็กซ์พลอยต์ หากเข้าถึงแอปพลิเคชันเครื่องลูกข่ายได้ซึ่ง การเอ็กซ์พลอยต์แอปพลิเคชันเครื่องลูกข่าย อาจต้องใช้การโต้ตอบกับผู้ใช้ด้วย

หรือแบ่งประเภทด้วยการกระทำต่อระบบ เช่น

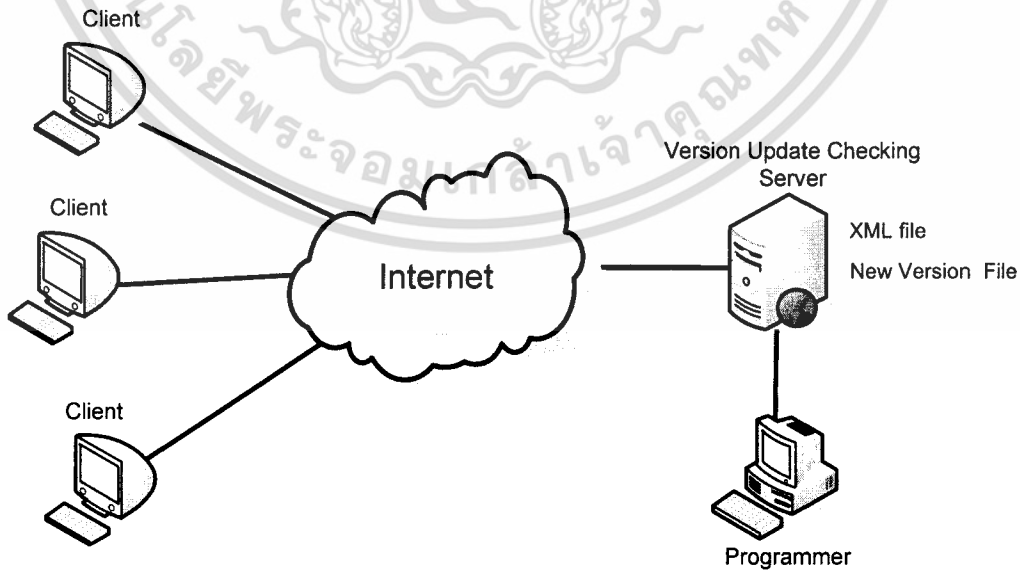
- การเข้าถึงข้อมูลที่ไม่ได้รับอนุญาต
- รันโค้ดของผู้บุกรุก
- การโจมตีเพื่อปิดบริการ

การเอ็กซ์พลอยต์จำนวนมากมุ่งหมายที่จะใช้สิทธิผู้ดูแลระบบในการเข้าสู่ระบบ อย่างไรก็ตามยังมีอีกหลายแนวทางในการเอ็กซ์พลอยต์เริ่มแรกคือการเข้าระบบด้วยสิทธิระดับต่ำก่อน และยกระดับสิทธิขึ้นไปจนกระทั่งได้รูต

โดยทั่วไปเอ็กซ์พลอยต์หนึ่งจะสามารถเข้าหาผลประโยชน์ต่อซอฟต์แวร์หนึ่งๆ โดยเฉพาะเจาะจงเท่านั้น โดยมากเมื่อเอ็กซ์พลอยต์นั้นถูกเผยแพร่ออกไป จะทำให้ซอฟต์แวร์ทำการปรับปรุงแพทช์เพื่อแก้ปัญหารอยรั่วนั้น แต่เอ็กซ์พลอยต์ที่ยังคงไม่เป็นที่เปิดเผย ถูกเรียกว่าซีโรเดย์ เอ็กซ์พลอยต์

2.6 รูปแบบการตรวจสอบการปรับปรุงเวอร์ชันผ่านเอ็กซ์เอ็มแอล

ในการศึกษารูปแบบของการตรวจสอบการปรับปรุงเวอร์ชันนั้น แต่ละโปรแกรมซึ่งแตกต่างกันย่อมจะมีวิธีการที่แตกต่างกันไปแล้วแต่ผู้พัฒนา แต่สำหรับโครงงานนี้ ได้นำเสนอรูปแบบวิธีหนึ่ง ซึ่งเป็นที่นิยมในปัจจุบัน ซึ่งคือการใช้เอ็กซ์เอ็มแอล สำหรับใช้เปรียบเทียบเวอร์ชัน



รูปที่ 2.12 รูปแบบของเซิร์ฟเวอร์ที่ให้บริการตรวจสอบการปรับปรุงเวอร์ชัน

จากรูป จะมีเซิร์ฟเวอร์ตัวหนึ่งซึ่งทำหน้าที่คอยให้บริการตรวจสอบการปรับปรุงเวอร์ชันกับบรรดาไคลเอ็นท์ทั้งหลายที่จะมีการร้องขอตรวจสอบเวอร์ชันใหม่ๆ ของซอฟต์แวร์ ในรูปแบบของเอ็กซ์เอ็มแอลโดยหากโปรแกรมเมอร์ มีซอฟต์แวร์เวอร์ชันใหม่ ก็จะแก้ไขตรงไฟล์เอ็กซ์เอ็มแอล (เพื่อไว้เปรียบเทียบเวอร์ชัน) และอัพโหลดซอฟต์แวร์ใหม่ไว้ให้บริการบนเซิร์ฟเวอร์

โดยทางฝั่งผู้พัฒนาโปรแกรม ก็มีรูปแบบในการเพิ่มฟีเจอร์ตรวจสอบการปรับปรุงเวอร์ชันตามขั้นตอนต่อไปนี้

ขั้นที่ 1 : ดึงข้อมูลเวอร์ชันปัจจุบันของโปรแกรม

สำหรับการดึงข้อมูลเวอร์ชันของโปรแกรมนั้น มีวิธีการที่หลากหลาย และมีคลาสจำนวนมากที่สนับสนุน แต่มีคลาสหนึ่งที่แนะนำคือ CFileVersionInfo เขียนโดย อาร์เมน ฮาโกบยอน ซึ่งสามารถดึงเวอร์ชันออกมาได้จากไฟล์

โค้ดตัวอย่าง

```
CFileVersionInfo fVer;

if( FALSE == fVer.Open( "your_exe_file.exe" ) )
    return;

DWORD dwExeMajor = fVer.GetFileVersionMajor();
DWORD dwExeMinor = fVer.GetFileVersionMinor();
DWORD dwExeBuild = fVer.GetFileVersionBuild();
```

Major , Minor , Build เป็นส่วนหนึ่งของตัวเลขเวอร์ชันทั้งหมด เวอร์ชัน 1.2.40 หมายถึง Major=1 , Minor=2 , Build=40 ถ้าคุณต้องการดึงค่าเวอร์ชันจากเอ็กซีคิวอาเบิลไฟล์ คุณต้องใช้อ๊อบเจ็ค CFileVersionInfo และใช้ฟังก์ชัน GetModuleFileName() สำหรับรับชื่อไฟล์

ขั้นที่ 2 : ดาวน์โหลดไฟล์เอ็กซ์เอ็มแอลจากอินเตอร์เน็ท มาไว้ยังเครื่อง

สำหรับดาวน์โหลดไฟล์จากอินเตอร์เน็ท ก็มีหลากหลายทาง แต่ที่นี้แนะนำให้ใช้คลาส CInternetSession และ CHttpFile ซึ่งฟังก์ชันนี้จะมีพารามิเตอร์รับยูอาร์แอล, ดาวน์โหลดไฟล์ลักษณะเท็กซ์ เป็นอ๊อบเจ็ค CString และรีเทิร์นอ๊อบเจ็คสำหรับการดำเนินการที่เพิ่มเข้ามา

ขั้นที่ 3 : วิเคราะห์ไฟล์เอ็กซ์เอ็มแอลดึงเอาค่าเวอร์ชันเก็บไว้

นี่คือส่วนหนึ่งของ เอ็กซ์เอ็มแอล ไฟล์ที่จำเป็นต่อการตรวจสอบการปรับปรุงเวอร์ชัน

```
<?xml version="1.0" encoding="UTF-8"?>
<XML_DIZ_INFO>
.
.
```

```
        <Program_Info>
                .
                .
        <Program_Name>UpdateCheck_Example</Program_Name>
        <Program_Version>1.0.0</Program_Version>
        .
        .
    </Program_Info>
.
.
</XML_DIZ_INFO>
```

จากโค้ดแท็ก "Program_Info" มีส่วนประกอบเป็น "Program_Version" ซึ่งค่าตรงนี้เป็นเลขเวอร์ชันล่าสุดของโปรแกรม สิ่งที่ต้องทำจากนี้คืออ่าน String ที่เป็นเลขเวอร์ชันและนำไปเทียบกับเวอร์ชันของโปรแกรมปัจจุบันของผู้ใช้

ซึ่งเราต้องการโซลูชันสำหรับตัดคำในเอ็กซ์เอ็มแอลไฟล์ เช่น เขียนให้ค้นหาคำ <Program_Version> และ </Program_Version> และตัดเอาค่าตรงกลางออกมา (ที่เป็น สตริงค่าเวอร์ชัน) หรือใช้ไลบรารีไทนี่เอ็กซ์เอ็มแอลซึ่งรวดเร็วและง่ายต่อการใช้เป็นโอเพ่นซอร์ส

```
// Get root node
TiXmlNode* pDIZ = xmlDoc.FirstChild( "XML_DIZ_INFO" );

if( NULL == pDIZ )
    return;

// Get the program info node (parent node is root node above)
TiXmlNode* pPInfo = pDIZ->FirstChild( "Program_Info" );

if( NULL == pPInfo )
    return;

// Get the version node (parent node is program info above)
TiXmlNode* pPVer = pPInfo->FirstChild( "Program_Version" );

if( NULL == pPInfo )
    return;

// Cast our version node to an element
TiXmlElement* pPVerEl = pPVer->ToElement();
ASSERT( NULL != pPVerEl );

// Extract the text of this node into a CString object
CString strVer = pPVerEl->GetText();

DWORD dwFileMajor = 0;
DWORD dwFileMinor = 0;
DWORD dwFileBuild = 0;

// Extract the version information parts into separate variables
char szBuf[ 32 ];
sprintf( szBuf, "%s", strVer.Getบัฟเฟอร์( 1 ) );
sscanf( szBuf, "%d.%d.%d", &dwFileMajor, &dwFileMinor, &dwFileBuild );
```

ขั้นที่ 4 : เปรียบเทียบระหว่างเวอร์ชันทั้งคู่ , และแจ้งเตือนผู้ใช้ เมื่อมีการอัพเดต

เมื่อเราได้มาทั้ง 2 เวอร์ชัน อันหนึ่งเป็นของเอ็กซ์เอ็มแอล ส่วนอีกอันเป็นเวอร์ชันของโปรแกรมตัวปัจจุบัน. ซึ่งเวอร์ชันประกอบด้วย 3 ส่วน (major , minor , build) ซึ่งเราจะนำแต่ละค่ามาเปรียบเทียบกัน

นี่เป็นตัวอย่างโค้ดเปรียบเทียบค่าเวอร์ชัน

```
DWORD dwExeSum = ( dwExeMajor * 100 ) + ( dwExeMinor * 10 ) + dwExeBuild;
DWORD dwFileSum = ( dwFileMajor * 100 ) + ( dwFileMinor * 10 ) +
dwFileBuild;

if( dwFileSum > dwExeSum )
{
    m_strVer.Format( "%d.%d.%d", dwFileMajor, dwFileMinor, dwFileBuild );
    bUpdate = true;
}

if( bUpdate )
{
    MessageBox()
}
```

นำการคูณเข้าไป เพื่อให้แน่ใจได้ว่า major เวอร์ชันค่าต่ำ จะเปรียบเทียบกับ minor ค่าสูง ได้ถูกต้อง เช่น 4.2.1 มากกว่า 3.7.3

บทที่ 3
ความปลอดภัยในขั้นตอนการวิเคราะห์และออกแบบ

"Malicious hackers don't create security holes; they simply exploit them. Security holes and vulnerabilities – the real root cause of the problem – are the result of bad software design and implementation."

-Terry Stanley, VP Security, Master Card-

การพัฒนาซอฟต์แวร์ตามหลักของวิศวกรรมซอฟต์แวร์ ตามวงจรชีวิตของซอฟต์แวร์ ให้มีความทนทานต่อการละเมิดความปลอดภัย ในขั้นตอนแรกเริ่มก็คือขั้น การวิเคราะห์ (Analysis) ซึ่งรวมถึงการสำรวจความต้องการของโปรแกรม ว่ามีความต้องการอะไร ลำดับงานการทำงานของความจำเป็น กำหนดทรัพยากรที่สนับสนุน กำหนดทีมงานในการพัฒนาโปรแกรม

ศึกษาความเป็นไปได้ในการที่จะใช้โปรแกรมเพื่อแก้ปัญหา , วิเคราะห์รายละเอียดภายในว่าระบบทำงานอย่างไรและตอบสนองต่อความต้องการของผู้ใช้แค่ไหน ในขั้นตอนของการวิเคราะห์นั้น เป็นขั้นตอนที่มีความสำคัญอย่างมากไม่ควรรีบเร่ง เนื่องจากโครงการพัฒนาจำนวนมากประสบความล้มเหลวจากการวิเคราะห์ที่ผิดพลาด

การออกแบบ (Design) ในขั้นตอนนี้จะเป็นการนำเอาสิ่งที่ได้จากการวิเคราะห์ มาทำการออกแบบเป็นระบบงาน สำหรับการพัฒนาในขั้นตอนถัดไป เช่น การออกแบบ ฟอร์ม(Form) , รายงาน(Report) , ไฟล์(Files) , ฐานข้อมูล(Database) , โปรแกรม(Program) และ การออกแบบกระบวนการทำงาน (Process design) เป็นต้น ในการออกแบบจะเน้นการวิเคราะห์ข้อมูลระบบตรรกะของการนำไปพัฒนาระบบงานจริง หรือ ให้ความสนใจในแง่เชิงเทคนิค ทั้งตัวภาษา หรือเครื่องมือในการพัฒนาระบบงาน ผลที่ได้จากขั้นตอนนี้ คือ เอกสารการออกแบบ หรือ Program specification ซึ่งจะถูกนำไปพัฒนา ในลำดับถัดไป

ซึ่งแต่ละซอฟต์แวร์ต่างมีจุดประสงค์ในการทำงานที่แตกต่างกันไปรวมทั้งมีโครงสร้างที่แตกต่างกันด้วย ในการที่จะเพิ่มความปลอดภัยเข้าไปในซอฟต์แวร์ในขั้นตอนนี้นั้น จึงอาจเป็นลักษณะของข้อควรปฏิบัติ ดังนี้

3.1 หลักการ 18 ข้อในการพัฒนาโปรแกรมให้มีความปลอดภัย

3.1.1 เริ่มต้นจากการตั้งคำถาม

เมื่อเริ่มต้นที่จะออกแบบหรือแก้ไขโปรแกรมประยุกต์ใดๆก็ตามสิ่งที่ตามมาก็คือการปรับปรุงแก้ไขส่วนต่างๆรวมถึงช่องโหว่ความปลอดภัยที่อาจเกิดขึ้นได้อีกด้วย เพราะฉะนั้น

การเริ่มต้นที่ดีควรจะมีการตั้งคำถาม เพื่อให้ทราบถึงปัญหาทั้งหมด โดยสามารถแยกปัญหาออกเป็นส่วนใหญ่ๆ ได้แก่

- ปัญหาที่เกี่ยวกับตัวผู้ออกแบบ เช่น
  - อะไรที่ทำให้ผิดพลาดได้บ้าง
  - อะไรคือสิ่งที่ควรป้องกัน
  - อะไรคือจุดอ่อนที่สำคัญที่สุดในการป้องกัน
  - ปัญหาที่เกี่ยวกับทรัพยากร
  - ได้นำใช้ไลบรารีที่มีอยู่มาใช้หรือไม่
  - อะไรคือแนวทางที่เหมาะสม
- ปัญหาที่เกี่ยวกับโปรแกรมประยุกต์ เช่น
  - ใครคือผู้ใช้หลักของโปรแกรมประยุกต์นี้
  - ใครคือผู้ที่เข้าถึงโปรแกรมประยุกต์นี้ทั้งซอร์สโค้ด หรือขณะที่ทำงานอยู่
  - โปรแกรมประยุกต์นี้ทำงานอยู่บนสิ่งแวดล้อม (environment) อะไร เช่น ทำงานอยู่บนเว็บไซต์ หรือทำงานอยู่ในองค์กรประเภทใด
  - สามารถทราบจำนวนของผู้ใช้ในปัจจุบัน หรือประมาณการณ์จำนวนของผู้ใช้ในอนาคตได้หรือไม่ รวมถึงผลกระทบที่อาจจะเกิดขึ้นเมื่อมีการเปลี่ยนแปลงส่วนต่างๆ ของโปรแกรมประยุกต์ในอนาคต
- ปัญหาที่เกี่ยวกับจุดมุ่งหมาย
  - ความปลอดภัยเท่าใดถึงจะเพียงพอที่โปรแกรมประยุกต์ควรจะมี จำนวนเงินเท่าใดที่จะต้องลงทุนเพื่อให้โปรแกรมประยุกต์มีความปลอดภัย
  - ถ้าผู้ใช้ตัดสินใจละเลยต่อความปลอดภัย จะต้องทำอย่างไร และจะรู้ได้อย่างไร

คำถามเหล่านี้จะช่วยวิเคราะห์และอธิบายถึงความเสี่ยงในขั้นต้นที่ผู้ออกแบบโปรแกรมประยุกต์ไม่ควรมองข้าม เพราะสิ่งเหล่านี้เป็นกระบวนการหนึ่งในวิศวกรรมซอฟต์แวร์อีกด้วย

3.1.2 เลือกจุดมุ่งหมาย

ในหลักการทั้งหมด หลักการข้อนี้อาจจะเป็นข้อที่ยากที่สุดในการปฏิบัติ มันยากที่ต้องรอจนกระทั่งมั่นใจว่าผู้ออกแบบโปรแกรมประยุกต์เข้าใจว่าสิ่งใดต้องทำบ้างก่อนที่จะตัดสินใจ

เช่น ในการสร้างบ้านนั้น จะต้องมีการคุยกับสถาปนิก และวิศวกรว่าจะสร้างไปเพื่อทำสิ่งใด บ้าน ที่อยู่อาศัย ตึกแถว หรือคอนโดมิเนียม สถานที่กว้างเท่าใด เพื่อที่จะได้สร้างได้ตามความต้องการและตรงจุดประสงค์ของงานมากที่สุด เป็นต้น

3.1.3 ตัดสินใจว่าความปลอดภัยเท่าใดที่เพียงพอ

คำถามนี้เป็นคำถามที่มีผู้คนมากมายมักจะเกิดความสงสัยในการออกแบบโปรแกรมประยุกต์ ว่าโปรแกรมประยุกต์ที่ดีควรจะมีความปลอดภัยเท่าใด จุดมุ่งหมายจึงไม่ใช่การออกแบบโปรแกรมประยุกต์ที่มีความปลอดภัยมากที่สุด ที่สามารถรับประกันข้อผิดพลาดได้ทุกประการ เพราะนั่นหมายถึงการใช้เวลา หรือทรัพยากรต่างๆ ที่มากมาย ในความเป็นจริง ถ้าเป็นสภาวะแวดล้อมที่มีการแข่งขันทางการค้า จะต้องทราบว่าความปลอดภัยเท่าใดที่ควรมีเมื่อเปรียบเทียบกับคู่แข่งทางการค้า เพราะฉะนั้นในทางธุรกิจ จึงมักจะทำให้ซอฟต์แวร์นั้นเน้นทางด้านความสามารถในการใช้งานมากกว่าความปลอดภัย

3.1.4 มีการทำงานตามมาตรฐานวิศวกรรม

ความปลอดภัยของโปรแกรมประยุกต์นั้นจะเกิดขึ้นได้ก็ต่อเมื่อผู้ออกแบบนั้นมีการออกแบบที่ดี และมีความชำนาญในการออกแบบที่เพียงพอ โปรแกรมที่ถูกโจมตีส่วนมากนั้นประกอบด้วยปัจจัยสำคัญ ได้แก่

- ขาดการออกแบบ
- จุดอ่อนของผู้ออกแบบ
- การเขียนโปรแกรมที่ผิดพลาด

สถาปัตยกรรมความปลอดภัยที่ดีนั้นสามารถกำจัดปัจจัยใน 2 ข้อแรกได้ แต่ถ้าเกิดจากการเขียนโปรแกรมที่ไม่ชำนาญก็มีโอกาสเกิดช่องโหว่ความปลอดภัยได้มาก

3.1.5 กำหนดสมมติฐาน

สิ่งที่สำคัญสิ่งหนึ่งในการวิเคราะห์มาตรฐานความปลอดภัยคือ สมมติฐาน ข้อผิดพลาดหรือจุดอ่อนของโปรแกรมในปัจจุบันมักจะเกิดจากสมมติฐานที่ผิดของผู้ที่ออกแบบโปรแกรม เนื่องจากโดยปกติผู้ออกแบบโปรแกรมมักจะถือว่าทรัพยากรของระบบ เช่น พื้นที่เก็บข้อมูลหรือ หน่วยความจำ นั้นมีเพียงพอเสมอ ดังนั้นควรหลีกเลี่ยง การใช้ทรัพยากรจนหมด เช่น เมื่อรู้ว่าธรรมชาติของมนุษย์มักจะมีการใส่ข้อมูลลงในซอฟต์แวร์ ก็ไม่ควรที่จะให้ซอฟต์แวร์นั้นมีการทำงานจนพื้นที่เก็บข้อมูลหรือหน่วยความจำเต็ม เป็นต้น

3.1.6 ออกแบบความปลอดภัยตั้งแต่ต้น

ผู้พัฒนาโปรแกรมส่วนมากนั้นมักจะไม่เห็นความสำคัญของการออกแบบความปลอดภัยในขั้นต้น แต่มักจะเพิ่มความปลอดภัยลงไปภายหลัง ซึ่งเป็นสิ่งที่ไม่ถูกต้อง การออกแบบ

โปรแกรมให้มีความปลอดภัยนั้นควรออกแบบการเข้ารหัส และมีการใช้งานทดสอบไปด้วย เพื่อป้องกันการเกิดช่องโหว่ความปลอดภัยในภายหลัง

3.1.7 ออกแบบด้วยความคิดที่ว่ามีผู้ที่ไม่ประสงค์ดี

ผู้พัฒนาโปรแกรมควรมีความกระตือรือร้นว่ามีผู้ที่ไม่ประสงค์ดีต้องการโจมตีโปรแกรม เพราะเมื่อโปรแกรมนั้นมีการใช้งานจริงนั้น เป็นไปได้ที่จะมีผู้ที่ต้องการจะโจมตีโปรแกรมซึ่งอาจจะมาจากภายนอกเครือข่าย หรือภายในเครือข่าย ดังนั้นเมื่อผู้พัฒนาตระหนักดังนี้แล้ว ก็จะช่วยให้การออกแบบโปรแกรมประยุกต์มีความปลอดภัยมากยิ่งขึ้น

3.1.8 ระมัดระวังในการใช้สิทธิพิเศษ (Privileges)

โปรแกรมควรจะทำงานด้วยระดับสิทธิพิเศษที่พอเพียงเท่านั้น เช่น ถ้าต้องการเพียงแค่จะเปิดบันทึกเพื่ออ่าน ก็ไม่ควรที่จะเปิดไฟล์สำหรับการเขียนด้วย และเมื่อต้องการจะสร้างไฟล์เพื่อแชร์ ควรจะสร้างกลุ่มที่สามารถเข้าถึงได้ และสร้างรายชื่อผู้ที่สามารถเข้าถึงข้อมูลได้ หลักการในข้อนี้บางครั้งเรียกว่า การจำกัดการใช้สิทธิต่ำสุด(Least privilege) ถ้าระบบปฏิบัติการที่ใช้สามารถจัดการได้ ต้องตรวจสอบให้มั่นใจได้ว่ามีการลดสิทธิพิเศษลงไปอยู่ในขั้นต่ำสุดเท่าที่เป็นไปได้ และจะเลื่อนระดับสิทธิพิเศษขึ้นมาก็ต่อเมื่อมีความจำเป็นเท่านั้น และควรใช้เวลาที่น้อยที่สุดเพื่อลดโอกาส และเวลาในการที่ผู้บุกรุกสามารถรันโค้ดอันตรายและยกระดับสิทธิพิเศษ

3.1.9 ตรวจสอบการกระทำที่ขัดต่อนโยบาย

เพื่อความปลอดภัยที่แน่นหนา ผู้ออกแบบต้องมั่นใจว่าเหตุการณ์หรือการกระทำต่างๆในแต่ละขั้นตอนนั้นไม่ขัดต่อนโยบาย

3.1.10 สร้างให้มีความทนทานต่อความผิดพลาดในขั้นที่เหมาะสม

โปรแกรมประยุกต์ที่ดีควรมีความทนทานสามารถรองรับความผิดพลาดได้ หากความผิดพลาดนั้นไม่ทำความเสียหายให้แก่ระบบมากมายนัก

3.1.11 การควบคุมแอดเดรสที่ผิดพลาดด้วยความเหมาะสม

ผู้ออกแบบควรคำนึงถึงการควบคุมแอดเดรสที่มีการอ้างอิง หรือการคืนค่าโดยโปรแกรมประยุกต์ และตรวจสอบแอดเดรสว่ามีความถูกต้องหรือไม่ ก่อนนำแอดเดรสไปใช้งานต่อไปในโปรแกรมประยุกต์

3.1.12 มีการจำกัดปริมาณการบริโภคทรัพยากรของโปรแกรม

กระบวนการโจมตีระบบของโปรแกรมประยุกต์ซึ่งเป็นที่นิยมมากก็คือการทำให้โปรแกรมประยุกต์นั้นมีการใช้ทรัพยากรของระบบ เช่น หน่วยความจำ หรือเวลาที่ใช้ในการประมวลผลจนหมดซึ่งจะทำให้ระบบนั้นล่ม ทำงานไม่ได้ ดังนั้นจึงควรมีการจำกัดปริมาณการบริโภคทรัพยากรของโปรแกรม เช่น การกำหนดหน่วยความจำสูงสุดที่โปรแกรมสามารถใช้งานจากระบบได้ เป็นต้น

3.1.13 กำจัดการเชื่อมต่อที่เป็นจุดอ่อน

ควรให้ความระมัดระวังในเรื่องของช่องโหว่ที่ผู้บุกรุกได้วางเอาไว้ (back door) ซึ่งมักจะเป็นจุดอ่อนที่พบได้บ่อยในโปรแกรมประยุกต์ทั่วไป

3.1.14 มีการป้องกันหลายชั้น

การป้องกันในเชิงลึกนั้นสำคัญมาก โปรแกรมที่ดีควรมีการป้องกันจากหลายจุด ไม่ใช่เพียงการใส่รหัสผ่านที่ถูกต้องแล้วสามารถเข้าถึงข้อมูลทุกอย่างได้ โปรแกรมควรจะตรวจสอบสิทธิการใช้งาน (permission) และรหัสผ่านที่ถูกต้อง ซึ่งหลักการในข้อนี้ต้องการความละเอียดในการทำงานของผู้พัฒนาอีกด้วย จัดการกับความเสี่ยงด้วยแผนการป้องกันหลายๆแบบ เผื่อว่าชั้นป้องกันหนึ่งไม่พอเพียง อีกชั้นก็จะสามารถปกป้องช่องโหว่ทางความปลอดภัยจากการถูกเอ็กซ์พลอยต์และจำกัดขั้นตอนการเอ็กซ์พลอยต์ที่สำเร็จ ตัวอย่างเช่น รวมเอาเทคนิคการเขียนโปรแกรมที่ปลอดภัย กับ สภาวะแวดล้อมขณะรันไทม์ที่ปลอดภัย ก็จะช่วยลดความเป็นไปได้ที่ช่องโหว่ที่มีในโค้ด จะถูกเอ็กซ์พลอยต์ในขณะโอเปอร์เรชันไทม์

3.1.15 ทุกสิ่งภายในโปรแกรมต้องการความปลอดภัย

หนึ่งปัจจัยที่นักพัฒนาโปรแกรมมักจะมองข้ามไปก็คือ ทุกสิ่งภายในระบบของโปรแกรมประยุกต์นั้นต้องการความปลอดภัย เช่น โปรแกรมที่จะทำงานร่วมกับโปรแกรมที่ผู้พัฒนาออกแบบ, ความสามารถในการเชื่อมต่อเครือข่าย เช่น การเปิดเว็บเพจที่สร้างมาจากไมโครซอฟต์เวิร์ด นั้นสามารถใส่มาโครลงไปได้ เมื่อมีผู้เปิดเว็บเพจนั้นก็สามารถให้มาโครทำงานซึ่งเป็นช่องโหว่ของโปรแกรมที่เกิดขึ้นได้ ดังนั้นควรระมัดระวังในทุกๆ ขั้นตอนของการทำงาน

3.1.16 นำซอร์สโค้ดที่ทราบมีความปลอดภัยมาใช้งาน

ผู้ออกแบบไม่ควรที่จะใช้เวลาไปกับการแก้ไขข้อผิดพลาดหากมีผู้พัฒนาไว้แล้วการนำซอร์สโค้ดที่ทราบว่ามีความปลอดภัยมาประยุกต์ใช้งาน จะช่วยให้การทำงานทำได้อย่างรวดเร็ว มีประสิทธิภาพสูง และลดข้อผิดพลาดลงได้ โดยทั่วไปแล้วหากมีผู้พัฒนาเอาไว้แล้วและผ่านการใช้งานโดยผู้พัฒนาบนอินเตอร์เน็ตมาเป็นระยะเวลาหนึ่งแล้ว จะทำให้เรามั่นใจได้มากขึ้นว่าข้อผิดพลาดต่างๆ ในตัวโปรแกรมหรือไลบรารีนั้นก็จะได้รับการแก้ไขไปแล้วในระดับหนึ่ง จึงมีความเชื่อถือได้สูง ก่อนนำมาใช้ ให้ตรวจสอบว่าใครคือผู้พัฒนา หากมีปัญหาจะติดต่อได้อย่างไร ให้ดาวน์โหลดจากเว็บไซต์ที่เชื่อถือได้สูง และหาวิธีการในการตรวจสอบว่าโปรแกรมหรือไลบรารีนั้นเป็นของแท้หรือไม่ เช่น ดูจากขนาด ดูจากลายมือชื่ออิเล็กทรอนิกส์ ทั้งนี้เพื่อป้องกันไม่ให้มีม้าโทรจันติดมาด้วย

ข้อควรระวังเกี่ยวกับการนำซอร์สโค้ดที่มีอยู่แล้วมาใช้งานก็คือ ผู้พัฒนาไม่ควรพิจารณาว่าซอร์สโค้ดนั้นมีความปลอดภัยมากเนื่องจากว่ามีการเปิดเผยสู่สาธารณะเป็นระยะเวลานาน เนื่องจาก หากซอร์สโค้ดนั้นมีการเปิดเผยมานาน อาจจะมีผู้ไม่ประสงค์ดีที่ทราบช่องโหว่ของซอร์สโค้ดนั้นก็เป็นได้

3.1.17 ตรวจทานสิ่งที่อาจหลงลืม

เมื่อตรวจสอบหลักการข้อต่างๆ มาแล้ว ผู้พัฒนาโปรแกรมก็ไม่ควรที่จะลืมหลักการในข้อสุดท้าย ซึ่งต้องถามตัวเองหรือผู้ร่วมงานว่าได้ลืมสิ่งใดไปหรือไม่ในส่วนต่างๆ ของงาน และตรวจสอบดูให้แน่ใจว่าทุกอย่างถูกต้องสมบูรณ์ สิ่งเหล่านี้เป็นสิ่งที่จำเป็นมากที่ต้องทำให้เคยชิน และโปรแกรมประยุกต์ที่พัฒนาจะปลอดจากช่องโหว่ความปลอดภัย

3.1.18 ทำให้ข้อมูลปลอดภัยก่อนส่งไประบบอื่น

ทำให้ข้อมูลปลอดภัยก่อนส่งไปยังซับซิสเต็ม เช่น คอมมานด์เชลล์ , รีเลชันนอล ดาต้าเบส. ผู้บุกรุกอาจจะสามารถเรียกฟังก์ชันที่ไม่สมควร โดยการใช้เอสคิวแอล , คอมมานด์ หรือการโจมตีแบบอินเจ็คชัน ซึ่งมันไม่เกี่ยวกับปัญหาการตรวจสอบอินพุตเพราะว่าซับซิสเต็มที่ซับซ้อนเหล่านี้ไม่เข้าใจข้อความที่ซึ่งตัวคอลนั้นสร้างขึ้นมา. เพราะว่าคอลลิ่งโปรเซสเข้าใจข้อความ มันเป็นความรับผิดชอบที่จะต้องทำให้ข้อมูลปลอดภัยก่อนจะส่งไปยังซับซิสเต็ม

ที่กล่าวมาข้างต้นเป็นแนวทางเบื้องต้นในการที่จะพัฒนาซอฟต์แวร์ให้มีความปลอดภัย โดยหลักการข้างต้นนี้ใช้ได้ในขั้นตอนของการออกแบบ ซึ่งถ้าผู้พัฒนาซอฟต์แวร์ดำเนินงานตามหลักนี้

จะทำให้ซอฟต์แวร์ได้พัฒนาออกมามีความปลอดภัยสูงขึ้น แต่สิ่งที่เราต้องสนใจต่อคือ การเขียนโปรแกรมให้ปลอดภัย เพราะถึงแม้ว่าซอฟต์แวร์มีการออกแบบมาดี แต่มีการเขียนโค้ดที่ไม่ดีก็อาจเกิดปัญหาได้

3.2 ตรวจสอบจากนโยบายความปลอดภัย

นโยบายทางความปลอดภัย เป็นข้อควรกระทำ ที่พึงระลึกไว้เสมอในการเริ่มออกแบบซอฟต์แวร์

3.2.1 ไม่เขียนโค้ดที่มีการใช้ชื่อไฟล์โดดๆ ควรอ้างที่อยู่ด้วย
ชื่อไฟล์ควรมีการอ้างแบบเต็มๆ กล่าวคือ ควรขึ้นต้นด้วย '/' หรือ '\' (การอ้างแบบเต็มๆ มีหลายแบบขึ้นอยู่กับว่าเป็นระบบปฏิบัติการไหน) การอ้างแค่ชื่อไฟล์อาจมีผลทำให้เกิดปัญหาได้ เช่น เราจะทำงานไฟล์ /home/passwd แต่เราเขียนแค่ชื่อไฟล์ ทำให้เราอาจไปแก้ไขไฟล์ที่ /etc/passwd ได้ เป็นต้น

3.2.2 ไม่อ้างถึงไฟล์ 2 ครั้งในโปรแกรมเดียวกันโดยใช้ชื่อของมัน ให้เปิดไฟล์โดยใช้ชื่อครั้งเดียวและใช้แฮนเดิลทำงาน

3.2.3 อย่าคิดว่าผู้ใช้งานไม่ใช่ผู้ไม่ประสงค์ดี

3.2.4 อย่าคิดว่าการกระทำทุกอย่างจะต้องสำเร็จซิสเต็มคอล เช่น การเปิดไฟล์ อ่านไฟล์ ควรตรวจสอบสภาวะการหยุด และ ควรมีมาตรการจัดการเมื่อมีการเกิดล้มเหลวขึ้น

3.2.5 ไม่พิสูจน์ตัวตนโดยใช้ข้อมูลที่เชื่อถือไม่ได้ เช่น ไอพี , เลขแมคแอดเดรส

3.2.6 ไม่เก็บข้อมูลที่ไวต่อการเปลี่ยนแปลงในฐานข้อมูลโดยไม่มีพาสเวิร์ดป้องกัน

3.2.7 ไม่พิมพ์หรือแสดงพาสเวิร์ดออกมาทางหน้าจอของผู้ใช้งานไม่ว่ากรณีใดๆ

3.2.8 ไม่เขียนยูสเซอร์เนม หรือพาสเวิร์ดลงในแอพพลิเคชัน

3.2.9 ไม่ควรส่งหรือเก็บพาสเวิร์ดหรือ ข้อมูลสำคัญแบบไม่เข้ารหัส

3.2.10 อย่าใช้การตัดสินใจการเข้าระบบโดยดูจากตัวแปรแวดล้อมหรือพารามิเตอร์จากคอมมานไลน์ ตัวอย่างเช่น แทนที่เราจะหายูสเซอร์ไอดีจาก สภาพแวดล้อมของผู้ใช้งาน เราก็หันมาใช้ getuid() จากไลบรารีแทน

บทที่ 4
ความปลอดภัยในขั้นตอนการเขียนโค้ดและตรวจสอบ

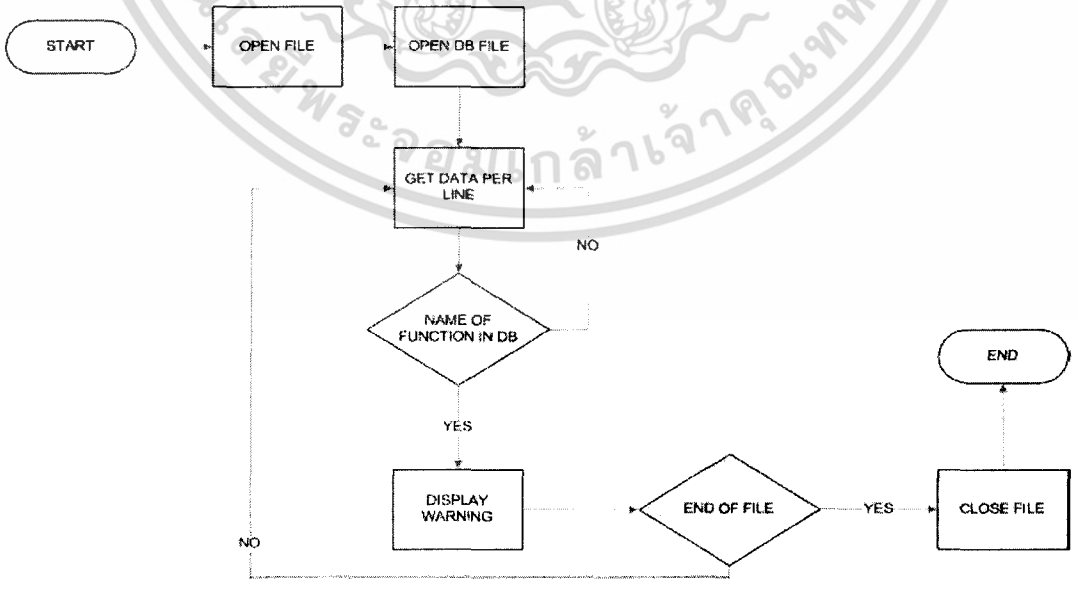
ปัญหาในขั้นตอนการเขียนโค้ดโปรแกรม(Implement) นั้นเกิดจากความผิดพลาดของโปรแกรมเมอร์ ซึ่งเขียนโปรแกรมออกมาไม่ดี ทำให้มีช่องโหว่ให้ผู้บุกรุกใช้ประโยชน์จากช่องโหว่นั้นในการบุกรุกหรือสร้างความเสียหายให้กับซอฟต์แวร์ โดยปัญหาที่เราให้ความสนใจเป็นปัญหาที่เกี่ยวข้องกับการเอ็กซ์พลอยต์เพื่อให้ได้สิทธิเหนือปกติ (gain privilege) ด้วยวิธีการบัฟเฟอร์โอเวอร์โฟลว์

สำหรับขั้นตอนการเขียนโค้ดนี้ ได้จัดทำโปรแกรมขึ้นมา 2 โปรแกรม นั่นคือโปรแกรม VulnScan ซึ่งทำงานลักษณะ Static code analysis และโปรแกรม OverflowGuard ที่ป้องกันการเกิดบัฟเฟอร์โอเวอร์ ขณะที่ซอฟต์แวร์นั้นรันการทำงานอยู่

ส่วนในขั้นตอนการตรวจสอบ(Maintenance) ซึ่งเป็นช่วงหลังที่ได้ปล่อยซอฟต์แวร์นั้นออกมาใช้งานแล้ว ก็ต้องมีการทดสอบ แก้ไข ปรับปรุง จึงได้พัฒนาโปรแกรมขึ้นมาอีก 2 โปรแกรม คือ โปรแกรมฟอลต์อินเจ็คชันที่ไว้คอยทดสอบความแข็งแกร่งต่อการบุกรุก และการตรวจสอบการปรับปรุงเวอร์ชันที่ช่วยให้ซอฟต์แวร์สามารถตรวจหาเวอร์ชันใหม่ๆ จากนักพัฒนา

4.1 การตรวจสอบการเขียนโค้ดภาษาซี

VulnScan เป็นการป้องกันการเกิดบัฟเฟอร์โอเวอร์โฟลว์ โดยจะทำการตรวจสอบโค้ดโปรแกรมว่ามีการใช้ฟังก์ชันที่เสี่ยงต่อการเกิดบัฟเฟอร์โอเวอร์โฟลว์หรือไม่ ถ้าพบจะทำการแจ้งเตือนและให้คำแนะนำ ตลอดจนจะมีการตรวจสอบการใช้ตัวแปรอาร์เรย์ที่มีการใช้งานเกินขอบเขต



รูปที่ 4.1 ผังการทำงานของโปรแกรม VulnScan

รูปโฟลว์การทำงานของโปรแกรม

ตัวอย่างฟังก์ชันที่เสี่ยงต่อการเกิดโอเวอร์โฟลว์

ฟังก์ชัน	strcpy
สาเหตุ	Does not check for buffer overflows when copying to destination
ข้อแนะนำ	Consider using strncpy or strlcpy (warning, strncpy is easily misused)
ฟังก์ชัน	strcat
สาเหตุ	Does not check for buffer overflows when concatenating to destination
ข้อแนะนำ	Consider using strncat or strlcat (warning, strncat is easily misused)
ฟังก์ชัน	lstrcat
สาเหตุ	Does not check for buffer overflows when copying to destination\n
ข้อแนะนำ	Make sure destination can always hold the source data
ฟังก์ชัน	wcscat
สาเหตุ	Does not check for buffer overflows when copying to destination
ข้อแนะนำ	Make sure destination can always hold the source data

ฟังก์ชันทั้งหมดสามารถตรวจสอบได้ที่ ภาคผนวก ข. คู่มือป้องกันการเขียนโค้ดให้ปลอดภัยด้วยภาษาซี

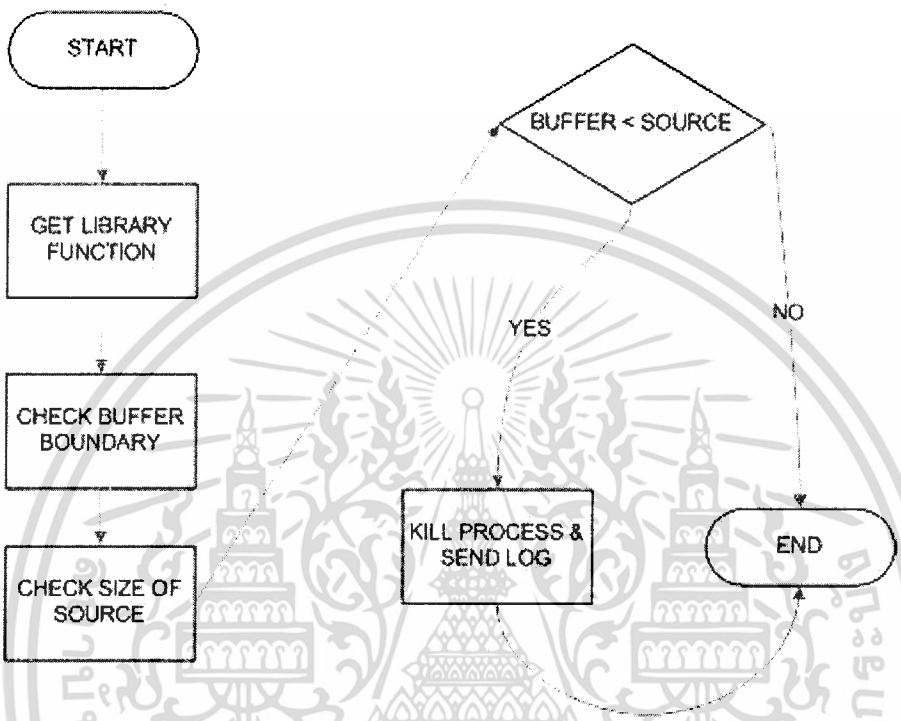
การใช้งานตัวแปรอาร์เรย์เกินขอบเขต

```
char str[5];
str[5] = Z;
```

จากตัวอย่าง พบว่าเมื่อเราทำการคอมไพล์โค้ดตัวอย่าง จะไม่พบสิ่งผิดปกติ แต่เมื่อเราทำการรันโปรแกรม จะเกิดเซ็กเมนเทชันฟอลต์ขึ้น ซึ่งจุดนี้จะเป็นช่องโหว่ซึ่งผู้ไม่ประสงค์ดีจะให้เป็นช่องทางจู่โจมได้

4.2 โปรแกรมป้องกันการเกิดบัฟเฟอร์โอเวอร์

OverflowGuard เป็นโปรแกรมตรวจสอบและป้องกันการเกิดบัฟเฟอร์โอเวอร์โฟลว์ในขณะทำการรันโปรแกรม โดยอาศัยหลักการสำคัญ คือ สแตกต้องไม่มีการเปลี่ยนแปลงค่าเมื่อฟังก์ชันคืนค่ากลับไป



รูปที่ 4.2 ผังการทำงานหลักของ Overflow Guard

OverflowGuard มีหลักการทำงานโดยจะทำการดักจับการเรียกฟังก์ชัน เช่น strcpy() และทำการตรวจสอบว่าข้อมูลที่เขียนมีขนาดมากกว่าบัฟเฟอร์ที่มีอยู่หรือไม่

ฟังก์ชันที่สามารถดักจับและตรวจสอบได้ มีดังนี้

- strcpy()
- strcat()
- sprintf()
- vsprintf()
- getwd()
- gets()
- realpath()

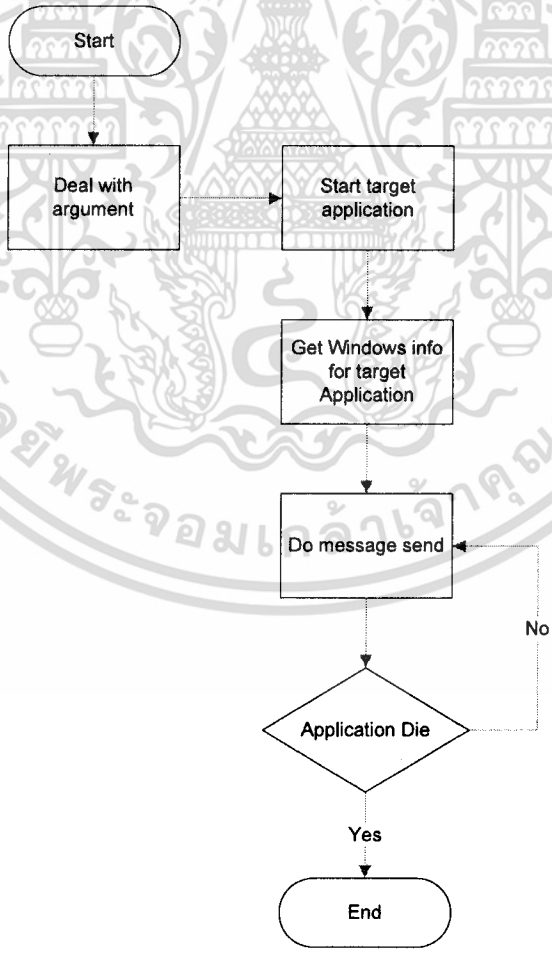
4.3 ฟอลต์อินเจ็คชัน

4.3.1 เครื่องมือทำฟอลต์อินเจ็คชัน

FIT (Fault Injection Tool) เป็นโปรแกรมที่พัฒนาขึ้นมาใช้สำหรับการอินเจ็ค ข้อมูลแบบสุ่มเพื่อทดสอบความทนทานของโปรแกรมต่อสภาวะอินพุตที่คาดเดาไม่ได้ ซึ่งเป็นลักษณะของการทดสอบแบบแบล็คบ็อค คือ เป็นวิธีที่ไม่ขึ้นโดนตรงกับโครงสร้างการเขียนโปรแกรมใดๆ โดยเฉพาะ ขณะที่การทำการทดสอบแบบไวท์บ็อค นั้นเป็นการทดสอบที่รู้โครงสร้างโค้ดของโปรแกรม ฉะนั้นจึงต้องระบุเป้าหมายที่เจาะจง ทั้งภาษา และแพลตฟอร์ม

FIT นั้นจะส่ง Random Win32 messages เข้าไปยังแอพพลิเคชัน โดยใช้ Win32 API function PostMessage และ SendMessage โดยที่ PostMessage นั้นจะส่ง Win32 message ไปกับ message queue ไปยังหน้าต่างที่เลือก และตอบกลับ ขณะที่ SendMessage จะส่ง Win32 message และรอจนกว่าเมสเสจจะประมวลผลเรียบร้อยก่อนที่จะตอบกลับมา

Win32 messages นั้นมีขนาดและฟิลด์ที่แน่นอน มี 3 ฟิลด์ คือ 1 คือ message ID , อีก 2 คือ integer parameters โดยจะสุ่ม 1 ใน 2 ฟิลด์หลังนี้



รูปที่ 4.3 ผังการทำงานหลักของ Fault Injection Tool

4.3.2 ช่องโหว่จาก DCOM RPC

DCOM RPC หรือ Distributed Component Object Model Remote Procedure Call ของบริษัทไมโครซอฟต์ ทำงานบนพอร์ตทีซีพีหมายเลข 135 เป็นโปรโตคอลที่ช่วยทำให้ซอฟต์แวร์คอมโพเนนท์สามารถติดต่อกันโดยตรงผ่านเน็ตเวิร์ก ชื่อเดิมคือ Network OLE. DCOM ออกแบบมาสำหรับใช้ข้ามระหว่างเน็ตเวิร์ก รวมถึงอินเทอร์เน็ตโปรโตคอล เช่น HTTP ด้วย

ขณะที่ผู้บุกรุกได้ค้นพบช่องโหว่ทางความปลอดภัยของ DCOM และสร้างเครื่องมือที่จะเข้าไปทางพอร์ต 135 (ทีซีพี) และจะสร้างประตูลับ (backdoor) เพื่อให้บุกรุกสามารถกลับเข้าไปใช้คอมมานด์เชลล์แบบมีสิทธิพิเศษได้ ในเครื่องที่ถูกบุกรุก ส่วนบางวิธีจะใช้พอร์ตทีซีพีหมายเลข 4444 ในการสร้างประตูลับ บางวิธีจะใช้หมายเลขพอร์ตใดๆ แล้วแต่ที่ผู้บุกรุกจะตั้งค่าในเวลาบุกรุก นอกจากนี้ CERT/CC ยังได้รับรายงานว่ามีการสแกนพอร์ตต่างๆ ที่มักถูกใช้เป็นประตูลับ เช่น 4444/TCP ในบางกรณีระบบที่ถูกบุกรุกอาจจะรีบูตจากการหยุดบริการของ RPC หลังจากที่ผู้บุกรุกได้เข้าไปใช้ประตูหลังแล้ว

ผู้โจมตีจากภายนอกอาจฉวยโอกาสในช่องโหว่เหล่านี้เพื่อใช้ในการรันโปรแกรมโดยใช้สิทธิของผู้ใช้ที่อยู่ภายในระบบ หรือเพื่อให้เกิดสถานะ การปฏิเสธการให้บริการในเครื่องที่ถูกโจมตี

4.3.3 เอ็กซ์พลอยต์ DCOM RPC

ทำการศึกษาโค้ดไฟล์ MS Windows DCOM Exploit.c ที่เป็นที่แพร่หลายตามอินเตอร์เน็ท ซึ่งสามารถทำการเอ็กซ์พลอยต์ บน Windows 2000 SP 0-4 ,Windows XP Service Pack0 ได้โดยที่มีค่า อ็อปเซ็ทไว้สำหรับทุกๆ เวอร์ชัน โดยการทำงานคือ จะส่งเชลล์โค้ดเข้าไปผ่านพอร์ต 135 เพื่อทำการโอเวอร์โฟลว์และเปิดพอร์ต 4444 ในการสร้างประตูลับ หลังจากนั้นจึงเข้าใช้คอมมานด์เชลล์บนเครื่องเหยื่อได้บนสิทธิระดับผู้ดูแลระบบได้

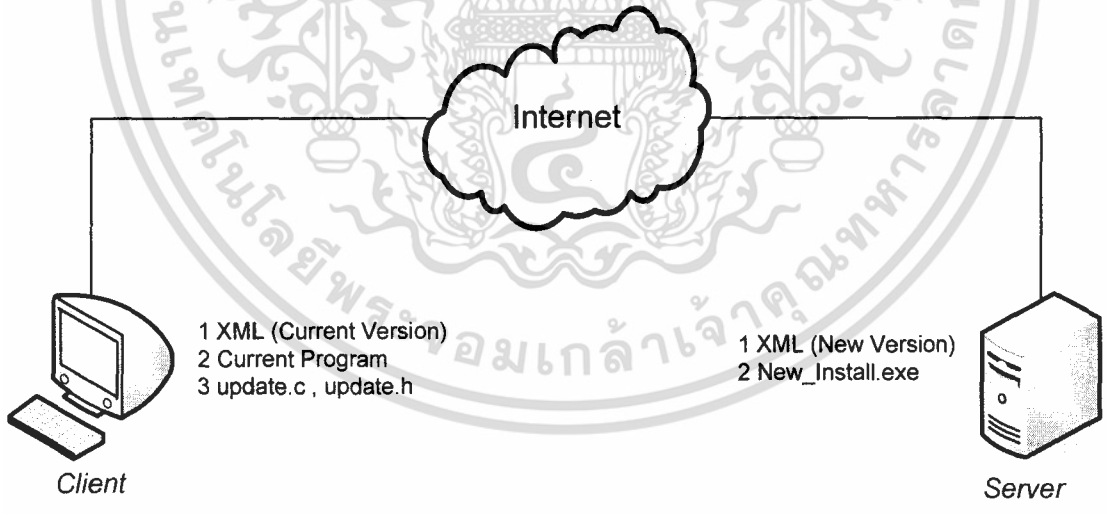
4.4 การตรวจสอบการปรับปรุงเวอร์ชัน

ส่วนหนึ่งของการทำให้ซอฟต์แวร์มีความปลอดภัยต่อการถูกเอ็กซ์พลอยต์นอกเหนือไปจาก การเขียนโค้ดที่รัดกุมแล้ว การปรับปรุงซอฟต์แวร์ก็เป็นสิ่งสำคัญที่ขาดไม่ได้ ซึ่งก็เหมือนจะอยู่ในส่วนของขั้นตอนการตรวจสอบ ดูแล ในวงจรชีวิตซอฟต์แวร์

ซึ่งแน่นอนว่าไม่มีโปรแกรมเมอร์คนไหนเขียนโปรแกรมได้อย่างถูกต้องเสมอไป และต้องมีการทดสอบหาบักของโปรแกรมเสมอๆ และเมื่อตรวจพบบัก และช่องโหว่ทางความปลอดภัยต่างๆ เหล่านั้นแล้ว ก็ปรับปรุงแก้ไขโปรแกรม และออกโปรแกรมเวอร์ชันใหม่ออกมาเรื่อยๆ และหากโปรแกรมนั้นมีฟีเจอร์ที่สามารถเช็คอัพเดตเวอร์ชันใหม่ๆ ได้ (ลักษณะเหมือนโปรแกรมแอนตี้ไวรัส ที่คอยตรวจไวรัสซิกเนเจอร์ใหม่ๆ เสมอ) ก็ยิ่งจะทำให้ซอฟต์แวร์นั้นสามารถปกปิดช่องโหว่ด้านความปลอดภัยได้รวดเร็ว และสร้างความสะดวกต่อผู้ใช้งานยิ่งขึ้น

4.4.1 แนวทางการตรวจสอบการปรับปรุงเวอร์ชัน

โครงงานได้พัฒนารูปแบบฟีเจอร์ของการเช็คเวอร์ชันสำหรับซอฟต์แวร์ทั่วไป ที่เขียนด้วยภาษาซี ซึ่งจะทำให้ง่ายต่อนักพัฒนาโปรแกรม ที่สามารถนำเอาไปประยุกต์ใช้ได้โดยง่าย อาศัยรูปแบบผ่านเอ็กซ์เอ็มแอลประกอบด้วยสิ่งสำคัญ ดังต่อไปนี้



รูปที่ 4.4 แสดงส่วนจำเป็นในการใช้แนวทางการตรวจสอบเวอร์ชัน

1. ไฟล์เอ็กซ์เอ็มแอล(update.xml)

ระบุถึงรายละเอียดของโปรแกรม ที่สำคัญ คือ GUID (globally unique identifier) , เวอร์ชัน , ลิงค์ไปยังที่อยู่ไฟล์เอ็กซ์เอ็มแอล , ลิงค์ไปยังไฟล์ติดตั้ง โดยต้องมีไฟล์เอ็กซ์เอ็มแอลนี้ 2 ไฟล์ ไว้ที่โปรแกรม(เวอร์ชันปัจจุบัน) และไว้ที่เซิร์ฟเวอร์(เวอร์ชันใหม่)

```
<?xml version="1.0" ?>
<XML_DIZ_INFO>
        <Program_Info>
                <id>{407b1420-ec9e-415e-a45e-7e2e396746f8}</id>
                <version>2.0</version>
                <xmlLink>http://www.your.com/update.xml</xmlLink>
                <fileLink>http://www.your.com/new_install.zip</fileLink>
        </Program_Info>
</XML_DIZ_INFO>
```

รูปที่ 4.5 ตัวอย่างเอ็กซ์เอ็มแอล (update.xml)

2. ไฟล์ฟังก์ชัน และเฮดเดอร์ (update.c , update.h)

ไฟล์ฟังก์ชันที่ใช้สำหรับการเช็คเวอร์ชัน รวมทั้งเฮดเดอร์ ในการนำไปอิมพลีเมนต์เข้ากับโปรแกรม โดยที่ไฟล์ update.c

3. เว็บเซิร์ฟเวอร์

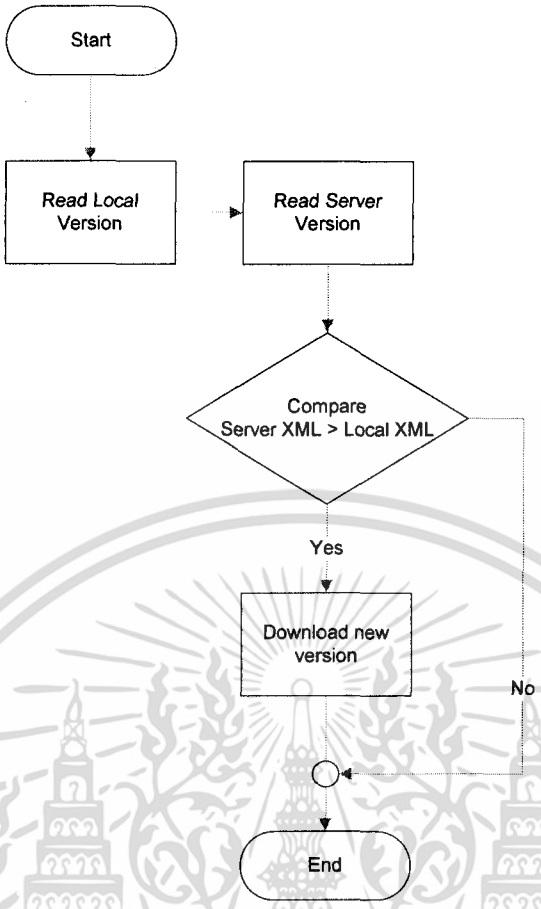
สำหรับให้บริการเมื่อโปรแกรมมีการเช็คเวอร์ชัน เก็บไฟล์เอ็กซ์เอ็มแอล และไฟล์ติดตั้งโปรแกรมตัวใหม่ หรือไฟล์ใดๆ ที่โปรแกรมต้องการสำหรับการอัพเดต

ขั้นตอนการนำฟีเจอร์เข้าไปใช้งาน

1. นำไฟล์ update.xml, update.c , update.h มาไว้ไดเรคทอรีเดียวกับซอร์สโค้ด
2. กำหนดแท็กต่างๆ ในไฟล์ update.xml

```
<id>{GUID}</id>
<version>Version_number</version>
<xmlLink>URL_to_xml</xmlLink>
<fileLink>URL_to_newfile</fileLink>
```

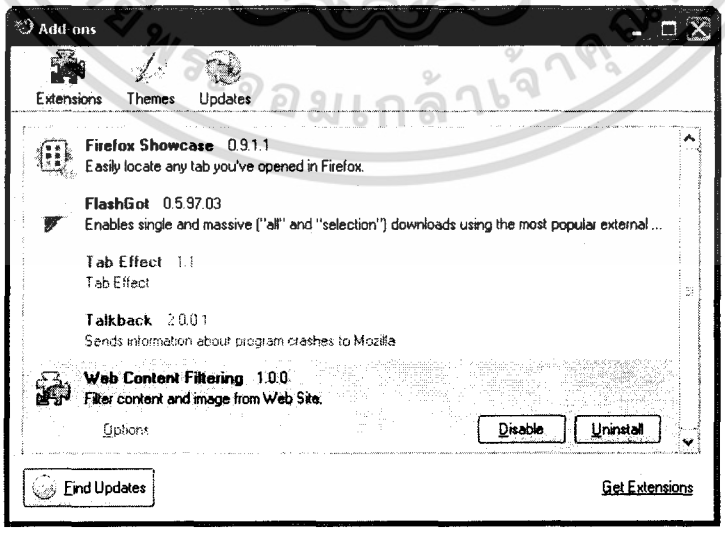
3. ให้โปรแกรมเพิ่ม #include “update.h”
4. เรียกใช้ฟังก์ชัน update() ซึ่งจะทำงานดังนี้
   4.1 เปิดไฟล์ update.xml ในเครื่อง แล้วอ่านเอาค่าตัวแปรต่างๆ ออกมาเก็บไว้
   4.2 ดาวน์โหลดไฟล์ update.xml ที่อยู่บนเซิร์ฟเวอร์ที่ระบุไว้ แล้วอ่านเก็บใส่ตัวแปร
   4.3 นำเอาค่าตัวแปรทั้งสองมาเปรียบเทียบกัน โดยเปรียบเทียบก่อนว่า GUID ตรงกัน จากนั้นก็เปรียบเทียบเลขเวอร์ชัน
   4.4 เมื่อพบว่ามีเวอร์ชันใหม่กว่า นำไฟล์เอ็กซ์เอ็มแอลใหม่มาคัดลอกทับเอ็กซ์เอ็มแอลเดิม และดาวน์โหลดไฟล์ตัวใหม่ตามที่ระบุยูอาร์แอลไว้
5. เสร็จสิ้นขั้นตอนการอัพเดต



รูปที่ 4.6 ผังการทำงานของแนวทางการตรวจสอบเวอร์ชัน

4.4.2 การปรับปรุงเวอร์ชันส่วนขยายของไฟร์ฟ็อกซ์

ทำการศึกษากลุ่มโครงงาน Web Content Filtering (ห้องวิจัย ISAG) ซึ่งเป็นลักษณะของส่วนขยายบนไฟร์ฟ็อกซ์ที่คอยทำหน้าที่กรองเนื้อหาเว็บเพจที่ส่อไปในทางลามก อนาจาร



รูปที่ 4.7 Web Content Filtering Extension Version 1.0.0

4.4.2.1 การสร้างไฟล์แพ็คเกจ สำหรับติดตั้งส่วนขยาย

1. wcf.xpi เป็นไฟล์ แพ็คเกจที่ใช้ในการ install extension ลงบนไฟร์ฟ็อกซ์
2. โดยที่ wcf.xpi มาจากการ zip ไฟล์ install.rdf กับ chrome (folder) เข้าด้วยกัน แล้วเปลี่ยนนามสกุล .zip เป็น .xpi
   - chrome (folder) ประกอบไปด้วยไฟล์ WebContentFiltering.jar
3. WebContenFiltering.jar มาจากการ zip content (folder) แล้วเปลี่ยนนามสกุลเป็น .jar
   - content (folder) ประกอบไปด้วยไฟล์ contents.rdf และ WebContentFiltering-Overlay.xul

ในที่นี่จะไปกล่าวถึงรายละเอียดอื่นๆ ไฟล์ที่เราสนใจในการทำการตรวจสอบการปรับปรุงเวอร์ชันนั้นคือ ไฟล์ install.rdf

4.4.2.2 ขั้นตอนในการทำการตรวจสอบการปรับปรุงเวอร์ชัน

1. ที่ไฟล์ install.rdf มีแท็กที่สำคัญสำหรับการทำตรวจสอบการปรับปรุงเวอร์ชันคือ
   - 1.1 <em:version> ระบุเวอร์ชันของส่วนขยายตัวปัจจุบัน
   - 1.2 <em:id> ระบุ GUID ของส่วนขยายซึ่งต้องเป็นค่าเฉพาะเจาะจง
   - 1.3 <em:updateURL> ระบุถึง URL ของไฟล์ที่ให้บริการการอัพเดต
   - 1.4 <em:minVersion> ระบุถึงเวอร์ชันไฟร์ฟ็อกซ์ต่ำสุดที่ส่วนขยายรองรับ
   - 1.5 <em:maxVersion> ระบุถึงเวอร์ชันไฟร์ฟ็อกซ์สูงสุดที่ส่วนขยายรองรับ

```
<?xml version="1.0"?>

<RDF xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:em="http://www.mozilla.org/2004/em-rdf#">

 <Description about="urn:mozilla:install-manifest">
  <em:id>{12a1584b-2123-473d-8752-e82e74e3cb1b}</em:id>
  <em:version>1.0.0</em:version>
  <em:targetApplication>
   <Description>
    <em:id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</em:id>
    <em:minVersion>1.0</em:minVersion>
    <em:maxVersion>2.0.0.1</em:maxVersion>
   </Description>
  </em:targetApplication>
  <em:name>Web Content Filtering</em:name>
  <em:description>Filter content and image from Web Site.</em:description>
  <em:creator>Prueak and Worawit CE@KMITL</em:creator>
  <em:homepageURL>http://www.ce.kmitl.ac.th/</em:homepageURL>
  <em:updateURL>http://161.246.5.10/project/update.rdf</em:updateURL>
```

```
    <em:contributor>NECTEC and SIPA</em:contributor>
        <em:file>
          <Description about="urn:mozilla:extension:file:WebContentFiltering.jar">
                <em:package>content/</em:package>
          </Description>
        </em:file>
  </Description>
</RDF>
```

จะสังเกตเห็นได้ว่า มี <em:id> 2 อันคือ

{12a1584b-2123-473d-8752-e82e74e3cb1b} เป็น GUID สำหรับส่วนขยายของเรา ซึ่งต้องเป็นค่าไม่ซ้ำกับใคร และ {ec8030f7-c20a-464f-9b0e-13a3a9e97384} เป็น GUID สำหรับไฟร์ฟ็อกซ์

2. บนเซิร์ฟเวอร์ที่ให้บริการการอัพเดต นั้นมีไฟล์ update.rdf อยู่

2.1 <version>	ระบุเวอร์ชันของส่วนขยายใหม่
2.2 Urn:mozilla:extension:	ระบุ GUID ของส่วนขยายที่ต้องการ
2.3 <updateLink>	ระบุไฟล์แพ็คเกจสำหรับติดตั้งเวอร์ชันใหม่

```
<?xml version="1.0" ?>
<r:RDF xmlns:r="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns="http://www.mozilla.org/2004/em-rdf#">
<r:Description about="urn:mozilla:extension:{12a1584b-2123-473d-8752-e82e74e3cb1b}">
<updates>
    <r:Seq>
        <r:li>
        <r:Description>
        <version>2.0.0</version>
        <targetApplication>
            <r:Description>
                <id>{ec8030f7-c20a-464f-9b0e-13a3a9e97384}</id>
                <minVersion>1.0</minVersion>
                <maxVersion>2.0.0.1</maxVersion>
                <updateLink>http://161.246.5.10/project/install.xpi</updateLink>
            </r:Description>
        </targetApplication>
        </r:Description>
        </r:li>
    </r:Seq>
</updates>
<version>2.0.0</version>
<updateLink>http://161.246.5.10/project/install.xpi</updateLink>
</r:Description>
</r:RDF>
```

ซึ่งไม่เพียงแค่รองรับการอัพเดตส่วนขยายเพียงไฟล์เดียว แต่ยังสามารถรองรับการอัพเดตหลายๆ ส่วนขยายได้ (ด้วยการระบุไอดี)

3. วางไฟล์ install.xpi ซึ่งเป็นไฟล์ install package version ใหม่ไว้บนเซิร์ฟเวอร์ตามที่อยู่ที่ถูกต้อง
4. เมื่อมีการ กด Find Update จากไฟร์ฟ็อกซ์ของเครื่องไคลเอนต์ก็จะมีการเข้ามาเรียกดูไฟล์เอ็กซ์เอ็มแอลบนเซิร์ฟเวอร์ (update.rdf) เพื่อนำค่าพารามิเตอร์ต่างๆ เช่น GUID , เวอร์ชัน ไปเปรียบเทียบกับอันเดิม หากมีเวอร์ชันที่ใหม่กว่าไฟร์ฟ็อกซ์ก็จะแสดงให้ไคลเอนต์ ทราบ

บทที่ 5

การทดลองและผลการทดลอง

5.1 การทดสอบโปรแกรม VulnScan

โปรแกรม VulnScan เป็นโปรแกรมที่มีจุดประสงค์เพื่อที่จะสแกนซอร์สโค้ดที่เขียนด้วยภาษาซี เพื่อที่จะหาช่องโหว่ทางความปลอดภัย เช่น การใช้ฟังก์ชันที่อันตรายต่อบัฟเฟอร์โอเวอร์โฟลว์ หรือการใช้งานตัวแปรอาร์เรย์เกินขอบเขต

ระบบที่ทำการทดสอบ

Intel Pentium 4 2.0 GHz

Ram 512 MB

ระบบปฏิบัติการณ์ลินุกซ์ Fedora Core 1

Option ต่างๆ

-f เพื่อตรวจจับการใช้ฟังก์ชันที่เสี่ยงต่อการถูกโจมตีและความเสี่ยงอื่นๆ

-r เพื่อตรวจหาค่าเกินขอบเขตของตัวแปร

-v เพื่อเช็คค่าเวอร์ชั่นของโปรแกรม

-u เพื่ออัพเดตโปรแกรม

ขั้นตอนการทดลอง

1. ทำการทดสอบโปรแกรมกับโค้ด Flaw.c โดยใช้ คำสั่ง

   python VulnScan.py –f Flaw.c

   Option -f เพื่อตรวจจับการใช้ฟังก์ชันที่เสี่ยงต่อการถูกโจมตีและความเสี่ยงอื่นๆ

รายละเอียดโค้ด *Flaw.c*

```
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <fcntl.h>
#include <sys/stat.h>
#include <string.h>
#include <ldap.h>

char *ldap_host = "192.168.1.100";                    // Firestation host name or IP
address
```

```
char *root_dn = "cn=Administrator, cn=Users, dc=hephaestus, dc=com"; // Root dn of
Active directory
char *root_pw = "123456";                                    // Root dn's password
char *dn = NULL;

// Edit By waan
char arr1[10];
char arr2[20];
// ----------

int logging(LDAP *ld, unsigned char *log)
{
        char* dn;
        LDAPMod *list_of_attrs[2];
        LDAPMod attribute;
        LDAPControl **srvrctrls, **clntctrls;

        // Distinguished name of the entry that you want to modify
        dn = "cn=firescreen, cn=computers, dc=hephaestus, dc=com";

        // Values to add or change
        char *logEntry[] = { log, NULL };

        // Specify each change in separate LDAPMod structures
        attribute.mod_type = "firewallLog";
        attribute.mod_op = LDAP_MOD_ADD;
        attribute.mod_values = logEntry;

        // Add the pointers to these LDAPMod structures to an array
        list_of_attrs[0] = &attribute;
        list_of_attrs[1] = NULL;

        // Change the entry
        if ( ldap_modify_s( ld, dn, list_of_attrs ) != LDAP_SUCCESS ) {
                ldap_perror( ld, "ldap_modify_s" );
                return( 1 );
        }

        //Edit by waan
        arr1[11] = "555";

        return 0;
}

int main( int argc, char *argv[] )
{
        printf("\n[ PEBKAC ]\n\n");

        LDAP *ld;

        FILE *fp;
        unsigned char log[999], temp[999];
        unsigned char *token;

        int chk=0;

        // Init LDAP connection
```

```
        if ((ld = ldap_init(ldap_host, LDAP_PORT)) == NULL ) {
                perror( "ldap_init failed!!!" );
                exit( EXIT_FAILURE );
        }

        // LDAP binding
        if (ldap_bind_s(ld, root_dn, root_pw, LDAP_AUTH_SIMPLE) != LDAP_SUCCESS ) {
                ldap_perror( ld, "ldap_bind failed!!!" );
                exit( EXIT_FAILURE );
        }

        fp=fopen("/var/log/firescreen.log", "r");
        int i=0, counter=0;
        while (fgets(log, 999, fp) != NULL) {

                //printf("\nlog na ja: %s\n", log);
                strcpy(temp, log);

                // tokens seperated with " "
                token = strtok(temp, " ");

                while (token != NULL) {
                        if (!strcmp(token, "FIRESCREEN")) {
                                logging(ld, log);
                                printf("%s\n", log);
                        }
                        //printf("%s\n", token);
                        token = strtok(NULL, " ");
                }
        }
        counter++;

        //Edit by waan
        arr2[20] = "555";

        fclose(fp);
        fp=fopen("/var/log/firescreen.log", "w+");              // clear log file
        fclose(fp);
        printf("counter: %d\n", counter);
        chk = counter;

        exit(0);
}
```

2. ผลลัพธ์ที่ได้

```
Entries in C database: 69

Line: 15 Fixed sized buffer
Line: 16 Fixed sized buffer
Line: 22 Fixed sized buffer
Line: 38 Fixed sized buffer
Line: 39 Fixed sized buffer
Line: 48 Fixed sized buffer
Line: 60 Fixed sized buffer
Line: 99 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on
the stack are used safely. They are prime targets for buffer overflow attacks.

Line: 82 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)

Line: 79 : gets
gets is unsafe!! No bounds checking is performed, buffer is easily overflowable
by user. Use fgets(buf, size, stdin) instead.

Line: 79 : fgets
Double check that your buffer is as big as you specify.When using functions
that accept a number n of bytes to copy, such as strncpy, be aware that if the
dest buffer size = n it may not NULL-terminate the string.

Line: 77 : fopen
Line: 102 : fopen
A potential race condition vulnerability exists here.  Normally a call to this
function is vulnerable only when a match check precedes it.  No check was
detected, however one could still exist that could not be detected.

Line: 77 : open
Line: 102 : open
A potential race condition vulnerability exists here.  Normally a call to this
function is vulnerable only when a match check precedes it.  No check was
detected, however one could still exist that could not be detected.
```

จากผลการทดลองจะพบว่ามีการพบการใช้ฟังก์ชันที่เสี่ยงต่อการถูกโจมตีหลายอย่าง เช่น strcpy() ซึ่งโปรแกรมตรวจพบที่ บรรทัดที่ 82 และให้คำแนะนำว่า ควรใช้ strncpy() หรือ strlcpy() แทนจะดีกว่า เป็นต้น

3. ทดสอบอีกครั้งโดยใช้คำสั่ง

python VulnScan.py –r Flaw.c

Option -r เพื่อตรวจหาค่าเกินขอบเขตของตัวแปร

4. ผลลัพธ์ที่ได้

```
Find array out of bound

Out of Bound variable ' arr1 ' in line : 48
Out of Bound variable ' arr2 ' in line : 99
```

จากผลการทดลอง จะพบว่ามีการใช้งานตัวแปรเกินขอบเขตที่บรรทัด 48 และ 99 ซึ่งต้องทำการแก้ไข

5.2 การทดสอบโปรแกรม OverflowGuard

OverflowGuard เป็นโปรแกรมที่ตรวจจับและป้องกันบัฟเฟอร์โอเวอร์โฟลว์ แบบ Stack-smashing protection ซึ่งพัฒนาด้วยภาษาซี

ระบบที่ทำการทดสอบ

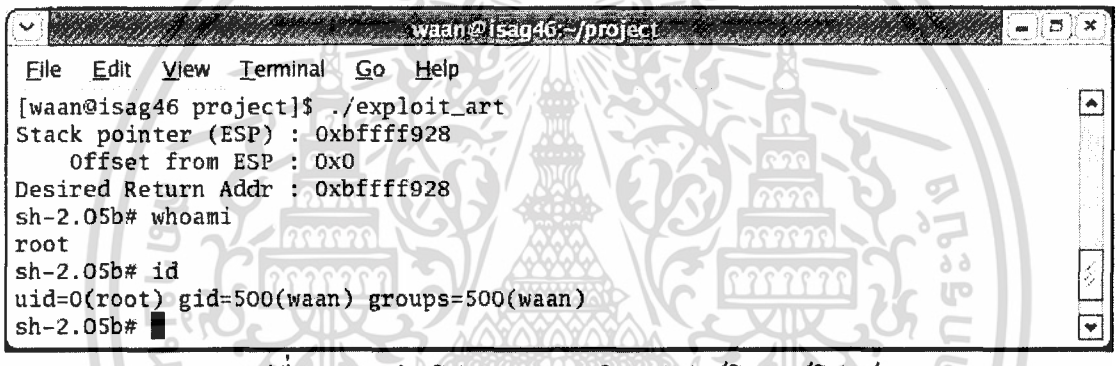
Intel Pentium 4 2.0 GHz

Ram 512 MB

ระบบปฏิบัติการณ์ลินุกซ์ Fedora Core 1

ขั้นตอนการทดลอง

ทำการทดสอบโปรแกรมที่เกิดโอเวอร์โฟลว์



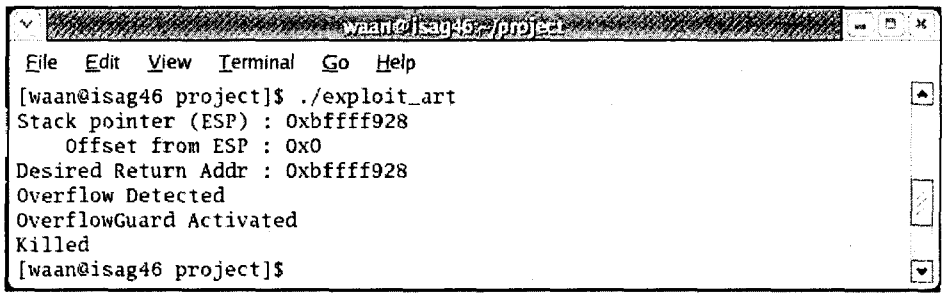
รูปที่ 5.1 ตัวอย่างโปรแกรมการเกิดบัฟเฟอร์โอเวอร์โฟลว์

จะเห็นว่า เครื่องเราจะถูกโจมตีได้อย่างอิสระ จะเห็นว่าค่าของ uid จะเปลี่ยนไปเป็นรูต แสดงว่าการทำสแตกโอเวอร์โฟลว์ประสบผลสำเร็จ

1. ทำการคอมไพล์ซอร์สโค้ดโปรแกรม ด้วยคำสั่ง

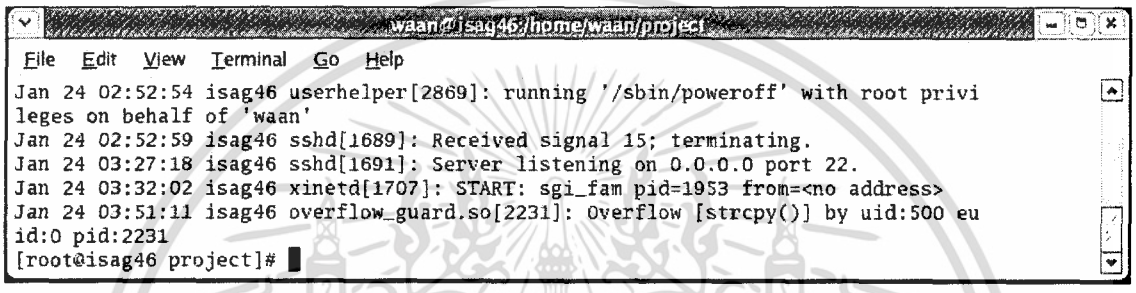
   gcc –fPIC –c –o Overflow_Guard.o Overflow_Guard.c

   gcc –shared –o Overflow_Guard.so Overflow_Guard.o –ldl
2. ย้าย Overflow_Guard.so ไปยัง /lib/Overflow_Guard.so
3. แก้ไขไฟล์ /etc/ld.so.preload โดยใส่ค่า /lib/Overflow_Guard.so เข้าไป
4. ทดสอบด้วยโปรแกรมเดิมอีกครั้ง
5. ผลลัพธ์ที่ได้จะเห็นว่าโปรแกรมจะตรวจสอบ แล้วทำการฆ่าโพรเซสทิ้งไป



รูปที่ 5.2 แสดงให้เห็นว่าป้องกันการเกิดบัฟเฟอร์โอเวอร์โฟลว์ได้

เราสามารถตรวจสอบล็อกได้จาก /var/log/secure ซึ่งจากการทดลองได้ผลลัพธ์ ดังนี้



รูปที่ 5.3 ล็อกไฟล์

5.3 การทดสอบฟอลต์อินเจ็คชัน

ทำการทดสอบโปรแกรม Fault injection tool กับโครงงาน Web Content Filtering ซึ่งเป็นแอปพลิเคชั่นบน Windows

ทดสอบการ Exploit DCOM RPC บน Windows ด้วยไฟล์โค้ด MS Windows DCOM Exploit.c โดยทดลองกับ Windows SP1 ที่ยังไม่มีการแพตช์ใดๆ เพิ่มเติมจาก Microsoft และไม่มีการ Disable DCOM

Fault Injection Tool

ระบบที่ทำการทดสอบ

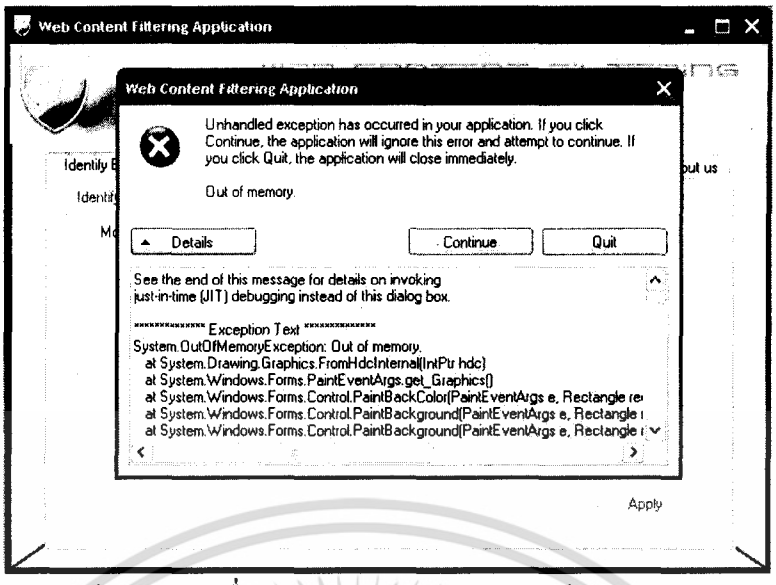
Intel Pentium M 1.7 GHz

Ram 1024 MB

ระบบปฏิบัติการณ์ Windows XP SP2

ขั้นตอนการทดลอง

1. โปรแกรม Fault Injection Tool มีการใช้งาน Option ต่างๆ ต่อไปนี้
   Process ID : ใส่หมายเลขโปรเซสไอดีของโปรแกรมที่ต้องการทดสอบ
   Number of message : จำนวนเมสเสจ
   Please Choose type message : เลือกประเภทของเมจเสจ
   Random Win32 message test โดย SendMessage API function
   Random Win32 message test โดย PostMessage API function
2. ทดสอบกับโปรแกรม Web Content Filtering โดยทำการส่งเมจเสจทั้งแบบ PostMessage และ SendMessage เข้าไปจำนวนอย่างละ 1000 เมจเสจ
3. ผลลัพธ์การทดสอบด้วย 1000 message แบบ SendMessage ,PostMessage ปรากฏว่าโปรแกรมยังทำงานปกติ ไม่เกิด error แต่อย่างใด
4. ทดสอบกับโปรแกรม Web Content Filtering โดยทำการส่งเมจเสจแบบ SendMessage เข้าไปจำนวน 5000 เมจเสจ
5. ไม่เกิด Error
6. ทดสอบกับโปรแกรม Web Content Filtering โดยทำการส่งเมจเสจแบบ PostMessage เข้าไปจำนวน 5000 เมจเสจ
7. ผลลัพธ์ คือเกิด Error ขึ้น Out of memory



รูปที่ 5.4 แสดงข้อผิดพลาดเมื่อถูกใส่ค่าแมสเสจสุ่ม 5000 ข้อความแบบ PostMessage

Exploit DCOM RPC

ระบบที่ทำการทดสอบ

Intel Pentium M 1.7 GHz

Ram 1024 MB

ระบบปฏิบัติการณ์ Windows XP SP2

VMWare1 Debian version 2.4

Ram 64 MB

IP 172.16.0.2 /16

VMWare2 Windows XP SP1

Ram 256 MB

IP 172.16.0.1 /16

ขั้นตอนการทดลอง

1. บน Debian ทำการคอมไพล์ไฟล์ MS Windows DCOM Exploit.c โดยใช้คำสั่ง

gcc MS\ Windows\ DCOM\ Exploit.c –o wins.o

```
TUMZA:/etc/download/exploit# ./wins.o
------------------------------------------------------------
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjurry
- Rewritten by HDM <hdm [at] metasploit.com>
- Usage: ./wins.o <Target ID> <Target IP>
- Targets:
-          0    Windows 2000 SP0 (english)
-          1    Windows 2000 SP1 (english)
-          2    Windows 2000 SP2 (english)
-          3    Windows 2000 SP3 (english)
-          4    Windows 2000 SP4 (english)
-          5    Windows XP SP0 (english)
-          6    Windows XP SP1 (english)
```

รูปที่ 5.5 แสดงแพลตฟอร์มวินโดวส์ ที่สามารถเอ็กซ์พลอยต์ได้

2. ทำการเอ็กซิคิวโปรแกรมเอ็กซ์พลอยต์โดยใช้คำสั่งดังนี้

   ./wins.o 6 172.16.0.1

3. ได้ผลลัพธ์คือได้เชลล์คอมมานด์ไลน์ของ Windows SP1 เรียบร้อย

```
TUMZA:/etc/download/exploit# ./wins.o 6 172.16.0.1
------------------------------------------------------------
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjurry
- Rewritten by HDM <hdm [at] metasploit.com>
- Using return address of 0x77e626ba
- Dropping to System Shell...

Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.

C:\WINDOWS\system32>_
```

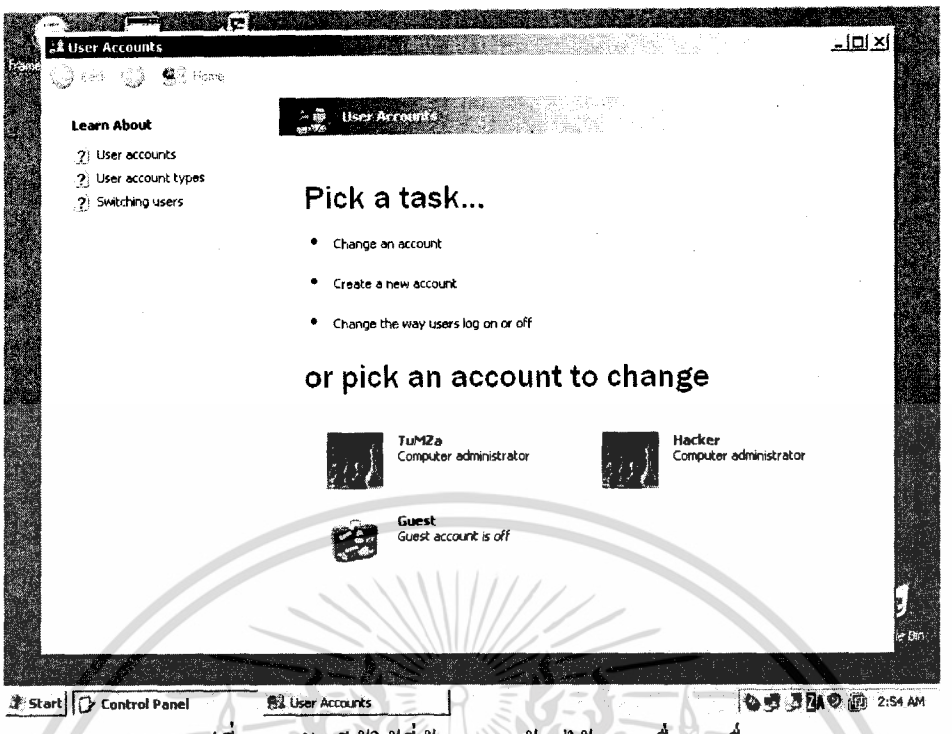
รูปที่ 5.6 แสดงการเอ็กซ์พลอยต์ได้สำเร็จ

4. ทดลองสร้างรายชื่อผู้ใช้ที่มีสิทธิเป็นผู้ดูแลระบบโดยใช้คำสั่ง

   net user Hacker /add

   net localgroup Administrators Hacker /add

5. ฝั่ง Windows SP1 ที่เมื่อเปิดดูบัญชีผู้ใช้จะพบว่า ถูกสร้างบัญชีไว้เรียบร้อยแล้ว



รูปที่ 5.7 บัญชีผู้ใช้ที่ผู้บุกรุกสร้างไว้บนเครื่องเหยื่อ

5.4 การทดสอบการตรวจสอบการปรับปรุงเวอร์ชัน

โครงงานได้นำการตรวจสอบการปรับปรุงเวอร์ชัน มาอิมพลีเมนต์เข้ากับโปรแกรม VulnScan (เพื่อไว้อัพเดต Signature ฟังก์ชันที่ไม่ปลอดภัยใหม่ๆ) และ OverflowGuard และทดสอบการตรวจสอบการอัพเดตของส่วนขยายของไฟร์ฟ็อกซ์จากกลุ่มโครงงาน Web Content Filtering

5.4.1 โปรแกรม VulnScan และโปรแกรม OverflowGuard

ระบบที่ทำการทดสอบ

Intel Pentium 4 2.0 GHz

Ram 512 MB

ระบบปฏิบัติการณ์ลินุกซ์ Fedora Core 1

โปรแกรม VulnScan

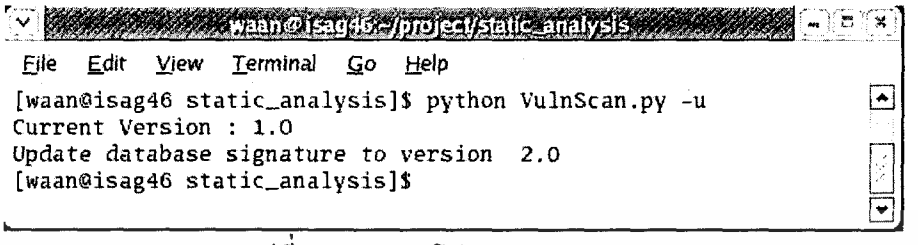
ขั้นตอนการทดลอง

1. ทำการทดสอบโปรแกรมโดยใช้คำสั่ง

python VulnScan.py –u

Option –u เพื่อให้โปรแกรมอัพเดต

2. พบว่ามี signature ใหม่จึงได้ทำการอัพเดตกลายเป็นเวอร์ชั่น 2.0



รูปที่ 5.8 อัพเดตโปรแกรม VulnScan

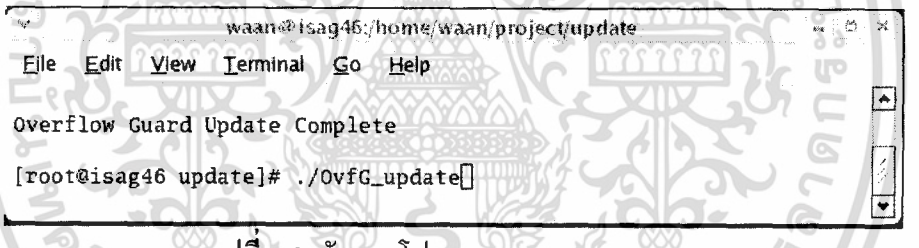
โปรแกรม OverflowGuard

ขั้นตอนการทดลอง

1. ทำการทดสอบโปรแกรมโดยใช้คำสั่ง

./OvfG_update

2. พบว่ามีไฟล์ใหม่อัพเดต จึงทำการอัพเดต



รูปที่ 5.9 อัพเดตโปรแกรม OverflowGuard

5.4.2 กลุ่มโครงงาน Web Content Filtering

ระบบที่ทำการทดสอบ

Intel Pentium M 1.7 GHz

Ram 1024 MB

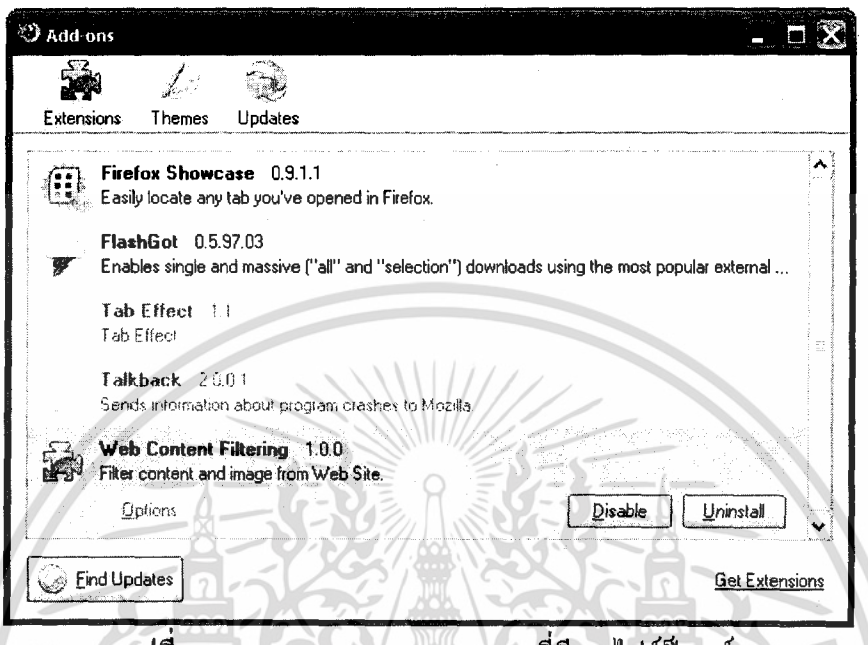
ระบบปฏิบัติการณ์ Windows XP SP2

Mozilla Firefox 2.0.0.1

ขั้นตอนการทดลอง

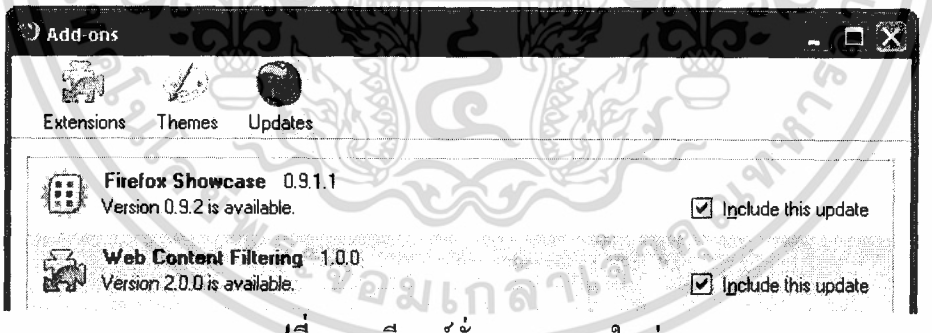
3. สร้างไฟล์ update.rdf ไว้ที่ Version update checking Server ให้เป็น Web Content Filtering เวอร์ชั่น 2.0.0

4. ทำการ install Web Content Filtering 1.0.0 ลงบน Firefox โดยการกด File -> Open File แล้ว browse เลือกไฟล์ wcf.xpi
5. Restart Firefox แล้วเลือกที่ Tool -> Add-Ons จะปรากฏหน้าต่างแสดงรายการ Extension



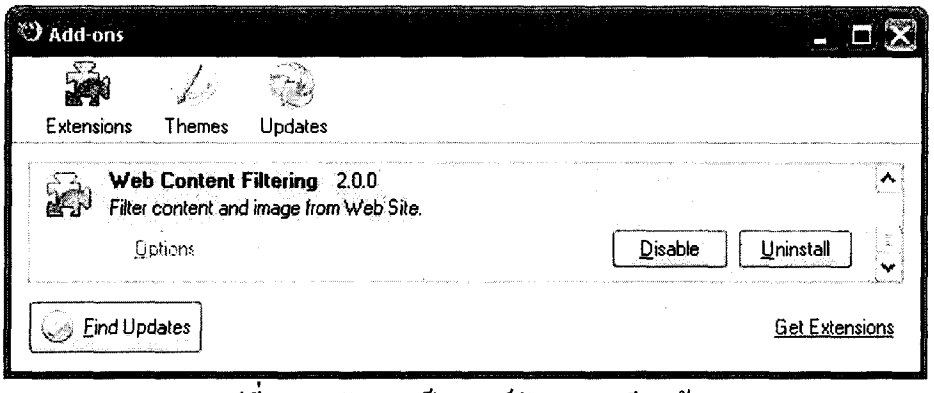
รูปที่ 5.10 แสดงรายการ Extensions ที่มีบนไฟร์ฟ็อกซ์

6. คลิกที่ Find Updates เพื่อทำการตรวจสอบการอัพเดตสำหรับทุกๆ Extensions



รูปที่ 5.11 มีเวอร์ชั่น 2.0.0 ออกใหม่

7. หากต้องการ อัพเดตเป็นเวอร์ชันใหม่นี้ เลือกกด Install Updates และรอจนกระทั่ง อัพเดตเรียบร้อย แล้วจึงรีสตาร์ทไฟร์ฟ็อกซ์
8. ผลลัพธ์ที่ได้ คืออัพเดตเวอร์ชันใหม่ได้เรียบร้อย



รูปที่ 5.12 อัพเดตเป็นเวอร์ชัน 2.0.0 เรียบร้อย

บทที่ 6
วิจารณ์และสรุป

6.1 บทสรุป

การพัฒนาโปรแกรมที่ดีนั้น จะต้องควบคู่ไปกับการพัฒนาโปรแกรมให้มีความปลอดภัย จากช่องโหว่ความปลอดภัย ซึ่งเป็นสิ่งที่มักจะไม่แพร่หลายมากนัก แต่เมื่อคำนึงถึงการใช้งานจริงๆ แล้วการพัฒนาโปรแกรมให้มีความปลอดภัยตั้งแต่เริ่มต้น จะสามารถแก้ไขปัญหา อีกทั้งเสียค่าใช้จ่ายในการแก้ไขปัญหาได้ดีกว่าการแก้ปัญหาเมื่อพัฒนาโปรแกรมเสร็จแล้ว

จากโครงงานที่ได้นี้จัดทำในครั้งนี้ ทำให้ผู้จัดทำได้ความรู้ ความเข้าใจในการตระหนักถึงความปลอดภัยของการพัฒนาโปรแกรม รวมถึงได้ศึกษาการสร้างความปลอดภัยในแต่ละขั้นตอนต่างๆ ของวงจรชีวิตซอฟต์แวร์ (SDLC) การเขียนโปรแกรมด้วยภาษาซีอย่างไรให้ปลอดภัยต่อการถูกเอ็กซ์พลอยต์ และสามารถสร้างเครื่องมือต่างๆ ที่ช่วยทำให้โปรแกรมปลอดภัยต่อการถูกบัฟเฟอร์โอเวอร์โฟลว์ได้

6.2 วิจารณ์สิ่งที่ได้จากโครงงาน

โปรแกรมเมอร์ที่เขียนโปรแกรมโดยไม่ระมัดระวังในการใช้ฟังก์ชัน การประกาศตัวแปร เป็นต้น เป็นเหตุให้ถูกโจมตีโดยอาศัยช่องโหว่ทางซอฟต์แวร์ได้ ซึ่งอาจจะนำพาไปสู่การบุกรุกของผู้โจมตี โดยผู้โจมตีสามารถที่จะได้สิทธิรูต รวมไปถึงการโจมตีที่ทำให้ระบบปิดบริการ ซึ่งสามารถก่อให้เกิดความเสียหายได้เป็นอันมาก โดยเฉพาะอย่างยิ่งในภาษาซี ซึ่งไม่มีการจัดการหน่วยความจำไม่ดีพอ ซึ่งอาจเป็นช่องโหว่ให้ถูกบุกรุกได้

ฉะนั้นโปรแกรมเมอร์จึงควรศึกษาถึงช่องโหว่เบื้องต้นในการเขียนโปรแกรม ข้อควรระมัดระวัง เพื่อให้ซอฟต์แวร์นั้นมีความแข็งแกร่งยิ่งขึ้น

6.3 ปัญหาอุปสรรคและแนวทางในการแก้ไข

1. ในการเขียนโค้ด สามารถเขียนได้หลายแบบ ทำให้การตรวจสอบโดย โปรแกรม VulnScan อาจเกิดข้อผิดพลาดได้ ให้ผลลัพธ์ไม่ถูกต้องทั้งหมดได้
2. โปรแกรม VulnScan นั้นสามารถตรวจสอบโค้ดได้เพียงภาษาซี
3. โปรแกรม OverflowGuard สามารถป้องกันได้แค่ส่วนของสแตก และตรวจสอบได้เพียงไม่กี่ฟังก์ชัน

4. ในการทำการตรวจสอบการปรับปรุงเวอร์ชันสำหรับแต่ละโปรแกรมนั้น แตกต่างทั้งภาษา , ลักษณะการทำงานของโปรแกรม จึงยากที่จะสร้างโมดูลกลางที่สามารถนำไปใช้กับทุกๆ โปรแกรมได้
5. เครื่องมือฟอลต์อินเจ็คชันยังจำกัดเพียงแค่บนระบบปฏิบัติการวินโดวส์เท่านั้น

6.4 แนวทางพัฒนาต่อ

1. พัฒนาโปรแกรม VulnScan ให้สามารถรองรับการตรวจสอบการเขียนโปรแกรมได้หลายภาษา เช่น ซีพลัสพลัส , จาวา , ไพธอน เป็นต้น
2. พัฒนาโปรแกรม OverflowGuard ให้ป้องกันการฟังก์ชันให้ได้หลากหลายมากขึ้น
3. พัฒนาโปรแกรมป้องกันการเกิดโอเวอร์โฟลว์อื่นๆ เช่น ฮีปโอเวอร์โฟลว์
4. พัฒนาโปรแกรม Fault injection tool ให้สามารถทำงานได้บนยูนิกซ์ , ลินุกซ์ และมีออปชันอื่นๆ เพิ่มเติม

บรรณานุกรม

เอกสารอ้างอิงที่เป็นหนังสือ

[1] McGraw, Gary. **Software Security: Building Security In**, Addison Wesley Professional

[2] Howard, Michael, 1965 and LeBlanc, David, 1960. **Writing Secure Code**, Microsoft Press

[3] Graff, Mark G. and Wyk, Kenneth R. van. **Secure Coding: Principles & Practices,** O'Reilly & Associates, Inc.,

[4] Messier, Matt. And Viega, John. **Secure Programming Cookbook for C and C++,** O'Reilly & Associates, Inc.

เอกสารอ้างอิงที่เป็นปริญญานิพนธ์

[5] กิจชัย รังสิมันต์ไพบูลย์ และ ดลวัต กัณฑ์สุข "ระบบต่อต้านการบุกรุกช่องโหว่ทางซอฟต์แวร์" ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2548

[6] พรรณสิริ งามมณีวัฒน์ และ วรรณา พิสิฐภิญโญ "โปรแกรมตรวจสอบช่องโหว่ความปลอดภัยระบบ" ปริญญานิพนธ์วิศวกรรมศาสตร์บัณฑิต สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง 2545

เอกสารอ้างอิงที่เป็น Web-Site

[7] SecuiTeam , "Windows RPC DoS Exploit Code (from SPIKE to C)"
URL : http://www.securiteam.com/exploits/6V00P0K5SE.html

[8] The Code Project , "Effecive online update checking mechanism using PAD/XML files"
URL : http://www.codeproject.com/cpp/padversioncheck.asp

[9] Roachfiend , "Enabling Extension Updates"
URL : http://roachfiend.com/archives/2005/03/09/enabling-extension-updates/

[10] Mozilla Develop , "Extension Versioning, Update and Compatibility"
URL : http://developer.mozilla.org/en/docs/Extension_Versioning%2C_Update_and_Compatibility#Custom_Update_RDF_Format

[11] Milw0rm , "exploits : vulnerabilities : video : papaers : shellcode"
URL : http://www.milw0rm.com

[12] The Metasploit Project ,

URL : http://www.metasploit.com

[13] Duffblog , "Writing an extension for Firefox" ,

URL : http://www.orablogs.com/duffblog/archives/000536.html

[14] CERT Secure Coding Standards

URL : https://www.securecoding.cert.org/

[15] AusCERT, "Secure Unix Programming Checklist"

URL : http://www.auscert.org.au/render.html?it=1975&cid=1920

ภาคผนวก

ภาคผนวก ก.

ผลจากการตรวจซอร์สโค้ดภาษาซีของกลุ่มโครงงานต่างๆ ในห้อง ISAG

ภาคผนวก ก.

ผลจากการตรวจโค้ดภาษาซีของกลุ่มโครงงานต่างๆ ในห้อง ISAG

1. กลุ่มโครงงาน Network Security Suite

3 ไฟล์ คือ 1 Firelog.c 2 Firescreen.c 3 Firescreend.c

1 Firelog.c

Entries in C database: 69

Line: 17 Fixed sized buffer
Line: 33 Fixed sized buffer
Line: 34 Fixed sized buffer
Line: 52 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

Line: 74 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)

Line: 71 : fgets
Double check that your buffer is as big as you specify.When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

Line: 69 : fopen
Line: 91 : fopen
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

จากผลการตรวจสอบพบจุดที่อาจก่อให้เกิดปัญหา รวมทั้งสิ้น 8 จุด จากโค้ดทั้งหมด 97 บรรทัด โดยเป็นจุดที่ต้องให้ความระวังระวังสุง อยู่ 2 จุด ได้แก่

Line: 52 : fixed size local buffer

จากการตรวจสอบการใช้งานของตัวแปร พบว่าไม่มีปัญหาที่อาจเกิดขึ้นจากจุดนี้

Line: 74 : strcpy

จากการตรวจสอบพบว่าเป็นการคัดลอกโดยข้อมูลมีขนาดเท่ากันเสมอ จึงไม่เป็นที่น่ากังวล

2 Firescreen.c

Entries in C database: 69

Line: 25 Fixed sized buffer
Line: 38 Fixed sized buffer
Line: 80 Fixed sized buffer
Line: 87 Fixed sized buffer
Line: 172 Fixed sized buffer
Line: 198 Fixed sized buffer
Line: 201 Fixed sized buffer
Line: 243 Fixed sized buffer
Line: 382 Fixed sized buffer
Line: 411 Fixed sized buffer
Line: 412 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

Line: 195 : strcpy
Line: 315 : strcpy
Line: 317 : strcpy
Line: 320 : strcpy
Line: 322 : strcpy
Line: 327 : strcpy
Line: 329 : strcpy
Line: 334 : strcpy
Line: 336 : strcpy
Line: 341 : strcpy
Line: 343 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)

Line: 255 : strcat
Line: 256 : strcat
Line: 263 : strcat
Line: 264 : strcat
Line: 271 : strcat
Line: 272 : strcat
Line: 279 : strcat
Line: 280 : strcat
Line: 287 : strcat
Line: 288 : strcat
Line: 294 : strcat
Line: 295 : strcat
Line: 301 : strcat
Line: 302 : strcat
Line: 308 : strcat
Line: 309 : strcat
Line: 316 : strcat
Line: 321 : strcat
Line: 323 : strcat
Line: 324 : strcat
Line: 328 : strcat
Line: 330 : strcat
Line: 331 : strcat
Line: 335 : strcat
Line: 337 : strcat

Line: 338 : strcat
Line: 342 : strcat
Line: 344 : strcat
Line: 345 : strcat
Line: 350 : strcat
Line: 352 : strcat
Line: 354 : strcat
Does not check for buffer overflows when concatenating to destination
Consider using strncat or strlcat (warning, strncat is easily misused)

Line: 370 : system
Line: 371 : system
Argument 1 to this function call should be checked to ensure that it does not come from an untrusted source without first verifying that it contains nothing dangerous.

Line: 182 : fopen
Line: 187 : fopen
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

Line: 178 : open
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

จากผลการตรวจสอบโค้ดด้วยโปรแกรม พบว่าพบจุดที่อาจก่อให้เกิดปัญหาทั้งหมด 59 จุด จากโค้ด 470 บรรทัด โดยจากการตรวจสอบพบว่าเป็นจุดที่ต้องใช้ความระมัดระวังสูง ดังนี้

Line:38: fixed size local buffer

จากการตรวจสอบพบว่าเป็นตัวแปรที่ไม่มีการใช้งาน

Line:243: fixed size local buffer

จากการตรวจสอบพบว่าเป็นตัวแปรที่มีการใช้งานมาก เสี่ยงต่อการเกิดบัฟเฟอร์โอเวอร์โฟลว์สูง แนะนำให้ใช้ ฟังก์ชันที่ปลอดภัยในการทำงาน กับตัวแปรนี้

Line:256: strcat	Line:309: strcat	Line:338: strcat
Line:264: strcat	Line:316: strcat	Line:342: strcat
Line:272: strcat	Line:321: strcat	Line:345: strcat
Line:280: strcat	Line:324: strcat	Line:354: strcat
Line:288: strcat	Line:328: strcat	
Line:295: strcat	Line:331: strcat	
Line:302: strcat	Line:335: strcat	

จากการตรวจสอบพบว่าอาจเกิดปัญหาได้ ถ้าข้อมูลมากเกินกว่าที่ตัวแปรจะรองรับได้

Line:317: strcpy

Line:322: strcpy

Line:329: strcpy

Line:336: strcpy

Line:343: strcpy

พบว่าเป็นจุดที่เสี่ยงต่อการเกิดปัญหามาก ควรเปลี่ยนมาใช้ strncpy() แทนเพื่อป้องกันปัญหาที่อาจเกิดขึ้นได้

3 Firescreend.c

```
Entries in C database: 69

Line: 45 : system
Line: 48 : system
Argument 1 to this function call should be checked to ensure that it does not come from an
untrusted source without first verifying that it contains nothing dangerous.

Line: 28 : umask
umask() can easily be used to create files with unsafe priviledges. It should be set to
restrictive values.
```

จากการตรวจสอบพบว่ามีจุดที่อาจก่อให้เกิดปัญหา 1 จุด จากโค้ดทั้งหมด 52 บรรทัด

Line :28: umask

ไม่พบปัญหาใดๆ

2. กลุ่มโครงงาน Incident Respond System

1 ไฟล์ คือ 1 Watsonkung.c

1 Watsonkung.c

```
Entries in C database: 69

Line: 88 Fixed sized buffer
Line: 92 Fixed sized buffer
Line: 94 Fixed sized buffer
Line: 97 Fixed sized buffer
Line: 98 Fixed sized buffer
Line: 99 Fixed sized buffer
Line: 100 Fixed sized buffer
Line: 101 Fixed sized buffer
Line: 102 Fixed sized buffer
Line: 103 Fixed sized buffer
Line: 104 Fixed sized buffer
Line: 160 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are
used safely. They are prime targets for buffer overflow attacks.

Line: 33 : wcscpy
Line: 279 : wcscpy
Line: 281 : wcscpy
Line: 285 : wcscpy
Line: 295 : wcscpy
Line: 296 : wcscpy
Does not check for buffer overflows when copying to destination
Consider using a function version that stops copying at the end of the buffer

Line: 31 : memcpy
Line: 43 : memcpy
Line: 238 : memcpy
Line: 261 : memcpy
Line: 326 : memcpy
Line: 331 : memcpy
Line: 339 : memcpy
Line: 340 : memcpy
Line: 352 : memcpy
Line: 407 : memcpy
Line: 463 : memcpy
Does not check for buffer overflows when copying to destination
Make sure destination can always hold the source data

Line: 32 : strcpy
Line: 232 : strcpy
Line: 234 : strcpy
Line: 240 : strcpy
Line: 248 : strcpy
Line: 249 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)
```

Line: 36 : strcat
Line: 345 : strcat
Line: 347 : strcat
Line: 354 : strcat
Line: 367 : strcat
Line: 368 : strcat
Does not check for buffer overflows when concatenating to destination
Consider using strncat or strlcat (warning, strncat is easily misused)

Line: 38 : wcscat
Line: 400 : wcscat
Line: 402 : wcscat
Line: 409 : wcscat
Line: 422 : wcscat
Line: 423 : wcscat
Does not check for buffer overflows when copying to destination
Make sure destination can always hold the source data

Line: 30 : strncpy
Line: 209 : strncpy
Line: 211 : strncpy
Line: 215 : strncpy
Line: 223 : strncpy
Line: 224 : strncpy
Easily used incorrectly; doesn't always \\0-terminate or
check for invalid pointers

Line: 37 : strncat
Line: 375 : strncat
Line: 377 : strncat
Line: 382 : strncat
Line: 393 : strncat
Line: 394 : strncat
Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add)
Consider strlcat or automatically resizing strings

Line: 40 : gets
Line: 481 : gets
Line: 483 : gets
Line: 485 : gets
Line: 486 : gets
gets is unsafe!! No bounds checking is performed, buffer is easily overflowable by user. Use fgets(buf, size, stdin) instead.

Line: 90 : sprintf
Line: 161 : sprintf
Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow.
Use snprintf or vsnprintf

Line: 448 : strlen
This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

Line: 41 : realpath

Line: 455 : realpath
Line: 457 : realpath
Line: 465 : realpath
Line: 474 : realpath
Line: 475 : realpath
This function does not protect against buffer overflows and some implementations can overflow internally
Ensure that the destination buffer is at least of size MAXPATHLEN, and to protect against implementation problems, the input argument should also be checked to ensure it is no larger than MAXPATHLEN

Line: 39 : getwd
Line: 431 : getwd
Line: 433 : getwd
Line: 438 : getwd
Line: 449 : getwd
Line: 450 : getwd
This does not protect against buffer overflows by itself, so use with caution
Use getcwd instead

Line: 485 : fgets
Line: 486 : fgets
Double check that your buffer is as big as you specify.When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

Line: 447 : getc
Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

จากการตรวจสอบพบจุดที่อาจเกิดปัญหารุนแรง 17 จุด จากโค้ด 590 บรรทัด ดังนี้

Line :53: syslog m

จากการตรวจสอบไม่น่าจะเกิดปัญหาจากจุดนี้ เพราะ ความยาวสตริงมีขนาดไม่ยาวเกินไป

Line :93: fixed size local buffer

Line :103: fixed size local buffer

Line :108: fixed size local buffer

Line :109: fixed size local buffer

Line :165: fixed size local buffer

Line :558: fixed size local buffer

จากการตรวจสอบพบว่า การใช้งานเป็นแบบการใช้งานง่ายๆ ไม่มีการเสี่ยงต่อการเกิดปัญหา

Line :95: sprintf

Line :166: sprintf

จากการตรวจสอบพบว่าปัญหาที่อาจเกิดขึ้นเนื่องมาจาก argument ตัวที่ 2 นั้นไม่สามารถเกิดขึ้นได้เพราะข้อมูลมาจากแหล่งที่เชื่อถือได้

Line :260: strcpy

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :328: wcscpy

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :430: strcat

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :465: strncat

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :496: wcscat

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :528: getwd

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :553: realpath

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

Line :583: gets

ไม่เกิดปัญหาจากจุดนี้ เพราะเป็นการ overwrite library function

3. กลุ่มโครงงาน Public Key Infrastructure and Biometric Application

3 ไฟล์ คือ 1 Ssl_connect.c 2 Certutil.c 3 Isagq.c

1 Ssl_connect.c

Entries in C database: 69

Line: 12 Fixed sized buffer
Line: 13 Fixed sized buffer
Line: 17 Fixed sized buffer
Line: 18 Fixed sized buffer
Line: 100 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

Line: 32 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)

Line: 27 : sprintf
Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow.
Use snprintf or vsnprintf

Line: 31 : fgets
Double check that your buffer is as big as you specify.When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

Line: 66 : read
Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

Line: 24 : fopen
Line: 48 : fopen
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

Line: 25 : open
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

จากการตรวจสอบพบจุดที่ต้องระวัง 6 จุด จากโค้ด 148 บรรทัด

Line : 12 : fixed size local buffer

Line : 13 : fixed size local buffer

Line : 17 : fixed size local buffer

Line : 18 : fixed size local buffer

Line : 100 : fixed size local buffer

ใช้ความระมัดระวังในการใช้งานให้มาก หรือ อาจเพิ่มขนาดเพื่อลดความเสี่ยง

Line : 27 : sprintf

ระวังการใช้งานอาจเกิดปัญหาได้ ถ้าขนาดของตัวรองรับไม่พอ

2 Certutil.c

```
Entries in C database: 69

Line: 19 Fixed sized buffer
Line: 27 Fixed sized buffer
Line: 48 Fixed sized buffer
Line: 49 Fixed sized buffer
Line: 53 Fixed sized buffer
Line: 108 Fixed sized buffer
Line: 110 Fixed sized buffer
Line: 114 Fixed sized buffer
Line: 174 Fixed sized buffer
Line: 175 Fixed sized buffer
Line: 179 Fixed sized buffer
Line: 208 Fixed sized buffer
Line: 230 Fixed sized buffer
Line: 231 Fixed sized buffer
Line: 232 Fixed sized buffer
Line: 233 Fixed sized buffer
Line: 290 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are
used safely. They are prime targets for buffer overflow attacks.

Line: 210 : memcpy
Does not check for buffer overflows when copying to destination
Make sure destination can always hold the source data

Line: 216 : strcpy
Line: 217 : strcpy
Line: 225 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)

Line: 164 : strncpy
Line: 223 : strncpy
Easily used incorrectly; doesn't always \\0-terminate or
check for invalid pointers

Line: 50 : sprintf
Line: 181 : sprintf
Line: 221 : sprintf
Check to be sure that the format string passed as argument 2 to this function call does not
come from an untrusted source that could have added formatting characters that the code is
not prepared to handle.  Additionally, the format string could contain `%s' without precision
that could result in a buffer overflow.
```

Use snprintf or vsnprintf

Line: 164 : strlen
Line: 223 : strlen
Line: 225 : strlen
This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

Line: 89 : read
Line: 91 : read
Line: 189 : read
Line: 297 : read
Line: 299 : read
Line: 315 : read
Line: 324 : read
Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

Line: 311 : system
Argument 1 to this function call should be checked to ensure that it does not come from an untrusted source without first verifying that it contains nothing dangerous.

Line: 66 : fopen
Line: 75 : fopen
Line: 88 : fopen
Line: 152 : fopen
Line: 296 : fopen
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

Line: 3 : open
Line: 4 : open
Line: 5 : open
Line: 6 : open
Line: 10 : open
Line: 183 : open
Line: 184 : open
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

พบจุดที่ต้องระวัง 9 จุด จากโค้ด 349 บรรทัด

Line : 48 : fixed size local buffer

Line : 49 : fixed size local buffer

Line : 174 : fixed size local buffer

Line : 175 : fixed size local buffer

Line : 208 : fixed size local buffer

Line : 290 : fixed size local buffer

ใช้ความระมัดระวังในการใช้งานให้มาก หรือ อาจเพิ่มขนาดเพื่อลดความเสี่ยง

Line : 50 : sprintf

Line : 181 : sprintf

Line : 221 : sprintf

ไม่พบปัญหาใดๆ ที่อาจจะเกิดขึ้น

3 Isagq.c

Entries in C database: 69

Line: 50 Fixed sized buffer
Line: 60 Fixed sized buffer
Line: 61 Fixed sized buffer
Line: 62 Fixed sized buffer
Line: 96 Fixed sized buffer
Line: 111 Fixed sized buffer
Line: 116 Fixed sized buffer
Line: 121 Fixed sized buffer
Line: 126 Fixed sized buffer
Line: 148 Fixed sized buffer
Line: 151 Fixed sized buffer
Line: 182 Fixed sized buffer
Line: 207 Fixed sized buffer
Line: 226 Fixed sized buffer
Line: 229 Fixed sized buffer
Line: 230 Fixed sized buffer
Line: 231 Fixed sized buffer
Line: 232 Fixed sized buffer
Line: 233 Fixed sized buffer
Line: 241 Fixed sized buffer
Line: 245 Fixed sized buffer
Line: 246 Fixed sized buffer
Line: 247 Fixed sized buffer
Line: 335 Fixed sized buffer
Line: 396 Fixed sized buffer
Line: 467 Fixed sized buffer
Line: 468 Fixed sized buffer
Line: 789 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

Line: 196 : memcpy
Line: 210 : memcpy
Does not check for buffer overflows when copying to destination
Make sure destination can always hold the source data

Line: 153 : strcat
Line: 157 : strcat
Does not check for buffer overflows when concatenating to destination
Consider using strncat or strlcat (warning, strncat is easily misused)

Line: 318 : strncpy

Easily used incorrectly; doesn't always \\0-terminate or
check for invalid pointers

Line: 66 : sprintf
Line: 100 : sprintf
Line: 114 : sprintf
Line: 119 : sprintf
Line: 124 : sprintf
Line: 129 : sprintf
Line: 152 : sprintf
Line: 156 : sprintf
Line: 166 : sprintf
Line: 253 : sprintf
Line: 257 : sprintf
Line: 336 : sprintf
Line: 420 : sprintf
Line: 472 : sprintf
Line: 500 : sprintf
Line: 693 : sprintf
Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow.
Use snprintf or vsnprintf

Line: 81 : strlen
Line: 196 : strlen
Line: 218 : strlen
Line: 255 : strlen
Line: 256 : strlen
Line: 259 : strlen
Line: 260 : strlen
This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

Line: 164 : fgets
Double check that your buffer is as big as you specify.When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

Line: 74 : read
Line: 286 : read
Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

Line: 67 : system
Argument 1 to this function call should be checked to ensure that it does not come from an untrusted source without first verifying that it contains nothing dangerous.

Line: 69 : fopen
Line: 189 : fopen
Line: 282 : fopen
Line: 473 : fopen
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

Line: 66 : open
Line: 70 : open
Line: 163 : open
Line: 190 : open
Line: 194 : open
Line: 283 : open
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

ตรวจสอบพบจุดอันตราย 31 จุด จากโค้ดทั้งหมด 1005 บรรทัด

Line : 60 : fixed size local buffer

Line : 61 : fixed size local buffer

Line : 62 : fixed size local buffer

Line : 148 : fixed size local buffer

Line : 151 : fixed size local buffer

Line : 182 : fixed size local buffer

Line : 207 : fixed size local buffer

Line : 226 : fixed size local buffer

Line : 241 : fixed size local buffer

Line : 247 : fixed size local buffer

Line : 335 : fixed size local buffer

Line : 396 : fixed size local buffer

Line : 467 : fixed size local buffer

Line : 468 : fixed size local buffer

เป็นการใช้งานโดยไม่เสี่ยงต่อการเกิดปัญหาทั้งหมด

Line : 66 : sprintf

Line : 100 : sprintf

Line : 114 : sprintf

Line : 119 : sprintf

Line : 124 : sprintf

Line : 129 : sprintf

Line : 152 : sprintf

Line : 156 : sprintf

Line : 166 : sprintf

Line : 253 : sprintf

Line : 257 : sprintf

Line : 336 : sprintf

Line : 420 : sprintf

Line : 472 : sprintf

Line : 500 : sprintf

Line : 693 : sprintf

Argument 2 มาจากแหล่งที่เชื่อถือได้ จึงไม่น่าเป็นปัญหา

Line : 67 : system

ตรวจสอบแล้วไม่พบปัญหาที่อาจเกิดขึ้นได้

4. กลุ่มโครงงาน Anti Rootkits

1 ไฟล์ คือ 1 Syscall_check_code.c

1 Syscall_check_code.c

```
Entries in C database: 69

Line: 76 Fixed sized buffer
Line: 97 Fixed sized buffer
Line: 421 Fixed sized buffer
Line: 428 Fixed sized buffer
Line: 441 Fixed sized buffer
Line: 448 Fixed sized buffer
Line: 451 Fixed sized buffer
Line: 452 Fixed sized buffer
Line: 468 Fixed sized buffer
Line: 483 Fixed sized buffer
Line: 485 Fixed sized buffer
Line: 486 Fixed sized buffer
Line: 495 Fixed sized buffer
Line: 498 Fixed sized buffer
Line: 502 Fixed sized buffer
Line: 507 Fixed sized buffer
Line: 508 Fixed sized buffer
Line: 509 Fixed sized buffer
Line: 510 Fixed sized buffer
Line: 511 Fixed sized buffer
Line: 569 Fixed sized buffer
Line: 572 Fixed sized buffer
Line: 845 Fixed sized buffer
Line: 850 Fixed sized buffer
```

Line: 859 Fixed sized buffer
Line: 861 Fixed sized buffer
Line: 863 Fixed sized buffer
Line: 864 Fixed sized buffer
Line: 866 Fixed sized buffer
Line: 873 Fixed sized buffer
Line: 919 Fixed sized buffer
Line: 920 Fixed sized buffer
Line: 922 Fixed sized buffer
Line: 923 Fixed sized buffer
Line: 952 Fixed sized buffer
Line: 954 Fixed sized buffer
Line: 956 Fixed sized buffer
Line: 960 Fixed sized buffer
Line: 961 Fixed sized buffer
Line: 962 Fixed sized buffer
Line: 972 Fixed sized buffer
Line: 974 Fixed sized buffer
Line: 981 Fixed sized buffer
Line: 984 Fixed sized buffer
Line: 985 Fixed sized buffer
Line: 1094 Fixed sized buffer
Line: 1099 Fixed sized buffer
Line: 1100 Fixed sized buffer
Line: 1101 Fixed sized buffer
Line: 1103 Fixed sized buffer
Line: 1107 Fixed sized buffer
Line: 1108 Fixed sized buffer
Line: 1110 Fixed sized buffer
Extra care should be taken to ensure that character arrays that are allocated on the stack are used safely. They are prime targets for buffer overflow attacks.

Line: 76 : memcpy
Line: 117 : memcpy
Does not check for buffer overflows when copying to destination
Make sure destination can always hold the source data

Line: 954 : strcpy
Line: 960 : strcpy
Line: 961 : strcpy
Line: 962 : strcpy
Line: 984 : strcpy
Does not check for buffer overflows when copying to destination
Consider using strncpy or strlcpy (warning, strncpy is easily misused)

Line: 452 : strncat
Line: 508 : strncat
Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add)
Consider strlcat or automatically resizing strings

Line: 439 : fgets
Line: 480 : fgets
Double check that your buffer is as big as you specify.When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

Line: 46 : read

Line: 622 : read
Line: 772 : read
Line: 778 : read
Line: 1084 : read
Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

Line: 424 : system
Line: 431 : system
Line: 435 : system
Line: 465 : system
Line: 471 : system
Line: 475 : system
Line: 681 : system
Line: 704 : system
Line: 708 : system
Line: 712 : system
Line: 716 : system
Line: 720 : system
Line: 725 : system
Line: 739 : system
Line: 740 : system
Line: 745 : system
Line: 746 : system
Line: 751 : system
Line: 752 : system
Line: 844 : system
Line: 866 : system
Line: 873 : system
Line: 967 : system
Line: 995 : system
Line: 1000 : system
Line: 1005 : system
Line: 1010 : system
Argument 1 to this function call should be checked to ensure that it does not come from an untrusted source without first verifying that it contains nothing dangerous.

Line: 431 : fopen
Line: 436 : fopen
Line: 471 : fopen
Line: 476 : fopen
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

Line: 99 : open
Line: 435 : open
Line: 475 : open
Line: 756 : open
Line: 760 : open
Line: 1074 : open
Line: 1078 : open
A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

จากการตรวจสอบพบจุดที่อาจก่อให้เกิดปัญหาใหญ่ 7 จุด จากโค้ด 1300 บรรทัด

Line : 207: fixed size local buffer

Line : 540: fixed size local buffer

Line : 547: fixed size local buffer

Line : 587: fixed size local buffer

ไม่พบจุดที่เสี่ยงต่อการเกิดปัญหา

Line : 571: strncat

Line : 627: strncat

เพื่อความปลอดภัยควรใช้ strlcat() แทน

Line : 1087: strcpy

ไม่เกิดปัญหาในจุดนี้ เพราะการคัดลอกไม่เกินขอบเขตตัวแปรที่รับได้

ภาคผนวก ข.

คู่มือป้องกัน การเขียนโค้ดให้ปลอดภัยด้วยภาษาซี

ภาคผนวก ข

คู่มือป้องกัน การเขียนโค้ดให้ปลอดภัยด้วยภาษาซี

- ออกแบบโปรแกรมอย่างระมัดระวังก่อนที่จะเริ่ม

  สร้างโครงสร้างของซอฟต์แวร์และออกแบบซอฟต์แวร์ของคุณบังคับตามนโยบายความปลอดภัย. ตัวอย่างเช่น ถ้าระบบของคุณต้องการระดับสิทธิที่แตกต่างกันในแต่ละช่วงเวลา ก็ให้แบ่งระบบเป็น ระบบย่อยหลายๆ ระบบ ซึ่งแต่ละระบบก็มีระดับสิทธิที่เหมาะสม

- ตรวจสอบค่าอินพุต

  ตรวจสอบค่าอินพุตที่มาจากแหล่งข้อมูลที่ไม่น่าเชื่อถือทั้งหมด ค่าอินพุตที่ถูกต้องเหมาะสมช่วยลดปัญหาใหญ่ของช่องโหว่ทางความปลอดภัยของซอฟต์แวร์ได้ ให้สงสัยแหล่งข้อมูลภายนอกไว้ รวมทั้ง command line arguments , network interfaces , environmental variables และ user controlled files

  - ค่าตัวแปรที่ส่งไปยังโปรแกรมผ่านทางคอมมานด์ไลน์
  - ค่าตัวแปรที่ส่งไปยัง ฟังก์ชันต่างๆ ของยูนิกซ์
  - ตรวจสอบขอบเขตของตัวแปรทุกๆตัว
  - ค่าตัวแปรที่ได้มากจากสิ่งแวดล้อม
  - ค่าตัวแปรหรืออินพุตที่ได้มากจากการอ่านไฟล์
  - ตัวแปรที่มีการ cast

- ไม่ใช้ฟังก์ชันที่ไม่มีการตรวจสอบขอบเขตของตัวแปรเมื่อมีการทำงาน

หลีกเลี่ยง	ควรใช้
gets()	fgets()
strcpy()	strncpy()
strcat()	strncat()
sprintf()	bcopy()
scanf()	bzero()
sscanf()	memcpy() , memset()
chmod()	fchmod()
chown()	fchown()

  - ทำการคำนวณขนาดของตัวแปรอย่างถูกต้อง เช่น ตัวแปรปลายทางต้องมีขนาดใหญ่เพียงพอ
  - ไม่คิดว่าอักษรลงท้ายด้วย "\0"

- อย่าลืมว่าอักษรลงท้ายด้วย "\0" ดังนั้น อักษร "abcde" มีความยาว 6 byte ไม่ใช่ 5 (แม้ว่า strlen บอกว่า 5)

- ตรวจสอบค่าส่งกลับจาก system call ทุกๆอัน
- ต้องมีการทำล็อกที่ดี ซึ่งต้องประกอบไปด้วยสิ่งต่างๆ ดังนี้
  - เวลาที่โปรแกรมทำงาน
  - UID ของโพรเซสที่ทำงาน
  - Terminal ที่ทำงาน
  - PID (Process number)
  - Command line arguments
  - Invalid argument
  - ระวังปัญหาโอเวอร์โฟลว์ของ syslog จำกัดขนาดของแมสเซจทั้งหมดที่จะส่งไปยัง syslog
- ทำให้ส่วนที่มีความเสี่ยงของโปรแกรมเล็กและง่ายที่สุดเท่าที่จะทำได้
- อ่านโค้ดของตัวเองแล้วคิดว่าจะสามารถโจมตีได้อย่างไรบ้าง
- ถ้าโปรแกรมจำเป็นต้องมีการใช้ special permission เช่น เกมส์ที่ต้องเขียนคะแนนสูงสุดไปยังไดเรกทอรี่ ต้องให้รัน SUID ไปยังแอคเคาท์ผู้ใช้งานปกติแทนที่จะเป็นรูต
- ต้องใช้ path เต็มๆ สำหรับค่าตัวแปรต่างๆ ทั้งที่เป็นคอมมานด์และไฟล์
- ทุกอย่างที่ถูกส่งมาโดย user ทั้งที่เป็นการ เขียนลงไฟล์ ใช้เป็นชื่อไฟล์ ต้องมีการตรวจสอบ shell meta character
  - อย่าแค่ตรวจสอบแค่ ../ เพราะ อาจมีการใช้วิธีการแบบ "."./
  - เป็นการดีที่ตรวจหา "valid" character และตัดส่วนที่เหลือทิ้ง
- ทดสอบโค้ดอย่างระมัดระวัง เกี่ยวกะเรื่อง operating environment เช่น
  - ถ้าเราคาดว่าโปรแกรมจะถูกรันโดยคนอื่นๆที่ไม่ใช่รูต อะไรจะเกิดขึ้นถ้ารันโปรแกรมโดยรูต
  - ถ้าคิดว่าจะรันโดยรูตจะเกิดอะไรขึ้นถ้าไม่ได้รันโดยรูต
  - โปรแกรมโดยมากถูกออกแบบให้ทำงานแบบ deamon หรือ bin สามารถก่อให้เกิดปัญหาได้เมื่อรันเป็นรูต
  - ถ้าเราคาดหมายไว้ว่าโปรแกรมจะรันใน /tmp จะเกิดอะไรขึ้นถ้ารันที่อื่นๆ
- ใช้เครื่องมือต่างๆช่วยในการตรวจสอบโค้ด เช่น VulnScan, RATS
- ระวังเรื่อง race condition เพราะสามารถนำไปสู่ deadlock หรือ การล้มเหลวการทำงานในกรณีเกิดการรันในเวลาไล่เลี่ยกัน
  - Deadlock: จำไว้ว่าสำเนาโปรแกรมมากกว่า 1 ตัวสามารถทำงานในเวลาเดียวกันได้ ควรมีการล็อกไฟล์ที่เราจะแก้ไขและต้องมีหนทางแก้ไขในกรณีที่เกิดปัญหา

- อย่าให้โปรแกรม dump core ให้ล็อกปัญหาแล้ว exit แทน
- หลีกเลี่ยง shell escapes
- ไม่ใช้ system() หรือ popen() รวมทั้ง execlp และ execvp
- ถ้าทำงานกับไฟล์ เช่น เปลี่ยน owner เปลี่ยน mode ให้เปิดไฟล์แล้วใช้ fchown(), fstat(), หรือ fchmod() ซึ่งจะช่วยป้องกันไฟล์จากการถูกทับ(race condition)
- ถ้าต้องการสร้างไฟล์ชั่วคราว ควรใช้ tmpfile()
- ระวัง /etc/utmp ไฟล์ ซึ่งสามารถเขียนได้ในบางระบบ สามารถถูก exploit แล้วทำการแก้ไขไฟล์ในระบบได้ ถ้าจะใช้งานไฟล์ utmp ต้องมีการตรวจสอบก่อน เช่น ถ้าต้องการเขียนไฟล์ไป tty ไฟล์ต้องควรจะเป็นประมาณ /dev/ttyxx
- Dynamically linked libraries , เป็นไปได้ที่จะแทนที่ของ system library ด้วยไลบรารีที่ผู้ใช้งานสร้างมา ถ้า SUID โปรแกรมเรียก non-SUID โปรแกรมในขณะที่ running privileged
- ถ้าคิดว่าไฟล์ควรเป็นไฟล์ ใช้ lstat() เพื่อให้มั่นใจว่าไม่ใช่ link
  - เพื่อจะเปิดไฟล์ต้องทำสิ่งต่อไปนี้
  - lstat() path แล้วตรวจสอบว่าได้ผลสำเร็จ
  - ตรวจสอบว่าเป็นเรื่องที่รับได้ เช่น ไม่ใช่ symlink
  - open() และตรวจว่าทำงานสำเร็จ
  - fstat() fd ถูกส่งค่ากลับโดยopen
- เอาใจใส่กับคำเตือนของคอมไพเลอร์
  - คอมไพล์โค้ดด้วยระดับคำเตือนสูงสุด เช่น gcc –Wall และแก้ไขคำเตือนเหล่านั้นด้วยการปรับปรุงตัวโค้ด
- ให้ค่าเริ่มต้นของโปรแกรมนั้นปฏิเสธการเข้าถึงก่อนเสมอ

การเขียน SUID/SGID โปรแกรม

- ไม่เขียน SUID shell scripts
- อย่าใช้ SUID เพื่อเข้าถึงไฟล์พิเศษ ให้สร้างผู้ใช้งานพิเศษ เพื่อใช้งานนี้ หรือสร้างกลุ่มพิเศษของไฟล์ และให้โปรแกรม SGID ไปยังกลุ่มนั้น
- ลบ execution environment ถ้าเป็นไปได้ และเริ่มต้นด้วยของใหม่
- หลีกเลี่ยงการใช้ system() และ popen() ถ้าต้องการสร้างโพรเซส พิเศษใช้ fork() กับexecve(2), exec(3) หรือ execl(3) และใช้ด้วยความระมัดระวัง หลีกเลี่ยงการใช้ execlp(3) และ execvp(3)
- ไม่ควรมี shell escapes ถ้าต้องมีต้องมั่นใจว่า setgid(getgid()) และ setuid(getuid()) ก่อนรันคำสั่งจากผู้ใช้งาน
- โดยปกติ ใช้ setuid() และ setgid() เท่าที่จำเป็นเท่านั้น

- ใช้ Path เต็มๆสำหรับทุกไฟล์ที่เปิด อย่าคิดเอาเองเกี่ยวกับไดเรกทอรีปัจจุบัน เราสามารถบังคับไดเรกทอรีได้โดย chdir("/tmp") เป็นขั้นแรกๆของโปรแกรม
- ถ้าต้องรันโดยเป็นรูต แต่ต้องการแค่เข้าถึงไฟล์ในไดเรกทอรีเฉพาะ ควร chdir() ก่อนแล้ว chroot()ไดเรกทอรีนั้นๆ

พาสเวิร์ด - Passwords

- อย่าแสดงพาสเวิร์ดเมื่อผู้ใช้งานพิมพ์มัน
- ถ้าเก็บพาสเวิร์ดในคอมพิวเตอร์ต้องมีการเข้ารหัส เช่น ใช้ crypt(3) ใช้ random number เพื่อเอามาทำเป็น salt
- ถ้าใช้ shell script ใช้ /usr/lib/makekey แทน crypt(3) ได้

การสร้าง Random Numbers

- แต่ละบิตของตัวเลขควรมีความน่าจะเป็นที่จะเป็น 0 หรือ 1 เท่ากัน
- Random number ไม่ควรที่จะถูกคาดเดาได้จาก เลขก่อนหน้าหรือว่าข้อมูลเกี่ยวกับการสร้าง random number
- เป็นไปไม่ได้ที่จะกำหนด internal state ของการสร้าง random number
- เป็นไปไม่ได้ที่จะใส่ค่าเริ่มต้นให้ตัวสร้าง random number

การประกาศตัวแปรและการให้ค่าเริ่มต้น – Declarations and Initialization

- ประกาศตัวแปรที่ไม่มีการเปลี่ยนแปลงค่าด้วยการใช้ const หรือ enum
- ไม่ควรใช้ชื่อตัวแปรซ้ำกันในสโคปย่อยๆ ต่างๆ
- ใช้ชื่อตัวแปรที่สื่อความหมาย
- วาง const ไว้ขวาสุดของการประกาศตัวแปร
- ไม่ควรประกาศตัวแปรมากกว่า 1 ตัวในแต่ละบรรทัด
- ไม่อ้างถึง object นอก Lifetime ของมัน
- ตัวแปรควรมีลักษณะเฉพาะ (Unique)

นิพจน์ - Expressions

- ใช้เครื่องหมาย "( )" สำหรับการทำงานที่มาก่อน
- อย่าเอาขนาดของ pointer มาตัดสินขนาดของ type
- ไม่ควร cast away ค่าตัวแปรที่เป็น const (ค่าคงที่)
- โอเปอร์แรนที่ใช้กับ sizeof ไม่ควรส่งผลกระทบต่อนิพจน์อื่น

- ไม่ควรขึ้นอยู่กับลำดับการประมวลผลระหว่างจุดที่เป็นลำดับ
- อย่าเปลี่ยนแปลงค่าคงที่
- อย่าเข้าถึง volatile object ผ่านทาง non-volatile reference
- อย่าอ้างถึงตัวแปรที่ไม่มีการตั้งค่าเริ่มต้น

ตัวเลข - Integers

- อย่าตั้งสมมติฐานเกี่ยวกับชนิดของ bit-field เมื่อใช้ในนิพจน์
- ใช้ size_t สำหรับค่าของตัวเลขทุกตัวเพื่อแสดงถึงขนาดของ object
- ต้องเข้าใจกฎการเปลี่ยนตัวเลข
- ใช้ไลบรารีตัวเลขที่ปลอดภัย
- บังคับให้ลิมิตค่าตัวเลขที่มาจากแหล่งที่ไม่น่าเชื่อถือ
- ไม่ควรใส่ค่าตัวเลขโดยใช้ scanf() หรือ ฟังก์ชัน formatted input อื่นๆ
- ใช้ strtol() เพื่อเปลี่ยนจากอักษรเป็นตัวเลข
- ระบุให้ชัดเจนว่า signed หรือ unsigned สำหรับตัวแปรชนิด character
- ตรวจสอบค่าตัวเลขทุกค่าว่าอยู่ในขอบเขต
- ต้องมั่นใจว่าค่าคงที่ enum ชี้ไปยังค่าลักษณะเฉพาะ
- กำหนดค่าคงที่ตัวเลขให้เป็นค่า enum
- ระวังเรื่องการเปลี่ยนค่า signed integer เล็กๆ ไปเป็น unsigned integer ที่ใหญ่กว่า
- ต้องมั่นใจว่าการเปลี่ยนตัวเลขต้องไม่มีการเสียหรือการแปลความหมายของข้อมูลผิดไป
- ต้องมั่นใจว่าการดำเนินการของข้อมูลไม่มีผลของโอเวอร์โฟลว์
- ต้องมั่นใจว่าการหารต้องไม่เกิดการหารด้วย 0
- ต้องมั่นใจว่าค่าตัวเลขต้องอยู่ในขอบเขตที่ใช้งานได้
- ตัวแปรที่ส่งไปยัง character handling function ต้องเป็น unsigned char

ทศนิยม - Floating Point

- ระมัดระวังเรื่องจุดทศนิยมเมื่อการคำนวณนั้นต้องการความละเอียด
- ใส่ใจในเรื่องการเรียงลำดับใหม่ของนิพจน์ของทศนิยม
- สนใจค่าละเอียดเวลาเปรียบเทียบค่าของ Floating
- ป้องกัน domain errors ใน math function

อาร์เรย์ - Arrays

- ระวังเรื่องการใช้ sizeof เพื่อหาขนาดของอาร์เรย์

- ต้องมั่นใจว่า array index อยู่ในขอบเขตที่ใช้งานได้
- ใช้ชื่ออาร์เรย์ให้ตรงกันในทุก source file
- ต้องมั่นใจว่าตัวแปรขนาดของ variable length array อยู่ในขอบเขตที่ใช้งาน

อักษร - Strings

- ตรวจสอบข้อมูลที่จะถูกส่งเข้าระบบที่ซับซ้อน
- อย่าพยายามที่จะเปลี่ยนแปลง string literals
- อย่าคัดลอกข้อมูลจาก unbounded source ไปยัง fixed-length array
- Allocated พื้นที่ให้พอเพียงเมื่อทำการคัดลอก bounded strings
- ต้องมั่นใจว่า string ทุกตัวต้องจบด้วย null-terminated
- อย่าตัด string ในขณะคัดลอก
- แบ่งขนาด wide character string อย่างถูกต้อง

ฟังก์ชั่นการจัดการหน่วยความจำ - Memory Management Functions

- Allocate และ Free หน่วยความจำในโมดูลเดียวกัน และใน level ของ abstraction
- ตั้งค่าพอยท์เตอร์ไปยัง dynamically allocated memory ให้เป็น NULL หลังจากปล่อย
- อย่า cast ค่าส่งกลับจาก malloc()
- ล้างค่าข้อมูลที่ไวต่อการเปลี่ยนแปลงที่เก็บใน dynamic memory ก่อน deallocation
- อย่า access freed memory
- Free dynamically allocated memory แค่ครั้งเดียว
- ตรวจจับและจัดการ critical memory allocation errors
- อย่าสันนิษฐาน memory allocation routines initialize memory
- Free memory เท่านั้นที่จะ allocate แบบ dyamically
- มั่นใจว่าตัวแปรขนาดของ memory allocation functions ต้องใช้งานได้
- อย่าตั้งสันนิษฐานเกี่ยวกับผลลัพธ์ของการ allocating 0 bytes
- มั่นใจว่าขนาดของตัวแปรที่ส่งไป calloc() จะไม่ก่อให้เกิดอินทีเจอร์โอเวอร์โฟลว์

อินพุท เอาท์พุท – Input Output

- ฟังก์ชันควรไม่อ้างถึงชื่อไฟล์สำหรับการระบุ
- อย่าตั้งสันนิษฐานเกี่ยวกับ fopen() และการสร้างไฟล์
- ตรวจสอบและจัดการค่าความผิดพลาดของอินพุตและเอาท์พุต
- ระบุไฟล์โดยใช้ file attribute หลายๆตัว

- สร้างไฟล์โดยใช้การจำกัดการเข้าถึงอย่างเหมาะสม
- อย่าสร้างไฟล์ใน shared directories
- ไม่สนใจ user input จาก format strings
- ชื่อตัวแปรชั่วคราวต้องมีลักษณะเฉพาะเมื่อไฟล์ถูกสร้าง
- ตรวจจับและจัดการผลลัพธ์ค่าความผิดพลาดของอินพุตและเอาท์พุตใน undefined behavior
- ใช้ int เพื่อจับค่าส่งกลับของฟังก์ชัน charater IO
- ใช้ feof() และ ferror() เพื่อดักจับ end-of-file และ file errors
- อย่าสันนิษฐานว่า newline character ถูกอ่าน
- อย่าสัญนิษฐานว่า character data ได้ถูกอ่าน
- อย่าใช้ตัวคัดลอกของ FILE object สำหรับ IO
- ตัวสร้างชื่อไฟล์ชั่วคราวต้องสร้างชื่อที่ปืนลักษณะเฉพาะ
- ไฟล์ชั่วคราวต้องถูกเปิดด้วยการเข้าถึงแบบพิเศษ
- ไฟล์ชั่วคราวต้องมีชื่อที่ไม่สามารถคาดเดาได้
- ไฟล์ชั่วคราวต้องต้องกำจัดก่อนโปรแกรมจบการทำงาน

POSIX

- ใช้ฟังก์ชั่น readlink() อย่างเหมาะสม

เบ็ดเตล็ด – Miscellaneous

- การคอมไพล์ควรตั้งค่าการแจ้งเตือนในระดับสูงๆ
- หลีกเลี่ยงความผิดพลาดที่เกิดจากการละเลย
- หลีกเลี่ยงความผิดพลาดที่เกิดจาการเพิ่ม
- ใช้คอมเมนต์ที่มีความหมายและอยู่ในรูปที่อ่านได้
- ไม่ควรใช้ฟังก์ชัน rand

ภาคผนวก ค.

ฟังก์ชันที่ไม่ปลอดภัย

ภาคผนวก ค.

ฟังก์ชันที่ไม่ปลอดภัย

lstrcpy : Does not check for buffer overflows when copying to destination Consider using a function version that stops copying at the end of the buffer

wcscpy : Does not check for buffer overflows when copying to destination Consider using a function version that stops copying at the end of the buffer

_tcscpy : Does not check for buffer overflows when copying to destination Consider using a function version that stops copying at the end of the buffer

_mbscpy : Does not check for buffer overflows when copying to destination Consider using a function version that stops copying at the end of the buffer

memcpy : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

CopyMemory : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

bcopy : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

strcpy : Does not check for buffer overflows when copying to destination Consider using strncpy or strlcpy (warning, strncpy is easily misused)

strcat : Does not check for buffer overflows when concatenating to destination Consider using strncat or strlcat (warning, strncat is easily misused)

lstrcat : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

wcscat : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

_tcscat : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

_mbscat : Does not check for buffer overflows when copying to destination Make sure destination can always hold the source data

strncpy : Easily used incorrectly; doesn't always \\0-terminate or check for invalid pointers

lstrcpyn : Easily used incorrectly; doesn't always \\0-terminate or check for invalid pointers

wcsncpy : Easily used incorrectly; doesn't always \\0-terminate or check for invalid pointers

_tcsncpy : Easily used incorrectly; doesn't always \\0-terminate or check for invalid pointers

_mbsnbcpy : Easily used incorrectly; doesn't always \\0-terminate or check for invalid pointers

strncat : Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) Consider strlcat or automatically resizing strings

lstrcatn : Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) Consider strlcat or automatically resizing strings

wcsncat : Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) Consider strlcat or automatically resizing strings

_tcsncat : Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) Consider strlcat or automatically resizing strings

_mbsnbcat : Easily used incorrectly (e.g., incorrectly computing the correct maximum size to add) Consider strlcat or automatically resizing strings

strccpy : Subject to buffer overflow if buffer is not as big as claimed CEnsure that destination buffer is sufficiently large

strcadd : Subject to buffer overflow if buffer is not as big as claimed CEnsure that destination buffer is sufficiently large

gets : gets is unsafe!! No bounds checking is performed, buffer is easily overflowable by user. Use fgets(buf, size, stdin) instead.

_getts : Does not check for buffer overflows Use fgets() instead

sprintf : Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow. Use snprintf or vsnprintf

vsprintf : Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow. Use snprintf or vsnprintf

swprintf : Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow. Use snprintf or vsnprintf

vswprintf : Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow. Use snprintf or vsnprintf

_stprintf : Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow. Use snprintf or vsnprintf

_vstprintf : Check to be sure that the format string passed as argument 2 to this function call does not come from an untrusted source that could have added formatting characters that the code is not prepared to handle. Additionally, the format string could contain `%s' without precision that could result in a buffer overflow. Use snprintf or vsnprintf

scanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

vscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

wscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

_tscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

Fscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

sscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

vsscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

vfscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

_ftscanf : The scanf() family's input operation, without a limit specification specify a limit to %s, or use a different input function

strlen : This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

wcslen : This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

_tcslen : This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

_mbslen : This function does not properly handle non-NULL terminated strings. This does not result in exploitable code, but can lead to access violations.

MultiByteToWideChar : The last argument is the number of wide chars, not the number of bytes. Getting this wrong can cause a buffer overflow since you will indicate that the buffer is twice the size it actually is. Don't forget about NULL termination.

streadd : This function does not protect against buffer overflows Ensure the destination has 4 times the size of the source, to leave room for expansion

strecpy : This function does not protect against buffer overflows Ensure the destination has 4 times the size of the source, to leave room for expansion

strtrns : This function does not protect against buffer overflows Ensure that destination is at least as long as the source

realpath : This function does not protect against buffer overflows and some implementations can overflow internally Ensure that the destination buffer is at least of size MAXPATHLEN, and to protect against implementation problems, the input argument should also be checked to ensure it is no larger than MAXPATHLEN

getopt : Some older implementations do not protect against internal buffer overflows Check implementation on installation, or limit the size of all string inputs

getopt_long : Some older implementations do not protect against internal buffer overflows Check implementation on installation, or limit the size of all string inputs

getpass : Some implementations may overflow buffers

getwd : This does not protect against buffer overflows by itself, so use with caution Use getcwd instead

getchar : Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

fgets : Double check that your buffer is as big as you specify.When using functions that accept a number n of bytes to copy, such as strncpy, be aware that if the dest buffer size = n it may not NULL-terminate the string.

getc : Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

read : Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

_gettc : Check buffer boundaries if calling this function in a loop and make sure you are not in danger of writing past the allocated space.

getenv : Environment variables are untrustable input if they can be it returns untrustable input if the environment can be set by an attacker. It can have
any content and length, and the same variable can be set more than once Check environment variables carefully before using them

curl_getenv : Environment variables are untrustable input if they can be it returns untrustable input if the environment can be set by an attacker. It can have any content and length, and the same variable can be set more than once Check environment variables carefully before using them

g_get_home_dir : This function is synonymous with 'getenv(\"HOME\")'; it returns untrustable input if the environment can be set by an attacker. It can have any content and length, and the same variable can be set more than once Check environment variables carefully before using them

g_get_tmp_dir : This function is synonymous with 'getenv(\"TMP\")'; it returns untrustable input if the environment can be set by an attacker. It can have any content and length, and the same variable can be set more than once Check environment variables carefully before using them

system : Argument 1 to this function call should be checked to ensure that it does not come from an untrusted source without first verifying that it contains nothing dangerous.

fopen : A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

open : A potential race condition vulnerability exists here. Normally a call to this function is vulnerable only when a match check precedes it. No check was detected, however one could still exist that could not be detected.

umask : umask() can easily be used to create files with unsafe privileges. It should be set to restrictive values.

fork : Remember that sensitive data get copied on fork. For example, a random number generator's internal state will get duplicated, and the child may start outputting identical number streams.